Etude des Interactions Temporisées dans la Composition de Services Web

Nawal Guermouche

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Etude des Interactions Temporisées dans la Composition de Services Web

THÈSE

présentée et soutenue publiquement le 23/06/2010

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Nawal Guermouche

Composition du jury

Rapporteurs :	Jean Paul Bahsoun, Professeur à l'Université Paul Sabatier, IRIT
	Farouk Toumani, Professeur à l'Université Blaise Pascal, LIMOS
Examinateurs :	Djamel Benslimane, Professeur à l'Université Claude Bernard, LIRIS
	Christine Collet, Professeur à l'INP Grenoble, LIG
	Bruno Lévy, Directeur de recherche à l'INRIA grand Est, LORIA
	Olivier Perrin, Maître de conférence à l'Université Nancy 2, LORIA
Directeur de thèse :	Claude Godart, Professeur à l'Université Henri Poincaré Nancy 1, LORIA

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Aux plus chers à mon cœur :

mes parents,

ma sœur et mes frères.

" La confiance est un élément majeur : sans elle, aucun projet n'aboutit "
Éric Tabarly

Remerciements

La préparation de cette thèse m'a permis de rencontrer plusieurs personnes qui m'ont beaucoup apportée, que ça soit sur le plan professionnel que sur le plan humain, que je voudrais remercier. Tout d'abord, j'exprime ma reconnaissance et ma gratitude à mon directeur de thèse, M Claude Godart, professeur à l'université Henri Poinacré, Nancy 1 de m'avoir permis de mener ce travail de recherche au sein de son équipe et pour la confiance, la liberté, et l'appui qu'il m'a accordée.

J'adresse mes vifs remerciements à :

M Olivier Perrin, maître de conférence à l'université Nancy 2, pour ses remarques lors de la lecture de ma thèse.

M Bruno Lévy, directeur de recherche à l'INRIA, qui me fait l'honneur de présider le jury de ma thèse.

M Farouk Toumani, Professeur à l'université Blaise Pascal, et M Jean-Paul Bahsoun, Professeur à l'université Paul Sabatier d'avoir accepté de rapporter ma thèse.

M Djamel Benslimane, Professeur à l'université Claude Bernard, et Mme Christine Collet, Professeur à l'INP Grenoble pour avoir accepté d'être examinateurs de ma thèse.

Un grand merci aux membres de mon équipe SCORE pour l'accueil et la bonne ambiance et en particulier mes collègues pour les moments agréables qu'on a passés au déjeuné ou à jouer au ping-pong. Egalement, je n'oublie pas les secrétaires Roxane Auclair et Isabelle Herlich pour leur efficacité et disponibilité et pour leur bonne humeur.

Je ne voudrais terminer sans remercier le personnel enseignant et administratif de l'ESSTIN où je suis intervenue en tant que vacataire puis en tant que ATER, de m'avoir permis d'enrichir mes connaissances et mes capacités pédagogiques.

Je n'oublie pas de remercier tous mes amis pour les moments inoubliables qu'on a pu passer ensemble.

Je remercie toute ma famille, en particulier mes parents pour tout le soutient et l'aide qu'ils m'ont toujours apportée, ma sœur et mes frères, en particulier mon frère Nabil, et mes amies Nadia et Sabrina.

Table des matières

Table des figures **11**

Introduction générale

Chapitre 1
Contexte et problématique

1.1 Contexte de la thèse 20
1.2 Problématique 22
1.3 Scénario illustratif 25
1.3.1 Le problème de compatibilité temporisée 27
1.3.2 Le problème de composition temporisée 28
1.4 Apport de la thèse 31
1.5 Conclusion 33

Chapitre 2
Etat de l'art

2.1 Introduction 36
2.2 Le paradigme de calcul orienté service 36
2.2.1 Service Web 37
2.2.2 Architecture orientée service 37
2.2.3 Description et spécification des services Web 39
2.2.4 Les points de vues de composition 42
2.3 Les automates temporisés 44
2.3.1 Les contraintes temporelles 44
2.3.2 Sémantique 45
2.3.3 Les problèmes de clôture dans les automates temporisés 46

2.3.4 Automate temporisé déterministe 47
2.3.5 Le model checker UPPAAL 48
2.4 Travaux existants 50
2.4.1 Analyse et vérification des services Web 50
2.4.2 Composition de services Web 55
2.5 Conclusion 60

Chapitre 3
Modélisation des protocoles de conversations temporisées

3.1 Introduction 62
3.2 Présentation informelle des protocoles de conversations temporisées 62
3.2.1 Protocoles de conversations temporisées 63
3.2.2 Les propriétés temporelles dans les protocoles de conversations . . . 65
3.3 Le modèle formel 67
3.3.1 Modélisation du comportement temporisé des services Web 67
3.3.2 Valuation d'horloges 70
3.3.3 Conversations temporisées 71
3.3.4 Sémantique des protocoles de conversations temporisées 71
3.4 Conclusion 72

Chapitre 4
Analyse de l'interopérabilité de services Web asynchrones temporisés

4.1 Introduction 76
4.2 Le problème de compatibilité des services Web asynchrones temporisés . . 77
4.2.1 Aspects non-temporisés 78
4.2.2 Le rôle des propriétés temporelles 81
4.2.3 Classes de compatibilité 83
4.3 Transformation des protocoles de conversations 87
4.3.1 Abstraction des messages 88
4.3.2 Abstraction des contraintes de données 90
4.3.3 Association des invariants 92
4.3.4 Association d'un canal urgent aux transitions 93
4.3.5 Définition des états finaux 94
4.4 Analyse formelle de compatibilité 95
4.4.1 Compatibilité totale parfaite 97

4.4.2 Compatibilité totale non-parfaite . . . 98
4.4.3 Compatibilité partielle non-parfaite . . . 99
4.4.4 Compatibilité partielle parfaite . . . 100
4.4.5 Incompatibilité totale . . . 101
4.5 Implantation de l'analyse de compatibilité dans UPPAAL . . . 102
4.5.1 Compatibilité totale parfaite . . . 102
4.5.2 Compatibilité totale non-parfaite . . . 103
4.5.3 Compatibilité partielle non-parfaite . . . 104
4.5.4 Compatibilité partielle parfaite . . . 105
4.5.5 Incompatibilité totale . . . 106
4.6 Conclusion . . . 108

Chapitre 5
Composition de services Web temporisés

5.1 Introduction . . . 110
5.2 Le problème de composition temporisée . . . 110
5.2.1 Nécessité de mécanismes de composition temporisée . . . 111
5.2.2 Le rôle du médiateur . . . 112
5.3 Les élements clés du cadre de composition . . . 115
5.4 Démarche formelle de composition . . . 116
5.4.1 Construction des connexions P2P temporisées . . . 117
5.4.2 Expliciter les dépendances entre les contraintes temporelles . . . 123
5.4.3 Génération du médiateur temporisé . . . 129
5.5 Exemple récapitulatif . . . 130
5.5.1 Composition sans l'intervention temporisée du médiateur . . . 131
5.5.2 Composition avec l'intervention temporisée du médiateur . . . 133
5.6 Conclusion . . . 135

Chapitre 6
Validation et mise en œuvre

6.1 Introduction . . . 137
6.2 Architecture . . . 138
6.2.1 Le vérificateur de la compatibilité temporisée asynchrone d'une chorégraphie . . . 138
6.2.2 Composition de services temporisée . . . 139

6.3 Les détails d'implantation 140
6.3.1 Représentation des protocoles de conversations temporisés 141
6.3.2 Implantation du vérificateur 142
6.3.3 Implantation de la composition 145
6.4 Conclusion 148

Chapitre 7
Conclusion générale et perspectives

7.1 Synthèse des travaux et résultats 153
7.2 Limites et perspectives 156

Annexe A
Propriètés de l'algorithme de composition temporisée

A.1 Propriété de terminaison 159
A.2 Propriété de sûreté 159

Annexe B Descritpion en WSDL d'un service Web 161

Annexe C Génération du protocole de conversations temporisées de la spécification OWL-S étendue 165

Bibliographie 169

Table des figures

1.1 Le schéma global d'interaction du scenario e-gouvernement 26
1.2 Un exemple d'impact des propriètés temporelles lors de la collaboration de services Web 28

2.1 Publication, recherche et invocation de services Web 38
2.2 La pile des standards des Web services 39
2.3 Chorégraphie et orchestration de services Web 43
2.4 Automate temporisé 45
2.5 Capture d'écran de l'interface graphique du simulateur d'UPPAAL 51
2.6 Capture d'écran de l'interface graphique du vérificateur d'UPPAAL 52

3.1 Comportement du service de la préfecture (SP) 64
3.2 Comportement du service de gestion de la mairie (SM) 65
3.3 Comportement du service de gestion de l'entité médicale (SEM) 66
3.4 Protocoles de conversations des services du e-pension 69

4.1 Des services Web asynchrones compatibles 79
4.2 Incompatibilité due à l' hétérogénéité des types des messages 79
4.3 Incompatibilité due à l'hétérogénéité des contraintes de données 80
4.4 Incompatibilité due à l'hétérogénéité des comportements 80
4.5 Incompatibilité dûe aux propriétés temporelles 81
4.6 Conflit temporisé dû à la différence entre la valeur des horloges 83
4.7 Compatibilité totale 84
4.8 Compatibilité totale non-parfaite 84
4.9 Compatibilité partielle 85
4.10 Compatibilité partielle non-parfaite 86
4.11 Incompatibilité totale 87

4.12 Abstraction des messages 90
4.13 Abstraction des contraintes de données 91
4.14 Association des invariants 93
4.15 Association d'un canal urgent 94
4.16 Simulation des états finaux en UPPAAL 94
4.17 Les automates temporisés supportés par UPPAAL résultants du processus de transformation 96
4.18 Analyseur de compatibilité temporisée 107

5.1 Intervention du médiateur 113
5.2 Intervention du médiateur 114
5.3 Le service client. 116
5.4 Détection de conflits temporels via le processus d'ordannancement d'horloges 127
5.5 Détection de conflits temporels via le processus d'ordonnancement d'horloges 128
5.6 Détection de conflits temporels via le processus d'ordannancement d'horloges 129
5.7 Génération de ASCT sans l'intervention temporisée de médiateur 132
5.8 Génération de ASCT avec l'intervention temporisée du médiateur 134

6.1 L'architecture de l'approche proposée 138
6.2 Analyseur de compatibilité temporisée 139
6.3 L'architecture de l'approche proposée 140
6.4 Extrait de la description XML du protocole de conversations du service SP 143
6.5 Description XML de l'automate UPPAAL généré du service SP 144
6.6 Chargement des protocoles de conversations 146
6.7 Une capture d'écran de la spécification XML des automates UPPAAL générés 147
6.8 Une capture d'écran des propriétés CTL générées 148
6.9 Une capture d'écran du vérificateur de UPPAAL 149
6.10 Description XML du ASCT généré 150
6.11 Description XML du médiateur généré 151
6.12 Evaluation de la performance de l'approche proposée par rapport au nombre des transitions du service client 151
6.13 Evaluation de la performance de l'approche proposée par rapport au nombre des transitions de réception de messages du service client 152

7.1 Nécessité de considérer l'instanciation dynamique dans les interactions des services Web 157

C.1 Représentation de la description du processus atomique étendue en automate 166
C.2 Représentation des opérateurs de OWL-S en automate 167

Introduction générale

De nos jours, le Web qui était initialement un simple moyen de partage d'information, a eu un développement considérable qui lui a permis de devenir un moyen universel pour externaliser les applications des organisations, donnant ainsi naissance au concept d'*intégration d'applications*. En effet, ce concept permet aux organisations de collaborer et d'interagir entre elles, produisant ainsi des processus a valeur ajoutée, sans pour autant mettre en péril leur autonomie. Cependant, assurer une collaboration transparente des applications autonomes reste un défi majeur. L'autonomie des applications implique et engendre une forte hétérogénéité lors d'une collaboration inter-applications. Afin de remédier à ces problèmes d'hétérogénéité, plusieurs méthodes visant l'intégration d'application ont vu le jour tels que les intergiciels (middleware). Un intergiciel est une couche logicielle intermédiaire entre les applications et le réseau, permettant le dialogue entre des applications hétérogènes.

Néanmoins, avec la croissance incessante du besoin d'ouverture des différents systèmes d'informations des différentes organisations ainsi que la familiarisation de l'utilisation de l'Internet, les intergiciels ne répondent plus d'une manière satisfaisante aux besoins d'intégration d'applications qui dépasse l'échelle de l'intranet. En effet, le passage à l'échelle de l'internet est devenu plus que primordial. Afin de répondre aux exigences des clients et du marché à des coûts et dans des délais optimaux, les organisations se sont orientées vers la création de relations de collaboration et de coopération à l'échelle internationale en s'appuyant sur internet. Pour faciliter une telle collaboration inter-organisations, les organisations doivent être capables d'utiliser et de réutiliser les fonctionnalités offertes mutuellement. Ainsi, une nouvelle vision d'intégration et d'interopérabilité basée sur une architecture orientée service (Service Oriented Architecture SOA) est née.

Les architectures à base de services visent à fournir des supports permettant d'abstraire les différentes applications à intégrer par des services qui interagissent les uns avec les autres de manière faiblement couplée. D'un autre coté, l'émergence de l'Internet et le développement technologique sous-jacent se révèle propice aux opportunités de collabora-

tions mondiales et de coopérations inter-organisations. Quand la notion du service repose sur le Web et les technologies associées, on parle de *service Web*.

Les services Web qui sont des composants autonomes, réutilisables et indépendants des plateformes et des langages de programmations ont été créés dans le but de faciliter l'interopérabilité et l'intégration d'applications, via l'échange de messages. Le privilège accordé aux services Web est dû au fait qu'ils reposent sur des interfaces standards basées sur le langage de description XML [53] qui est un standard du W3C [1] (World Wide Web Consortium). Le but derrière l'apparition du langage XML est de fournir un langage de balise qui permette une description standardisée des données qui soit indépendante des plateformes et des langages de programmation. De ce fait, ces dernières années on a assisté à un engouement massif des organisations pour adopter les services Web afin d'externaliser et de mettre en œuvre leur processus métiers, permettant ainsi de créer des applications complexes. Le besoin de créer des applications complexes en combinant ou en composant des applications plus élémentaires surgit, par exemple, quand un client requiert certaines fonctionnalités qu'un service Web seul est incapable d'assurer. Dans ce cas, on parle de *composition de services*.

Dans une composition, les services Web collaborent par l'échange de séquences de messages. A part les séquences des messages, d'autres facteurs affectent le comportement global engendré par une collaboration de services Web. Particulièrement, dans cette thèse nous nous intéressons aux *propriétés temporelles* qui spécifient les délais nécessaires pour échanger des messages. La considération des propriétés temporelles dans la spécification du comportement des services Web apporte un degré plus élevé d'expressivité et donc de flexibilité mais d'un autre coté, apporte aussi un lot de complexité et de difficulté considérable dont il faut tenir compte.

Les propriétés temporelles jouent un rôle important dans les architectures orientées service (SOA). Précisément, dans cette thèse, nous nous intéressons à l'impact de ces propriétés dans le cadre de la composition de services Web. La notion de composition soulève plusieurs problématiques. Parmi celles ci, nous nous intéressons particulièrement à deux aspects : (1) analyser l'interopérabilité que peuvent mener un ensemble de services Web dans le cadre d'une chorégraphie, et (2) comment coordonner des services Web, i.e., construire une composition afin de satisfaire le besoin du client.

L'analyse de l'interopérabilité, appelée *compatibilité*, joue un rôle essentiel dans le

1. W3C est un consortium industriel international fondé en 1994 pour développer des protocoles communs pour l'évolution du web.

cycle de vie des applications complexes. En effet, elle permet de savoir si un ensemble de services Web peuvent mener une interaction correcte. Egalement, ce type d'analyse permet de cerner les éventuelles incompatibilités qui peuvent surgir. Ceci peut être exploité afin d'étudier des mécanismes de génération de médiateurs pour masquer les éventuelles hétérogénéités [19, 14, 63, 62, 103].

Parallèlement à l'analyse de la compatibilité de services Web, les approches de construction de compositions de service Web orientées par le besoin du client sont de même très importantes. La composition de services Web permet de construire un service Web complexe qui puisse répondre aux exigences du client. Le deuxième aspect auquel nous nous intéressons dans cette thèse est l'étude des propriétés temporelles dans le cadre de la construction d'une composition qui soit orientée par le besoin du client. Le but étant de coordonner un ensemble de services Web afin de satisfaire un client.

En résumé, les problèmes auxquels il faut répondre et les aspects auxquels nous nous intéressons dans cette thèse sont multiples. Le premier aspect consiste à définir un modèle qui tienne compte des abstractions nécessaires afin de pouvoir analyser et synthétiser une composition, à savoir les *messages*, les *types de données*, les *contraintes de données* et les *propriétés temporelles*. Le deuxième aspect consiste à proposer une approche d'analyse de compatibilité. Cette analyse vise à caractériser la compatibilité ou la non-compatibilité des services Web et ce en prenant en considération les abstractions précédemment citées. Nous étudions particulièrement l'impact des propriétés temporelles dans une chorégraphie dans laquelle les services Web supportent des communications asynchrones. Nous proposons des techniques qui permettent de détecter les éventuels conflits implicites qui peuvent surgir dans une chorégraphie. Finalement, le dernier problème auquel nous nous intéressons est celui de la construction de composition qui essaye de répondre au besoin du client et ce en prenant en compte les aspects temporels. L'approche que l'on propose est basée sur la génération d'un médiateur pour essayer, quand c'est possible, de masquer les incompatibilités temporisées et non-temporisées qui peuvent surgir lors d'une collaboration. Des mécanismes et des algorithmes ont été développés afin de mettre en œuvre ces objectifs.

Le reste de ce manuscrit est organisé comme suit. Le chapitre 1 adresse le contexte et la problématique du travail que nous proposons. Le chapitre 2 présente un état de l'art qui est constitué de trois parties. La première partie présente les concepts fondamentaux du paradigme de calcul orienté service. Etant donné que le travail présenté dans ce manuscrit repose sur les automates temporisés, nous exposons dans la deuxième partie les

principaux éléments de ce formalisme. La dernière partie est dédiée à l'étude des travaux exitants portant sur l'analyse et la composition de services Web. Le chapitre 3 introduit la modélisation formelle du comportement des services Web (protocoles de conversations) doté de propriétés temporelles et qui est basée sur le formalisme des automates temporisés déterministes. En s'appuyant sur ce modèle, dans le chapitre 4 nous présentons la première contribution de cette thèse qui consiste en l'analyse de compatibilité de services Web temporisés et asynchrones. D'abord, nous introduisons intuitivement le problème de compatibilité de services Web temporisés et asynchrones. Ensuite nous présentons les primitives que nous proposons pour essayer d'apporter une solution à ce problème. Le chapitre 5 détaille la deuxième contribution de cette thèse qui est l'approche de composition de services Web temporisés et asynchrones. Nous présentons ce problème intuitivement, puis nous présentons les primitives que nous proposons pour la mise en œuvre de cette approche. L'étape de la validation de notre travail est présentée dans le chapitre 6. Nous commençons tout d'abord par montrer l'architecture du prototype developpé. Puis, nous exposons les détails d'implantation des différents modules constituants ce prototype. Enfin, nous concluons ce manuscrit en rappelons la contribution de cette thèse et en exposant les limites et les directions de recherches que ce travail a permis d'identifier.

Chapitre 1

Contexte et problématique

Sommaire

1.1 Contexte de la thèse 20

1.2 Problématique 22

1.3 Scénario illustratif 25

1.3.1 Le problème de compatibilité temporisée 27

1.3.2 Le problème de composition temporisée 28

1.4 Apport de la thèse 31

1.5 Conclusion 33

1.1 Contexte de la thèse

L'évolution rapide de la société moderne et sa familiarisation rapide avec l'utilisation des nouvelles technologies a fait apparaître de nouveaux besoins et de nouvelles exigences. Dans la perspective de faire face à ces besoins et à ces exigences, les organisations se sont tournées vers l'automatisation et l'ouverture mutuelle de leurs systèmes d'informations. Le but est d'assurer via des collaborations inter-organisations les prestations requises à moindre effort et à moindre coût. De cela découle la problématique d'intégration et d'interopérabilité des systèmes d'informations hétérogènes et répartis. Des technologies ont été proposées telles que, entre autres, les intergiciels (middelware) [49] qui avaient pour but de masquer les éventuels conflits lors d'une collaboration des systèmes hétérogènes. Ces technologies ont prouvé leur efficacité à l'échelle de l'intranet pour assurer une intégration d'application à application (Application to Application A2A). En effet, dans le cadre d'une intégration A2A, la gestion des applications à intégrer est centralisée, car toutes les applications appartiennent à la même organisation.

Cependant, de nos jours, afin de satisfaire des besoins, une variété de services offerts par différentes organisations peuvent être requis. Par conséquent, les outils développés pour assurer des interactions A2A ne sont plus satisfaisants pour assurer une interaction métier à métier (business to business B2B). Ce mode d'interaction vise l'intégration d'applications qui appartiennent à des organisations différentes. Ce nouveau point de vue de collaboration et d'interopérabilité inter-organisations a suscité beaucoup d'intérêt et a donné lieu au paradigme SOC (Service Oriented Computing) [121, 107]. Ce paradigme repose principalement sur le concept de *service* et le but est de soutenir principalement le développement rapide, simple et peu coûteux des applications distribuées pour des environnements hétérogènes en se basant sur le concept de *service*. Ainsi, les systèmes d'informations évoluent vers des architectures à base de services (Service Oriented Architecture SOA) [46, 4, 21].

Selon l'architecture SOA, les processus métiers des entreprises sont décomposés en un ensemble de fonctions, appelées *services*, qui peuvent interagir par le biais d'échange de messages. Quand ces services reposent sur le Web, on parle alors de *services Web*. Un service Web est un composant logiciel indépendant de la plateforme d'exécution et qui repose sur les technologies du Web afin d'interagir. En effet, du fait de la maturation du Web et du développement accéléré des technologies sous-jacentes, le passage de l'intégration d'applications à l'échelle de l'internet est devenu plus que nécessaire.

