République Algérienne Démocratique et Populaire.

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.
Université A. Mira de Béjaia.

Faculté des Sciences exactes
Département Informatique

Mémoire de fin de cycle

En vue de l'obtention du diplôme de Master Recherche en Informatique
Option : Réseaux et Systèmes distribués

Thème

Validation d'un modèle de composition de services web à base d'automates hiérarchiques avec la méthode B-Événementiel

Présenté par :

Me^{lle}. **DJOUHRI Nawal.**
M^{lle} **HAMECHE Sara.**

Soutenu le 29 Juin 2013

Devant le jury composé de :

Président	*M*. KHANOUCHE Md Essaid.
Examinateur	*M*. ABBACHE Bournane.
Examinateur	*Dr*. OMAR Mawloud.
Encadreur	*M*. FARAH Zobeyr.

Promotion "2012-2013"

Résumé

Les technologies utilisant les services web sont en développement constant. Les chercheurs tentent en vain de faire en sorte que les échanges d'informations soient de machine-à-machine sans l'intervention de l'être humain. La composition des services web est un pas très important dans cette direction. L'automatisation d'un tel processus étant un problème parmi les plus compliqués, une grande variété d'approches ont été proposées afin d'y trouver une solution partielles, adaptées à des cas bien spécifiques. Nous nous sommes intéressées dans ce travail aux approches formelles. Nous proposons une solution de composition de services web, qui consiste en une représentation d'une Communication d'Automates à Etats Finis Complexes (CAEFC). Le modèle mis en œuvre est vérifié et validé étape par étape en utilisant la méthode B-événementiel.

Mots clés **:** services web, composition, B-événementiel.

Abstract

Technologies using web services are in constant development. Searchers try desperately to make possible machine-to-machine information interchange without any human intervention. Web service composition is an important step in this direction. The automatization of such a process being one of the most complicated problems, a wide variety of approaches have been proposed in order to find solutions, adapted to specific cases.

We focused in this work on formal approaches. We propose a solution for Web service composition, consisting of a Communication of Complex Finite-State Machines (CCFSM). The model built up is verified and validated step by step using the EventB method.

Keywords **:** Web services, Composition, EventB.

Remerciements

Nous tenons à remercier tout d'abord le Bon Dieu, qui nous a donné la force, et la volonté de réaliser ce modeste travail. Nos remerciements les plus chaleureux vont vers notre Encadreur, Monsieur Farah Zobeyr, qui nous a guidé, soutenu, et beaucoup aidé. Sans oublier Monsieur Salhi, Docteur Akrour Nawel, Mademoiselle Rebouh, Monsieur Khanouche, notre chef de département, Docteur OMAR Mawloud pour ces encouragements. Nos amis : Abderraouf, Kamel, Yacine et sans oublier Soussou Nous remercions également toute personne ayant contribuée de loin ou de prêt.

Dédicaces

À mes chers parents,

Mes chers frères Zak, Abdou, Adel

Mes chères soeurs Amel,Djidja,Asma

Mon cher Neveux Rayan

Mes tantes et oncles

Mes cousins et cousines

Mes amis(es)

For You...

Nawal

Dédicaces

Ce modeste travail est dédié à ma famille, ainsi qu'aux personnes qui sont chères à mon cœur.

Sara

Table des matières

Table des figures

Liste des tableaux

Glossaire

AOS : Architecture Oriented Service BPEL : Business Process Execution Language

BPEL4WS : Business Process Execution Langage for Web Services

CORBA : Common Object Request Broker Architecture

CCFSM : Complex Communication Finite State Machine

HTTP : HyperText Transfer Protocol

URI : Uniform Resource Identifier

SOA : Service Oriented Architecture

SOAP : Simple Object Access Protocole

SW : Service Web OWLS : Ontology Web Langage Semantic Services

RODIN : Rigorous Open Development Environment for Complex Systems

RMI : Remote Method Invocation

UDDI : Universal Description Discovery and Integration

WS-CDL :Web Services Choregraphy Description Language

WSDL : Web Services Description Language

WSFL :Web Service Flow Language

W3C :World Wide Web Consortium

WSCI :Web Services Choregraphy Interface

XML :Extensible Markup Language

Introduction générale

L'évolution des technologies Web et leur utilisation par les entreprises à des fins commerciales, donne naissance aux Services Web, dans le but d'élargir, et propager leurs produits à l'extérieur de l'entreprise. Aujourd'hui, le domaine des services web s'étend de plus en plus, et devient alors un sujet d'actualité, assez intéressant et en développement constant[27]. Etant donné qu'un service web soit une entité logicielle qui permet d'exposer une ou plusieurs fonctionnalités définie par une interface. Comme chaque technologie, ce domaine rencontre beaucoup de problèmes, et plusieurs approches ont été proposées pour combler les lacunes qui existent.

Notre travail se concentre sur la composition des services web, qui se résume au fait de combiner plusieurs Services Web, pour n'en former qu'un seul appelé service composite, cela lorsqu'un seul service ne peut satisfaire la requête du client, notre approche consiste à représenter un modèle de composition à base d'automates hiérarchiques, qui est exactement une Communication d'Automates à Etats Finis Complexes,où chaque service est représenté par un automate, dont chaque état peut être aussi un automate (un sevice), la construction, la vérification, ainsi que la validation se font pas-à-pas, pour qu'à la fin, le modèle résultant soit correcte, car son comportement est vérifié à chaque étape. Pour construire et valider le modèle proposé, nous avons choisi la méthode B-événementiel dite EventB. Ces méthodes formelles sont connues par leur système, et leur raisonnement assez rigoureux, intolérant aux erreurs, et ce sont des méthodes sûres[27], car la base de toute approche c'est le modèle proposé. La question qui revient, est " est ce que ce modèle est correcte ? ", ces méthodes nous permettent une connaissance préalable, et cela empêche les problèmes lors de sa mise en œuvre.

Ce mémoire comporte quatre chapitre :

- Le premier chapitre présente une vue globale sur les services web, et les importants aspects les concernant.
- Le deuxième chapitre aborde la composition des services Web en générale : classification, langages de description, problèmes liés à celle-ci, différentes approches proposées pour y remédier.
- Le troisième chapitre illustre la méthode formelle EventB d'une manière assez détaillée, en citant les différents principes sur lesquels cette méthode est basée.
- Le quatrième chapitre comporte notre proposition, un modèle représenté formellement à base d'automates, et qui sera validé par la suite avec la méthode EventB.

Et enfin,nous terminons avec conclusion et perspectives.

1

GÉNÉRALITÉS : SERVICES WEB

1.1 Introduction

Le Web nous a servi pendant plus d'une vingtaine d'années. Il a été conçu pour les interactions entre humains et applications. Depuis quelques années, des efforts ont été déployés pour utiliser le Web comme une interface machine-à-machine par le biais de la notion de services Web. Les Services Web facilitent grandement l'échange d'information à l'échelle mondiale. En effet, depuis un simple ordinateur connecté au réseau, on accède à une multitude de contenus (tel que : achats et ventes en ligne) Ce chapitre représente une vue globale des services web, des définitions et les standards de base.

1.1.1 Définition 1

" les services web sont des applications auto-descriptives, modulaires et faiblement couplées qui fournissent un modèle de programmation et de déploiement d'applications, basé sur des normes et séxécutant au travers de línfrastructure Web "[4].

1.1.2 Définition 2

" Un service web est une application accessible à partir du Web il utilise les protocoles internet pour communiquer, et emploit un langage standard pour décrire son interface"[5].

1.1.3 Définition 3

" un service web est un système logiciel identifié par un URI, dont les interfaces publiques et les incarnations sont définies et décrites en XML. Sa définition peut être découverte dynamiquement par d'autres systèmes logiciels. Ces autres systèmes peuvent ensuite interagir avec le service Web d'une façon décrite par sa définition, en utilisant des messages XML transportés par des protocoles internet"[6].

Un service web peut être défini de différentes manières, mais toutes convergent vers une seule définition qui satisfait les points suivants[1] :

- Les services web proposent aux utilisateurs du web des fonctionnalités pratiques grâce à un protocole Web standard ;
- Les services web offrent un moyen de décrire leurs interfaces suffisamment en détail pour permettre à un utilisateur de créer une application cliente capable de converser avec eux. Cette description est généralement fournie dans un document XML [1] ;
- Les services Web sont publiés afin que les utilisateurs potentiels puissent les trouver facilement ;
- Un service Web ne doit être lié à un système d'exploitation ou à un langage de programmation.

1.2 Caractéristiques des Services Web

Les services web sont caractérisés par[1] :

- Une eprésentation de données basée sur XML.
- Une interface flexible ;
- Une habilité à interagir de manière synchrone et asynchrone ;
- un support des appels de fonctions distantes ;
- un support d'échange de documents.

1. L'Extensible Markup Language (« langage de balisage extensible » en français) est un langage informatique de balisage générique

1.3 Architecture des services web

L'architecture des services Web est une architecture orientée composant (SOA *Service Oriented Architecture*). L'architecture SOA est un modèle qui définit un système par un ensemble de composants logiciels distribués, les composants dans un système distribué n'opèrent pas dans un même environnement de traitement et sont obligés de communiquer par échanges de messages afin de solliciter des services dans le but d'accomplir le résultat souhaité. La définition de l'architecture des services Web consiste à mettre en évidence les concepts, ainsi que l'ensemble des relations et des contraintes entre ces concepts. Les principaux concepts intervenant dans l'architecture des services Web sont cités dans la table 1.1[2] :

Concepts	**Défintions**
le fournisseur du service	désigne le serveur qui héberge les services déployés
le client du service	représente l'application cliente qui invoque le service
le service	désigne les fonctionnalités d'un agent logiciel qui implémente le service
la description du service	c'est la spécification du service exprimée dans un langage de description interprétable par les machines, c'est-à-dire une description technique dans laquelle le service est vu en termes de messages, de types, de protocoles de communication et d'une adresse physique
les messages	c'est la plus petite unité d'échange entre les clients et les services. La structure des messages qui permettent l'invocation d'un service doit être exprimée dans la description du service
la ressource	désigne l'identifiant du service, c'est-à-dire son URI (*Uniform Resource Identifier*)

TABLE 1.1 – Ensemble de concepts et leurs définitions

La mise en oeuvre d'une application basée sur l'AOS repose sur un cycle de vie composé de trois phases **Préliminaire**, **Modélisation** et **Assemblage** [3],la figure 1.1 l'illustre [3] ;

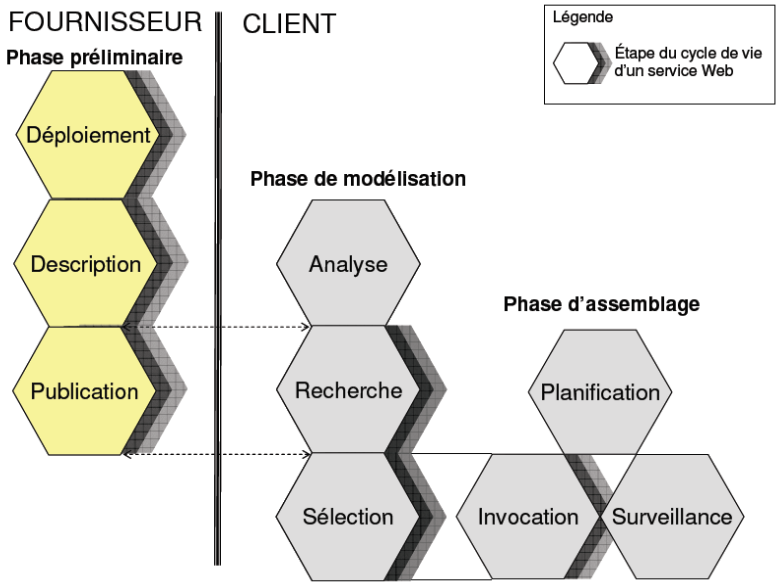


FIGURE 1.1 – Illustration du cycle de vie d'une application à base d'architecture orientée service

La première phase du cycle de vie est la **phase préliminaire** réalisée par un fournisseur de services élémentaires. Cette phase est composée de trois étapes :

- **Le déploiement** :est spécifique à chaque fournisseur et correspond à l'implémentation du service qui, par la suite, sera décrit et publié
- **La description** : Afin que le service Web déployé puisse être visible, dans le but d'être utilisé (et réutilisé), le fournisseur doit le décrire à l'aide du standard WSDL

- **La publication** : La recherche des services en vue de les utiliser (et réutiliser) repose sur leur publication préalable dans un registre. UDDI (Universal Description Discovery, and Integration) [Clement et al., 2004] avait à l'origine l'ambition de devenir le registre universel de services Web.

Les deux phases suivantes du cycle de vie sont réalisées par le **client** (le concepteur d'applications à base d'AOS). La **phase** de **modélisation** est composée de trois étapes :

- l'**analyse** des besoins préalable à la conception et se situe au niveau interne des organisations.
- **La recherche** : La recherche peut se faire via les registres dans lesquels les fournisseurs ont publiés leurs services Web.
- **La sélection** : Le client doit ensuite sélectionner parmi la collection de services Web issus de l'étape de recherche, le service Web qui convient le mieux à ses attentes.

La dernière phase du cycle de vie est la **phase d'assemblage**. Cette phase est composée de trois étapes :

- **La planification** : lors de cette étape, le client définit la composition de services Web.
- **L'invocation** lorsque le client connaît au préalable le service Web qu'il veut invoquer (par l'intermédiaire des étapes précédentes telles que la recherche et la sélection), son invocation peut être réalisée via le protocole de communication standard SOAP.
- **La surveillance** : l'exécution d'une composition de services Web engendre des problèmes tels que l'indisponibilité d'un service Web. Lorsqu'un service Web faisant partie d'une composition est indisponible, l'étape de surveillance doit reposer sur des mécanismes de compensation afin de sélectionner un nouveau service Web répondant au mieux aux objectifs du service Web défaillant.

Dans notre travail nous allons nous intéresser à la composition des services web, qu'on peut définir telle qu'une combinaison de plusieurs SW, pour n'en former qu'un qui satisfait les exigences du client.

1.4 Les standards de bases des Services Web

1.4.1 Le protocole SOAP (Simple Object Access Protocole)

SOAP est un protocole de transmission de messages basé sur XML. c'est un standard recommandé par le W3C [2],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

2. *Word Wide Web Consortium* :est une communauté internationale qui développe des standards pour assurer la croissance à long terme du Web

exécuter des dialogues requête-réponse RPC(Remote Procedure Call).SOAP permet une communication hétérogène, il est assez léger, simple facile à déployer, extensible et ouvert mais ne garde pas la trace des requêtes envoyées des diffèrents clients vers les services web invoqué[8].

Message SOAP

La structure d'un message SOAP est constituée d'un entête HTTP, une enveloppe, qui contient un entête et un corps, comme la figure 1.2 le montre[1] ;

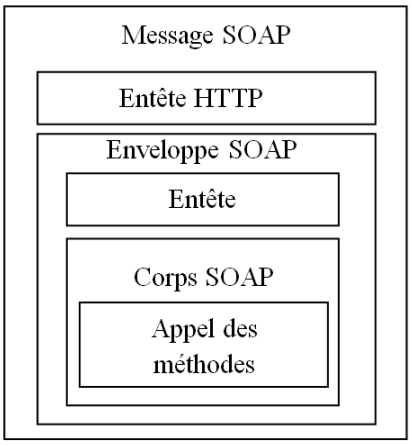


FIGURE 1.2 – Structure d'un message SOAP

- **Entête HTTP** : cet entête contient :
  - La version de HTTP utilisée.
  - La date de génération de la page.
  - Le type d'encodage du contenu.
- **L'enveloppe SOAP** (SOAP Envelope) : L'élément Envelope est obligatoire et sert de conteneur aux autres éléments du message SOAP, c'est la racine du document XML. Cette enveloppe est constituée de :
  - **L'entête SOAP (SOAP Header)** : L'élément Header est optionnel mais s'il est présent, il vient immédiatement après l'élément Envelope. Il permet d'intégrer au message SOAP des directives lui permettant d'externaliser certains services. Ainsi, le message contiendra différentes instructions destinées à être prises en charge par des services extérieurs.
  - **Corps SOAP (SOAP Body)** : L'élément Body Constitue un conteneur. Il contient les données spécifiques à l'application. Il est utilisé pour effectuer des appels de procédures à distance (RPC) ou transporter les rapports d'erreurs.

Formats de messages SOAP

Il existe deux formats de message SOAP [1] :

- Message SOAP sans pièces jointes : Un message SOAP sans pièces jointes comporte trois parties :
  - l'enveloppe (obligatoire)
  - L'entête (optionnel)
  - Le corps (obligatoire)
- Message SOAP avec pièces jointes : Le message peut comporter dans ce cas une ou plusieurs parties d'*attachment* ajoutées au message SOAP de base, si un contenu de message n'est pas au format XML, il doit apparaître dans la partie attachment.

