BADJIMOKHTAR UNIVERSITY -ANNABA-
UNIVERSITEBADJIMOKHTAR -ANNABA-

جامعة باجي مختار
- عنابة -

Faculté:Sciences de l'Ingénieur -Année 2018-2019

Département: Informatique

THESE

Présentée en vue de l'obtention du diplôme de **Doctorat en Sciences**

La vérification de la compatibilité des processus métiers

Option : Intelligence Artificielle

Par

Mendjel Mohamed Said Mehdi

Soutenue le 19/09/2019

Devant le jury

Directrice de thèse	Hassina Seridi	Pr.	U.Badji Mokhtar-Annaba, Algérie.
Co-directeur de thèse	Farouk Toumani	Pr.	U.Blaise Pascal, France.
Président	Meslati Djamel	Pr.	U. Badji Mokhtar-Annaba, Algérie.
Examinateur	Farid Mokhati	Pr.	U. Oum El Bouaghi, Algérie.
Examinateur	Fateh Boutekkouk	Dr.	U. Oum El Bouaghi, Algérie.

Thèse

La vérification de la compatibilité des processus métiers.

Mohamed Said Mehdi Mendjel

À ceux qui me sont chers

Avant-propos

Remerciements

Je tiens à exprimer mes plus vifs remerciements à Mme le professeur Seridi-Bouchelaghem.H, qui fut pour moi une directrice de thèse attentive et disponible malgré ses nombreuses charges. Sa compétence, sa rigueur scientifique et sa clairvoyance m'ont beaucoup appris.
Soyez assuré de mon attachement et de ma profonde gratitude.

Je remercie Monsieur Toumani Farouk, Professeur à l'université Blaise Pascal de Clermont Ferrand pour m'avoir accueilli dans le laboratoire de recherche LIMOS et de m'avoir permis de travailler dans d'aussi bonnes conditions. Je lui suis reconnaissant de m'avoir fait bénéficier de sa grande compétence et de sa rigueur intellectuelle.

J'exprime tous mes remerciements à l'ensemble des membres du jury qui m'ont fait l'honneur de bien vouloir évaluer mon travail, et plus précisément :
Monsieur Djamel Meslati, Professeur à l'université Badji Mokhtar-Annaba- pour l'honneur qu'il m'a fait, en acceptant de présider ce jury.
Monsieur Farid Mokhati, Professeur à l'université d'Oum el Bouaghi, pour avoir accepté de juger le présent document.
Monsieur Fateh Boutekkouk, Professeur à l'université D'Oum el Bouaghi, d'avoir accepté de faire partie de ce jury.

J'adresse toute ma gratitude à tous mes ami(e)s et à toutes les personnes qui m'ont aidé dans la réalisation de ce travail.

Finalement et à titre plus personnel, je remercie chaleureusement mes parents et ma femme Houda pour leur soutien moral ininterrompu et leurs nombreux conseils tout le long de ma thèse.

Résumé

Ces dernières années, le monde des entreprises a connu une croissante utilisation des architectures logicielles indépendantes de la plate-forme et du langage de développement, ce type d'architecture est appelé architecture orientée services (*SOA*). L'arrivée de cette architecture avec la technologie des services Web a offert à la gestion des processus métiers une interopérabilité dynamique aussi bien au niveau intra qu'inter organisationnel. L'utilisation des services Web permet de réduire l'hétérogénéité des applications (en s'appuyant sur des interactions directes entre services ou bien en passant par une composition de service) et cela nécessite une vérification de la compatibilité des services participants à la conversation. Cette dernière peut être traduite par la compatibilité des interfaces en se basant sur les protocoles standards et les fichiers descriptifs de la spécification des services Web tels que *SOAP* et *WSDL*, mais ce type de protocoles installent un cadre minimum pour faire fonctionner un service. En effet, *WSDL* affiche l'interface du service adéquat mais omet une information importante qui lui permet d'interagir avec d'autres services. Cette information est la séquence correcte de messages envoyés et reçus par le service. Pour remédier à ce problème, les chercheurs ont proposé un modèle formel pour représenter les séquences correctes de messages, ce modèle est appelé les protocoles de services ou bien les protocoles métiers (business protocols). Dans cette thèse, on s'est intéressé particulièrement à la l'analyse de la compatibilité des business protocoles qui ont un lien avec les bases de données, on les appelle les protocoles métiers orientés données. On a choisi cet axe de recherche, car la présence des données représente une source d'infinité et cela représente un problème majeur lors de l'analyse et la vérification.
Enfin, et dans le but de valider notre analyse théorique, nous avons réalisé un vérifieur de compatibilité basé sur des protocoles de services gardés par des instances de base de données.

Mots clés : protocoles de services Web, protocoles métiers, artéfacts, protocoles de services orientés données, la compatibilité des protocoles de services, l'infinité de données.

Abstract

In recent years, business world has experienced an increasing use of software architectures that are independent from platforms and development languages, this type of architectures is called service oriented architecture (SOA). The arrival of this architecture with Web services technology offered to the business processes management a dynamic interoperability in intra and inter organizational levels. The use of Web services reduces the heterogeneity between applications (relying on direct interactions between services or through a service's composition), this requires verification of compatibility between communicating services. The compatibility term can be translated by interfaces compatibility based on web services standard protocols such as *SOAP* and *WSDL*, but this type of protocols is very basic for service's runs . Indeed, *WSDL* displays service's interface but omits an important information that allows it to interact with other services. This information is the correct sequence of messages sent and received by the service. To address this problem , researchers have proposed a formal model to represent the correct sequences of messages called service protocol or business protocol. In this thesis, we are particularly interested in the analysis of compatibility between business protocols augmented with a databases. We chose this research because the presence of data is a source of infiniteness which is a major problem in analysis and verification.

Finally, and in order to assess the difference between the theoretical analysis and the practical side, our analysis leads to the implementation of a compatibility verifier based on service's protocols guarded by a symbolic database instances.

Keywords : Web services protocols, business protocols, artifacts, data-services protocols, compatibility of business protocols, infinity of data.

ملخص

في السنوات الأخيرة، عرف عالم الأعمال نمو كبير في إستعمال معمارية الأنظمة المستقلة عن منصات و لغات البرمجة. هذا النوع من المعمارية يطلق عليه إسم : معمارية ذات التوجه الخدماتي. ظهور هذه المعمارية مع تكنولوجيا خدمات الواب سمح لإدارة عمليات الأعمال إنشاء عمل بيني ديناميكي على المستويين الداخلي و الخارجي لشركات. إستعمال خدمات الواب تسمح بتخفيض عدم تجانس التطبيقات (بالإعتماد على التفاعل المباشر بين خدمات الواب أو خدمات الواب المركبة) ، و هذا يستدعي التأكد من توافق خدمات الواب المشاركة في المحادثة. هذه الأخيرة يمكن أن تترجم بتوافق واجهات خدمات الواب طبقا لبروتوكولاتها المعيارية مثل : $SOAP$ و $WSDL$ ، و لكن هذه النواع من البروتوكولات تعتبر بسيطة لتفعيل خدمة واب. و بالفعل $WSDL$ يعرض واجهة خدمة الواب المعنية لكن يحذف معلومة جد هامة و المتمثلة في التسلسل الصحيح في الرسائل عند إرسالها و إستقبالها عبر خدمة الواب.

لمعالجة هذه المشكلة، إقترح الباحثون نموذجا رسميا لتمثيل التسلسل الصحيح في الرسائل، هذا النموذج يسمى ببروتوكولات الخدمة أو البروتوكولات التجارية. في هذا العمل إهتمامنا كان منصبا بشكل خاص على عملية تحليل التوافق أو المطابقة بين البروتوكولات التجارية المتواصلة مع قواعد البيانات و تسمى بالبروتوكولات التجارية الموجهة للبيانات.

و من هذا فإن إهتمامنا بهذا الموضوع ناتج على أن وجود البيانات بشكل لا نهائي يطرح إشكال كبير في التحليل و التحقق.

وأخيرا، تم إجراء تجربة من خلال تنفيد فرعي من تحليلنا من أجل تحقيق نموذجا شامل متوافق مع أي نوع من البروتوكولات.

الكلمات الدلالية : ببروتوكولات خدمات الواب، البروتوكولات التجارية، البروتوكولات الموجهة للبيانات، التوافق بين البروتوكولات التجارية، لا نهاية البيانات.

Table des matières

Table des figures

Liste des tableaux

“ La preuve de la valeur d’un système informatique est son existence.

Alain Jay Perlis – « Epigrams on Programming- 1982 »

1

Introduction

1.1 Contexte

ACTUELLEMENT, l'objectif principal des entreprises est de faire face aux changements rapides de l'environnement (c'est à dire la nécessité de mettre en œuvre le cœur métier de l'entreprise sur internet, l'adaptation aux besoins des nouveaux clients, la manière d'agir vite en cas de défaillance du système interne, assurer une collaboration continue avec d'autres entreprises externes ou une collaboration à l'intérieur de l'entreprise, etc.). Pour cela, elles doivent assurer l'intégration et l'interopérabilité de leurs applications au niveau interne *EAI*[1] et au niveau partenaire *B2B*[2], ces applications sont généralement appelées processus métiers. Ces besoin ont donné naissance à de nouvelles technologies appelées les services web. Selon Alonso et *al,* dans [Alonso04], un service web est une plate-forme indépendante et un composant logiciel accessible par machine qui peut être décrit, publié et invoqué sur le réseau en utilisant des protocoles de communication standards et des langages de description par exemple : *SOAP*[3] qui est un protocole d'échange d'informations, *WSDL*[4] qui permet de décrire les interfaces des services web et *UDDI*[5] utilisé pour publier et découvrir les services web. Dans le prochain chapitre, on va détailler ces trois termes. Le but visé des technologies sous-jacentes est de créer des composants logiciels capables d'interagir ensemble d'une manière hétérogène c'est-à-dire faciliter l'interaction entre les différents composants s'exécutant sur des applications et des systèmes autonomes.

1.2 Problématique

Récemment, les modèles formels centrés processus utilisés pour décrire les spécifications des services ont évolué vers des modèles centrés données [Calvanese13]. Par exemple, dans certains travaux [Yellin97, Benatallah06a, Benatallah04b], le modèle centré processus utilisé pour la spécification se focalise uniquement sur

1. Entreprise Application Integration
2. Business To Business
3. Simple Object Access Protocol
4. Web services Description language
5. Universal Description Discovery and Integration

le flux des actions mais il omet complètement les données. Actuellement, les entreprises intègrent énormément de données dans leurs processus métiers dans le but de réaliser des systèmes d'informations flexibles [Reichert12]. De ce fait, les systèmes d'informations des organisations ayant atteint un certain stade de complexité sont confrontés à des problèmes classiques : comment intégrer un service web orienté données selon les besoins des clients ? Ou bien, comment remplacer un service web orienté données par un autre service web compatible, pour assurer le bon fonctionnement du système ?

Cette idée exige une vérification formelle des modèles centrés données. Car en réalité, la présence de données rend les modèles infinis, mais cette infinité de données ne conduit pas nécessairement à l'indécidabilité de la vérification.

1.3 Objectif de la thèse

Une description de service n'est pas nécessairement une interface de service, mais elle peut être représentée par le protocole métier pris en charge par le service (ou simplement le protocole de service). Ce protocole décrit la spécification des conversations soutenues par les services, cette spécification joue un rôle important pour assurer une interaction correcte entre deux services donnés.

Informellement, nous définissons un protocole de service centré données comme étant une description du comportement externe du service web qui est en interaction continue avec une base de données (c'est-à-dire nous décrivons le processus et les données au même niveau). Nous modélisons ce protocole avec une machine à états finis où les transitions sont étiquetées par des messages envoyés et reçus et les états représentent les différentes phases d'interaction du service web. Une conversation est une séquence de messages échangés entre deux services. Dans ce cas, l'étude de la compatibilité porte sur le problème de savoir si les deux protocoles peuvent échanger des messages ou pas.

Le principal objectif de notre travail est d'effectuer une analyse complète de la compatibilité des protocoles de services orientés données en définissant différents sous-modèles de protocoles (par exemple, des protocoles gardés, non gardés, des protocoles augmentés avec des instances de base de données ou bien augmentés avec un schéma de base de données) et d'identifier formellement le niveau de compatibilité entre deux protocoles de services donnés. Dans cette analyse, nous discutons seulement des aspects syntaxiques et comportementaux de la compatibilité, dans laquelle nous prenons en considération les types et les séquences possibles de messages échangés régulièrement entre les services web. Par ce qu'on peut trouver dans la littérature d'autres aspects importants et pertinents pour identifier la compatibilité entre les services web tels que l'aspect sémantique ou l'aspect qualité du service. Mais en fonction de la complexité du problème d'intégration des données qui est notre objectif principal, notre analyse est largement suffisante pour recouvrir toute la thèse.

1.4 Contributions de la thèse

Les principales contributions de cette thèse sont résumées comme suit :

1. La vérification formelle de la compatibilité de plusieurs sous-modèles de protocoles de services cela en fonction des langages utilisés :
   - Analyser la compatibilité des protocoles gardés de type $(L_G, \phi, \phi)^I$,
   - Analyser la compatibilité des protocoles non gardés de type $(\phi, \phi, L_U)^I$,
   - Analyser la compatibilité des protocoles de type $(L_G, L_M, L_U)^I$,
   - Analyser la compatibilité des protocoles gardés de type $(L_G, \phi, \phi)^{DB}$,
   - Analyser la compatibilité des protocoles non gardés de type $(\phi, L_M, \phi)^{DB}$.
2. l'implémentation d'un vérifieur de compatibilité qui se base sur des bases de données et des requêtes symboliques.

1.5 Organisation de la thèse

Cette thèse est structurée en deux parties principales : la première partie montre un état de l'art de la problématique traitée et la deuxième représente notre

contribution, elle est organisée comme décrit dans cette section.

Après un premier chapitre qui présente une introduction générale avec le contexte, la problématique et les objectifs, cette thèse se décline en six autres chapitres.

La partie état de l'art introduit les concepts de base manipulés dans cette thèse, l'arrivée des services web et leurs relations avec les processus métiers, les différentes manières de modéliser un protocole de service web et enfin les différentes méthodes de vérification formelle. Cet état de l'art comprend donc trois chapitres.

Le second chapitre intitulé « Architecture orientée services et processus métiers » mène le lecteur depuis l'architecture basée sur la programmation modulaire jusqu'à l'arrivée de l'architecture orientée services. Il présente aussi la technologie des services web avec ses différents protocoles et langages de communication. Enfin, il décrit la relation existante entre les services web et les processus métiers.

Le troisième chapitre intitulé « La modélisation des services web » présente les différents formalismes de modélisation des services web et plus particulièrement la modélisation des « business protocols » qui spécifient le comportement des services web. Le but de ce chapitre est d'effectuer une étude approfondie sur les différents paradigmes de modélisation pour pouvoir choisir le paradigme qui répond aux exigences de notre analyse.

Tandis que le quatrième chapitre qui se nomme « La vérification formelle des systèmes informatiques » étudie les différentes méthodes de vérification tout en mettant l'accent sur la vérification classique des systèmes infinis dans lesquels les états finis sont remplacés par une infinité d'instances de base de données. Dans ce chapitre, On va aussi aborder le problème de l'équivalence et de la compatibilité des processus orientés données en se penchant vers les systèmes adoptant une

architecture orientée services et nécessitant à un moment donné une intégration de nouveaux services.

Deux chapitres viennent ensuite, dévoiler notre contribution qui présente une analyse complète de la décidabilité de la compatibilité des protocoles de services. Ces deux chapitres sont présentés comme suit :

Le cinquième chapitre qui s'intitule « Analyse formelle de la compatibilité des protocoles de services orientés données » présente formellement l'analyse de la compatibilité des protocoles de services web orientés données cela en créant plusieurs sous-modèles de protocoles et en analysant à chaque fois la décidabilité de la compatibilité par la méthode de vérification théorème/preuve avec la définition des classes de compatibilité existantes.

Le sixième chapitre qui s'intitule « implémentation et expérimentation » présente la mise en œuvre d'un vérifieur générique nommé « A generic guarded service protocol verifier » qui permet de vérifier la compatibilité des protocoles des services web gardés par des requêtes conjonctives et une base de données symbolique.

Enfin, nous terminons cette thèse avec un dernier chapitre « Travaux connexes et Conclusion générale » dans lequel nous présentons quelques travaux connexes avec une synthèse de notre contribution ainsi que la suggestion de quelques perspectives.

“ La mémoire fonctionne comme un processus constructif qui ne se contente pas de reproduire, mais qui filtre, change et interprète le passé.

Michelle Richmond, « L’année brouillard »

2

Architecture orientée services et processus métiers

2.1 Introduction

Au cours des quarante dernières années, les systèmes informatiques se sont développés de manière exponentielle et les architectures logicielles sont devenues de plus en plus complexes et difficiles à gérer. Les architectures traditionnelles ont atteint leurs limites en termes de capacité, alors que les besoins traditionnels des organisations informatiques demeurent. Les départements informatiques doivent toujours répondre rapidement aux nouvelles exigences des entreprises, tout en réduisant constamment le coût informatique de l'entreprise et en absorbant et intégrant de manière transparente de nouveaux partenaires et clients commerciaux. Le secteur d'activité des logiciels a connu de nombreuses architectures conçues pour permettre un traitement entièrement distribué, des langages de programmation destinés à n'importe quelle plate-forme et une myriade de produits de connectivité conçus pour permettre une intégration plus rapide et plus efficace des applications, et pour réduire considérablement les temps de mise en œuvre. Toutefois, une solution globale ne semble être encore qu'une utopie.

Pour remédier à ces problèmes, une architecture orientée services (Service oriented Architecture) [Davies08] s'est présentée comme l'étape suivante de l'évolution qui permet d'aider les organisations informatiques à relever des défis toujours plus complexes.

Dans ce chapitre, nous donnons un aperçu du domaine d'intérêt en présentant en premier lieu l'apparition des différentes architectures jusqu'à l'arrivée de la SOA [1], on présentera par la suite les différentes caractéristiques du paradigme SOA ainsi que tous les concepts de base sur lesquels elle s'articule. En suite, nous nous intéressons à la technologie des services web avec ses principaux standards, langages et protocoles relatifs à sa mise en œuvre tout en mettant l'accent sur les services web orientés données. Enfin, nous introduisons les processus métiers (Business process), et nous clarifions leurs relations par rapport aux services web.

1. Service Oriented Architecture

2.2 L'évolution de l'architecture

L'évolution de l'architecture est la conséquence de principaux facteurs tels que :

- La prise de conscience que la méthode précédente était imparfaite (souvent due à une évolution des besoins de l'utilisateur également),
- L'évolution de la capacité des machines (tant en mémoire qu'en puissance de calcul) se traduisant par l'évolution de leurs fonctionnalités,
- La complexité des systèmes informatiques et leurs coûts.

Nous allons étudier les trois évolutions majeures de l'architecture de programmation : la programmation modulaire, la programmation orientée objet et enfin la programmation orientée composant. La section suivante présentera l'étape suivante : l'architecture orientée service et les technologies des services web.

2.2.1 Architecture basée sur la programmation modulaire

Les principales directives de la programmation modulaire sont :

- Supprimer la programmation reposant sur les instructions « goto » permettant de faire des sauts en avant ou en arrière dans le code. Ceci en privilégiant l'utilisation de boucles telles que while, repeat,... until, for, etc.,
- Découper le code en module cohérents (s'apparentant généralement à des fichiers) : un module traite un type de donnée abstrait comme une liste, ou un type « concret » comme les informations caractérisant une personne par exemple,
- Éviter les variables globales que tout le code peut accéder à n'importe quel moment et modifier sans aucune vérification,
- Utiliser des fonctions ou des procédures pour structurer le code : une fonction ou procédure pour un traitement bien précis sur un type de donnée bien précis. De plus, l'utilisation des fonctions permet de « factoriser » le code,
- Utiliser la compilation séparée : un module est compilable de façon autonome, et l'ensemble des fichiers compilés seront liés ensemble pour devenir un code exécutable,
- Même si les réseaux n'étaient pas aussi utilisés qu'aujourd'hui, le concept

d'utilisation et d'invocation à distance, dans le cadre d'un système réparti était déjà présent. Les programmeurs utilisent, dans le cadre de la programmation modulaire, les mécanismes Remote Procedure Call (RPC) [Tay90].

Le mécanisme RPC permet de développer une application basée sur le modèle client-serveur. Un appel à distance est effectué par le client en envoyant un message à un système distant (serveur), pour exécuter une procédure donnée en utilisant les arguments fournis. Le résultat (calculé par le serveur) est ensuite envoyé au client.

De nombreuses implémentations sont nées de cette norme, et beaucoup sont incompatibles entre elles. Mais le principe était là : le client et le serveur pouvaient utiliser un encodage des données différent, un proxy réalise une conversion juste après l'appel de la fonction par le client, juste avant l'envoi effectif au serveur ou lors de la réception du message de réponse.

2.3 Architecture basée sur la programmation orientée objet

Parallèlement à l'arrivée de la programmation structurée, la programmation objet est arrivée au début des années 80. La programmation objet reprenait le concept du regroupement des données et des fonctions (appelées méthodes en terminologie objet), pour obtenir un seul module : une classe, qui, une fois instanciée, devient un objet.

Deux avantages principaux à cette architecture :

– Il existe une notion de regroupement du code et des données dans un seul et même objet, et ce, en forçant, si nécessaire leur non accessibilité depuis l'extérieur de cet objet : cela permet, par exemple, d'empêcher la modification d'une variable par du code n'appartenant pas à l'objet. Ce principe renforce le principe de séparation modulaire, de compilation séparée et de portée

des variables dans les procédures et fonctions issues de la programmation modulaire,

– Il existe un système d'héritage permettant de raffiner le comportement d'une classe : par exemple, une personne peut être un étudiant, un professeur ou un stagiaire. Cet héritage fournit des mécanismes tels que le polymorphisme et l'encapsulation.

  SmallTalk [Goldberg83] fait parti des langages objets cités, le plus souvent, comme étant le plus fidèle du principe objet. Mais suivant les langages, des notions différentes peuvent apparaître. Par exemple, le C++ utilise le concept d'héritage multiple alors que le Java ne permet que l'héritage simple.

Une classe fournit donc un ensemble complet de méthodes permettant de traiter des données. Le programmeur peut alors :

– Instancier un nouvel objet (qui fera appel aux constructeurs de la classe pour l'initialiser),
– Accéder aux méthodes publiques (ou privées à l'intérieur d'un objet) pour obtenir ou modifier les propriétés de l'objet,
– Utiliser un mécanisme de destruction de l'objet (dont le comportement varie fortement d'un langage objet à un autre).

Au niveau des mécanismes de répartition, on parle alors d'objets répartis. Ce sont des objets de « base » qui évoluent dans une architecture répartie, bien souvent à base de bus (ORB [2] dans le cadre de CORBA et Java RMI [3] dans le cas de Java). Ce bus est une partie logiciel faisant partie d'un intergiciel (ou middleware), visant à s'affranchir des problèmes liés à la répartition, en gérant l'interopérabilité ou encore la portabilité des objets.

Pour permettre la portabilité du composant, la description de la partie dépendante de la plateforme est faite dans un langage portable, et un générateur de code produit du code spécifique pour une plateforme cible : ce code servira de passerelle entre les clients et l'application métier (mécanisme RMI de Java par exemple).

2. Object Request Broker
3. Remote Method invocation

2.3.1 Architecture orientée composant

L'architecture orienté objet et ses objectifs initiaux ne permettent pas d'utiliser la puissance et les nouvelles méthodes issues de l'évolution des architectures (grilles de calcul, l'utilisation du pair à pair, architecture 3-tiers ou même n-tiers) tendant toujours vers plus d'uniformisation tout en augmentant la complexité des modèles, et ce dans le but de séparer la logique métier et la logique système. Une nouvelle architecture est apparue progressivement répondant à ces besoins : l'architecture orientée composants.

Cette nouvelle architecture tend à proposer des solutions aux problèmes courants rencontrés dans les systèmes réparties qui sont :

- La possibilité d'utiliser à large échelle les solutions proposées, par exemple à travers internet,
- La gestion de l'hétérogénéité des systèmes,
- La prise en compte des communications asynchrones et la gestion du contrôle concurrent,
- La prise en compte des pannes partielles concernant le logiciel, le réseau (surcharge, coupure temporaire, etc.),
- Et enfin, un élément indispensable qui est la gestion de la sécurité [Gallien07].

La prise en compte de ces problèmes aboutit à des solutions basées sur les bus à objets vu précédemment. Elles sont généralement englobées dans un framework, c'est-à-dire une boîte à outils permettant de « tout » faire. Ces frameworks peuvent être à la fois visuels (par exemple, un IDE [4] de type RAD (développement rapide d'applications) ou abstraits gérant le déploiement à travers un réseau des différents composants, pouvant même gérer, dans certains cas, les différentes versions de ceux-ci.

4. abrégé EDI en français ou IDE en anglais, pour integrated development environment

2.4 Les problèmes liés à l'intégration d'applications

Depuis que CORBA a pensé que l'intégration d'applications sur des plates-formes hétérogènes était possible, beaucoup de problèmes sont apparus et deviennent de plus en plus complexes chaque année.

Parmi ces problèmes on peut citer :

2.4.1 La complexité

Aujourd'hui, certains aspects informatiques ont changé, les environnements sont plus complexes, l'accès à faible coût et omniprésent à internet a créé de nouveaux modèles de développement surtout pour les entreprises qui souhaitent fusionner des applications entières qui répondent à leurs besoins. Dans un environnement d'une telle complexité, les solutions point à point ne font qu'exacerber les problèmes et ne relèvent jamais véritablement les défis. Donc il faut développer des systèmes qui incorporent le caractère hétérogène des organisations, en tant que composant essentiel de l'environnement informatique, de sorte que ces systèmes puissent prendre en charge une gamme diverse et sans limite de matériel, de systèmes d'exploitation, de logiciels intermédiaires, de langages et de stockage de données.

2.4.2 Une programmation redondante et ne pouvant être réutilisée

Comme dans beaucoup de sociétés, l'éventail des applications a peut-être augmenté en raison de fusions ou d'acquisitions d'autres entreprises. Le résultat est de faire face à des applications redondantes ou à des applications dont les fonctions ne peuvent être facilement réutilisées. Il se peut que chaque unité de l'organisation ait travaillé indépendamment des autres, ce qui freine l'effort de coordination visant à créer un parc ou des services informatiques fonctionnels et réutilisables. Cette redondance augmente les coûts et le temps de mise sur le marché pour le déploiement

de nouveaux produits et services, car les modifications doivent être effectuées pour chaque application ou système concerné. Le fait de ne pouvoir réutiliser certaines fonctions requiert, en fin de compte, plus de ressources, et souvent plus de temps, pour livrer de nouvelles applications.

2.4.3 Des interfaces multiples

Il faut penser au problème d'intégration n×(n-1). Les développeurs d'entreprises par exemple doivent faire face à des problèmes d'intégration, quelle que soit leur nature, en raison d'une fusion, d'un nouveau partenariat commercial ou simplement d'un besoin d'interconnexion entre des systèmes existants. Si n systèmes d'applications doivent être directement interconnectés, le processus va créer n×(n-1) connexions ou interfaces.

2.5 Les exigences de la programmation d'aujourd'hui

Au vu des problèmes débattus précédemment, il devrait être évident qu'il est important de développer une architecture qui satisfait un ensemble d'exigences. Ces exigences devraient inclure les éléments suivants :

- L'exploitation du parc informatique existant,
- La prise en charge de tous les types d'intégrations requis,
- La possibilité de mises en œuvre incrémentielles et de migration du parc,
- La constitution dans un cadre de composants standards,
- La possibilité de mise en œuvre de nouveaux modèles informatiques.

L'architecture qui répond à ces exigences s'appelle « Architecture Orientée Services », elle permet de concevoir des systèmes logiciels fournissant des services à d'autres applications à l'aide d'interfaces publiées et découvrables, et où les services peuvent être sollicités via un réseau.

Ce nouveau modèle d'architecture est apparu avec l'apparition de ce qu'on appelle les services web, car son principal objectif était de publier (sur internet, un extranet ou encore un intranet) un ensemble d'offre (existant ou non) réalisant

un service bien précis (par exemple un service pour crypter des données, un autre pour authentifier une personne, etc.). Une apparition dans ce cadre appelle ces différents services quand elle en a besoin et peut être elle-même déployée comme un nouveau service.

2.6 Une architecture orientée services

L'évènement des services web a précipité le changement fondamental dans le développement, le déploiement et la gestion des infrastructures informatiques. La réussite de nombreux projets de services web a montré que la technologie permettant de mettre en œuvre une véritable architecture orientée services existe. Elle permet de prendre du recul pour examiner l'architecture des applications. D'un point de vue de l'entreprise, le problème n'est plus uniquement un problème technologique. Il s'agit de développer une architecture et un cadre d'applications au sein desquels on pourra définir des problèmes et mettre en œuvre des solutions qu'on pourra réutiliser de manière cohérente.
Toutefois, il est important d'abord de comprendre qu'une architecture orientée services ne se limite pas aux services web. Les services web constituent un ensemble de technologies [Alonso10], y compris les technologies XML, SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language) et UDDI (Universal Description, Discover and Integration), qui nous permet de créer des solutions de programmation pour des problèmes de messagerie et d'intégration d'applications spécifiques. Ces technologies sont suffisantes et ont déjà démontré qu'on peut dès à présent mettre en œuvre une architecture orientée services.

2.6.1 Les éléments constitutifs d'une architecture orientée services

Une architecture orientée services porte bien son nom car il s'agit en effet d'une architecture. Ce n'est pas uniquement un ensemble spécifique de technologies, tels que les services web. Elle transcende ces technologies et en théorie, en est complètement indépendante [Papazoglou01]. Au sein d'un environnement d'entreprise, une

définition purement architecturale d'une architecture orientée services pourrait être la suivante : « Architecture d'applications au sein de laquelle toutes les fonctions sont définies en tant que services indépendants, comprenant des interfaces bien définies qui peuvent être sollicitées dans des séquences définies » [Papazoglou07].
Donc un système basé sur ce type d'architecture doit effectuer et gérer la sollicitation du service et non l'application appelante. Cette fonction permet la réalisation de deux caractéristiques essentielles :
Tout d'abord, l'indépendance véritable des services et deuxièmement, la possibilité de gérer ces derniers.
La gestion comprend plusieurs fonctions :

- La sécurité, l'autorisation des requêtes, le cryptage et le décryptage des données, selon les instructions données, et la validation des informations,
- Le déploiement permet au service d'être déplacé au sein du réseau pour maximiser les performances et supprimer la redondance afin que la disponibilité des applications soit maximale,
- La maintenance permet de gérer de nouvelles versions du service.

2.6.2 La nature d'un service

Un service au sein d'un environnement d'entreprise constitue en général une fonction simple, une transaction plus complexe ou un service système. D'un point de vue de l'application, ces fonctions sont des fonctions non système atomiques. Les transactions peuvent sembler n'être que de simples fonctions pour l'application appelante, mais elles peuvent être mises en œuvre en tant que fonctions composites masquées par leur contexte transactionnel propre. Elles impliquent parfois de multiples fonctions de niveau inférieur transparentes pour l'appelant. Les fonctions systèmes sont des fonctions généralisées qui peuvent être extraites d'une plateforme spécifique comme Microsoft Windows ou Linux.
Ceci n'est pas une simple distinction artificielle entre les différents services. D'un point de vue de l'application, tous les services sont atomiques, mais le fait qu'il s'agisse de services d'entreprise ou système n'a aucune importance. Cette distinction ne sert qu'à introduire le concept important de granularité. La décomposition des applications d'entreprise en services ne constitue pas uniquement un proces-

sus abstrait, elle implique également des opérations très pratiques. Les services peuvent être composés de fonctions de niveau inférieur (granularité fine) ou de niveau supérieur complexe (granularité grossière). Il existe de véritables compromis de performance, de flexibilité, de possibilités de maintenance et de réutilisation basés sur les définitions de ces fonctions. Le niveau de granularité indique la richesse fonctionnelle d'un service. Par exemple, plus la granularité d'un service est grossière, plus la fonction offerte par le service est riche.

Les services sont en général composés de fonctions commerciales à granularité grossière, telles que le service « ouvrir un compte », car ces opérations peuvent supposer l'exécution de multiples opérations à granularité plus fine, telles que le service « vérifier l'identité du client » et le service « créer le compte client ». Ce processus de définition de services est en général effectué dans un cadre plus large, le cadre d'applications. C'est ce cadre qui doit être mis en place. Il s'agit de développer un cadre d'applications de composants dans lequel les services sont définis en tant que ensemble de composants réutilisables pour créer de nouvelles applications ou intégrer un parc logiciel existant.

2.7 Les avantages du déploiement d'une architecture orientée services

Une architecture orientée services peut souvent évoluer à partir de systèmes existants et ne requiert donc pas de réécriture système complète.

Les organisations qui concentrent leurs efforts de développement sur la création de services, en utilisant des technologies existantes associées à une approche basée sur les composants vis-à-vis du développement logiciel, profiteront de nombreux avantages.

2.7.1 L'exploitation du parc informatique existant

Cet avantage est le premier et le plus important de toutes les exigences mentionnées précédemment. On peut constituer un service en ajoutant des composants existants à l'aide d'un cadre d'architecture orientée services adapté et mis à la disposition de la plate-forme ou l'application sur laquelle on travaille. L'utilisation de ce nouveau service requiert uniquement la connaissance de l'interface et du nom du service, la complexité du flux de données au sein des composants du service, est transparente pour les appelants. L'anonymat des composants permet aux organisations de tirer profit des systèmes actuels, de créer des services à partir d'un ensemble de composants installés sur différents ordinateurs exécutant différents systèmes d'exploitation et développés dans des langages de programmation différents. Les systèmes existants peuvent être encapsulés et leur accès est possible grâce aux interfaces des services web. Plus important encore, ils peuvent être transformés et prendre ainsi de la valeur à mesure que leur fonctionnalité est transformée en services.

2.7.2 L'infrastructure comme matière première

Le développement et le déploiement de l'infrastructure deviennent de plus en plus cohérents sur les diverses applications qu'on veut rassembler. Les composants existants, les nouveaux composants peuvent être consolidés au sein d'un cadre d'architecture orientée services bien défini. Un tel ensemble de composants est déployé sous forme de services dans l'infrastructure existante. L'infrastructure sous-jacente devient alors une matière première. Ensuite, à mesure que le couplage des services et du matériel prenant ces derniers en charge devient plus lâche, on peut optimiser le matériel car le programme d'assemblage de services ne dépend plus de l'environnement matériel dans lequel le service fonctionne au moment de l'exécution.

2.7.3 Le temps de développement

Les bibliothèques d'organisation de services web deviennent les atouts essentiels des organisations en faisant partie du cadre d'architecture orientée services.

La création et le déploiement de services à l'aide de ces bibliothèques de services web réduit considérablement le temps de développement, car les nouvelles initiatives réutilisent les services et les composants existants, réduisent le temps de conception, de test et de déploiement du processus.

2.7.4 L'atténuation des risques

La réutilisation de composants existants réduit le risque d'introduction de nouveaux échecs dans le processus d'amélioration ou de création de nouveaux services. Le risque de maintenance et de gestion de l'infrastructure prenant en charge les services diminue également.

2.7.5 Une architecture centrée autour des processus

Dans une architecture centrée autour des processus, l'application est développée pour le processus. Ce dernier est décomposé en une série d'étapes qui représentent chacune un service. En réalité, chaque fonction de service ou de composant constitue une sous-application. Ces sous-applications peuvent être assemblées pour créer un flux de processus capable de satisfaire les besoins de l'organisation.