L'une des notions importantes qu'offrent les services Web est celle de *composition*.

Cette dernière permet de combiner des services Web afin de produire un service complexe. Le service a forte valeur ajoutée résultant de la combinaison de services Web peut aussi à son tour participer à la création d'autres services plus complexes. Cependant, comme pour toute technologie, si la composition est très prometteuse et apporte des avancées dans le domaine d'intégration d'application B2B, elle apporte aussi des problèmes sous-jacents. Ces problèmes sont principalement dûs au fait que les services Web sont développés d'une manière autonome par, éventuellement, différentes organisations. Dès lors que les services Web sont impliqués dans une composition, des problèmes d'hétérogénéité ressurgissent. Afin de savoir si et comment les services Web des différentes organisations peuvent collaborer, la spécification des services Web doit être plus riche qu'une simple description de leurs interfaces. L'interface d'un service Web, spécifiée par le langage WSDL (Web Service Description Language), décrit la signature de l'ensemble des opérations que le service offre. Pour comprendre comment un service Web peut être utilisé, sa spécification doit également inclure la description de son comportement. On entend par comportement, les séquences de messages, appelée *conversations*, qu'un service Web supporte. L'ensemble des conversations constitue ce que l'on appelle un *protocole de conversation* [17, 23, 18, 63, 64]. Un protocole de conversation peut être spécifié en utilisant les standards proposés ces dernières années tels que, entre autres, BPEL (Business Process Execution Language [1]), OWL-S (Semantic Markup for Web Services [91]), WSCL (Web Services Conversation Language) [12]. En outre, dans des applications réelles, d'autres facteurs affectent l'interopérabilité des services Web. Particulièrement, nous nous intéressons aux propriétés temporelles qui permettent de spécifier les délais nécessaires pour échanger des messages. Dans ce contexte, des travaux préliminaires affirment la nécessité d'étendre les protocoles de conversations par des propriétés temporelles.

L'idée d'une collaboration transparente inter-organisations ne doit pas porter atteinte au développement autonome des différents services. Toutefois, dès lors que ces services autonomes collaborent ensemble dans la perspective d'une éventuelle composition, des problèmes de compatibilité (interopérabilité) surgissent. On distingue deux types de problèmes : les problèmes de compatibilité non-temporisés et les problèmes de compatibilité temporisés. Les problèmes non temporisés sont principalement dûs aux divergences des types des messages, des types de données, des contraintes de données, et l'ordre dans lequel les messages peuvent être échangés. Le deuxième type de problème est dû aux propriétés temporelles. Les différentes propriétés temporelles sont locales et complètement indépendantes les unes des autres. Cependant, lors d'une interaction, les propriétés tem-

porelles de chaque service peuvent avoir implicitement un impact sur le comportement d'un autre service. Ainsi, des conflits entre les différentes propriétés temporelles peuvent exister. Donc, une première phase qui doit précéder la mise en œuvre de la composition est l'analyse de la compatibilité temporelle. Ce type d'analyse permet de comprendre si un ensemble de services Web peut collaborer correctement.

Parallèlement à l'analyse de l'interopérabilité des services Web, fournir des outils permettant de créer une composition afin de satisfaire les besoins des clients est très important. Dans des applications réelles, fournir des outils et des primitives qui permettent de créer des compositions temporisées sur-mesure, tout en essayant de masquer les éventuels échecs (temporisés et non-temporisés) afin de répondre à des fonctionnalités attendues par le client, est de nos jours un problème qui fait couler beaucoup d'encre.

C'est dans ce contexte d'analyse et de composition de services Web temporisés que cette thèse s'inscrit.

1.2 Problématique

La propriété de composition qu'offre la technologie des services Web est l'un des aspects les plus promoteurs de l'architecture SOA (Service Oriented Architecture). Elle permet de créer des applications complexes en combinant ou en composant des services Web plus élémentaires. L'intérêt majeur de cette propriété réside dans le fait que souvent, un seul service Web est incapable de fournir les fonctionnalités requises. Néanmoins, la réalisation d'une application complexe via une composition de services Web distribués et autonomes nécessite des investigations qui permettent d'assurer que l'aboutissement de la composition est bien une application correcte. En effet, les services Web impliqués dans une composition sont conçus et implantés indépendamment les uns des autres et non dans la perspective de participer à une composition bien spécifique.

Dans cette thèse, nous nous intéressons à l'étude des propriétés temporelles dans le cadre de la composition de services Web. Le comportement des services Web dans des applications réelles ne dépend pas seulement des messages, mais dépend aussi d'autres propriétés quantitatives telles que les propriétés temporelles. Ainsi, la construction d'une bonne composition exige d'examiner les séquences des messages augmentées de propriétés temporelles. Peu de travaux récents ont montré l'importance de considérer ce type de propriétés dans les services Web [43, 79, 78, 61].

Lors d'une composition, des incohérences peuvent se manifester. Donc, avant de se

lancer dans la construction d'une composition, il est tout à fait souhaitable de savoir si l'ensemble des services Web est capable de mener une collaboration correcte. Cette vérification, appelée *analyse de compatibilité*, permet d'une manière précoce de savoir si la composition est possible du point de vue de leurs conversations, i.e., échanges de messages. Des travaux préliminaires d'analyse de compatibilité de services Web ont été proposés ces dernières années [32, 15, 16, 113, 115]. Le but est d'affirmer si deux services Web sont capables d'échanger des messages. Tous ces travaux ne considèrent que des services Web synchrones. Cependant, la nature des systèmes distribués en général et en particulier les services Web est asynchrone. Par conséquent, le domaine d'application de ces approches est très restrictif. En outre, les travaux qui considèrent les propriétés temporelles permettent de découvrir des conflits temporisés d'un type particulier.

A l'instar des mécanismes d'analyse et de vérification des services Web, fournir des primitives de synthèse d'une composition orientée client est, de nos jours, très important. Ce problème a été abordé par plusieurs chercheurs.

Comme cité auparavant, les propriétés temporelles jouent un rôle très important dans le paradigme des services Web et en particulier dans la composition de services Web. Les travaux existants de composition ne considèrent pas les propriétés temporelles lors de la construction de la composition. En outre, les auteurs des travaux présentés dans [24, 27, 66, 101, 7, 29, 10, 23, 26] supposent qu'une description de l'ensemble des opérations de la composition requise est donnée. Ensuite, les auteurs vérifient si cette description correspond parfaitement à la description des services Web. Cependant, le problème d'applicabilité de ces approches dans des scénarios de composition réels est ouvert. En effet, si on suppose que le client est capable de spécifier une description de l'ensemble des opérations que la composition doit effectuer, chaque opération peut être satisfaite par une séquence d'opérations de l'ensemble des services Web. De ce fait, la vérification du matching (correspondance) total de la description de la composition avec l'ensemble des services Web n'est pas assez flexible. Un simple client n'a pas toutes les informations sur les services Web impliqués et donc il ne peut pas spécifier son besoin en se basant sur les opérations des services Web de telle sorte qu'une correspondance exacte existe. Afin de remédier à ces limites, nous avons besoin de définir un modèle plus flexible qui permette une *composition temporisée* qui dépasse l'échelle de la délégation.

Etant donné que les services Web sont hétérogènes, du fait qu'ils sont développés d'une manière autonome, la coordination peut échouer et par conséquent la composition échoue. Plusieurs alternatives sont envisageables pour remédier aux éventuelles hétérogé-

néités. La première consiste à essayer d'homogénéiser les services Web. Cette solution est onéreuse, étant donné que le même service peut être impliqué dans plusieurs compositions différentes. Une autre alternative consiste à générer des services intermédiaires, appelés *médiateurs*, qui visent à masquer les éventuelles hétérogénéités. Vu la prolifération des services Web, analyser et composer manuellement des services Web est devenu une tâche complexe, ce qui peut induire facilement en erreur. En conséquence, l'utilisation de méthodes formelles automatiques est devenue cruciale dans le cycle de vie de développement des applications complexes. Le nombre des méthodes formelles proposées ces dernières années en est témoin.

Pour pouvoir analyser et composer automatiquement des services Web, nous avons besoin de définir un modèle qui tienne compte des abstractions nécessaires, à savoir, les types de messages, les types de données, les contraintes de données, et les propriétés temporelles. Parmi les formalismes existants, nous avons opté pour les machines à états finis. Ce formalisme a été largement utilisé afin de modéliser les systèmes réels. Pour modéliser les propriétés temporelles, nous utilisons les horloges telles que définies dans les automates temporisés [6]. Dans le contexte de notre travail, les machines à états finis sont suffisamment expressives pour spécifier les propriétés que l'on considère (les séquences de messages, les données, les contraintes de données et les propriétés temporelles).

Pour récapituler, l'objectif de la thèse est de proposer un modèle de composition qui permette de modéliser, d'analyser, et de composer des protocoles permettant l'interaction entre les plusieurs services Web. Ainsi, la thèse tentera de répondre aux trois questions suivantes en ce qui concerne l'analyse de la compatibilité et la synthèse de composition temporisées :

- Comment définir un modèle qui permette de prendre en considération les propriétés que l'on considère, à savoir, les séquences de messages, les types de données, les contraintes de données, et les propriètés temporelles, c'est-à-dire trouver quelles sont les abstractions nécessaires à modéliser, et comment les modéliser ?
- Comment savoir si un ensemble de services Web temporisés sont compatibles et comment caractériser le niveau d'interopérabilité qu'ils peuvent atteindre dans une chorégraphie ?
- Comment créer une composition temporisée qui réponde à des fonctionnalités attendues tout en essayant, quand c'est possible, de masquer les éventuelles incompatibilités ?

Il est clair que ces objectifs sont étroitement liés, et que nous visons la définition

d'un modèle permettant à la fois d'analyser et de composer des services Web temporisés asynchrones.

1.3 Scénario illustratif

Afin d'illustrer la problématique évoquée précédemment, dans cette section, nous introduisons un scénario simplifié illustratif, inspiré de [97], dans lequel une application d'attribution de pensions à des handicapés est conçue (voir Figure 1.1). Cette application nécessite la collaboration de trois organisations : (1) la préfecture, (2) une entité médicale, et (3) la mairie.

Nous supposons que ces organisations sont gérées, respectivement, par :

- Le Service de la Préfecture (SP) qui assure la gestion de plusieurs services qu'offre la préfecture. Dans l'exemple simplifié que nous considérons, ce service gère l'attribution de bourses et la délivrance du permis de conduire à des handicapés.
- Le Service d'Entité Médicale (SEM) qui gère les services qu'offrent les entités médicales, tels que la négociation des dates de rendez vous avec les patients, la récolte des informations médicales.
- Le Service de la Mairie (SM) régit l'ensemble des fonctionnalités qu'offre une mairie, à savoir, l'établissement des attestations de domiciles, des extraits de naissances et bien d'autres services.

Dans ce scénario, le citoyen présentant un handicap sollicite une bourse du gouvernement. Afin d'entamer la procédure, le citoyen demande le formulaire correspondant à la préfecture. Une fois que le citoyen reçoit le formulaire, il le remplit et l'envoie à la préfecture. Cette dernière sollicite l'entité médicale afin d'examiner le handicap que le citoyen présente. L'entité médicale contacte par la suite le citoyen et négocie une date de rendez vous. Une fois le rendez vous fixé, et après avoir examiné le citoyen, l'entité médicale établit un rapport d'examen et le communique à la préfecture. Entre temps, la préfecture demande à la mairie d'établir une attestation de domiciliation du citoyen. Une fois que l'attestation de domiciliation ainsi que le rapport médical sont reçus, la préfecture prend la décision finale.

Chaque service spécifie le comportement qu'il supporte en décrivant les types des messages, les types des données, les contraintes de données, et l'ordre dans lequel il peut échanger ces messages (c-à-d., les séquences de messages supportées). L'ensemble des séquences, encore appelées *conversations*, forment ce que l'on appelle un *protocole de*

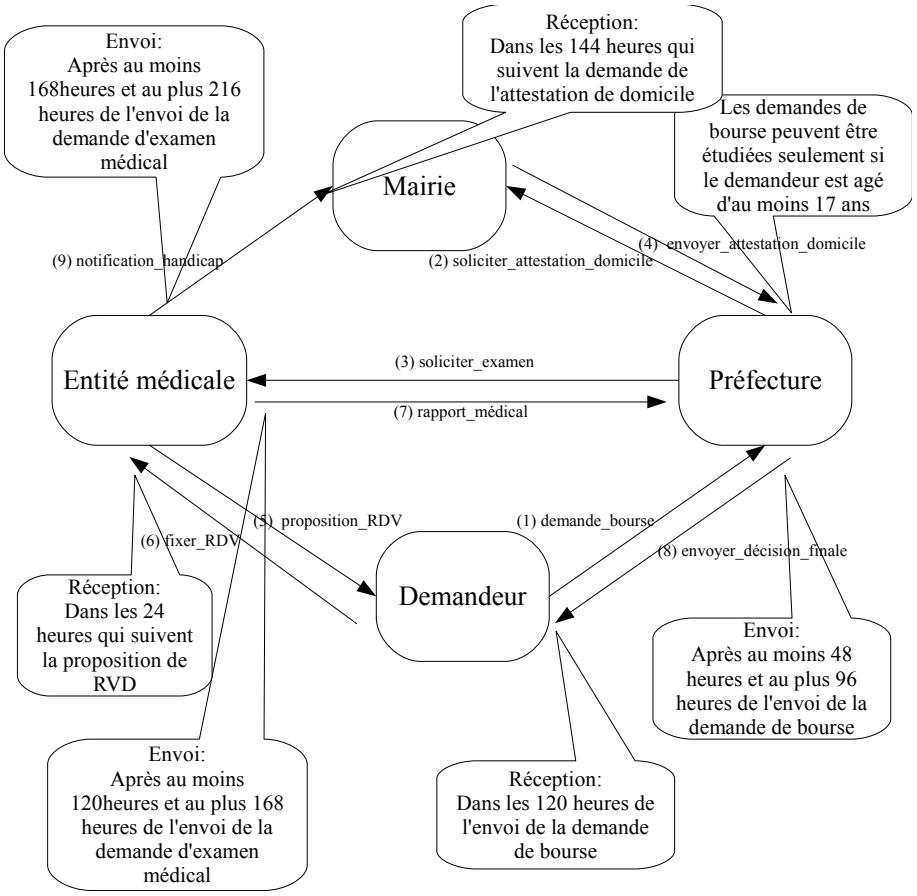


FIGURE 1.1 – Le schéma global d'interaction du scenario e-gouvernement

conversation. Parallèlement à ces propriétés conversationnelles qui ont un impact dans la réalisation d'une collaboration, d'autres propriétés telles que les propriétés temporelles jouent un rôle important, du fait que les messages peuvent/doivent être échangés dans des délais bien précis. Les protocoles de conversations dotés de ces propriétés temporelles sont appelés *protocoles de conversations temporisées*.

Les différents services sont contraints de respecter les différentes propriétés temporelles de chaque service. Dans ce qui suit, nous en citons quelques unes.

- Une fois que l'entité médicale propose des dates pour le rendez-vous au citoyen, elle doit recevoir la confirmation dans les 24 heures qui suivent.
- Une fois que la demande de bourse est reçue, la préfecture envoie sa décision finale au citoyen, après au moins 48 heures et au plus 96 heures.
- Le rapport médical peut être envoyé à la préfecture après au moins 120 heures et au plus 168 heures de l'envoi de l'examen médical.
- ...

Parallèlement aux contraintes temporelles, les services Web peuvent avoir des contraintes sur les données, par exemple, la préfecture spécifie que la demande de bourse ne peut être étudiée que si le demandeur est agé d'au moins 17 ans.

Dans la section suivante, nous discutons le problème de vérification de la compatibilité des services Web d'une marnière intuitive.

1.3.1 Le problème de compatibilité temporisée

Les propriétés temporelles (qui sont des contraintes) ont un impact considérable sur les interactions des services Web. Ceci dépend principalement des points suivants :

- Le type de communication que les services Web supportent, i.e., les services Web communiquent ils d'une manière synchrone ou asynchrone ?
- Les contraintes temporelles possèdent-elles toutes le même point de repère ou chaque contrainte a-t-elle un point de repère différent des autres ? On entend par point de repère le moment où le temps commence à s'incrémenter.

Dans ce travail, nous supposons que les services Web sont asynchrones et les différentes contraintes temporelles peuvent avoir des points de repère différents. Donc, pour détecter les éventuelles incompatibilités lors d'une collaboration, il n'est pas suffisant de vérifier les contraintes temporelles associées à l'envoi du message avec les contraintes associées à sa réception, comme cela peut être appliqué sur les contraintes de données. Par exemple, si on considère le service préfecture (SP) et le demandeur de la Figure 1.1, la décision finale peut être envoyée par la préfecture après au moins 48 heures et au plus 96 heures de l'envoi de la demande de bourse. Cette décision peut être reçue par le demandeur dans les 120 heures à partir de l'envoi de la demande de bourse. En vérifiant ces deux contraintes, la décision finale de la préfecture peut être échangée (i.e., la préfecture l'envoie et le demandeur la reçoit). Donc, si on vérifie les services deux à deux, en comparant les contraintes temporelles associées à l'envoi et à la réception des messages, le scenario que l'on considère va être désigné comme étant correct.

Cependant, si on examine bien le déroulement des interactions, on peut remarquer qu'afin que la préfecture puisse envoyer sa décision finale, il faut qu'elle reçoive le rapport de l'entité médicale. Ce rapport peut être envoyé après au moins 120 heures et au plus 168 heures de l'envoi de la demande de l'examen médical. En d'autres termes, la préfecture ne peut recevoir le rapport qu'après 120 heures de l'envoi de la demande d'examen. Sachant que la préfecture ne peut envoyer sa décision finale qu'après avoir reçu le rapport médical, i.e., après 120 heures, alors la décision finale ne peut pas être envoyée dans les 96 heures

à partir de l'envoi de la demande de bourse. Sur la Figure 1.2, on présente explicitement l'interaction des services Web qui contient un blocage. La préfecture envoie sa décision finale après au moins 48 heures et au plus 96 heures à partir de l'envoi de la demande de la bourse. Mais durant cette exécution, il y a un intervalle d'au moins 120 heures pour fournir le rapport d'examen médical.

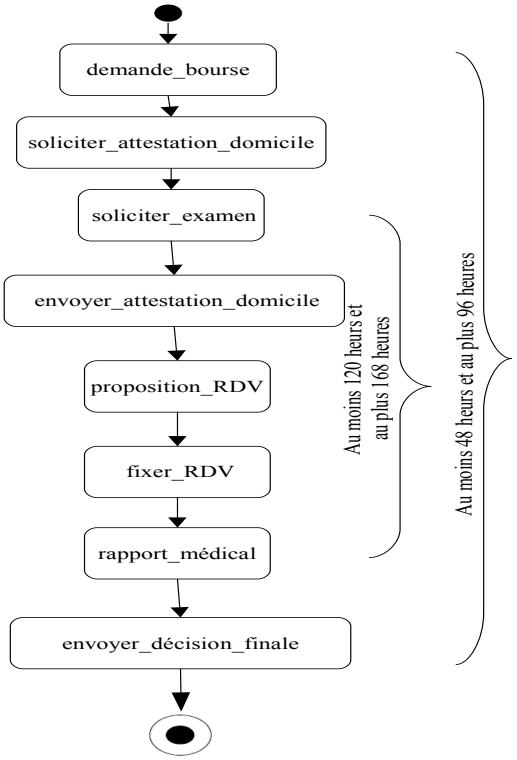


FIGURE 1.2 – Un exemple d'impact des propriètés temporelles lors de la collaboration de services Web

Avec cet exemple simple et intuitif, on constate qu'un service Web peut ne pas être dans la mesure de respecter ses propres contraintes, car lors d'une collaboration, les contraintes d'autres services peuvent avoir un impact sur son exécution. Dans la section suivante, nous discutons le problème de composition temporisée et la génération de médiateur.

1.3.2 Le problème de composition temporisée

Les services Web peuvent être impliqués dans des compositions différentes. Par exemple, le service de préfecture peut être invoqué pour attribuer des bourses ou pour délivrer des permis de conduire à des handicapés. Donc, l'implication d'un service Web dans une composition est toujours dans le but de satisfaire un besoin bien particulier. D'où l'importance

de fournir des primitives de composition qui soient orientées par le besoin du client.

Dans un tel cadre, lors de la construction d'une composition, des blocages peuvent surgir, comme par exemple, le problème temporel présenté ci-dessus. Comme abordé précédemment, une alternative qui consiste à essayer de générer un service intermédiaire (i.e., médiateur) peut être envisagée. Le rôle de ce médiateur serait de tenter, quand c'est possible, de contourner les blocages i.e., les masquer. Ceci, en créant les connexions requises entre les services. Ci-après, nous présentons intuitivement un exemple dans lequel le médiateur peut intervenir pour tenter de masquer les hétérogénéités temporisées et/ou non-temporisées.

Problème temporisé

Revenons à la Figure 1.1. Comme on peut voir, la mairie attend le rapport de l'entité médicale pour pouvoir par exemple établir une carte d'handicapé. La mairie exige que le rapport soit rendu dans les 144 heures qui suivent la demande de l'attestation de domicile pour une bourse d'handicapé. L'entité médicale peut envoyer son rapport à la mairie après 168 heures. Donc la mairie ne pourra pas recevoir le rapport à cause des contraintes temporelles. Sans aucune intervention externe, la mairie ne pourra pas délivrer la carte au demandeur et donc la composition échoue.

Si on examine la situation, on peut remarquer que l'entité médicale envoie son rapport à la préfecture après 120 heures. Donc, le rapport médical devient disponible après 120 heures. Intuitivement, il suffit d'essayer de créer un lien indirect entre l'entité médicale et la mairie pour faire parvenir le rapport médical à la mairie. Ce lien indirect peut être crée en générant un médiateur qui a son tour récupère le rapport disponible et génère le message d'envoi de ce rapport à la mairie.