1.4.2 WSDL (Web Services Description Language)

WSDL est un langage de la famille XML, un standard proposé par W3C permettant de décrire les types de données supportés et les fonctions offertes par un service Web. Ce standard est au coeur de la technologie des Services Web. Son utilisation est un critère déterminant pour qualifier un service de Service Web. La description d'un service doit inclure la définition des composants nécessaires au protocole de communication (SOAP pour les services Web) et à l'interaction avec un client ou un autre service Web. Les problématiques de réutilisation et d'interaction ont guidé le W3C afin de définir les catégories d'information à prendre en compte dans la description d'un service Web. Les différents éléments décrits dans WSDL sont les suivants [3] :

- **Les opérations proposées** par le service Web ;
- **Les données** et **messages** échangés lors de l'appel d'une opération ;
- **Le protocole** de communication ;
- **Les ports d'accès** au service.

Dans WSDL, il existe une séparation entre deux niveaux indépendants, respectivement nommés *abstrait* et *concret*. Le niveau abstrait regroupe les informations pouvant être réutilisées (non spécifique à un service), tandis que le niveau concret est constitué de la description des protocoles d'accès au service Web (information particulière à un service). Le niveau abstrait est utilisé principalement lors du processus de sélection, tandis que le niveau concret est seulement utilisé lors de l'invocation des méthodes du service Web.

1.4.3 UDDI (Universal Description Discovery and Integration)

Proposé par le consortium OASIS (*Organization for the Advancement of Structured Information Standards*) afin de publier et rechercher des services Web, UDDI a été conçu en vue de devenir le registre standard de la technologie des services Web. Pour convenir à la technologie des services Web, les services référencés dans UDDI sont accessibles par l'intermédiaire du protocole de communication SOAP ;la publication des informations concernant les fournisseurs et les services doit être spécifiée en XML afin que la recherche et l'utilisation soient faites de manière dynamique et automatique[3]. La communication entre un client et un service passe par une phase de découverte et de localisation de ce service, à l'aide du protocole SOAP et des annuaires UDDI[4].

Principe de l'annuaire UDDI

UDDI permet de classer et de rechercher des services Web. Si l'accès à l'information est trop élevé, le recours aux services web ne devient plus intéressant, c'est pourquoi le principe d'annuaire universel a été mis en place. Cette API est composée de deux bibliothèques [1] :

- API de **requête** : cela consiste à effectuer des interrogations, des recherches d'information sur les entrées d'un UDDI. Tout utilisateur a la possibilité d'effectuer des recherches.
- API de **publication** : pour permettre la publication ou la suppression d'un service dans un annuaire. Cette API impose des autorisations d'accès et est destinée aux fournisseurs de services et aux entreprises désirant publier des informations les concernant.

Composant de l'annuaire UDDI

UDDI est composé de deux parties [1] :

a) L'*UDDI Business Registry* : Annuaire d'entreprise et de services web contenant des informations, au format XML, organisées en trois catégories [20] :

- **Les pages blanches** : nom, adresse, contacts, et identifiants des entreprises enregistrées. Ces informations sont décrites dans des entités de type *Business Entity*. Cette description inclue des informations de catégorisation permettant de faire des recherches spécifiques dépendant du métier de l'entreprise.
- **Les pages jaunes** : comportent les détails sur le métier des entreprises et les services qu'elles proposent. Ces informations sont décrites dans des entités de type *Business Service*.
- **Les pages vertes** :regroupent les informations techniques sur les services proposés. Elles

incluent des références

b) *Les interfaces d'accès à ces annuaires*, et *les modèles de données* qui comportent cinq structures de données principales :

- *BusinessEntity* : Les informations concernant l'entreprise qui publie ses services Web dans l'annuaire et le type de services qu'elle offre sont contenues dans la structure " BusinessEntity " et correspond notamment aux pages blanches concernant l'entreprise.
- *BusinessService* : cet élément contient des informations sur les services métiers. Il s'agit des informations de description, son code. Une entreprise peut enregistrer plusieurs services. Le type d'information contenu dans l'élément " BusinessService " correspond aux informations pages jaunes d'une entreprise.
- *BindingTemplate* (informations de liaison) : Elles contiennent des informations techniques sur un service Web. Elles servent également de pages vertes de l'annuaire UDDI. Il s'agit des informations indiquant les références aux " tModels " désignant les spécifications de l'interface pour un service Web, ainsi que le point de terminaison (adresse internet) de ce dernier. Ces informations aident le client à se connecter puis à invoquer le service désiré, un service peut avoir plus d'un modèle de liaison.
- *Tmodel (information technique)* : La structure " modèle " a pour rôle de décrire les spécifications des services Web à enregistrer. Elle comporte les informations permettant de connaitre les normes auxquelles le service est conforme afin d'avoir une description précise du service. Le nom doit doit être présenté suivant le format URI et doit pointer vers l'emplacement du fichier correspondant, généralement au format WSDL.
- *PublishAssertion* (Assertion contractuelle entre partenaires) : Met en correspondance deux ou plusieurs structures " BusinessEntity " selon le type de relation (Filiale de, Département de).Cette structure comporte les assertions contractuelles entre organismes pour les services publiés. Ces assertions représentent un ensemble de règles contractuelles d'invocation de services représentées sous la forme de protocoles entre deux partenaires métiers. Chaque rôle entre partenaires est défini à travers ces assertions.

1.5 Service Web et Web Sémantique

Les informations du Web actuel ne sont compréhensibles que par les humains, les machines peuvent lire ces informations, mais elles sont incapables de les interpréter. Le Web est limité, et ne permet pas un réel partage du savoir ; tout au plus, il permet de présenter des connaissances, mais en aucun cas de ne les rendre directement utilisables [9]. *Selon Tim Berners-Lee* [13] : " *Le*

Web sémantique n'est pas un Web distinct mais bien un prolongement du Web que l'on connaît et dans lequel on attribue à l'information une signification clairement définie, ce qui permet aux ordinateurs et aux humains de travailler en plus étroite collaboration "

"L'idée du Web sémantique ne s'agit pas de faire en sorte que les ordinateurs puissent comprendre le langage humain ou fonctionner en langage naturel, ni de créer une intelligence artificielle permettant au web de réfléchir, mais simplement de regrouper les informations de manière utile, où elles seront comprises par les ordinateurs afin d'apporter à l'utilisateur ce qu'il cherche vraiment",définie par W3C.

1.5.1 Architecture du Web Sémantique

Comme toute technologie, le Web sémantique nécessite une architecture (exprimée par la figure 1.3) recommandée par le W3C partagée par tous pour échanger des ressources sur l'Internet. Elle s'appuie sur une pyramide de langages, proposée par son inventeur Tim Berners-Lee pour représenter des connaissances sur le web en satisfaisant les critères de standardisation, interopérabilité et flexibilité. Il s'agit de la vision schématique du web sémantique[15] ;

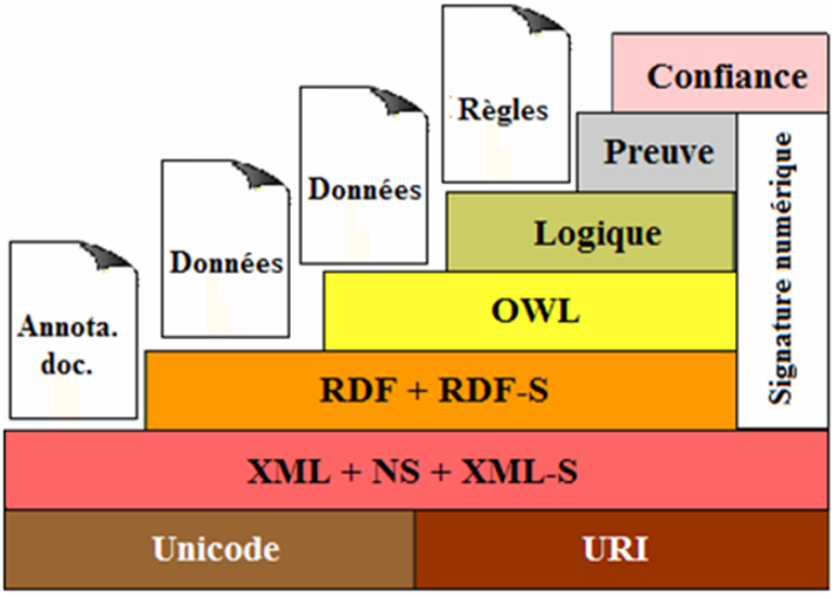


FIGURE 1.3 – Les couches du Web sémantique

La liste suivante introduit la fonction principale de chaque couche dans l'architecture du web sémantique [16] :

– L'**URI** (*Uniform Resource Identifier*) est un protocole simple et extensible pour identifier, d'une manière unique et uniforme, toute ressource sur le web. Il s'agit d'un aspect central de

l'infrastructure, c'est pour cette raison que cet élément se trouve à la base de l'architecture en couches proposée.

- les données sont toujours encodées avec le jeu de caractères Unicode pour un maximum d'interopérabilité. C'est pourquoi cet élément figure dans cette couche de bas niveau, au même titre que l'URI.
- Les couches **Preuve** et **confiance** supportent un mécanisme de communication inter-agent pour valider les résultats de raisonnement. Cela pourra assurer la fiabilité des services automatiques du web sémantique.
- La couche **Logique** s'appuie sur des règles d'inférence qui permettent le raisonnement intelligent exécuté par des agents logiciels.
- La couche **Ontologie**, fondée sur une formalisation commune, spécifie la sémantique de métadonnées fournies dans le web sémantique.
- La couche **RDF** [3] représente les métadonnées pour les ressources web.
- XML est utilisé comme couche de base syntaxique du web sémantique. Le langage XML est actuellement considéré comme standard de transport de données sur le web.

1.5.2 Les ontologies

Le Web sémantique est une infrastructure pour permettre l'utilisation de connaissances formalisées (ontologies) en plus du contenu informel actuel du Web. L'objectif principal du Web sémantique est le partage et la réutilisation d'information pour l'homme et la machine, la connaissance est ajoutée nécessairement à l'information pour améliorer la compréhension de la machine. L'Ontologie a joué un rôle très important pour le Web sémantique, en représentant la sémantique des documents et permettant son exploitation par des applications et des agents intelligents. Elle est très utile pour structurer et définir la signification des termes méta-données. D'où à l'aide des ontologies, les applications sur Web de demain pourront devenir intelligentes, au sens où elles pourront opérer plus précisément au niveau conceptuel humain [17].

D'après encyclopédie Wikipédia, en informatique, une ontologie est un ensemble structuré de concepts. Les concepts sont organisés dans un graphe dont les relations peuvent être :

1. Des relations sémantiques ;
2. Des relations de composition et d'héritage (au sens objet).

La structuration des concepts dans une ontologie permet de définir des termes les uns par rapport aux autres, chaque terme étant la représentation textuelle d'un concept. Par exemple, pour décrire

3. Resource Description Framework (RDF) est un modèle de graphe destiné à décrire de façon formelle les ressources Web et leurs métadonnées, de façon à permettre le traitement automatique de telles descriptions. Développé par le W3C, RDF est le langage de base du Web sémantique.

les concepts entrant en jeu dans la conception de cartes électroniques, on pourrait définir l'ontologie suivante[4] :

- Une carte électronique est un ensemble de composants
- Un composant peut être soit un condensateur, soit une résistance, soit une puce.
- Une puce peut être soit une unité de mémoire, soit une unité de calcul ;
- Une carte électronique qui contient une unité de calcul contient aussi au moins une unité de mémoire.

1.5.3 Le langage OWL-S

OWLS(*Ontology Web Langage Semantic Services*) est un langage de description de services basé sur XML utilisant le modèle des logiques de descriptions (une proposition d'ontologie des services Web). Il est aussi un langage de haut niveau pour la description et l'invocation des services Web dans lequel la sémantique est incluse et interface avec UDDI/WSDL/SOAP. Il propose une sémantique des données, et pas seulement des champs prédestinés par la structure des standards utilisés pour décrire les services. OWL utilise des logiques de descriptions pour modéliser les services, donc, il nous permet une grande puissance d'expression, que ne possèdent pas les autres systèmes. Si des machines pouvaient comprendre, et donc interpréter, le contenu des pages Web, alors la recherche d'information avec une grande précision serait possible. Si l'appui sur les traitements de la langue naturelle n'est toujours pas possible en général, le recours à des langages formels permettant de décrire le contenu plutôt que la représentation ouvre de nouvelles perspectives. Bien que XML soit un outil standard qui a apporté une vision originale au problème de l'interopérabilité, la divergence dans le vocabulaire le rend difficile d'utilisation pour la recherche d'information sur le Web. Un langage approprié pour la spécification de la sémantique du Web doit supporter les ontologies de façon à répondre à des besoins qui sont [17] :

- Partage d'ontologies : les ontologies doivent être disponibles de telle sorte que différentes sources de données puissent les accéder pour le partage de la sémantique.
- Evolution des ontologies : les ontologies peuvent changer au cours du temps et les sources de données doivent spécifier quelle version de l'ontologie qu'elles utilisent.
- Interopérabilité des ontologies : Différentes ontologies peuvent modéliser les mêmes concepts de différentes manières. Le langage doit fournir des primitives pour relier différentes représentations, permettant ainsi à des données d'être converties dans différentes structures.
- Compromis expressivité-utilisabilité : le langage doit pouvoir permettre l'expression d'une variété de connaissances, mais doit aussi pouvoir offrir des possibilités de raisonnement sur ces connaissances. Comme ces deux objectifs sont en général antagonistes, le but du langage est de

trouver un compromis qui permet d'exprimer les types de connaissances les plus importants.
- Facilité d'utilisation : le langage doit nécessiter un moindre effort d'apprentissage et avoir des concepts et une sémantique clairs. Les concepts doivent être indépendants de la syntaxe.

1.6 Avantages,inconvénients des services web

Les services web permettent à des portions de logiciels écrits dans des langages différents ou évoluant sur différents systèmes d'exploitation, de communiquer facilement entre elles. Des applications supportant différents processus d'une ou plusieurs organisations peuvent, aussi, communiquer entre elles et s'échanger des données grâce aux services web (Babin et Leblanc, 2003). Selon Wiképédia,les avantages sont :

- Les services Web fournissent l'interopérabilité entre divers logiciels fonctionnant sur diverses plates-formes.
- Les services Web utilisent des standards et protocoles ouverts.
- Les protocoles et les formats de données sont au format texte dans la mesure du possible, facilitant ainsi la compréhension du fonctionnement global des échanges.
- Basés sur le protocole HTTP, les services Web peuvent fonctionner au travers de nombreux pare-feu sans nécessiter des changements sur les règles de filtrage.
- Les outils de développement, s'appuyant sur ces standards, permettent la création automatique de programmes utilisant les services Web existants.

Par contre la technologie des services web comporte aussi plusieurs inconvénients qui sont :

- Problèmes de performance : Les services web sont encore relativement faibles par rapport à d'autres approches de l'informatique répartie telles que CORBA ou RMI.
- Confiance : Les relations de confiance entre différentes composantes d'un service web sont difficiles à bâtir, puisque parfois ces mêmes composantes ne se connaissent même pas.
- Syntaxe et sémantique : On se concentre beaucoup sur comment invoquer des services (syntaxe) et pas assez sur ce que les services web offrent (sémantique).
- Fiabilité : Il est difficile de s'assurer de la fiabilité d'un service car on ne peut garantir que ses fournisseurs ainsi que les personnes qui l'invoquent travaillent d'une façon fiable.
- Disponibilité : Les services web peuvent bien satisfaire un ou plusieurs besoins du client. Seront-ils pour autant toujours disponibles et utilisables ? Ça reste un défi pour les services web.

1.7 Conclusion

Dans ce chapitre nous avons parlé des généralités des services web, commençant par des définitions, tout en précisant l'architecture utilisée, ainsi que les standards de base. Par la suite, nous avons abordé le web sémantique, son impact sur les services web. Dans le chapitre qui suit, nous allons mettre en évidence les solutions proposées pour résoudre les problèmes rencontrés lors de la composition des services web.

2

COMPOSITION DE SERVICES WEB : ÉTAT DE L'ART

2.1 Introduction

Les services Web sont considérés comme autonomes, auto-descriptifs, qui peuvent être publiées, localisés et invoqués à travers le Web. De nos jours, un nombre croissant d'entreprises et d'organisations[17], mettent en oeuvre cette technologie, et tentent en vain de satisfaire les exigences de leurs clients, si un service ne répond pas à la requête du client, il doit y avoir une possibilité de combiner plusieurs SW pour la satisfaire. Ce fait de combiner crée un important flux de recherches pour résoudre les problèmes liés à la composition des services web. Dans ce chapitre nous présentons les importantes approches existantes.