2.8 Définition et description de Services Web

Le consortium W3C[5] (The World Wide Web consortium) qui travaille sur les services web a utilisé la définition de service web suivante : « Un service web est un système logiciel destiné à supporter l'interaction ordinateur-ordinateur sur le réseau. Il a une interface décrite en un format traitable par l'ordinateur (par exemple : WSDL). Autres systèmes réagissent réciproquement avec le service web d'une façon prescrite par sa description en utilisant des messages SOAP, typiquement transmis avec le protocole http et une sérialisation XML, en conjonction avec d'autres standards relatifs au web » [Austin04].

5. http ://www.w3.org/

2.9 Les principales technologies de développement des Services Web

Dans la littérature, plusieurs taxonomies ont été proposées pour classer les différents aspects des services web. Le point commun entre ces taxonomies réside au niveau des propriétés fonctionnelles qui décrivent ce que fait exactement un service et comment ses fonctionnalités sont accomplies. Le modèle fonctionnel des services web s'articule autour de différents langages et protocoles qui sont les suivants :

2.9.1 XML (eXtensible Markup Language)

XML « eXtensible Markup Language » [Bray08] est un format texte simple et très flexible tiré du SGML[6].

Nous allons prendre un exemple simple pour visualiser graphiquement quelle serait la hiérarchie du document XML correspondant et quelle est la structure du document. L'exemple choisi est celui d'un livre.

```
<livre>
  <titre> les bases de données </titre>
    <chapitre>
        <numero>1</numero>
        <titre>titre du chapitre 1</titre>
        <contenu>le contenu du chapitre 1</contenu>
    </chapitre>
    <chapitre>
    .....
    </chapitre>
</livre>
```

FIGURE 2.1: La représentation d'un livre en xml

XML possède une galaxie de technologies. Parmi celles-ci nous pouvons citer XML Query (langage de requêtes), XSL (eXtensible Stylesheet Language) qui est la définition des présentations de documents XML, XSLT (XSL Transformations)

6. Standard Generalized Markup Language

un langage permettant la transformation de documents XML, XPath (XML Path) un langage de requêtes permettant d'accéder à chaque élément d'un document XML.

2.9.2 SOAP : Simple Object Access Protocol

SOAP [Box00], est un protocole pour l'échange d'information dans un environnement réparti, basé sur le standard XML. Ce protocole consiste en trois parties : une enveloppe qui définit un canevas pour décrire le contenu du message et comment le traiter, un jeu de règles de codage pour exprimer les types de données et une convention pour représenter des appels de procédure éloignés.

SOAP n'est lié à aucun système d'exploitation ni langage de programmation. Il est indépendant du langage d'implémentation des applications client et serveur. En plus, il peut potentiellement être utilisé avec une variété de protocoles (par exemple : HTTP, HTTP Extension Framework).

Modèles d'échange de messages en SOAP

Les messages SOAP sont des transmissions à sens unique d'un expéditeur à un récepteur (Client/Serveur). Lorsqu'une transmission d'un message (invocation d'un service web) commence, le client génère un message SOAP (ou document XML). Ce message est envoyé à partir d'une entité appelée le SOAP sender, localisée dans un SOAP nœud vers le serveur SOAP via le protocole HTTP. Le message peut être transmis à zéro ou plusieurs nœuds intermédiaires (SOAP intermediates) et le processus fini lorsque le message arrive au serveur SOAP qui à son tour génère les réponses appropriées et les envoies vers le client. Le chemin suivi par un message SOAP est nommé message path. La figure suivante montre les éléments qui participent au processus.

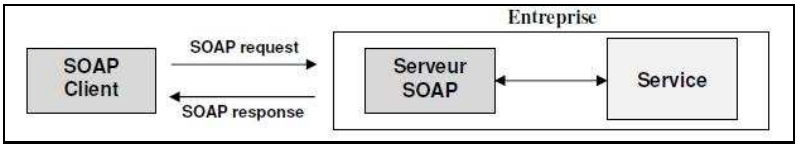


FIGURE 2.2: L'architecture d'un message SOAP [Corporation02]

Quand un message SOAP arrive dans une entité SOAP, il suit le processus décrit ci-dessous :

1. Identifier toutes les parties du message SOAP destiné à cette entité,
2. Vérifier que ces parties sont soutenues par l'entité pour entamer leur traitement. Si ce n'est pas le cas rejeter le message,
3. Si l'entité n'est pas la destination suprême du message, il faut alors enlever toutes les parties identifiées avant l'expédition du message.

Enveloppe SOAP

Un message SOAP est composé d'un élément obligatoire appelé le SOAP enveloppe. L'enveloppe SOAP définit l'espace de nom (namespace : URI [7] permettant de connaître la provenance de chaque balise) précisant la version supportée de SOAP. Cet espace de nom est standard et permet de différencier les éléments du schéma.
L'enveloppe SOAP est constituée d'un en-tête facultatif (SOAP header) et d'un corps obligatoire (SOAP body).

En tête SOAP

L'en-tête peut avoir plusieurs fils (SOAP blocks). Ces fils sont utilisés pour ajouter des fonctionnalités au message comme l'authentification et la gestion des transactions. L'en-tête peut utiliser les attributs mustUnderstand et/ou SOAP actor pour indiquer comment traiter l'entrée et par qui.

7. Uniform Ressource Identifier

L'attribut mustUnderstand peut être utilisé pour indiquer si une entrée d'en-tête est obligatoire ou facultative pour être traitée par le destinataire. Le destinataire d'une entrée d'en-tête est défini par l'attribut d'acteur de SOAP (décrit dans la partie suivante). La valeur de l'attribut de mustUnderstand est « 1 » ou « 0 ». L'absence de cet attribut est sémantiquement équivalente à sa présence avec la valeur « 0 ».
Un message SOAP voyage du SOAP sender au SOAP receiver, en passant par un groupe de SOAP intermédiaires. Un SOAP intermédiaire est une entité qui est capable de recevoir et transmettre les messages SOAP. Les nœuds intermédiaires aussi bien que le SOAP receiver sont identifiés par une URL [8]. L'attribut acteur de SOAP peut être utilisé pour indiquer le destinataire d'un élément d'en-tête. La valeur de l'attribut d'acteur de SOAP est une URL. Par exemple, l'URL « http : // schemas.xmlsoap.org/soap/actor/next » indique que l'élément d'en-tête est destiné à la première entité SOAP qui traite le message.

Corps SOAP

Le corps SOAP contient l'information destinée au receveur. Il doit fournir le nom de la méthode invoquée par une requête, ou le nom de la méthode pour générer la réponse. Il doit aussi, fournir l'espace de nom correspondant au nom du service. Le contenu du corps SOAP est utilisé dans le processus comme le marshalling d'appels RPC [9] et le rapport des erreurs.
Le corps SOAP peut contenir un SOAP fault. Ce bloc est utilisé pour transmettre l'information des erreurs durant le traitement du message.
Malgré que l'en-tête et le corps soient définis comme des éléments indépendants, ils ont une relation : une entrée de corps est sémantiquement équivalente à une entrée d'en-tête destinée pour l'acteur de défaut et avec une valeur d'attribut mustUnderstand de 1.

La structure générale d'un message soap est donnée par la figure 2.3.

8. Uniform Ressource Locator
9. Remote Procedure Call

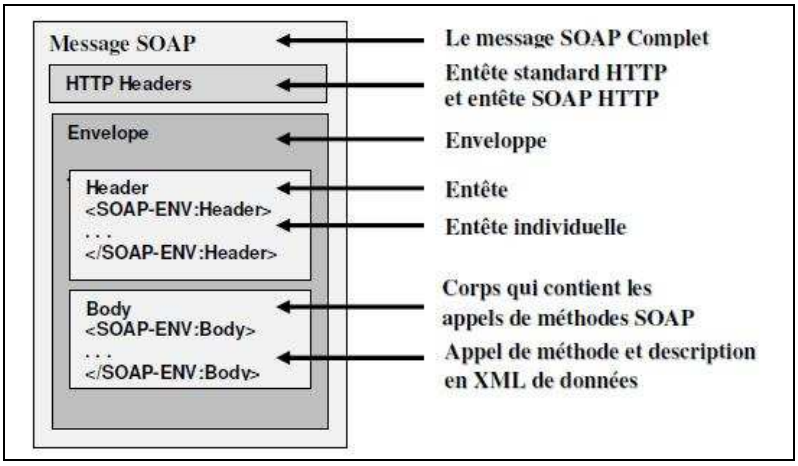


FIGURE 2.3: La structure d'un message SOAP [Papazoglou08]

La description générale de l'enveloppe SOAP et ses composants est donnée par l'extrait de code suivant (voir figure 2.4) :

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

FIGURE 2.4: Définition du type d'enveloppe SOAP [Gudgin03]

L'extrait de code vu ci-dessus, montre un message de notification exprimé en SOAP. Le message contient deux données définies par l'application : Un bloc d'en-tête SOAP avec un nom local *alertcontrol* et un élément corps avec un nom local *alert*. En général, les blocs d'en-tête SOAP contiennent des informations pouvant être utiles à des intermédiaires SOAP aussi bien qu'au destinataire final du message. Dans cet extrait, un intermédiaire peut réordonner la livraison du message

en fonction de la priorité et de la date d'expiration du bloc d'en-tête SOAP. Le corps contient la vraie charge du message, dans le cas présent le message d'alerte.

2.9.3 WSDL : Web Service Description Language

Le WSDL [Chinnici07], est un langage qui permet de décrire les services web, et en particulier, leurs interfaces. Ces descriptions sont des documents XML. WSDL décrit un service web en deux étapes fondamentales : une abstraite et une concrète. Dans chaque étape, la description utilise un nombre de constructions pour favoriser la réutilisation de la description et pour séparer les préoccupations de conception indépendantes (voir figure 2.5).

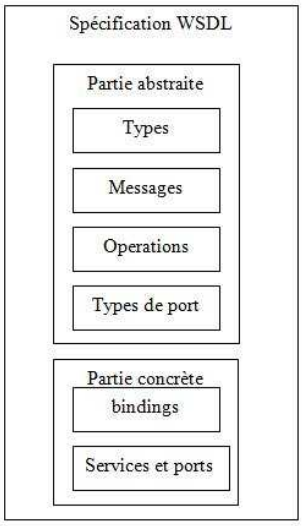


FIGURE 2.5: La spécification d'un Service web avec WSDL [Newcomer03]

Au niveau abstrait, WSDL décrit un service web en termes des messages qu'il envoie et reçoit ; les messages sont décrits de façon indépendante d'un format spécifique en utilisant un système de types, typiquement un schéma XML. La partie abstraite est composée de définitions de « port type » qui sont analogues aux interfaces des « middleware » traditionnels. Chaque « port type » est une collection logique d'opérations. Une « opération » associe un modèle d'échange de message à un ou plusieurs messages. Un message est une unité de communication avec un ser-

vice web. Il représente les données échangées dans une unique transmission logique.

Un modèle d'échange de messages identifie l'ordre et la cardinalité des messages envoyés et/ou reçus. Une interface groupe un ensemble d'opérations.

Au niveau concret, un « binding » indique des détails de format de transport pour une ou plusieurs interfaces. Un « endpoint » (point final) associe une adresse de réseau à un « binding » (attache). Finalement, un service groupe un ensemble d'endpoints qui implémente une interface commune.

2.9.4 UDDI : Universal Description Discovery and Integration

L'objectif primaire d'UDDI [Bellwood02], est de définir les mécanismes permettant de répertorier les services web dans le but de les décrire (Description), de les découvrir (Discovery) et d'y accéder (Integration). Toutes ces opérations s'effectuent à partir de requêtes XML et au moyen du protocole SOAP. L'UDDI travaille avec la notion de « business registry », qui est un service sophistiqué de noms et répertoires. Plus précisément, UDDI définit des structures de données et APIs [10] pour publier les descriptions des services dans le registre et pour interroger ce registre afin de chercher des descriptions publiées. Parce que les APIs d'UDDI sont aussi spécifiés en WSDL avec une attache SOAP, le registre peut être invoqué comme un service web (en conséquence, ses caractéristiques peuvent être décrites dans le même registre, comme un autre service).
Les spécifications du « registry » UDDI ont deux buts principaux en ce qui concerne la découverte d'un service :

1. Soutenir les développeurs dans la découverte d'informations sur les services,
2. permettre la liaison dynamique et en conséquence habiliter les clients pour interroger le registre et obtenir des références aux services d'intérêt.

La structure d'un annuaire UDDI est généralement divisée en trois parties selon la figure ci-dessous :

10. Application Programming Interface

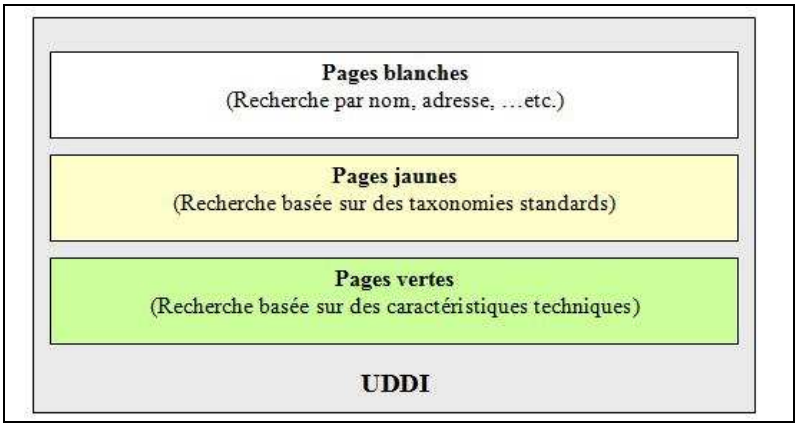


FIGURE 2.6: La structure d'un annuaire UDDI [Papazoglou08]

La figure 2.6 illustre les différentes catégories d'informations présentes dans un annuaire UDDI et elles sont comme suit :

- **Les pages blanches (BusinessEntity) :** elles constituent la fiche signalétique de l'entreprise et contiennent le nom, l'adresse et d'autres informations sur les contacts,
- **Les pages jaunes (BusinessService) :** indiquent la classification de la fiche signalétique en s'appuyant sur des taxonomies standards,
- **Les pages vertes (BindingTemplate) :** fournissent une description technique des différents services offerts par l'entreprise.

Après la définition des différents protocoles et langages utilisés par les services web, la figure 2.7 nous montre les différentes technologies des services web (XML, SOAP, WSDL et UDDI) avec le rôle que joue chaque technologie pour assurer le bon fonctionnement du service web.

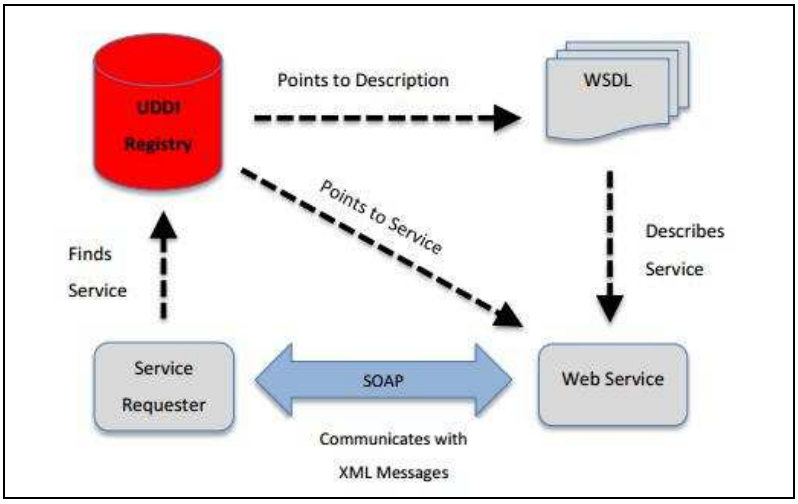


FIGURE 2.7: Les technologies des services web [Papazoglou08]

2.10 Les interactions des services web

En plus des interfaces de services, l'interaction avec un service web oblige l'utilisateur à connaitre des informations supplémentaires pour pouvoir effectuer une interaction fiable, à savoir l'ordre des opérations et le comportement potentiel du service web au cours de ses interactions avec d'autres entités. Ces informations sont appelée « Le modèle d'interaction du service » [Curbera03, Benatallah06a]. Fondamentalement, il existe trois scénarios d'interaction avec un service web :

1. *L'interaction avec le client :* Un client peut communiquer avec un service en échangeant des messages qui doivent suivre un certain ordre afin d'obtenir une communication correcte. Cette séquence de messages est appelée une conversation et le modèle général des conversations constitue un protocole de service web (dans la suite du chapitre on détaillera ce point),
2. *La composition :* elle peut être définie comme le processus de sélection, de combinaison et d'exécution de services en vue d'accomplir un objectif donné [Martin05]. Benatallah et *al,* dans [Benatallah05c] ont considéré la composition de service web comme étant efficace pour créer, exécuter et maintenir des services qui dépendent des autres services web,

3. *L'orchestration :* elle décrit comment les services web peuvent interagir ensemble au niveau des messages en précisant la logique métier et l'ordre d'exécution des interactions. Ces interactions forment par la suite un processus transactionnel étendu qui sera toujours contrôlé par un chef d'orchestre. Le formalisme BPEL [11] [Andrews03] est devenu un standard essentiel pour l'orchestration des services web [Papazoglou04](voir Figure 2.8-a),
4. *La chorégraphie :* est une approche décentralisée pour l'exécution des compositions. Par exemple, les requêtes des utilisateurs pourraient inclure des exigences qui ne peuvent pas être remplies par un seul service, donc il est nécessaire de faire collaborer un certain nombre de services pour répondre aux exigences des utilisateurs [Peltz03]. Dans ce cas, la composition se fait sans le chef d'orchestre, c'est à dire chaque service possède des accords avec le reste des services pour définir la manière de collaborer (voir Figure 2.8-b).

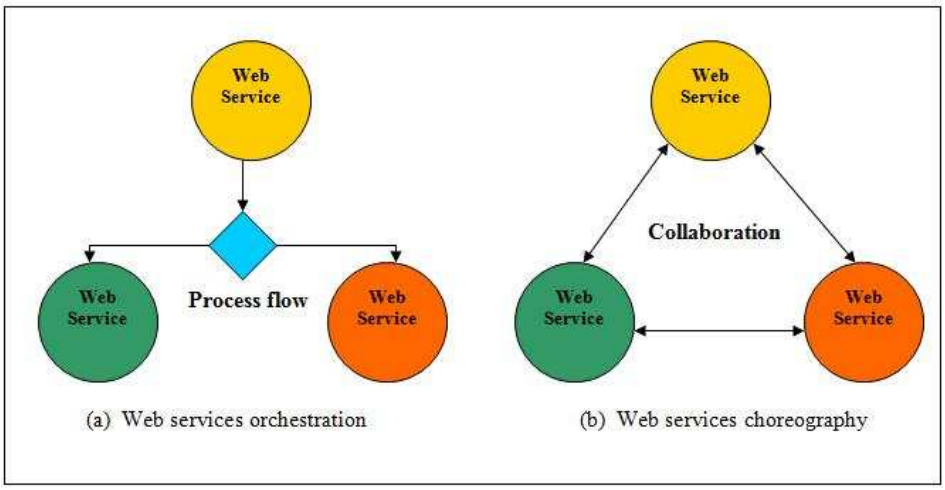


FIGURE 2.8: L'orchestration et la chorégraphie [Peltz03]

2.11 Les processus métiers

Un « processus métier » ou un « processus d'affaire » (Business Process) désigne les activités qui s'appuient sur un ensemble de tâches et du savoir faire d'une

11. Business Process Execution Language

entreprise pour produire une valeur ajoutée aux clients. Le Workflow Management Coalition (WFMC) définit un processus métier comme : « Un ensemble d'une ou plusieurs procédures ou activités liées entre elles pour réaliser collectivement un objectif ou une politique métier en définissant les rôles et les interactions fonctionnelles au sein d'une structure organisationnelle » [Coalition99].
Ces processus sont des processus opérationnels qui décrivent l'ordre d'exécution des activités principales de l'entreprise en transformant les éléments d'entrée en éléments de rendement pour atteindre un objectif métier ou stratégique. Un processus métier est donc considéré comme un ensemble de relations logiques entre un groupe d'activités incluant des interactions entre partenaires sous la forme d'échange de données pour fournir une valeur ajoutée aux clients [Georgakopoulos95, Van der Aalst03, Ryan09]. Dans le contexte des services web d'aujourd'hui, un processus métier spécifie « l'ordre potentiel d'exécution des opérations à partir d'un ensemble de services web, les données partagées entre ces services web dont les partenaires sont impliqués et comment ils sont impliqués dans le processus d'affaire » [Leymann97].

Selon [O'Riordan02, Reza06, Reza08], un processus métier peut être divisé en plusieurs types en respectant certains critères. Par exemple, dans [Reza06] les auteurs ont donné la séparation suivante :

- **Un processus métier publique :** Ce type de processus est utilisé pour effectuer l'interaction entre l'entreprise est ses partenaires c'est à dire dans le contexte *business-to-business integration (B2Bi).*,
- **Un processus métier privé :** Ce type de processus est interne à l'entreprise, il est utilisé pour décrire des tâches internes par exemple dans le contexte de l'intégration d'applications d'entreprise *(EAI).* Mais généralement les deux types sont utilisés ensemble pour effectuer la totalité des opérations de l'entreprise.

2.11.1 La gestion des processus métiers (BPM)

La gestion des processus métiers (Business Process Management) est une approche de gestion destinée à aligner tous les aspects d'une organisation avec les besoins des clients. Dans la littérature, on trouve plusieurs définitions. Van der Aalst et *al* définissent la BPM comme suit : « La gestion des processus métiers consiste à utiliser des méthodes, des techniques et des logiciels pour modéliser, exécuter, contrôler et analyser des processus opérationnels en s'appuyant sur des acteurs qui peuvent être des humains, des organisations, des applications ou d'autres sources d'information » [Van der Aalst03]. Selon ABPMP (The Association of Business Process Management Professionals), la BPM est une approche pour identifier, concevoir, exécuter, surveiller, contrôler et mesurer à la fois des processus métiers automatisés et non automatisés pour atteindre des résultats cohérents avec les objectifs stratégiques des entreprises [CBOK09]. D'une manière générale, la gestion des processus métiers permet de coordonner et gérer les relations inter-entreprises ou intra-entreprises en utilisant des technologies aidant à contrôler et à améliorer l'exécution des processus métiers.

2.11.2 Les systèmes de gestion des processus métiers (BPMS)

Les systèmes de gestion des processus métiers c'est un ensemble de logiciels destinés à modéliser et à exécuter des processus métiers. Généralement, ils se décomposent en plusieurs parties :

- *Un éditeur de modélisation :* il permettra de construire le processus entier avec les tâches et les relations en utilisant le standard BPMN (Business Process Management Notation),
- *Des outils de développement :* il formalisera la logique des processus,
- *Un moteur d'exécution :* il supervisera le déroulement des processus ainsi que les échanges de paramètres,
- *Un moteur de règles :* il évaluera l'état de tous les objets impliqués dans le déroulement des processus et il déterminera si les conditions sont remplies pour poursuivre ou arrêter l'exécution du processus,

– *Des outils d'administration :* ils permettront de régler les paramètres de tout le système et d'obtenir des statistiques concernant l'exécution des processus.

Avec l'arrivée des services web, de nombreuses organisations ont eu tendance à utiliser des systèmes existants pour réaliser et gérer leurs travaux, donc un BPMS doit être aussi en mesure d'intégrer des systèmes existants pour pouvoir contrôler le travail, obtenir de l'information ou mesurer le rendement des processus d'une manière rapide et efficace. Ce point, nous donne déjà une première idée sur le lien qui existe entre les processus métier et les services web.

2.11.3 La relation entre processus métiers et services web)

Depuis la naissance des processus métiers, les entreprises se trouvaient toujours face au problème d'automatiser la conception, la maintenance et la gestion de leurs processus métiers. Cela se fait en réduisant l'intervention humaine dans les tâches de gestion des processus et en utilisant l'intégration des systèmes avec une exécution automatique de la logique métier pour pouvoir assurer une flexibilité à l'intérieur de l'entreprise et pouvoir interagir plus facilement avec les partenaires externes. L'apparition des services web a donné une solution plus appropriée à ce problème, grâce à leurs langages et protocoles de communication, ils assurent l'intégration, l'implémentation des processus métiers et permettent une connexion entre les composants sans être influencés par la plate-forme et l'hétérogénéité des langages de programmation. Cela va réduire les coûts de développement et va accroitre l'agilité métier des entreprises. D'un autre point de vu, on peut aussi utiliser la notion des processus métiers pour pouvoir spécifier les différentes conversations d'un service web avec ses partenaires. Cette spécification connue sous le nom de protocole métier fournit aux développeurs des informations sur la façon de programmer les clients qui peuvent interagir correctement avec un service et permet aussi de vérifier si un service est compatible avec un autre service, ce dernier point sera par la suite le centre de nos recherches.

2.12 Les protocoles de services web

Les protocoles standards de spécification des services web vu précédemment (SOAP et WSDL) jouent un rôle trop important pour faire fonctionner un service web de base, mais il est tout de même nécessaire d'explorer d'autres aspects qui doivent être adressés pour le bon fonctionnement d'un service web tels que : L'aspect sécurité, la gestion des transactions et l'interaction avec d'autres services web (dans le cadre d'étudier la compatibilité entre les services pour une éventuelle composition, une intégration ou une découverte de services).

Ce dernier aspect, s'est avéré crucial dans beaucoup de domaines notamment pour les systèmes d'informations qui gèrent le e-commerce [Berardi04, Castro02, Mecella02, Berardi03c]. En effet, le fichier WSDL expose l'interface des services proposés, mais il omet l'information qu'il lui permet d'interagir avec les autres partenaires. Cette information c'est la séquence correcte des appels de fonctions (méthodes, procédures ou un accès à une base de données).

Pour résoudre ce problème, les chercheurs ont proposé le concept du "business protocol" dans le but d'utiliser un modèle formel pour représenter la séquence correcte de l'envoi et la réception des messages. Dans le prochain chapitre, on va voir les différents modèles formels utilisés pour représenter les services web et plus particulièrement les « protocoles métiers » (en anglais « business protocols »).

2.12.1 La représentation formelle d'un protocole de service web

Un protocole de service est généralement représenté par un quintuplé comme suit : P = (Q, q_0, F, M, Δ) sachant que :

- Q est l'ensemble des états,
- q_0 est l'état initial,
- F $\subseteq$ Q l'ensemble des états finaux et si F = ϕ alors P est considéré comme étant un protocole vide,
- M représente un ensemble fini de messages,

– $\Delta \subseteq Q \times M \times Q$ est la fonction de transition sachant que chaque transition (q, m, q′) spécifie un état source q ∈ Q, un message m ∈ M et état destination q′ ∈ Q.

La figure ci-dessous (Figure 2.9) illustre un exemple concret d'un protocole de service web modélisé par un automate à états finis.

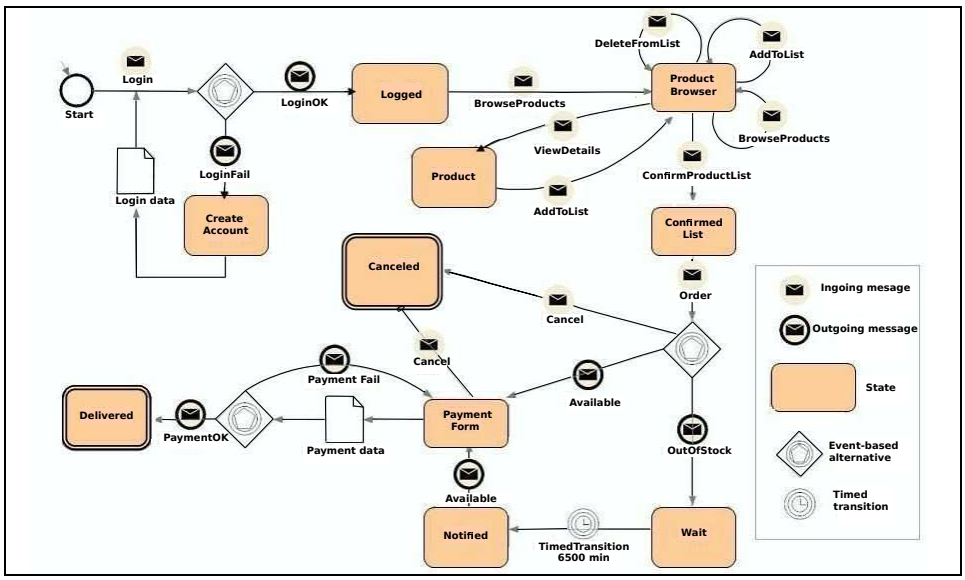


FIGURE 2.9: Un protocole métier pour un achat en ligne [Musaraj10]

Dans cette figure, les états représentent les différentes situations par les quelles le service web client est passé, par contre les transitions reflètent les actions effectuées par le services sous formes de messages entrants et sortants. Par exemple, l'état **Start** modélise le début du protocole qui va envoyé par la suite un message d'authentification *Login*. Si le *Login* est correct, il passe à l'état **Logged** sinon il refait l'opération ou bien il crée un nouveau compte. Une fois qu'il est à l'état **logged**, il peut voir la liste des produits disponibles avec les détails, il peut aussi choisir les produits, cela est modélisé par les transitions *DeleteFromList*, *AddToList*, *View Details*. Une fois que le client confirme sa liste de produits, il passe à l'état **Confirmed List** et la commande est envoyée. dans ce cas il y a plusieurs

alternatives : le client peut annuler sa commande et le protocole passe à l'état **Canceled**, soit il reçoit un message qui indique que le produit est disponible et il passe à l'état **Payment Form** ou bien le produit n'est pas disponible et dans ce cas il passe à l'état **Wait** . Une fois que le payement est effectué soit il reçoit un message *PaymentOk* et il passe à l'état **Delivered** soit il y a une erreur de payement et dans ce cas il reçoit un message *paymentfail*.

Dans la littérature, il existe plusieurs travaux qui utilisent la notion de « protocoles métiers ». Parmi ces travaux, on a ceux de Benattallah et *al* [Benatallah04a, Benatallah04b, Benatallah06a] dans lesquels les auteurs ont réalisé des modèles formels de protocoles métiers, dans le but d'analyser leurs compatibilité et leurs remplaçabilité, ils ont même réalisé un outil logiciel qui implémente ce concept et qui a été présenté dans [Benatallah06b], dans ces travaux les auteurs ont traité uniquement les aspects fonctionnels et comportementaux des services web.

Il existe aussi d'autres catégories de protocoles de services tels que ceux qui prennent en compte l'aspect temporel. Dans ce cadre, on trouve plusieurs travaux de recherche qui étudient cet aspect comme dans les systemes de workflow [De Maria06, Tiplea06] et dans les services web ([Berardi04, Benatallah05a, Benatallah05b, Daiz06, Kazhamiakin06] et [Ponge10]).

Par exemple dans [Ponge10], les auteurs ajoutent aux protocoles de services étudiés dans [Benatallah04a, Benatallah04b] et [Benatallah06a] des contraintes temporelles en utilisant une sous classe appelée ECTA (Event Clock Timed Automata) extraite du travail de Alur et *al* [Alur99] dans le but de rendre le protocole déterminisable, ce qui n'est pas le cas des automates temporisés.

Ce type de protocole est représenté formellement comme suit : P = (Q, q_0, F, M, χ, C, Δ) sachant que :

- Q est l'ensemble des états,
- q_0 est l'état initial,

- F $\subseteq$ Q l'ensemble des états finaux et si F = ϕ alors P est considéré comme étant un protocole vide,
- M représente un ensemble fini de messages,
- Si on suppose que chaque transition $\delta \in \Delta$ est identifiée par une étiquette id $(\delta).\chi = \{T_i | \exists \delta \in \Delta, T_i = id(\delta)\}$ est un ensemble de variables horloges définies sur l'ensemble des transitions Δ,
- C est un ensemble de contraintes temporelles définies sur l'ensemble des variables χ, l'absence de contrainte veut dire qu'elle est toujours évaluée à vrai,
- $\Delta \subseteq Q^2 \times M \times C$ est un ensemble fini de transitions où chaque transition (q, q′, m, c) spécifie un état source q $\in$ Q, un message m $\in$ M et état destination q′ $\in$ Q selon la contrainte c.

Pour voir les choses plus clairement nous proposons dans la figure ci-dessous une modélisation d'un protocole temporisé extrait des travaux de Benatallah et *al* dans [Benatallah05a, Benatallah05b].

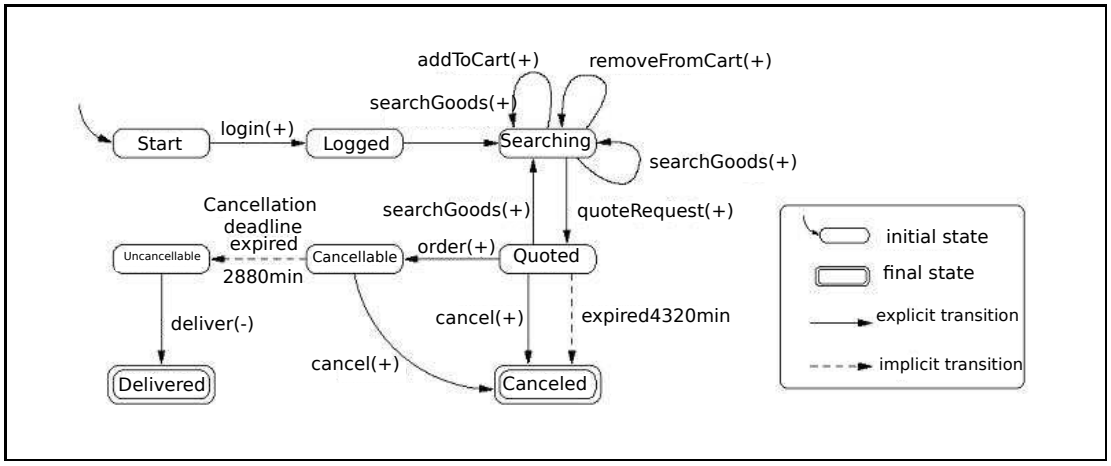


FIGURE 2.10: Un modèle de protocole de service temporisé [Benatallah05a, Benatallah05b]

La figure 2.10 nous montre un protocole métier temporisé décrivant le comportement externe d'un service de gestion des commandes. Ce protocole spécifie que client doit d'abord se connecter. En suite, il peut chercher des produits, les ajouter ou les enlever de son panier et demander un devis qui sera valide uniquement pour 4320 minutes. Pendant cette période si aucun produit n'a été commandé, la commande sera annulée automatiquement.

En plus des aspects cités plus haut, on peut aussi trouver des protocoles de services qui utilisent d'autres aspects qualifiés de non-fonctionnels tels que : la sécurité, la confiance, la qualité de service, la privacité, etc. Mais ces aspects restent les moins soutenus par la communauté de recherche sur les services web. Néanmoins, il existe quelques travaux qui étudient ce genre d'aspects tels que dans [Mokhtari09, Mokhtari12], où les auteurs mettent l'accent sur les protocoles de services et proposent un modèle qui décrit un protocole métier privé (BPPT) exprimant les exigences temporelles et de privacité.

2.13 Conclusion

Dans ce chapitre nous avons traité deux parties. Dans la première partie et en partant du constat du besoin d'une interopérabilité toujours plus grande dans beaucoup de domaines par exemple le e-commerce, nous avons donné un petit historique sur l'évolution des architectures de programmation jusqu'à l'arrivée de la SOA. Dans la deuxième partie de ce chapitre, nous avons traité la relation entre les processus métier et les services web tout en mettant en évidence la spécification des services web par leurs protocoles de services (Business Protocols) qui seront le centre de nos préoccupations dans la suite de ce mémoire. Car en réalité le couplage faible des services web donne une très grande interopérabilité entre eux, c'est à dire qu'un service web n'est pas conçu spécialement pour interagir avec un client spécifique, mais il peut aussi satisfaire les besoins d'autres clients développés par d'autres équipes. En conséquence, un développeur de service web doit connaitre tous les aspects d'un service pour pouvoir développer une application cliente pouvant interagir correctement avec ce dernier. Cela exige une description

des services plus détaillée que la description d'interface conventionnelle des Middlewares avec l'ajout du protocole métier (Business Protocol) supporté par le service, cela donne une spécification des séquences de messages (conversations) échangées ainsi que d'autres informations utiles telles que les transactions, le type de données échangées, les conditions, etc.