Le médiateur peut échouer quand les données requises, i.e., les donnés impliquées dans le message manquant (i.e., le message qu'il faut générer) ne sont pas disponibles. Une donnée est dite indisponible si elle n'a pas été calculée et échangée avant que le message manquant ne soit requis.

Problème non-temporisé

Les problèmes non-temporisés sont engendrés par les différences au niveau des types des messages, des types des données, des contraintes de données, ou de l'ordre dans lequel les messages peuvent être échangés.

1. *Divergences au niveau des types des messages et/ou des types des données* : Quand un service attend un message impliquant un ensemble de données et en même temps ce message ne peut pas être envoyé par un autre service, alors de la même manière, on essaye de voir si les données qu'implique le message attendu sont disponibles pour générer le médiateur qui s'occupe de la génération du message manquant.

   Par exemple, supposons que le service de préfecture attend le message *attestation_Domicile* qui a comme paramètres (i.e., les données) *numéro_securité_sociale* et *attestation*. Si le service de la mairie envoie le message *attestation_Domicile* qui a comme paramètres *attestation*, *nom*, *prénom* et *adresse* alors le service de préfecture reste bloqué et la composition échoue. Donc en suivant le principe décrit ci-dessus, on vérifie si les données impliquées dans le message *attestation_Domicile* sont disponibles, i.e., elles ont été déjà calculées et échangées auparavant. Si c'est le cas, le médiateur récupère les données et génère le message manquant.

2. *Inter-blocage comportemental* : Ce type de blocage peut arriver quand deux services attendent chacun un message de l'autre pour pouvoir envoyer un message. Par exemple, si on suppose que la mairie attend un message de demande de la carte d'handicapé pour notifier de la disponibilité de cette carte et en même temps le demandeur attend la notification de la mairie pour envoyer une confirmation. Dans ce cas, les deux services restent bloqués dans l'attente des messages. En suivant le même principe, on vérifie si les données sont disponibles pour générer les messages attendus.

3. *Divergence des contraintes de données* : dans certains cas, un service Web peut envoyer un message si des contraintes de données (i.e., des conditions sur la valeur des données) sont satisfaites. En même temps, la réception de ce message peut être conditionnée par une contrainte de données. Si ces deux contraintes sont disjointes, alors un blocage surgit. Par exemple, supposons que la préfecture demande un examen médical pour handicapés agé de plus que 17 ans. Si le service d'entité médical régit les services d'une pédiatrie qui prend en charge que des enfant de moins de 15 ans, alors la composition échoue. On note que dans ce cas particulier, le médiateur ne peut en aucun cas contourner le blocage. Une éventuelle solution serait de changer (remplacer) le service découvert.

Pour récapituler, lors de la construction d'une composition, on doit s'assurer que tous les blocages temporisés et non-temporisés peuvent être détectés. Quand des blocages se manifestent, nous envisageons l'alternative de tenter de générer automatiquement un mé-

diateur pour essayer de contourner les éventuels problèmes.

Tout au long de ce manuscrit, nous utilisons l'exemple présenté dans cette section afin d'expliquer les différents aspects que nous abordons dans cette thèse. Les protocoles de conversations peuvent être décrits avec plusieurs langages notamment BPEL [1] ou OWL-S [91]. Cependant, ces langages et ces spécifications ne permettent pas d'effectuer des raisonnements sur les services Web comme analyser si un ensemble de services Web peut mener une interaction cohérente et qui ne contient pas de blocage. Afin de palier ces limites, plusieurs approches formelles, reposant sur, entre autre, le formalisme des machines à états finis [26, 28, 11, 37, 31, 55, 54], sur l'algèbre de processus [38, 35, 8, 52, 82] et sur les réseaux de pétri [105, 68, 126, 130, 137, 73, 134, 135, 40, 72], ont été proposées. Le niveau d'expressivité de chaque formalisme est relatif et dépend des propriétés que l'on veut modéliser.

Dans le contexte de notre travail, nous adoptons les machines à états finis pour modéliser les protocoles de conversation. En outre, afin de caractériser les propriétés temporelles que l'on considère, nous nous basons sur les horloges définies dans les automates temporisés [6].

1.4 Apport de la thèse

Le but que nous visons dans cette thèse est de définir un modèle de *composition de services Web munis de propriétés temporelles*. Comme cité précédemment, nous nous intéressons particulièrement aux propriétés temporelles qui permettent de spécifier les délais nécessaires pour échanger des messages. Pour ce faire, la première étape consiste à définir un modèle formel qui permette de spécifier et de modéliser les propriétés requises afin d'analyser et composer des services Web. Dans ce contexte, plusieurs modèles ont été proposés. Certains d'entre eux considèrent seulement les types des messages que les services Web peuvent échanger sans considérer les types des données et les propriétés temporelles (par exemple [18, 17]), d'autres considèrent les opérations qu'ils peuvent effectuer sans prendre en considération les propriétés temporelles ([26, 28, 23, 24, 66, 29]). D'autres travaux considèrent des propriétés temporelles dans l'analyse de la compatibilité/remplaçabilité de services Web synchrones sans la prise en compte des types de données et la nature asynchrone des services Web [15, 16, 115, 113].

Comme cité auparavant, une première étape qui doit précéder la composition de service Web est l'analyse de l'interopérabilité. Ce type d'analyse peut être utile dans le cycle de vie

des services dans plusieurs phases. Par exemple, elle peut être utilisée dans le processus de découverte de services Web pour savoir si les services découverts sont compatibles ou pas. Egalement, cette analyse permet de savoir à quel degré les services Web sont composables, i.e., les services sont-ils-totalement composables, partiellement composables ou totalement incomposables.

L'analyse de compatibilité est importante dans le cycle de vie de la composition de services Web. Mais en parallèle, afin de produire des services complexes qui satisfassent des besoins bien particuliers des clients, on a besoin de définir des mécanismes et des primitives de composition. La dernière partie de cette thèse est dédiée à l'étude des propriétés temporelles dans le problème de composition.

Afin d'atteindre ces objectifs, nous proposons une démarche qui repose sur les points suivants :

1. *Proposition d'un modèle formel de spécification du comportement des services Web augmenté de propriétés temporelles* : La première étape de notre travail consiste à définir un modèle formel permettant la spécification des protocoles de conversation qui prenne en compte les séquences de messages, les types de données, les contraintes de données, les propriétés temporelles et l'aspect asynchrone des services.
2. *Proposition d'une approche d'analyse de compatibilité de services Web* : La deuxième partie de ce travail est dédiée à l'analyse de la chorégraphie de services Web. Dans la littératures, les travaux visant l'analyse de compatibilité [18, 17, 15, 16, 115, 113] supposent que les services Web ne supportent que des communications synchrones [18, 17, 15, 16, 115, 113, 59]. Cependant, les systèmes distribués en général et en particulier les services Web sont souvent de nature asynchrone. Par conséquent, l'application de ces approches est restreinte et ne peut pas être appliquée en général afin de détecter les éventuels conflits temporisés et non-temporisés. Afin de remédier à ces limitations, nous avons proposé une approche basée sur le model checking.
3. *Proposition d'une approche de composition de services Web temporisés* : A l'instar de l'analyse des chorégraphies de services Web, fournir des mécanismes de composition est très important pour ne pas dire primordial dans le cycle de vie de construction des applications complexes reposant sur des services. Les approches de composition proposées dans la littérature (par exemple [111, 30, 36, 102, 75, 123, 37, 110, 26, 23, 24, 66, 63, 64, 7, 101, 10, 29, 28]) ne prennent pas en considération les propriétés temporelles. En outre, dans ces travaux, la composition de services Web est réduite

au problème de délégation d'opérations. Pour calculer une composition de services Web, les auteurs (par exemple [24, 27, 66, 101, 7, 29, 10, 23, 26]) supposent que la description abstraite de la composition souhaitée est donnée. Ensuite, le problème de composition se réduit à trouver pour chaque opération de la description abstraite, un service qui la réalise. Néanmoins, ces approches ne peuvent être appliquées que dans un contexte intra-organisationnel, où on connait les opérations des différents services. Par contre, l'application de ces approches à l'échelle de l'Internet est toujours un problème ouvert. Une dernière contribution que nous avons apportée dans le cadre de cette thèse est la proposition d'un cadre de composition de services Web temporisés orientée par les données.

1.5 Conclusion

Les services Web figurent parmi les technologies sur lesquelles les architectures distribuées s'appuient afin de permettre aux applications de se composer pour fournir des services plus complexes. Ce problème de composition a suscité l'intérêt de plusieurs chercheurs. La composition permet de profiter de la variété de fonctionnalités qu'offrent les services les plus élémentaires pour les incorporer en un service plus complexe, permettant ainsi de répondre aux exigences que les services élémentaires ne peuvent pas satisfaire.

Plusieurs facteurs jouent un rôle important dans la composition de services Web. Particulièrement, dans ce chapitre nous avons évoqué l'intérêt que nous apportons aux propriétés temporelles dans le cadre de la composition de services Web qui représente le contexte général de notre travail. En présence de ces propriétés temporelles, nous avons rapporté l'importance des approches formelles d'analyse et de composition.

Ensuite, nous avons exposé la problématique de notre travail qui consiste en l'analyse et la composition de services Web autonomes, temporisés et asynchrones. L'analyse consiste à savoir si les services Web temporisés peuvent collaborer correctement. En outre, cette analyse doit permettre de définir le type d'interopérabilité que peuvent mener les services Web. Etant donné que des hétérogénéités peuvent surgir lors de la collaboration de services Web, cette analyse seule ne permet pas de produire un schéma de composition qui soit correct. Pour cela, nous avons introduit la nécessité de primitives de composition temporisées qui soient capable de masquer, quand c'est possible, les éventuels conflits. Via un scénario illustratif d'attribution de bourses à des handicapés, nous avons discuté informellement et intuitivement ces problèmes. Enfin, nous avons présenté les démarches

et les contributions que nous apportons dans cette thèse.

Chapitre 2

Etat de l'art

Sommaire

2.1 Introduction 36

2.2 Le paradigme de calcul orienté service 36

2.2.1 Service Web 37

2.2.2 Architecture orientée service 37

2.2.3 Description et spécification des services Web 39

2.2.4 Les points de vues de composition 42

2.3 Les automates temporisés 44

2.3.1 Les contraintes temporelles 44

2.3.2 Sémantique 45

2.3.3 Les problèmes de clôture dans les automates temporisés 46

2.3.4 Automate temporisé déterministe 47

2.3.5 Le model checker UPPAAL 48

2.4 Travaux existants 50

2.4.1 Analyse et vérification des services Web 50

2.4.2 Composition de services Web 55

2.5 Conclusion 60

2.1 Introduction

L'évolution incessante de l'informatique et des technologies sous-jacentes a donné naissance à plusieurs paradigmes tel que le paradigme de calcul orienté service (Service Oriented Computing SOC). Dans ce paradigme, les services Web sont une manière de mettre en œuvre des applications complexes et distribuées. Dans la première partie de ce chapitre, nous rappelons comment le développement des systèmes d'information ces dernières années a abouti au paradigme SOC. Nous présentons les concepts fondamentaux de ce paradigme et en particulier la technologie des services Web qui est le principal composant de l'architecture orientée services. Nous abordons également les différents niveaux de description des services Web, à savoir, la description structurelle et comportementale. Ces descriptions sont cruciales pour la composition de services, que nous adressons à la fin de la première partie de ce chapitre. Quant à la deuxième partie, elle est dédiée à l'étude des automates temporisés sur lesquels cette thèse se base. Avant de conclure, nous exposons les travaux relatifs à la problématique d'analyse et de composition de services Web.

2.2 Le paradigme de calcul orienté service

Pour dissimuler la complexité d'intégration d'applications autonomes et hétérogènes, trois architectures par composants ont été proposées [131, 118] : CORBA, EJB(Enterprise Java Beans), et .NET. Dans ces technologies à base de composants distribués, le couplage entre les objets est fort. Par exemple, la norme CORBA de l'OMG qui permet de manipuler des objets à distance avec n'importe quel langage, nécessite une connaissance de la structure des objets manipulés par les applications clientes et par les applications fournisseurs. Des règles de transformation objets/messages doivent être définies au préalable par les partenaires [3]. En outre, étant donné que chaque architecture propose sa propre infrastructure, on ne peut assembler que des objets CORBA (ou COM) entre eux. Par conséquent, la mise en œuvre de ces architectures dans le cadre d'une infrastructure ouverte telle que Internet soulève d'énormes difficultés.

Dans le but de palier les limites de ces architectures, les efforts de conception ont donné lieu au concept d'*architecture orientée service* (*Service Oriented Architecture SOA*) [107]. Cette architecture exploite le concept de *service* comme le principal élément autour duquel les applications complexes sont développées. L'idée de base consiste à encapsuler les fonctionnalités qu'offrent les organisations sous forme de services. Ainsi, l'intégration et la gestion des applications se concentrent au niveau des fonctionnalités en cachant les détails

de l'implémentation.

D'autre part, la maturité de l'Internet, le développement technologique associés et les efforts de standardisation, ont provoqué un engouement massif des organisations pour utiliser ce moyen de collaboration et de partage. Quand la notion de service repose sur les technologies du Web, on parle alors de *Service Web*.

2.2.1 Service Web

Intuitivement, un service Web est un composant logiciel accessible via le Web dont le but est d'offrir des fonctionnalités qui peuvent être requises par des clients. Il repose sur le Web comme infrastructure pour gérer la communication entre les différents composants (services), et ce en se basant sur plusieurs standards pour la description de leurs caractéristiques fonctionnelles et non fonctionnelles, comme par exemple WSDL [132], BPEL [1], et WSCL [12], et sur un standard pour la définition des formats des messages échangés entre les services (SOAP [57]). Les interactions entre les services Web se font par envoi de messages. Ainsi, cette technologie permet de faire communiquer des composants entre eux, indépendamment de la plateforme sur laquelle ils sont hébergés, en utilisant un protocole de communication et un format de fichier unique et standardisé basé sur XML. C'est cet avantage majeur qui a assuré la propagation de l'utilisation des services Web comme l'une des technologies les plus utilisées pour construire des applications distribuées.

Le consortium W3C définit un service Web comme étant une application ou un composant logiciel qui vérifie les propriétés suivantes [81] :

- Il est identifié par un URI ;
- Ses interfaces et ses liens (binding) peuvent être décrits en XML ;
- Sa définition peut être découverte par d'autres services Web ;
- Il peut interagir directement avec d'autres services Web à travers le langage XML et en utilisant des protocoles Internet

La notion de service est l'élément de base de l'architecture orientée service présentée ci-après.

2.2.2 Architecture orientée service

Les architectures orientées service SOA supportent un modèle de développement qui permet aux services Web d'êtres publiés, de se découvrir et de s'invoquer indépendamment des plateformes et des langages de programmation [80]. Les principaux éléments interve-

nant dans cette architecture sont illustrés sur la Figure 2.1. Le rôle de chaque élément est décrit comme suit :

- *Le fournisseur du service* : il désigne l'entité propriétaire du service. D'un point de vu technique, un fournisseur peut désigner la plateforme d'accueil du service.
- *Le client du service* : correspond au demandeur de service. D'un point de vue opérationnel, c'est le service client qui sollicite et invoque le service requis.
- *L'annuaire des services* : correspond à un registre de description de services offrant des facilités de publication de services à l'intention des fournisseurs ainsi que des facilités de recherche de services à l'intention des clients [81]. En d'autres termes, l'annuaire joue le rôle d'intermédiaire entre les clients et les fournisseurs de services. En effet, un service Web est implanté, décrit, et publié par un fournisseur dans un annuaire UDDI [125]. Le client peut donc explorer cet annuaire pour retrouver et sélectionner un service qui implémente des fonctionnalités requises. Une fois que le service adéquat est sélectionné, le client examine la description du service sélectionné et récupère les informations nécessaires lui permettant de se connecter et d'invoquer les opérations du service.

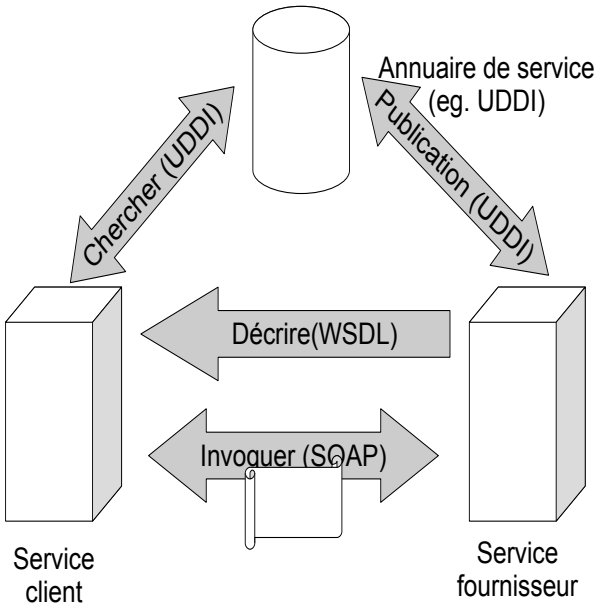


FIGURE 2.1 – Publication, recherche et invocation de services Web

Les interactions des éléments conceptuels décrits ci-dessus reposent sur les éléments de l'architecture orientée services. Comme le montre la Figure 2.2, la technologie des services Web repose sur plusieurs standards :

– *SOAP (Simple Object Access Protocol) [57]* fournit un cadre pour l'encapsulation et l'échange de messages XML qui peuvent être transmis par le biais d'une variété de protocoles réseau, tels que HTTP, SMTP, FTP.
– *WSDL [132] (Web Service Description Language)* permet la description des fonctionnalités qu'offre un service Web sous forme d'une collection d'opérations de services de leurs paramètres d'entrée/sortie (définition des messages), et des types de données impliqués dans les messages.
– *UDDI [125] (Universal Description Discovery and Integration)* permet aux services Web de s'inscrire en fournissant, entre autres, leurs URI (Uniform Resource Identifier) et leurs descriptions d'interfaces dans le standard WSDL afin qu'ils soient visibles aux autres services qui veulent les utiliser.

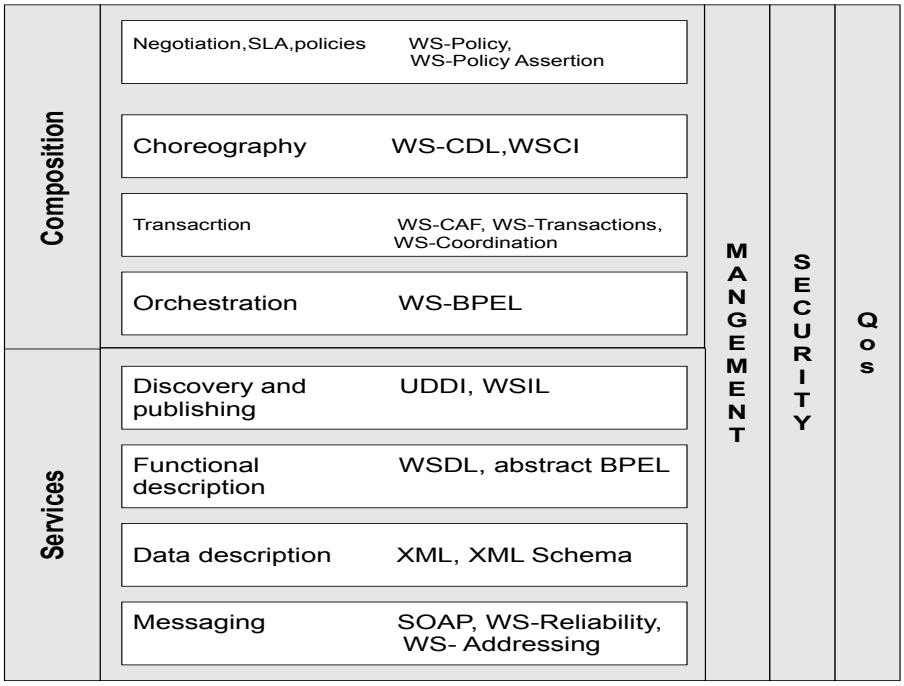


FIGURE 2.2 – La pile des standards des Web services

Dans le but de supporter le développement de services Web faiblement couplés et autonomes, plusieurs spécifications Web ont vu le jour. Leur but est de fournir des mécanismes de description des propriétés des services Web.

2.2.3 Description et spécification des services Web

La description des services Web peut être considérée de deux point de vues complémentaires :

- Point de vue structurel
- Point de vue comportemental

Description structurelle

Les services Web sont décrits via leur interface qui définit les opérations et les structures de données utilisées (i.e., la signature des opérations) pour la réalisation d'une fonctionnalité. La description de l'interface structurelle d'un service Web est spécifiée par le langage WSDL. L'objectif de cette spécification est de formaliser la description syntaxique des Services qui spécifie les fonctionnalités offertes par un service Web, en se basant sur XML. Les fonctionnalités sont représentées sous forme d'une collection d'opérations. En outre, ce langage spécifie les types de données, les informations nécessaires pour se connecter au service tels que le protocole de transport utilisé (exemple : HTTP), l'URL du service. Le but du langage WSDL est de séparer la description abstraite du service de son implantation.