2.2 Définition de la composition des services web

La composition de services consiste à combiner les fonctionnalités de plusieurs services dans le but de répondre à des demandes complexes qu'un seul service ne pourrait pas satisfaire [10]. La

composition de services vise à faire intéropérer, interagir et coordonner plusieurs services pour la réalisation d'un but donné[11]. La figure 2.1 [11] illustre le principe de la composition de services à partir d'un ensemble de services disponibles dans un registre (annuaire).

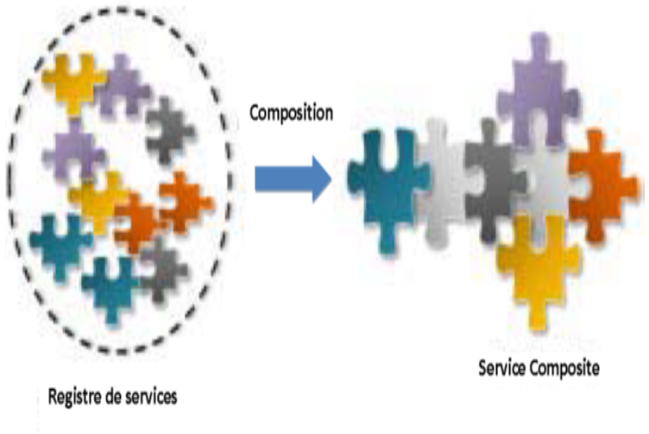


FIGURE 2.1 – Principe de composition de services

il existe deux domaines d'application de la composition de services Web (*e-business* et *Web sémantique*). Cependant, chacun d'eux utilise la composition à des fins qui lui sont propres :

- **L'e-business** : Les organisations utilisent des méthodes afin de représenter les flots de données inter et intra entreprises. Les concepteurs des applications d'entreprise (ou les experts métiers) définissent au préalable les processus. La définition du processus repose sur la spécification du besoin, des flots de contrôle entre les activités, et des contraintes. La composition de services Web permet d'attribuer à chaque activité du processus un service Web [55].
- **Le Web sémantique** : Les travaux dans le domaine du Web sémantique rendent le Web exploitable par les machines elles-mêmes, sans intervention humaine. On parle alors de services Web sémantiques. Dans le contexte de la composition de services Web, le Web sémantique intègre l'automatisation aux processus de découverte, d'invocation, et de surveillance des services Web [55].

2.3 La classification des méthodes de composition de services

Nous passons en revue les différentes techniques de composition de services Web. Nous pouvons classer ces techniques en deux grandes familles : les techniques de composition dites " *statiques* ",

qui sont définies à l'aide de processus métier (orchestration et chorégraphie) ; et les techniques de composition dites " *dynamiques* ", où les services web sont composés dynamiquement en fonction de leurs fonctionnalités mais qu'aucune technologie n'implémente à l'heure actuelle. Nous nous intéressons à la technique *statique*, qui est divisée en deux classes ;

- **Orchestration de services Web** L'orchestration décrit la manière dont les services Web peuvent interagir ensemble au niveau des messages, incluant l'ordre d'exécution des messages et la logique métier.En d'autres termes, c'est un ensemble de processus exécutés dans un ordre prédéfini afin de répondre à un but. Dans l'orchestration, un seul processus (appelé orchestrateur) est responsable de la composition et contrôle les interactions entre les différents services. Cet orchestrateur coordonne de manière centralisée les différentes opérations des services partenaires qui n'ont aucune connaissance de cette composition. BPEL est reconnu comme un langage standard d'orchestration de services Web[19]. Figure 2.2[3] illustre l'orchestration, La requête du client (logiciel ou humain) est transmise au moteur d'exécution (Moteur). Ce dernier, d'après le processus préalablement défini, appelle les services Web (ici, SW1, SW2, SW3 et SW4) selon l'ordre d'exécution.

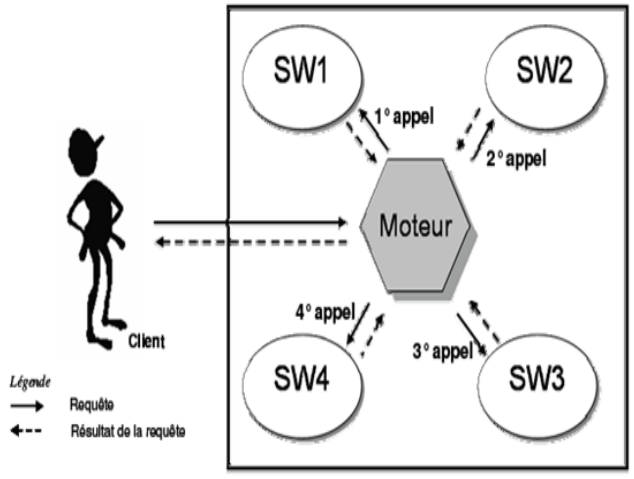


FIGURE 2.2 – Vue générale de l'orchestration

- **Chorégraphie de services Web** La chorégraphie décrit une collaboration entre services pour accomplir un certain objectif. A l'inverse de l'orchestration, la chorégraphie ne repose pas sur un seul processus principal. C'est une coordination décentralisée où chaque participant est responsable d'une partie du workflow, et connaît sa propre part dans le flux de messages échangés.WS-CDL [1] est l'un des langages de description de chorégraphie pour les services

1. WEB Service Choreography Description Langage est un langage basé sur XML qui décrit pair-à-pair les collaborations des parties en définissant, à partir d'un point de vue global, leur comportement observable commun

Web. Il peut être considéré comme un complément à un langage d'orchestration tel que BPEL en lui apportant un modèle global nécessaire pour assurer la cohérence des interactions des services partenaires, et pour prendre en considération les éventuelles situations de compétition entre partenaires[19]. La Figure 2.3 [3] montre une vue générale d'une composition de services Web de type chorégraphie. Le client (logiciel ou humain) établit une requête qui est satisfaite par l'exécution automatique de quatre services Web (SW1, SW2, SW3 et SW4). La requête de l'utilisateur est transmise au premier service Web (SW1) qui est exécuté. Le SW1 découvre ensuite le service Web lui succédant. Une fois le service découvert, les deux services (SW1 et SW2) échangent des messages, le résultat de l'action du SW1 est transmis au SW2 qui l'utilise comme paramètre d'entrée. Le SW4 termine le processus et le résultat de son action est transmis au client[3].

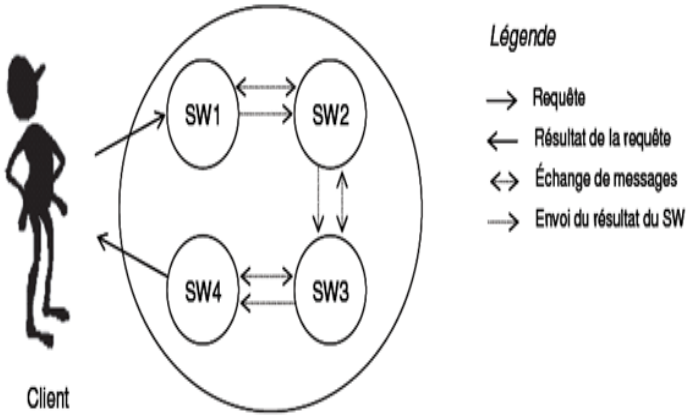


FIGURE 2.3 – Vue générale de la chorégraphie

En résumé, la différence majeure entre la chorégraphie et l'orchestration est que la chorégraphie offre une vision globale et plus collaborative de la coordination, alors que l'orchestration offre une vision centralisée de la composition.

2.4 Langages de composition de services web

Les langages de description de compositions de services Web sont basés sur une syntaxe XML et sont généralement décrits à l'aide d'un schéma XSD [2]. La phase de description lors de la composition des Services Web est décisive, pour cela diverses approches ont proposé de nouveaux langages de

et complémentaire

2. Abréviation pour XML Schema Definition. Il s'agit d'une extension du nom d'un fichier qui comporte une définition XML Schema.

balisage basés sur XML " Langages d'orchestration et de chorégraphie ", qui offrent différents mécanismes pour la composition et le contrôle des invocations entre services. Alors que WSDL permet de décrire les opérations sous forme de signature entrées/sorties, ces langages permettent de décrire comment sont utilisées ces opérations (signature comportementale). En d'autres termes, ces langages proposés complémentent en quelques sortes le langage WSDL. On distingue deux catégories de langages de description de services Web composés[39] :

2.4.1 Langages de description de services sémantiques

Nous citons les langages OWL-S [W3C, 2004a] et WSMO [CMS-WG, 2006] qui sont les plus connus.Ces derniers permettent de décrire une composition de services Web sous forme d'un processus orchestrant des activités. Chaque activité peut avoir des informations sémantiques, provenant d'une ontologie, sur sa fonctionnalité, son rôle et les données qu'elle manipule. Une activité peut avoir comme rôle l'invocation d'un service Web partenaire. Un lien vers le fichier WSDL, décrivant ce service, est défini[27].

2.4.2 Langages de description de services sans références sémantiques

Dans cette catégorie il existe des langages dédiés à la description de la chorégraphie, et des langages dédiés à la description de l'orchestration de services,et d'autres aux deux.

Langages d'orchestration de services web

- **BPEL**(*Business Process Execution Language*)[27] BPEL est un langage basé sur une syntaxe XML modélisant un processus métier par une composition d'un ensemble de services Web élémentaires. Chaque processus BPEL peut être considéré également comme un service Web. La spécification BPEL utilise des normes W3C : WSDL [W3C, 2004b] pour la description des services Web partenaires, XML Schéma [W3C,2001] pour la définition des structures de données, et XPath [3] [W3C, 1999] pour la récupération d'éléments XML.
- **WSFL**(*Web Service Flow Language*)[39] Dialecte XML qui sert à décrire un processus métier et à y associer toutes les informations à l'invocation des services Web sous-jacents . Il a été conçu par IBM pour faire partie du cadre des techniques de services Web.WSFL considère deux types de compositions : flowModel et globalModel.Le flowModel spécifie le processus exécutable, alors que le globalModel spécifie la collaboration entre les participants(business partners).

3. XPath est un langage (non XML) pour localiser une portion d'un document XML.

– **BPEL4WS**(*Business Process Execution Langage for Web Services*)[40] BPEL4WS est l'un des langages les plus utilisés pour la description de l'orchestration d'un ensemble de services Web. Le langage BPEL4WS est le resultat de la fusion des deux langages de description de Workflow : XLANG et WSFL. BPEL4WS permet de décrire d'une part l'orchestration d'un ensemble de services Web à travers la description de processus exécutables, et d'autres part, la chorégraphie d'un ensemble de services Web.

Langages de chorégraphie de services Web

– **WSCI**(**Web Services Choregraphy Interface**) [41]
Est un langage reposant sur XML.Il propose de se focaliser sur représentation des services Web en tant qu'interfaces décrivant le flux de messages changés (la chorégraphie de messages). Il propose ainsi de décrire le comportement externe observable du services. Pour cela, WSCI propose d'exprimer les dépendances logiques et temporelles entre les messages échangés à l'aide de contrôles de séquences, corrélation, gestion des erreurs et transactions. On remarque que WSDL et ses définitions abstraites sont réutilisés afin de pouvoir également décrire par la suite les modalité de concrétisation des éléments manipulés pour modéliser un service.

– **WS-CDL** (*Web Services Choregraphy Description Language*)[42]
WS-CDL est un langage propose par le W3C, et il est susceptible de prendre le dessus pour la spécification de la collaboration entre services Web.WS-CDL est un langage basé sur XML pour la description de la collaboration pair-à-pair entre un ensemble de services Web. Il permet de définir le comportement externe, en termes de messages échangés entre un ensemble de SW qui collaborent pour atteindre un objectif commun. Les auteurs de WS-CDL , se sont inspirés des langages WSCI,XLANG,WSFL,BPEL,BPML pour construire un langage complet répondant à la majorité des besoins des entreprises en terme de puissance de modélisation.

Nous pouvons souligner qu'il existe aussi d'autres langages qui sont dédiés aux deux techniques(chorégraphie et orchestration) comme BPMN [OMG, 2010]. Malgré les différences syntaxiques existantes, ces langages s'accordent sur les constructeurs offerts et sur la capacité à traiter les problèmes suivants [Shapiro,2001][Kazhamiakin, 2007][27] :

– Représentation des différentes propriétés comportementales de la composition. Ceci inclut la nécessité de modéliser les interactions entre services (échanges de messages) ; contraintes de contrôle de flux (i.e., la séquence, le branchement conditionnel, l'exécution itérative des activités) ; contraintes de flux de données (i.e., l'échange, la modification, l'évaluation des données et des expressions de données) ;

– Gestion des différents modes d'exécution des services. En dehors de la modélisation du flux d'exécution normal, le langage est destiné à représenter le comportement exceptionnel et transactionnel, fournissant des mécanismes de coordination, de récupération et de compensation d'un comportement anormal du service. Le problème de la prise en charge du comportement transactionnel est particulièrement difficile dans le cas de la composition de services Web. Il est nécessaire de traiter de longues transactions où les processus peuvent durer des jours, des semaines voire des mois. Ceci nécessite également la prise en compte explicite des contraintes de temps.

2.5 Problème rencontré et solutions proposées

Dans le Web d'aujourd'hui, les services Web sont créés et mis à jour si rapidement que leur analyse est déjà au-delà de la capacité humaine[17], ainsi que la génération manuelle de leurs plans de composition. Nombreuses approches ont été fournies pour combler ces lacunes, tout en s'inspirant des recherches faites dans les entreprises généralement sur le plan industriel, Dans ce qui suit, nous présentons brièvement, les travaux existants pour la résolution du problème de la composition

2.5.1 Approches existantes

Différents travaux de recherche s'intéressent au problème de la composition de service, des approches qui ont chacune des manières différentes de considérer les services, la notion de composition et la représentation du but. Nous allons décrire quelques-uns de ces travaux.

2.5.1.1.Approches par planification

La recherche dans la planification est souvent concentrée sur l'automatisation de la composition des services web, étant perplexe, de nombreux travaux ont été réalisés ces dernières années.

Les travaux de Sheshagiri et al[23], proposent l'utilisation d'ontologies de domaines afin d'ajouter des contraintes pour optimiser la recherche dans un espace de plans.

Ceux de McDermott [24] proposent d'utiliser la planification par régression estimée (*Estimated-Regression Planning*) qui permet d'utiliser des heuristiques pour guider la recherche dans un espace de situations. Le principe est le suivant : un agent qui désire interagir avec un service Web va construire un plan puis l'exécuter. Si l'exécution échoue, l'agent aura appris de nouvelles informations de cette interaction, qui seront utilisées dans le prochain cycle de planification et exécution.

2.5.1.2.Approche par la composition orientée contexte

Un modèle de composition de services Web, nommé ProbCWS (*Problem-solving for Composition of Web Services*)[3], basé sur la méthode de résolution de problèmes à base de tâches. Le problème à résoudre représente *la composition* à exécuter, et les tâches sont les *services Web*.

- Cette méthode a été choisie du fait qu'elle répond à un ensemble d'aspects relatifs à la composition de services Web : Ce type de méthode permet d'impliquer dans la définition de la planification de services Web la mise en oeuvre de l'adaptation au contexte d'utilisation.
- Le moteur d'inférence qui exécute les méthodes de résolution de problème supporte la *surveillance* et ainsi permet de mettre en œuvre ***la compensation*** de services Web

2.5.1.3.Approches par méthodes formelles

Selon [25],[26] Les méthodes formelles sont bien adaptées pour la spécification, la vérification et la validation de la composition de service web.

Il existe plusieurs formalismes pour la modélisation de la composition de services web. Nous pouvons classer ces modèles selon les familles suivantes : réseaux de Pétri, automates, algèbre de processus et machine d'états abstraite.dans notre travail nous allons donc, nous baser sur la méthode B Événementiel pour la modélisation et la vérification d'une Communication d'Automates à Etats Finis Complexes(CAEFC).