“ Attention aux bugs dans le code ci-dessus. Je ne l'ai pas testé, j'ai seulement prouvé qu'il était correct.

Donald Knuth – « Notes dans : the van Emde Boas construction of priority deques : An instructive use of recursion »

3

La modélisation des services web

3.1 Introduction

Les services web interagissent les uns avec les autres par le biais d'émission et de réception de messages. Dans ce cadre, une interaction entre deux services S_1 et S_2 est décrite par un échange de messages entre les deux services qui donne à la fin une sequence de messages appelée conversation ou un protocole de conversation selon Benatallah *et al* dans [Benatallah04a]. Les protocoles de conversation indiquent les séquences d'échanges de messages qui sont permises par le service, exprimés en termes de contraintes sur l'ordre dans lequel les opérations de services devraient être invoquées. De ce fait, la modélisation de services web se focalise principalement sur la description des interactions et de leurs interdépendances en tenant compte de l'aspect structurel (le types de messages échangés) et l'aspect comportemental (en particulier les protocoles de conversation). Dans ce chapitre, nous proposons quelques formalismes utiles pour la modélisation des services web et plus particulièrement les protocoles de conversations dans le but de pouvoir choisir un formalise qui va avec notre processus de vérification.

3.2 La modélisation par réseaux de pétri

Les réseaux de pétri [Petri62] est une approche de modélisation qui représente un nouveau cadre pour la représentation théorique d'un service web. Une des forces de cette approche est la base mathématique forte qu'elle offre avec une représentation graphique qui facilite énormément la modélisation conceptuelle des différents systèmes, elle permet d'utiliser plusieurs opérateurs algébriques tels que : séquence, choix alternatif et arbitraire pour la vérification des processus métiers et aussi les services web [Van der Aalst98, Van der Aalst99]. En effet un réseau de pétri (RDP) est un graphe dirigé biparti connecté dont chaque nœud est soit une transition soit une place. D'une manière générale les réseaux de pétri représentent les processus métiers en conversant les entrées (le nœud du début) dans une place sans arcs entrants et la sortie (le nœud de la fin) dans une place sans arcs sortants. Les conditions sont représentées par des places, et les tâches par des transitions.

Dans un processus métier modélisé par un réseau de pétri, une transition active correspond à un work-item et le tir d'une transition à une instance de l'activité. Un déclencheur pourrait correspondre à une initiative du participant, à un événement externe ou à un signal du temps initié par l'environnement. A chaque transition correspondante à une tâche qui exige un déclencheur, une autre place d'entrée est ajoutée. Une occurrence du déclencheur apporte un jeton dans cette place supplémentaire. Le jeton est consommé une fois la transition appropriée est franchie. Un échec pendant l'exécution d'une tâche exige un rollback (revenir à l'état antérieur au début de l'activité). Quant une activité sera complétée avec succès, des changements deviennent définitifs.

Dans la littérature et après une exploration approfondie de cette approche, nous avons constaté qu'il existe aussi une similitude entre les concepts de services web est les réseaux de pétri notamment dans la modélisation de la composition de services [Hamadi03, Yi04, Zhang04, Feng06, Zhang08], dans ces travaux les auteurs utilisent les réseaux de pétri pour la modélisation du contrôle du flux dans les services web, cette approche fait correspondre une transition à un appel de service et une place à un état entre les appels en introduisant une algèbre pour la composition de services. Les services web sont considérés dans cette approche de modélisation comme un ensemble d'opérations ordonnées et un ensemble d'états. Les opérations sont les transitions et les états constituent les places.

Formellement un réseau de pétri est défini par le tuple SPN (P, T, A, i, o, l) avec :

- P : représente l'ensemble des places,
- T : c'est l'ensemble des transitions représentant les opérations du service,
- A : représente l'ensemble d'arcs tel que : $A \subseteq (P \times T) \cup (T \times P)$,
- i (input) : représente le point d'entrée du service tel que $i = \{ x \in P \cup T / (x,i) \in A \} = \phi$,
- o (output) : représente le point de sortie du service tel que $o = \{ x \in P \cup T / (o,x) \in A \} = \phi$,

– l : T $\longrightarrow$ $O \cup \{t\}$ avec : O comme étant l'ensemble des noms des opérations et si t $\notin$ O alors t est une opération qui ne porte pas de nom.

Un service est défini par le tuple S = (NS, DS, Serv, URl, SPN) tels que :

– NS : est le nom du service,
– DS : est la description du service,
– Serv : est le serveur du service,
– URL : est le moyen d'invocation du service,
– SPN (P, T, A, i, o, l) : est le réseau de pétri représentant la spécification du service.

3.3 La modélisation par les contrats

Dans la littérature, on peut trouver d'autres approches qui permettent de modéliser les services web, on cite par exemple l'approche basée sur les contrats qui sont des règles de transformations de graphes spécifiés par des assertions dans le but d'apporter des interprétations opérationnelles avancées qui ne peuvent pas être exprimées avec des expressions logiques simples. Ces assertions viennent à la base des travaux de Floyd [Floyed67, Floyed93] et de Hoare [Hoare69], elles servent essentiellement dans la spécification des programmes, des objets, des composants et elles peuvent être des pré-conditions, des post-conditions ou des constantes. On distingue deux types de contrat, ceux qui décrivent le comportement nécessaire par le service appelés « les contrats requis » et ceux qui décrivent le comportement offert par le service appelés « les contrats fournis ». La particularité de cette approche, c'est qu'elle utilise aussi les signatures d'opérations et les modèles de données dans le but d'enrichir la spécification du service par l'ajout de l'information comportementale.

En effet, établir et s'assurer de la qualité de service (QdS) d'un composant tel que le service web, représente un enjeu crucial puisque ceci permet d'établir une relation de confiance entre le fournisseur du service et le client. Cependant, en plus des spécifications établies dans le domaine fonctionnel des services web (telles que WSDL,SOAP et UDDI), les contrats de service (SLA : Service Level Agreemnt)

permettent la description et l'établissement de propriétés de QdS . Plusieurs travaux ont utilisé cette approche tels que dans [Keller03, Lamanna03, Song18] où les auteurs utilisent les contrats dans le but de réaliser un accord entre le fournisseur de service et le client portant sur le niveau de QdS que doit fournir le service.

En se basant sur le travail de Tosic et *al* dans [Tosic03] l'auteur a classé les contrats de service selon trois types :

1. **Les contrats fonctionnels :** Ce type de contrat permet de définir ce que fait un service web,
2. **Les contrats de qualité de service :** permettent de différencier entre les services offrant les mêmes fonctionnalités cela en se focalisant sur leurs propriétés non fonctionnelles telles que la fiabilité, la performance, etc.,
3. **les contrats d'infrastructure :** ce type de contrats est généralement utilisé pour décrire les protocoles de communication tel que SOAP.

3.4 La modélisation fondée sur les algèbres de processus

Les algèbres de processus sont des formalismes de description formelle pour la spécification de systèmes logiciels, en particulier pour les systèmes concurrentiels. Elles fournissent des outils pour la description de haut niveau des interactions entre les processus. Plusieurs algèbres de processus ont été décrites. Parmi les plus anciennes, on a l'algèbre *CSP*[1] décrite dans [Hoare78] et *CCS*[2] proposée par Milner dans [Milner82, Milner89]. On peut aussi citer d'autres algèbres qui ont été proposées plus récemment telles que *LOTOS*[3] [Bolognesi87] et π-*calcul* [Milner92] qui sont toutes les deux inspirées de *CCS*.

1. Communicating sequential processes
2. Calculus of Communicating Systems
3. Language Of Temporal Ordering Specification- norme ISO 8807

Les algèbres de processus utilisent des structures simples telles que l'émission ou la réception de messages par un processus dans le but de pouvoir exprimer la mobilité des processus. Cette approche de modélisation est utilisée plus particulièrement dans la composition des services web avec pour principal objectif de vérifier formellement que la composition respecte les propriétés demandées. Par exemple dans [Camara06], l'auteur a proposé une technique de formalisation basée sur *CCS* pour la chorégraphie des services avec *WSCI* . Cette technique se base sur la création des règles pour traduire en *CCS* des chorégraphies de service écrites en *WSCI* pour ensuite vérifier la compatibilité de deux services. Deux services sont considérés compatibles si tous les messages échangés sont mutuellement compris et si leur communication est sans blocage. Dans [Liu06], l'algèbre *CCS* a été utilisée pour la modélisation et la spécification des services web afin de raisonner sur les propriétés comportementales de la composition.

Parmi les avantages des algèbres de processus c'est qu'elles sont plus adaptées pour la description de systèmes larges et complexes, et grâce à leurs niveau d'expressivité, les développeurs sont en mesure de raffiner itérativement les descriptions abstraites des processus. Le seul inconvénient de ce formalisme c'est qu'il utilise des notations textuelles moins lisibles que celles fournies par les réseaux de pétri et les systèmes de transitions.

3.5 Les systèmes de transitions

Un système de transition appelé aussi système de transition d'états (STS) est un modèle de machine abstraite utilisée en informatique théorique pour l'étude du déroulement des calculs. Un tel système comporte un ensemble d'états et des transitions entre ces états. Les états correspondent à des situations particulières par exemple : état *connexion établie* dans un protocole de communication. Les transitions peuvent être étiquetées par des éléments appartenant à un ensemble prédéfini. Les étiquettes peuvent avoir différentes significations :

- **Événements** (ou signaux) : qui, une fois reçus, permettent le franchissement de la transition,
- **Condition** (ou garde) : qui définit une condition c'est à dire le franchissement de la transition ne peut se faire que si la condition est vraie (c'est une condition nécessaire mais pas suffisante),
- **Action** : pour traduire le passage d'un état à un autre.

Différents types de STS ont été considérés par les chercheurs pour la modélisation des protocoles de service web dans l'intention de prendre en charge la vérification. Dans ce qui suit, nous allons présentés ces approches.

3.5.1 Les automates

L'automate est un modèle très connu dans le domaine de la spécification formelle des systèmes informatiques. Il est composé d'un ensemble d'états et d'un ensemble de transitions qui relient les états. Chaque transition et étiquetée par des étiquettes dans le but de modéliser l'exécution de l'action qui permet le passage d'un état à un autre.

Formellement un automate est un *5-uplet* $(Q, \Sigma, q_0, F, \delta)$, tels que :

- Q : représente l'ensemble des états de l'automate,
- Σ : est un ensemble fini de symboles appelés alphabet ou langage reconnu par l'automate,
- q_0 : est l'état initial,
- F : est l'ensemble des états finaux, appelés aussi les états d'acceptation tel que $F \subseteq Q$,
- δ : est la fonction de transition sachant que $\delta : Q \times \Sigma \longrightarrow Q$.

Le principe de fonctionnement d'un automate est très simple, il prend en entrée un mot formé à partir d'un alphabet reconnu par l'automate. Lorsque le mot est lu entièrement, l'automate est arrêté et selon l'état où l'automate est stoppé on dit que l'automate accepte ou refuse le mot, c'est-à-dire si le dernier état est un état

final alors le mot est accepté par l'automate sinon le mot est refusé par l'automate.

A partir de la forme générale d'un automate, on peut modéliser différents types de processus métier et de protocoles de services en jouant sur les étiquettes des transitions. Si l'étiquette est vide cela signifie que la transition se fait d'une manière implicite connue par celui qui a fait la modélisation, si l'étiquette ne contient pas d'événement, cela signifie que le passage d'un état à un autre se fait uniquement par la garde si elle existe et si la garde n'existe pas cela signifie que le passage se fait uniquement par l'action c'est-à-dire, selon la classe de l'automate (automate simple, automate avec échange de messages, automate avec variables, automate avec une base de données, etc.), la forme des étiquettes change.

Plusieurs chercheurs ont proposé d'utiliser la notion d'automates dans le domaine des services web et plus particulièrement dans la composition des services. Parmi ces approches, on peut citer le modèle Roman [Berardi03b, Pla-Castells15] basé sur les automates à états finis (FSA) et le modèle Colombo [Berardi05b] qui se base principalement sur le modèle Roman avec l'ajout de la prise en charge des données et de la communication basée sur l'échange de messages. Une autre approche appelée COCOA a été introduite dans [Ben Mokhtar06] mais dans le domaine des services web sémantiques, son but est de convertir les processus OWL-S [4] en FSA. D'autres approches vont être étudiées plus en détail dans la partie travaux connexes.

3.5.2 La communication entre les automates

La communication entre les automates constitue un cas particulier des automates synchrones ou synchronisés qui est très répandue dans les systèmes informatiques, en particulier les systèmes parallèles ou distribués là ou il y a un échange de données via un réseau ou une mémoire partagée.

4. Web Ontology Language for web Services

3.5.3 Les automates communicants par messages

Dans la communication par messages, les transitions sont marquées par des étiquettes pour décrire l'envoie et la réception des messages. Le message peut être un nom de message avec des paramètres et dans ce cas on parle d'un message de données ou bien un nom d'un signal qui représente un message de contrôle. Par exemple dans le travail de Benatallah et *al* dans [Benatallah04a] les auteurs se focalisent sur la création d'un service client qui peut interagir correctement avec un ou des services données (voir figure 3.1).

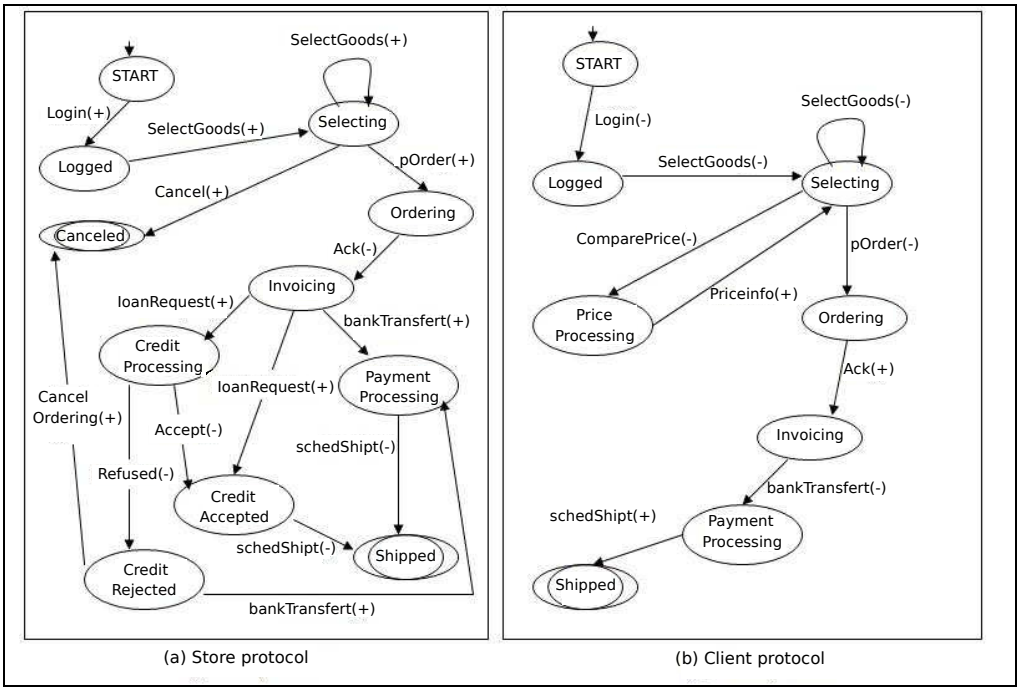


FIGURE 3.1: Communication par message [Benatallah04a]

La figure ci-dessus représente une conversation entre deux protocoles de services modélisés sous forme d'automates, un protocole de service store et un protocole client. La communication entre les deux protocoles se fait par l'envoie et la réception des messages selon une fonction polarité, c'est à dire si la polarité est positive cela veut dire qu'il y a une réception de message et on note message(+), sinon si la polarité est négative alors il y a un envoie de mes-

sage et on note message(-). La communication entre les deux automates se fait selon les traces d'exécution c'est à dire des séquence d'états et de messages. Par exemple la séquence **Start.Loggin(+).Logged.selectGoods (+).Selecting.Porder(+).Ordering** du protocole store est cohérente avec la séquence **Start.Loggin(-).Logged.selectGoods(-).Selecting.Porder(-). Ordering** du protocole client.

3.5.4 Les automates communicants par variables partagées

Dans le cas d'une communication entre automates par des variables partagées, la variable peut être utilisée par plusieurs automates. Les valeurs de cette variable permettent une certaine synchronisation entre les automates utilisant cette variable. Par exemple, un automate peut modifier la valeur de la variable et d'autres automates peuvent la lire tout en protégeant l'accès à cette variable partagée à l'aide de message de synchronisation (Exclusion mutuelle). Ce type de communication est utilisé généralement dans la vérification des réseaux informatiques et dans la spécification des systèmes en temps réel [Okano99].

3.5.5 Les automates communicants par une file FIFO

La file FIFO est généralement utilisée pour une communication asynchrone entre les automates. En termes d'automates, la file d'attente est modélisée par un automate qui par exemple gère une variable de type file d'attente. Parmi les travaux qui utilisent ce type de communication, on a le travail de Vardhan et *al* [Vardhan05] dans lequel les auteurs vérifient la sécurité des propriétés des machines à état finis qui communiquent sur des canaux FIFO illimités.

3.5.6 Les automates temporisés

Contrairement aux automates classiques c'est-à-dire les automates modélisant des systèmes à événements discrets, il existe dans la littérature d'autres formalismes utilisés pour la modélisation des systèmes en temps réel tels que les réseaux de Pétri temporisés (timed Petri Nets), les algèbres de processus

temporisés (timed process algebras) et les logiques en temps réel (real time logics)[Berthomieu91, Reed88, Nicollin94, Chaochen99]. L'origine de cette tendance était les travaux d'Alur et Dill dans [Alur90, Alur94] qui ont proposé une nouvelle classe d'automates appelée la classe des automates temporisés (timed automata). Un automate temporisé est un automate doté d'une ou de plusieurs horloges qui comptent le temps qui s'écoule, ces horloges peuvent être remises à zéro indépendamment les unes des autres. Le tir des transitions tient compte des valeurs des horloges. Dans cette classe d'automates, on fait appel au temps qui peut être discret ou continu. Lorsqu'on utilise le temps discret, les valeurs sont des entiers positives et la modélisation se fait à l'aide de temps discret. Dans le temps continu, on associe aux variables représentant le temps des nombres réels positifs et dans ce cas l'espace des valeurs d'horloge devient infini.

La figure suivante montre un exemple d'automate temporisé. Le comportement en temps réel de l'automate est contrôlé par deux horloges x et y. L'horloge x est utilisée pour contrôler la boucle. La transition de la boucle peut se produire lorsque x=1. L'horloge y contrôle l'exécution de l'ensemble de l'automate. L'automate peut commencer l'exécution à n'importe quel moment, si la valeur de y est dans l'intervalle [10,20], l'automate peut aller à la boucle et lorsque y est compris entre 40 et 50, alors il peut aller à end.

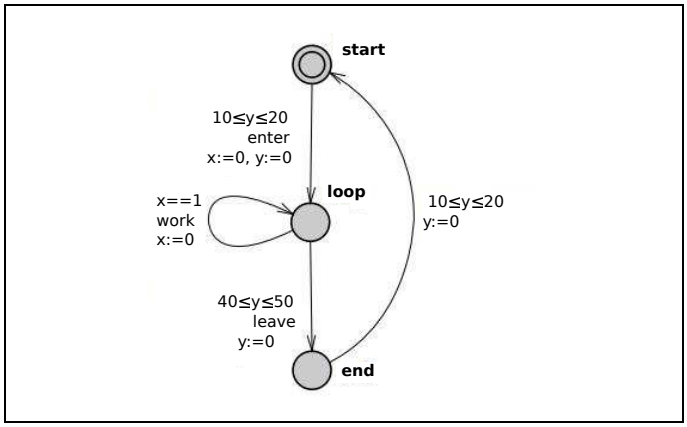


FIGURE 3.2: Un automate temporisé

3.5.7 Les automates hiérarchiques

La classe des automates hiérarchiques a été définie par Harel dans son travail sur les *Statecharts* [Harel87] qui sont caractérisés par des diagrammes d'états hiérarchiques, des états composés et une communication de type diffusion. Cette classe d'automate facilite la modélisation des systèmes larges en termes d'états et de transitions car elle dispose d'opérateurs de structuration qui limitent l'explosion combinatoire horizontale des états et des transitions cela en rajoutant des *super-états* qui englobent plusieurs autres états. Cette imbrication des états permet plusieurs niveaux d'abstraction et facilite donc la lisibilité du modèle. Toute transition sortante d'un *super-état* vers un autre état équivaut à une transition sortante de chacun des états du *super-état* vers cet autre état. Toute transition entrante vers un *super-état* et une transition qui va vers un des états de ce *super-état*. De nombreux travaux de recherche sont toujours en cours sur la sémantique de ce type d'automates, en particulier des travaux sur les problèmes de choix du modèle de communication en se basant sur la liaison (globale ou locale) et le mode (synchrone ou asynchrone). Dans [Mikk98] et [Dong01] les auteurs ont défini un autre type d'automate hiérarchique appelé automate hiérarchique étendu **EHA** (Extended Hierarchical Automata) qui est composé d'un ensemble d'automates séquentiels et ses états peuvent être représentés par un ensemble d'automates. Formellement un automate séquentiel A est un quadruplet (Q_A, q_{0A}, λ_A, δ_A) avec :

- Q_A : un ensemble fini d'états,
- q_{0A} : l'état initial,
- λ_A : un ensemble fini d'étiquettes,
- $\delta_A \subseteq Q_A \times \lambda_A \times Q_A$: est la fonction de transition.

Un **EHA** H est un quintuplet (F, E, ρ, A_0, V) avec :

- F : est un ensemble fini d'automate séquentiel, $\forall$ A_1, $A_2 \in$ F, $Q_{A_1} \cap Q_{A_2} = \phi$,
- E : est un ensemble fini d'événements,
- V : est un ensemble fini de variables,
- ρ : est une fonction de raffinement tel que $\rho : \bigcup_{A \in F} Q_A \longrightarrow 2^F$ qui impose trois conditions :

1. Un unique automate racine $A_0 \in F$ et il n'existe pas un état $s \in \bigcup_{A \in F} Q_A$ tel que $A_0 \in \rho(s)$,
2. Hormis l'automate racine, chaque automate a exactement un ancêtre,
3. Il n'y a pas de cycle.

– A_0 : est l'automate racine.

La figure 3.3 montre un exemple d'automate hiérarchique correspondant au fonctionnement d'un téléviseur (TV-set), cet exemple est inspiré par [Huizing91] et utilisé aussi dans [Mikk97]. L'automate racine *TV* peut être dans l'état ON ou OFF. L'état OFF est raffiné par un automate *Power* et l'état ON par une composition parallèle de deux automates *Image* et *sound*. A chaque fois que l'automate *TV* est dans l'état ON, les automates *Image* et *sound* sont actifs. Le système change d'état des-qu'il reçoit un événement venant du monde extérieur, par exemple l'événement *on* initie un changement d'état de OFF à ON, cela conduit à une désactivation de l'automate *Power* et l'activation des deux autres automates *Image* et *sound*. Un téléviseur possède aussi un état déconnecté qui signifie que le téléviseur n'est pas branché et seulement après l'événement *in* qu'on aura une certaine activité du système, la même chose avec les deux automates *Image* et *sound* on peut aller à l'état VIDEOTXT ou SHOW selon l'arrivée de l'événement *txt* et à l'état MUTE ou ON selon l'arrivée des événements *mute* et *sound*.

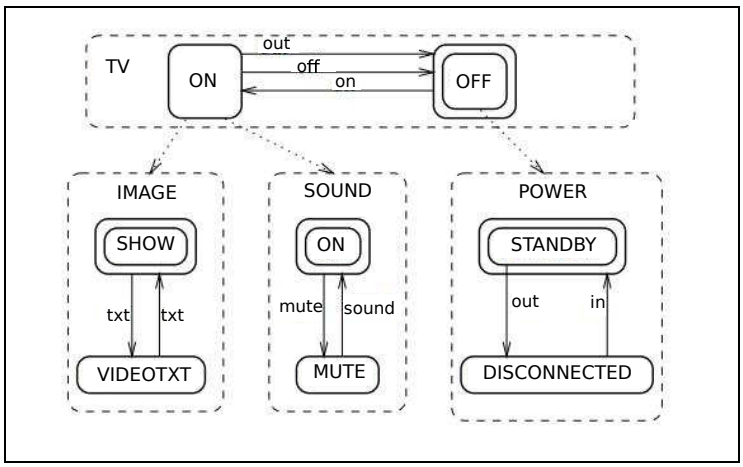


FIGURE 3.3: L'automate hiérarchique téléviseur [Mikk97]

3.6 Quelques standards de conversations

Les protocoles de conversations entre les services web définis dans le travail de [Benatallah04a] et exprimés selon l'ordre des opérations invoquées, peuvent être aussi modélisés à l'aide de standards tels que WSCL et BPEL ou tout autre langage de protocoles. Dans la suite de cette partie, on va vous présenter une vue globale sur ce type de standards.

3.6.1 Le standard WSCL

Contrairement au standard WSDL définissant la description de service, le WSCL (Web Services Conversation Language) est une proposition de vocabulaire XML représentant et simplifiant les interactions ou la relation entre les services web sans connaitre la manière avec laquelle les messages échangés ont été crées. Ce standard a été proposé en 2001 par l'entreprise Hewlett- Packard dans le cadre du développement de la plate-forme E-speak qui est une plate-forme de développement et de déploiement de services web rebaptisée par la suite sous le nom de HP Web services Platform. Un document WSCL est constitué de trois blocs importants :

- Les schémas XML des documents échangés durant l'interaction entre les services,
- La description des transactions qui permettent le passage d'une interaction à l'autre,
- La description des interactions qui peuvent être sous l'une des formes suivantes :
    1. Une émission de message puis une réception,
    2. Une réception de message puis une émission.

La figure 3.4 montre un diagramme UML d'une interaction entre services web (Acheteur-Vendeur) qui va être représenté en WSCL dans la figure 3.5. Cet exemple est extrait du livre de Chauvet dans [Chauvet02].

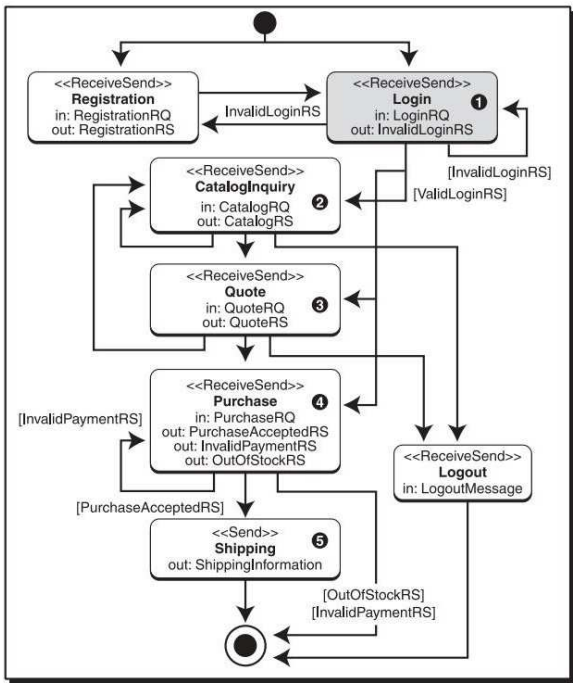


FIGURE 3.4: Diagramme UML d'une interaction Acheteur-Vendeur [Chauvet02]

Dans cette figure, on peut distinguer cinq phases :

1. L'acheteur se connecte au site du vendeur,
2. Le site reçoit des requêtes de navigation et de sélection dans le catalogue des produits,
3. Le site répond à une demande de devis sur les produits sélectionnés,
4. Le site reçoit le bon de commande, vérifie les moyens de paiement, puis accepte ou rejette le bon,
5. Les produits sont envoyés à l'acheteur.

achatVente.wscl

```
<?xml version="1.0" encoding="UTF-8"?>
<Conversation name="StoreFrontServiceConversation"
 initialInteraction="Start"  finalInteraction="End" >

  <ConversationInteractions>

    <Interaction interactionType="ReceiveSend" id="Login">
       <InboundXMLDocument id="LoginRQ"
          hrefSchema="http://conv123.org/LoginRQ.xsd"  />
       <OutboundXMLDocument id="ValidLoginRS"
          hrefSchema="http://conv123.org/ValidLoginRS.xsd" />
       <OutboundXMLDocument id="InvalidLoginRS"
          hrefSchema="http://conv123.org/InvalidLoginRS.xsd" />
    </Interaction>

    <Interaction …  />
            …
  </ConversationInteractions>

  <ConversationTransitions>
    <Transition …  />
    <Transition …  />
            …
  </ConversationTransitions>
</Conversation>
```

FIGURE 3.5: La représentation en WSCL de l'interaction Acheteur-Vendeur [Chauvet02]

La figure ci-dessus montre un exemple d'un fichier WSCL qui représente l'interaction entre l'acheteur et le vendeur selon le diagramme représenté dans la figure 3.4. On remarque que le fichier est structuré sous forme de balises XML, il contient aussi la description des interactions et des transactions entre les services. Ce document WSCL peut être enregistré dans un annuaire UDDI grâce au *TModel* d'UDDI et son nouveau type de taxonomie uddi-org :wsclspec qui est similaire à celui de WSDL (uddi-org :wsdlspec).

3.6.2 Le standard BPEL

BPEL (Business Process Execution Language) est un standard en matière de composition de services web basé sur le langage XML. Ce langage est né suite à une fusion de deux langages WSFL [5] développé par IBM qui est une extension de

5. Web Service Flow Language

FL [6] et le langage XLANG [7] de Microsoft qui est une extension de WSDL. La collaboration de ces deux entreprises a abouti à la création de la norme BPEL4WS1.1 puis la norme WSBPEL2.0.

Les premiers objectifs de BPEL étaient de :
- Supprimer les conflits liés à la composition de services et à l'intégration de services dans des processus métiers,
- Définir un langage standard et portable de spécification des processus métiers, notamment pour la composition de services web.

La figure 3.6 montre la pile des protocoles utilisés dans BPEL4WS avec le langage BPEL qui est situé juste au dessus du langage WSDL car il l'utilise pour spécifier les actions qui doivent s'effectuer au sein du processus et pour décrire les services web offerts par ce dernier.

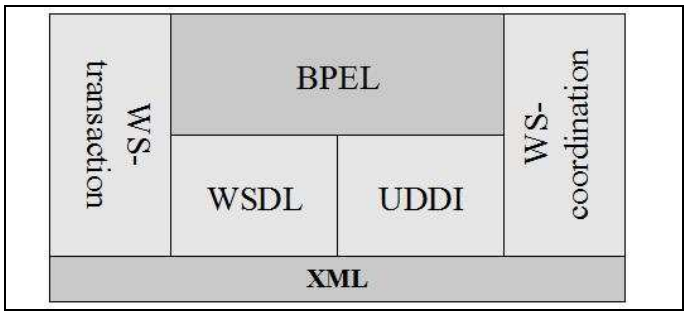


FIGURE 3.6: La pile des protocoles de BPEL4WS [Benmerzoug09]

BPEL fait également appel à SOAP pour définir la manière de structurer les messages échangés par les services et enfin il utilise UDDI pour se connecter aux annuaires qui référencent l'ensemble des services web existants. Tous ces composant se présentent sous la forme d'un fichier XML décrivant les rôles impliqués dans l'échange de messages, les aspects du processus, les types de données structurées,

6. Flow language
7. Xml LANGuage

un modèle de choix de service et un mécanisme pour manipuler les exceptions, les événements et la compensation.

On trouve aussi un composant WS-coodination qui décrit l'interaction des différents services web associés à une tâche donnée et le WS-transaction qui gère le déroulement des tâches et garantit que toutes les transactions aboutissent ou échouent en groupe.

Les éléments d'un processus BPEL sont : les liens de partenaires (partnerLink), les activités et les variables.

3.6.3 Les liens de partenaires

Un lien de partenaire représente la relation de conversation entre deux services partenaires. Chaque lien de partenaire est typé par un *partnerlinkType* qui est chargé de définir le rôle que joue chacun des deux partenaires dans une conversation.

3.6.4 Les activités

Chaque processus BPEL est constitué d'activités basiques ou structurées liées par un flot de contrôle.

Les activités basiques sont :
- <*invoke*> : pour invoquer une opération dans un service web,
- <*reply*> : pour répondre à une source externe,
- <*receive*> : pour recevoir un message d'une source externe,
- <*assign*> : pour copier les données,
- <*wait*> : pour attendre un certain temps,
- <*terminate*> : pour terminer l'instance de service,
- <*empty*> : ne fait rien mais elle est utile pour la synchronisation des activités parallèles,

– <*throw*> : pour lancer une erreur d'exécution.

Les types d'activités structurées sont :

– <*sequence*> : pour définir un ordre d'exécution,
– <*while*> : pour les boucles,
– <*switch*> : pour le traitement conditionnel,
– <*flow*> : pour l'acheminement parallèle,
– <*scope*> : pour invoquer les activités de compensation par le gestionnaire d'erreur,
– <*pick*> : pour attendre l'arrivée d'un événement.

3.6.5 Les variables

Chaque processus BPEL a un état qui est maintenu par des variables contenant des données afin de contrôler le comportement du processus, ces variables sont généralement utilisées dans les affectations et les expressions permettant d'ajouter des conditions de transition ou de jointure au flot de contrôle.

Avec les éléments vus précédemment, BPEL permet de modéliser deux types de processus :

– **Le processus abstrait :** spécifie les échanges de messages entre les différentes parties sans spécifier le comportement externe de chacune d'elles.
– **Le processus exécutable :** spécifie l'ordre d'exécution des activités constituant le processus, des partenaires impliqués dans le processus, les messages échangés entre ces partenaires et le traitement d'exceptions.

L'apprentissage de BPEL et la réalisation d'un client sont d'une difficulté trop importante car c'est un langage à balises donc il a un bas niveau d'expressivité. Les difficultés liées à son apprentissage ont donné naissance à un nouveau langage appelé BPELJ [Blow04]. Ce langage est venu pour améliorer les insuffisances de BPEL en termes d'expressivité et d'extensibilité, car avec BPELJ on a la possibilité

de définir nos propres classes java pour exprimer des données qui n'existent pas dans la sémantique de BPEL, utiliser des structures de programmation telles que les boucles, l'intégration de nouvelles fonctionnalités qui seront décrites comme des objets java, ainsi que l'utilisation de l'alternative à branches multiples.

3.7 Discussion sur les formalismes de modélisation des services web

Dans ce chapitre, nous avons présenté quelques formalismes de modélisation des services web et plus particulièrement les formalismes utilisés par les chercheurs pour modéliser les processus métiers spécifiant le comportement des services web appelés aussi les protocoles de services. Cependant, nous avons constaté que la modélisation en utilisant l'algèbre des processus est beaucoup plus utilisée dans le cas des systèmes complexes (distribués et concurrentiels). De ce fait, plusieurs travaux de recherche ont utilisé ce formalisme et surtout dans le domaine de la composition des services, mais la notation textuelle de ce dernier le rend moins lisible que d'autres.