En clair, il définit de manière abstraite et indépendante du langage, l'ensemble des opérations et des messages qui peuvent être transmis vers et depuis un service Web donné. Un document WSDL est divisé en plusieurs parties :

- *la balise <type>* : définit les types de données utilisés par le service en utilisant généralement xml schema
- *les balises <message>* : une abstraction de haut niveau des paramètres (messages) pour les appels possibles du service Web
- *la balise <portType>* : contient des abstractions de haut niveau des opérations possibles du service Web.
- *la balise <binding>* : définit les liens entres les opérations du service Web et le protocole de communication utilisé (généralement SOAP). C'est ici qu'on définit comment on utilise SOAP, les paramètres d'entrées et de sorties.
- *les balises <service> et <port>* : la balise service contient les points limites (port). Le port est défini par les binding du protocole réseau et une adresse réseau

Dans WSDL, la définition de la signature d'une opération s'appuie sur trois sous éléments possibles : *<input>* pour le message en entrée, *<output>* pour celui en sortie et *<fault>* pour un message d'erreur émis par le service. Les combinaisons possibles de ces sous -éléments permettent de décrire divers modèles d'opérations :

- *Réception de message* : Le service reçoit un message envoyé par un autre service ;
- *Opération de notification* : Le service envoie un message ;

– *Opération réception-émission* : Le service reçoit un message et en retour, il envoie un message de réponse ;
– *Opération emission-Retour* : Le service envoie un message et attend comme réponse, un message ;

Pour résumer, la spécification WSDL joue un rôle important dans l'interopérabilité des services Web. Moyennant cette spécification, WSDL permet aux services de définir ce qui est nécessaire à leur invocation et à la réception de leur résultat. La spécification WSDL sépare la définition abstraite du service (échange de messages) de ses mécanismes de liaison (définition des protocoles applicatifs). Cette dernière caractéristique permet au service d'interagir même si l'application a été modifiée ce qui est un point important pour assurer l'interopérabilité des services.

Cependant, étant donnée la croissance de la complexité des besoins d'interaction et d'intégration d'applications, seule la considération de l'interface des services Web lors de leurs interactions est insuffisante. Pour permettre l'intégration d'application (comme par exemple dans le cadre d'une intégration B2B), il s'avère primordial de fournir des spécifications plus riches pour permettre une meilleure efficacité au niveau de l'interopérabilité.

Description du comportement des services Web

La description structurelle de l'interface des services Web est essentielle pour faciliter l'utilisation d'un service Web. Cependant, cette spécification seule est insuffisante. Afin de savoir comment un service Web peut être utilisé, on a besoin d'abstractions de haut niveau qui caractérisent le comportement des services Web.

L'interface comportementale d'un service permet de décrire les enchaînements des opérations décrites dans l'interface structurelle. L'interface comportementale peut être décrite par des standards tels que BPEL ou WSCI [133]. Ces langages offrent les opérations classiques des langages de programmation pour décrire des enchaînements d'interactions (séquence, parallélisme, alternative, itération, etc.) [3].

Les services Web interagissent les uns avec les autres par le biais d'envoi de messages. Dans ce cadre, une interaction entre deux services, est décrite par un envoi de message par l'un des services et par la réception de ce message par l'autre service. Une séquence de telles interactions est appelée une conversation [19]. Cet aspect conversationnel est très important dans le contexte de la composition de services, qui peut être définie selon deux points de vues que nous allons expliquer dans la suite de ce chapitre.

2.2.4 Les points de vues de composition

L'architecture orientée service SOA fournit une base générique et souple afin de faciliter l'intégration d'applications inter-entreprises. Le but est de faciliter la création des services à valeur ajoutée en combinant des services distribués et indépendants les uns des autres. Le résultat de cette combinaison, encore appelée *composition* [124], est un service qui peut être à son tour publié. La construction d'une composition peut être décrite selon deux concepts complémentaires : (1) *Orchestration* et (2) *Chorégraphie* [108], qui permettent de distinguer la composition d'un point de vue local ou d'un point de vue global.

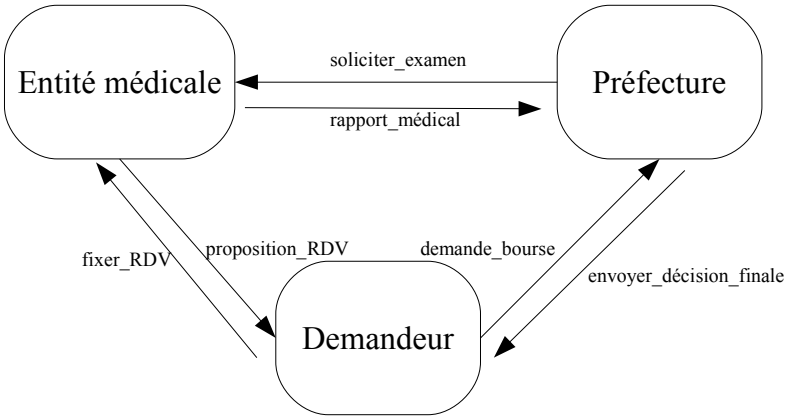
Une *orchestration* décrit un ensemble d'opérations de communication et d'action internes dans lesquelles un service donné peut ou doit s'engager ainsi que les dépendances entre ces actions. L'orchestration se base sur un procédé métier exécutable pouvant interagir avec les services Web internes ou externes. La coordination d'une orchestration d'un service est un processus externe et centralisé qui contrôle et qui coordonne l'ordre et l'exécution des interactions du service avec les partenaires impliqués dans la composition.

Quant à la composition par *chorégraphie*, elle décrit d'une part les interactions entre les services et d'autre part les relations qui existent entre ces interactions. La chorégraphie est de nature plus collaborative, chaque participant impliqué dans le processus décrit le rôle qu'il joue dans l'interaction. Contrairement à l'orchestration, dans une chorégraphie, il n'y a pas de coordinateur central. Chaque service impliqué dans la chorégraphie connaît exactement avec qui interagir et quand exécuter les opérations dont il est responsable [50, 22, 76].

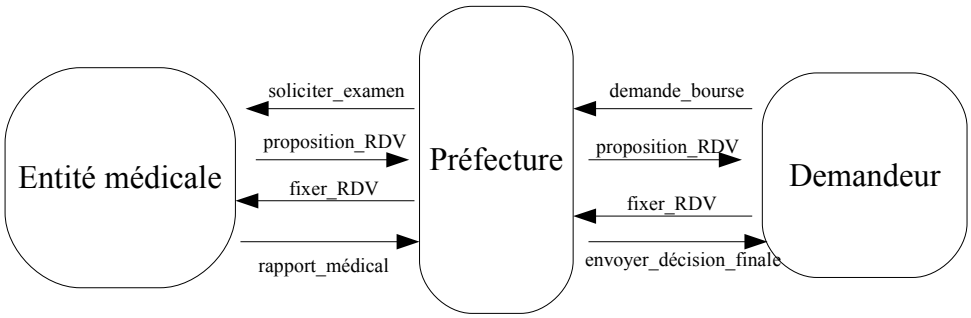
En récapitulant, il y a une différence importante entre l'orchestration et la chorégraphie de services Web. L'orchestration se base sur un procédé métier exécutable pouvant interagir avec les services Web internes ou externes. Elle offre une vision centralisée, le procédé est toujours contrôlé du point de vue d'un des partenaires métiers. La chorégraphie est de nature plus collaborative, chaque participant impliqué dans le procédé décrit le rôle qu'il joue dans l'interaction.

La figure 2.3 montre la modélisation de la composition des trois services Web entité médicale, préfécture, et le demandeur du scénario présenté dans la section 1.3. La composition par chorégraphie illustrée sur la Figure 2.3 (a) montre que les trois services interagissent ensemble. Aucun des services ne possède le contrôle de l'execution de la composition. Par contre, la composition par orchestration (voir Figure 2.3 (b)), les interactions sont décrites d'un point de vue centralisé (du service de préfécture).

Etant donnée les avantages et les facilités qu'offre le concept de service Web, ces



(a) Chorégraphie



(b) Orchestration

FIGURE 2.3 – Chorégraphie et orchestration de services Web

dernières années, on a assisté à une augmentation considérable des services sur le Web. De ce fait, réaliser les tâches sous-jacentes manuellement est devenue une tâche ardue et source d'erreurs. De ce fait est né le besoin d'automatiser le développement des applications complexes à base de services s'appuyant sur le Web et ses technologies sous jacentes [39, 51, 81]. Dans ce contexte, les efforts de recherche se sont concentrés autour de la proposition d'approches formelles rigoureuses se basant sur des formalismes tels que les réseaux de Petri(exemple [130, 137, 126, 89, 72]), les machines à états finis ou les automates temporisés (exemple [78, 79, 44, 45]). Dans ce qui suit, nous nous focalisons en particulier sur les automates temporisés.

2.3 Les automates temporisés

Ce formalisme d'automates temporisés reprend celui des automates à états finis [74] en l'étendant avec des variables de type horloge [6]. Informellement, un automate temporisé est défini par un ensemble de sommets (états), de transitions, d'un ensemble d'alphabet et d'horloges. Les sommets représentent les états des systèmes, les transitions sont étiquetées par un élément de l'alphabet. Une horloge est une variable réelle positive pouvant être remise à 0 lors du déclenchement d'une transition qui peut correspondre à l'occurrence d'un événement.

La valeur des variables de types horloges s'incrémentent avec le passage du temps. Ces variables peuvent être utilisées pour définir des contraintes sur les transitions et/ou sur les états. Les contraintes associées aux états, appelées *invariants*, permettent de spécifier que le système peut séjourner dans l'état tant que la contrainte est satisfaite. Quant aux contraintes associées aux transitions, appelées *guards*, elles permettent de spécifier que le système peut passer d'un état vers un autre en déclenchant la transition seulement si la contrainte associée est satisfaite. Les valeurs des horloges peuvent également être remises à zéro lors du déclenchement d'une transition.

2.3.1 Les contraintes temporelles

L'ensemble des horloges est noté par $X = \{x_1, \ldots, x_n\}$. Une contrainte d'horloge est une formule $x_i \bowtie a$; avec x_i une horloge, a une valeur entière positive et $\bowtie \in \{<, >, \leq, \geq, =\}$. L'ensemble des contraintes d'horloge utilisant les horloges de X sera noté Ψ_X.

L'interprétation d'une horloge x pour un ensemble X assigne une valeur réelle à chaque horloge.

Définition 1 *Un automate temporisé Q est défini par le tuple $(S, s_0, F, A, X, Inv, T)$ où :*

- *S est un ensemble d'états,*
- *s_0 est l'état initial ($s_0 \in S$),*
- *F est l'ensemble des états finaux ($F \subseteq S$),*
- *A est l'alphabet,*
- *X est l'ensemble des horloges,*
- *$T \subseteq S \times A \times \Psi_X \times 2^X \times S$ est un ensemble fini de transitions $t = (s, \alpha, \psi, Y, s') \in T$ représente une transition de s vers s', α est l'étiquette de t, ψ est la garde associée à t, Y est l'ensemble d'horloges devant être remises à zero,*
- *$Inv : S \to \Psi_X$ associe un invariant à chaque état de l'automate,*

Exemple 1 *La figure 2.4 présente un automate temporisé. Dans cet automate, les états sont s_0 et s_1. L'état s_0 est l'état initial et s_1 est l'état final de l'automate. Deux transitions peuvent être déclenchées : (1) la transition qui permet de passer de l'état s_0 vers l'état s_1, et (2) la transition permettant le passage de l'état s_1 vers s_0. Lors du déclenchement de la première transition, l'horloge t_1 est remise à zero et le système atteint l'état s_1 où il peut rester tant que la valeur de l'horloge t_1 est inférieur à 5 unités de temps. Cependant à partir du moment où t_1 est supérieure à 2, le système peut revenir à l'état s_0.*

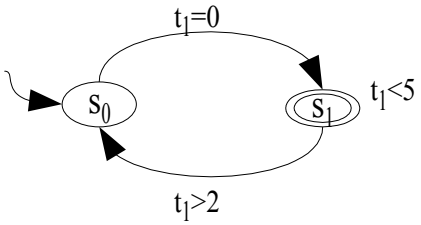


FIGURE 2.4 – Automate temporisé

2.3.2 Sémantique

La sémantique d'un automate temporisé Q est donnée par un système de transitions étiquetées (STE) où chaque état est défini par une pair (s, v) telle que s est un sommet de S et v une interprétation pour X telle que v satisfait l'invariant $Inv(s)$.

Informellement, le système part de la configuration initiale, i.e., l'état courant est s_0 et toutes les horloges sont initialitées à zéro, puis effectue l'un des deux types de transitions : (1) *les transitions d'actions* ou (2) *les transitions de temps* [6].

1. *Les transitions d'actions* : Les transitions d'actions sont étiquetées par un élément de l'alphabet. Ce type de transitions peut être déclenché d'une manière instantanée si et seulement si la valeur courante des horloges le permet. Lors du déclenchement de ce type de transitions, des horloges peuvent être remises à zero.
2. *Les transitions de temps* : Ce type de transition permet d'augmenter toutes les horloges d'une même durée en séjournant dans un état de telle sorte que l'invariant associé soit respecté.

Définition 2 *(La sémantique d'un automate temporisé)*
Soit $Q = (S, s_0, F, A, X, Inv, T)$ un automate temporisé. La sémantique de Q est définie par le STE $S_Q = (L, l_0, \rightarrow, A)$ où :

- $L = S \times \mathbb{R}^X_{\geq 0}$, *tels que* $\mathbb{R}^X_{\geq 0}$ *est l'ensemble des réels positifs,*
- $l_0 = (s_0, v_0)$ *avec* $v_0(x) = 0$ *est la valuation initiale de l'horloge* x *pour tout* $x \in X$,
- *la relation de transition* $\rightarrow$ *correspond à deux types de transitions :*
  - $s \overset{\psi,\alpha,Y}{\rightarrow} s' \in T$ *tel que* $v \models \psi$, $v' = [Y \leftarrow 0]v$ *et* $v' \models Inv(s')$.
  - *les transitions de temps : si* $\rho \in \mathbb{R}_{\geq 0}$, $(s, v) \overset{\rho}{\rightarrow} (l, v + \rho)$ *si et seulement si* $v + \rho \models Inv(s)$.

- Une exécution (ou chemin) de Q est une suite finie $\varpi = s_0 \rightarrow^{t_1} s'_0 \rightarrow^{\alpha_1} s_1 \ldots \rightarrow^{\alpha_n} s_n$. Partant d'une configuration initiale s_0 et où les délais alternent strictement avec les actions.
- Le mot temporisé $\omega = (\alpha_1, t_1)(\alpha_2, t_2) \ldots (\alpha_n, t_n)$ correspond à l'observation de la suite des événements du chemin ϖ.
- Le langage temporisé $L(A)$ accepté par Q est l'ensemble des mots temporisés associés à toutes les exécutions acceptées par l'automate.

Exemple 2 *Reprenons l'automate temporisé illustré sur la figure 2.4. Initialement l'automate est dans l'état* s_0. *Une exécution possible de l'automate est :* $(s_0, 0) \rightarrow^{0.6} (s_0, 0.6) \rightarrow^{a} (s_1, 0) \rightarrow^{2.1} (s_1, 2.1) \rightarrow^{b} (s_0, 2.1) \rightarrow^{3.2} (s_0, 3.2) \rightarrow^{a} (s_1, 0)$.

2.3.3 Les problèmes de clôture dans les automates temporisés

Vues la simplicité et l'expressivité des automates temporisés dans la modélisation des systèmes réels, plusieurs investigations ont été menées pour prouver la décidabilité (ou la non-décidabilité) de certains problèmes basiques. Dans cette section, nous survolons le problème de clôture des automates.

Clôture des automates temporisés

Les automates temporisés sont clos par les opérations d'union et d'intersection, par contre, ils ne sont pas clos par complémentarité. La cloture des automates temporisés est basée sur celle des automates non temporisés [74].

Problèmes basiques

Quelques problèmes basiques des automates temporisés sont liés à la vérification, notamment le test d'équivalence, d'inclusion, et le problème d'universalité. Le problème d'inclusion des langages temporisés consiste, étant donnés deux automates temporisés Q

et Q', à vérifier si $L(Q) \subset L(Q)'$. D'une manière analogue, le problème d'équivalence des langages temporisés consiste à vérifier si $L(Q) = L(Q')$. Quant au problème d'universalité, il revient à vérifier si un automate temporisé Q, défini sur un alphabet A, reconnaît tous les mots temporisés sur A. Ces problèmes se réfèrent au problème de complémentarité. La non-clôture par passage au complémentaire des langages reconnus par des automates temporisés fait que les trois problèmes précédemment cités sont indécidables. En effet, la vérification de l'inclusion de deux langages $L(Q) \subset L(Q')$ revient à vérifier $Q' \cap \bar{Q} = \emptyset$. D'une manière analogue, l'équivalence de deux langages temporisés $L(Q) = L(Q')$ revient à vérifier $L(Q) \subset L(Q')$ et $L(Q') \subset L(Q)$. Finalement, le problème d'universalité peut être réduit au problème d'inclusion. Plus de détails peuvent être trouvés dans [6].

L'indécidabilité de la complémentarité a incité les recherches à définir des sous-classes des automates temporisés pour tenter d'obtenir des résultats dans lesquels l'inclusion serait décidable. Dans la section qui suit, nous nous intéresserons en particulier aux automates temporisés déterministes.

2.3.4 Automate temporisé déterministe

Comme cité auparant, plusieurs problèmes sont indécidables pour les automates temporisés en général. De ce fait, plusieurs études et extensions ont été proposées ces dernières années, comme les automates temporisés déterministes sur lesquels cette thèse se base.

Un automate temporisé est dit déterministe si pour toutes les transitions (s, α, ψ, Y, s') et $(q, \alpha, \psi', Y', q')$ les deux conditions suivantes sont satisfaites [33, 104] :

- L'automate possède seulement un seul état initial,
- Pour chaque paire de transitions ayant le même état source et le même alphabet, leurs contraintes temporelles sont disjointes.

Définition 3 *(Automate temporisé déterministe)*
Un automate temporisé $Q = (S, s_0, F, A, X, T, Inv)$ est dit déterministe, si et seulement si :

- *Il n'a qu'un seul état initial : $|s_0| = 1$*
- *et pour tout $s \in S$, pour tout $\alpha \in A$, pour toute paire de transitions $(s, \alpha, \psi_1, Y_1, s_1)$ et $(s, \alpha, \psi_2, Y_2, s_2)$ les contraintes ψ_1 et ψ_2 sont exclusives, i.e., $\psi_1 \wedge \psi_2$ est insatisfiable.*

Il est a noter que le problème de déterminisation d'automate non-déterministe non temporisé est décidable [74]. Cependant, dans le cas des automates temporisés, ce problème est indécidable [129].

2.3.5 Le model checker UPPAAL

Afin de supporter la modélisation à base d'automates temporisés, plusieurs *model checker* ont été développés, tels que SMV [96], KRONOS [136], HYTECH [70], et UPPAAL [85]. Comme pour les automates temporisés, d'autres formalismes sont dotés de model checker, comme WofBpel [106] et Lola [119, 90] pour les réseaux de Petri. Dans cette thèse, nous nous intéressons en particulier au *model checker UPPAAL*, que nous exposons dans la section suivante.

Etant donné un modèle, le model-checking consiste à vérifier si ce modèle satisfait certaines propriétés. Un model-checker est une boite noire qui a comme entrée deux paramètres : un modèle et un ensemble de propriétés, et a comme sortie l'affirmation de la satisfaction ou la non-satisfaction de ces propriétés.

Dans la littérature, plusieurs types de propriétés ont été définis :

- *Propriété d'atteignabilité (Reachability)* : Ce type de propriété énonce si un état donné peut être atteint ou non.
- *Propriété de sûreté (Safety)* : exprime que sous certaines conditions, un évènement ne peut jamais se produire.
- *Propriété de vivacité (Liveness)* : sous certaines conditions, un évènement finira par avoir lieu.

UPPAAL est un outil pour la modélisation, la validation et la vérification des systèmes temps réel. Il est approprié pour les systèmes qui peuvent être modélisés par des automates temporisés. Le modèle sur lequel UPPAAL repose est un ensemble d'automates temporisés. Ces automates sont caractérisés par un ensemble d'horloges, des canaux pour les systèmes (automates) de synchronisation, des variables et des éléments supplémentaires [85]. Chaque automate a un état initial. La synchronisation entre les différents processus peut avoir lieu en utilisant des canaux. Un canal peut être un canal de sortie (noté par *nom_canal*!), ou peut être un canal d'entrée (noté par *nom_canal*?).

Variables et horloges peuvent être associées aux automates. Les contraintes sur ces horloges peuvent être associées à des transitions et aux états. Un état peut être défini comme étant *urgent* pour préciser que le système doit quitter immédiatement l'état, c'est-à-dire, immédiatement et sans attendre dans cet état.

Ce *model checker* dispose d'un simulateur et d'une interface graphique très performante facilitant ainsi l'assistance à la vérification pour l'utilisateur. Il permet d'assurer la vérification des propriétés d'atteignabilité, de vivacité, de sûreté, et de non-blocage.

Syntaxe

La base du modèle d'UPPAAL est la notion d'automates temporisés [6], comme extension des automates à états-finis classiques avec les variables d'horloges. Pour avoir une modélisation plus expressive, les automates temporisés sont étendus avec des types plus généraux de variables telles que les variables booléennes et entières. Les éléments du modèle de UPPAAL sont :

- *Les gardes* : Les gardes expriment les conditions sur les variables d'horloges et des variables entières qui doivent être satisfaites. Formellement, les gardes sont la conjonction des contraintes temporisées et des contraintes de variables entières.
- *Opération de remise à zéro* : La remise à zéro d'une horloge ou d'une variable de données sur une transition est une initialisation de la valeur de l'horloge.
- *Canaux, synchronisation et urgence* : Un modèle UPPAAL consiste en un réseau d'automates temporisés. Les automates peuvent communiquer via des canaux de communication, permettant l'envoi et la réception d'un message. La communication sur un canal apparaît comme étant la synchronisation entre deux processus. a ! (Envoi) et a ? (Réception) dénotent qu'un processus se synchronise avec un autre. Pour empêcher un automate de s'attarder dans un état, le canal doit être déclaré comme étant urgent.
- *Invariant* : Les états peuvent être étiquetés par des invariants, qui expriment les contraintes sur les valeurs d'horloges, associées à l'état, afin qu'il puisse y séjourner.