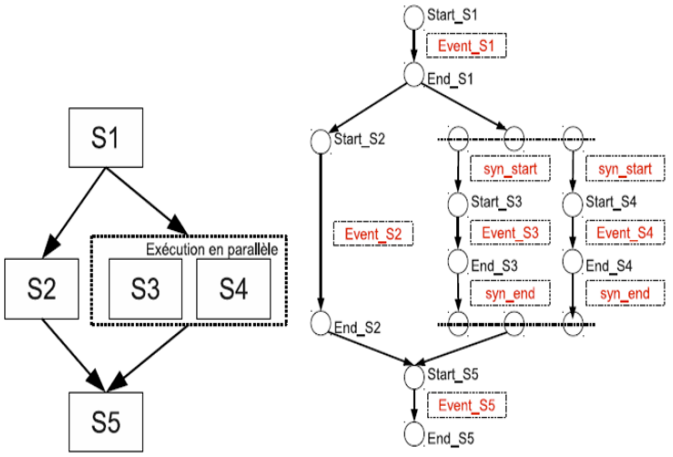
2.5.1.3.1. Modélisation par automates

La modélisation par Automates à Etats Finis est représentée par un graphe qui contient des nœuds ou états et des transitions, chaque automate représente généralement un service, les transitions sont des échanges de messages, ce qui impliquent des changements dans le système,un état est dit complexe lorsqu'il représente à son tour un automate. Selon [20],la composition de services se fait en trois étapes : Premièrement, trouver tous les services web dans l'annuaire de la Figure 2.1 qui peuvent être utilisés dans cette composition. Ceci est basé sur l'exécution de l'algorithme BFS (Breadth First Search) qui consiste à retourner l'ensemble de noeuds en choisissant le chemin le plus court s'il en existe plusieurs. Deuxièmement, il faut trouver la composition qui satisfait la requête. Et enfin une description du service composite qui indique l'enchaînement des appels aux services sélectionnés

On modélise un service par une machine à état fini tel que P=(S, s0, F , M, R) dont [20] :

- S : est l'ensemble d'états finis : ils définissent les différentes phases qu'un service peut mettre durant son interaction avec le demandeur et vice versa.

– s0 : est l'état initial.
– F : (F ⊆ S) est l'ensemble d'états finaux.
– R : est l'ensemble de fonctions de transitions de l'automate. On identifie l'état source, l'état destinataire et la fonction produite durant cette transition (s, s', f).



(a) Scénario de composition (b) Modélisation avec les automates

FIGURE 2.4 – Exemple de modélisation de service composé avec les automates

Nous citons les travaux pertinents qui englobent la composition des services web à bases d'automates ;

San-Yih Hwanget et al. proposent dans [20] une solution pour le problème de la sélection dynamique lors de la modélisation du WS composite par automates, appelée AR (Aggregated Reliability), elle consiste à modéliser la composition sous forme d'une chaine de Markov, plus les états existants, ajoutant deux états supplémentaires, succès et échec. Qui détermine une terminaison réussite ou échouée. Tel que Pour chaque paire de ci et cj d'états dans la chaine de Markov, une probabilité de transition Pij est associée.

Zobeyr Farah et Abdelkamel Tari ont proposé dans [21] un modèle nommée RLRA(Reverse Leap Reachability Analysis)pour résoudre le problème de l'explosion combinatoire de l'espace des états globaux , l'approche utilisée est celle avec retour arrière, le but est de réduire le nombre d'états examinés (réduction de l'espace des états globaux) , tout en préservant l'importante capacité de détecter les impasses(les erreurs d'inter-blocage).Son principe est d'identifier initialement un ensemble de tous les états suspectés, qui peuvent contenir des impasses. Puis raffiner l'ensemble précédent, tout en sélectionnant les états qui peuvent probablement causer les impasses, par la suite faire un retour arrière, tout en vérifiant les erreurs jusqu'à ce qu'on atteint la racine.

2.5.1.3.2. Modélisation sous forme de Réseau de Pétri

Un réseau de Pétri est un graphe dirigé, connecté, où chaque noeud est soit une place, soit une transition. Des jetons occupent les places. Quand il y a au moins un jeton dans chaque place connectée à une transition, nous parlons d'une transition activée (enabled). Suite à cette activation, un jeton est enlevé de chaque place en amont (input place) et mis dans une place en aval (output place) [Hamadi et al., 2003]. Un service web est modélisé sous forme de réseau de Pétri en associant les *opérations* aux transitions et les états de service aux places [Hamadi et al., 2003]. Ensuite, pour composer des services web, il suffit d'appliquer des opérateurs de composition aux réseaux de Pétri représentant les dits services web (chaque service web est représenté par un réseau de Pétri). Un réseau de Pétri avec une place d'entrée (input place) pour recevoir des informations et une place de sortie (output place) pour émettre des informations, facilite la définition des opérateurs de composition, ainsi que l'analyse et la vérification de certaines propriétés comme l'accessibilité et l'absence de cycles.

Un réseau de Pétri est un 3-uplet <S,T,W>, où [20] :

- S est un ensemble fini de places.
- T est un ensemble fini de transitions
- S et T sont distincts, aucun objet ne peut être à la fois une place et une transition
- W est un multiensemble d'arcs. Il définit les arcs et il assigne à chacun une valeur entière positive qui indique combien de jetons sont consommés depuis une place vers une transition, ou sinon, combien de jetons sont produis par une transition et arrivent à chaque place. Notez qu'un arc ne peut pas connecter deux places ou deux transitions.

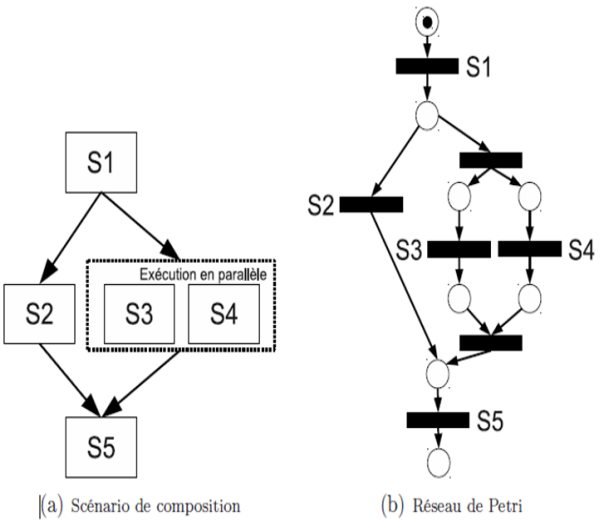


FIGURE 2.5 – Exemple de modélisation par réseau de Pétri d'un service composé

Enfin, les réseaux de Pétri fournissent un langage indépendant de tout outil, n'étant pas basé sur un logiciel spécifique. Nous présentons quelques travaux effectués à ce propos,la majorité des travaux utilisant ce formalisme, transforme une description d'un processus BPEL en un modèle de réseau de Petri. Ainsi, le modèle obtenu est analysé à l'aide des techniques et outils définis autour de cette technique formelle.

Christian Stahl et Schmidt dans[51] [52] ont défini une transformation d'une description BPEL en un réseau de Petri, cette transformation est assez complète, car elle décrit tous les comportements d'un processus BPEL, y compris la gestion des exceptions. Le processus de transformation est automatisé dans l'outils BPEL2PN[53],[54].

2.5.1.3.3. Approches par la méthode EventB

Les approches proposées se concentrent généralement sur la description dans les langages connus, tels que WSDL et BPEL. Le principe est de proposer un modèle d'une manière formelle qui permettra la vérification, et la validation, c'est-à-dire que les problèmes de dis-fonctionnement tels que l'interblocage ne seront pas présents dans le modèle. Les solutions citées utilisent la méthode EventB (ie B Evénementiel), qui sera détaillée dans le chapitre qui suit ;

Hong Anh et Ninh Thuan Truong dans [32] proposent une approche pour formaliser et vérifier la description de la chorégraphie avec le langage WS-CDL, l'absence de spécification formelle lors de la description dans ce langage en est une lacune. Pour faire face à ce problème,la méthode citée précédemment a été utilisée et un modèle d'interactions pour la chorégraphie a été proposé et traduit en un modèle formel.Les entités du WS-CDL sont donc, transformées en éléments d'EventB, le modèle résultant est par la suite vérifié et interprété. Un exemple a été simulé pour illustrer cette approche proposée.

Idir Ait Saadoune dans [27] propose une approche avec la méthode B Événementiel pour modéliser une composition de services Web décrite avec le langage BPEL. Le but principal de cette approche est la vérification formelle des propriétés comportementales d'un processus BPEL. Les activités simples et structurées sont les éléments offerts par le langage BPEL pour décrire le comportement du processus BPEL. Une représentation de ces activités avec B Événementiel permet de modéliser et de vérifier formellement le comportement d'une composition de services Web.

2.6 Synthèse et Discussions

La composition de services Web existants dans l'annuaire, offre un nouveau service Web avec une fonctionnalité plus complexe et plus satisfaisante pour le client. Faire fonctionner des services Web

ensemble, alors qu'à la base, chaque service Web est conçu pour fonctionner indépendamment, pose des problèmes d'interopérabilité. S'assurer à priori que la composition de services Web se comporte correctement est un avantage certain. Ce problème est traité par un grand nombre de travaux de recherche. Les deux premières approches (planification et contexte d'utilisation) se sont plutôt des approches qui explorent les techniques dynamiques, vu qu'elles penchent surtout sur l'aspect sémantique, nous les avons mentionnés pour donner une vue générale des approches proposées pour remédier aux problèmes de la composition des services, nous nous intéressons aux approches formelles.

En parcourant les différents travaux existants dans le domaine de la modélisation et de la vérification formelle de la composition de services Web, nous constatons qu'en dehors des approches basées sur la méthode B Événementiel, le reste des travaux se base essentiellement sur la technique de vérification sur modèles. Bien que cette technique soit automatique, le problème de l'explosion du nombre d'états explorés, et que le modèle obtenu ne sont vérifiés qu'une fois construit.

Dans notre approche nous allons opter pour la méthode de vérification pas-à-pas, le modèle est une Communication d'Automates à Etats Finis Complexes,et la construction et la vérification se feront à chaque étape.

2.7 Conclusion

Dans ce chapitre une vue globale de la composition des services web a été présentée,ses classifications, et les langages de description,ensuite nous a avons mentionné les recherches pertinentes faites à ce propos, et enfin une synthèse de ces solutions a été présentée afin d'en tirer profit pour notre proposition. Dans le chapitre qui suit nous présentons la méthode formelle EventB, d'une manière assez explicite.

3

LA METHODE FORMELLE : EventB

3.1 Introduction

Ce chapitre englobe une synthèse des aspects fondamentaux de la méthode ***B événementiel*** -Event B-, d'une façon assez explicite : contexte, machine abstraite, événement, obligations de preuves, ainsi que la platform utilisée.

3.2 L'utilisation des méthodes formelles

Les méthodes formelles sont un terme générique qui représente l'ensemble des techniques disponibles permettant la modélisation de tout ou partie d'un système. Cela est en particulier applicable aux applications logicielles. Ces techniques reposent sur des fondements mathématiques précis qui permettent de raisonner sur des propriétés et de construire des preuves. Ces preuves montrent et démontrent que les propriétés exprimées sur un programme sont bien respectées [33], de nouveaux outils et de nouvelles méthodologies de développement ont été prises et regroupés sous la dénomination Méthodes Formelles vue la complexité des systèmes informatiques et le besoin en qualité de ces systèmes.

3.3 Notion de B événementiel

Le B événementiel, introduit par J.-R. Abrial [45] [46] est une méthode de développement formel ainsi qu'un langage de spécification [34], l'unité de spécification, dans cette méthode, est un module appelé *machine abstraite*, constitué *d'invariants* et des méthodes spécifiées[47] à l'aide d'évènements, dont on prouve la cohérence. On construit ensuite des raffinements,on finit par en avoir un modèle concret qui est proche de la réalité.

3.4 La méthode formelle EventB

Les méthodes formelles sont des techniques qui peuvent être utilisées lors de chacune des phases du cycle de développement logiciel pour aider à structurer le raisonnement et apporter des garanties sur le développement. Pour cela, ces méthodes reposent sur un raisonnement mathématique rigoureux fondé sur un langage formel. Parmi les méthodes formelles, on peut citer la méthode Event-B. C'est une évolution de la méthode B classique [45].Ces deux méthodes partagent d'ailleurs les mêmes principes fondamentaux ; la logique du premier ordre et la théorie des ensembles.

3.5 Logique et théorie des ensembles

Le langage logico-ensembliste d'Event-B est base sur la logique classique du premier ordre et la theorie des ensembles.Les symboles utilisés pour exprimer des predicats logiques sont [35] :

- $\top$ vrai
- $\bot$ faux
- P $\bigwedge$ Q conjonction
- P $\bigvee$ Q disjonction
- $\neg$ negation
- P $\Rightarrow$ Q implication
- P $\Leftrightarrow$ Q equivalence
- $\forall$ x_1, x_2,.., x_n . P(x_1, x_2,.., x_n) quantification universelle
- $\exists$ x_1, x_2,.., x_n . P(x_1, x_2,.., x_n) quantification existentielle

Le langage logico-ensembliste d'Event-B supporte les notations ensemblistes usuelles comme :

- l'inclusion ($\subseteq$)
- l'inclusion stricte ($\subset$)
- l'union ($\bigcup$)
- l'intersection ($\bigcap$)

- la difference d'ensemble ($\setminus$)
- l'ensemble vide ($\emptyset$ ou {})
- l'ensemble des parties non vides ($\mathbb{P}1$)
- l'ensemble des parties ($\mathbb{P}$)
- les ensembles enumeres x_1, x_2,..,x_n

En outre, le langage logico-ensembliste d'Event-B supporte les concepts mathematiques couple, relation et fonction. Les concepts usuels sur les relations (s$\rightarrow$t) sont definis en Event- B tels que :

- domaine (dom)
- codomaine (ran)
- relation d'identite (id)
- relation reciproque (r^{-1})
- image d'un ensemble par une relation (r[s])
- et la composition (r ;q)

De plus, le langage logico-ensembliste d'Event-B propose des opérateurs pour restreindre les relations :

- la restriction sur le domaine (S$\lhd$r)
- la corestriction qui est une restriction sur le codomaine (r$\rhd$T)).

Les fonctions sont considerees comme des relations particulieres.La méthode event-B permet donc également de manipuler les fonctions. Pour ce faire, nous avons la possibilité de définir une fonction f comme étant [38] :

- une fonction partielle d'un ensemble E1 vers un ensemble E2 : "f : E1 $\rightarrowtail$ E2" ;
- une fonction totale : "f : E1 $\rightarrow$ E2" ;
- une injection partielle : "f : E1 $\rightarrowtail$ E2" ;
- une injection totale : "f : E1 $\mapsto$ E2" ;
- une surjection partielle : "f : E1 $\twoheadrightarrow$ E2" ;
- une surjection totale : "f : E1 $\twoheadrightarrow$E2" ;
- une bijection partielle : "f : E1 $\mapsto$ E2"

Les couples en Event-B sont notés sous la forme x$\mapsto$y. Les constructions offertes par le langage logico-ensembliste permettent de typer les ensembles, les constantes et les variables decrivant la partie statique d'un modele Event-B. En outre, elles sont utilisées pour decrire les propriétés invariantes d'un modele Event-B sous forme des predicats logiques. Enfin, elles decrivent les axiomes et theoremes des modèles Event-B.

3.6 Le modèle Event B

Un modèle est une représentation mathématique d'un système, utilisée pour vérifier ou assurer les propriétés du système considéré. Un système peut être un ensemble d'applications ou d'éléments dialoguant entre eux. Le modèle est le premier concept d'Event-B. Il est composé d'un ensemble de *machines* et de *contextes*. Une machine Event-B contient deux parties : *statique* et *dynamique*. La partie statique comporte les variables modélisant l'état du système. Ces variables sont typées et peuvent avoir des propriétés invariantes décrites par des prédicats logiques. La partie dynamique comporte des évènements permettant d'agir sur l'état du système. Le nouvel état obtenu suite au déclenchement de l'événement doit préserver l'invariant. Un événement particulier appelé *Initialisation* doit établir l'invariant. De plus, une machine Event-B peut comporter des théorèmes qui devraient êtres prouvés. Un contexte Event-B comporte les paramètres du système à modéliser : ensembles et constantes. Les propriétés de ces paramètres sont formalisées par des axiomes et des théorèmes à prouver. Il est à noter qu'un modèle peut contenir uniquement des contextes (c'est un modèle qui représente une structure purement mathématique avec les constantes, les axiomes et les théorèmes), ou bien uniquement des machines (non paramétré) ou bien les deux ensembles (modèle paramétré par les contextes) [35]. Pour ce troisième type de modèle,il existe un ensemble de relations entre les machines et les contextes, comme représenté dans la figure 3.1[35].

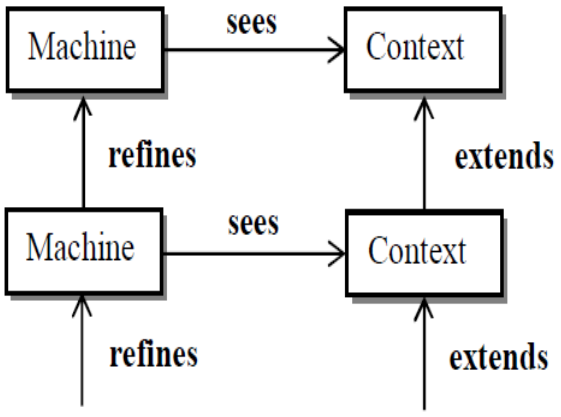


FIGURE 3.1 – Les relations entre les contextes et les machines

Les machines et les contextes ont plusieurs relations : une machine peut être raffinée par une autre machine, un contexte peut être étendu par un autre contexte, et une machine peut importer un ou plusieurs contextes. La figure 3.2 [27] présente les deux composants avec leurs différentes clauses telles qu'elles sont déclarées dans la plateforme Rodin [43],[44].