Un deuxième formalisme a été présenté et qui représente un moyen très efficace pour la modélisation des systèmes dynamiques à événements discrets, se formalisme s'appelle les réseaux de pétri. Parmi les avantages des réseaux de pétri (RDPs) est la façon naturelle dans laquelle les aspects mathématiques et conceptuels des systèmes concurrentiels sont identifiés. Les réseaux de pétri sont aussi considérés comme un excellent outil de simulation et permettent d'offrir de nombreux outils d'analyse structurelle comme la théorie des réductions de réseaux qui ramène l'étude des propriétés à un réseau de taille plus petite ou le calcul algébrique d'invariants qui permet de générer des flots et des semi-flots. Néanmoins, ce formalisme reste toujours un moyen efficace pour modéliser les systèmes à événements.

Le troisième formalisme vu dans ce chapitre est le formalisme de modélisation

par contrats. Nous avons constaté que ce formalisme est spécifique à la description et l'établissement de propriétés qui assurent principalement la qualité de service.

Finalement, nous avons présenté le formalisme basé sur les systèmes de transitions et plus particulièrement les automates qui constituent un formalisme de base pour la représentation du comportement d'un système donné. Les automates sont adaptés à la modélisation des systèmes à événements discrets. Les automates peuvent être une représentation finie d'un ensemble de comportements infinis d'un système, c'est pour cette raison qu'ils sont très utilisés dans le domaine de la modélisation des artéfacts (processus orienté données). Les automates permettent aussi de formuler certains problèmes et de les résoudre de façon algorithmique, donc ils apportent un atout dans le domaine de la vérification formelle. Il existe de nombreuses variantes de la notion d'automate de telle sorte que chaque variante est adaptée à résoudre des problèmes particuliers. Leur coté graphique les rend faciles à comprendre. Leur principal défaut est l'explosion combinatoire de nombre d'états et de transitions en fonction de la complexité et la nature du système à modéliser. Ils sont utilisés pour vérifier des propriétés sur les systèmes modélisés. Cependant, pour vérifier des propriétés, il faut les écrire dans une logique appropriée telles que LTL [8], CTL [9], PLTL [10], etc., avant d'utiliser les outils de vérification. Ils sont aussi largement utilisés dans le domaines industriel.

Après une recherche approfondie dans la littérature sur les différents formalismes de modélisation des processus métiers et plus particulièrement les services web, on a pu faire une petite synthèse qui sera présentée par le tableau ci-dessous.

Dans le prochain chapitre, on va présenter un état de l'art sur la vérification formelle en se focalisant principalement sur la vérification des systèmes qui utilisent des processus orientées données, on peut citer par exemple les systèmes utilisant des services web orientés données, car ce type de système utilise des base de données ce qui les rend infinis.

8. Linear Timed Logic
9. Computation Tree Logic
10. Propositional Linear-time Temporal Logic

TABLE 3.1: Les formalismes de modélisation des processus métiers

Les formalismes de modélisation	**Utilisés pour**	**Leurs difficulté de représentation**
L'algèbre des processus	La modélisation des systèmes complexes (distribués et concurrentiels)	Moins lisibles
Les réseaux de petri	La modélisation des systèmes dynamiques à événements discrets	Plus lisibles que l'algèbre des processus
Les contrats	La description des propriétés qui assurent la qualité de services	Plus lisibles que l'algèbre des processus
les systèmes de transitions (les automates)	La modélisation des systèmes à événements discrets, la représentation finie d'un ensemble de comportements infinis, etc.	Lisibles

“ Trois critères permettent de considérer une affirmation comme valide : la vérification par l'expérience directe, la déduction irréfutable, et le témoignage digne de confiance.

Jean François Ricard, 1997 – « Le Moine et le philosophe »

4

La vérification formelle des systèmes informatiques

4.1 Introduction

Ces dernières décennies, les systèmes informatiques ont connu un progrès technologique considérable, car ils contrôlent de plus en plus de tâches dans différents domaines et sont souvent composés de plusieurs éléments pouvant s'exécuter en parallèle, communiquer entre eux ou interagir avec des événements internes ou externes tels que les protocoles de communication, les systèmes embarqués, les systèmes concurrentiels, etc. Donc une caractéristique importante de ces systèmes est leur propension à avoir un très grand nombre de comportements possibles. Cependant, il est évident qu'ils doivent satisfaire deux propriétés importantes :

- *La sûreté (safety)* : Le fonctionnement ne doit pas conduire à des situations catastrophiques ou dangereuses tel que le non-blocage (Safety : bad things do not happen on all the execution of a system),
- *la vivacité (liveness)* : Le fonctionnement du système doit toujours conduire à de bonnes situations, par exemple la terminaison d'un programme est souvent considérée comme une propriété de vivacité (good things eventually happen on all the execution of a system).

Donc pour être fiables et sûrs, les systèmes doivent être validés par des méthodes de validation telles que : la simulation, la vérification formelle basées sur des méthodes déductives, des tests ou bien des méthodes basées sur le model-cheking.

Dans ce chapitre, on va présenter un état de l'art sur l'étude des différentes méthodes de vérification tout en mettant le point sur la vérification classique des systèmes infinis dans lesquels les états finis sont remplacés par une infinité d'instances de base de données. On va aussi aborder le problème de l'équivalence et de la compatibilité des processus orientés données, car ces deux propriétés jouent un rôle trop important surtout s'il s'agit de systèmes adoptant une architecture orientée services et qui nécessitent à un moment donné l'intégration de nouveaux services ou la remplaçabilité d'un service défaillant par un autre service équivalent qui est compatible avec les autres services.

4.2 La vérification formelle

Dans la littérature, il existe essentiellement deux approches de vérification formelle qui ont été étudiées intensivement : une approche basée sur les modèles (*model-checking*) et une autre approche basée sur la preuve des théorèmes (*theorem-proving*). Les méthodes de vérification basées modèles sont généralement restreintes à des systèmes ayant un nombre fini d'états, cependant le système à vérifier doit d'abord être décrit dans un langage de spécification formel de haut niveau dont la sémantique décrit tout le comportement du système à spécifier. Une fois que le système a été transformé sous forme d'un modèle sous-jacent qui décrit tous les comportements possibles du système, on peut vérifier la satisfaisabilité des propriétés sur le systèmes à l'aide des évaluateurs model checking. Contrairement aux approches model checking, les approches basées sur la preuve de théorèmes permettent de traiter généralement des systèmes ayant un nombre infini d'états. Dans le cadre de notre travail, nous avons choisi d'utiliser la technique de vérification basée sur la preuve de théorèmes car les processus orientés données interrogent des bases de données d'où le problème d'infinité de données qui donne un nombre infini d'états.

4.3 La vérification des propriétés sur les automates

Comme notre formalisme de modélisation se base principalement sur les automates, il est nécessaire de définir les propriétés élémentaires à vérifier sur les automates qui sont les suivantes :

- *L'équité :* Dans le cas des automates non déterministes, l'équité indique que si deux chemins sont possibles à partir d'un état quelconque, alors ce n'est pas toujours le même chemin qui est emprunté,
- *Le blocage :* Le blocage est la situation dans laquelle l'automate se bloque c'est à dire si on est dans n'importe quel état non final et depuis cet état, il n'existe aucune transition qui mène vers un autre état,
- *L'accessibilité :* Un état est accessible s'il est atteignable depuis l'état initial,

- *La ré-initialisabilité :* Un automate est réinitialisable s'il existe un chemin depuis chaque état vers l'état initial,
- *La boucle infinie :* C'est la situation où l'automate boucle d'une manière infinie sur une partie, ce qui empêche tout accès à d'autres parties de l'automate.

Une fois que l'élaboration du modèle sous forme d'automate a été faite, on peut analyser ou vérifier des propriétés dépendantes des abstractions faites au niveau du modèle. Dans la littérature, il existe plusieurs méthodes de vérification. Le plus souvent sont des raisonnements basés sur la vérification des invariants et des propriétés. Par exemple, des propriétés exprimées dans une logique appropriée telles que la LTL (Linear Timed Logic) ou bien la TCTL (Timed Computation Tree Logic) et des invariants qui sont vrais dans tous les états accessibles d'un modèle. On peut aussi trouver des méthodes de vérification qui se basent principalement sur la comparaison entre deux modèles, c'est à dire étudier l'équivalence ou la compatibilité entre deux modèles. Comme notre but est d'étudier la compatibilité entre les protocoles de services orientés données, dans la suite de ce chapitre, on va voir les différentes méthodes de vérification entre deux modèles tout en mettant le point sur les systèmes de transitions, puis on va présenter quelques travaux qui étudient la vérification des processus métiers orientés données et qui donnent des solutions pour remédier au problème d'infinité de données.

4.3.1 La Bisimulation

La bisimulation est la manière la plus simple pour définir l'équivalence entre deux systèmes de transitions. On dit qu'un système est équivalent à un autre système s'il y a une correspondance entre les états et les actions exécutées par les deux systèmes. Il existe deux types de bisimulation :

4.3.1.1 La bisimulation forte

Définition 1.

La bisimulation forte (*Strong bisimulation*) entre deux systèmes de transitions $S = (Q, q_0, A, \delta)$ et $S' = (Q', q'_0, A', \delta')$ est une relation binaire R entre les deux systèmes sachant que :

- Q et Q′ sont les ensembles des états des deux systèmes,
- q_0 et q'_0 sont les états initiaux,
- A et A' représentent les ensembles des actions sachant que $A = A' = \{a\}$,
- δ et δ' les fonctions de transitions.
- R est la relation binaire entre S et S' sachant que :
  - $R \subseteq S \times S'$, elle est définie par un ensemble de couples (q, q′) de telle manière que le choix d'association d'un état d'un système à un état d'un autre système dépend de l'analyse qu'on veut faire sur les deux systèmes, c'est à dire si (q, q′) appartient à R, alors on dit que q′ simule q,
  - Pour tout état $q \in Q$, il existe un état $q' \in Q'$ tel que q R q′,
  - Pour tout état $q' \in Q'$, il existe un état $q \in Q$ tel que q′ R q,
  - Pour toute transition $q_1 \xrightarrow{a} q_2 \in \delta$ et pour tout état $q'_1 \in Q'$ tel que q_1 R q'_1 il existe une transition $q'_1 \xrightarrow{a} q'_2 \in \delta'$ tel que q_2 R q'_2,
  - Pour toute transition $q'_1 \xrightarrow{a} q'_2 \in \delta'$ et pour tout état $q_1 \in Q$ tel que q_1 R q'_1, il existe une transition $q_1 \xrightarrow{a} q_2 \in \delta$ tel que q_2 R q'_2.

4.3.1.2 La bisimulation faible

La bisimulation faible (*Weak bisimulation*) est définie de la même manière que la bisimulation forte mais elle prend en compte des séquences d'exécution à la place des actions, c'est à dire elle remplace les transitions de type $q_1 \xrightarrow{a} q_2$ par des séquences d'exécution de type $q_1 \xrightarrow{\sigma} q_2$ tel que σ représente le chemin entre les deux états.

4.3.2 L'équivalence de traces

Définition 1.

Une séquence d'actions $\langle a_1, ..., a_n \rangle$ est l'observation des actions $a_1, ..., a_n$ effectuées par un processus donné selon le même ordre.

Définition 2.

Une trace d'un système S notée $\mathrm{Tr}(S)$ est l'ensemble des séquences observables sur S. Formellement, si S = (Q, A) sachant que :

- Q : représente l'ensemble des états,
- A : représente l'ensemble d'actions.

Alors $\mathrm{Tr}(S) \stackrel{def}{=} \{\langle a_1, ..., a_n \rangle | \exists q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} ... \xrightarrow{a_n} q_n \in A\}$.

Selon les définitions 1 et 2, on peut dire que l'évolution d'un système peut être représentée par la trace du chemin issu de l'état initial ou un autre état. On peut alors analyser les traces et voir le comportement d'un système par rapport à un autre. En particulier, on peut chercher si les traces sont identiques, compatibles et si l'une est incluse dans l'autre, etc.

Définition 3.

Soient deux systèmes S et S', on dit que les deux systèmes sont fortement trace-équivalents si et seulement si $\mathrm{Tr}(S) = \mathrm{Tr}(S')$.

Exemple :

Soient deux systèmes de transitions S et S' décrits par la figure ci-dessous. On remarque que les deux états q_0 et q'_0 sont trace-équivalents car ils génèrent des langages égaux $\mathrm{L}^1(\mathrm{q}_0) = \mathrm{L}^2(\mathrm{q}'_0) = \{ab, ac\}$.

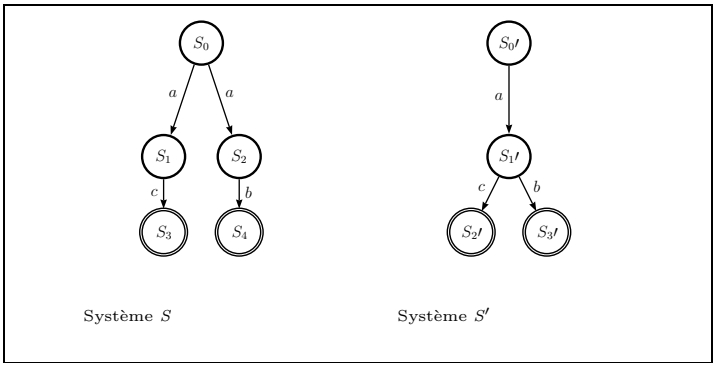


FIGURE 4.1: L'équivalence de traces

4.3.3 La compatibilité

Comme notre travail se base principalement sur la décidabilité de la compatibilité des protocoles de services web, on peut définir la compatibilité par rapport aux services web comme suit : « que deux ou plusieurs services sont dits compatibles s'ils peuvent communiquer et collaborer ensemble d'une manière correcte ». Cette communication (appelée aussi conversation) se fait par le biais d'échange de messages. Une conversation durant laquelle aucun blocage n'apparait est dite correcte et dans ce cas, on peut dire que les services sont compatibles.

La vérification de la compatibilité consiste à vérifier si deux services peuvent interagir correctement et engager des conversations entre eux. En d'autres termes, il faut voir si une conversation peut être établie entre les services considérés sachant que chaque service peut se terminer correctement. Il existe trois types de compatibilité :

4.3.3.1 La compatibilité syntaxique

La compatibilité syntaxique entre les services se focalise principalement sur le coté structurel, c'est à dire voir si les types de messages et de données sont les mêmes. En effet, un message envoyé par un service doit avoir la même structure que le message reçu par le service partenaire.

4.3.3.2 La compatibilité sémantique

La compatibilité sémantique assure que les contenus des messages échangés entre les services sont correctement interprétés même si les services sont syntaxiquement compatibles. En effet, il est important de s'assurer que la signification d'une même donnée est identique dans chacun des deux services en communication.

4.3.3.3 La compatibilité comportementale

La compatibilité du comportement assure la cohérence des opérations à exécuter, des messages à envoyer et à recevoir et des résultats à livrer entre les services web.

4.4 La vérification classique sur des systèmes à états infinis

Les systèmes orientés base de données sont de plus en plus omniprésents dans de nombreux domaines tels que le commerce électronique, l'administration électronique, les systèmes d'information d'entreprises, etc. Ces systèmes sont souvent très complexes sujettes à des bugs couteux, cela oblige les experts à vérifier certaines propriétés critiques sur ces systèmes. Cette vérification s'applique généralement sur des systèmes finis. Dans cette dernière décennie, les chercheurs ont mis l'accent sur les systèmes qui traitent des données infinies. Cette infinité est due principalement à la configuration relationnelle de ces derniers. Dans cet axe, on peut citer quelques travaux qui ont utilisé la méthode du model checking pour vérifier des systèmes de transitions infinis [Burkart01], d'autres travaux plus récents on mis l'accent sur l'augmentation des procédures récursives par un paramètre entier [Bouajjani03], des réseaux de petri avec des données associées aux jetons [Lazic07, Lazic08] ainsi que les automates et les logiques sur des alphabets infinis [Neven04, Bojanczyk06, Bojanczyk11, Demri09].

Dans le domaine du e-commerce et avec l'apparition des services web orientés données, on trouve dans [Vianu09] un exemple de vérification dans lequel les clients commencent par ouvrir la première page d'un site du e-commerce, après ils peuvent choisir un parmi les boutons suivants : (*login*, *register*, *cancel*). Si par exemple le choix a été fait sur *login*. La réaction du site web est déterminée par une requête de vérification du nom et du mot de passe de l'utilisateur au niveau de la base de données des clients enregistrés. Si la réponse est positive, le *login* est réussi et le client pourra en suite consulter la page client ou la page administration selon son statut. Sinon il y a une transition vers la page d'erreurs.

La spécification S du système dans WebMl [1] génère un système de transition à états infinis Γ_S dont les configurations sont des instances de base de données relationnelles. Le schéma de la configuration a plusieurs composantes : la base de données de l'application web, les relations d'états, les tuples d'entrée et les relations de sortie. Selon la configuration de Γ_S, la spécification S définie des prochaines configurations possibles, selon les entrées de l'utilisateur. En particulier la relation de transition de Γ_S est décidable en PTIME et la taille de chaque configuration possible est polynomiale.

Pour spécifier les propriétés temporelles, l'auteur a utilisé la logique du premier ordre L pour décrire les configurations. Soit *price* comme étant une relation binaire de base de données qui donne le prix de chaque produit, *ship* une relation unaire qui donne les produits sélectionnés et *pay* est une relation binaire d'entrée qui donne le payement du produit. Pour dire que chaque prix doit être payé avant, l'auteur utilise la formule LTL **G** (p **B** q), sachant que q est interprétée comme étant une formule *ship(x)* et p comme étant $\exists z(pay(x, z) \wedge price(x, z))$ (la formule FO qui remplace les propositions pour donner une LTL(FO) est appelée les composantes de la formule FO). Noter que la variable x est libre dans toute la formule. donc elle va être quantifiée en utilisant le quantificateur universel ($\forall$). Voici la formule LTL(FO) générale : $\forall x(\mathbf{G}(\exists z(pay(x, z) \wedge price(x, z))\mathbf{B}ship(x)))$.

1. Web Modeling Language : est un langage graphique et une méthode pour le développement d'applications Web complexes

Les auteurs de [Hughes68, Garson01] ont travaillé sur l'axiomatisation complète de la logique modale du premier ordre sous hypothèses syntaxiques et sémantiques. L'infinité de données a été aussi introduite dans le domaine des services web orientés données utilisant un modèle relationnel [Berardi05b] ou bien XML [Abiteboul08b, Abiteboul09, Abiteboul08a]. Par exemple, dans [Abiteboul08a], les développeurs d'INRIA [2] ont travaillé sur ACTIVE XML qui intègre les paradigmes d'XML et des services web en permettant à des appels de services web d'être intégrés explicitement dans des documents XML. Les appels des services web retournent comme réponses des documents Active XML, qui vont être intégrés avec le document appelant. Dans [Abiteboul08b], les auteurs ont travaillé sur la vérification des systèmes dirigés par les bases de données d'où la sémantique est un système de transition Γ_S et les configurations sont des instances de base de données. Si L est considérée comme étant la logique pour décrire ces configurations et que la propriété $\varphi \in$ LTL (L), le problème de la vérification du modèle pour S et φ est de vérifier si $\Gamma_S \models \varphi$. Il n'est pas surprenant que cette formule est indécidable pour une logique L comme la FO. Le défi des auteurs est alors de trouver des restrictions sur les systèmes et les propriétés qui donnent la décidabilité, tout en restant suffisamment expressif pour être utile dans la pratique.

La vérification du modèle S en respectant φ peut être considérée comme une recherche d'un contre exemple de Γ_S. La difficulté immédiate par rapport à l'approche classique, provient du fait que Γ_S est un système à états infinis. En conséquence, la procédure de résiliation décrite dans le cas d'états finis échoue pour deux raisons. Premièrement, il y a un nombre infini de configurations initiales pour les exécutions. Deuxièmement, les exécutions n'ont pas besoin de répéter les configurations, donc la violation de φ ne garantit pas l'existence d'une exécution contre exemple périodique avec une représentation finie. Pour obtenir la décidabilité dans ce contexte, plusieurs approches sont possibles. La première approche est une variante de l'approche « small model property » une technique utilisée dans la

2. Institut National de Recherche en Informatique et en Automatique : C'est un institut de recherche français en informatique et en mathématiques

logique pour prouver la décidabilité de la satisfaisabilité pour une certaine classe de phrases. l'idée est alors de considérer que pour une classe donnée de spécifications et de propriétés, il est possible de montrer l'existence d'une exécution du système S qui viole la propriété φ qui peut être vérifiée en ne considérant que les préfixes finis des exécutions dont la taille (longueur et taille de la configuration) est plus petite qu'une limite calculable de S et de φ. La capacité de ne considérer que les préfixes finis peut être une conséquence d'une propriété périodique ou d'une restriction en s'assurant que le système se termine après un nombre limité de transitions (prolongé artificiellement à une exécution infinie).

La propriété de la plus petite exécution donne immédiatement une efficace procédure de vérification qui consiste à explorer l'espace fini de l'exécution et avec cette méthode les auteurs ont montré la décidabilité de la vérification du modèle pour une classe des systèmes AXML et les propriétés LTL(tree). D'autres solutions plus efficaces consistent à utiliser des représentations symboliques des exécutions. Cela peut être efficace que si la taille de la représentation est plus petite que la taille des exécutions qu'elle représente. Cette approche a été utilisée dans [Deutsch04, Deutsch06, Deutsch07] pour obtenir une procédure de vérification du modèle PSPACE pour les systèmes relationnels.

Les représentations symboliques ont été aussi utilisées dans [Berardi03a] et [Berardi05a], où l'accent est mis sur la synthèse automatique des compositions de services. Par exemple dans [Berardi05a], les auteurs ont crée un modèle appelé COLOMBO qui présente un framework basé sur des services web caractérisés par des processus atomiques qui sont des opérations qu'ils peuvent accomplir, leur impacte sur le monde réel est modélisé comme une base de données relationnelle, leurs transitions sont basées comportement caractérisées par les messages qu'ils peuvent envoyer ou recevoir. L'objectif principale de ce travail est la composition automatique des services web.

Dans la partie qui suit, on va vous présenté ce modèle d'une manière détaillée car il est considéré comme étant la base de notre recherche. Ce qui nous intéresse

dans cet article se sont les techniques de traitement des données qui varient de façons symbolique d'un domaine infini vers un ensemble fini, la communication par messages entre les services web, la représentation relationnelle des données et l'étude de la compatibilité entre les services.

4.5 Le modèle colombo

Le modèle colombo intègre quatre aspects fondamentaux :

- L'état world qui représente le monde réel (L'univers) vu comme étant une instance de base de données dans un schéma relationnel,
- Des processus atomiques (opérations) qui peuvent consulter ou modifier l'état world ou inclure des effets conditionnels (ces processus ressemblent aux processus atomiques de OWL-S),
- L'envoi de messages avec la notion de ports et de liens vue dans les protocoles de service web (WSDL, WS-BPEL),
- Le comportement des web services (qui fait appel à des processus atomiques et à des activités de transmission de messages) est spécifié par un système de transitions à état fini.

Pour faciliter les choses, l'auteur suppose que chaque instance du service web a un « local store », utilisé pour capturer les paramètres des messages entrants, les valeurs de sortie des processus atomiques et pour remplir les paramètres des messages sortants avec les paramètres d'entrée des processus atomiques. Donc le branchement conditionnel dans un web service sera basé sur les valeurs des variables du « local store ».

Le schéma word est constitué de quatre relations définies sur :

- Un domaine booléen « Bool »,

– Un ensemble infini d'éléments non interprétés $\text{Dom}_{=}$ (Dans lequel, on trouve que la relation d'égalité) noté par des chaines de caractères alphanumériques,
– Un ensemble infini et ordonné $\text{Dom}_{\leq}$, noté par des nombres.

Une instance du schéma word est vue dans la figure 4.2 suivante :

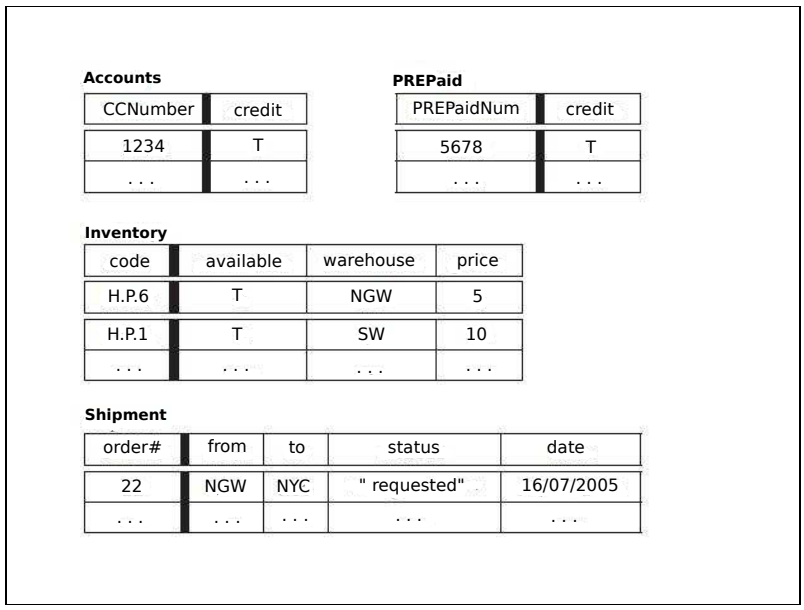


FIGURE 4.2: Une instance du schéma world dans colombo [Berardi05a]

Dans la figure ci-dessus, la relation *Accounts* stocke les numéros des cartes de crédit avec des informations de chargement, la relation *PREPaid* stocke les numéros des cartes prépayées avec des informations d'utilisation, *Inventory* contient le code article, le prix et l'entrepôt où les articles sont disponibles, *Shipment* contient l'identifiant de la commande, l'entrepôt source, l'entrepôt cible et la date d'expédition.

La figure 4.3 montre l'alphabet A des processus atomiques invoqués par les services web. Dans cette figure, la valeur nulle est notée par "w", le symbole « - » regroupe les éléments du tuple qui sont restés stables après l'exécution du processus atomique et la syntaxe « insert, delete, modify » spécifie les effets potentiels sur l'état world. Finalement, la fonction d'accès $\text{f}_j^R(\langle a_1, ..., a_n \rangle)$ est utilisée pour

récupérer le n+j ème élément dans R identifié par la clé $\langle a_1, ..., a_n \rangle$(Le j^{eme} élément après la clé).

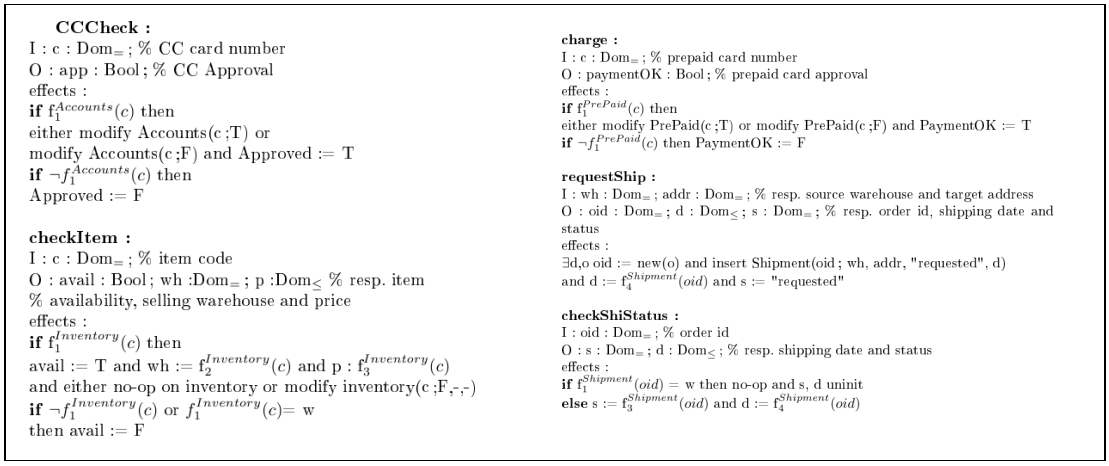


FIGURE 4.3: L'alphabet des processus atomiques dans colombo [Berardi05a]

La figure 4.4 montre les systèmes de transitions des services web suivant : *Bank* qui vérifie si la carte de crédit peut être utilisée pour le payement, *Storefront* donne le prix et l'entrepôt où se trouve l'article, Next Generation Warehouse qui permet à un client d'utiliser les deux cartes, Standard Warehouse qui peut être utilisé uniquement avec la carte de crédit. Dans la même figure, la transmission d'un message et représentée par « ! » et la réception par « ? ».

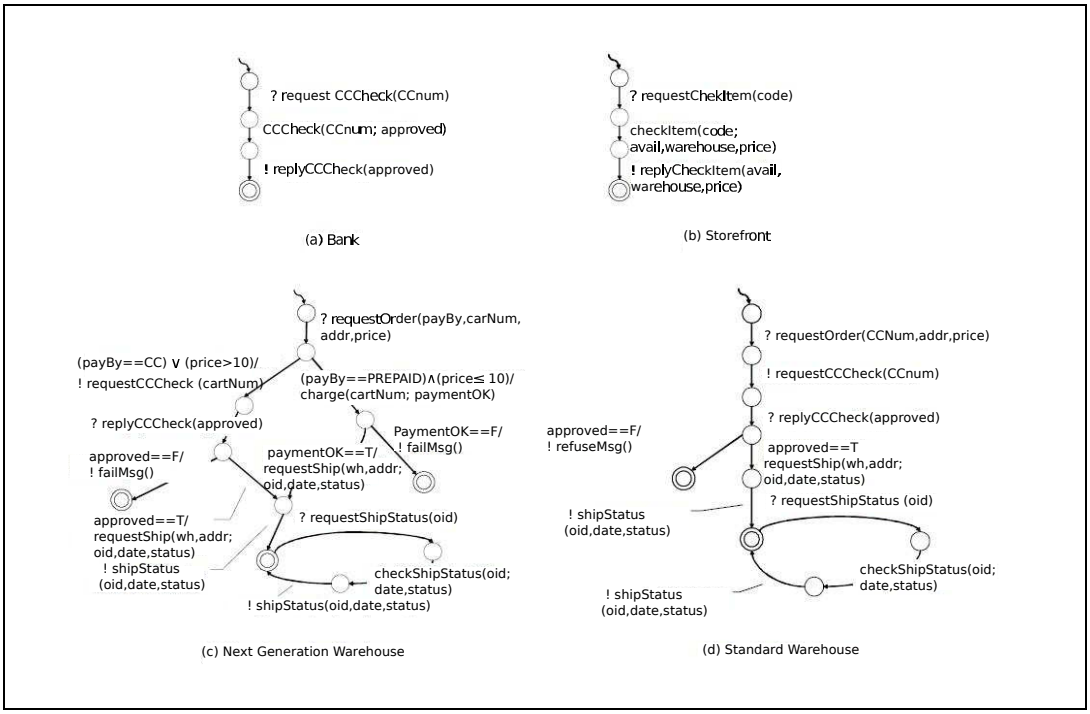


FIGURE 4.4: Le système de transitions des services disponibles [Berardi05a]

La figure 4.5 montre le système de transitions du service global qui spécifie d'une manière générale l'interaction avec le client.

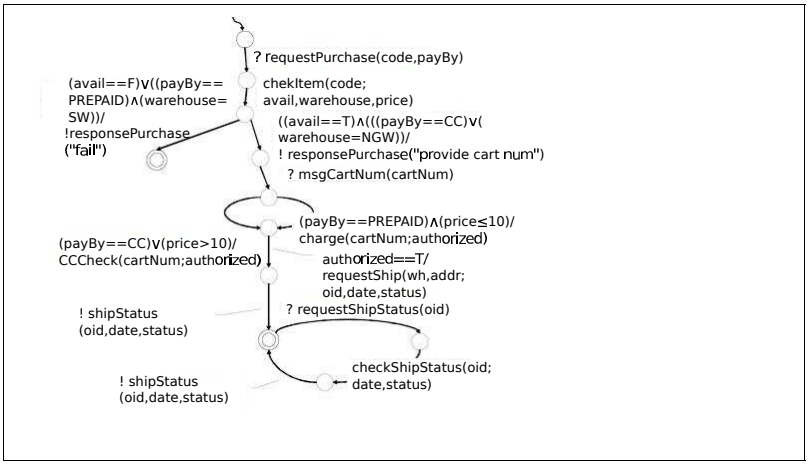


FIGURE 4.5: Le système de transitions du service global [Berardi05a]

4.5.1 La représentation formelle du modèle colombo

Le schéma de base de données est un ensemble fini W de relations qui ont la forme suivante : R_K (A_1,.....,Am_K ;B_1,.....,Bn_l), où A_1,.....,Am_K est la clé de R_K, et chaque attribut A_i, B_j peut prendre des valeurs dans Bool, $Dom_=$, $Dom_\leq$.

La contrainte clé accessible est une expression qui a la forme suivante $\varphi = \forall x_1, .., x_n(\psi)$, où les x_i représentent des variables distinctes et ψ comme étant une expression booléenne sur des atomes, et des termes accessibles. L'instance world I satisfait cette contrainte si pour toutes les affectations α aux variables $x_1, ..., x_n$, la formule ψ est vraie dans I.

La syntaxe pour la description des conditions, les contraintes d'intégrités, et les « Local stores » des services web est basée sur l'utilisation des symboles nommés constantes (Dom= Bool $\cup$ $Dom_=$ $\cup$ $Dom_\leq$).

Un processus atomique est un objet p avec la signature suivante (I, O, CE) avec I et O respectivement comme signatures d'entrée et de sortie (ensembles de variables) et CE (Effets conditionnels) qui est un ensemble contenant des paires (c, E) sachant que "c" est la condition du processus atomique et "E" l'ensemble non vide des effets. La condition "c" est une expression booléenne des atomes, des termes accessibles, une famille de constantes et les variables d'entrées u_1,...,u_n. Un effet e $\in$ E est une paire (es, ev) avec es (effet sur le monde « world ») comme étant un ensemble d'expressions qui ont la forme suivante :

- Insert R(t1,....,t_K ;s_1,....,s_l),
- Delete R(t1,....,t_K),
- Modify R(t1,....,t_K ;r_1,...,r_l),
  avec t_i et s_j comme étant des termes accessibles sur un ensemble de constantes et de variables u_1,....,u_n, et chaque r_j comme étant un terme accessible ou un symbole spécial « - » (qui montre que la position du tuple dans R ne doit pas être changée).

Le « ev » représente l'effet sur les sorties qui est un ensemble d'expressions avec la forme suivante :

- v_j := t /j ∈[1..m] et t comme étant un terme accessible sur un ensemble de constantes et de variables u_1,...,u_n,
  ou
- v_j := /j ∈[1..m].

Donc la définition de la sémantique d'un processus atomique est relativement simple, elle est basée sur les valeurs des variables d'entrée et l'instance du monde actuel, si un effet conditionnel (c, E) a une vraie condition alors on choisit un élément $e \in$ E. Si l'application de e sur l'instance world satisfait les contraintes globales $\sum$ alors e est utilisée pour modifier l'instance du world et pour déterminer les valeurs de sortie (on écrit (α, I) $\vdash_{p(r_1,..,r_n;v_1,..,v_m)}$ (α',I') si (α', I') est le résultat de l'exécution du processus atomique p avec les entrées r_i et les sorties v_j).

Les messages, ports et liens :

Le type message a le nom « m » et la signature $\langle d_1, .., d_n \rangle$, la signature du port est un ensemble P de paires qui ont la forme suivante :(m, in) ou bien (m, out) avec « m » comme étant le type de message et (in, out) la direction du message. On considère que F= { S_1,...,S_n } est une famille de services qui ont les ensembles de paires suivants { P_1,..,P_n }. Un lien est un tuple qui a la forme suivante (S_i, m, S_j, n) sachant que (m, out) ∈ P_i, (n, in) ∈ P_j et m, n ont les mêmes signatures.