Le langage de spécification des propriétés de UPPAAL

Le langague sur lequel UPPAAL repose pour la spécification des propriétés est un sous ensemble de la logique CTL (*Computation Tree Logic*) [71]. Les formules CTL qui peuvent être analysées dans UPPAAL sont :

- **EF** ψ : il est possible d'atteindre un état où la propriété ψ est vérifiée. En d'autres termes, il existe une exécution (opérateur E) conduisant à un état où la propriété ψ est vérifiée (opérateur F).
- **EG** ψ : il existe une exécution (opérateur E) où ψ est toujours vérifiée (opérateur G)
- **AF** ψ : pour toute exécution (opérateur A), il existe un état où ψ est vérifiée (opérateur F)
- **AG** ψ : pour toute exécution (opérateur A), ψ est toujours vérifiée (opérateur G)

Dans UPPAAL, ces formules CTL sont implantées comme suit :

– **A[]** ψ : est l'implantation dans UPPAAL de la formule **AG** ψ.
– **A<>** ψ : est l'implantation dans UPPAAL de la formule **AF** ψ.
– **E[]** ψ : est l'implantation dans UPPAAL de la formule **EG** ψ.
– **E<>** ψ : est l'implantation dans UPPAAL de la formule **EF** ψ.
– $\psi \rightsquigarrow \phi$: est l'implantation dans UPPAAL de la formule **AG** $(\psi \Rightarrow$ **AF** $\phi)$.

Figure 2.5 montre une capture d'écran du simulateur de UPPAAL qui dispose de :
– Le visualiseur des transitions déclenchées
– Le simulateur de traces
– Le visualiseur de l'évolution des valeurs des variables temporisées et non temporisées
– Le visualiseur des processus.

UPPAAL dispose également d'un vérificateur (voir la figure 2.6) qui permet d'éditer et vérifier la validité des requêtes.

2.4 Travaux existants

Dans cette section, nous présentons les travaux relatifs à la problématique d'analyse et de composition de services Web. Nous commençons tout d'abord par aborder les travaux d'analyse et de vérification des services Web. Ensuite, nous présentons les approches de composition qui ont été proposées ces dernières années.

2.4.1 Analyse et vérification des services Web

Vérifier et analyser automatiquement les interactions des services Web est un problème considéré comme très important [32, 15, 113, 79, 60].

Dans [47], Dumas et al. et dans [5] Altenhofen et al., s'intéressent au problème d'incompatibilité structurelle des messages mais sans prendre en considération l'aspect comportemental. En outre, les auteurs ne considèrent pas les aspects temporels.

Dans [77], Kallel et al., utilisent un langage formel appelé *XTUS-Automata* qui est une combinaison des automates temporisés [6] et XTUS (extended time unit system) [34] pour la spécification des propriétés temporelles. Le but est de vérifier si une composition, qui est déjà spécifiée manuellement par un concepteur, vérifie des propriétés telles que la durée qui sépare deux activités, tout en assurant le monitoring des contraintes temporelles lors de l'exécution en temps réel. Les auteurs de ce travail ne s'intéressent pas à la problématique de vérification de la compatibilité des services dans le cadre d'une chorégraphie, ni à la construction de compositions, mais supposent qu'une composition est déjà créée.

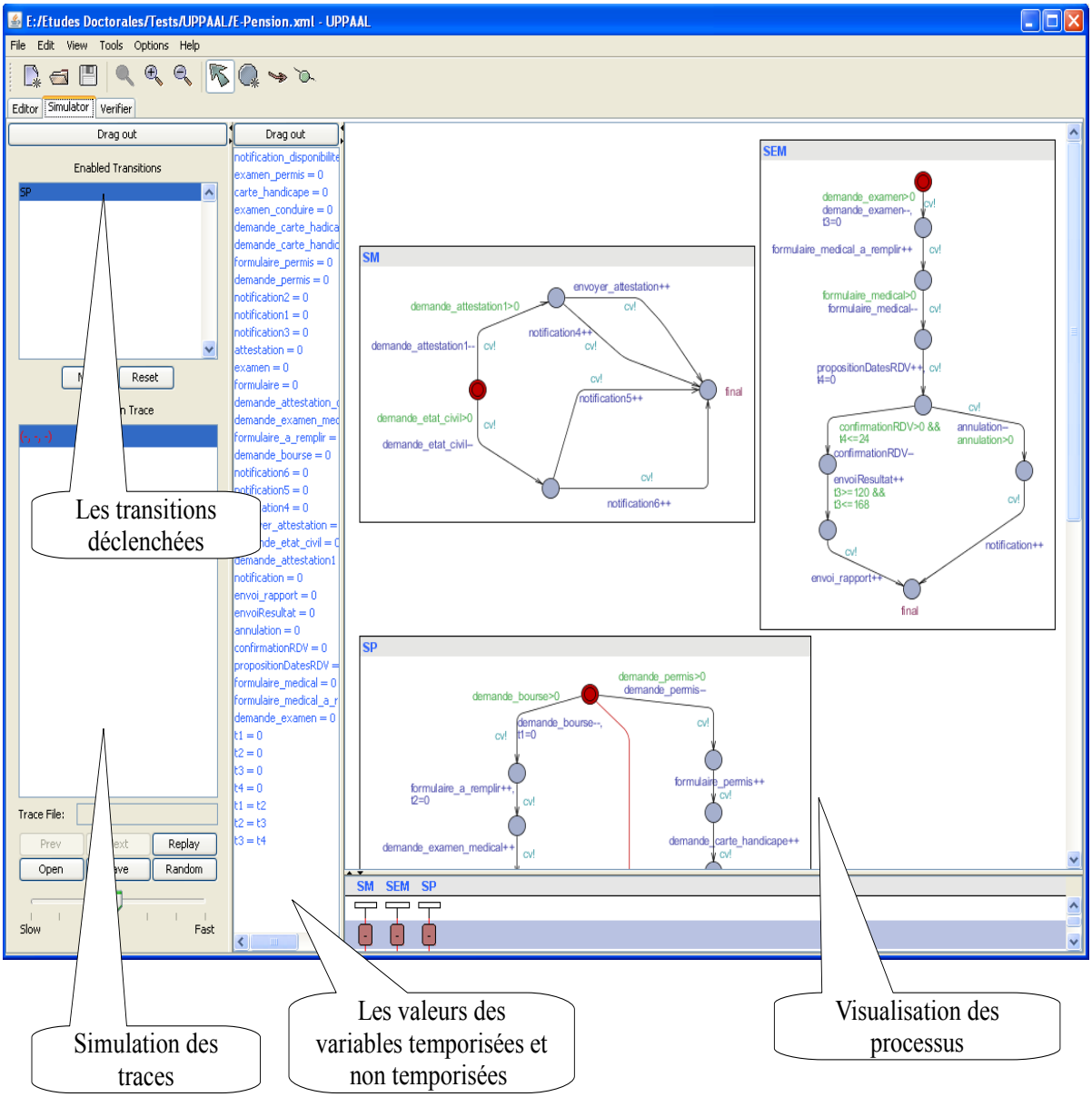


FIGURE 2.5 – Capture d'écran de l'interface graphique du simulateur d'UPPAAL

Dans [93], Mazzara se base sur le formalisme webπ [94, 84] pour spécifier formellement la composition de services Web. Webπ est une extension de π-calculus par des contraintes temporelles. L'auteur a montré comment webπ peut être utilisé pour spécifier des processus comme des processus webπ qui permettent la simulation. Le travail présenté dans [93] s'intéresse seulement à la manière de spécifier les processus avec l'algèbre de processus webπ, mais aucun raisonnement n'a été considéré.

Dans [32] Bordeaux et al. et dans [13] Benatallah et al., reposent sur le formalisme

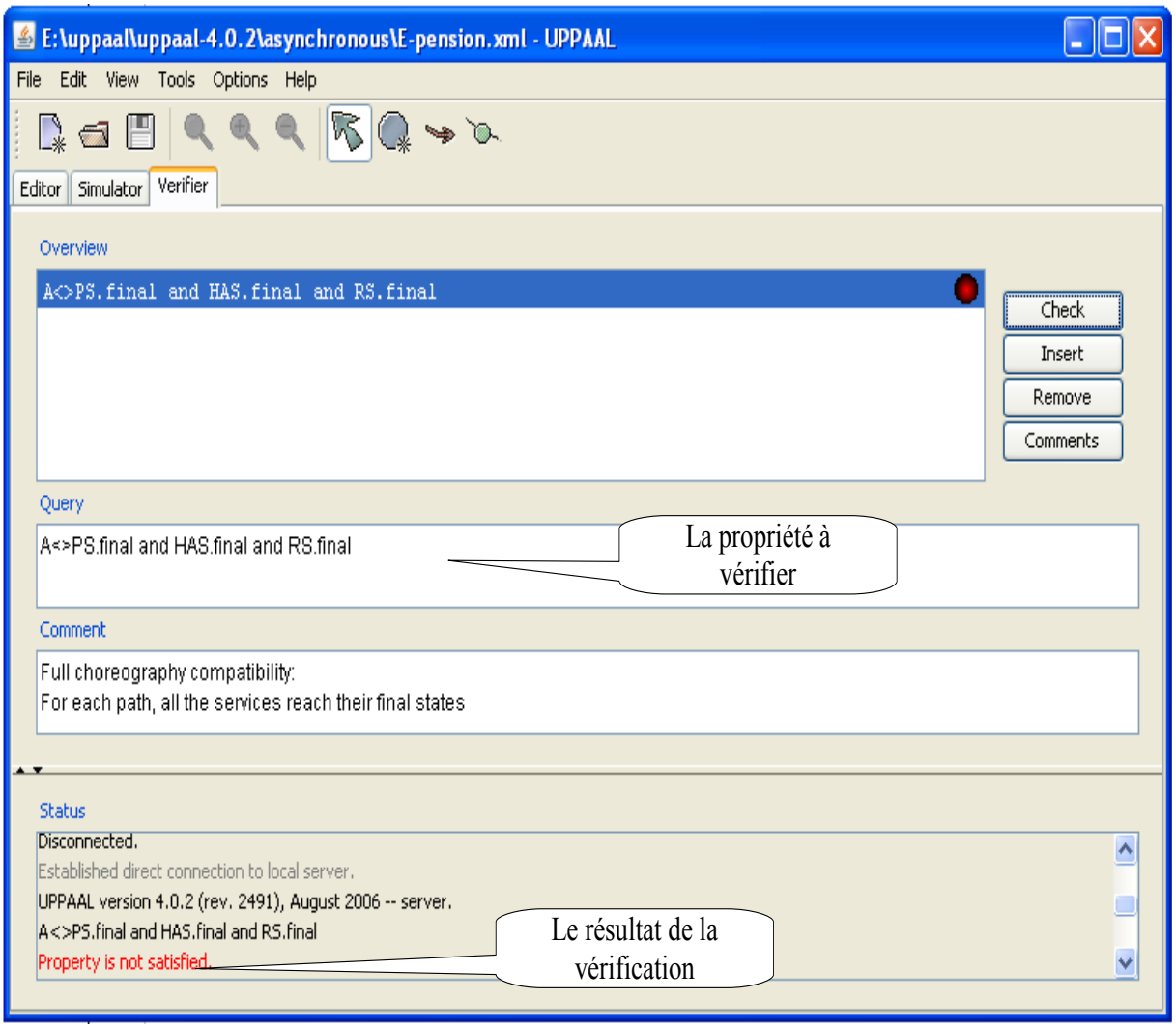


FIGURE 2.6 – Capture d'écran de l'interface graphique du vérificateur d'UPPAAL

des machines à états finis pour modéliser le comportement des services Web en vue d' une analyse de compatibilité [32] et de substitution de services Web [13]. Dans ce cadre, le comportement des services Web est spécifié par les séquences de messages qui peuvent être échangées d'une manière synchrone. Parallèlement aux séquences de messages, afin de réussir une conversation, d'autres paramètres peuvent avoir un impact capital, comme les contraintes temporelles. Ce genre de contraintes n'est pas considéré dans ces travaux. En outre, les auteurs ne considèrent que des services Web synchrones. Cette hypothèse est très restrictive puisque deux services peuvent réussir une conversation bien qu'ils ne supportent pas la même structure de séquencement de messages.

Le cadre de compatibilité présenté dans [15] prend en considération une forme de

contraintes temporelles qui permettent de spécifier des contraintes d'activation. Une fois un état est atteint, ces contraintes permettent de déclencher automatiquement une transition vide après un délai et ce si le service ne change pas d'état. Ces contraintes peuvent être définies sur une seule transition, i.e., une fois qu'un état est atteint, si aucune transition n'est pas déclenchée après un délai, alors une autre transition vide est déclenchée. Le modèle temporisé de [15] est très restrictif et ne permet pas, par exemple, de spécifier des intervalles de temps entre l'échange de deux messages.

Dans [113, 115] Ponge et al. et dans [88] Mancioppi et al., considèrent un modèle plus riche de contraintes temporelles. Le problème abordé dans [113, 115] est celui de la compatibilité de services Web synchrones. Dans [88] les auteurs montrent comment il est possible d'exécuter d'une manière consistante un ensemble de protocoles et ce en se basant sur le modèle proposé dans [115]. Dès qu'il s'agit de services Web asynchrones, les cadres [113, 115, 88] sont restrictifs. En outre, certains conflits temporels ne sont pas pris en compte, tels que les conflits du type $x - y \leq v$ $(resp. x - y \geq v)$ telles que x, y sont des horloges et v une valeur entière (voir chapitre 5). En plus, ces travaux ne prennent pas en considération les données ainsi que les contraintes de données.

Díaz et al., montrent dans [44, 45] comment des services Web décrits par WS-CDL peuvent être traduits vers le formalisme des automates temporisés qui peuvent être ensuite vérifiés par le *model checker* UPPAAL. Etant donné que les auteurs utilisent la synchronisation binaire de UPPAAL, seuls des services synchrones peuvent être vérifiés. En outre, les auteurs ne prennent pas en considération les données et les contraintes de données.

D'autre travaux visent la synchronisation de services asynchrones [54, 55]. Le but est de vérifier si les communications asynchrones d'un ensemble de services Web peuvent être substituées par des communications synchrones sans perte d'information ni de cohérence. Ces travaux s'intéressent à des services Web non-temporisés.

Dans [48], Eder et al., traitent le problème de la conformité qui consiste à vérifier si une orchestration satisfait une chorégraphie temporisée. Dans ce cadre, les auteurs considèrent le coût temporel des opérations. Ce travail repose sur l'hypothèse que la chorégraphie est correcte et ne contient aucun conflit. Le but de notre travail est de détecter les conflits qui peuvent apparaître lors de l'interaction des services Web dans une chorégraphie.

D'autres travaux de vérification de composition de services Web ont été proposés [37, 78, 79]. Dans [37] Bultan et al., se focalisent sur la vérification des propriétés d'une composition donnée de services Web qui supportent des communications asynchrones. Ces

propriétés sont spécifiées par des formules LTL [112]. Le modèle proposé dans [37] est basé sur le formalisme des machines à états finis. Ce travail ne prend pas en considération les aspects temporisés. Quant au travail présenté dans [78, 79], Kazhamiakin et al., proposent un modèle de spécification de propriétés temporelles dans une composition donnée. Ce travail ne prend pas en considération les données et les contraintes de données. En outre, le problème auquel [37, 78, 79] s'intéressent consiste à vérifier une composition déjà construite. Dans notre travail, nous nous intéressons à la vérification de la compatibilité d'un ensemble de services Web qui peuvent être impliqués dans une chorégraphie, donc avant leur composition.

Le tableau suivant récapitule les aspects considérés et non-considérés des approches étudiées.

Approche	Type de vérification	Critères considérés	Critères non-considérés
[47, 5]	Compatibilité	Compatibilité structurelle	– Aspect comportemental – Aspect temporel
[77]	Vérification d'une composition	Vérification de certaines propriètés	– Détection d'incompatibilité – Création de composition
[93]	Modélisation	Spécification des processus	Aucun raisonnement n'est proposé
[32, 13]	Compatibilité et substituvité	– Comportement de services – Services synchrones	– Services asynchrones – Aspect temporel – Données et contraintes de données
[15, 113, 115, 88]	Compatibilité et substituvité	– Services sychrones – Aspect temporel	– Services asynchrones – Donnée et contraintes de données

[48]	Conformité	– Aspect comportemental – Aspect temporel	Détection de conflits dans une chorégraphie
[44, 45]	Analyse de la chorégraphie	– Aspect comportemental – Aspect temporel – Services synchrones	– Services asynchrones – Données et contraintes de données
[54, 55]	Synchronisation de services	– Aspect comportemental – Services asynchrones	Aspect temporel
[37]	Vérification d'une composition	– Aspect comportemental – Aspect conversationnel	– Aspect temporel – Données et contraintes de données
[78, 79]	Vérification d'une composition	– Aspect comportemental – Aspect temporel	– Construction d'une composition – Données et contraintes de données

2.4.2 Composition de services Web

La composition de services est un secteur de recherche très actif [9, 20, 19]. Précédemment, nous avons présenté les approches d'analyse et de vérification des services Web. Dans cette section, nous examinons les approches qui ont été proposées dans le cadre du problème de synthèse de composition.

Dans [42], Deutsch et al., considèrent un service Web comme une source de données qui est caractérisée par une base de données et des pages Web. Une fois que le client a

activé une page Web, un ensemble d'entrées lui sont proposées. Le client choisi l'une de ces entrées et comme réponse, le service met à jour les bases de données et permet de passer d'une page Web à une autre. Les auteurs de [42] s'appuient sur le modèle relationnel pour modéliser l'ensembles des éléments considérés. Le problème traité dans ce travail consiste à vérifier des propriétés lors de l'interaction entre les services Web et l'utilisateur. Deux types de propriétés sont considérés. Le premier type permet de spécifier quelles sont les propriétés qui doivent être satisfaites durant toutes les exécutions (par exemple, dans le cadre du e-commerce, il est souhaitable de vérifier que le produit commandé ne soit pas livré avant que le paiement ne soit effectué). Afin de spécifier ce type de propriétés, les auteurs utilisent la logique temporelle linéaire. Quant au deuxième type, il porte sur un ensemble d'exécutions (par exemple, vérifier que pour chaque exécution, l'utilisateur peut retourner à la page d'accueil). Les auteurs formalisent ce genre de propriétés par le biais de la logique temporelle de branchement CTL.

Dans [122, 127, 128], Thakkar et al., proposent d'utiliser les techniques proposées dans le cadre de l'intégration de données afin d'apporter une solution au problème de composition, ce en se basant sur le modèle Local-As-View [87]. Chaque service est décrit principalement par l'ensemble des données d'entrée, des données de sortie, et des contraintes sur ces données qui permettent par exemple de spécifier le type de données attendu (par exemple, la donnée de retour doit concerner un restaurant d'une certaine spécialité). Dans cette approche, un expert définit un ensemble de prédicats et décrit l'ensemble des services Web en utilisant les prédicats définis. Afin de construire un service composite, les auteurs proposent une approche basée sur un médiateur. Le rôle du médiateur est de reformuler la requête du client (i.e., l'ensemble des données fournies et l'ensemble des données requis) de telle sorte qu'elle soit compatible avec les sources de données (les services Web). Une fois que la requête est reformulée, les sources de données sont interrogées afin de fournir les données requises.

Dans [98], Medjahed et al., présentent un cadre de composition de services Web atomiques. Les services Web sont décrits en terme de certaines propriétés non fonctionnelles telles que le but du service. En se basant sur ce modèle, les auteurs proposent une démarche afin de vérifier et comparer les propriétés des services Web et ce dans le but de savoir si ils peuvent être composés. Le besoin du client est spécifié en terme de séquences des opérations désirées que la composition de services Web doit réaliser. En utilisant la description du besoin du client et l'ensemble des opérations des services Web atomiques, les auteurs proposent une méthode de composition semi-automatique. Le principe est de

trouver pour chaque opération du client, un service (atomique) qui permette de réaliser cette opération, i.e., fournit une opération correspondant exactement à l'opération souhaitée.

Dans [120, 95] McIlraith et al., proposent une approche de composition basée sur le formalisme de Situation Claculus. Dans ce travail, le client spécifie sa requête sous forme d'une procédure Golog [56, 116]. Ensuite, l'idée de base consiste à instancier des services Web décrit en OWL-S pour essayer de satisfaire la requête du client. La spécification des services Web est ensuite traduite vers des procédures Golog/ConGolog [56]. Ceci, permet de spécifier un service Web comme étant une action atomique qui a des paramètres d'entrées et des paramètres de sorties.

Dans [42, 122, 127, 128, 98, 120, 95] les services Web considérés sont atomiques. L'aspect comportemental n'est pas considéré de même que l'aspect temporel.

Dans [37], Bultan et al., utilisent le formalisme des machines de Mealy (Mealy machine) afin de modéliser les services Web. Les services Web peuvent échanger des messages d'une manière asynchrone. Dans ce cadre, les services Web peuvent envoyer un message, recevoir un message, consommer un message de la queue. L'approche proposée dans [37] ne concerne pas le problème de la création d'une composition qui puisse satisfaire un client, mais elle vise l'analyse du comportement local et global des services Web dans une composition. En outre, les auteurs ne considèrent pas les propriétés temporelles avec les données et les contraintes sur les données. Salaün et al. montrent dans [117] ainsi que Bordeaux et al. dans [32] comment modéliser une orchestration et une chorégraphie en utilisant CSS [99, 67]. En utilisant ce langage, les auteurs associent une sémantique opérationnelle au comportement des services Web en vue de la vérification automatique des propriétés souhaitées telle que l'absence des blocages.

Pistore et al. [111] et Bertoli et al. [30] proposent de traduire la spécification BPEL d'un ensemble de services Web vers le formalisme des machines à états finis. Ensuite, le problème de composition consiste à calculer le produit cartésien de l'ensemble des automates. Le but est de trouver des chemins dans le produit cartésien qui satisfassent les propriétés d'accessibilité (reachability). Ce cadre ne prend pas en considération les propriétés temporelles. En outre, les auteurs ne considèrent pas l'aspect conversationnel ainsi que les données impliquées dans la composition.