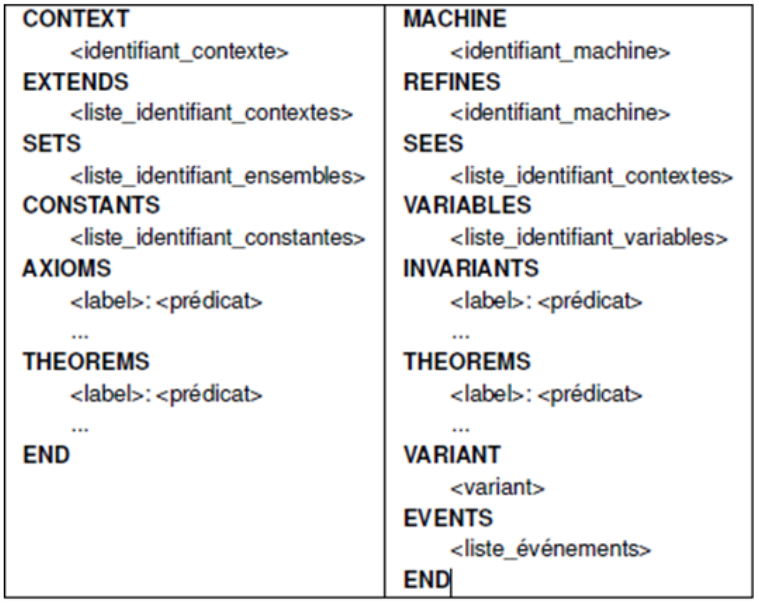


FIGURE 3.2 – La structure d'un modèle B-Event

3.6.1 Le composant CONTEXT

Un contexte "*Context*" permet de spécifier des données statiques. Il se compose d'ensembles, de constantes avec leurs axiomes et éventuellement des théorèmes. La structure du contexte est constituée d'un ensemble de clauses introduites par des mots clés [35] :

- **CONTEXT** : permet de définir le nom du contexte. Il doit être distinct de tous les autres noms qui existent dans le modèle,
- **EXTENDS** : contient la liste des contextes hérités,
- **SETS** : regroupe la liste des ensembles qui ne sont pas vides et deux à deux disjoints s'ils sont de même type.
- **CONSTANTS** : définit la liste des constantes introduites dans le contexte.
- **AXIOMS** : définit la liste des prédicats que les constantes doivent respecter.
- **THEOREMS** : exprimant un ensemble de propriétés qui peuvent être déduites à partir des axiomes

3.6.2 Le composant MACHINE

Le composant *Machine* formalise la partie dynamique du système modélisé. Généralement, la MACHINE contient la définition de l'état du système, des propriétés du système et des événements qui font évoluer l'état du système. La structure d'une machine ressemble à la structure du contexte. La MACHINE est composée par les clauses suivantes [35] :

- **MACHINE** : donne un identifiant à cette machine. Un tel identifiant doit être différent des identifiants de tous les autres composants du même modèle.
- **REFINES** : représente l'identificateur de la machine (si elle existe) à partir de laquelle laa machine actuelle est raffinée. Une machine raffine au plus une autre machine.
- **SEES** : déclarant la liste des contextes importés par la machine décrite,
- **INVARIANTS** : exprimant les propriétés invariantes du modèle à l'aide des prédicats en logique du premier ordre. Des informations sur les types des variables et des propriétés de sûreté sont décrites dans cette clause. Ces propriétés doivent être vérifiées dans la machine et ses raffinements. Les invariants doivent être aussi préservés par les événements de la clause EVENTS,
- **THEOREMS** : exprimant un ensemble de propriétés qui peuvent être déduites à partir des invariants.
- **VARIANT** : définissant l'expression du variant du modèle,
- **EVENTS** : déclarant les événements qui peuvent prendre le contrôle dans le système modélisé. Chaque événement est décrit par une garde et par un corps. Un événement prend le contrôle lorsque sa garde est évaluée à vrai. Une machine possède un événement particulier qui correspond à l'initialisation du système modélisé.

La figure 3.3 [27],présente la structure d'un évènement ;

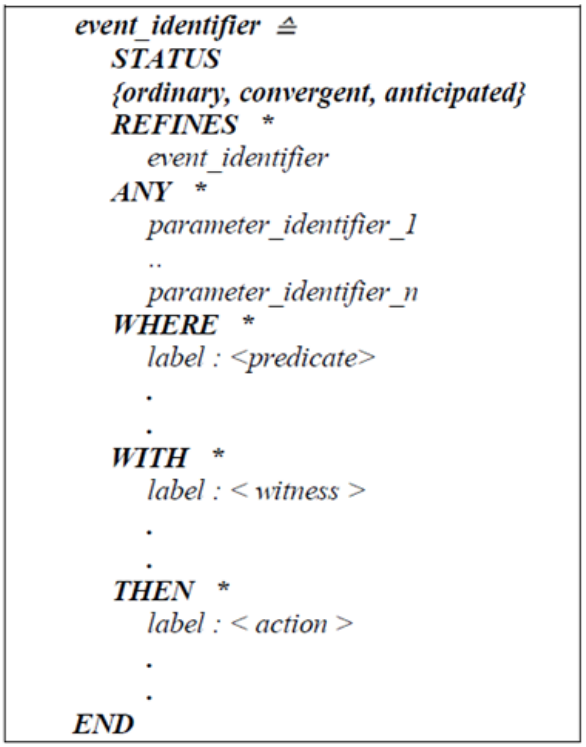


FIGURE 3.3 – La structure d'un événement

3.6.3 Les événements

Un événement correspond à un changement d'état dénotant une transition dans le système modélisé. Il est composé des clauses suivantes [27] [35] (cf. figure 3.3) :

- **STATUS** : décrit l'état d'un événement. Un événement peut être :
  - *ordinary*,
  - *convergent* : il doit décrémenter le variant,
  - *anticipated* : sera convergent ultérieurement dans un raffinement.
- **REFINES** : liste le(s) événement(s) abstraits que l'événement actuel raffine (s'il existe).
- **ANY** : énumère la liste des paramètres de l'événement.
- **WHERE** : contient les différents gardes de l'événement. Ces gardes sont des conditions nécessaires pour déclencher l'événement. Il faut noter que si la clause " *any* " est omise le mot clé " *where* " est remplacé par " *when* ".
- **WITH** : lorsqu'un paramètre dans un événement abstrait disparaît dans la version concrète de cet événement, il est indispensable de définir un témoin sur l'existence de ce paramètre : c'est ce qu'on appelle " *witness* : variables et/ou paramètres de l'événement de l'abstraction qui disparaissent dans le raffinement ".

– **THEN** : déclarant les actions de l'événement.

3.7 Platforme RODIN(*Rigorous Open Development Environment for Complex Systems*)

La plateforme RODIN est un environnement dédié à Event-B. Hormis la structure d'accueil offerte par RODIN, ce dernier intègre les composants logiciels suivants [35] :

– Un éditeur dirigé par la syntaxe pour saisir les modèles Event-B.

– Un analyseur lexico-syntaxique d'Event-B.

– Un vérificateur de typage d'Event-B.

– Un générateur d'obligations de preuve selon les règles qui seront exposées dans la section qui suit.

– Un prouveur automatique et interactif assez puissant.

Les deux figures 3.4 et 3.5 présentent une vue macroscopique de la plateforme RODIN et la table 3.1 fournit la signification des différents éléments de cette vue macroscopique.

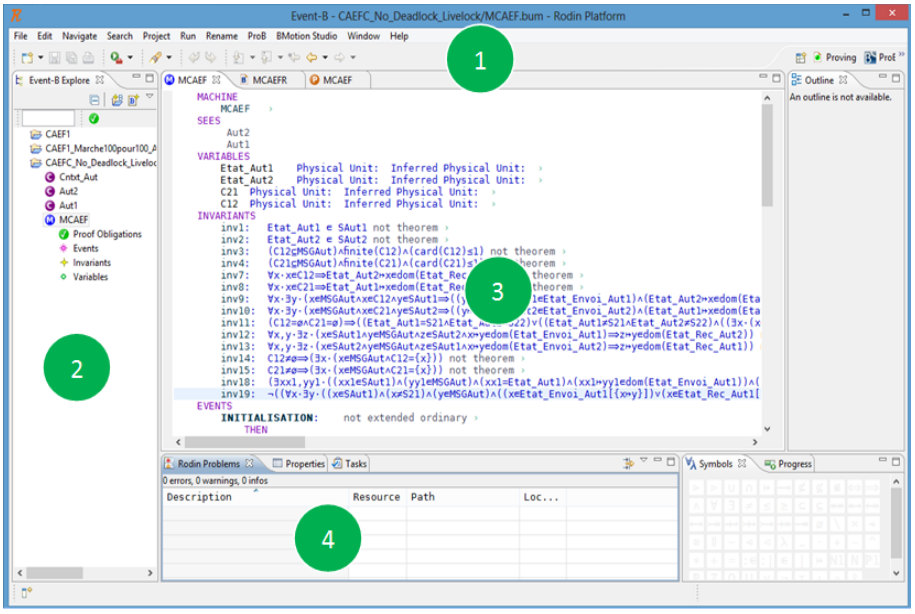


FIGURE 3.4 – Vue macroscopique 1 de la platforme RODIN

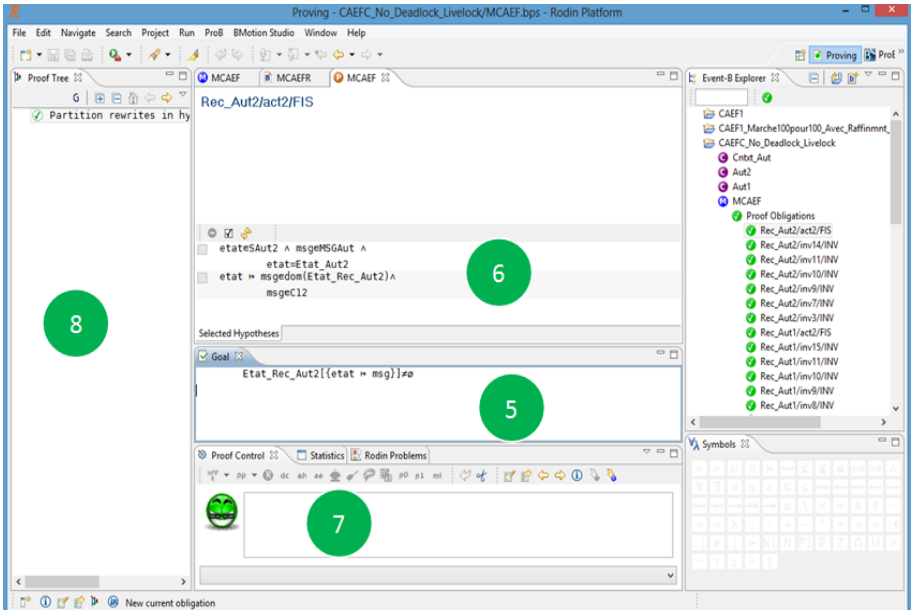


FIGURE 3.5 – Vue macroscopique 2 de la platforme RODIN

1	Barre d'outils
2	Explorateur de projets
3	Zone de saisie
4	Problèmes rencontrés
5	Hypothèses sélectionnées
6	But à atteindre
7	Contrôle de preuves
8	Arbre de preuve

TABLE 3.1 – Structure d'accueil de la platforme RODIN

3.8 Obligations de preuves

Afin de garantir la correction de notre modèle, il est indispensable de le prouver. Pour y parvenir, un outil de la plateforme RODIN appelé *générateur d'obligations* de preuve génère automatiquement des obligations de preuve [37]. Une obligation de preuve définit ce qui doit être prouvé pour un modèle. Il s'agit d'un prédicat dont on doit fournir une démonstration pour vérifier un critère de correction sur le modèle[35].

L'outil cité ci-dessus vérifie statiquement les contextes et les machines et génère des ***séquents***. " Un *séquent* est un nom générique pour quelque chose qu'on veut prouver " [35]. Il est de la forme H $\vdash$ G : le but G est à démontrer en partant de l'ensemble H des hypothèses.

3.8.1 Les théorèmes

Rappelons que les théorèmes dans les modèles Event-B sont des formules qui peuvent être utiles pour réécrire l'invariant sous une forme spécifique ou démontrer des lemmes. L'obligation de preuve consiste à prouver que ces formules sont déduites à partir de l'invariant de la machine ainsi que l'ensemble des prédicats dans les clauses AXIOMS et THEOREMS du contexte.

3.8.2 Le non-blocage

Le model-checking est un outil complémentaire à la preuve, indispensable pour la validation des modèles Event-B. il permet de déceler les éventuelles fautes de comportement dans le modèle, i.e : l'existence d'interblocages, mais ne permet pas de prouver l'absence d'interblocages, bien que des obligations de preuve pour cette dernière soient prévues dans Event-B, elles ne sont pas encore implémentées dans RODIN. La raison est que cette obligation de preuve prend la forme d'une grande disjonction (la disjonction des gardes de tous les événements), qui est souvent très difficile à prouver[36].

Quand les événements sont gardés, les évènements peuvent ne plus se produire dans certains états, ce qui peut bloquer le système. Il faut vérifier donc que, dans tous les états, il existe au

moins un évènement qui peut se déclencher (sa garde est vraie). Cette vérification est faite par un théorème montrant la disjonction des gardes des évènements. Cette obligation de preuve est décrite comme suit :

$$\mathrm{I(x)} \Longrightarrow (G_1(\mathrm{x}) \vee ... \vee\ G_m(\mathrm{x}))$$

L'invariant I(x) est une propriété que le système doit préserver quelle que soit son évolution.

3.8.3 Le principe de la contre-preuve

La plate-forme RODIN contient un générateur d'obligations de preuve ainsi que plusieurs " *prouveurs* " automatiques qui peuvent décharger automatiquement des obligations de preuve. Evidemment, toutes les preuves ne peuvent pas être déchargées automatiquement. Dans ce cas, l'utilisateur se pose la question suivante [36] :

- est-ce que les obligations non prouvées sont valides et les preuves sont simplement trop compliquées pour le prouveur automatisé ? En d'autres termes, l'utilisateur se demande s'il doit lancer le prouveur interactif et essayer de prouver l'objectif de la preuve manuellement,
- ou est-ce qu'il y a un problème au sein de la spécification et s'il doit chercher l'erreur dans le modèle et ensuite corriger la spécification. Les cas précédents peuvent mener à un gaspillage considérable d'effort. Un " *contre-prouveur* " (*disprover*) peut aider l'utilisateur dans cette situation. Si une obligation de preuve n'est pas déchargée automatiquement, l'utilisateur peut appliquer le contre-prouver, qui va essayer de trouver un contre-exemple pour l'obligation de preuve problématique.

Si le contre-prouveur trouve un contre-exemple, nous savons qu'il est inutile d'utiliser le prouveur interactif. Egalement, le contre-exemple donne un indice concernant le problème et peut nous aider à trouver plus rapidement l'erreur dans la spécification [36].

3.9 Le Raffinement

"... le programme concret implémente (ou raffine) le programme abstrait correctement lorsque toute utilisation du programme concret ne conduit pas à une observation qui n'est pas aussi une observation du programme abstrait" -*Paul Gardiner* et *Carroll Morgan*

La notion de *raffinement* a été introduite dans les années 1970 par Dijkstra [48], puis formalisée par Back[49] [51] dans les années 1980. Le raffinement est le processus de construction d'un modèle progressivement en le rendant de plus en plus précis. Ce qui aboutit à un modèle très proche de la réalité [35]. Il consiste à reformuler, par étapes successives, une machine abstraite en une suite de

machines plus précises (modèle concret) où plus de détails apparaissent, dans l'état du système en ajoutant des variables, et dans le comportement en détaillant les événements de l'abstraction ou en ajoutant de nouveaux événements [27].