4.5.2 Le modèle abstrait du processus interne des services

Cela nécessite l'utilisation de l'automate suivant : A (Q, δ, F, LS, QS) sachant que Q est un ensemble fini d'états, LS (local store) qui est un ensemble fini de variables déterminant s'il y a un message dans la file d'attente, QS (queue store) est un ensemble de variables utilisées pour garder les paramètres des messages entrants et la fonction de transition δ qui est un quadruplet (s, c, μ, s')/ s, s' ∈ Q, "c" est une condition sur LS ∪ QS et μ est un envoi de message ou bien une lecture

ou une invocation d'un processus atomique. Dans la pratique, il existe deux types d'automates :

1. A(S), la signature de l'automate pour les services hors le client,
2. A(C), la signature de l'automate pour le service client qui contient uniquement deux états, ReadyToTransmit et ReadyToRead.

Donc à un moment donné le service passe d'un état à un autre, ce passage peut être modélisé par la relation $\vdash$(moves-to), exemple : (id^s, I) $\vdash$ ($id^{s'}$, I′), avec id^s, $id^{s'}$ comme étant des descriptions instantanées de S et I, I′ des instances de la base de données.

4.5.3 L'exécution du système

Généralement le système est un triplet S = (C, F, L)/ "C" est le client, F= { S_1,...,S_n } une famille de services web et "L", le lien entre (C, F).

Le déroulement de S est une sequence finie $\varepsilon = \langle (id_1, I_1),...,(id_q, I_q)\rangle$, avec id_1 comme étant l'id initial de S. Le déroulement se terminera avec succès si id_q est dans un état final dans A(C) et dans chaque A(S).

Dans ce cas, les auteurs ont représenté l'exécution de S par un arbre d'exécution T dans lequel chaque nœud est étiqueté par la paire (id, I) sauf la racine, et chaque arête est étiquetée par la trace d'exécution.

4.5.4 L'élimination de l'infinité de données

Dans le modèle colombo, l'objectif principal des auteurs était la proposition d'une nouvelle technique de composition automatique des services et comme les services utilisent des données qui varient sur un domaine infini, les auteurs ont proposé une manière symbolique pour rendre ces données finies. cette technique a été inspirée par le travail de Hull et Su dans [Hull89, Hull94]. L'idée principale est de construire une image symbolique de l'univers I et de la relation $\vdash$, pour pouvoir

construire un arbre d'exécution symbolique T′ avec un nombre fini de branches qui a une relation d'homomorphisme avec l'arbre T qui a une infinité de valeurs.

4.5.5 L'étude de l'équivalence entre des services colombo

Parmi les rares travaux qui traitent la comparaison (équivalence ou compatibilité) entre deux systèmes infinis, BERARDI et *al* dans [Berardi05a] ont traité le problème de l'équivalence entre un modèle de service web classique et un modèle composé, c'est à dire ils supposent qu'ils ont un modèle de service ϑ = (C, { V }, L), et à partir de ce système, ils font une composition de services à partir d'un ensemble de services S_1 jusqu'à S_n avec un service médiateur S_0 qui peut envoyer, recevoir, et lire les messages. Après le choix d'un service médiateur, il construisent le lien L′ à partir du client C et de l'ensemble { S_0,S_1,...,S_n } de telle manière que ϑ le système global et S = (C, { S_0, S_1,...,S_n }, L′) soient équivalents.

Pour résoudre le problème de l'équivalence les auteurs ont placé des restrictions :

1. **Le blocage du comportement** : Si un service envoie un message, alors il va être bloqué dans son état jusqu'à ce qu'il reçoit un message.

   Cette restriction est utilisée pour éliminer la concurrence dans le système.

2. **La délimitation de l'accès** :
   - Il existe un K > 0 tel que dans chaque déroulement du client C, le nombre des valeurs qui peuvent être envoyées est $\leqslant$ K + le nombre des valeurs reçues par le client C,
   - Pour chaque p > 0 il y a un q > 0 de telle manière que dans chaque exécution du système ϑ, si le client envoie p nouvelles valeurs, alors l'invocation du processus atomique peut exécuter q clés.

     Cette restriction est utilisée pour délimiter les valeurs du domaine actif pour chaque exécution dans ϑ.

Après la désignation des restrictions, l'infinité de données peut être éliminée par l'utilisation des valeurs symboliques.

Récemment, Akroun et *al* dans [Akroun14b] ont aussi travaillé sur le modèle colombo dans le cadre du problème d'existence d'une relation de simulation entre des services colombo et ils ont trouvé que la simulation entre les services non bornés est indécidable par contre pour les services bornés qui n'accèdent pas à la base de données, elle est 2EXPTIME-complet quant les services peuvent accéder à une base de données bornée.

4.6 Un modèle formel des systèmes réactifs dirigés base de données

Un deuxième travail qui traite le problème de l'équivalence des données est le travail de Spielmann [Spielmann03]. Dans ce travail l'auteur traite le problème de la vérification des propriétés temporelles des ASM's relationnels dans un exemple de commerce électronique. D'une manière générale le vocabulaire d'un ASM relationnel (abtract State Machine) est le suivant : $\Upsilon = (\Upsilon_{in}, \Upsilon_{db}, \Upsilon_{mem}, \Upsilon_{out}, \Upsilon_{log})$ sachant que :

- Υ_{in}, Υ_{db}, Υ_{mem}, Υ_{out} représentent le vocabulaire relationnel,
- $\Upsilon_{log} \subset \Upsilon_{in} \cup \Upsilon_{out}$.

Le programme d'un ASM relationnel « Π » est un ensemble fini de règles qui ont la forme suivante : **if** $\varphi(\bar{x})$ **then** $(\neg)$ R $(\bar{x})$ sachant que :

- $\varphi(\bar{x})$ la condition de la règle (the guard) est une formule FO (First Order) sur $\Upsilon_{in} \cup \Upsilon_{db} \cup \Upsilon_{mem}$ avec free $(\varphi) = \{\bar{x}\}$,
- R $(\bar{x})$ est une formule FO atomique sur $\Upsilon_{mem} \cup \Upsilon_{out}$, cette formule est positive si R $\in \Upsilon_{out}$.

Dans cet article, l'auteur a donné l'exemple d'un ASM relationnel qui représente un fournisseur là où les clients peuvent commander des produits qui vont être facturés, payés et enfin livrés. Après la définition formelle de l'ASM fournisseur, l'auteur a étudié le problème de la vérification des propriétés tem-

porelles et le problème de l'équivalence des ASMs dans le contexte où un client peut personnaliser l'ASM fournisseur selon ses propres besoins, et dans ce cas il est nécessaire de comparer entre l'ASM personnalisé par le client et l'ASM original.

Il y a aussi l'extension des ASMs appelée « Extended Abstract State Machine transducer » ou bien ASM$^+$ [Deutsch04], qui captent d'une manière simple les caractéristiques essentielles des systèmes réactifs dirigés par des bases de données relationnelles. Le modèle est une extension du modèle ASM (Abstract State Machine transducer) étudié par Spielmann dans [Spielmann03]. ASM$^+$ répond aux évènements d'entrée en produisant une sortie, et en gardant les informations des états dans les relations désignées. Le contrôle du dispositif est spécifié en utilisant des requêtes de premier ordre. La principale motivation du ASM$^+$, c'est qu'ils sont suffisamment puissants pour simuler des spécifications de services web dans le style de WebML.

Un capteur ASM$^+$ A a été défini dans [Deutsch04] comme étant un tuple $\langle$ **D, S, I, O**, R $\rangle$, où :

- **D, S, I, O** sont des schémas relationnels appelés des schémas de : base de données (database), état(state), entrée (input), sortie (output), on note par $\mathbf{Prev_I}$ le vocabulaire relationnel $\{$ prev$_I$ $\setminus$ I $\in$ **I**$\}$, où prev$_I$ a la même arité que I,
- R est un ensemble qui contient :
  1- Pour chaque relation d'entrée R $\in$ **I** d'arité $k > 0$ une pré-condition $\varphi_R(\bar{x})$ où $\varphi_R(\bar{x})$ est une formule FO sur le schéma $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I}$ avec k variables libres $\bar{x}$.
  2- Pour chaque relation d'état S $\in$ **S** :
  * Une règle d'insertion S$(\bar{x}) \leftarrow \varphi_S^+(\bar{x})$,
  * Une règle de suppression $\neg$ S$(\bar{x}) \leftarrow \varphi_S^-, (\bar{x})$
  Où l'arité de S est k, $\bar{x}$ est un k-tuple de variables distinctes et $\varphi_S^{+/-}(\bar{x})$ sont des formules FO sur le schéma $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I} \cup \mathbf{I}$, avec des variables libres $\bar{x}$.

3- Pour chaque relation de sortie $O \in \mathbf{O}$, une règle de sortie $O\ (\bar{x}) \leftarrow \varphi_O(\bar{x})$, où l'arité de O est k, $\bar{x}$ est un k-tuple de variables distinctes et $\varphi_O(\bar{x})$ est une formule sur le schéma $\mathbf{D} \cup \mathbf{S} \cup \mathbf{Prev_I} \cup \mathbf{I}$, avec des variables libres $\bar{x}$.

Intuitivement, les relations d'état désignent la partie de la base de données qui peut être modifiée pendant l'exécution du ASM$^+$. Les relations de la base de données qui restent inchangées pendant l'exécution sont utiles dans la formulation des restrictions nécessaires à la décidabilité.

Afin de spécifier les propriétés du ASM$^+$, l'auteur a utilisé le langage de la LTL(FO). Donc pour un ASM$^+$ A $= \langle\ \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{O}, R\rangle$, les composants FO des formules LTL(FO) sont sur le vocabulaire

$$\langle\ \mathbf{D}, \mathbf{S}, \mathbf{I}, \mathbf{Prev}_I, \mathbf{O}\ \rangle$$

Il est facile de voir qu'il est indécidable si un ASM$^+$ satisfait une formule LTL(FO) et pour obtenir la décidabilité, les auteurs utilisent la méthode de restriction appelée « input boundeness ». L'idée principale des entrées bornées c'est que les quantifications utilisées dans les formules de spécification et des propriétés sont gardées par des atomes d'entrée. Par exemple, la formule LTL(FO) :

$\forall x\ (\mathbf{G}\ (\exists z(pay(x,z) \wedge price(x,z))\ \mathbf{B}\ ship(x)))$ est une entrée bornée, la quantification $\exists z$ est gardée par $pay(x,z)$ et pay est une relation d'entrée.
Après la vérification formelle des ASM$^+$, l'auteur a traité aussi le problème de l'équivalence entre deux modèles (Deux ASMs relationnels sont équivalents s'ils produisent deux sorties sémantiquement équivalentes avec les mêmes valeurs d'entrées).

4.7 Les systèmes centrés artéfacts

Les artéfacts métiers (business artifacts) ou tout simplement artéfacts sont des enregistrements de données sur lesquels opèrent des services. Le cycle de vie d'un

type artéfact spécifie généralement le séquencement des services invoqués sur ce type d'artéfact. Cette nouvelle approche a été introduite dans plusieurs travaux, par exemple dans [Bhattacharya09], les auteurs ont travaillé sur un modèle d'artéfacts dans lequel chaque type a une famille d'attributs avec un état relationnel. Initialement, chaque artéfact est représenté par un ensemble d'attributs qui vont être par la suite complétés ou remplacés selon la nature du service invoqué. L'association des services avec les artéfacts est déclarative c'est à dire en utilisant des paramètres d'entrée, des paramètres de sortie, des pré-conditions et des post-conditions. L'utilisation des post-conditions (appelées aussi effets conditionnels) permet le non déterminisme dans le résultat d'un service. Une fois que la couche logique a été bien définie, les auteurs ont traité le passage d'une spécification déclarative vers une réalisation optimale.

Des travaux sur l'analyse formelle des processus métier centrés artéfacts dans des contextes restreints ont été aussi publiés dans [Chandra85, Burkart01, Bhattacharya07, Deutsch09, Xu17]. Les propriétés étudiées dans ces études contiennent en général l'accessibilité des contraintes temporelles [Burkart01] et l'existence d'une exécution complète [Bhattacharya07]. L'article [Burkart01] se base sur une version essentiellement procédurale de flux de travail centré artéfacts, et les auteurs de l'article [Bhattacharya07] sont les premiers qui ont étudié une version déclarative qui a été utilisée dans [Bhattacharya09]. Dans chaque article la vérification est indécidable, des résultats de décidabilité ont été obtenus sauf s'il y a des restrictions, par exemple limiter le langage pour que les conditions ne concernent que les artéfacts et non leurs valeurs d'attributs [Chandra85]. Dans [Deutsch09], les auteurs ont identifié une classe expressive d'artéfacts sur lesquels la vérification est néanmoins décidable avec une complexité PSPACE-complete qui n'est pas pire que la vérification des modèles classiques à états finis.

Le tableau ci-dessous illustre une classification des différents travaux de recherche selon le formalisme de modélisation utilisé, la présence ou l'absence des données, l'approche de vérification utilisée et la solution apportée.

TABLE 4.1: La classification des travaux de vérification avec la présence des données

Travail	Approche de vérification utilisée		Données	Formalisme de modélisation utilisé		Solution utilisée				
	Model Checking	Théo-pr		Sys tran	Rés Pet	Input boundedness	SMP	Rep sym	Région d'automates	RS
[Burkart01]	Oui	Non	Oui	Oui	Oui	Non	Non	Oui	Non	Oui
[Lazic07]	Non	Oui	Oui	Non	Oui	Non	Non	Non	Non	Oui
[Lazic08]	Non	Oui	Oui	Non	Oui	Non	Non	Non	Non	Oui
[Neven04]	Non	Oui	Oui	Oui	Non	Non	Non	Non	Non	Oui
[Bojanczyk06]	Non	Oui	Oui	Oui	Non	Non	Non	Non	Non	Oui
[Bojanczyk11]	Non	Oui	Oui	Oui	Non	Non	Non	Non	Non	Oui
[Demri09]	Non	Oui	Oui	Oui	Non	Non	Non	Non	Non	Oui
[Vianu09]	Oui	Non	Oui	Oui	Non	Oui	Non	Non	Non	Oui
[Berardi03a]	Non	Oui	Oui	Oui	Non	Non	Non	Oui	Oui	Oui
[Berardi05b]	Non	Oui	Oui	Oui	Non	Non	Non	Oui	Oui	Oui
[Abiteboul08b]	Non	Oui	Oui	Oui	Non	Non	Oui	Non	Non	Oui
[Abiteboul09]	Non	Oui	Oui	Oui	Non	Non	Oui	Non	Non	Oui
[Abiteboul08a]	Non	Oui	Oui	Oui	Non	Non	Oui	Non	Non	Oui
[Deutsch04]	Non	Oui	Oui	Oui	Non	Non	Non	Oui	Non	Oui
[Deutsch06]	Non	Oui	Oui	Oui	Non	Non	Non	Oui	Non	Oui
[Deutsch07]	Non	Oui	Oui	Oui	Non	Non	Non	Oui	Non	Oui
[Spielmann03]	Non	Oui	Oui	Oui	Non	Non	Non	Oui	Non	Oui
[Akroun14a]	Non	Oui	Oui	Oui	Non	Non	Non	Non	Oui	Non
[Mezni18]	Oui	Non	Oui	Oui	Non	Non	Non	Oui	Non	Oui

Dans Le tableau ci-dessus nous avons utilisé les abréviations suivantes :

- Théo-pr : Théorème-preuve,
- Sys tran : Systèmes de transitions,
- Rés pet : Réseaux de Petri,
- SMP : Small Model Property,
- Rep sym : Représentation symbolique,
- RS : Restrictions spécifiques.

4.8 Conclusion

Dans ce chapitre, nous avons traité les différentes méthodes de vérification en se focalisant principalement sur la vérification classique des systèmes infinis dans lesquels les états finis sont remplacés par une infinité d'instances de base de données. On a aussi aborder le problème de la compatibilité des processus orientés données car les applications orientées services et grâce au couplage faible des services web nécessitent à un moment donné l'intégration de nouveaux services ou la remplaçabilité d'un service défaillant par un autre service équivalent qui est compatible avec les autres services.
Dans la prochaine partie, nous allons présenter formellement l'analyse de la compatibilité des protocoles de services web orientés données en traitant plusieurs sous-modèles de protocoles et en analysant à chaque fois la décidabilité de la compatibilité par la méthode de vérification théorème/preuve avec la définition des classes de compatibilité existantes.

“ L’infini n’est pas un état stable, mais la croissance elle-même.

Aristote

5

Analyse formelle de la compatibilité des protocoles de services orientés données

5.1 Introduction

Ce chapitre permet d'étudier la compatibilité des protocoles de services qui sont représentés avec une même instance de base de données, c'est à dire on considère qu'on a une base de donnée fixe et on jouera après sur les langages de requêtes à savoir (L_G qui est le langage de la condition Guard, L_M le langage des messages et L_U le langage de la mise à jour).

5.2 La définition informelle de la compatibilité

Un protocole de service est la description du comportement externe d'un service. On modélise ce protocole avec une machine à états finis où les transitions sont étiquetées avec les messages envoyés et reçus et les états représentent les différentes phases par lesquelles le service passe durant son interaction. Une conversation est la séquence de messages échangés entre deux services. L'étude de la compatibilité traite le problème de savoir si deux services peuvent s'échanger des messages tel que si $S1$ envoie un message, $S2$ peut le lire et inversement (sachant que $S1$ et $S2$ sont deux services différents).

5.3 L'analyse des différents types de protocoles de services avec une base de données fixe (une instance)

Dans cette partie, on va analyser la compatibilité des protocoles de services augmentés par une base de données fixe, dans cet axe on peut déterminer plusieurs modèles.

5.3.1 Le cas 1 : les protocoles de services gardés sur une instance fixe I $(L_G,\emptyset,\emptyset)^I$

Selon le tableau 5.1, on peut donner la définition suivante :

TABLE 5.1: Un protocole P de type $(L_G,\emptyset,\emptyset)^I$

BDD	L_G	L_M	L_U
Finie	CQ	$\emptyset$	$\emptyset$

Définition 1.

Considérant P comme étant un protocole de service tel que P = (S, s_0, F, I, L_G, M, Δ) sachant que :

- S est un ensemble fini d'états,
- $s_0 \in S$ est l'état initial,
- F est l'ensemble des états finaux, si $F = \emptyset$, le protocole P est appelé un protocole vide,
- M est un ensemble fini de messages , pour chaque message $m \in M$, on a la fonction polarité (p, m) qui sera positive si m est un message entrant pour cela on note $m(+)$ et négative si m est un message sortant et on note m(-),
- I est une instance d'une base de données,
- L_G est un langage de requêtes booléennes q_i(I),
- $\Delta \subseteq S \times L_G(I) \times M \times S$ est la fonction de transition.

5.3.1.1 La sémantique d'exécution du protocole

Soit le protocole de service P = (S, s_0, F, I, L_G, M, Δ), une séquence d'exécution du protocole est une séquence d'états, de requêtes booléennes et de messages telle que la configuration des états est représentée par le tuple (S_i, I)(voir figure 5.1).

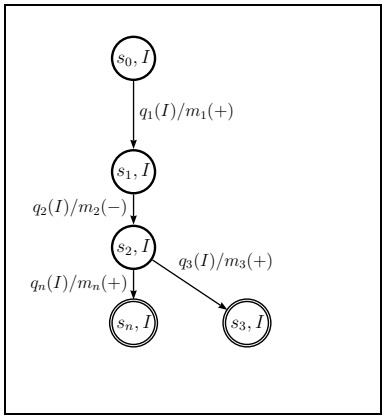


FIGURE 5.1: Le protocole P

A partir de la figure 5.1, on peut donner une séquence d'exécution par exemple : $s_0, I \xrightarrow{m_1(+)} s_1, I \xrightarrow{m_2(-)} s_2, I \xrightarrow{m_3(+)} s_3, I$, d'une manière générale on peut donner la séquence d'exécution comme suit :

$s_0, I \xrightarrow{m_1(p)} s_1, I \xrightarrow{m_2(p)} s_2, I \longrightarrow \ldots\ldots\ldots \xrightarrow{m_n(p)} s_n, I.$

Sachant que :

- (s_0, I) est une configuration initiale avec s_0 comme état initial.
- quelque soit $0 < i \leq n\text{-}1$ on a la transition $(s_i, I).q_{i+1}(I)m_{i+1}(s_{i+1}, I)$ tel que $q_{i+1}(I)$= vrai.

Une séquence d'exécution complète est une séquence d'exécution qui part d'une configuration initiale et aboutit à une configuration finale. Par exemple dans la figure 5.1, on considère que la configuration (s_3, I) comme étant une configuration finale car l'état s_3 est un état final.

5.3.1.2 L'étude de la compatibilité des protocoles

La compatibilité des protocoles tels qu'ils ont été définis précédemment avec leurs sémantiques d'exécutions peut être étudiée avec l'étude de la compatibilité des séquences d'exécutions complètes. Nous définissons cette compatibilité des séquences comme suit :

Définition 2.

On dit qu'un état (s_i, I) est compatible avec un état (s'_i, I) et on note (s_i, I) comp (s'_i, I) ssi :

$\forall\ (s_i, I) \xrightarrow{q()/m(p)} (s_j, I)$ alors $\exists\ (s_i', I) \xrightarrow{q'()/m(\overline{p})} (s'_j, I)$.

Et (s_j, I) comp (s'_j, I).

Définition 3.

On dit que deux protocoles P = (S, s_0, F, I, L_G, M, Δ) et P′ = (S', s'_0, F', I, L_G, M, Δ') sont compatibles ssi :

(s_0, I) comp (s'_0, I).

Selon cette définition deux classes de compatibilité peuvent être déduites :

La compatibilité totale

Soient deux protocoles P = (S, s_0, F, I, L_G, M, Δ) et P′ = (S', s'_0, F', I, L_G, M, Δ') , on dit que P est totalement compatibles avec P′ et on note P Tcomp P′ ssi :

- (s_0, I) comp (s'_0, I).
- $\forall\ (s_i, I) \xrightarrow{q()/m(p)} (s_j, I)$ alors $\exists\ (s_i', I) \xrightarrow{q'()/m(\overline{p})} (s'_j, I)$ et (s_j, I) comp (s'_j, I).

La compatibilité partielle

Soient deux protocoles P = (S, s_0, F, I, L_G, M, Δ) et P′ = (S', s'_0, F', I, L_G, M, Δ') , on dit que P est partiellement compatibles avec P′ et on note P Pcomp P′ ssi :

- (s_0, I) comp (s'_0, I).
- $\exists\ (s_i, I) \xrightarrow{q()/m(p)} (s_j, I)$ alors $\exists\ (s_i', I) \xrightarrow{q'()/m(\overline{p})} (s'_j, I)$ et (s_j, I) comp (s'_j, I).

Théorème 1

Étant donné deux protocoles de services P = (S, s_0, F, I, L_G, M, Δ) et P′ = (S', s'_0, F', I, L_G, M, Δ'), le problème de test si P comp P′ est décidable.

Preuve :

La preuve du théorème 1 est trés simple, dans ce cas on procède par l'observation. Si on a deux protocoles ayant la forme vue dans la figure 5.1, selon les définitions 1 et 2 on voit que les séquences d'exécution des protocoles sont finies car l'instance I est finie (le nombre d'états des deux protocoles de services est fini), ce pendant, le problème P comp P$'$ est décidable.

5.3.2 Le cas 2 : les protocoles de services non-gardés sur une instance fixe I $(L_G,\emptyset,\emptyset)^I$ (requête d'insertion)

Dans ce sous modèle, on va étudier la compatibilité des protocoles de services qui sont représentés avec une même instance de base de données sachant que le passage d'un état à un autre se fait par l'exécution d'une requête d'insertion, c'est à dire le cas suivant :

TABLE 5.2: Un protocole P de type $(\emptyset,\emptyset,L_U)^I$

BDD	L_G	L_M	L_U
Finie	$\emptyset$	$\emptyset$	CQ

5.3.2.1 La définition du protocole de service

Considérant P comme étant un protocole de service tel que P = (S, s_0, F, I_0, L_{insert}, Δ) sachant que :

- S est un ensemble fini d'états,
- $s_0 \in S$ est l'état initial,
- F est l'ensemble des états finaux, si $F = \emptyset$, le protocole P est appelé un protocole vide,

- I_0 est une instance initiale d'une base de données,
- L_{insert} est un langage de requêtes conjonctives ex : $q_i(I)$,
- $\Delta \subseteq S \times L_{insert}(I) \times S$ est la fonction de transition.

5.3.2.2 La sémantique de la requête conjonctive

Dans notre sous-modèle $\hat{q}(I)$ représente le résultat de la requête d'insertion sur l'instance I et il est donné de la manière suivante : $\hat{q}(I) = I'$ sachant que $I' = q(I) \cup I$ c'est à dire le résultat de $q(I)$ est inséré dans I.

5.3.2.3 La sémantique d'exécution du protocole

Soit le protocole de service P = (S, s_0, F, I_0, L_{insert},Δ), une séquence d'exécution du protocole est une séquence d'états et d'un ensemble de requêtes conjonctives telle que la configuration des états est représentée par le tuple (s_0,I_i) et elle est de la manière suivante :

$s_0, I \xrightarrow{q(I)} s_0, I'$
tel que $I' = q(I) \cup I$.

Voici un exemple de protocole qui utilise ce type de requêtes , on suppose qu'on a un schéma de relation R(A,B) avec A et B comme des attributs et I_0 comme étant une instance initiale de R exemple :

$I_0 =$

A	B
1	2
2	3
3	4

Le protocole P = (S, s_0, F, I_0, L_{insert}, Δ) est défini de la manière suivante (voir figure 5.2) :

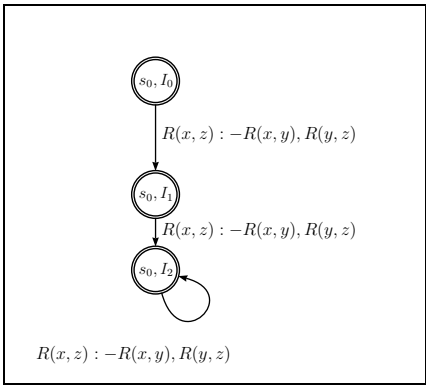


FIGURE 5.2: P avec une requête d'insertion

Après la première exécution de la requête d'insertion, $I_1 =$

A	B
1	2
2	3
3	4
1	3
2	4

et après la deuxième exécution, $I_2 =$

A	B
1	2
2	3
3	4
1	3
2	4
1	4

Une séquence d'exécution complète est une séquence d'exécution qui part d'une configuration initiale et abouti à une configuration finale exemple dans la figure 5.2 l'état (s_0, I_2) est un état final.

5.3.2.4 L'étude de la compatibilité des protocoles

La compatibilité des protocoles tels qu'ils ont été définis précédemment avec leurs sémantiques d'exécutions peut être étudiée avec l'étude de la compatibilité des séquences d'exécutions complètes. Nous définissons cette compatibilité comme suit :

Définition 4.

On dit qu'un état (s_0, I_i) est compatible avec un état (s'_0, I'_j) et on note (s_0, I_i) comp (s'_0, I'_j) ssi : $I_i = I'_j$.

5.3.2.5 La compatibilité totale

On dit que P = (S, s_0, F, I_0, L_{insert},Δ) est totalement compatible avec P′ = (S', s'_0, F', I'_0, L_{insert},Δ') et on note P Tcomp P′ ssi :

– $\forall\ (s_0, I_i) \xrightarrow{q_i(I_i)} (s_0,\ I_{i+1})$ alors $\exists\ (s_0', I'_j) \xrightarrow{q'_j(I_j)} (s'_0,\ I'_{j+1})$.

Tel que (s_0,I_i) comp (s'_0,I'_j) et (s_0,I_{i+1}) comp (s'_0,I'_{j+1}). Pour i ∈ [0,..,n], j ∈ [0,..,m] et n, m ∈ ℕ.

5.3.2.6 La compatibilité partielle

On dit que P = (S, s_0, F, I_0, L_{insert}, Δ) est partiellement compatible avec P′ = (S', s'_0, F', I'_0, L_{insert}, Δ') et on note P Pcomp P′ ssi :

– $\exists\ (s_0, I_i) \xrightarrow{q_i(I_i)} (s_0,\ I_{i+1})$ alors $\exists\ (s_0', I'_j) \xrightarrow{q'_j(I_j)} (s'_0,\ I'_{j+1})$.

Tel que (s_0, I_i) comp (s'_0,I'_j) et (s_0, I_{i+1}) comp (s'_0, I'_{j+1}). Pour i ∈ [0,..,n], j ∈ [0,..,m] et n, m ∈ ℕ.

Théorème 2

Étant donné deux protocoles de services P = (S, s_0, F, I_0, L_{insert}, Δ) et P′ = (S', s'_0, F', I'_0, L_{insert}, Δ'), le problème de test si P comp P′ est décidable.

Preuve :

Par observation, les séquences d'exécution des protocoles de services avec la requête d'insertion sont finies car les instances de base de données sont finies et dans ce cas les mises à jour ne créent pas de nouvelles valeurs dans adom(I)^n et adom(I')^n tel que "n" est le nombre des attributs et adom(I), adom(I') sont

respectivement les domaines actifs de I et I′. Ce pendant, tester la compatibilité entre P et P′ est décidable.

5.3.3 Le cas 3 : les protocoles de services non-gardés sur une instance fixe I (L_G,∅,∅)^I (requête de suppression)

On reste toujours avec le langage de mise à jour d'une base de données, mais dans ce sous-modèle on va étudier la compatibilité des protocoles de services qui sont représentés avec une même instance de base de données sachant que le passage d'un état à un autre se fait par l'exécution d'une requête de suppression c'est à dire on garde les mêmes paramètres du tableau de la requête d'insertion.

5.3.3.1 La définition du protocole de service

Considérant P comme étant un protocole de service tel que P = (S, s_0, F, I_0, L_{delet}, Δ) sachant que :

- S est un ensemble fini d'états,
- $s_0 \in S$ est l'état initial,
- F est l'ensemble des états finaux, si $F = \emptyset$, le protocole P est appelé un protocole vide,
- I_0 est une instance initiale d'une base de données,
- L_{delet} est un langage de requêtes conjonctives exemple : $q_i(I)$,
- $\Delta \subseteq S \times L_{delet}(I) \times S$ est la fonction de transition.

5.3.3.2 La sémantique de la requête conjonctive

Dans ce cas $\hat{q}(I)$ est interprétée de la manière suivante : le résultat de q(I) sera supprimé de l'instance I $\hat{q}(I) = I'$ tel que $I' = I$ - q(I).

5.3.3.3 La sémantique d'exécution du protocole

Soit le protocole de service P = (S, s_0, F, I_0, L_{delet}, Δ), une séquence d'exécution du protocole est une séquence d'états et d'un ensemble de requêtes conjonctives telle que la configuration des états est représentée par le tuple (s_0,I_i) et elle est de la manière suivante :

$s_0, I \xrightarrow{q(I)} s_0, I'$
tel que $I' = I$ - q(I).

Voici un exemple de protocole qui utilise ce type de requêtes, on suppose qu'on a un schéma de relation R(A,B) avec A et B comme des attributs et I_0 comme étant une instance initiale de R exemple : $I_0 =$

A	B
1	2
2	3
3	4

Le protocole P est défini de la manière suivante (voir figure 5.3) :

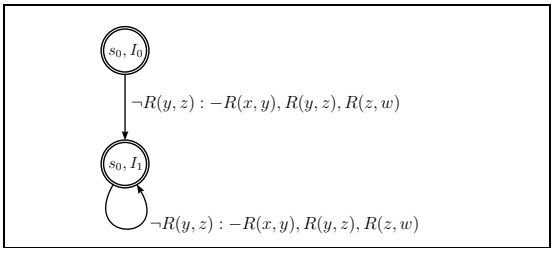


FIGURE 5.3: P avec une requête de suppression

avec $I_1 =$

A	B
1	2
3	4

Une séquence d'exécution complète est une séquence d'exécution qui part d'une configuration initiale et abouti à une configuration finale, exemple dans la figure 5.3 l'état (s_0,I_1) est un état final.

5.3.3.4 L'étude de la compatibilité des protocoles

La compatibilité des protocoles tels qu'ils ont été définis précédemment avec leurs sémantiques d'exécutions peut être étudiée avec l'étude de la compatibilité des séquences d'exécutions complètes. Nous définissons cette compatibilité comme suit :

Définition 5.

On dit qu'un état (s_0,I_i) est compatible avec un état (s'_0, I'_j) et on note (s_0, I_i) comp (s'_0, I'_j) ssi : $I_i = I'_j$.

La compatibilité totale

On dit que P = (S, s_0, F, I_0, L_{delet},Δ) est totalement compatible avec P$'$ = (S', s'_0, F', I'_0, L_{delet},Δ') et on note P Tcomp P$'$ ssi :

– $\forall$ $(s_0, I_i) \xrightarrow{q_i(I_i)} (s_0, I_{i+1})$ alors $\exists$ $(s_0', I'_j) \xrightarrow{q'_j(I_j)} (s'_0, I'_{j+1})$.

Tel que (s_0, I_i) comp (s'_0,I'_j) et (s_0, I_{i+1}) comp (s'_0, I'_{j+1}). Pour i $\in$ [0,..,n], j $\in$ [0,..,m] et n, m $\in \mathbb{N}$.

La compatibilité partielle

On dit que P = (S, s_0, F, I_0, L_{delet}, Δ) est partiellement compatible avec P$'$ = (S', s'_0, F', I'_0, L_{delet}, Δ') et on note P Pcomp P$'$ ssi :

– $\exists$ $(s_0, I_i) \xrightarrow{q_i(I_i)} (s_0, I_{i+1})$ alors $\exists$ $(s_0', I'_j) \xrightarrow{q'_j(I_j)} (s'_0, I'_{j+1})$.

Tel que (s_0, I_i) comp (s'_0,I'_j) et (s_0, I_{i+1}) comp (s'_0, I'_{j+1}). Pour i $\in$ [0,..,n], j $\in$ [0,..,m] et n, m $\in \mathbb{N}$.

Théorème 3

Étant donné deux protocoles de services P = (S, s_0, F, I_0, L_{delet}, Δ) et P$'$ = (S', s'_0, F', I'_0, L_{delet},Δ'), le problème de test si P comp P$'$ est décidable.

Preuve :

Par observation, les séquences d'exécution des protocoles de services avec la requête de suppression sont finies car les instances de base de données sont finies et dans ce cas les mises à jour ne créent pas de nouvelles valeurs dans $\text{adom}(I)^n$ and $\text{adom}(I')^n$ tel que "n" est le nombre des attributs et $\text{adom}(I)$, $\text{adom}(I')$ sont respectivement les domaines actifs de I et I′. Ce pendant, tester la compatibilité entre P et P′ est décidable.

5.3.4 Le cas 4 : les protocoles de services avec une mise à jour Datalog¬

Dans ce sous modèle, on étudie la compatibilité des protocoles de services qui sont représentés avec une même instance de base de données, sachant que le passage d'un état à un autre se fait soit par un envoie de message soit par une mise à jour au niveau de l'instance. Cela se fait selon une condition qui est une requête booléenne : (voir le tableau ci-dessous)

TABLE 5.3: Un protocole P avec le langage datalog

BDD	L_G	L_M	L_U
Finie	Datalog (booléen)	Datalog (envoyer message)	Datalog¬

5.3.4.1 La définition du protocole de service

Considérant P comme étant un protocole de service tel que P = (S, s_0, F, I_0, L_G, L_U, L_M, Δ) sachant que :

- S est un ensemble fini d'états,
- $s_0 \in S$ est l'état initial,
- F est l'ensemble des états finaux, si $F = \emptyset$, le protocole P est appelé un protocole vide,

- I_0 est une instance initiale d'une base de données,
- L_G est un langage Datalog exprimé par une requête booléenne,
- L_U est un langage Datalog de mise à jour,
- L_M est un langage Datalog des messages, c'est à dire le contenu des messages prend la forme suivante m($q_i(I)$) tel que m$\in L_M$ avec un contenu qui est le résultat de la requête q sur l'instance I noté comme suit : $q_i(I)$,
- $\Delta \subseteq S \times L_G(I) \times L_U(I) \times L_M(I) \times S$ est la fonction de transition.