Dans [25, 23] Berardi et al., présentent une approche automatique de composition de services Web. Le comportement des services Web est modélisé par des machines à états finis déterministes. Le comportement des services Web décrit les séquences des opéra-

tions qu'un service Web peut effectuer. Le problème de composition que traite les auteurs dans [25] est particulièrement lié au problème de délégation. Les auteurs supposent qu'une description abstraite de la composition attendue est donnée. Ensuite, l'approche de composition essaie de déléguer les opérations de la description abstraite à des services Web. Cependant, dans [25], l'aspect conversationnel des services Web n'est pas considéré et de ce fait la délégation peut échouer si deux services ne peuvent pas communiquer.

Dans [25, 24], les approches reposent principalement sur le besoin du client (appelé *service but*). Ce dernier spécifie les séquences des opérations que la composition doit réaliser. Ceci a pour but de trouver les services Web qui peuvent réaliser chaque opération de la description fournie. Néanmoins, dans des applications réelles, un simple client n'a pas toutes les informations des services Web et donc il est incapable de spécifier son besoin en fonction des opérations des services Web. En plus, les aspects temporels ne sont pas considérés.

Dans [7], Anca et al., présentent une borne inférieur EXPTIME pour le problème de composition qui est basé sur le modèle présenté dans [25]. Dans [7], la composition de services Web consiste en la construction du produit cartésien des différents automates modélisant le comportement des services Web. Le modèle présenté dans [7] ne tient pas compte des aspects conversationnels ainsi que du flux de données. L'aspect conversationnel est très important dans la composition de services Web étant donné qu'afin de coopérer, les services Web doivent être capable d'échanger des messages. Puisque l'approche proposée dans [7] ne considère ni les messages ni les données, la composition construite peut ne pas être correcte. Egalement, l'aspect temporel n'est pas considéré. Dans la même direction, les auteurs de [10] présentent des résultats de décidabilité du problème de composition.

Dans [2, 86], un langage de requêtes a été proposé. Le but de ce langage est de permettre aux utilisateurs de spécifier leurs besoins. Ce langage est basé sur le langage EaGLe [83]. Le langage EaGLe est une extension de la logique CTL qui permet de spécifier des formules de la forme *Essayer de satisfaire la contrainte c. Dans le cas d'un échec, essayer de satisfaire la contrainte d* [83]. Dans [109], Pistore et al., proposent une approche de composition qui prend comme entrée un ensemble de services Web modélisés comme des machines à états finis non-déterministes ainsi que la description EaGLe du besoin du client et produit en sortie un plan qui spécifie comment les services Web doivent être coordonnés afin de satisfaire la requête du client. Dans les travaux présentés dans [2, 86, 109], les aspects temporel et conversationnel ne sont pas pris en compte.

Le tableau suivant récapitule les caractéristiques et les limites des approches de com-

position étudiées.

Approche	Avantages	Limites
[42, 122, 127, 128, 98, 120, 95]	Services atomiques	– Aspect comportemental – Aspect temporel
[37]	– Aspect comportemental – Communications asynchrones – Analyse du comportement global	Aspect temporel
[117, 32]	– Aspect comportemental – Vérification des propriétés de non-blocage	– Aspect temporel – Aspect conversationnel – Données
[111, 30]	– Aspect comportemental – Vérification des propriétés d'accessibilité	– Aspect temporel – Aspect conversationnel – Données
[25, 23, 7, 10]	– Aspect comportemental – Délégation	– Aspect conversationnel – Aspect temporel
[24]	– Aspect comportemental – Aspect conversationnel – Données	– Aspect temporel
[2, 86]	– Aspect comportemental – Model checking d'une composition	– Aspect temporel – Aspect conversationnel – Données

2.5 Conclusion

Dans ce chapitre, nous avons introduit le paradigme de clacul orienté services. Nous avons présenté la technologie des services Web et les concepts relatifs. Ensuite, nous avons exposé le formalisme des automates temporisés sur lesquels se base cette thèse. Dans la dernière partie de ce chapitre, nous avons examiné les approches qui traitent du problème d'analyse et de vérification ainsi que la composition de services Web. Les approches d'analyse et de vérification de services Web que nous avons étudiées traitent le problème de l'incompatibilité structurelle des messages [47, 5], de la vérification d'une composition déjà construite[77], de la compatibilité de services Web synchrones [32, 13] et de services Web synchrones temporisés [113, 115, 88, 44, 45], de la conformité [48], et de la synchronisation de services asynchrones [54, 55]. Les travaux portant sur l'analyse de compatibilité ne traitent que des services Web synchrones, tandis que la nature des services Web peut être asynchrone. En outre, les interactions des services Web impliquent des données et peuvent être contraints de respecter des conditions sur la valeur des données (i.e., contraintes de données). Quant aux travaux qui considèrent l'aspect conversationnel asynchrone, ils ignorent l'aspect temporisé et son impact dans l'interaction des services Web.

En ce qui concerne les approches de composition que nous avons étudiées, certaines d'entre elles ne considèrent que des services atomiques, ignorant ainsi l'aspect comportemental de ces services. Les approches qui prennent en considération l'aspect comportemental, ignorent l'aspect conversationnel des services et/ou les données et les contraintes de données. D'autre part, ces approches de composition traitent des services non temporisés.

Fournir un modèle qui tienne compte des abstractions citées précédamment permettrait de fournir des primitives plus flexibles. Dans le chapitre suivant, nous présentons le modèle formel que nous considérons. Il s'agit d'une modélisation basée sur le formalisme des machines à états finis temporisés.

Chapitre 3

Modélisation des protocoles de conversations temporisées

Sommaire

3.1 Introduction 62

3.2 Présentation informelle des protocoles de conversations temporisées 62

3.2.1 Protocoles de conversations temporisées 63

3.2.2 Les propriétés temporelles dans les protocoles de conversations 65

3.3 Le modèle formel 67

3.3.1 Modélisation du comportement temporisé des services Web . . 67

3.3.2 Valuation d'horloges 70

3.3.3 Conversations temporisées 71

3.3.4 Sémantique des protocoles de conversations temporisées 71

3.4 Conclusion 72

3.1 Introduction

Comme évoqué précédemment, la description de l'interface des services Web permet de décrire seulement les signatures des opérations qui peuvent être invoquées. Dans le cadre de notre travail, une telle description est insuffisante. En plus de la description de l'interface, on a besoin de connaître comment les services Web se comportent pour pouvoir les analyser et les composer. Pour ceci, le travail proposé dans cette thèse s'appuie sur les protocoles de conversations qui peuvent être spécifiés par les langages tels que BPEL, WS-CDL, OWL-S.

Dans ce chapitre, nous présentons le modèle formel que nous utilisons et qui repose sur le formalisme des automates temporisés. Le but est de pouvoir spécifier et modéliser formellement le comportement temporisé des services Web. En particulier, dans le contexte de cette thèse, le comportement des services Web est décrit par les séquences de messages, les types des données, les contraintes de données et les propriétés temporelles qui spécifient les délais dans lesquels les messages doivent être échangés. Ces messages peuvent être échangés d'une manière asynchrone. De ce fait les services Web sont dotés d'une file queue pour stocker les messages entrants.

Nous commençons ce chapitre par introduire intuitivement le modèle formel que nous proposons pour modéliser les protocoles de conversations temporisés. Ce modèle est basé sur les automates temporisés déterministes à états finis. Notre choix est dû au fait que les automates temporisés, qui sont basés sur les machines à états finis, sont simples à comprendre et en même temps, ils sont suffisamment expressifs pour modéliser les aspects et les propriétés que l'on considère dans notre travail. En outre, plusieurs problèmes ont été prouvés décidables pour les automates temporisés déterministes tels que la complémentarité, l'équivalence, et l'inclusion. Nous présentons ensuite la sémantique que nous définissons, qui nous a permis de mettre en œuvre des techniques et des méthodes formelles.

3.2 Présentation informelle des protocoles de conversations temporisées

Avant de présenter les primitives formelles de ce qu'on appelle *protocole de conversations temporisées*, nous commençons dans cette section par présenter et discuter informellement ces protocoles.

3.2.1 Protocoles de conversations temporisées

Un protocole de conversations temporisées indique les séquences d'échanges de messages, appelés conversations, qui sont permises par un service Web, exprimées en termes de contraintes sur l'ordre dans lequel les opérations de service devraient être invoquées. Ces protocoles peuvent être exprimés à l'aide de standards comme BPEL ou WSCL ou tout autre langage de protocoles. La spécification des protocoles de conversations est importante car il est rare que les opérations réalisées par les services puissent être invoquées dans n'importe quel ordre et indépendamment les unes des autres [41].

Exemple 3 *Le comportement du service PS est spécifié via le diagramme d'activité illustré sur la figure 3.1. Ce service permet de gérer les demandes de bourses et les permis de conduire pour handicapés en exécutant les différentes opérations suivant un ordre bien défini. Par exemple, quand ce service reçoit une demande de bourse d'un demandeur qui est âgé de moins de 16 ans, alors il envoie une notification d'irrecevabilité de la demande, sinon il envoi un formulaire à remplir. Afin de vérifier si réellement le demandeur présente un handicap, le PS sollicite une entité médicale pour examiner le handicap du demandeur. En outre, la préfecture sollicite la mairie pour demander une attestation de domicile. Une fois que le formulaire, l'attestation de domicile et le rapport médical sont reçus, la préfecture prend une décision. Comme on peut le constater, l'exécution des différentes activités doit se faire selon un ordre défini.*

Le diagramme d'activité montré sur la figure 3.2, illustre le comportement du service SM. Ce service permet l'établissement de l'état civil ou l'attestation de domicile. Après la réception de la demande (de l'état civil ou l'attestation de domicile), le service envoie soit une notification de non délivrance de la demande soit l'attestation requise.

Figure 3.3 décrit le comportement du service SEM qui permet d'examiner et de spécifier le handicap dont souffre une personne. Une fois que ce service reçoit la demande d'examen, il envoie à la personne à examiner un formulaire médical. Une fois que le formulaire est reçu et rempli, SEM propose une liste de dates de Rendez-vous. La personne peut alors soit confirmer et choisir une date de Rendez-vous, soit annuler l'examen. Si le Rendez-vous est confirmé et après que la consultation ait lieu, SEM envoie le résultat des analyses au demandeur. Ensuite, il envoi le rapport médical à l'entité qui l'a demandé.

Le comportement des services Web peut être contraint de respecter certaines conditions définies sur les données, appellée *contraintes de données*. Par exemple, quand le service SP reçoit une demande de bourse et si le demandeur a moins de 16 ans, i.e., la valeur de

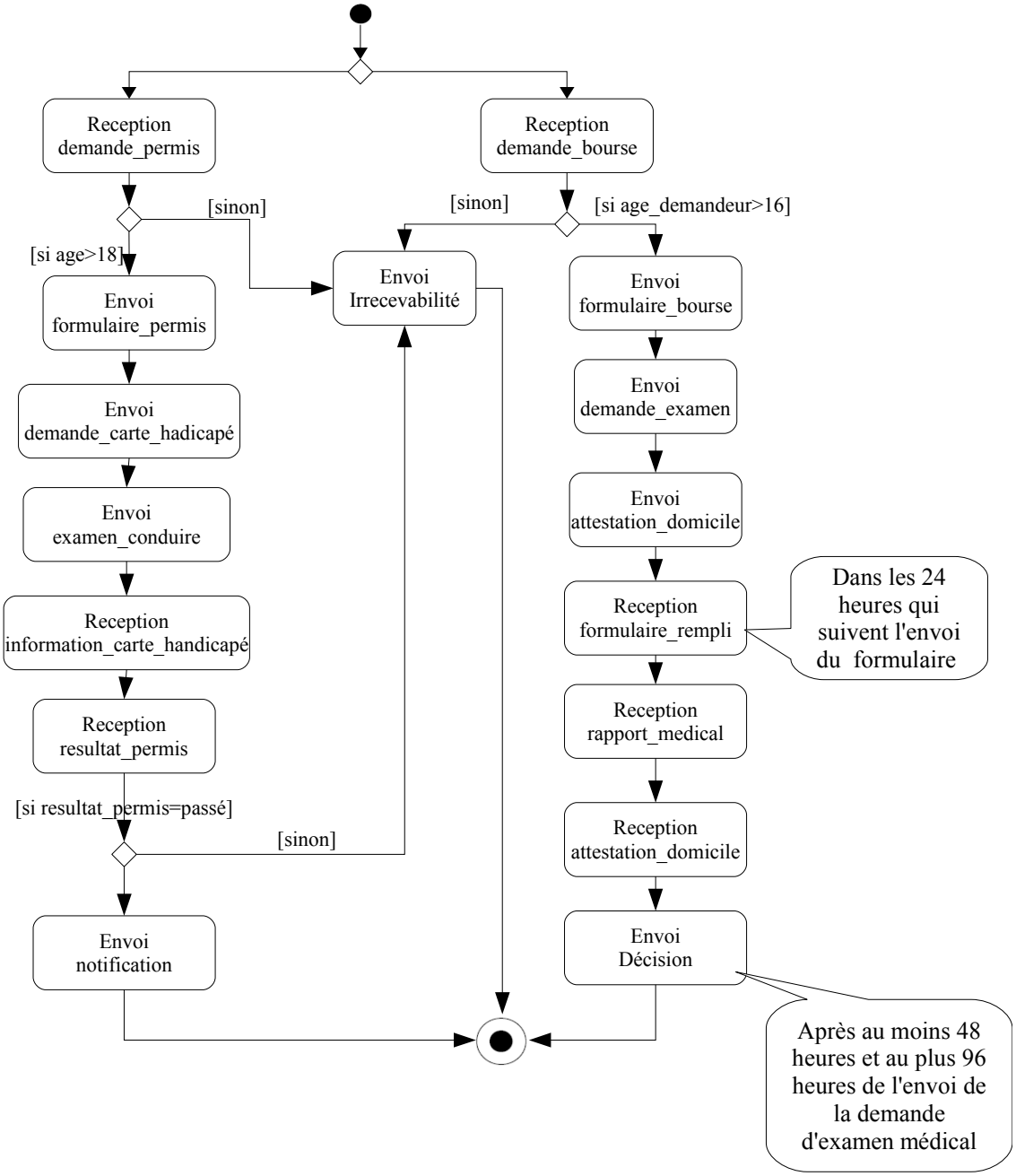


FIGURE 3.1 – Comportement du service de la préfecture (SP)

la donnée *age* est inférieure à 16 , alors ce service envoie un message d'irrecevabilité de la demande, sinon, la préfecture procède à l'étude de la demande.

Le deuxième type de contraintes concerne les contraintes temporelles que l'on présente dans la section suivante.

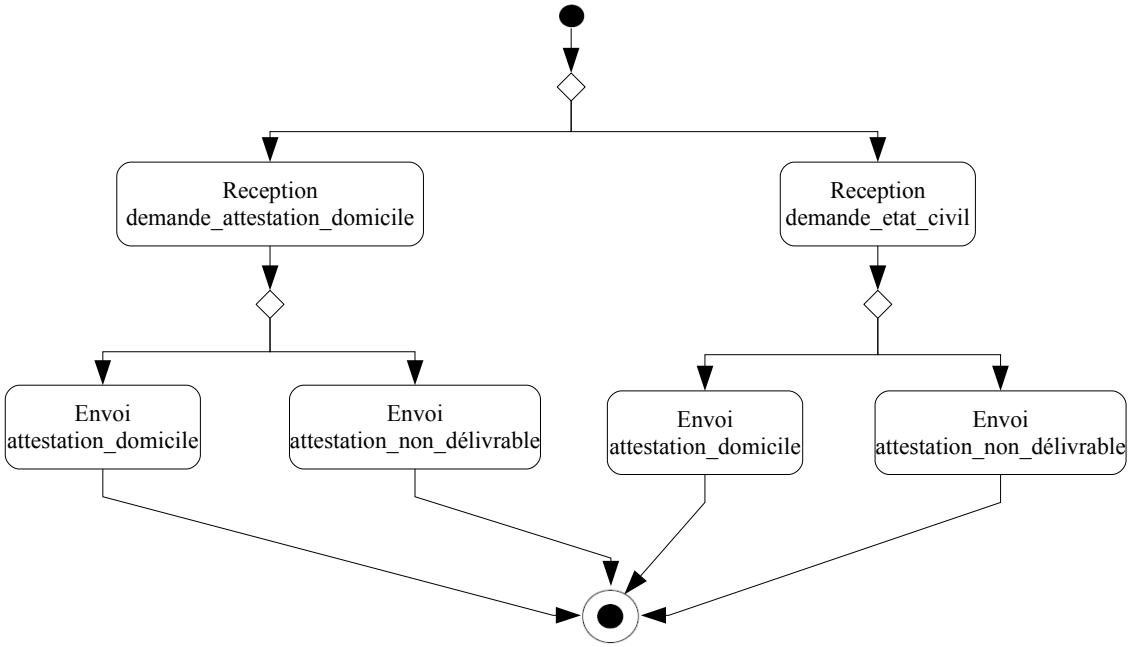


FIGURE 3.2 – Comportement du service de gestion de la mairie (SM)

3.2.2 Les propriétés temporelles dans les protocoles de conversations

Souvent, les échanges de messages entre les services Web doivent respecter des contraintes temporelles. L'envoi (resp. la réception) peut être fait dans un délai (resp. après un délai) bien précis. En dehors de ce délai, les conversations échouent.

Les propriétés temporelles se manifestent dans plusieurs scénarios, par exemple, dans le e-commerce (le client doit payer des articles dans les 24 heures qui suivent leur réservation), dans le e-healthcare (le médecin peut exiger de recevoir le résultat des analyses dans les 48 heures qui suivent leur demande).

Considérons le service de la préfecture illustré sur la Figure 3.1. Ce service a deux contraintes temporelles :

- Une fois que le formulaire de demande de bourse est envoyé au demandeur, ce dernier doit le retourner rempli dans les 24 heures.
- Une fois que la demande de bourse est reçue, la décision finale peut être envoyée après au moins 48 heures et au plus 96.

Pour tenir compte de ces propriétés temporelles, nous avons besoin d'un mécanisme qui permette de prendre en considération et de caractériser le temps entre deux événements qui correspondent à l'envoi et la réception de messages. Les séquences d'échanges de

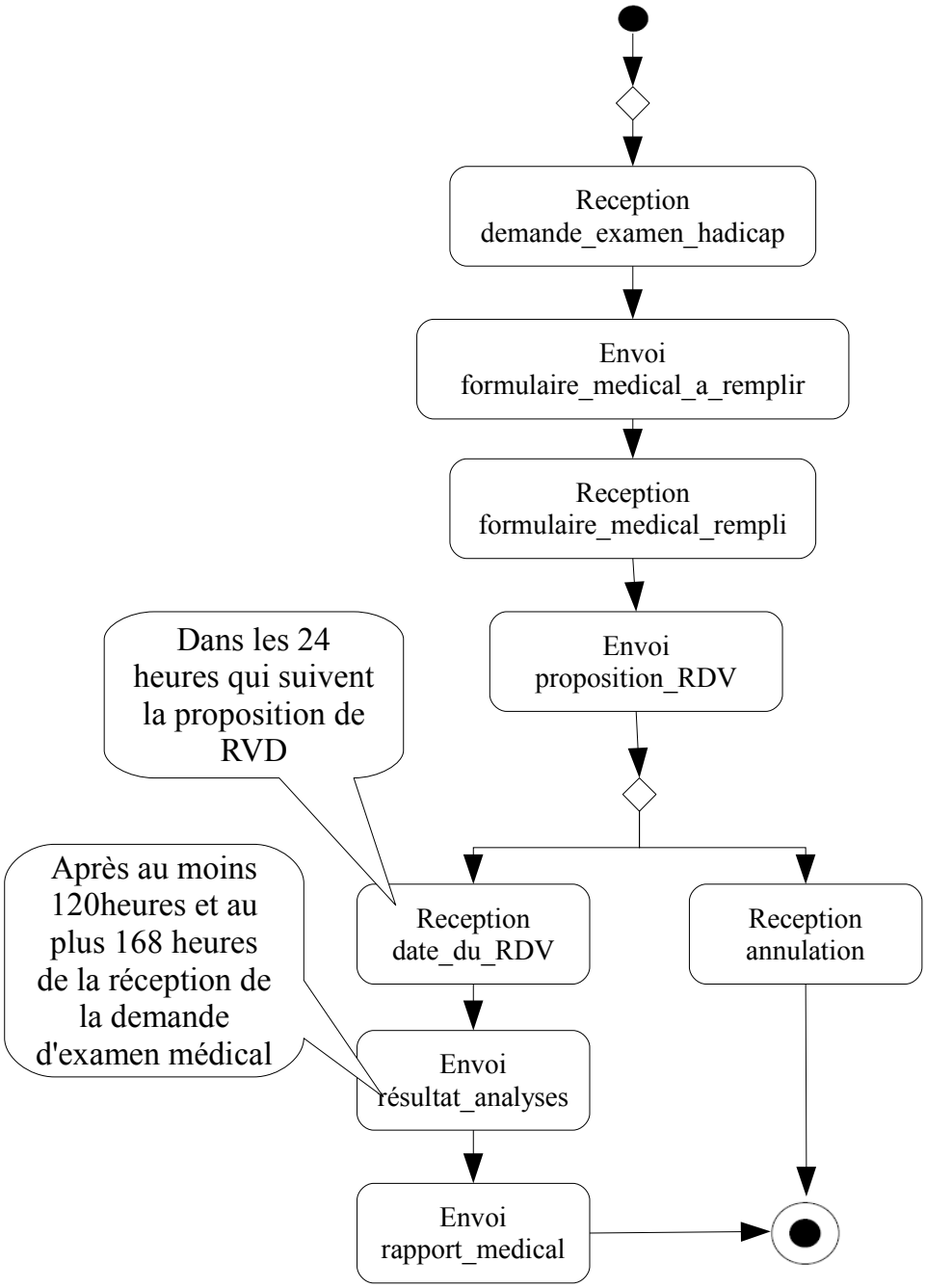


FIGURE 3.3 – Comportement du service de gestion de l'entité médicale (SEM)

messages dotées de contraintes temporelles constitue ce que l'on appelle des conversations temporisées. L'ensemble de ces conversations temporisées constituent un protocole de conversations temporisées. Dans ce qui suit, nous présentons formellement comment on modélise les protocoles de conversations temporisées.