Afin de mieux comprendre le raffinement, nous proposons de traiter l'exemple suivant :

Soit l'événement abstrait *check_out* d'une application de gestion de réservations dans un hôtel, un individu p désire annuler sa réservation après avoir enregistré sa demande dans l'annuaire de l'hôtel $(r, p) \in owns$.(Cf Fig 3.6 [35])

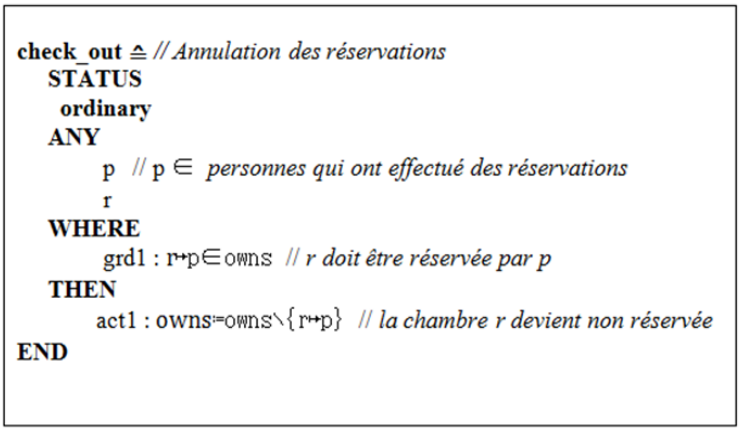


FIGURE 3.6 – Exemple de programme à raffiner

Son raffinement appelé *check_out1* (1 pour dire premier raffinement) obéit au protocole cité ci-dessus. En fait, il est augmenté par un seul paramètre c, de plus ses gardes et ses actions sont renforcés à l'augmentation.

L'interprétation réelle de cet évènement " annulation de réservation d'un client " réside dans l'action que le client retire sa carte.

Les éléments ajoutés sont colorés en bleu.(Cf Fig 3.7[35])

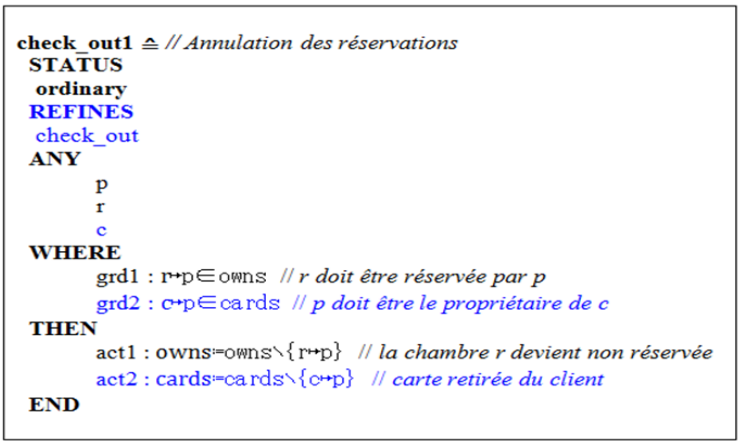


FIGURE 3.7 – Exemple de programme raffiné

Dans une machine raffinée, le générateur d'obligations de preuves génère automatiquement des obligations de preuves relatives au raffinement.

3.10 Conclusion

Dans ce chapitre, nous avons présenté d'une façon détaillée les concepts de base de la méthode formelle *Event-B*. Cette méthode est très recommandée pour le développement pas-à-pas de logiciels corrects par construction en se servant de la technique de *raffinement*.Dans la suite de ce travail, nous allons utiliser Event-B pour modéliser formellement la spécification d'une communication d'Automates Etats Finis Complexes(CCFSM).

4

Proposition : Validation d'un modèle d'une CCFSM par la mèthode B-Événementiel

4.1 Introduction

Dans ce chapitre, nous comptons établir une spécification formelle en Event-B d'une Communication d'Automates d'Etats Finis Complexes (**CCFSM**). Pour y parvenir, nous allons appliquer deux phases : *spécification abstraite* et *Raffinement* en garantissant que l'implémentation soit **correcte par construction**.Une activité de validation du modèle Event-B s'impose en se servant des outils d'animation associés à Event-B tels que ProB.

4.2 Problématique

Les exigences d'un logiciel, souvent rédigées en langage naturel, sont à la base des phases de conception et de test fonctionnel. Le langage naturel est par nature ambigu, et les exigences peuvent

donc être différemment interprétées lors de la construction et de la validation du logiciel. d'autres approches tentent en vain à développer des logiciels corrects. Tandis que, l'approche formelle permet l'obtention des logiciels corrects par construction. En effet, chaque modèle produit par cette approche est soumis à des preuves. Si une preuve échoue, alors le modèle est revu et corrigé. Puis, on crée un modèle plus raffiné, auquel on applique des preuves. Et ainsi de suite jusqu'à l'obtention d'un modèle final sans impasses. En outre, contrairement à aux autres approches, la phase des tests est très simplifiée dans l'approche formelle. Dans le domaine des Services Web, les protocoles de communication peuvent être modélisés sous forme de d'Automates à Etats Finis. Dans ce modèle, le comportement de chaque entité communicante est spécifié par un Automate à Etats Finis (AEF). Ces entités échangent des messages via des canaux de communication de capacité limitée. La validation de protocoles, consiste à vérifier l'absence des erreurs logiques de spécification telles que l'inter-blocage, les réceptions non spécifiées, et l'explosion combinatoire.

4.3 Modèle de Communication d'Automates à Etats Finis Complexes (CAEFC)

Un modèle de CAEFC introduit la notion d'état complexe, qui est un AEF dont certains de ses états sont eux-mêmes des AEF (internes). Les états qui rentrent dans la composition d'un état complexe sont appelés états internes.

Soit I = {1, 2, ..., n} un ensemble fini d'indices, avec cardinal I $\geq$2. Dans un modèle de Communication d'automate d'états finis :

- P=(S, s0, F, M, R) est l'ensemble des n entités communicantes du protocole, et Chaque entité P_i est un automate d'états finis(simple ou complexe), est composé de :
  - un ensemble d'états S,
  - un ensemble d'états finaux F,
  - un alphabet de messages M ,
  - un ensemble de transitions R,
  - un état initial s0.
- C = C_{ij}/(i, j)$\in$ I est un ensemble non vide de canaux de communication.
- Chaque canal C_{ij}, reliant l'entité P_i à l'entité P_j.
- L'ensemble des états S comprend des états simples et des états complexes.
- Chaque état complexe doit être défini par un AEF. Le contenu du canal C_{ij} , noté c_{ij} , est une suite de messages de M, i.e $c_{ij} \in$ M .
- Chaque canal peut contenir un nombre maximum de K (entier positif, dans notre cas K=1)

messages.

- Une entité P_i envoie des messages sur son canal de sortie C_{ij} (i, j ∈I & I≠ j) et reçoit des messages sur ses canaux d'entrées C_{ji} (i, j ∈I & i≠j).
- La relation de transition R est définie par : R : S × M'⟶ S, une transition T= (S,m,S) est dite transition d'envoi de message lorsque m = $-x$, indiquant un envoi d'un message x par l'entité P_i sur le canal C_{ij}. T est dite transition de réception de message lorsque m = $+y$, indiquant une réception d'un message y par l'entité P_i depuis le canal C_{ij} .
- Les signes − et + indiquent respectivement un envoi et une réception de message. Dans le cas où s_i est un état complexe (représente un AEF interne), le résultat de la transition va mettre l'automate interne sur son état initial, la transition ne peut être exécutée que si l'automate lui correspondant atteint son état final.

4.3.1 Représentation de notre modèle

La figure 4.1 schématise notre modèle d'une CAEF simple, le modèle complexe sera exposé un peu plus loin dans le chapitre.

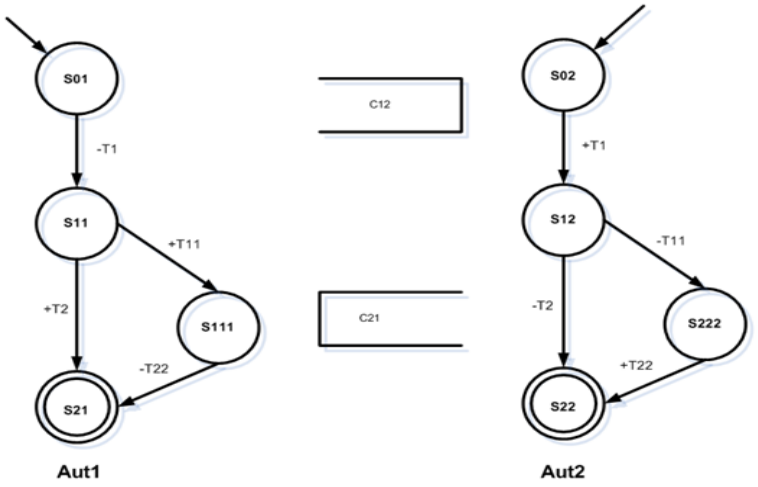


FIGURE 4.1 – Représentation d'une Communication d'Automates à Etats Finis

4.3.2 Spécification en Event-B

L'architecture générale de la modélisation en Event-B de la Communication d'Automates à Etats Finis comporte une *machine abstraite* (MCAEF), une machine *raffinée* (MCAEF0) et trois *contextes* (Cntxt_Aut, Aut1 et Aut2). Trois types de relations sont utilisés dans cette architecture : la relation ***refines*** liant soit une machine abstraite à une machine raffinée, soit une machine raffinée

de niveau i à une machine raffinée de niveau i+1, la relation ***extends*** entre deux contextes et la relation ***sees*** entre soit une machine abstraite et un contexte, soit entre une machine raffinée et un contexte. Sachant que les deux relations ***refines*** et ***extends*** sont transitives.

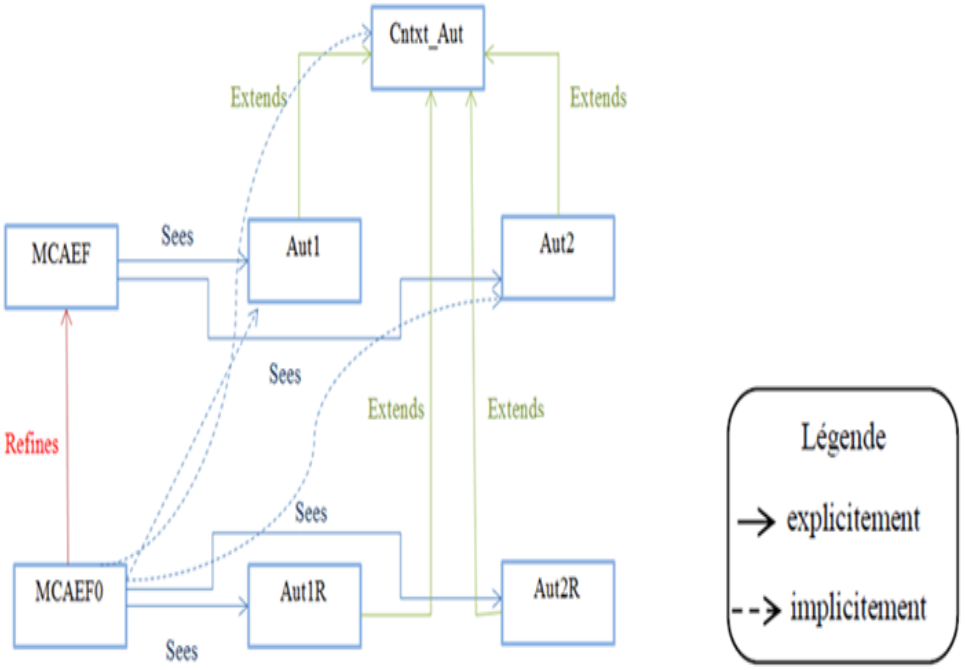


FIGURE 4.2 – Architecture générale du modèle proposé en Event-B d'une CAEFC

Afin de modéliser pas-à-pas la Communication d'Automates à Etats Finis Complexes (CAEFC), nous avons utilisé la notion de *raffinement*, ainsi nous avons suivi une démarche orientée état afin d'établir les constituants d'une machine, Une telle démarche comporte les étapes suivantes :

- Définir les *paramètres* (constantes, axiomes et théorèmes) en les regroupant au sein d'un contexte pouvant étendre un contexte existant.
- Définir l'*état* d'une machine par une ou plusieurs variables typées. Lors de cette étape, on décrit les contraintes sous forme de prédicats qu'elles doivent satisfaire ou respecter en permanence, qu'on appelle les *Invariants*. (*La propriété d'invariant doit être vérifiée à tout moment du traitement, c'est-à-dire après l'initialisation des variables, ainsi qu'avant et après chaque opération*).
- Définir les *transitions* autorisées permettant de changer l'état d'une machine. Ceci est exprimé par des événements en Event-B.

$Axiom_i$	**Spécification**	**Description**
Axiom1	partition(SAut1, {S01},{S11},{S111}, {S21})	SAut1 = {S01,S11,S111,S21}
Axiom7	Etat_Envoi_Aut1∈SAut1×MSGAut↔SAut	Domaines de définition de l'ensemble Etat_envoi_Aut1.
Axiom8	Etat_Envoi_Aut1={S01↦T1↦S11, S111↦T22↦S21}	L'ensemble de transitions (envoi) possible pour l'automat1.
Axiom9	Etat_Rec_Aut1∈SAut1× MSGAut↔SAut1	Domaines de définition de l'ensemble Etat_Rec_Aut1.
Axiom10	Etat_Rec_Aut1={S11↦T11↦S111, S11↦T2↦S21 }	L'ensemble de transitions (réception) possible.

TABLE 4.1 – Ensemble d'axioms et leur description

4.3.1.1. Modèle initial abstrait en Event-B

a) Paramètres

Les paramètres du système sont regroupés au sein d'un contexte. Celui-ci comporte des ensembles abstraits (*Sets* en Event-B), des constantes (*CONSTANTS* en Event-B) et des axiomes (*AXIOMS* en Event-B) pour notre modèle, trois contextes sont implémentés *Aut1, Aut2 et Cntxt_Aut.*

- **Les Ensembles** :
  - **SAut1** modélisant l'ensemble des états de l'automate1.
  - **SAut2** modélisant l'ensemble des états de l'automate2.
  - **MSGAut** modélisant l'ensemble des messages intervenant lors de la communication.
- **Les constantes :**
  - S01, S11, S111, S21 : constantes formant l'ensemble *SAut1.*
  - S02, S12, S222, S22 : constantes formant l'ensemble *SAut2.*
  - T1, T2, T11, T22 messages formant *MSGAut.*
- **Les Axiomes** :

  Par conséquent, notre système est initialement paramétré sur les ensembles abstraits déclarés comme suit :

```
CONTEXT
   Aut1
EXTENDS
   Cntxt_Aut
SETS
   SAut1
CONSTANTS
   S01
   S11
   S21
   S111
   Etat_Envoi_Aut1
   Etat_Rec_Aut1
AXIOMS
   axm1   :   partition(SAut1, {S01}, {S11},{S111}, {S21})
   axm7   :   Etat_Envoi_Aut1∈SAut1×MSGAut⟶SAut1
   axm8   :   Etat_Envoi_Aut1={S01↦T1↦S11, S111↦T22↦S21}
   axm9   :   Etat_Rec_Aut1∈SAut1×MSGAut⟶SAut1
   axm10   :   Etat_Rec_Aut1={S11↦T11↦S111, S11↦T2↦S21}
END
```

FIGURE 4.3 – *Context Aut1* au sein du système

```
CONTEXT
   Aut2   ›
EXTENDS
    Cntxt_Aut
SETS
   SAut2   ›
CONSTANTS
   S02  --undefined-- Physical Unit:  Inferred Physical Unit:  ›
   S12  --undefined-- Physical Unit:  Inferred Physical Unit:  ›
   S22  --undefined-- Physical Unit:  Inferred Physical Unit:  ›
   S222    --undefined-- Physical Unit:  Inferred Physical Unit:  ›
   Etat_Envoi_Aut2  not symbolic Physical Unit:  Inferred Physical Unit:  ›
   Etat_Rec_Aut2    not symbolic Physical Unit:  Inferred Physical Unit:  ›
AXIOMS
   axm1:  partition(SAut2, {S02}, {S12}, {S22}, {S222}) not theorem ›
   axm7:  Etat_Envoi_Aut2∈SAut2×MSGAut⟶SAut2 not theorem ›
   axm8:  Etat_Envoi_Aut2={S12↦T2↦S22, S12↦T11↦S222} not theorem ›
   axm9:  Etat_Rec_Aut2∈SAut2×MSGAut⟶SAut2 not theorem ›
   axm10:  Etat_Rec_Aut2={S02↦T1↦S12,S222↦T22↦S22} not theorem ›
END
```

FIGURE 4.4 – *Context Aut2* au sein du système

```
CONTEXT
   Cntxt_Aut
SETS
   MSGAut
   MSGAutR
CONSTANTS
   T1
   T2
   T11
   T22
   T111
   T222
AXIOMS
   axm1   :   partition(MSGAut, {T1}, {T2},{T11}, {T22})
   axm2   :   partition(MSGAutR, {T111}, {T222})
END
```

FIGURE 4.5 – *Context_Aut* au sein du système

b) Etat du système

L'état du système est décrit par quatre *variables* appelées ***Etat_Aut1, Etat_Aut2, C12, C21*** modélisant respectivement l'état global de l'automate 1 au court de la communication, idem pour l'automate 2, état du canal C12 qui va de l'automate 1 vers l'automate 2 et état du canal C21 qui va de l'automate 2 vers l'automate 1, et typées par des *invariants*. Ces derniers indiquent :

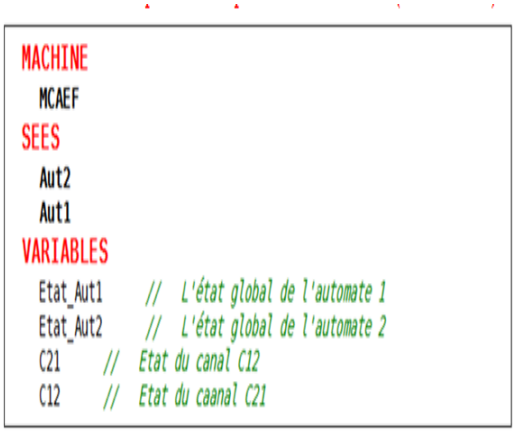


FIGURE 4.6 – Les variables qui décrivent l'état du système

Ainsi nous pouvons,définir l'*état initial* de notre système. Un tel état est représenté dans l'événement **INITIALISATION** par des actions notées act1, act2, act3 et act4 modélisant les états des entités communicantes qui sont toutes à l'état initial et ayant les canaux vides.