5.3.4.2 La sémantique du langage de mise à jour (L_U)

Dans notre sous-modèle on utilise le programme Datalog$^\neg$ P qui a comme entrée un graphe représenté dans une relation binaire G. Le programme calcule la relation closer (x,y,x′,y′) définie comme suit :
closer (x,y,x′,y′) signifie que la distance d(x,y) allant de x à y dans G est plus petite que la distance d(x′,y′) allant de x′ vers y′
T(x,y)⟵ G(x,y)
T(x,y)⟵ T(x,z), G(z,y)
closer (x,y,x′,y′) ⟵ T(x,y), ¬ T(x′,y′)
q()⟵ closer (x,y,x′,y′).
Le programme est évalué comme suit : Les règles sont déclenchées simultanément avec l'application des valuations. Cela se répète jusqu'à ce qu'on n'aura pas de nouveaux faits. Un fait négatif comme ¬ T(x′,y′) est vrai si T(x′,y′) n'est pas encore inféré. Cela n'empêche pas que T(x′,y′) va être inféré par la suite. La fermeture transitive de G est calculée dans T. Une inférence des règles est appelée une étape d'évaluation du programme, on note aussi que si le fait T(x,y) est inféré à l'étape n alors d(x,y) = n . Donc si T(x′,y′) n'a pas été inféré, on dit que d(x,y)< d(x′,y′) closer (x,y,x′,y′) est inféré et q()= vrai. D'une manière générale on considère qu'on a un programme Datalog$^\neg$ P qui va être évalué et qui donne à la fin une requête booléenne, on suppose que s'il existe dans le programme une règle qui a la forme p ⟵ ..... , ¬ q alors il n'existe pas de chemin allant de p vers q dans le graphe de dépendance.

5.3.4.3 La sémantique d'exécution du protocole

Soit le protocole de service P = (S, s_0, F, I_0 , L_G, L_U, L_M, Δ), une séquence d'exécution du protocole est une séquence d'états et d'un programme Datalog avec une possibilité d'envoyer des messages, la configuration des états est représentée par le tuple (S_i,I_i) et elle est de la manière suivante :

$$s_i, I_i \xrightarrow{q():-P/m(q(I_i))} s_{i+1}, I_{i+1}$$
$$\text{tel que } I_i = I_{i+1}$$

Ou bien

$$s_i, I_i \xrightarrow{q():-P/q(I_i)} s_{i+1}, I_{i+1}$$
$$\text{tel que } I_{i+1} = \mathrm{q}(I_i) \cup I_i$$

5.3.4.4 L'étude de la compatibilité des protocoles

Nous définissons la compatibilité de ce type de protocoles comme suit :

Définition 6.

On dit qu'un état (s_i,I_i) est compatible avec un état (s'_i, I'_i) et on note (S_i, I) comp (s'_i, I'_i) ssi :

$\forall\ (s_i, I_i) \xrightarrow{m(q(I_i))} (s_{i+1}, I_{i+1})$ alors $\exists\ (s_i', I'_i) \xrightarrow{m(q'(I'_i))(\overline{p})} (s'_{i+1}, I'_{i+1})$.

Sachant que $I_i = I'_i$ et $\mathrm{q}(I_i) = \mathrm{q}(I'_i)$

Et (s_{i+1},I_{i+1}) comp (s'_{i+1},I'_{i+1}).

ou bien

$\forall\ (s_i, I_i) \xrightarrow{q(I_i)} (s_{i+1}, I_{i+1})$ alors $\exists\ (s_i', I'_i) \xrightarrow{q'(I'_i)(\overline{p})} (s'_{i+1}, I'_{i+1})$.

Sachant que $I_i = I'_i$ et q(I_i) = q(I'_i)

(s_{i+1},I_{i+1}) comp (s'_{i+1}, I'_{i+1}).

La compatibilité totale

On dit que P = (S, s_0, F, I_0, L_G,L_U,L_M,Δ) est totalement compatible avec P′ = (S', s'_0, F', I'_0, L'_G, L'_U, L'_M,Δ') et on note P Tcomp P′ ssi :

$\forall\ (s_i, I_i) \overset{m(q(I_i))}{\longrightarrow} (s_{i+1}, I_{i+1})$ alors $\exists\ (s_i', I'_i) \overset{m(q'(I'_i))(\overline{p})}{\longrightarrow} (s'_{i+1}, I'_{i+1})$.

Sachant que $I_i = I'_i$ et q(I_i) = q(I'_i)

et (s_{i+1},I_{i+1}) comp (s'_{i+1},I'_{i+1}).

et

$\forall\ (s_i, I_i) \overset{q(I_i)}{\longrightarrow} (s_{i+1}, I_{i+1})$ alors $\exists\ (s_i', I'_i) \overset{q'(I'_i)(\overline{p})}{\longrightarrow} (s'_{i+1}, I'_{i+1})$.

Sachant que $I_i = I'_i$ et q(I_i) = q(I'_i)

(s_{i+1}, I_{i+1}) comp (s'_{i+1}, I'_{i+1}).

et

(s_0, I_0) comp (s'_0, I'_0).

La compatibilité partielle

On dit que P = (S, s_0, F, I_0, L_G,L_U, L_M, Δ) est partiellement compatible avec P′ = (S', s'_0, F', I'_0, L'_G, L'_U, L'_M, Δ') et on note P Pcomp P′ ssi :

$\exists\ (s_i, I_i) \overset{m(q(I_i))}{\longrightarrow} (s_{i+1}, I_{i+1})$ alors $\exists\ (s_i', I'_i) \overset{m(q'(I'_i))(\overline{p})}{\longrightarrow} (s'_{i+1}, I'_{i+1})$.

Sachant que $I_i = I'_i$ et q(I_i) = q(I'_i)

et (s_{i+1}, I_{i+1}) comp (s'_{i+1}, I'_{i+1}).

et

$\exists\ (s_i, I_i) \xrightarrow{q(I_i)} (s_{i+1}, I_{i+1})$ alors $\exists\ (s_i', I_i') \xrightarrow{q'(I'_i)(\overline{p})} (s'_{i+1}, I'_{i+1})$.

Sachant que $I_i = I'_i$ et q(I_i) = q(I'_i)

(s_{i+1}, I_{i+1}) comp (s'_{i+1}, I'_{i+1}).

et

(s_0, I_0) comp (s'_0, I'_0).

Théorème 4

Étant donné deux protocoles de services P = (S, s_0, F, I_0, L_G, L_U, L_M,Δ) et P′ = (S', s'_0, F', I'_0,L'_G,L'_U, L'_M, Δ'), le problème de test si P comp P′ est décidable.

Preuve :

Le nombre d'états des deux protocoles de services est fini car le domaine actif de l'instance est fini (identique aux preuves du théorème 2 et 3).

5.3.5 Le cas 5 général : Les protocoles de services $(L_G, L_M, L_U)^I$

Ce sous-modèle modélise le cas général c'est à dire :

TABLE 5.4: Un protocole P de type $(L_G, L_M, L_U)^I$

BDD	L_G	L_M	L_U
Finie	Langage de requête	Langage de requête	Langage de requête

Définition 7.

Considérant P comme étant un protocole de service tel que P = (S, s_0, F, L_G, L_M, L_U, Δ) sachant que :

- S est un ensemble fini d'états,
- $s_0 \in S$ est l'état initial,
- F est l'ensemble des états finaux, si $F = \emptyset$, le protocole P est appelé un protocole vide,
- L_G est un langage de requêtes pour la condition,
- L_M est un langage de requêtes pour l'envoie de messages,
- L_U est un langage de requêtes pour la mise à jour,
- $\Delta \subseteq S \times L_G \times L_M \times L_U \times S$ est la fonction de transition.

5.3.5.1 La sémantique d'exécution du protocole P

Étant donné un protocole P = (S, s_0, F, L_G, L_M, L_U, Δ), l'exécution de P sur une instance de base de données notée par P(I) est un ensemble d'états (S_i,I_i) sachant que i ∈ [0,n] et (S_0,I_0) comme étant l'état initial. Une transition dans P est un passage d'un état à un autre, elle est définie comme suit :

$s_i, I_i \xrightarrow{\alpha} s_{i+1}, I_{i+1} \in \Delta_I$ ssi :

Cas 1

$\alpha = \mathrm{m}(\mathrm{q}(I_i))$
$s_i \xrightarrow{q_c()/m(q())} s_{i+1}$ sachant que $I_i = I_{i+1}$ et $I_i \models \mathrm{q}_c()$.

Cas 2

$\alpha = \mathrm{q}(I_i)$
$s_i \xrightarrow{q_c()/q()} s_{i+1}$ sachant que $I_{i+1} = I_i \cup \mathrm{q}(I_i)$ et $I_i \models \mathrm{q}_c()$.

Définition 8.

On dit qu'un état (s_i, I_i) dans P est compatible avec un état (s'_i,I'_i) dans P′ et on note (s_i, I_i) comp (s'_i, I'_i) ssi :

$\forall\ s_i, I_i \xrightarrow{\alpha} s_{i+1}, I_{i+1}$
$\exists\ s'_i, I'_i \xrightarrow{\alpha'} s'_{i+1}, I'_{i+1}$ sachant que si :
$\alpha = \mathrm{m}(\mathrm{q}(I_i))$ alors $\alpha' = \mathrm{m}(\mathrm{q}'(I'_i))$ et $I_i = I'_i$ et $\mathrm{q}(I_i) = \mathrm{q}'(I'_i)$ ou
$\alpha = \mathrm{q}(I_i)$ alors $\alpha' = \mathrm{q}'(I'_i)$ tel que $\mathrm{q}(I_i) = \mathrm{q}'(I'_i)$.

Définition 9.

On dit que deux protocoles de services P = (S, s_0, F, L_G, L_M, L_U,Δ) et P′ = (S', s'_0, F', L'_G,L'_M, L'_U, Δ') s'exécutant sur deux instances initiales I_0 et I'_0 sont compatibles ssi : (s_0,I_0) comp (s'_0,I'_0).

Théorème 5

Étant donné un protocole P = (S, s_0, F, L_G, L_M, L_U,Δ) et une instance de base de données I. On dit que P(I) est un système de transitions fini si :

1. Il n'y a pas de création de nouvelles valeurs,
2. l'évaluation est décidable.

preuve 1 :

Selon les définitions 8 et 9, on déduit que si on a pas de nouvelles valeurs crées, alors toutes les valeurs possibles crées par les mises à jour appartiennent à un sous ensemble de (adom^n) tel que "n" est le nombre d'attributs de l'instance I.

preuve 2 :

Dans ce cas, on procède par contradiction. On suppose que l'évaluation du langage de requêtes est indécidable, et selon la définition 7 on ne pourra pas avoir la transition $s_i, I_i \xrightarrow{\alpha} s_{i+1}, I_{i+1}$, qui est une contradiction. Ce pendant, si l'évaluation est décidable, alors P(I) est un système de transitions fini.

5.4 L'analyse des différents types de protocoles de services avec une base de données

Dans la première partie du travail, nous avons étudié la compatibilité des services dans le cas d'une base de données fixe (une instance de base de données), dans ce cas on a trouvé que s'il n'y a pas de création de nouvelles valeurs et si l'évaluation est décidable alors le problème est décidable.
Maintenant on s'intéresse au problème de la compatibilité des données avec une description d'un service orienté données qui intègre en plus des opérations d'émission et de réception de messages des actions sur une base de données. Dans ce cas un état de service prend l'instance actuelle de la base de données et l'état de contrôle de l'automate, sachant que les instances de base de données sont définies par rapport à un domaine infini car le nombre d'états est infini.
Le modèle étudié est donné par le tableau 5.5 suivant :

TABLE 5.5: Un protocole P avec une base de données infinie

BDD	L_G	L_M	L_U
Infinie	requête booléenne	Langage de requête	Langage de requête

5.4.1 préliminaire

On suppose qu'on a un domaine infini de données **Dom** avec l'élément $\perp$ qui représente les valeurs de données indéfinies. On suppose L comme étant un langage de requête, R un schéma de base de données qui est un ensemble fini de schémas de relations, R un schéma de relation avec un ensemble d'attributs distincts, I une base de données dans R, r $\in$ R une relation, q et q' des requêtes, une instance d'un schéma de base de données est une fonction I qui associe pour chaque schéma de relation R une relation finie I(R) sur **Dom**. Une instance ou tuple qui contient l'élément $\perp$ est dite partielle sinon elle est totale.
On suppose que L_U est un langage de mise à jour cela pour définir les requêtes de mise à jour sur la base de données. Dans notre modèle, les requêtes utilisées sont

les requêtes d'insertion, on peut formaliser sous forme d'une expression qui a la forme suivante U = insert R(q) pour dire que le résultat de la requête q est inséré dans R.
Si U est une requête de mise à jour alors U(I) est la base obtenue après l'application de l'insertion U sur I.

5.4.2 La définition d'un protocole de service

Un protocole de service orienté données générique est un automate qui communique avec une base de données globale et une base de données locale, ce service est étiqueté par des conditions qui sont des requêtes booléennes exprimées dans un langage L_G.
Les services communiquent par des messages exprimés dans un langage L_M. Un service peut modifier la base de données locales ou globales par des requêtes d'insertion exprimées dans un langage L_U.

5.4.3 La définition formelle d'un protocole de service

Considérant P comme étant un protocole de service tel que P = (W, M, S,s_0, F, Δ) sachant que :

- W : est un schéma de relation qui représente la disjonction de trois schémas de la manière suivante : $W = W_L \cup W_G \cup W_M$ sachant que :
  - W_L : est le schéma local du service,
  - W_G : est le schéma global visible par tous les services,
  - W_M : est l'ensemble des schémas de message.
- M : est un ensemble de messages, on associe à un message m (p_1,....., p_n) le schéma de relation R_m où le schéma R_m = (A_1,.....,A_n),
- S : est un ensemble d'états de contrôle,
- $s_0 \in S$ est l'état initial,
- F est l'ensemble des états finaux,

– La relation de transition Δ contient le tuple (s,q,μ,s') où s, $s' \in S$, q est une requête booléenne dans L_G définie sur W et μ a la forme suivante :

  – μ = +m (p_1,....., p_n)un message entrant,
  – μ = -m (q) un message envoyé,
  – μ = u sachant que u est une requête exprimée dans L_U définie sur $W_L \cup W_G$.

5.4.4 La sémantique d'exécution d'un protocole de service

La sémantique d'exécution d'un protocole de service P = (W, M, S,s_0, F, Δ) est représentée par un ensemble d'états où chaque état forme une signature sign = (S, I) sachant que S forme l'état de contrôle et I représente l'instance de la base de données sur W. Donc on peut noter que $I = I_L \cup I_G \cup I_M$ sachant que :

– I_L : représente l'instance de la base de données locale,
– I_G : représente l'instance de la base de données globale,
– I_M : représente l'instance de la base de données des messages.

Donc on peut dire qu'une exécution du protocole de service est une séquence finie qui a la forme suivante : $\sigma = sign_0 \xrightarrow{\mu_0} ..... \xrightarrow{\mu_{n-1}} sign_n$ et qui satisfait les conditions suivantes :

– On suppose que $sign_0 = (s_0, I_0)$ est un état initial sachant que $I_0 = I_{0L} \cup I_{0G} \cup I_{0M}$ avec I_{0L}, I_{0G} comme étant des instances aléatoires et I_{0M} est une instance vide.
– $\forall$ i $\in$ [0,i−1] sign (s_i,I_i) $\models$ q.

Exemple

Pour voir les choses clairement, on va étudier un exemple concret, soit un service gestionnaire qui gère les stages des chercheurs dans un institut de recherche scientifique. L'exécution du service est comme suit :

– Un chercheur envoie un message dem-stage(idcher, date, destination, days),
– Le service gestionnaire va vérifier la disponibilité des stages dans une base de données globale appelée chercheur son schéma est le suivant :

- Chercheur (idcher, grade, sta-type)

– Si les conditions sont vérifiées, le service rajoute le tuple dans la base de données validée avec l'attribut statut = « on », la table a le schéma suivant :
  - Valider (idcher, statut, destination)

– Après l'ajout du tuple (idcher, statut, destination), le service envoie un message dem-valid donné par la requête $q_{dem-valid}$.

5.4.5 Étude de la compatibilité des protocoles de services

Soient P et P′ deux protocoles de services tel que p = (W, M, S, s_0, F, Δ) et P′ = (W', M', S', s'_0, F', Δ') avec leurs signatures respectives $sign_i$ et $sign'_i$.
On dit qu'un état $sign_i$ est compatible avec un état $sign'_i$ et on note $sign_i$ comp $sign'_i$ ssi :

– si $sign_i \in F$ alors $sign'_i \in F'$,
– $I_{iG} = I'_{iG}$ (instance de la base de données globale),
– $\forall\ sign_i \xrightarrow{\mu_i} sign_j \in \Delta$ alors $\exists\ sign'_i \xrightarrow{\mu'_i} sign'_j \in \Delta'$ sachant que :
  - Si $\mu_i = +m(r_m)$ alors $\mu'_i = -m(r_m)$ et $sign_j$ comp $sign'_j$.
  - Si $\mu_i = -m(q)$ alors $\mu'_i = +m(q')$ et $q(I_{iG}) = q(I'_{iG})$
  - Si $\mu_i = u$ alors $\mu'_i = u'$ et $u(I_{iG}) = u(I'_{iG})$ et $sign_j$ comp $sign'_j$.

Définition 9.

On dit que deux protocoles de services P et P ′ sont compatibles et on note P comp P′ ssi $\forall\ sign_0$ dans P, $\exists\ sign'_0$ dans P′ tel que $sign_0$ comp $sign'_0$.

5.4.6 L'analyse des différentes classes du modèle

Dans cette partie, on va étudier la décidabilité de la compatibilité des protocoles de services selon le modèle vu précédemment. Pour cela on va découper notre modèle en plusieurs sous modèles selon les langages existants.

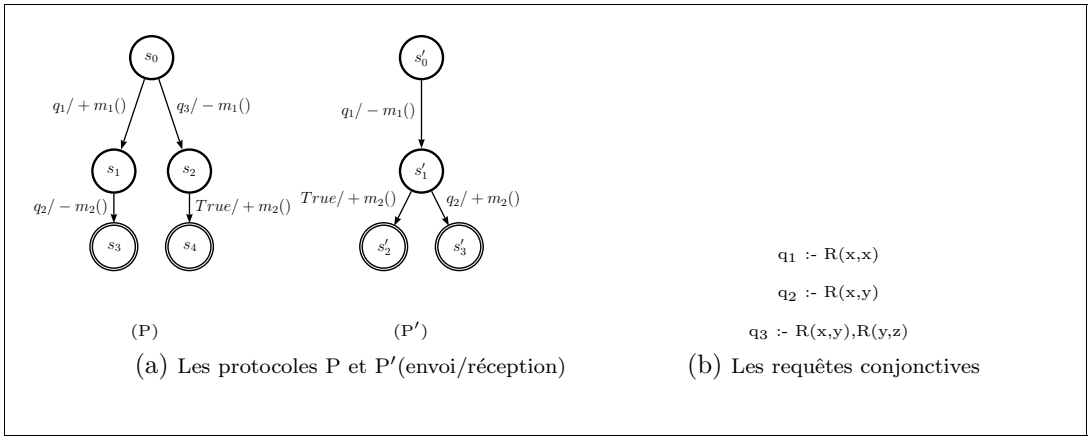
5.4.7 Cas 1 : Un protocole de type $(L_G,\emptyset,\emptyset)^{DB}$

TABLE 5.6: Un protocole de type $(L_G,\emptyset,\emptyset)^{DB}$

L_G	L_M	L_U	type d'action	W_G	W_L
CQ	$\emptyset$	$\emptyset$	envoi et réception	+	-

Ce tableau définit le sous modèle qui opère seulement sur une base de données globale car $W_G = +$ et qui peut envoyer et recevoir un message vide car le $L_M = \emptyset$ avec bien sur un langage guard L_G = CQ.

Le schéma suivant montre l'interaction entre deux services décrits selon le premier cas c-à-d l'envoie et la réception de messages avec un langage guard décrit par des requêtes conjonctives (voir figure 5.4).



(a) Les protocoles P et P′(envoi/réception) (b) Les requêtes conjonctives

FIGURE 5.4: Un protocole de service gardé

Le test de compatibilité entre les protocoles de services (L_G, $\emptyset$, $\emptyset$) est décidable ssi la vérification de la satisfiabilité de la formule exprimée en $L_G \cup \{ \wedge^b, \neg^b \}$ est décidable, sachant que L_G est le langage des requêtes booléennes et $\wedge^b$, $\neg^b$ sont des opérateurs logiques pour définir les formules exprimées en L_G.

Définition 10.

On note par tree($sign_0$) tout arbre d'exécution d'un protocole de service qui a comme racine $sign_0$.

Le test de compatibilité est décidable car chaque arbre d'exécution (éventuellement infini) tree($sign_0$) du protocole (L_G, $\emptyset$, $\emptyset$) peut être représenté par une machine à états finis FSM_{sign_0}(P).
Donc la vérification de la compatibilité peut être réduite vers une vérification entre deux machines à états finis FSM_{sign_0}(P) et FSM_{sign_0}(P′).

Définition 11.

Soit P = (W, M, S, s_0, F, Δ) un protocole de service (L_G, $\emptyset$, $\emptyset$). On suppose que I est une instance dans W_G. On note par FSM_I (P)= (W,σ_I,S,s_0, F,Δ_I) générée en utilisant l'instance sachant que :

- Chaque message dans P va être transformé en une chaine de caractère c_m dans la FSM_I (P) avec une polarité soit positive (+) ou négative (-),
- Δ_I = { (s_i, +/- c_m,s_{i+1}) | ($\exists$ (s_i,q,+m(),s_{i+1}) $\cup$ $\exists$(s_i,q,-m(),s_{i+1})) $\in \Delta$ et q(I) = True },
- $\forall$ s $\in \Delta_I$, il existe un chemin allant d'un état initial s_0 vers un état final qui passe par l'état s,
- σ_I représente l'alphabet obtenu en renommant les messages.

On note par tree(FSM_I(P)) l'arbre d'exécution (infini) du FSM_I(P).
Pour définir l'algorithme qui teste la compatibilité des protocoles de services, on doit éliminer le nombre de tests infinis. On suppose qu'on a deux protocoles de services P et P′ avec un schéma de base de données globale W_G, pour ce problème l'idée est de partitionner l'ensemble infini d'instances sur W_G en un ensemble fini de partitions sachant que :

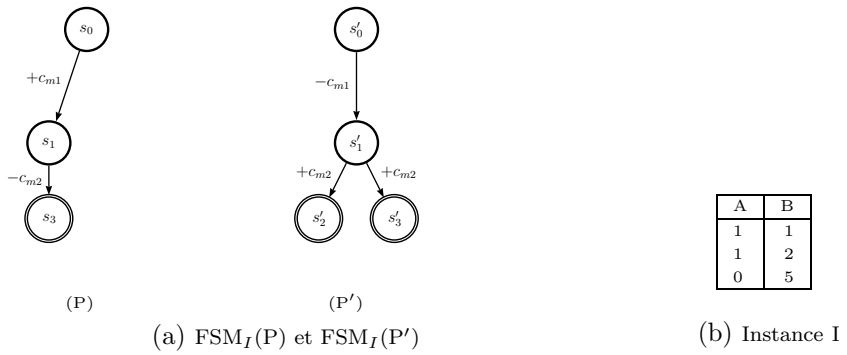
- Le nombre de partitions est fini (mais une partition peut représenter un nombre infini d'instances sur W_G),

- La compatibilité entre P et P′ peut être réduite vers un ensemble de test de compatibilité entre les partitions,
- Le test de compatibilité entre deux partitions est décidable car il peut être transformé en un test de compatibilité entre deux machines à états finis.

Lemme 1

Soit P = (W, M, S, s_0, F, Δ) un protocole de service (L_G, $\emptyset$, $\emptyset$) avec $\text{sign}_0 = (\text{s}_0, \text{I}, \emptyset)$ sachant que I est une instance aléatoire dans W_G, $\text{tree}(\text{sign}_0) \cong \text{tree}(\text{FSM}_I(\text{P}))$.

Exemple :



(a) FSM_I(P) et FSM_I(P′)

A	B
1	1
1	2
0	5

(b) Instance I

FIGURE 5.5: FSM_I(P) et FSM_I(P′)

preuve :

Le lemme 1 est une conséquence directe de la définition 11.
étant une conséquence directe du lemme 1, la compatibilité entre deux protocoles de services (L_G, $\emptyset$, $\emptyset$) P et P′ peut être reformulé comme suit :

Lemme 2

Soient deux protocoles de services P et P′ dans (L_G, $\emptyset$, $\emptyset$) sur un schéma global W_G. On dit que P comp P′ ssi FSM_I(P) comp FSM_I(P′).

Définition 12.

Soit H un ensemble de requêtes booléennes dans L_G sur le schéma de base de données W_G et h $\subseteq$ H. Soit I_W un ensemble infini de toutes les instances possibles dans L_G. On note par q_H^h la formule obtenue par la conjonction des requêtes dans h et la négation des requêtes qui ne sont pas dans h et qui sont dans H c-à-d $q_H^h := (\bigwedge_{q \in h} q) \wedge (\bigwedge_{q \in H \setminus h} \neg q)$.

Définition 13.

Soit q_H^h une formule , on note par $P_h(I_W) = \{ I \in I_W \mid q_H^h (I) = True \}$ l'ensemble des instances dans I_W qui satisfaisaient la formule q_H^h, et on note par $P_H = \{P_h(I_W) \mid h \in 2^H \}$ l'ensemble qui forme les partitions de I_W sachant que :
- $\forall$ h, h$' \in 2^H$, avec h$\neq$ h$'$, on a $q_H^h \wedge q_H^{h'}$ est insatisfaisable donc $P_h(I_W) \cap P_{h'}(I_W) = \emptyset$.
-La requête $\bigvee_{h \in 2^H} q_H^h$ est valide donc $\bigcup_{h \in 2^H} P_h(I_W) = I_W$.

Lemme 3

Soit un protocole de service P dans la classe (L_G, $\emptyset$, $\emptyset$) avec un schéma W_G et soit H^P l'ensemble des requêtes booléennes utilisées comme requêtes de condition (guard) dans P. Alors $h \subseteq H^P$, on a $\forall$ I, I$' \in P_h(I_W)$, $FSM_I(P) \cong FSM_{I'}(P)$.

Preuve :

Le lemme est une conséquence directe des définitions 11, 12 et 13.
Soit une partition donnée $P_h(I_W)$ qui satisfait les conditions du lemme 3, on note par $FSM_{P_h(I_W)}$ (P) le FSM des instances de I_W contenues dans la partition $P_h(I_W)$.

Lemme 4

Soit deux protocoles de services P et P$'$ dans (L_G, $\emptyset$, $\emptyset$) sur le même schéma de base de données W_G et soit H l'ensemble des requêtes booléennes utilisées comme conditions dans P et P$'$. Alors on dit que P est totalement compatible avec P$'$ et on note P Tcomp P$'$ ssi $\forall$ $P_h(I_W) \in P_H$ tel que $P_h(I_W)$ est non vide, on a $FSM_{P_h(I_W)}$ (P) comp $FSM_{P_h(I_W)}$ (P$'$).

Lemme 5

Soit deux protocoles de services P et P′ dans (L_G, $\emptyset$, $\emptyset$) sur le même schéma de base de données W_G et soit H l'ensemble des requêtes booléennes utilisées comme conditions dans P et P′. Alors on dit que P est partiellement compatible avec P′ et on note P Pcomp P′ ssi $\exists$ $P_h(I_W) \in P_H$ tel que $P_h(I_W)$ est non vide, on a $FSM_{P_h(I_W)}$ (P) comp $FSM_{P_h(I_W)}$ (P′).

Preuve :

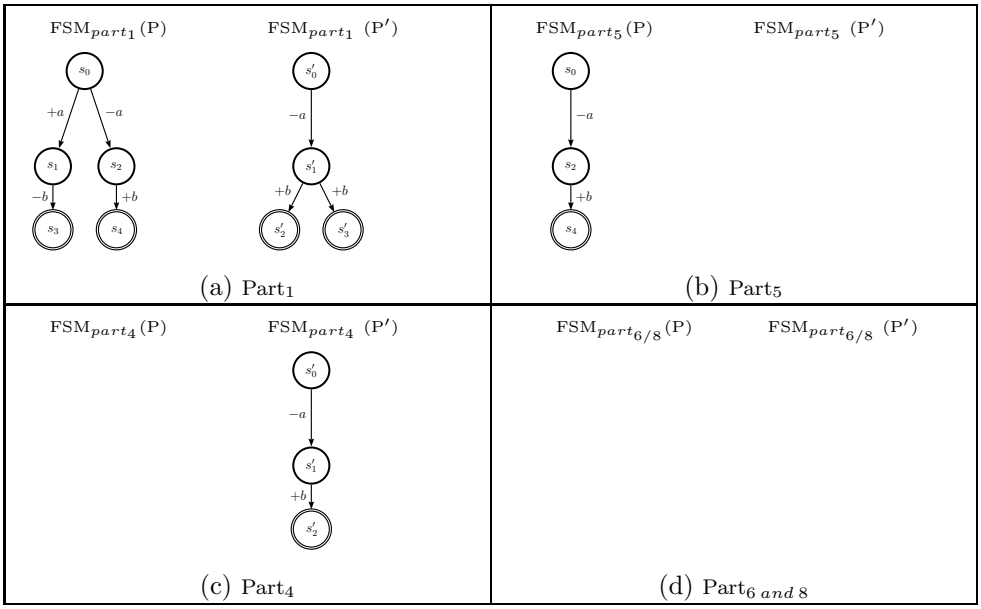
Les lemmes 4 et 5 sont des conséquences directes des lemmes 2 et 3 .

Exemple

Soient P et P′ les deux protocoles de services définis dans la figure 5.4. Alors l'ensemble H = { q_1,q_2,q_3 }, on peut déduire que l'ensemble P_H contient les partitions suivantes :

- Part 1 = $q_1 \wedge q_2 \wedge q_3$.
- Part 2 = $q_1 \wedge q_2 \wedge \neg q_3$ insatisfaisable car $q_1 \subseteq q_3$.
- Part 3 = $q_1 \wedge \neg\, q_2 \wedge q_3$ insatisfaisable car $q_3 \subseteq q_2$ et $q_1 \subseteq q_2$.
- Part 4 = $q_1 \wedge \neg\, q_2 \wedge \neg\, q_3$.
- Part 5 = $\neg\, q_1 \wedge q_2 \wedge q_3$.
- Part 6 = $\neg\, q_1 \wedge q_2 \wedge \neg\, q_3$.
- Part 7 = $\neg q_1 \wedge \neg\, q_2 \wedge q_3$ insatisfaisable car $q_3 \subseteq q_1$.
- Part 8 = $\neg q_1 \wedge \neg q_2 \wedge \neg q_3$ satisfaite par l'instance vide.

On note que les partitions satisfaisables forment l'ensemble suivant : { part 1, part 4, part 5, part 6, part 8}.
La figure 5.6 montre les différentes partitions et leurs $FSM_{P_h(I_W)}$.



(a) Part$_1$ (b) Part$_5$ (c) Part$_4$ (d) Part$_{6\ and\ 8}$

FIGURE 5.6: Les FSM$_{part_i}$(P)et FSM$_{part_i}$(P′) des partitions satisfaisables

Selon le lemme 5, on peut dire que P Pcomp P′.

Lemme 6

La compatibilité entre deux protocoles de services gardés est décidable si la vérification de la satisfiasabilité des requêtes associées aux partitions est décidable (partitions non vides), telle que la requête associée à la partition est une formule exprimée en $L_G \cup \{ \wedge^b, \neg^b \}$.

Preuve :

Le lemme est une conséquence directe du lemme 2 et 3.

Lemme 7

Soit F une formule exprimée en $L_G \cup \{ \wedge^b, \neg^b \}$. Alors il existe deux protocoles de services P et P′ dans (L_G, $\emptyset$, $\emptyset$) telle que la formule F est satisfaisable ssi P n'est pas compatible avec P′.

Preuve :

Soit F une formule qui a la forme suivante : F = $q_1() \wedge ... \wedge q_i() \neg q_{i+1}() \wedge ..... \wedge \neg q_n()$ où chaque q_j est une requête booléenne exprimée dans L_G tel que j∈ [1,n], et pour chaque k ∈ [1,i] q_k représente une requête positive. Dans ce cas, on peut construire la requête booléenne $q^+ = \bigwedge_{k\in[1,i]} q_k()$. On suppose que le protocole de service P passe de l'état initial s_0 vers un état final s_1 par la transition gardée par la requête q^+ et avec un envoie de message +m() et un protocole de service P′ passe de l'état initial s'_0 vers des états finaux par les transitions gardées par les requêtes négatives q_k tel que k ∈ [i+1,n] et avec une réception de message -m(). P n'est pas compatible avec P′ ssi il existe une instance I sachant que I $\models q^+$ et pour chaque k ∈ [i+1,n] I $\not\models q_k$. C'est le cas où la formule F est satisfaisable. Donc La compatibilité dans (L_G, ∅, ∅) est indécidable si la satisfiasabilité dans $L_G \cup \{ \wedge^b, \neg^b \}$ est indécidable.

Théorème 6

La compatibilité des protocoles de services dans (L_G, ∅, ∅) est décidable ssi la vérification de la satifiasabilité de la formule dans $L_G \cup \{ \wedge^b , \neg^b \}$ est décidable.

5.4.8 Cas 2 : Un protocole de type $(\emptyset,L_G,\emptyset)^{DB}$

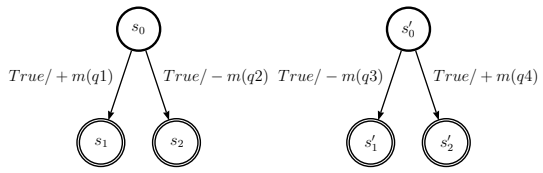
TABLE 5.7: Un protocole de type $(\emptyset,L_G,\emptyset)^{DB}$

L_G	L_M	L_U	type d'action	W_G	W_L
∅	CQ	∅	envoi et réception	+	-

Ce tableau montre le sous modèle qui définit les protocoles de services non gardés (∅, L_M, ∅), donc ce protocole de service est capable d'envoyer et de recevoir un message qui a comme contenu le résultat d'une requête qui opère sur un schéma de base de données globale W_G. On suppose qu'un message peut contenir le résultat de plusieurs requêtes.

Exemple

La figure 5.7 montre deux protocoles de services P et P′ tel que P envoie le résultat de la requête q_1 et reçoit le résultat de la requête q_2 alors que P′ reçoit le résultat de la requête q_3 et envoie le résultat de la requête q_4. On suppose que le langage L_M = CQ (voir figure 5.7).



(a) Les protocoles P et P′(envoi/réception)

$q_1 : q_1(x) :- R(x,y), R_1(z,w)$

$q_2 : q_2(x) :- S(x,y)$

$q_3 : q_3(x) :- R(x,y)$

$q_4 : q_4(x) :- S(x,y), R_1(z,w)$

(b) Les requêtes conjonctives

FIGURE 5.7: Deux protocoles de type ($\emptyset$, L_M, $\emptyset$)

On peut dire que P comp P′ ssi $\forall$ I $q_1(I) = q_3(I)$ ou $q_2(I) = q_4(I)$ (on prend uniquement les cas où la polarité est différente). On voit que la compatibilité ne peut pas être réduite en un test d'équivalence entre une disjonction de requêtes. En effet dans l'exemple de la figure 5.7, il y a une compatibilité par rapport aux requêtes q_1 et q_3 mais $q_1 \not\equiv q_3 \cup q_4$.