3.3 Le modèle formel

Le modèle sur lequel se base le travail présenté dans cette thèse est principalement constitué de cinq parties : (1) les messages, (2) les types des données, (3) les contraintes de données, (4) les contraintes temporelles, et (5) le comportement qui est spécifié par les séquences de messages temporisées. Dans ce qui suit, nous évoquons les différents éléments liés au modèle formel.

3.3.1 Modélisation du comportement temporisé des services Web

Comme cité auparavant, nous adoptons le formalisme des automates temporisés pour modéliser les protocoles de conversations. Intuitivement, les états représentent les états du service Web et les transitions correspondent à l'envoi ou à la réception d'un message. L'échange d'un message déclenche le passage de l'état origine de la transition vers l'état cible. Les messages sortants sont notés par $!m$ et les messages entrant par $?m$. Un message impliquant des données est noté par $m(d_1, \ldots, d_n)$ ou $m(\overline{d})$. Des contraintes de données peuvent être associées à l'émission et/ou la réception de messages (i.e., aux transitions). Une contrainte de données, notée par c, spécifie une condition sur la valeur d'une donnée.

Etant donné qu'un message peut être envoyé dans un délai bien précis, les transitions peuvent être étiquetées par des contraintes temporelles (ou gardes). Pour modéliser ces contraintes, on adopte les horloges des automates temporisés [6]. Dans notre modèle, le fait qu'un message ne puisse être reçu (resp. envoyé) qu'après (resp. durant) une durée de la réception (resp. envoi) d'un autre message est spécifié par l'association d'horloges qui s'incrémentent par le passage de temps. Ces horloges peuvent être également remises à zéro.

Une contrainte temporelle est une conjonction de contraintes atomiques qui comparent la valeur d'une horloge $x \in X$, à une constante réelle positive $a \in \mathbb{R}_+$.

Soit X l'ensemble des horloges. L'ensemble des *contraintes* sur X, dénoté $\Psi(X)$, est défini par :

- true $\mid x \bowtie a \mid \psi_1 \wedge \psi_2$, tel que $\bowtie \in \{\leq, <, =, \neq, >, \geq\}$, $x \in X$, $\psi_1, \psi_2 \in \Psi(X)$ et a est une constante réelle.

On note que les contraintes de données sont aussi des conjonctions de contraintes atomiques qui comparent la valeur d'une donnée $d \in D$ à une constante qui peut être un entier, un réel, une chaine de caractère, un booléen.

Définition 4 *(Protocole de conversations temporisées)*

Un protocole de conversations temporisées d'un service Web Q est défini par le tuple (S, s_0, F, M, C, X, T) tels que

- *S est l'ensemble des états.*
- *s_0 est l'état initial ($s_0 \in S$).*
- *F est l'ensemble des états finaux ($F \subseteq S$).*
- *M est l'ensemble des messages.*
- *C est l'ensemble des contraintes de données.*
- *X est l'ensemble d'horloges.*
- *T est l'ensemble des transitions, tel que $T \subseteq S \times M \times C \times \Psi(X) \times 2^X \times S$. Une transition $(s, \alpha, c, \psi, Y, s')$ spécifie qu'à partir de l'état source s, le service atteint un nouvel état s' et ce en échangeant un message qui impliquent des données ($\alpha = ?m(\overline{d})$: message entrant ou $\alpha = !m(\overline{d})$:un message sortant) si les contraintes de données c et les contraintes temporelles horloges ψ sont satisfaites. Lors du déclenchement de cette transition, des horloges Y peuvent être réinitialisées.*

Une transition $(s, \alpha, c, \psi, Y, s')$ est dénotée par $s \rightarrow^{\alpha}_{c,\psi,Y} s'$.

Une trace est définie par une séquence de transitions qui, a partir d'un état initial, atteint l'état final. Ceci est dénoté par : $s_0 \rightarrow^{\alpha_0}_{c_0,\psi_0,Y_0} s_1 \rightarrow^{\alpha_1}_{c_1,\psi_1,Y_1} \ldots \rightarrow^{\alpha_{n-1}}_{c_{n-1},\psi_{n-1},Y_{n-1}} s_n$ tel que s_n est l'état final.

Les protocoles de conversations sont déterministes. Pour ceci, pour chaque paire de transitions $(s, m_1(\overline{d}), c_1, \psi_1, Y_1, s')$ et $(s, m_2(\overline{d}), c_2, \psi_2, Y_2, s'')$ sortantes de l'état s, les conditions suivantes doivent être satisfaites :

- $m_1(\overline{d}) \neq m_2(\overline{d})$, ou
- $c_1 \wedge c_2 = false$, ou
- $\psi_1 \wedge \psi_2 = false$

Exemple 4 *La figure 3.4 illustre les protocoles de conversations des services introduits dans la section 1.3.*

Sur cette figure, on voit que le service SP a comme état initial l'état p_0, l'ensemble des états est $\{p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}\}$ et l'ensemble des états finaux est $\{p_7, p_9, p_{16}\}$. Ce service peut envoyer des messages, comme par exemple le message demande_examen(ns, handicap,motif), dénoté !demande_examen(ns, handicap,motif). Ce message a comme paramètres le numéro de sécurité sociale (ns) et le type de handicap à examiner (handicap) et le motif de la demande de l'examen. D'une manière analogue, ce service peut recevoir un mesysage, comme par exemple le message $demande\_bourse(ns, age, handicap)$, dénoté $?demande\_bourse(ns, age, handicap)$.

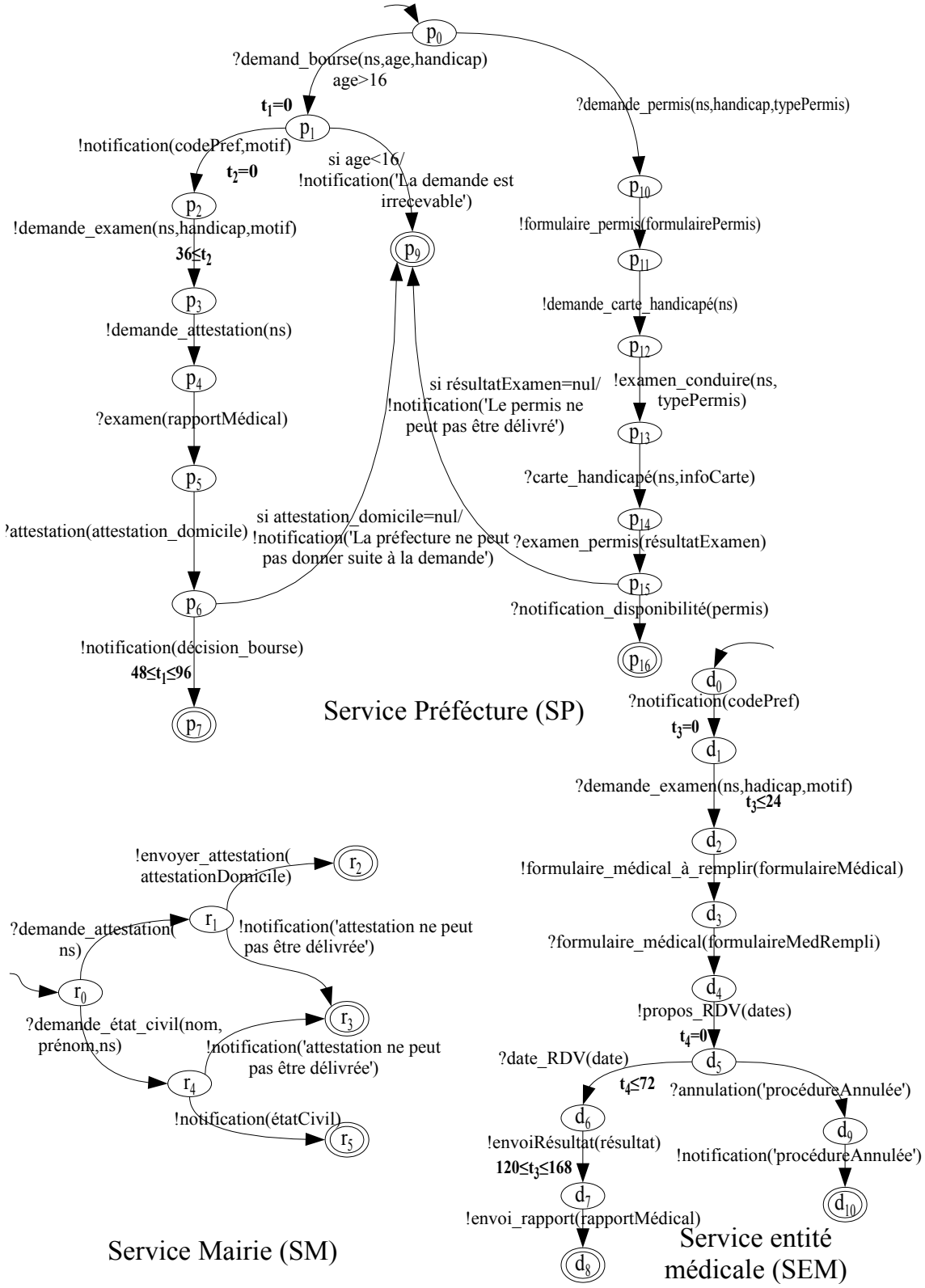


FIGURE 3.4 – Protocoles de conversations des services du e-pension

Ce message a comme paramètres le numéro de sécurité sociale (ns), l'age du demandeur (age), et le handicap que présente le demandeur (handicap). Ce service termine correctement son exécution si, pour chaque interaction, il atteint un état final.

Pour spécifier la contrainte qui dit que la décision de la préfecture peut être envoyée dans un délais de 48 à 96 heures après la réception de la demande, nous associons à la réception de la demande de bourse une initialisation d'horloge $t_1 = 0$ *et à l'envoi de la décision la contrainte* $48 \leq t_1 \leq 96$ *. Quand la demande est reçue, la valeur de l'horloge* t_1 *commence à s'incrémenter et la décision de la préfecture peut être envoyée si* $48 \leq t_1 \leq 96$.

Dans la suite, nous présentons la sémantique des protocoles de conversations en nous basant sur leur définition introduite ci-dessus. Nous commençons tout d'abord par introduire les concepts clés pour ensuite exposer la sémantique des protocoles de conversations temporisés.

3.3.2 Valuation d'horloges

La sémantique d'un protocole de conversation temporisé est basée sur la notion de valuation d'horloges v. Cette notion permet d'associer à chaque horloge x une valeur $v(x)$. Quand $v(x) = 0$, alors on dit que l'horloge est réinitialisée. On suppose que le domaine temporel est l'ensemble des réels positifs $\mathbb{R}_+$.

On dit qu'une valuation d'horloge $v(x)$ satisfait une contrainte $\psi = x \bowtie a$, noté par $v \models \psi$, si $v(x) \bowtie a$ (i.e., quand l'horloge x est remplacée par sa valeur, la contrainte ψ est satisfaite).

Définition 5 *(Valuation d'horloges)*

Une valuation des horloges de X *est une fonction* $v : X \to \mathbb{R}_+$*. Etant donné un réel* $\rho \in \mathbb{R}_+$*, on note* $v + \rho$ *la valuation qui associe à l'horloge* x *la valeur* $v(x) + \rho$*. Si* Y *est un sous ensemble de* X*,* $[Y \leftarrow 0]v$ *représente la valuation* v' *définie par :* $v'(x) = 0$ *pour tout* $x \in Y$ *et* $v'(x) = v(x)$ *pour* $x \in X \backslash Y$*.*

D'une manière analogue, on définit la valuation de données u qui permet d'associer à chaque donnée d une valeur $u(d)$. On dit qu'une valuation de donnée $u(d)$ satisfait une contraintes $c = d \bowtie b$, notée par $u \models c$, si $u(d) \bowtie b$ (i.e., quand la donnée d est remplacée par sa valeur, la contrainte c est satisfaite). Les valeurs des données changent via les opérations des services et ce en les calculant et en les mettant à jours.

3.3.3 Conversations temporisées

En utilisant la notion de valuation d'horloges (resp. de données), nous définissons la notion de conversations temporisées des protocoles de conversations inspirée de la notion des mots temporisés des automates temporisés [6]. Soit $Q = (S, s_0, F, M, C, X, T)$ un protocole de conversation. Une exécution de Q est une séquence de paires $s_0.(m_0(\overline{d}), t_0).s_1 \ldots s_{n-1}$ $(m_{n-1}(\overline{d}), t_{n-1}).s_n$ tels que s_0 est l'état initial et s_n est l'état final. Cette exécution est dite correcte si le temps avance d'une manière monotone de telle sorte que pour chaque transition de la trace, les contraintes de données et les contraintes temporelles sont satisfiables.

Une conversation est tout simplement l'ensemble des messages observables de l'extérieur qui peuvent être échangés entre les services Web.

Définition 6 *(Exécution temporisée correcte)*
Soit $Q = (S, s_0, F, M, C, X, T)$ un protocole de conversations temporisées. Une exécution, qui est une séquence $s_0.(m_0(\overline{d}), t_0).s_1 \ldots s_{n-1}(m_{n-1}(\overline{d}), t_{n-1}).s_n$ est dite correcte si :

- *$t_0 \leq t_1 \leq \ldots \leq t_{n-1}$*
- *s_0 est l'état intial et s_n est l'état final*
- *$\forall i \in [1, n], (s_{i-1}, m_{i-1}, c_{i-1}, \psi_{i-1}, Y_{i-1}, s_i)$, $v_{i-1} \models \psi_{i-1}$ et $u_{i-1} \models c_{i-1}$*

Une conversation correcte est alors définie par la séquence des paires $(m_0(\overline{d}), t_0) \ldots$ $(m_{n-1}(\overline{d}), t_{n-1})$. L'ensemble des conversations temporisées constitue un protocole de conversations temporisées.

3.3.4 Sémantique des protocoles de conversations temporisées

Après avoir exposé notre modèle, dans cette section nous présentons sa sémantique qui est spécifiée par un système de transition temporisé (STT). Ce système est défini par un ensemble de configurations et de transitions. Une configuration est définie sur l'ensemble des états ainsi que la valuation des valeurs des horloges et des données. La valuation v_0 spécifie la valuation initiale, telle que $\forall x \in X$, $v_0(x) = 0$. Ce qui revient à dire qu'au niveau de la configuration initiale, la valeur de toutes les horloges est égale à zéro. La valuation u_0 spécifie la valuation initiale de données, tel que $\forall d \in D$, $u_0(d) = null$. En outre, dans une configuration initiale, toutes les queues de tous les services sont vides.

A partir d'un état source s, le service y reste pendant que le temps s'incrémente, si il n'existe aucune transition $(s, \alpha, c, \psi, Y, s')$ telle que les contraintes c et ψ sont satisfiables et α est soit un message sortant soit un message entrant qui est disponible dans la queue.

Comme on considère des services Web asynchrones, quand un message est envoyé, alors il est inséré dans la queue, et quand le message est attendu, il peut être consommé de la queue si il y est disponible.

Définition 7 *(Semantique des protocoles de conversations temporisées)*
Soit $Q = (S, s_0, F, M, C, X, T)$ *un protocole de conversation. La sémantique de* Q *est définie par un système de transition temporisé STT* $(\Gamma, \gamma_0, \rightarrow)$*, tels que* $\Gamma \subseteq S \times V_T \times U_D$ *est l'ensemble des configurations, tel que* V_T *est l'ensemble des valuations des horloges,* U_D *est l'ensemble des valuations des données,* $\gamma_0 = (s_0, u_0, v_0)$ *est la configuration initiale, et* $\rightarrow$ *est défini comme suit :*

- Incrémentation du temps *:* $(s, u, v) \overset{tick}{\rightarrow} (s, u, v + \rho)$
- Changement d'état *:* $(s, u, v) \overset{\alpha}{\rightarrow} (s', u', v')$*, si* $\exists t = (s, \alpha, c, \psi, Y, s')$ *tel que* $u \models c$ *et* $v \models \psi$ *et* $\forall y \in Y, v'(y) = 0$, $\forall x \in X \backslash Y, v'(x) = v(x)$*, où* $Y \subseteq X$ *et*
  - *si* $\alpha = !m(\overline{d})$, $Que := Que + m(\overline{d})$
  - *sinon, si* $\alpha = ?m(\overline{d})$ *et* $m(\overline{d}) \in Que$, $Que := Que - m(\overline{d})$

Exemple 5 *Quand le service SP, illustré sur la figure 3.4, atteint son état* p_6 *et que la valeur de l'horloge* t_1 *est égale à* 30*, i.e., inférieure à* 48 *heures, alors le service séjourne dans l'état* p_6*. Pendant le séjour, le temps s'incrémente et par conséquence, la valeur de l'horloge* t_1 *s'incrémente sans que les valeurs des données soient changées. Quand la valeur de l'horloge* t_1 *est égale ou supérieure à 48, et en même temps, inférieure ou égale à 96, alors la transition* $(p_6, !notification(décision\_bourse), 48 \leq t_1 \leq 96, p_7)$ *se déclenche. Lors de ce déclenchement, des calculs peuvent être effectués, ce qui peut mettre à jour les valeurs des données. Comme la transition correspond à l'envoi du message notification(décision_bourse), alors il sera ajouté à la queue du service récepteur.*

3.4 Conclusion

Dans ce chapitre, nous avons introduit le modèle formel sur lequel la suite de ce manuscrit se base. Il s'agit d'une modélisation des protocoles de conversations temporisées basées sur les automates temporisés. La modélisation en automates temporisés des systèmes réels est simple à comprendre et en même temps expressive pour permettre de modéliser certaines propriétés, comme celles que l'on considère dans ce travail.

Nous avons présenté de manière informelle qui est un protocole de conversations temporisées. Puis, nous avons introduit formellement notre modèle qui est défini par : les

séquences des messages, les paramètres des messages, les contraintes de données, et les contraintes temporelles. Ensuite, nous avons présenté les éléments clé de notre modèle en nous appuyant sur des exemples. Nous avons ainsi introduit la notion de valuation des horloges qui a son tour nous permet de définir la notion de conversations temporisées. En utilisant ces concepts, nous avons présenté la sémantique des protocoles de conversations temporisées.

Chapitre 4

Analyse de l'interopérabilité de services Web asynchrones temporisés

Sommaire

4.1 Introduction 76

4.2 Le problème de compatibilité des services Web asynchrones temporisés 77

4.2.1 Aspects non-temporisés 78

4.2.2 Le rôle des propriétés temporelles 81

4.2.3 Classes de compatibilité 83

4.3 Transformation des protocoles de conversations 87

4.3.1 Abstraction des messages 88

4.3.2 Abstraction des contraintes de données 90

4.3.3 Association des invariants 92

4.3.4 Association d'un canal urgent aux transitions 93

4.3.5 Définition des états finaux 94

4.4 Analyse formelle de compatibilité 95

4.4.1 Compatibilité totale parfaite 97

4.4.2 Compatibilité totale non-parfaite 98

4.4.3 Compatibilité partielle non-parfaite 99

4.4.4 Compatibilité partielle parfaite 100

4.4.5 Incompatibilité totale 101

4.5 Implantation de l'analyse de compatibilité dans UPPAAL . . 102

4.5.1 Compatibilité totale parfaite 102

4.5.2 Compatibilité totale non-parfaite 103
4.5.3 Compatibilité partielle non-parfaite 104
4.5.4 Compatibilité partielle parfaite 105
4.5.5 Incompatibilité totale 106
4.6 Conclusion 108

4.1 Introduction

Le modèle défini pour spécifier et représenter les protocoles de conversations joue un rôle très important dans l'analyse des systèmes distribués et en particulier les services Web. En effet, la flexibilité et la pertinence de l'analyse dépendent des spécifications et des abstractions considérées. Dans le chapitre précédent, nous avons introduit le modèle formel sur lequel nous nous basons dans le travail présenté dans cette thèse.

Etant donné que différents services Web conçus et implémentés différemment peuvent être utilisés dans la même application, des outils d'analyse sont devenus cruciaux. Le but de ces outils est de vérifier que les interactions des différents services sont correctes, i.e., les services Web peuvent mener leurs conversations sans blocage. Ces vérifications sont très importantes, du fait qu'une mauvaise interaction (une interaction qui n'est pas correcte) peut avoir des conséquences dangereuses. En effet, les services Web sont de nos jours utilisés dans plusieurs domaines, notamment dans des domaines critiques, tel que le domaine médical. Donc, assurer que les interactions sont correctes est primordial.

Dans ce chapitre, nous nous concentrons sur un type d'analyse qui consiste en l'analyse de compatibilité globale des services Web, i.e., compatibilité dans le cadre de la chorégraphie. Ce type d'analyse est concerné par la vérification des conversations que peuvent mener un ensemble de services Web. Particulièrement, nous nous intéressons aux propriétés temporelles et à leur impact dans l'interaction des services Web qui communiquent d'une manière asynchrone.

Comme mentionné dans le chapitre 3, le modèle formel que nous proposons est basé sur les automates temporisés. Afin de mettre en œuvre l'analyse de compatibilité, nous proposons une démarche basée sur le model checking. Les model checkers proposés ces dernières années ont prouvé leur efficacité dans la simulation et l'analyse des systèmes réels. En général, le model ckecking repose sur des techniques automatiques d'exploration du comportement possible du système à analyser afin de pouvoir vérifier si des propriétés, qui correspondent à des formules logiques, sont satisfaites.