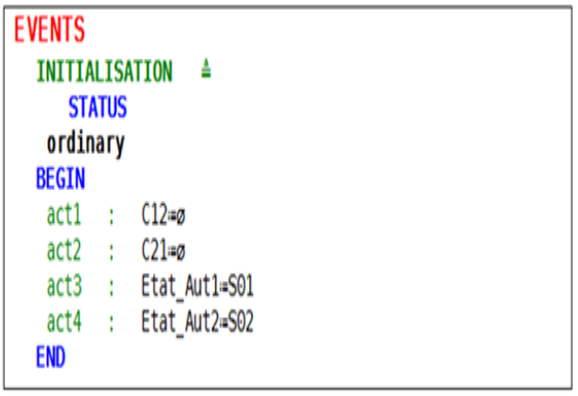


FIGURE 4.7 – Etat initial du système

Les obligations de preuve relatives à cet événement ont été déchargées automatiquement par le prouveur de la plateforme **RODIN**. Elles sont de la forme *INITIALISATION/inv$_i$/INV* et

Invariant	**Description**
Etat_Aut1 ∈SAut1	L'état global du système appartient à l'ensemble fini des états de l'automate 1, tel que SAut1= {S01, S12, S111,S21}.
(C12⊆MSGAut)∧finite(C12)∧(card(C12))≤1	Au plus un message dans le canal de communication
∀x·x∈C12⇒Etat_Aut2↦x∈dom(Etat_Rec_Aut2)	L'existence d'un message dans le canal C12 indique l'évènement de réception de ce message par l'automate 2
∀x·∃y·(x∈MSGAut∧x∈C12∧y∈SAut1⇒ ((y↦x↦Etat_Aut1∈Etat_Envoi_Aut1) ∧(Etat_Aut2↦x∈dom(Etat_Rec_Aut2))))	L'envoie d'un message par l'automate 1 provoque une réception du même message par l'automate 2
(C12=∅∧C21=∅)⇒ ((Etat_Aut1=S21∧Etat_Aut2=S22)∨ ((Etat_Aut1≠S21∧Etat_Aut2≠S22) ∧ ((∃x·(x∈MSGAut∧ Etat_Aut1↦x∈dom(Etat_Envoi_Aut1))) ∨ (∃x·(x∈MSGAut∧Etat_Aut2 ↦ x∈dom(Etat_Envoi_Aut2))))))	Les canaux vides impliquent la terminaison de la communication entre les entités i.e : Les états globaux des automates sont aux états finaux respectivement Etat_Aut1 et Etat_Aut2 sont à S21 et S22 OU les entités sont à l'envoie de message i.e : les états globaux diffèrent les états finaux.
∀x,y·∃z·(x∈SAut1∧y∈MSGAut ∧z∈SAut2∧x↦y∈dom(Etat_Envoi_Aut1)⇒ z↦y∈dom(Etat_Rec_Aut2))	L'envoie du message par l'automate 1 implique obligatoirement une réception du même message par l'automate 2.
C12≠∅⇒(∃x·(x∈MSGAut∧C12={x}))	Les messages qui appartiennent au canal C12 proviennent de l'automate 1
(∃xx1,yy1·((xx1∈SAut1)∧ (yy1∈MSGAut)∧(xx1=Etat_Aut1)∧ (xx1↦yy1∈dom(Etat_Envoi_Aut1))∧ (card(C12)<1)))∨ (∃xx2,yy2·((xx2∈SAut2) ∧(yy2∈MSGAut)∧(xx2=Etat_Aut2) ∧(xx2↦yy2∈dom(Etat_Envoi_Aut2)) ∧(card(C21)<1)))∨ (∃xx3,yy3·((xx3∈SAut1) ∧(yy3∈MSGAut)∧ (xx3=Etat_Aut1)∧ (xx3↦yy3∈dom(Etat_Rec_Aut1))∧ (yy3∈C21)))∨(∃xx4,yy4·((xx4∈SAut2)∧ (yy4∈MSGAut)∧ (xx4=Etat_Aut2)∧ (xx4↦yy4∈dom(Etat_Rec_Aut2))∧ (yy4∈C12)))∨((Etat_Aut1=S21)∧ (Etat_Aut2=S22)∧ (C12=∅)∧ (C21=∅))	Théorème montrant la disjonction des gardes des évènements, ces derniers peuvent ne plus se produire dans certains états, ce qui peut bloquer le système. Il faut vérifier donc que, dans tous les états, il existe au moins un évènement qui peut se déclencher, en d'autre terme le théorème est toujours vrai.

TABLE 4.2 – Description de la spécification formelle (les invariants)

signifient que l'événement d'initialisation préserve l'invariant inv_i.

d) Les transitions

Ce système comporte quatre transitions dites encore *événements* (autres que l'événement ***INITIALISATION***) appelés *Envoi_Aut1, Envoi_Aut2, Rec_Aut1 et Rec_Aut2* et modélisant respectivement les actions d'envoie et de réception d'un message,Envoi_Aut1, Rec_Aut1 de l'automate 1 ont respectivement, le même comportement que Envoi_Aut2, Rec_Aut2 de l'automate 2.

L'événement Envoi_Aut1 : Cet événement nécessite, à ce niveau, deux paramètres : " etat " pour indiquer l'état de l'automate1 et un message " msg " pour se communiquer. Ces derniers doivent respecter deux gardes notées grd1 et grd2, la figure 4.7 illustre la spécification en Event-B de cet événement.

```
Envoi_Aut1  ≙
   STATUS
 ordinary
ANY
 etat
 msg
WHERE
 grd1   :   (etat∈SAut1)∧(msg∈MSGAut)∧(etat=Etat_Aut1)
 grd2   :   (etat↦msg∈dom(Etat_Envoi_Aut1))∧(card(C12)<1)
THEN
 act1   :   C12≔{msg}
 act2   :   Etat_Aut1:∈Etat_Envoi_Aut1[{etat↦msg}]
END
```

FIGURE 4.8 – Evènement d'envoi d'un message

- Pour pouvoir envoyer un message, l'état de l'automate 1 doit être à S01 ou S111,et le couple etat↦msg appartient au domaine Etat_Envoi_Aut1={S01↦T1↦S11, S11↦T22↦S21},tout en vérifiant que le canal soit libre (card(C12)<1).
- Un tel événement va agir sur la variable *C12* pour la modifier tel que le canal va contenir le message envoyé.

L'événement Rec_Aut1 : Il concerne la réception de messages. Pour y parvenir, il nécessite deux paramètres : " *etat* " indiquant l'état de l'automate 1 et un message " *msg* " intervenant lors de la communication. De plus, les contraintes sur ces paramètres sont exprimées via deux gardes notées ***grd1*** et ***grd2***.

```
Rec_Aut2  ≙
   STATUS
 ordinary
ANY
 etat
 msg
WHERE
 grd1   :   (etat∈SAut2)∧(msg∈MSGAut)∧(etat=Etat_Aut2)
 grd2   :   (etat↦msg∈dom(Etat_Rec_Aut2))∧(msg∈C12)
THEN
 act2   :   Etat_Aut2:∈Etat_Rec_Aut2[{etat↦msg}]
 act3   :   C12≔C12\{msg}
END
```

FIGURE 4.9 – Evènement de reception d'un message

Pour que l'automate2 puisse recevoir le message envoyé par l'automate1, les gardes 1 et 2 doivent être vérifiées. Par exemple : soient Etat_Aut2=S02, msg=T1

(T1∈C12) (T1∈MSGAut) (etat=S02) ***Vérifiée***

(S02↦T1∈dom(S02↦T1↦S12,S222↦T22↦S22) (T1∈C12) ***Vérifiée***

Alors :

Etat_aut2=S12 modification de l'état global de l'automate 2 . C12=C12\{T1} retirer le message du canal.

Les obligations de preuves relatives à ces invariants ont été déchargées automatiquement par le prouveur de la plateforme RODIN.

4.3.1.2. Premier raffinement

Il s'agit d'un modèle moins abstrait qui raffine le modèle initial sans le contredire. Ce modèle est plus précis en termes de détails ajoutés. En effet, nous avons introduit la notion complexe pour le modèle d'AEF. Une telle notion signifie qu'un état peut être lui-même un automate.

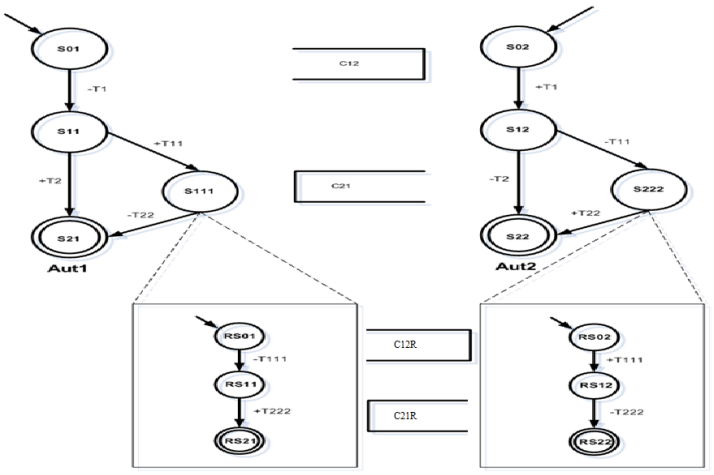


FIGURE 4.10 – Représentation du modèle raffiné

La modélisation en Event-B de la communication d'automates internes à états finis (modèle raffiné) se fait de la même manière que celle du modèle abstrait (ou aucun automate complexe n'est détaillé). Elle comporte une machine raffinée (MCAEF0) et trois contextes (Cntxt_AutR, Aut1R et Aut2R).

a) Les paramètres

Les paramètres à ce niveau sont décrits dans un contexte nommé comme suit :

Les ensembles :

- **SAut1R** modélisant l'ensemble des états de l'automate interne1.
- **SAut2R** modélisant l'ensemble des états de l'automate interne2.
- **MSGAutR** modélisant l'ensemble des messages intervenant lors de la communication entre les automates internes.

Les constantes :

- RS01, RS11, RS111, RS21 constantes formant l'ensemble *SAut1R*.
- RS02, RS12, RS222, RS22 constantes formant l'ensemble *SAut2R*.
- T111, T222 messages formant l'ensemble *MSGAutR*.

Les nouveaux paramètres introduits à ce niveau sont représentés comme suit :

```
CONTEXT
  Aut1R
EXTENDS
  Cntxt_Aut
SETS
  SAut1R
CONSTANTS
  RS01
  RS11
  RS21
  Etat_Envoi_Aut1R
  Etat_Rec_Aut1R
AXIOMS
  axm1  :  partition(SAut1R, {RS01}, {RS11}, {RS21})
  axm2  :  Etat_Envoi_Aut1R∈SAut1R×MSGAutR→SAut1R
  axm3  :  Etat_Envoi_Aut1R={RS01↦T111↦RS11}
  axm4  :  Etat_Rec_Aut1R∈SAut1R×MSGAutR→SAut1R
  axm5  :  Etat_Rec_Aut1R={RS11↦T222↦RS21}
END
```

FIGURE 4.11 – *Aut1R* au sein du système après raffinement

```
CONTEXT
  Aut2R
EXTENDS
  Cntxt_Aut
SETS
  SAut2R
CONSTANTS
  RS02
  RS12
  RS22
  Etat_Envoi_Aut2R
  Etat_Rec_Aut2R
AXIOMS
  axm1  :  partition(SAut2R, {RS02}, {RS12}, {RS22})
  axm2  :  Etat_Envoi_Aut2R∈SAut2R×MSGAutR→SAut2R
  axm3  :  Etat_Envoi_Aut2R={RS12↦T222↦RS22}
  axm4  :  Etat_Rec_Aut2R∈SAut2R×MSGAutR→SAut2R
  axm5  :  Etat_Rec_Aut2R={RS02↦T111↦RS12}
END
```

FIGURE 4.12 – *Aut2R* au sein du système après raffinement

b) Etat du système

L'état du système raffiné est décrit par quatre variables appelées ***Etat_Aut1R***, ***Etat_Aut2R***, ***C12R***, ***C21R*** modélisant respectivement l'état global de l'automate interne1 au court de la communication, idem pour l'automate interne2, état du canal C12R qui va de l'automate interne1 vers l'automate interne2 et état du canal C21R qui va de l'automate interne2 vers l'automate interne1, et typées par les mêmes invariants du modèle abstrait de la communication simple sans états complexes.

c) Les transitions

On y procède par les mêmes évènements que ceux du modèle abstrait, tout en garantissant la condition "*Etats globaux de l'automate1 et l'automate2 du modèle abstrait sont à leurs états complexes.*"

On note que les versions raffinées des événements Envoi_Aut1, Envoi_Aut2, Rec_Aut1 et Rec_Aut2 nommées respectivement Envoi_Aut1R, Envoi_Aut2R, Rec_Aut1R et Rec_Aut2R gardent les mêmes caractéristiques que leurs abstractions avec l'ajout d'une garde grd3 : (etat∈SAut1)∧(etat=Etat_Aut1)∧(Etat_Aut1=S111)

```
INITIALISATION:    extended ordinary ›
    THEN
        act1:   C12≔∅ ›
        act2:   C21≔∅ ›
        act3:   Etat_Aut1≔S01 ›
        act4:   Etat_Aut2≔S02 ›
        act5:   Etat_Aut1R≔RS01 ›
        act6:   Etat_Aut2R≔RS02 ›
        act7:   C12R≔∅ ›
        act8:   C21R≔∅ ›
    END
```

FIGURE 4.13 – Evènements *INITIALISATION* du raffinement

```
Envoi_Aut1R:    not extended ordinary ›
    ANY
        etat    ›
        etat1   ›
        msg1    ›
    WHERE
        grd1:   (etat1∈SAut1R)∧(msg1∈MSGAutR)∧(etat1=Etat_Aut1R) not theorem ›
        grd2:   (etat1↦msg1∈dom(Etat_Envoi_Aut1R))∧(card(C12R)<1) not theorem ›
        grd3:   (etat ∈ SAut1) ∧ (etat = Etat_Aut1) ∧ (Etat_Aut1= S111) not theorem ›
    THEN
        act1:   C12R={msg1} ›
        act2:   Etat_Aut1R:∈Etat_Envoi_Aut1R[{etat1↦msg1}] ›
    END
```

FIGURE 4.14 – Evènements *Envoi_Aut1R* du raffinement

```
Rec_Aut2R:  not extended ordinary ›
    ANY
        etat     ›
        etat1    ›
        msg1     ›
    WHERE
        grd1:   (etat1∈SAut2R)∧(msg1∈MSGAutR)∧(etat1=Etat_Aut2R) not theorem ›
        grd2:   (etat1↦msg1∈dom(Etat_Rec_Aut2R))∧(msg1∈C12R) not theorem ›
        grd3:   (etat ∈ SAut2) ∧ (etat = Etat_Aut2)∧(Etat_Aut2= S222) not theorem ›
    THEN
        act1:   Etat_Aut2R:∈Etat_Rec_Aut2R[{etat1↦msg1}] ›
        act2:   C12R=C12R\{msg1} ›
    END
```

FIGURE 4.15 – Evènements *Rec_Aut2R* du raffinement

Les obligations de preuves relatives à cette machine, ont été déchargées automatiquement par le prouveur de la plateforme RODIN.