Soient deux protocoles de services P et P′ dans ($\emptyset$, L_M, $\emptyset$), chaque arbre d'exécution tree($sign_0$)(P)(resp.tree($sign_0$)(P′) des protocoles P (resp. P′) peut être représenté par une machine à états finis FSM_I(P)(resp. FSM_I(P′)) où les transitions sont étiquetées par les résultats des requêtes sur I. Alors la vérification de la compatibilité entre FSM_I(P) et FSM_I(P′) est équivalente à la vérification de l'égalité des résultats des requêtes mais avec une polarité différente et sur la même base de données. Cette transformation est applicable car il n' y a pas de changement au niveau de la base de données.

Définition 14.

Soit P = (W, M, S, s_0, F, Δ) un protocole de service ($\emptyset$, L_M, $\emptyset$). On suppose que I est une instance dans W_G. On note par FSM_I (P)= (W,Σ_I,S,s_0, F, Δ_I) la machine à états finis générée sur l'instance I sachant que :

- Δ_I est la relation de transition qui remplace chaque message (pol)m(q) par le résultat de la requête (pol)q() sur I tel que Δ_I = { (s_i,(pol) q(I),s_{i+1}) | ($\exists$ (s_i,True,(pol)m(q),s_{i+1})},
- Σ_I = { (pol)q(I)| q est une requête dans P } sachant que :
  - (pol)q(I)= +q(I) si on a un envoie de message avec le résultat de la requête q,
  - (pol)q(I)= -q(I) si on a une réception de message avec le résultat de la requête q.

Lemme 9

Soit P = (W, M, S, s_0, F, Δ) un protocole de service ($\emptyset$, L_M, $\emptyset$) et $sign_0$ = (s_0, I , $\emptyset$) une configuration initiale de P avec une instance de base de données I aléatoire sur W_G, alors tree ($sign_0$) $\cong$ tree(FSM_I(P)).

Preuve :

Le lemme 9 est déduit de la construction du FSM_I(P) de la définition 14.

Lemme 10

Soient P et P′ deux protocoles de services dans ($\emptyset$, L_M, $\emptyset$) et sur un même schéma global W_G, alors P comp P′ ssi $\forall$ I $\in$ W_G, on a FSM_I(P) comp FSM_I(P′). Le nombre d'instances sur W_G est infini, donc le nombre des FSM_I(P) et FSM_I(P′) est infini. Pour résoudre le problème, on doit regrouper le nombre infini de FSM dans un ensemble fini.
Soit Ω l'ensemble des requêtes dans P et P′ et $Part$ = { b_1,b_2,....,b_n } une partition de Ω, alors b_i est un sous ensemble dans Ω et tous les b_i sont deux à deux disjoints.

Définition 15.

Soient P et P′ deux protocoles de services dans ($\emptyset$, L_M, $\emptyset$) et sur un même schéma global W_G. Soit Ω l'ensemble des requêtes qui opèrent sur P et P′ et *Part* est une partition dans Ω alors :
On note par $I_W(Part)$ l'ensemble des instances incluses dans la partition sachant que $I_W(Part) = \{ I \in I_W \mid \forall\ q_j, q_k \in b_i$ alors $q_j(I) = q_k(I)$ et $\forall\ q_j \in b_i$ et $q_k \in b_l$ alors $q_j(I) \neq q_k(I)$ tel que $i \neq l$ et $i,l \in [1,n]\}$.
Une instance I dans W_G appartient à une partition *Part* dans Ω si et seulement si les résultats des requêtes q sur I et dans le même b_i sont égaux et les résultats des requêtes sur I et dans le même b_l sont différents tel que b_i est différent de b_l pour $i,l \in [1,n]$.
On note par P_Ω l'ensemble des partitions de Ω. Une instance ne peut pas être dans deux partitions différentes et une partition peut être vide.
On associe pour chaque partition une fonction (une formule) qui va décider si la partition est satisfaisable ou non.

Définition 16.

Soient $P_\Omega = \{\ Part_1,.....,Part_m\ \}$ l'ensemble des partitions de Ω, $Part = \{\ b_1,b_2,...,b_n\ \}$ une partition dans P_Ω, Ω_P et $\Omega_{P'}$ les ensembles des requêtes qui opèrent respectivement dans P et P′. On note par f_{Part} la fonction qui prend comme entrée la partition $Part_i$ et décide si cette partition et satisfaisable ou non. La fonction f_{Part} est la suivante :

f_{Part} (*Part*)
Var *result* := False ;
Debut
if $\bigwedge_{i=1}^{n} b_i = \emptyset$ and $\exists\ q_j$, $q_k \in b_i$ tel que $q_j \in \Omega_P$ et $q_k\ \Omega_{P'}$
then *result* := True ;
return (*result*) ;
end.

Exemple

Si on prend l'exemple de la figure 5.7 (vue précédemment) et selon la définition 15 l'ensemble P_Ω est :
-$Part_{\ 1} = \{ (q_1), (q_2), (q_3) , (q_4) \}$
-$Part_{\ 5} = \{ (q_1,q_3), (q_2, q_4) \}$

Lorsqu'on applique la fonction $f_{Part}(Part_{\ 1})$, on trouve que la partition $Part_{\ 1}$ n'est pas satisfaisable car la fonction renvoie (false).
Après la détermination des fonctions satisfaisables, on applique la transformation vers la machine à états finis FSM_{Part_5} et on trouve le résultat suivant (voir figure 5.8) :

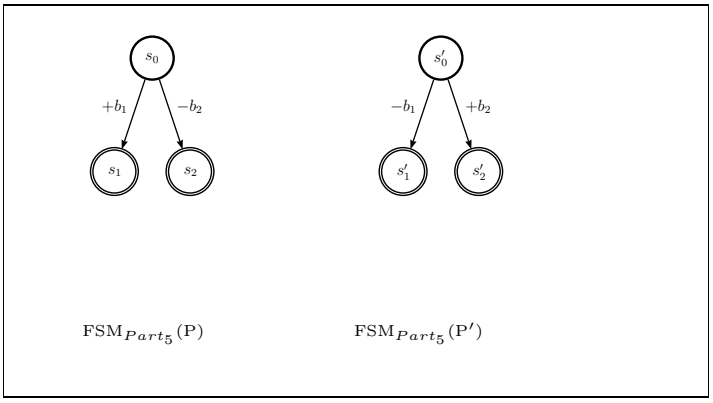


FIGURE 5.8: La $\text{FSM}_{Part_5}(\text{P})$ et $\text{FSM}_{Part_5}(\text{P}')$

Définition 17.

Soient $\text{P} = (W, M, S, s_0, F, \Delta)$ et $\text{P}' = (W, M', S', s'_0, F', \Delta')$ deux protocoles de services dans $(\emptyset, L_M, \emptyset)$. Soit Ω l'ensemble des requêtes qui opèrent dans les deux services et $Part = \{ \text{b}_1,\text{b}_2,....,\text{b}_n \}$ est une partition dans Ω. Alors $\text{FSM}_{Part}(\text{P}) = (\Sigma_{Part}, S, s_0, F, \Delta_{Part})$ est une machine à états finis sachant que :

- $\Delta_{Part} = \{ (\text{s} , +\text{b}_i,\text{s}') \mid \exists (\text{s,True, + m}(\text{q}_j), \text{s}') \in \Delta \text{ et } \text{q}_j \in \text{b}_i \}$ ou $\{ (\text{s} , \text{-b}_i,\text{s}') \mid \exists (\text{s,True, - m}(\text{q}_j), \text{s}') \in \Delta \text{ et } \text{q}_j \in \text{b}_i \}$,
- $\Sigma_{Part} = \{ \text{b}_1,\text{b}_2,....,\text{b}_n \} \cup \{ +, - \}$.

Comme les résultats des requêtes dans le même b_i sont égaux alors on peut réduire le test de compatibilité entre deux protocoles de services à un test de compatibilité entre deux machines à états finis qui représentent les partitions.

Lemme 11

Soit P un protocole de service dans ($\emptyset$, L_M, $\emptyset$) et *Part* une partition dans Ω, alors pour chaque instance sur W_G sachant que I $\in$ *Part*, $FSM_I(P) \cong FSM_{Part}$ (P).

Preuve :

Le lemme 11 est une conséquence directe de la définition 17.

Lemme 12

Soient P et P′ deux protocoles de services dans ($\emptyset$, L_M, $\emptyset$) sur W_G, Soit P_Ω l'ensemble des partitions. Alors P comp P′ ssi $\forall$ *Part* $\in P_\Omega$ sachant que *Part* n'est pas vide on a FSM_{Part} (P) comp FSM_{Part} (P′).

Preuve :

Le lemme 12 est une conséquence directe des lemmes 11 et 10.

Théorème 7

La vérification de la compatibilité entre deux protocoles de services dans ($\emptyset$, L_M, $\emptyset$) est décidable si le résultat de la fonction f_{Part} sur une partition *Part* est décidable.

Preuve

le théorème 7 est une conséquence du Lemme 12.

Remarque 1

On peut aussi remplacer la fonction f_{Part} définie dans la définition 16 par une formule f_{Part} associée à la la partition $Part_i$ qui est construite comme suit :

$$f_{Part} = \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{|b_i|} (q_{i1} = q_{ij}) \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} (q_{i1} = q_{j1})$$

Le théorème 7 redevient comme suit :

Théorème 7

La vérification de la compatibilité entre deux protocoles de services dans ($\emptyset$, L_M, $\emptyset$) est décidable si la vérification de la satisfaisabilité de la formule f_{Part} sur une partition *Part* est décidable.

Remarque 2

On peut aussi atteindre le théorème 7 en partant de la définition 16 et en procédant de la manière suivante :

Définition 16.

Soient P_Ω = { $Part_1$,.....,$Part_w$ } l'ensemble des partitions de Ω, $Part_j$ = { b_1,b_2,...,b_n } une partition dans P_Ω, b_i = { q_1,...., q_m } $\in Part_j$ un sous-ensemble de la partition $Part_j$ tel que i $\in$ [1,m] et j $\in$ [1,w] et Ω_P, $\Omega_{P'}$ les ensembles des requêtes qui opèrent respectivement dans P et P′.
On note par Q_{b_i} = ($q_1 \vee q_2 \vee ....\vee q_m$) la formule booléenne qui représente le sous-ensemble b_i sachant que :
- On associe à chaque requête dans Q_{b_i} la valeur "vrai" et donc Q_{b_i} = vrai ssi $\forall$ q_l , $q_k \in b_i$ alors $q_l(I) = q_k(I)$ et $\forall$ $q_l \in b_i$ et $q_k \in b_j$ alors $q_l \neq q_k$ sachant que i≠j et i,j$\in$ [1,n].
- On note par b_i = vrai (resp b_i = faux) ssi Q_{b_i} = vrai (resp Q_{b_i} = faux).

Définition 17.

Soit Z_{Part} une variable booléenne associée à la partition *Part* sachant que Z_{Part} = vrai ssi $\exists$ $q_l \in b_i$ i $\in$ [1,n] (La partition n'est pas vide).

Définition 18.

Soit $f_{Part} = \bigwedge_{i=1}^{n} \mathrm{b}_i \wedge \mathrm{Z}_{Part}$ une formule booléenne qui vérifie la satisfaisabilité de la partition *Part* sachant que :

- Si f_{Part} = vrai alors on dit que la partition *Part* et satisfaisable par la formule f_{Part} .
- Sinon on dit que la partition *Part* est insatisfaisable par la formule.

Exemple

Si on prend l'exemple de la figure 5.7 (vue précédemment) et selon la définition 15 l'ensemble P_Ω est :
-*Part* $_1$ = { (q_1), (q_2), (q_3) , (q_4) }
-*Part* $_5$ = { (q_1,q_3), (q_2, q_4) }

Lorsqu'on applique la formule f_{Part}(*Part* $_1$) on trouve que la partition *Part* $_1$ n'est pas satisfaisable car la formule renvoie (faux).
Après la détermination des partitions satisfaisables , on applique la transformation vers la machine à états finis FSM_{Part_5} et on trouve le résultat présenté sur la figure 5.8.

Définition 19.

Soient P = (*W*,*M*,*S*,s_0, *F*,Δ) et P′ = P = (*W*,*M*′,*S*′,s'_0, *F*′,Δ') deux protocoles de services dans ($\emptyset$, L_M, $\emptyset$). Soit Ω l'ensemble des requêtes qui opèrent dans les deux services et *Part* = { b_1,b_2,....,b_n } est une partition dans Ω . Alors FSM_{Part}(P) = (Σ_{Part},*S*,s_0, *F*,Δ_{Part}) est une machine à états finis sachant que :

- Δ_{Part} = { (s , +b_i,s′) | ∃ (s,True, + m(q_j), s′ ∈ Δ et q_j ∈ b_i } ou { (s , -b_i,s′) | ∃ (s,True, - m(q_j), s′ ∈ Δ et q_j ∈ b_i }
- Σ_{Part} = { b_1,b_2,....,b_n } × { +, - }.

Comme les résultats des requêtes dans le même b_i sont égaux alors on peut réduire le test de compatibilité entre deux protocoles de services à un test de compatibilité entre deux machines à états finis qui représentent les partitions.

Lemme 11

Soit P un protocole de service dans ($\emptyset$, L_M, $\emptyset$) et *Part* une partition dans Ω , alors pour chaque instance sur W_G sachant que I $\in$ *Part* $FSM_I(P) \cong FSM_{Part}$ (P).

Preuve :

Le lemme 11 est conséquence directe de la définition 19.

Lemme 12

Soient P et P′ deux protocoles de services dans ($\emptyset$, L_M, $\emptyset$) sur W_G , Soit P_Ω l'ensemble des partitions. Alors P comp P′ ssi $\forall$ *Part* $\in P_\Omega$ sachant que *Part* n'est pas vide, on a FSM_{Part} (P) comp FSM_{Part} (P′).

Preuve

Le lemme est une conséquence directe des lemmes 11 et 10.

Théorème 7

La vérification de la compatibilité entre deux protocoles de services dans ($\emptyset$, L_M, $\emptyset$) est décidable si le résultat de la formule f_{Part} sur une partition *Part* est décidable.

Preuve :

le théorème 7 est une conséquence du Lemme 12.

5.5 Conclusion

Dans ce chapitre, nous avons effectué une étude approfondie sur la compatibilité des protocoles de services avec l'ajout d'une base de données. Cette étude nous a permis de voir clairement les différents cas possibles en jouant sur les langages utilisés pour représenter les protocoles de services.
Dans le prochain chapitre, on va vous présenté nos deux applications réalisées en java. Un éditeur d'automate permettant de créer et de modifier des protocoles

existants et un vérifieur de compatibilité de type (L_G,ϕ,ϕ)^I qui permet d'analyser la compatibilité entre deux protocoles donnés.

“ Si debugger, c’est supprimer des bugs, alors programmer ne peut être que les ajouter.

Edsger Dijkstra

6

Applications et résultats

6.1 Introduction

Après avoir analyser formellement la compatibilité entre les différents types de protocoles. Dans le présent chapitre, nous avons enrichi notre travail par la conception et l'implémentation d'un vérifieur des protocoles gardés de type (L_G,ϕ,ϕ)^I. Ce vérifieur permet de vérifier la compatibilité entre deux systèmes de transitions générés par un générateur d'automate. Dans la suite de ce chapitre, nous allons voir les différentes étapes de développement et nous clôturons ce chapitre par des captures d'écrans qui nous montrent les différentes phases de test.

6.2 Choix du langage d'implémentation

En suivant une démarche classique, la première étape pour la mise en œuvre de nos outils consiste à choisir le langage de programmation. A cette fin nous avons choisi le langage java sous l'environnement de développement Netbeans. C'est vrai qu'on aurait pu choisir un langage fonctionnel tel que Haskell et cela pour plusieurs raisons. L'une des raisons principales réside dans l'utilisation des structures de listes. Entre autre, la liste des états, la liste des transitions, etc. Mais enfin de compte nous avons opté pour Java car :

- Java est un langage *full object* c'est-à-dire contrairement à d'autres langages de programmation tel que le C++, Java respecte une approche de programmation entièrement orientée objets et cela va nous aidé à réfléchir d'une manière orientée objets,
- Java est portable c'est-à-dire un programme écrit en java sur n'importe quel système peut être exécuté sans aucune modification sur un autre système à condition qu'un environnement d'exécution (une machine virtuelle) soit disponible sur ce dernier,
- Java est un langage interprété c'est-à-dire un programme écrit en Java sera exécuté par un interpréteur qui traduit en temps réel les instructions écrites en Java en instructions exécutables par le système hôte. D'une manière, une source Java va être transformée en un fichier qui sera interprété par une machine virtuelle,

– Java est doté de plusieurs APIs [1] utiles pour le développement de n'importe quelle application.

6.3 Un vérifieur de compatibilité des protocoles gardés de type (L_G,ϕ,ϕ)^I

Dans la pratique, on a développé un vérifieur générique de compatibilité des protocoles de services gardés qui opèrent sur une instance de base de données I, ce vérifieur est noté par $\text{Ver}^{(L_G,\phi,\phi)^I}$. Le framework accepte comme entrée deux protocoles de services gardés (L_G,ϕ,ϕ) modélisés par notre éditeur qui a été aussi développé en Java (voir Figure 6.1 qui montre l'interface de l'éditeur). Pour des raisons de complexité qui augmente avec l'augmentation du nombre des requêtes, le type et les tuples des données. La réalisation d'un vérifieur nécessite une représentation symbolique d'une instance de base de données I modifiable contenant deux attributs Att_1 et Att_2 qui ont le type numérique. Pour des raisons de flexibilité et pour être capable de pouvoir essayer plusieurs modèles d'automates, on utilise comme requêtes de condition trois types de requêtes conjonctives booléenne :

– Q_1 :- I(x,x)
– Q_2 :- I(x,y)
– Q_3 :- I(x,y), I(y,z)

Notre framework de vérification est divisé en quatre étapes :

1. ***La transformation de l'automate :*** dans cette étape, les automates en entrée vont être transformés en matrices de tailles (n x n) tel que n représente le nombre des états et chaque case dans la matrice représente la transition qui relie l'état de l'automate représenté par la ligne adéquate avec celui représenté par la colonne,
2. ***La minimisation :*** cette étape est caractérisée par la transformation des matrices obtenues dans la première étape vers des matrices avec des transitions valides c'est-à-dire la génération de nouvelles matrices après l'invocation des requêtes booléennes,

1. Application Programming Interface

3. ***La création des arbres complets :*** cette étape garde uniquement les arbres d'exécutions complets cela en vérifiant les états finaux,
4. ***Tester la compatibilité :*** Dans cette étape, on définie trois classes de compatibilité (La compatibilité totale, la compatibilité partielle et l'incompatibilité), après on propose un algorithme qui teste la compatibilité entre deux (L_G,ϕ,ϕ) protocoles donnés sur I et affiche comme résultat la classe de compatibilité adéquate. L'algorithme est définit comme suit :

Algorithm 1 compatibility test

Require: P $=(L_G, \emptyset, \emptyset)^I$, P$'$ $= (L_G, \emptyset, \emptyset)^I$, I
Ensure: $result \leftarrow Null$
if (s_0, I) comp (s_0', I) and $(\forall\ (s_i, I) \overset{q()/m(p)}{\longrightarrow} (s_j, I)$ then $\exists\ (s_i', I) \overset{q'()/m(\overline{p})}{\longrightarrow} (s_j', I)$ and (s_j, I) comp $(s_j', I))$ **then**
  $result \leftarrow$ (" P is totaly compatible with P$'$ ")
else if (s_0, I) comp (s_0', I) and $(\exists\ (s_i, I) \overset{q()/m(p)}{\longrightarrow} (s_j, I)$ then $\exists\ (s_i', I) \overset{q'()/m(\overline{p})}{\longrightarrow}$ (s_j', I) and (s_j, I) comp $(s_j', I))$ **then**
  $result \leftarrow$ (" P is partialy compatible with P$'$ ")
else if (s_0, I) comp (s_0', I) and $(\forall\ (S_i, I) \overset{q()/m(p)}{\longrightarrow} (s_j, I)$ then $\nexists\ (s_i', I) \overset{q'()/m(\overline{p})}{\longrightarrow} (s_j',$ $I)$ and (s_j, I) comp $(s_j', I))$ **then**
  $result \leftarrow$ (" P is not compatible with P$'$ ")
else
  $result \leftarrow$ (" ERROR ")
end if
return $result$

Pour bien illustrer les différentes fonctionnalités utilisées par nos deux applications l'éditeur des protocoles et le vérifieur, on va se baser sur un ensemble de captures d'écran avec quelques détails des d'algorithmes utilisés.

6.4 Un exemple d'exécution

Dans cette partie, on présente un exemple d'exécution complet de nos deux applications développées en Java. Premièrement, on commence par l'éditeur (voir figure 6.1).

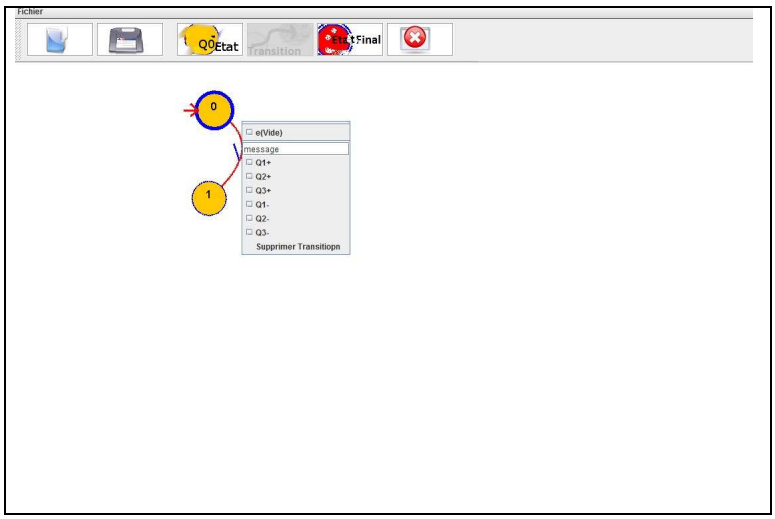


FIGURE 6.1: L'interface de l'éditeur

Cette interface nous permet de créer ou d'importer des automates (ou protocoles) existants avec leurs différents composants, elle nous permet de dessiner des états, des boucles et des transitions avec des requêtes en tant que conditions et des messages en tant que actions. À partir de cette interface on peut également supprimer ou modifier les états et les transitions et bien sûr enregistrer les automates.

6.4.1 Un vérifieur de protocoles de services gardés

Le vérifieur générique a une interface composée de plusieurs composants. la figure 6.2 montre cette interface qui contient un espace pour importer graphiquement les deux protocoles de services créer par l'éditeur. Après cette étape, les automates importés seront directement transformés en matrice en utilisant la classe « autoser ». À chaque itération, nous vérifions le type d'objet qui se trouve dans l'attribut « type » de la classe citée précédemment. Si cet objet est une transition nous l'ajoutant dans une liste de transitions sinon il sera dans la liste des états.

La matrice qui représente l'automate sera remplie par les objets de « autoser », cela se fait en analysant la liste des transitions précédemment formée. Chaque objet « autoser » contients deux attributs, le premier attribut avec deux valeurs « X » représentant l'identifiant de l'état source et « Y » l'identifiant de l'état destination et le second attribut « valuelink » contient la valeur de la transition.

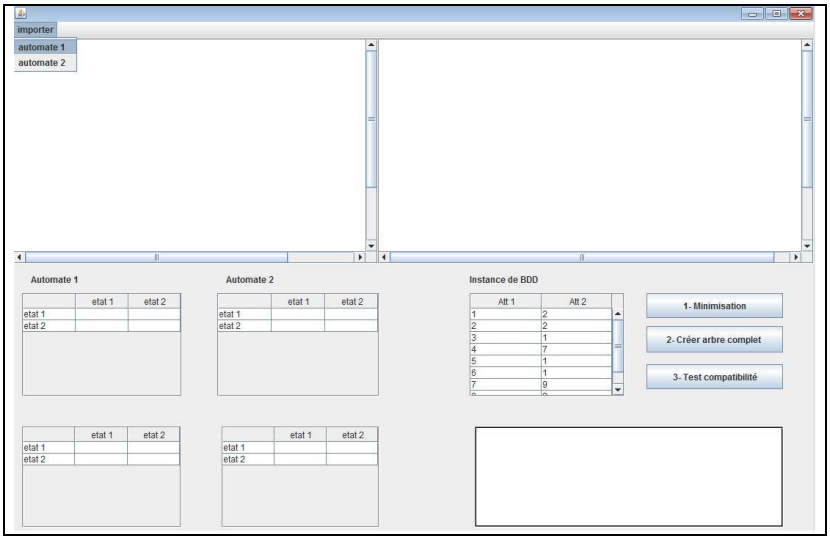


FIGURE 6.2: L'interface du vérifieur

L'interface du vérifieur contient également une instance de base de données, cette instance a été modélisée d'une manière symbolique et modifiable avec deux attributs numériques. Finalement, on trouve un espace résultat qui sert à afficher le résultat de l'algorithme et un espace pour afficher les matrices réduites.

La deuxième étape de notre algorithme de compatibilité (après l'étape de la transformation des automates) est activée par le bouton « 1.Minimisation ». Dans cette étape, le programme analyse les matrices obtenues par l'étape de transformation en se focalisant sur les valeurs des cellules si elles ne sont pas égales à *Null*, alors le programme génère des transitions valides selon l'instance de la base de données et bien sûr les valeurs des requêtes booléennes.

La troisième étape est activée par le bouton « créer l'arbre complet ». Dans cette étape le programme teste tous les états finaux et crée tous les arbres d'exécution complets c'est-à-dire tous les arbre qui commencent par des états initiaux et se terminent par des états finaux. Cela se fait en utilisant la procédure « matlist » qui utilise une liste temporaire afin de suivre les nœuds visités pour créer une liste de vecteurs dans laquelle chaque vecteur représente un arbre complet.

Finalement, nous utilisons l'algorithme 1 vu précédemment mais la comparaison se fait entre vecteurs. Par exemple, la figure 6.3 représente une compatibilité totale entre deux automates donnés , c'est-à-dire que tous les vecteurs représentant le premier protocole possèdent des vecteurs compatibles représentant le second protocole.

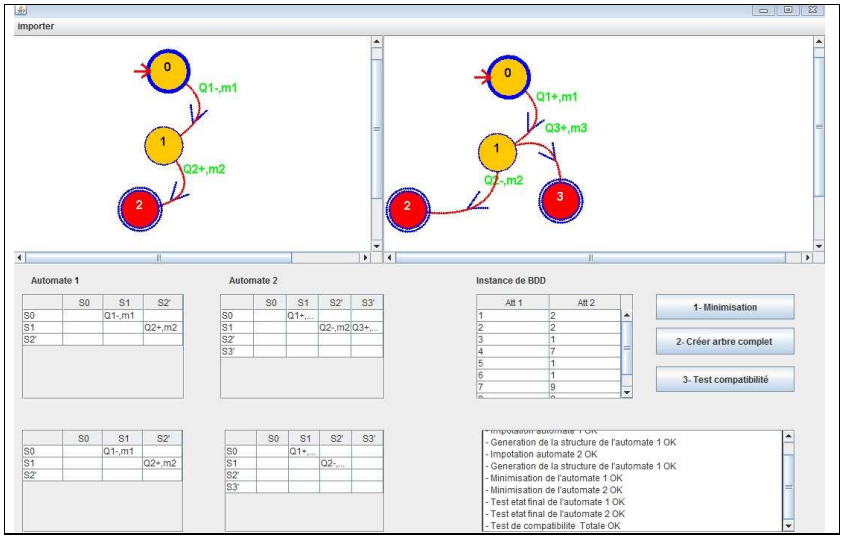


FIGURE 6.3: Une compatibilité totale entre protocoles

S'il existe au moins un vecteur représentant un arbre d'exécution complet dans le premier protocole qui n'est pas compatible avec un vecteur représentant l'arbre d'exécution complet dans le deuxième protocole, on peut dire que les protocoles données sont partiellement compatibles (voir figure 6.4).

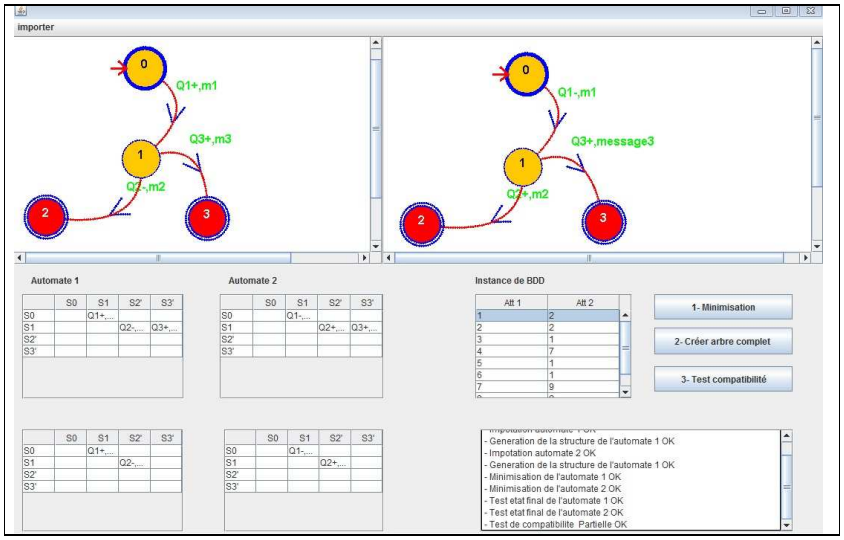


FIGURE 6.4: Une compatibilité partielle entre protocoles

Enfin, si pour chaque vecteur représentant la première liste, on ne trouve pas de vecteurs compatibles dans la deuxième liste. On peut dire que les protocoles ne sont pas compatibles. Par exemple, dans la figure 6.5 on a un exemple dans lequel nous avons appliqué un test de compatibilité entre deux protocoles équivalents, dans ce cas l'algorithme va répondre par non.

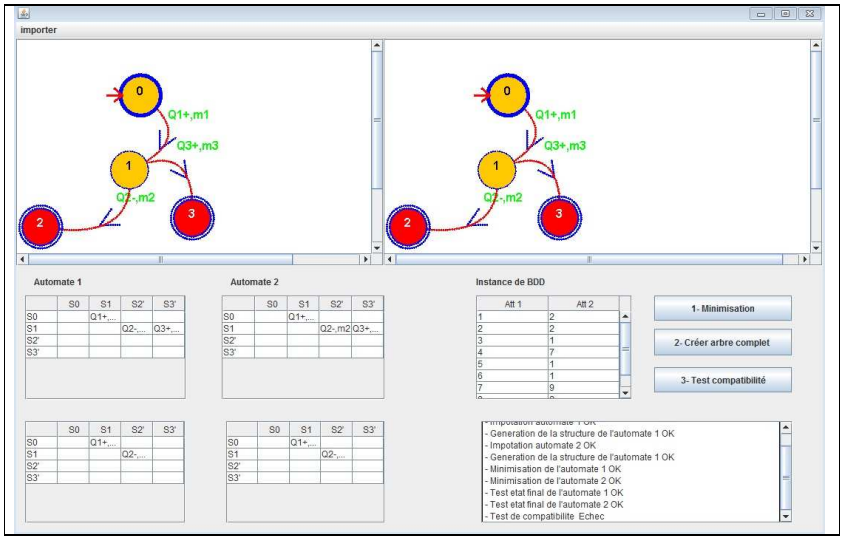


FIGURE 6.5: L'incompatibilité des protocoles

6.5 Conclusion

Dans ce chapitre, nous avons d'abord présenté l'environnement sur lequel on a développé nos applications. Une première interface utilisée pour créer les protocoles de services, elle permet aux utilisateurs de modéliser les protocoles en précisant les transitions, les conditions, les états initiaux et finaux ou bien ouvrir un protocole existant pour une éventuelle modification. Les protocoles crées vont être importés par une nouvelle plate-forme dans le but d'étudier leurs compatibilité. L'objectif de ce chapitre était principalement de mettre en place une plate-forme réelle et utilisable dans le monde de la découverte des services compatibles en se focalisant sur leurs protocoles d'exécution, cette initiative permet de basculer de la difficulté théorique vers le monde réel.

“ On observera que la conclusion précéda sans doute les preuves. Qui se résigne à chercher des preuves d’une chose à laquelle il ne croit pas ou dont la prédication ne l’intéresse pas ?

Jorge Luis Borges, 1944 - « Fiction »

7

Conclusion générale

7.1 Travaux connexes

Nos recherches ont été motivées par plusieurs travaux, Le premier travail qui nous a inspiré est le travail effectué en 2006 par Benatallah et *al* [Benatallah06a]. Dans ce travail, les auteurs discutent la compatibilité et la similarité des protocoles de services web mais avec l'absence des données. Le but principale des auteurs était de fournir aux développeurs des informations nécessaires sur la façon d'écrire des clients pouvant interagir correctement avec des services web donnés en fonction de leurs protocoles. Certains travaux se focalisent sur le problème de la vérification de la simulation des services web centrés données comme par exemple dans [Akroun14a] où les auteurs étudient la décidabilité et la complexité du framework Colombo introduit dans [Berardi05b]. Le point intéressant dans ce travail, est que les auteurs ont proposé une procédure symbolique basées sur la notion de région d'automates pour gérer l'infinité de données. Il existe aussi quelques travaux qui traitent le problème de la vérification formelle des GSM centrés artéfacts tels que dans [Solomakhin13] et le problème de la composition des artéfacts centrés processus métiers avec un domaine infini [Hull04, Bultan03, Berardi03a, Mezni18], où les solutions sont généralement basée sur des modèles spécifiques avec des restrictions spécifiques.

Il y a aussi des modèles de services centrés données qui incluent la notion de *transducteur relationnel* (relational transducer) comme dans [Abiteboul98, Abiteboul00]. Dans ces travaux, les auteurs motivés par les applications du commerce électronique ont étudié la spécification déclarative des modèles métiers en utilisant l'approche des transducteurs relationnels avec des relations d'entrée, des relations de sortie et des relations log qui sauvegardent les traces de l'échange. Cette approche a donné de bon résultats pour une classe restreinte des transducteurs relationnels appelée *Spocus transducers*. Dans [Abiteboul98], plusieurs problèmes de vérification concernant les transducteurs relationnels ont été identifiés et partiellement résolus (par exemple le problème de la vérification des propriétés temporelles, la décidabilité de l'équivalence entre les transducteurs relationnels et le problème de la validation du journal). Motivé par ces problèmes,

Spilmann dans [Spielmann03] a généralisé l'approche des transducteurs relationnel introduite dans [Abiteboul98] en utilisant une classe des transducteurs relationnel basée sur la machine abstraite de Gurevich. Dans ce travail, l'auteur a donné de bons résultats sur la décidabilité des problèmes de vérification à l'aide de restrictions syntaxiques sur le modèle appelée *entrée bornée* dans le but de restreindre l'exécution de la machine uniquement sur une base de données bornée. D'autres solutions plus efficaces traitent le problème de la décidabilité dans le cas d'infinité de données par l'utilisation de se qu'on appelle la représentation symbolique des exécutions telles que dans [Deutsch04, Deutsch06, Deutsch07].

Faisant référence aux travaux cités précédemment, nous constatons d'une part qu'il y a ceux qui analysent la compatibilité et la similarité entre les protocoles métiers mais sans les données. D'autre part, il y a des travaux qui analysent le problème d'infinité de données mais ils donnent toujours des solutions basées sur des restrictions bien spécifiques aux modèles étudiés, cela nous a poussé a explorer le problème de la compatibilité des protocoles en généralisant nos solutions avec le minimum de restrictions possibles.

7.2 Discussion

L'émergence de l'architecture orientée services (SOA) et les technologies des services web a déclenché une vague d'innovations qui a changé la façon dont les affaires sont menées. Aujourd'hui, les entreprises exigent de nouveaux frameworks et méthodologies pour soutenir le développement automatisé et l'interopérabilité de leurs services web. Donc il est nécessaire, de trouver des solutions robustes et intégrées, cela se fait par l'échange de données au sein de la même organisation ou avec d'autres entreprises. Cette thèse entre dans le cadre de la vérification formelle des protocoles de services web plus particulièrement des protocoles centrés données qui formalisent le paradigme de la modélisation des processus métiers. Ce domaine a récemment attiré l'attention des deux communautés industrielles et de recherche. Nous concentrons notre étude sur l'identification des abstractions et notations appropriées pour spécifier les exigences et les caractéristiques des

services, pour pouvoir étudier le problème de la décidabilité liée à la compatibilité entre les protocoles, c'est-dire-dire vérifier si le comportement d'un protocole de service modélisé sous forme de machine à états finis peut communiquer avec un autre service. Nos résultats actuels sont présentés comme suit :

La compatibilité des protocoles gardés de type $(L_G, \emptyset, \emptyset)^I$***.*** Dans ce cas la compatibilité est décidable car l'instance I est finie, cela conduit à un test de compatibilité entre deux protocoles $(L_G, \emptyset, \emptyset)^I$ avec des états finis.

La compatibilité des protocoles non gardés de type $(\emptyset, \emptyset, L_U)^I$***.*** Ce type de protocoles est capable de mettre à jour les instances de base de données en utilisant des requêtes d'insertion et de suppression. Dans ce cas, on a vu que la compatibilité est décidable car les domaines actifs des instances sont finis.

La compatibilité des protocoles de type $(L_G, L_M, L_U)^I$***.*** Après une multitude de tests sur les langages de condition, des messages et de mise à jour par exemple les requêtes conjonctives ou les programmes datalog, nous avons conclu que le test de compatibilité entre ces protocoles est décidable s'il n'y a pas une création de nouvelles valeurs et si l'évaluation du langage de requête est décidable.

La compatibilité des protocoles gardés de type $(L_G, \emptyset, \emptyset)^{DB}$***.*** Avec ce type de protocoles, nous avons abordé le problème d'infinité de données qui provient du nombre infini de messages et de bases de données initiales. Dans ce cas la compatibilité est décidable ssi la satisfaisabilité de $L_G \cup \{ \wedge^b , \neg^b \}$ est décidable.

La compatibilité des protocoles non gardés de type $(\emptyset, L_M, \emptyset)^{DB}$***.*** Nous avons constaté qu'avec ce type de protocoles, la compatibilité est décidable si la satisfaisabilité de la partition dans L_M est décidable.

Les résultats présentés ci-dessus, jouent un rôle important dans l'évaluation de la compatibilité des services web centrés données qui sont spécifiés par leurs

protocoles métiers. Car les protocoles spécifient le comportement du service et cela nous permet d'identifier les parties compatibles entre les services.

Notre analyse des protocoles fournit aussi des solutions basées comportement dans le domaine de la découverte des services web centrés données. On peut aussi utiliser nos résultats pour évaluer si les nouveaux clients sont compatibles avec les services existants pour facilité leur interopérabilité.

Dans la pratique ceci nous a conduit à développer un outil qui teste la compatibilité entre des protocoles de services centrés données selon la classe (L_G, $\emptyset$, $\emptyset)^I$ et avec une base de donnée symbolique qui est interrogée par des requêtes condition. Cet outil est appelé un vérifieur générique des protocoles gardés. Avec cet outil, les utilisateurs peuvent modéliser les protocoles par des machines à états finis pour pouvoir identifier automatiquement les protocoles compatibles. Dans le but d'envisager une éventuelle remplaçabilité en cas d'anomalies dans le système adéquat. On note aussi que la création des protocoles de services se fait manuellement selon les fichiers log (journal) du système global sur lequel on travaille.

Finalement, des limitations peuvent être observées dans notre travail surtout dans le coté pratique, parmi les quelles on peut citer :

- On n'a pas pu tester la scalabilité de notre vérifieur, c'est-à-dire sa capacité de s'adapter à un changement d'ordre de grandeur et sa capacité à maintenir ses fonctionnalités en cas d'utilisation de larges protocoles avec plusieurs bifurcations,
- Le vérifieur n'a pas été testé sur un système réel et avec une base de données réelle.

7.3 Conclusion et perspectives

Dans cette thèse, nous avons présenté le paradigme des services web orientés données qui a récemment attiré l'attention des deux communautés, industrielle et celle de la recherche. Dans un premier temps, nous avons étudié la décidabilité de la compatibilité entre les services web orientés données qui sont spécifiés par leurs protocoles conversationnels. L'idée principale derrière cette étude, est de pouvoir obtenir les différentes classes de protocoles dans lesquelles la vérification de la compatibilité est décidable. Cette étude nous a aussi permis d'explorer les différentes techniques et méthodes de la vérification formelle pour assurer le bon fonctionnement des systèmes informatiques.

Dans la deuxième partie de cette thèse, nous avons proposé un prototype de vérifieur qui teste la compatibilité entre deux protocoles donnés. Ce vérifieur peut jouer un rôle crucial dans le domaine de la découverte et l'intégration des services web orientés données.

Nos travaux futurs porteront notamment sur le plan pratique et les améliorations que nous proposons sont les suivantes :

- Afin de rendre notre vérifieur plus flexible, il est nécessaire de l'enrichir par des bases de données réelles et un générateur de requêtes,
- Se pencher vers l'automatisation de la modélisation des protocoles à partir des fichiers de description et des fichiers log,
- réaliser un autre prototype qui identifie tous les services compatibles avec un service donné, cela nécessite l'utilisation d'une base de services.

Dans le coté théorique, nous pouvons améliorer notre analyse par l'exploration du problème de la compatibilité en intégrant d'autres aspects tels que l'aspect temps, la qualité des services et la sémantique des services.

Bibliographie

[Abiteboul95] Serge Abiteboul, Richard Hull & Victor Vianu. Foundations of databases. Addison-Wesley, Boston, 1995.

[Abiteboul98] S. Abiteboul, V. Vianu, B. Fordham & Y. Yesha. *Relational Transducers for Electronic Commerce.* In Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '98, pages 179–187. ACM, 1998. 135, 136

[Abiteboul00] S. Abiteboul, V. Vianu, B. Fordham & Y. Yesha. *Relational Transducers for Electronic Commerce.* Journal of Computer and System Sciences, vol. 61, no. 2, pages 236–269, 2000. 135

[Abiteboul08a] S. Abiteboul, O. Benjelloun & T. Milo. *The Active XML project : an overview.* The VLDB Journal, vol. 17, no. 5, pages 1019–1040, 2008. 70, 84

[Abiteboul08b] S. Abiteboul, L. Segoufin & V. Vianu. *Static Analysis of Active XML Systems.* In Proceedings of the Twenty-seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '08, pages 221–230, New York, NY, USA, 2008. ACM. 70, 84

[Abiteboul09] S. Abiteboul, L. Segoufin & V. Vianu. *Static analysis of Active XML Systems.* ACM Trans. Database Syst., vol. 34, no. 4, pages 1–23, ACM, New York, NY, USA, 2009. 70, 84

[Akroun14a] L. Akroun, B. Benatallah, L. Nourine & F. Toumani. Service-oriented computing : 12th international conference, icsoc

2014, paris, france, november 3-6, 2014. proceedings, chapitre Decidability and Complexity of Simulation Preorder for Data-Centric Web Services, pages 535–542. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. 84, 135

[Akroun14b] M.L. Akroun. *Decidability and Complexity of Simulation Preorder for Data-centric Web service.* Doctorat, Blaise Pascal Clermont ferrand 2 University, 2014. 80

[Alonso04] G. Alonso, F. Casati, H.A. kuno & V. Machiraju. Web services-concepts architectures and applications. Springer-Verlag Berlin Heidelberg, 2004. 2

[Alonso10] G. Alonso, F. Casati, H. Kuno & V. Machiraju. Web services : Concepts, architectures and applications. Springer Publishing Company, Incorporated, 1st edition, 2010. 15

[Alur90] R. Alur & D. Dill. Automata, languages and programming : 17th international colloquium warwick university, england, july 16–20, 1990 proceedings, chapitre Automata for modeling real-time systems, pages 322–335. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990. 49

[Alur94] R. Alur & D. Dill. *A theory of timed automata.* Theoretical Computer Science, vol. 126, no. 2, pages 183–235, 1994. 49

[Alur99] R. Alur, L. Fix & T.A. Henzinger. *Event-clock automata : a determinizable class of timed automata.* Theoretical Computer Science, vol. 211, no. 1âĂŞ2, pages 253 –273, 1999. 35

[Andrews03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, F. Klein, K. Leymann, D. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic & S. Weerawarana. *Business process execution language for web services (version 1.1).* Rapport technique, Specification,IBM Corp.,Microsoft Corp., SAP AG,Siebel Systems, 2003. 29

[Austin04] D. Austin, A. Barbir, C. Ferris & S. Garg. *Web Services Architecture Requirements.* Rapport technique, W3C Working Draft, 2004. 19

[Bellwood02] T. Bellwood, S. Capell & J. Colgrave. *Universal Description, Discovery and Integration specification (UDDI) 3.0.* Rapport technique, Online Httpuddi Orgpubsuddi-V3 00-Publ.-20020719 Htm, 2002. 26

[Ben Mokhtar06] S. Ben Mokhtar, N. Georgantas & V. Issarny. *COCOA : ConversationBased Service Composition for Pervasive Computing Environments.* In 2006 ACS/IEEE International Conference on Pervasive Services, pages 29–38, 2006. 46

[Benatallah04a] B. Benatallah, F. Casati & F.i Touman. Conceptual modeling – er 2004 : 23rd international conference on conceptual modeling, shanghai, china, november 8-12, 2004. proceedings, chapitre Analysis and Management of Web Service Protocols, pages 524–541. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. xii, 35, 40, 47, 52

[Benatallah04b] B. Benatallah, F. Casati & F. Toumani. *Web Service Conversation Modeling : A Cornerstone for e-Business Automation.* IEEE internet Computing, vol. 8, no. 1, pages 46–54, IEEE Educational Activities Department, Piscataway, NJ, USA, 2004. 2, 35

[Benatallah05a] B. Benatallah, F. Casati, J. Ponge & F. Toumani. *Compatibility and replaceability analysis for timed web service protocols.* In 21èmes Journées Bases de Données Avancées, BDA 2005, Saint Malo, 17-20 octobre 2005, Actes (Informal Proceedings)., 2005. xii, 35, 36

[Benatallah05b] B. Benatallah, F. Casati, J. Ponge & F. Toumani. *On Temporal Abstractions of Web Service Protocols.* In CAiSE Short Paper Proceedings, volume 161, pages 39–44. Springer, 2005. xii, 35, 36

[Benatallah05c] B. Benatallah, R.M. Dijkman, M. Dumas & Z. Maamar. Service-oriented software system engineering : Challenges and practices, chapitre Service Composition : Concepts, Tech-

niques, Tools and Trends, pages 48–66. Idea Group Publishing, Hershey, 2005. 28

[Benatallah06a] B. Benatallah, F. Casati & F. Toumani. *Representing , analysing and managing web service protocols*. Data and Knowledge Engineering, vol. 58, no. 3, pages 327–357, 2006. Including the special issue : {ER} 2004ER 2004. 2, 28, 35, 135

[Benatallah06b] B. Benatallah, M. Nezhad & H. Reza. *ServiceMosaic Project : Modeling, Analysis and Management of Web Services Interactions*. In Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modelling - Volume 53, APCCM '06, pages 7–9, Darlinghurst, Australia, 2006. Australian Computer Society, Inc. 35

[Benmerzoug09] M.D. Benmerzoug. *ModÃĺles et outils formels pour lâĂŹintÃl'gration dâĂŹapplications dâĂŹentreprises*. Doctorat, Université Pierre et Marie Curie, 2009. xii, 55

[Berardi03a] D. Berardi, D. Calvanes, G. De Giacomo, R. Hull, M. Lenzerini & M. Mecella. Service-oriented computing - icsoc 2003 : First international conference, trento, italy, december 15-18, 2003. proceedings, chapitre Automatic Composition of E-services That Export Their Behavior, pages 43–58. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. 71, 84, 135

[Berardi03b] D. Berardi, D. Calvanes, G. De Giacomo, M. Lenzerini & M. Mecella. Service-oriented computing - icsoc 2003 : First international conference, trento, italy, december 15-18, 2003. proceedings, chapitre Automatic Composition of E-services That Export Their Behavior, pages 43–58. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. 46

[Berardi03c] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini & M. Mecella. *A Foundational Framework for e-Services*. Rapport technique, Department of computer and systems , Roma university, 2003. 33

[Berardi04] D. Berardi, F. De Rosa, L. De Santis & M. Mecella. *Finite State Automata As Conceptual Model For E-Services.* J. Integr. Des. Process Sci., vol. 8, no. 2, pages 105–121, IOS Press, Amsterdam, The Netherlands, 2004. 33, 35

[Berardi05a] D. Berardi, D. Calvanes, G. De Giacomo, R. Hull, M. Lenzerini & M. Mecella. Modeling data and processes for service specification in colombo, volume 160. 2005. xii, 71, 73, 74, 75, 79

[Berardi05b] D. Berardi, D. Calvanes, G. De Giacomo, R. Hull & M. Mecella. *Automatic composition of transition-based semantic web services with messaging.* In Proceedings of the 31st International Conference on Very Large Data Bases, VLDB '05, pages 613–624. VLDB Endowment, 2005. 46, 70, 84, 135

[Berthomieu91] B. Berthomieu & M. Diaz. *Modeling and verification of timed dependent systems using timed petri nets.* IEEE transactions on software Engineering, vol. 17, no. 3, pages 259–273, 1991. 49

[Bhattacharya07] K. Bhattacharya, C.E. Gerede, R. Hull, R. Liu & T. Su. Business process management : 5th international conference, bpm 2007, brisbane, australia, september 24-28, 2007. proceedings, chapitre Towards Formal Analysis of Artifact-Centric Business Process Models, pages 288–304. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. 83

[Bhattacharya09] K. Bhattacharya, R. Hull & J. Su. Handbook of research on business process modeling, chapitre A Data-centric Design Methodology for Business Proceses, pages 503–531. IGI-Global, Hershey, Canada, 2009. 83

[Blow04] M. Blow, Y. Goland, M. Kloppmann, F. Leymann, G. Pfau, D. Roller & M. Rowley. *Bpel for java.* Rapport technique, 2004. 57

[Bojanczyk06] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin & C. David. *Two-variable logic on words with data.* In 21st

Annual IEEE Symposium on Logic in Computer Science , LICS'06, pages 7–16, Seattle, United States, 2006. IEEE. 68, 84

[Bojanczyk11] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick & L. Segoufin. *Two-variable Logic on Data Words.* ACM Trans. Comput. Logic, vol. 12, no. 4, pages 1–27, ACM, New York, NY, USA, 2011. 68, 84

[Bolognesi87] T. Bolognesi & E. Brinksmai. *Introduction to the ISO Specification Language LOTOS.* Comput. Netw. ISDN Syst., vol. 14, no. 1, pages 25–59, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 1987. 43

[Bouajjani03] A. Bouajjani, P. Habermehl & R. Mayr. *Automatic verification of recursive procedures with one Prameter.* Theoretical Computer Science, vol. 295, no. 1-3, pages 85–106, 2003. Mathematical Foundations of Computer Science. 68

[Box00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte & D. Winer. *Simple object access protocol (SOAP) 1.1.* Rapport technique, W3C Working Draft, 2000. 21

[Bray08] T. Bray, J. Paolil, C.M. Sperberg, E. Maler & F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fifth Edition).* Rapport technique, W3C Recommendation, 2008. 20

[Bultan03] T. Bultan, X.Fu, R.Hull & J. Su. *Conversation specification : a new approach to design and analysis of e-service composition.* In Proceedings of the 12th International Conference on World Wide Web, WWW '03, pages 403–410. ACM, 2003. 135

[Burkart01] O. Burkart, D. Caucal, F. Moller & B. Steffen. Handbook of process algebra, chapitre Verification of Infinite Structures, pages 545–623. Citeseer, Netherlands, 2001. 68, 83, 84

[Calvanese13] D. Calvanese, G. De Giacomo & M. Montali. *Foundations of data-aware process analysis : a data theory perspective.* In

Proceedings of the 32Nd Symposium on Principles of Database Systems, PODS '13, pages 1–12. ACM, 2013. 2

[Camara06] J. Camara, C. Canal, J. Cubo & A. Vallecillo. *Formalizing {WSBPEL} Business Processes Using Process Algebra*. Electronic Notes in Theoretical Computer Science, vol. 154, no. 1, pages 159–173, 2006. Proceedings of the 4th International Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA 2005)Foundations of Coordination Languages and Software Architectures 2005. 44

[Castro02] J. Castro, M. Kolp & J. Mylopoulos. *Towards requirements-driven information systems engineering : the Tropos project*. Information Systems, vol. 27, no. 6, pages 365–389, 2002. 33

[CBOK09] BPM CBOK. *Guide to the business process management common body of knowledge*. Versão 2.0. 2009. Disponível em : www. abpmp. org. Acesso em : 25 nov 2012, 2009. 31

[Chandra85] A.K. Chandra & M. Vardi. *The implication problem for functional and inclusion dependencies in undecidable*. SIAM Journal on Computing, vol. 14, no. 3, pages 671–677, 1985. 83

[Chaochen99] Z. Chaochen. Algebraic methodology and software technology : 7th international conference, amast'98 amazonia, brazil, january 4–8, 1999 proceedings, chapitre Duration Calculus, a Logical Approach to Real-Time Systems, pages 1–7. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. 49

[Chauvet02] J.M. Chauvet. Services web avec soap, wsdl, uddi, ebxml... Eyrolles, 2002. xii, 52, 53, 54

[Chinnici07] R. Chinnici, J.J. Moreau, A. Ryman & S. Weerawarana. *Web services description language (wsdl) version 2.0 part 1 : Core language*. Rapport technique, W3C Working Draft, 2007. 25

[Coalition99] The Workflow Management Coalition. *Workflow Management Coalition Terminology and Glossary*. Rapport technique WFMC-TC-1011, WFMC, 1999. 30

[Corporation02] Borland Software Corporation. Web services developer's guide. 8 edition, 2002. xii, 22

[Curbera03] F. Curbera, R. Khalaf, N. Mukhi, S. Tai & S. Weerawarana. *The Next Step in Web Services.* Commun. ACM, vol. 46, no. 10, pages 29–34, ACM, New York, NY, USA, 2003. 28

[Daiz06] G. Daiz, M.E. Cambronero, J.J. Pardo, V. Valero & F. Cuartero. *Automatic generation of Correct Web Services Choreographies and Orchestrations with Model Checking Techniques.* In Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services, AICT-ICIW'06, page 186. IEEE, 2006. 35

[Davies08] J. Davies, D. Schorow, S. Ray & D. Rieber. The definitive guide to soa : Oracle service bus. Apress, Berkely, CA, USA, 2nd edition, 2008. 8

[De Maria06] E. De Maria, A. Montanari & M. Zantoni. *An automaton-based approach to the verification of timed workflow schemas.* In Thirteenth International Symposium on Temporal Representation and Reasoning, TIME'06, pages 87–94. IEEE, 2006. 35

[Demri09] S. Demri & R. Lazic. *LTL with the Freeze Quantifier and Register Automata.* ACM Trans. Comput. Logic, vol. 10, no. 3, pages 1–16, ACM, New York, NY, USA, 2009. 68, 84

[Deutsch04] A. Deutsch, L. Sui & V. Vianu. *Specification and verification of data-driven web services.* In Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '04, pages 71–82, New York, NY, USA, 2004. ACM. 71, 81, 84, 136

[Deutsch06] A. Deutsch, L. Sui, V. Vianu & D. Zhou. *verification of Communicating data-driven web services.* In Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '06, pages 90–99, New York, NY, USA, 2006. ACM. 71, 84, 136

[Deutsch07] A. Deutsch, L. Sui & V. Vianu. *Specification and verification of data-driven web services.* Journal of Computer and System Sciences, vol. 73, no. 3, pages 442–474, 2007. Special Issue : Database Theory 2004. 71, 84, 136

[Deutsch09] A. Deutsch, R. Hull, F. Patrizi & V. Vianu. *Automatic Verification of Data-centric Business Processes.* In Proceedings of the 12th International Conference on Database Theory, ICDT '09, pages 252–267, New York, NY, USA, 2009. ACM. 83

[Dong01] W. Dong, J. Wang, X. QI & Z.C. Qi. *Model checking UML statecharts.* In Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific, pages 363–370, 2001. 50

[Feng06] X. Feng, Q. Liu & Z. Wang. Advanced web and network technologies, and applications : Apweb 2006 international workshops : Xra, iwsn, mega, and icse, harbin, china, january 16-18, 2006. proceedings, chapitre A Web Service Composition Modeling and Evaluation Method Used Petri Net, pages 905–911. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. 41

[Floyed67] R.W. Floyed. *Assigning meanings to programs.* In Mathematical Aspects of Computer Science. Proceedings of Symposia in Applied Mathematics-Volume 19, pages 19–32, 1967. 42

[Floyed93] R.W. Floyed. Program verification : Fundamental issues in computer science, chapitre Assigning Meanings to Programs, pages 65–81. Springer Netherlands, Dordrecht, 1993. 42

[Gallien07] M. Gallien, F. Gargouri, I. Kahloul, M. Krichen, T. Nguyen, S. Bensalem & F. Ingrand. *DâĂŹune approche modulairea une approche orientée composant pour le développement de systemes autonomes : Défis et principes.* pages 34–49, Citeseer, 2007. 12

[Garson01] J.W. Garson. Handbook of philosophical logic, chapitre Quantification in Modal Logic, pages 267–323. Springer Netherlands, Dordrecht, 2001. 70

[Georgakopoulos95] D. Georgakopoulos, M. Hornick & A. Sheth. *An overview of workflow management : From process modeling to workflow automation infrastructure.* Distributed and Parallel Databases, vol. 3, no. 2, pages 119–153, 1995. 30

[Goldberg83] A. Goldberg & D. Robson. Smalltalk-80 : the language and its implementation. Addison-Wesley Longman Publishing Co., Inc., 1983. 11

[Gudgin03] M. Gudgin, M. Hadley, N. Mendelsohn, J.J. Moreau & H.F. Nielsen. *SOAP Version 1.2 Partie 1 : Structure pour l'Ãl'change de messages.* Rapport technique, W3C Recommendation, 2003. xii, 24

[Hamadi03] R. Hamadi & B. Benatallah. *A Petri net based model for web service composition.* In Proceedings of the 14th Australasian Database Conference - Volume 17, ADC '03, pages 191–200, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc. 41

[Harel87] D. Harel. *Statcharts : A visual formalism for complex systems.* Science of Computer Programming, vol. 8, no. 3, pages 231–274, 1987. 50

[Hoare69] C.A.R. Hoare. *An axiomatic basis for computer programming.* Commun.ACM, vol. 12, no. 10, pages 576–580, ACM, New York, NY, USA, 1969. 42

[Hoare78] C.A.R. Hoare. *Communicating Sequential Processes.* Commun. ACM, vol. 21, no. 8, pages 666–677, 1978. 43

[Hughes68] G. Hughes & M. Cresswell. An introduction to modal logic. Methuen, 1968. 70

[Huizing91] C. Huizing & W.P. De Roever. *Introduction to design choices in the semantics of Statecharts.* Information Processing Letters, vol. 37, no. 4, pages 205–213, 1991. 51

[Hull89] R. Hull & J. Su. *Domain independence and the Relational calculus.* Rapport technique 88-64, Computer Science Department University of Southern California, 1989. 78

[Hull94] R. Hull & J. Su. *Domain independence and the Relational calculus.* ACTA Informatica, vol. 31, no. 6, pages 513–524, 1994. 78

[Hull04] R. Hull & J. Su. *Tools for design of composite web services.* In Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, SIGMOD '04, pages 958–961. ACM, 2004. 135

[Kazhamiakin06] R. Kazhamiakin, P. Pandya & M. Pistore. *Timed modelling and analysis in Web service compositions.* In First International Conference on Availability, Reliability and Security, ARES'06, page 7. IEEE, 2006. 35

[Keller03] A. Keller & H. Ludwing. *the WSLA framework : Specifying and monitoring service level agreements for Web Services.* Journal of Network and Systems Management, vol. 11, no. 1, pages 57–81, 2003. 43

[Lamanna03] D.D Lamanna, J. Skene & W. Emmerich. *SLAng : A language for defining service level agreements.* In Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of, pages 100–106, 2003. 43

[Lazic07] R. Lazic, T. Newcomb, J. Ouaknine, A.W. Roscoe & J. Worrell. Petri nets and other models of concurrency – icatpn 2007 : 28th international conference on applications and theory of petri nets and other models of concurrency, icatpn 2007, siedlce, poland, june 25-29, 2007. proceedings, chapitre Nets with Tokens Which Carry Data, pages 301–320. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. 68, 84

[Lazic08] R. Lazic, T. Newcomb, J. Ouaknine, A.W. Roscoe & J. Worrell. *Nets with tokens which carry data.* Fundam. Inf., vol. 88, no. 3, pages 251–274, IOS Press, Amsterdam, The Netherlands, 2008. 68, 84

[Leymann97] F. Leymann & D. Roller. *Workflow-based Applications*. IBM Syst. J., vol. 36, no. 1, pages 102–123, 1997. 30

[Liu06] F. Liu, Y. Shi, L. Zhang, L. Lin & B. Shi. Data engineering issues in e-commerce and services : Second international workshop, deecs 2006, san francisco, ca, usa, june 26, 2006. proceedings, chapitre Analysis of Web Services Composition and Substitution Via CCS, pages 236–245. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. 44

[Martin05] D. Martin, M. Paolucci, S. Mcilraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan & K. Sycara. Semantic web services and web process composition : First international workshop, swswpc 2004, san diego, ca, usa, july 6, 2004, revised selected papers, chapitre Bringing Semantics to Web Services : The OWL-S Approach, pages 26–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. 28

[Mecella02] M. Mecella & B. Pernici. *Building Flexible and Cooperative Applications Based on e-Services*. Rapport technique, Department of computer and systems , Roma university, 2002. 33

[Mezni18] H. Mezni & M. Kbekbi. *Reusing process fragments for fast service composition : a clustering-based approach*. Enterprise Information Systems, vol. 0, no. 0, pages 1–29, Taylor & Francis, 2018. 84, 135

[Mikk97] E. Mikk, Y. Lakhnech & M. Siegel. Advances in computing science — asian'97 : Third asian computing science conference kathmandu, nepal, december 9–11, 1997 proceedings, chapitre Hierarchical automata as model for statecharts, pages 181–196. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. xii, 51

[Mikk98] E. Mikk, Y. Lakhnech, M. Siegel & G.J. Holzmann. *Implementing Statecharts in PROMELA/SPIN*. In Industrial

Strength Formal Specification Techniques, 1998. Proceedings. 2nd IEEE Workshop on, pages 90–101, 1998. 50

[Milner82] R. Milner. A calculus of communicating systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982. 43

[Milner89] R. Milner. Communication and concurrency. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. 43

[Milner92] R. Milner, J. Parrow & D. Walker. *A Calculus of mobile processes , i.* Information and Computation, vol. 100, no. 1, pages 1–40, 1992. 43

[Mokhtari09] K. Mokhtari, S. Benbernou, M. Rouached, M.S. Hacid & F. Leymann. *Privacy Time-Related Analysis in Business Protocols.* In Web Services, 2009. ICWS 2009. IEEE International Conference on, ICWS'09, pages 141–148. IEEE, 2009. 37

[Mokhtari12] K. Mokhtari, S. Benbernou, S. Sahri, V. Andrikopoulos, F. Leymann & M.S. Hacid. *Timed Privacy Aware Business Protocols.* international Journal of Cooperative Information Systems, vol. 21, no. 2, pages 85–109, 2012. 37

[Musaraj10] K. Musaraj. *Automatic extraction of communication protocols for web services composition.* Doctorat, Claude Bernard University- Lyon 1, France, 2010. xii, 34

[Neven04] F. Neven, T. Schwentick & V. Vianu. *Finite State Machines for Strings Over Infinite Alphabets.* ACM Trans. Comput. Logic, vol. 5, no. 3, pages 403–435, ACM, New York, NY, USA, 2004. 68, 84

[Newcomer03] E. Newcomer. Understanding web services : Xml, wsdl, soap, and uddi. David Chappell Series, 1st edition, 2003. xii, 25

[Nicollin94] X. Nicollin & J. Sifakis. *The Algebra of Timed Processes , ATP : Theory and Application.* Information and Computation, vol. 114, no. 1, pages 131–178, 1994. 49

[Okano99] K. Okano, S. Hattori, A. Yamamoto & T. Higashinoz. *Specification of real-time systems using a timed automata model with*

shared variables and verification of partial deadlock freeness. In Parallel Processing, 1999. Proceedings. 1999 International Workshops on, pages 576–581, 1999. 48

[O'Riordan02] D. O'Riordan. Web services business strategies and architectures, chapitre Business Process Standards For Web Services, pages 156–173. Apress, Berkeley, CA, 2002. 30

[Papazoglou01] M.P. Papazoglou. *Agent-oriented Technology in Support of e-Business.* Commun. ACM, vol. 44, no. 4, 2001. 15

[Papazoglou04] M.P. Papazoglou & J.J Dubray. *A survey of web service technologies.* Rapport technique, University of Trento, 2004. 29

[Papazoglou07] M.P Papazoglou & W.J. Van den Heuvel. *Service oriented architectures : approaches, technologies and research issues.* The VLDB Journal, vol. 16, no. 3, pages 389–415, 2007. 16

[Papazoglou08] M. Papazoglou. Web services : principles and technology. Pearson Education, 2008. xii, 24, 27, 28

[Peltz03] C. Peltz. *Web Services Orchestration and Choreography.* Computer, vol. 36, no. 10, pages 46–52, IEEE Computer Society Press, Los Alamitos, CA, USA, 2003. xii, 29

[Petri62] C.A. Petri. *Kommunikation mit automaten.* Doctorat, University of Bonn , Germany, 1962. 40

[Pla-Castells15] M. Pla-Castells, I. García-Fernández & R.J. Martínez-Durá. *Modelling and simulation of several interacting cellular automata.* International Journal of Computer Aided Engineering and Technology, vol. 7, no. 2, pages 192–206, Inderscience Publishers, 2015. 46

[Ponge10] J. Ponge, B. Benatallah, F. Casati & F. Toumani. *Analysis and Applications of Timed Service Protocols.* ACM Trans. Softw. Eng. Methodol., vol. 19, no. 4, pages 1–11, ACM, New York, NY, USA, 2010. 35

[Reed88] G.M. Reed & A.W. Roscoe. *A timed model for Communicating sequential processes.* Theoretical Computer Science, vol. 58, no. 1, pages 249–261, 1988. 49

[Reichert12] M. Reichert. On the move to meaningful internet systems : Otm 2012 : Confederated international conferences : Coopis, doa- svi, and odbase 2012, rome, italy, september 10-14, 2012. proceedings, part i, chapitre Process and Data : Two Sides of the Same Coin ?, pages 2–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. 3

[Reza06] H. Reza, N. Motahari, B. Benatallah, F. Casati & F. Toumani. *Web Services Interoperability Specifications*. Computer, vol. 39, no. 5, pages 24–32, IEEE Computer Society Press, Los Alamitos, CA, USA, 2006. 30

[Reza08] H. Reza & N. Motahari. *Discovery and adaptation of process views,*. Doctorat, Computer Science and Engineering, Faculty of Engineering, UNSW, Sydney, Australia, 2008. 30

[Ryan09] K.L.Ko. Ryan. *A Computer Scientist's Introductory Guide to Business Process Management (BPM)*. Crossroads, vol. 15, no. 4, pages 11–18, 2009. 30

[Solomakhin13] D. Solomakhin, M. Montali, S. Tessaris & R.D. Masellis. Service-oriented computing : 11th international conference, icsoc 2013, berlin, germany, december 2-5, 2013, proceedings, chapitre Verification of Artifact-Centric Systems : Decidability and Modeling Issues, pages 252–266. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. 135

[Song18] Y. Song, L. Hu & M. Yu. *A novel QoS-aware prediction approach for dynamic web services*. PLoS ONE, vol. 13, no. 8, 2018. 43

[Spielmann03] M. Spielmann. *Verification of relational transducers for electronic commerce*. Journal of Computer and System Sciences, vol. 66, no. 1, pages 40–65, 2003. Special Issue on {PODS} 2000. 80, 81, 84, 136

[Tay90] Tay, Beng Hang & L. Akkihebbal Ananda. *A survey of remote procedure calls*. ACM SIGOPS Operating Systems Review, vol. 24, no. 3, pages 68–79, ACM, 1990. 10

[Tiplea06] F.L. Tiplea & G.I. Macovei. *E-timed Workflow Nets*. In 2006 Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pages 423–429. IEEE, 2006. 35

[Tosic03] V. Tosic, B. Pagurek & K. Patel. *WSOL- A language for the formal specification of classes of service for web services*. In Web Services-ICWS-Europe 2003. Proceedings.InternationalConference ICWS-Europe 2003 Erfurt, Germany, pages 375–381, 2003. 43

[Van der Aalst98] W.M.P. Van der Aalst. *the application of Petri nets to workflow management*. Journal of Circuits, Systems and Computers, vol. 8, no. 1, pages 21–66, 1998. 40

[Van der Aalst99] W.M.P. Van der Aalst. *Interorganizational Workflows : An Approach based on message Sequence Charts and Petri Nets*. Systems Analysis-Modelling-Simulation, vol. 34, no. 3, pages 335–367, 1999. 40

[Van der Aalst03] W. M. P. Van der Aalst, A. H. M. ter Hofstede & M. Weske. Business process management : International conference, bpm 2003 eindhoven, the netherlands, june 26–27, 2003 proceedings, chapitre Business Process Management : A Survey, pages 1–12. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. 30, 31

[Vardhan05] A. Vardhan, K. Sen, M. Viswanathan & G. Agha. Fsttcs 2004 : Foundations of software technology and theoretical computer science : 24th international conference, chennai, india, december 16-18, 2004. proceedings, chapitre Actively Learning to Verify Safety for FIFO Automata, pages 494–505. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. 48

[Vianu09] V. Vianu. *Automatic Verification of Database-Driven Systems : A New Frontier*. In Proceedings of the 12th International Conference on Database Theory, ICDT '09, pages 1–13, New York, NY, USA, 2009. ACM. 69, 84

[Xu17] B. Xu, L.D. Xu, X. Fei, L. Jiang, H. Cai & S. Wang. *A method of demand-driven and data-centric Web service configuration for flexible business process implementation.* Enterprise Information Systems, vol. 11, no. 7, pages 988–1004, Taylor & Francis, 2017. 83

[Yellin97] D.M. Yellin & R.E. Storm. *Protocol Specifications and component adaptors.* ACM Trans. Program. Lang. Syst, vol. 19, no. 2, pages 292–333, 1997. 2

[Yi04] X. Yi & K. Kochut. *A CP-nets-based Design and verification Framework for Web services Composition.* In Web Services, 2004. Proceedings. IEEE International Conference on, pages 756–760, 2004. 41

[Zhang04] J. Zhang, C.K. Chang, J.Y. Chung & S.W. Kim. *WS-Net : a Petri-net based specification model for web services.* In Web Services, 2004. Proceedings. IEEE International Conference on, pages 420–427, 2004. 41

[Zhang08] Z. Zhang, F. Hang & H. Xiao. *A colored Petri net based model for web service composition.* Journal of Shanghai University (English Edition), vol. 12, no. 4, pages 323–329, 2008. 41