Ce dernier modèle traduit la spécification formelle en Event-B du modèle d'une Communication d'Automates à Etats finis Complexes. La construction de cette spécification formelle est obtenue suite à l'application de la stratégie de *raffinement*. Toutes les preuves liées aux propriétés de sûreté (préservation de l'invariant et correction du raffinement) ont été déchargées (ou démontrées). Ainsi, on peut dire que cette spécification formelle du modèle CAEFC est correcte par construction.

4.3.3 Animation de la spécification formelle

Nous avons validé les modèles Event-B construits dans ce travail en menant une activité d'animation supportée par l'outil *ProB*.

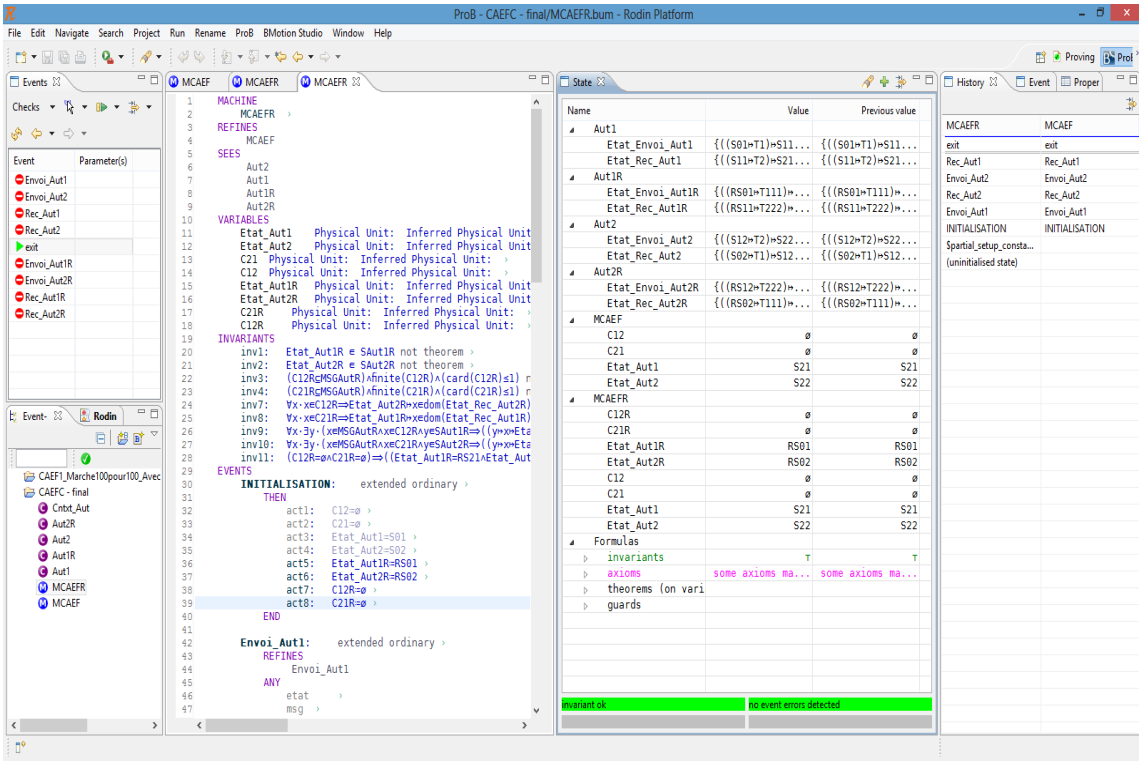


FIGURE 4.16 – Modèle CAEFC animé par ProB

En outre, une activité de preuve formelle permet de vérifier la cohérence du modèle.

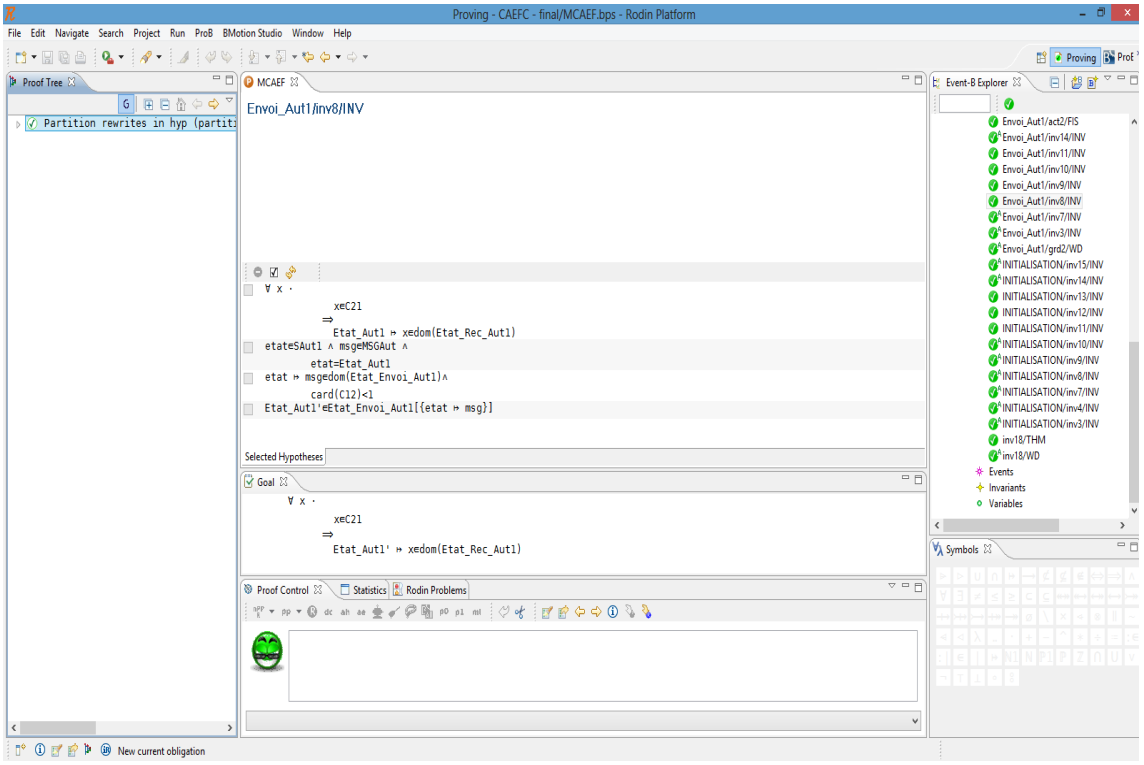


FIGURE 4.17 – Obligation de preuves du modèle

4.4 Conclusion

Dans ce chapitre, nous avons construit pas-à-pas une spécification formelle en Event-B d'une Communication d'Automates à Etats Finis Complexe (**CAEFC**). Pour l'avoir réaliser, nous avons élaboré une stratégie de raffinement adaptée au modèle de conversation complexe. Une telle stratégie a été appliquée avec succès et a donné naissance à une machine abstraite, une machine raffinée et trois contextes. En outre, nous avons validé la spécification formelle obtenue en utilisant avec profit le modele-***checkerProB***.

Conclusion générale et perspectives

La composition des services web vise à combiner les fonctionnalités de plusieurs services afin de répondre à des exigences auxquelles un seul service ne peut répondre individuellement. La manière dont la requête sera prise et l'enchaînement de services intervenant lors de la composition rencontrent de nombreux problèmes.

Les recherches menées dans ce domaine sont nombreuses. Plusieurs approches intéressantes ont été proposées afin de résoudre le problème de la composition, mais chacune d'elles présente des lacunes et impose ses contraintes auxquelles il faudra remédier au fil du temps. Après avoir parcouru les travaux réalisés, et étudié différentes approches, nous nous sommes intéressés aux approches formelles, ces dernières étant très avantageuses. Elles nous permettent d'avoir un certain niveau de connaissance préalable du comportement correct du modèle.

Nous avons proposé dans ce document un modèle formel, qui est une Communication des Automates à Etats Finis Complexes, qui a été vérifié et validé avec la méthode EventB. Le modèle obtenu est sans erreurs car la vérification et la validation sont faites pas-à-pas avec la construction du modèle.

Afin de compléter de rendre notre modèle de composition de services web plus rigoureux et plus exhaustif, de nouvelles perspectives s'ouvrent avec ce travail de recherche, notamment :

- La mise en pratique du modèle proposé ;
- La généralisation du modèle à un large nombre de services et la proposition d'un mécanisme efficace de mise en échelle ;
- La considéreration de l'outil de compensation de services ;
- L'ajout d'une couche sémantique au modèle.

Bibliographie

[1] DJEBARI.N,Spécification d'un Modèle Formel pour L'expression des autorisations, d'Accès aux Web Sevices,Mémoire magistère,2009

[2] Julien Guitton,Planification multi-agent pour la composition dynamique de services Web,Rapport de stage-Master 2 Recherche,2006

[3] Céline LOPEZ-VELASCO,Sélection et composition de services Web pour la génération d'applications adaptées au contexte d'utilisation,thèse , Université JOSEPH FOURIER, Novembre 2008

[4] H.Kadima,les web services, Dunod èdition,2003

[5] T.melliti, Interopérabilité des Services Web complexes, Université IX Dauphine, France, 2004

[6] R.de la Rosa-rosero.Découverte et selection de Service Web pour une application Malusine.PhD thesis, institut d'Informatique et de mathématiques Appliquées de Grenoble, France 2004

[7] J M.Chauvet, Services Web avec SOAP,WSDL,UDDI, ebXML...,Edition Eyrolles,2002

[8] Philippe Laublet, Jean Charlet et Chantal Reynaud, Introduction au Web sémantique,Univ. Paris-Sud,France,2004

[9] M. Mrissa, Sémantique Orientée Contexte pour la Composition de Services Web,Thèse doctorat, Université Claude Bernard Lyon I,2007

[10] Y.Charif, Chorégraphie dynamique de services basée sur la coordination d'agents introspectifs,Thèse doctorat, université pierre et marie curie paris vi,2007

[11] MADI.K, SÉLECTION GLOBALE DE SERVICES EN INFORMATIQUE UBIQUITAIRE,université de Montpellier,France ,2012

[12] T. Berners-Lee and L. Kagal The fractal nature of the semantic web. AI Magazine, 2008

[13] Berners-Lee,The original design and ultimate destiny of the World Wide Web,2000

[14] TA Tuan Anh,web sémantique et réseaux sociaux construction d'une memoire collective par recommandations mutuelles et (Re-)présentations.

[15] Samira Silhadi-Hacid,Malika Tarafi,web services,2006.

[16] Jinghai Rao and Xiaomeng Su,A Survey of Automated Web Service Composition Methods,Norwegian University of Science and Technology,2004

[17] A. Marconi, P. Bertoli , P. Traverso, M. Pistore, "Automated Composition of Web Services by Planning at the Knowledge Level",2005

[18] E.Newcomere,Undestanding Web Services Xml WSDL SOAP en UDDI,Oreilly edition,2004.

[19] M.Lallali,Modélisation et Test Fonctionnel de l'Orchestration de Services Web,INSTITUT NATIONAL DES TELECOMMUNICATIONS,2009.

[20] San-Yih Hwang, Ee-Peng Lim,Chien-Hsiang Lee, and Cheng-Hung ChenDynamic,Dynamic Web Service Selection for Reliable Web Service Composition,2008

[21] Farah Zoubeyr, Abdelkamel Tari, Aris M. Ouksel,Backward validation of communicating complex state machines in web services environments,2010

[22] Reynald Borer,Web Services Concepts, composants et mise en pratique,2007

[23] Sheshagiri, M., Desjardins, M., and Finin, T. A planner for composing services described in DAML-S. In Proceedings of the AAMAS Workshop on Web Services and Agent-based Engineering (Jun. 2003).

[24] McDermott, D. Estimated-regression planning for interactions with web services. In Proceedings of the 6th International Conference on AI Planning and Scheduling,Toulouse, France, 2002.

[25] Marlon Dumas and Reiko Heckel, editors. Web Services and Formal Methods, WS-FM'07 Workshop, Brisbane, Australia, Sep., 2007. Proceedings, volume 4937 of LNCS. Springer, 2008

[26] Mario Bravetti, Manuel Núñez, and Gianluigi Zavattaro, editors. Web Services and Formal Methods, WS-FM 2006, Proceedings, volume 4184 of Lecture Notes in Computer Science, Vienna, Austria, September 2006. Springer

[27] Idir AIT-SADOUNE, Modélisation et Vérification Formelles de Compositions de Services. Une Approche Fondée sur le Raffinement et la Preuve, Ecole Nationale Supérieure de Mécanique et d'Aérotechnique Laboratoire d'Informatique Scientifique et Industrielle, décembre 2010.

[28] John Hurst,Introduction to Event-B,Computer Science and Software Engineering,Monash University,2010

[29] Abrial J-R., Modeling in Event-B,System and Software Engineering, Cambridge,University Press, 2010.

[30] Abrial J-R., Le Cahier des Charges : Contenu, Forme et Analyse (en vue de la Formalisation),Juin 1998.

[31] Chill R., Logique et théorie des ensembles, Laboratoire de Mathématiques et Applications de Metz, Université de Metz, 2007/08.

[32] Hong Anh Le and Ninh Thuan Truong,Modeling and Verifying WS-CDL Using Event-B,VNU - University of Engineering and Technology,2013

[33] Ludovic Casset, Construction Correcte de Logiciels pour Carte à Puce Développement formel d'un vérifieur embarqué de byte code Java Card à l'aide de la méthode B,UNIVERSITE DE LA MEDITERRANEE,2002

[34] Nicolas Stouls,Explicitation du contrôle de développements B événementiel,29 avril 2004

[35] Imen SAYAR, D'Event-B vers UML/OCL en passant par UML/EM-OCL,Université de Sfax,Tunisie,13 Février 2012.

[36] Jens Bendisposto,Michael Leuschel,Olivier Ligot, Mireille Samia ,La validation de modèles Event-B avec le plug-in ProB pour RODIN,Université Notre-Dame de la Paix Namur Namur, Belgium , 2007

[37] Jean-Raymond Abrial,Summary of Event-B Proof Obligations,Mars 2008

[38] Thomas BOLUSSET,β-SPACE : Raffinement de descriptions architecturales en machines abstraites de la méthode formelle B,UNIVERSITE DE SAVOIE,2004

[39] Ariouat Assia,Aouicha Kahina, Approche bassée sur la thèorie des graphes pour la composition de services web,Université Abderrahmane Mira,Bejaia ; Algérie, 2008

[40] R.Shapiro."A technical comparison of xpdl,bpml and bpel4ws.Technicalreport,cape Vision,Software To Simplfy Complexity",New York,2002

[41] J.Ponge.Comptabilité et substitution dynamique des web services,Université Blaise Pascal-Clairmont II,2004

[42] S.Benmokhtar,Synthèse ad hoc de services Web dans les environnements de l'informatique diffuse,INRIA,2004

[43] [Rodin,2004] :Rodin (2004) European Project Rodin,University of Newcastle upon Tyne, UK,2004.

[44] [Rodin,2007] :Rodin (2007) User Manual of the RODIN Platform,October 2007.

[45] J.R. Abrial. The B-Book. Cambridge University Press, 1996

[46] J.-R. Abrial and L. Mussat. Introducing Dynamic Constraints in B. In D. Bert, editor, B'98 : The 2nd International B Conference, Recent Advances in the Development and Use of the B Method, volume 1345 of LNCS. Springer Verlag, 1998.

[47] Asma Tafat,Invariants et raffinements en présence de partage,Université Pierre et Marie Curie-Paris 6,18 septembre 2009.

[48] E.W. Dijkstra. A discipline of programming. 1976.

[49] R.J. Back. On the correctness of refinement in program development. PhD thesis, University of Helsinki, 1978.

[50] R.J. Back. A calculus of refinements for program derivations. Acta Informatica, 1988

[51] Christian Stahl,Transformation von BPEL4WS in Petrinetze. Master's thesis, Humboldt-Universität zu Berlin, Berlin, Germany,2004

[52] Schmidt, K. and Stahl, A Petri net semantic for BPEL4WS - validation and application. In Kindler, E., editor, 11th Workshop on Algorithms and Tools for Petri Nets,2004

[53] Hinz, S,Implementierung einer Petrinetz-semantik für BPEL. Master's thesis, Humboldt-Universität zu Berlin, Berlin, Germany,2005

[54] Hinz, S., Schmidt, K., and Stahl,Transforming BPEL to petri nets. In Springer-Verlag, editor, 3rd International Conference on Business Process Management, volume 2649 of Lecture Notes in Computer Science,2005

[55] IRIS T. STEWART,DANIEL R. CAYAN,MICHAEL D. DETTINGER,CHANGES IN SNOW-MELT RUNOFF TIMING IN WESTERN NORTH AMERICA UNDER A 'BUSINESS AS USUAL' CLIMATE CHANGE SCENARIO,Scripps Institution of Oceanography, California, U.S.A,2004.