Nous commençons ce chapitre par rappeler intuitivement et d'une manière informelle le problème d'analyse de compatibilité de services Web asynchrones et temporisés. On note que notre modèle correspond au modèle sur lequel UPPAAL se base (voir section 2.3.5 du chapitre 2). Particulièrement, la notion de *canal* en UPPAAL correspond à la notion d'envoi et de réception de messages. Cependant, avec cette notion, seuls des services synchrones peuvent être analysés [59]. D'un autre coté, dans notre modèle, on considère les contraintes de données. Ces contraintes peuvent être spécifiées dans UPPAAL comme des contraintes sur des variables. UPPAAL peut considérer des contraintes sur des variables non-temporelles seulement si ces variables ont des valeurs. Etant donné que l'analyse que nous visons est applicable avant l'exécution des services Web, les variables ne peuvent pas avoir des valeurs bien définies. Par conséquent, lors de l'analyse des services Web avec UPPAAL, les contraintes de données ne peuvent pas être prises en considération. Afin de tenir compte de la nature asynchrone des services Web et des contraintes de données, nous proposons un ensemble de transformations que nous présentons dans la deuxième partie de ce chapitre. Finalement, nous présentons les primitives que nous proposons afin de caractériser le type d'interopérabilité (i.e., classe de compatibilité) que peuvent mener des services Web dans une chorégraphie.

4.2 Le problème de compatibilité des services Web asynchrones temporisés

Dans cette section, nous discutons de la notion de compatibilité des services Web temporisés dans une chorégraphie. Cette notion permet de s'assurer, avant la mise en œuvre d'une application complexe, du bon déroulement de la collaboration et ce en s'appuyant sur leur description comportementale temporisée, i.e., leur protocole de conversations temporisées.

Intuitivement, un ensemble de services Web est dit compatible si les services peuvent collaborer correctement. La collaboration se réalise par l'échange d'un ensemble de messages, i.e., par le biais de conversations. Une conversation durant laquelle aucun blocage n'apparait, est dite correcte. Les blocages qui peuvent surgir sont de deux types différents : *les blocages non-temporisés* et *les blocages temporisés*. Un blocage non-temporisé est principalement dû à l'hétérogénéité des services au niveau des types des messages, des types des données, des contraintes de données, et l'ordre dans lequel ces messages peuvent survenir. Quant au blocage temporisé, il est principalement engendré par l'impact que

peuvent avoir les différentes contraintes temporelles sur l'ensemble des interactions des services Web. Dans ce qui suit, via des exemples, nous illustrons la problématique de compatibilité.

4.2.1 Aspects non-temporisés

Avant de discuter l'impact des contraintes temporelles sur la chorégraphie, nous commençons tout d'abord par rappeler les blocages dûs aux aspects non-temporisés. Principalement, nous analysons l'importance de considérer l'aspect asynchrone des services Web. Ensuite, nous présentons le blocage que peut entrainer la divergence des types des messages, des types des données, et des contraintes de données. Finalement, nous abordons le blocage comportemental dû principalement à l'ordre inadéquat des échanges de messages dans une conversation.

Communications asynchrones

Afin de montrer l'importance de considérer l'aspect asynchrone lors d'une collaboration, nous présentons l'exemple qui suit. Dans cet exemple, nous considérons deux protocoles de conversations non-temporisés Q et Q' illustrés sur la figure 4.1. En utilisant les travaux de compatibilité proposés ces dernières années (e.g., [32, 15, 115, 65, 59]), les deux protocoles seraient désignés comme des services non-compatibles, i.e., ils ne peuvent pas communiquer ensemble. Ceci est dû au fait que les auteurs de ces travaux ne considèrent que des services synchrones. Selon ces travaux, un service qui supporte une séquence de message $!m_0(d_0, d_1), ?m_1(d_2), ?m_2(d_3)$ ne peut être compatible qu'avec un service qui supporte la séquence $?m_0(d_0, d_1), !m_1(d_2), !m_2(d_3)$. Cependant, cette hypothèse est très restrictive du fait que les services Web peuvent être de nature asynchrone. Néanmoins, et comme on va le montrer dans ce qui suit, ces deux services peuvent collaborer correctement, i.e., ils sont compatibles.

On rappelle que les services sont dotés d'une queue dans laquelle les messages émis sont stockés. Le service Q entame la conversation en envoyant le message $m_0(d_0, d_1)$ qui sera stocké dans la queue de Q'. D'un autre coté, Q envoie le message $m_2(d_3)$ qui sera stocké dans la queue de Q. Comme le message $m_0(d_0, d_1)$ est disponible dans la queue de Q', alors Q' peut le consommer. Une fois que la consommation est faite, Q' envoie le message $m_1(d_2)$ qui sera stocké dans la queue de Q. Une fois que le message $m_1(d_2)$ est disponible dans la queue, Q le consomme ainsi que le message $m_2(d_3)$. Ainsi, les deux

services atteignent leur état final en réussissant leur conversation. Ceci, en envoyant et en consommant les messages requis. En d'autres termes, on dit qu'ils sont *compatibles*.

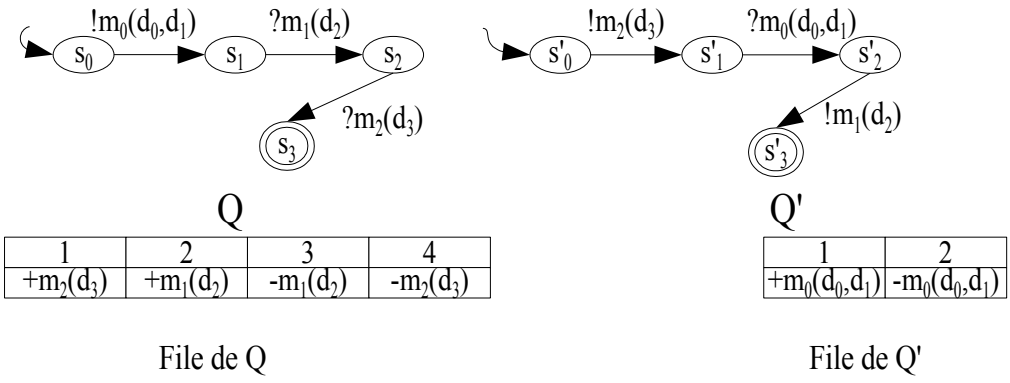


FIGURE 4.1 – Des services Web asynchrones compatibles

Hétérogénéités des types des messages

Comme nous l'avons expliqué précédemment, les services Web peuvent avoir des spécifications très différentes du point de vue des types des messages et de leur comportement. Comme le montre l'exemple illustré sur la figure 4.2, le service Q supporte les messages $m_3(d_0), m_2(d_2, d_1)$, et $m_0(d_1)$ et le service Q' supporte les messages $m_2(d_1), m_4(d_2)$, et $m_3(d_3)$. On peut remarquer que le message $m_0(d_1)$ est totalement incompréhensible pour le service Q', i.e., le service Q' ne peut pas envoyer ni consommer ce type de message car il n'existe pas dans les signatures de ses opérations.

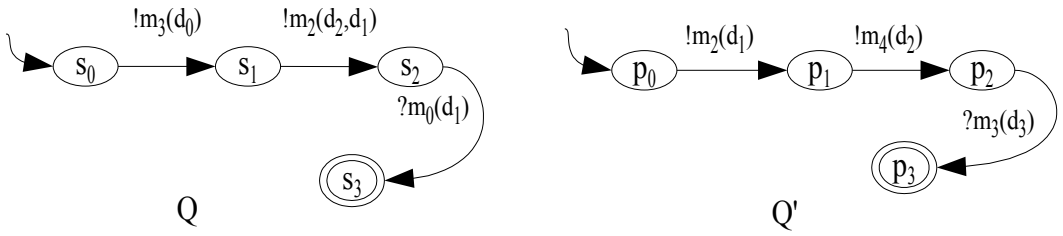


FIGURE 4.2 – Incompatibilité due à l' hétérogénéité des types des messages

Divergence des contraintes de données

La réception (resp. l'envoi) d'un message peut être conditionnée par une contrainte de données. Pour que deux services puissent échanger un message, il faut que les contraintes

de données associées à l'envoi et à la réception soient cohérentes, i.e., l'intersection des solutions des contraintes est non vide. Par exemple, à cause des contraintes de données les deux services Q et Q', illustrés sur la figure 4.3, la collaboration échoue. En effet, le service Q envoie le message $m_2(d_0)$ si la valeur de la donnée d_0 est supérieure à 20, mais pour que Q' puisse le consommer, la valeur de cette donnée doit être inférieure à 10. Comme on peut le remarquer, l'ensemble des solutions des deux contraintes est disjoint, i.e., les deux contraintes de données sont incohérentes, et par conséquent, la collaboration échoue.

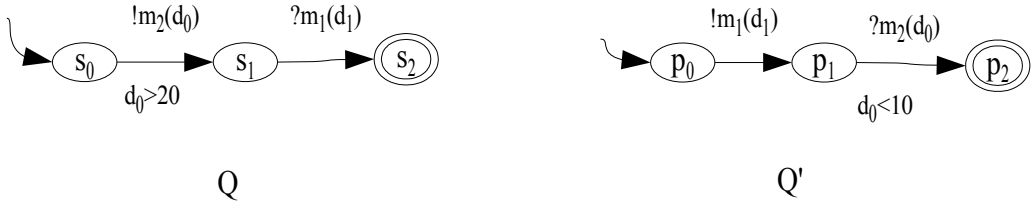


FIGURE 4.3 – Incompatibilité due à l'hétérogénéité des contraintes de données

Inter-blocage conversationnel

Chaque service a un comportement (i.e., séquencement des messages) qui diffère de celui des autres services Web. Comme le montre la figure 4.4, le service Q doit recevoir le message $m_3(d_0)$ pour envoyer le message $m_2(d_2, d_1)$ et en même temps, le service Q' doit recevoir le message $m_2(d_2, d_1)$ pour envoyer le message $m_3(d_0)$. Ce cas de figure présente un inter-blocage.

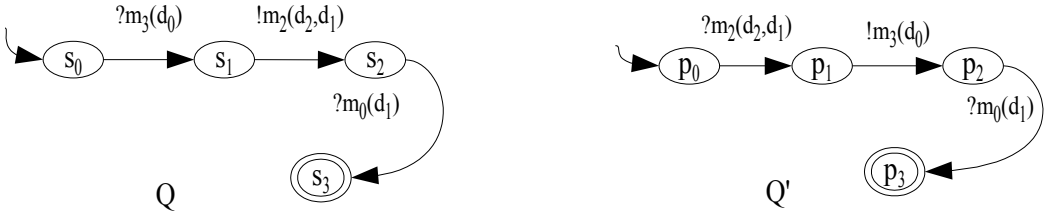


FIGURE 4.4 – Incompatibilité due à l'hétérogénéité des comportements

Après avoir présenté les aspects liés aux blocages non-temporisés, nous discutons les blocages temporisés dans ce qui suit.

4.2.2 Le rôle des propriétés temporelles

Etendre les protocoles de conversations par les propriétés temporelles augmente énormément la complexité. Ceci est dû au fait que les différentes contraintes temporelles de chaque service, utilisées pour spécifier les propriétés temporelles, sont locales et mutuellement indépendantes. Dès lors que les services Web collaborent, des dépendances entre les différentes contraintes temporelles peuvent être créées. En effet, par le biais de conversations qui peuvent être établies entre les différents services Web, les contraintes temporelles de chaque service peuvent avoir un effet sur le comportement des autres services.

Dans notre travail, nous supposons que les différentes contraintes temporelles ne sont pas synchronisées en se basant sur la synchronisation des messages, i.e., les horloges des contraintes temporelles associées à l'envoi et à la réception d'un même message peuvent ne pas être initialisées en même temps. Pour cela, lorsque les services interagissent entre eux, il ne suffit plus de comparer les contraintes temporelles associées à l'envoi avec celles associées à la réception des messages comme les contraintes de données. Pour illustrer ceci, nous présentons un exemple.

Soient les deux services Q et Q' illustrés sur la figure 4.5. Si on vérifie chaque couple de transitions qui permettent d'envoyer et de recevoir le même message, les deux services peuvent être définis comme compatibles. Par exemple, si on vérifie les contraintes des deux transitions $(s_1, ?m_1(d_2), s_2)$ et $(s'_2, !m_1(d_2), 20 \leq x \leq 40, s'_3)$, on peut dire que les deux services peuvent échanger le message $m_1(d_2)$. En effet, Q' envoie $m_1(d_2)$ si $20 \leq x \leq 40$ et Q n'a pas de contraintes pour le recevoir. Cependant, comme on va le voir, ces deux services ne peuvent pas collaborer correctement.

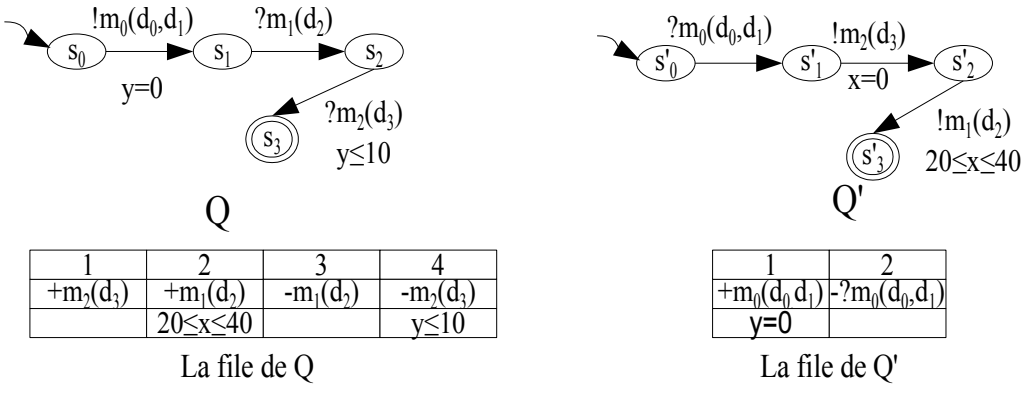


FIGURE 4.5 – Incompatibilité dûe aux propriétés temporelles

Le service Q commence par envoyer le message $m_0(d_0, d_1)$ qui sera stocké dans la queue

de Q'. En même temps, Q' peut consommer le message $m_0(d_0, d_1)$ qui est déjà envoyé par Q. Ensuite, Q' peut envoyer le message $m_2(d_3)$ qui peut être sauvegardé dans la queue de Q. Q reste bloqué dans l'attente du message $m_1(d_2)$ qui n'est pas encore envoyé. Q' envoie le message $m_1(d_2)$ après 20 unités de temps et avant 40 unité de temps de l'envoi de $m_2(d_3)$. Par conséquent, le message $m_1(d_2)$ devient accessible dans la queue de Q après 20 unités de temps de l'envoi du message $m_2(d_3)$. Finalement, Q consomme le message $m_2(d_3)$ dans 10 unités de temps. Cependant, ce message peut être envoyé après la consommation du message $m_1(d_2)$, i.e., après 20 unités de temps. En effet, le message $m_1(d_2)$ peut être envoyé par Q' seulement après 20 unités de temps. Donc si on examine la situation : le message $m_2(d_3)$ doit, d'une part être consommé dans les 10 unités de temps qui suivent l'envoi de $m_0(d_0, d_1)$ et, d'autre part, ne peut être consommé qu'après 20 unités de temps. Ceci représente un conflit temporel.

Dans le contexte de ce travail, nous supposons que le temps de communication (i.e., le temps du transit des messages sur le réseaux) est négligeable. Nous allons maintenant présenter un autre exemple de conflit qui ne peut pas être détecté en utilisant les travaux d'analyse de compatibilité temporisée existants (par exemple [15, 16, 113, 115, 114, 65, 59]). Considérons les deux services Q et Q' illustrés sur la figure 4.6. Q envoie le message $m_0(d_0)$, qui peut être stocké dans la queue de Q' qui peut ensuite le consommer. Q' envoie le message $m_1(d_1)$ qui peut être stocké dans la queue de Q, qui peut ensuite le consommer dans les 10 unités de temps qui suivent l'envoi de $m_0(d_0)$. Quand le message $m_1(d_1)$ est consommé, l'horloge y est remise à zéro et la valeur de l'horloge x doit être inférieure ou égale à 10. Etant donné que les horloges avancent d'une manière monotone, alors la différence entre les valeurs des deux horloges doit être inférieure ou égal à 10. Ensuite, Q envoie le message $m_2(d_1, d_0)$ 20 unités de temps après l'envoi du message $m_0(d_0)$. Q' peut consommer le message $m_2(d_1, d_0)$ dans les 5 unités de temps de l'envoi du message $m_1(d_1)$. Comme cité précédemment, la différence entre les valeurs des horloges x et y doit être inférieure ou égale à 10 unités de temps. Cependant, quand les deux services échangent le message $m_2(d_1, d_0)$, la valeur de l'horloge y doit être inférieure ou égale à 5 unités de temps ($y \leq 5$) et la valeur de l'horloge x doit être au moins 20 unités de temps ($x \geq 20$). Selon ces valeurs, la différence entre les deux horloges x et y ne peut jamais être inférieure ou égale à 10 unités de temps. Ce type de conflit n'est pas considéré dans les travaux existants sur la compatibilité temporisée des services Web.

Après avoir présenté les blocages qui peuvent surgir lors d'une collaboration de services

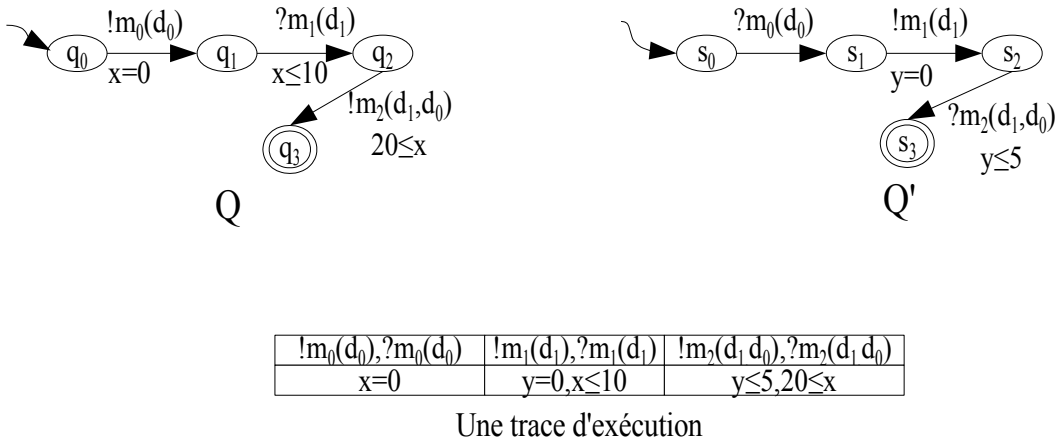


FIGURE 4.6 – Conflit temporisé dû à la différence entre la valeur des horloges

Web, dans la section suivante, nous présentons les différentes classes de compatibilité.

4.2.3 Classes de compatibilité

Les conflits peuvent avoir un impact partiel ou total sur la collaboration de services Web. En fonction de l'impact que peuvent avoir les conflits sur la collaboration de services Web, nous considérons trois classes de compatibilité asynchrone : (1) *compatibilité totale*, (2) *compatibilité partielle*, et (3) *incompatibilité totale*. La première et la deuxième classe regroupent deux sous classes : (a) *compatibilité parfaite* et (b) *compatibilité non-parfaite*. Dans ce qui suit, nous présentons informellement ces classes.

Compatibilité totale

Cette première classe de compatibilité regroupe des services Web qui peuvent collaborer sans blocage. Comme le montre la figure 4.7, les deux services Web sont totalement compatibles. En effet, aucun type de blocage présentés en Section 4.2 ne peut apparaître lors de l'interaction des deux services. Les deux services peuvent mener les conversations suivantes sans qu'un blocage temporisé ou non-temporisé ne survienne.

- $\Big((!m_0(d_1), t_1 = 0).!m_3(d_0).(!m_2(d_2, d_1), t_2 = 0).(?m_2(d_2, d_1), 168 \leq t_1 \leq 336).(?m_0(d_1), t_2 \leq 336).?m_3(d_0)\Big)$
- $\Big((!m_3(d_1), t_2 = 0).!m_4(d_1).(!m_5(d_1), t_2 \leq 24).?m_5(d_1).(?m_4(d_1), t_2 = 0).(?m_3, t_2 \leq 48)\Big)$

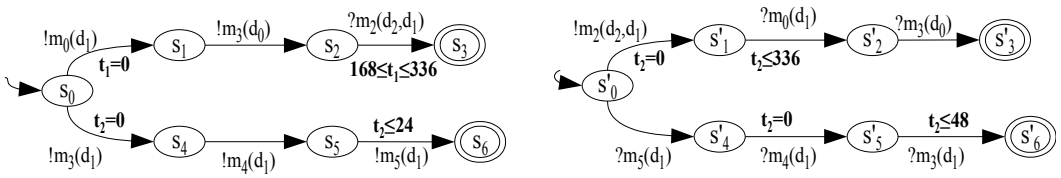


FIGURE 4.7 – Compatibilité totale

Etant donné que les services Web ne sont pas synchronisés, lors d'une collaboration, des messages peuvent être envoyés sans qu'ils soient consommés. Pour cela, nous distinguons deux sous classes : (1) compatibilité totale parfaite et (2) compatibilité totale non-parfaite.

1. *Compatibilité totale parfaite* : On dit qu'un ensemble de services Web est totalement et parfaitement compatible, si toutes les conversations ne contiennent aucun blocage et en même temps, tous les messages produits sont consommés. Par exemple, les services Web illustrés sur la figure 4.7 sont parfaitement et totalement compatibles.

2. *Compatibilité totale non-parfaite* : La compatibilité totale non-parfaite concerne des services Web qui peuvent mener à terme leur collaboration (i.e., sans blocage) mais en même temps, certains messages produits au cours de leurs conversations ne seraient pas consommés. Par exemple, les deux services illustrés sur la figure 4.8 sont totalement mais non-parfaitement compatibles. En effet, le service Q peut envoyer le message $m_3(d_4)$ qui ne sera pas consommé par Q'. Lors de l'interaction des deux services, aucun blocage ne peut survenir, mais il y a au moins un message qui est produit et qui ne sera pas consommé, c'est ce qu'on désigne par *compatibilité totale non-parfaite*.

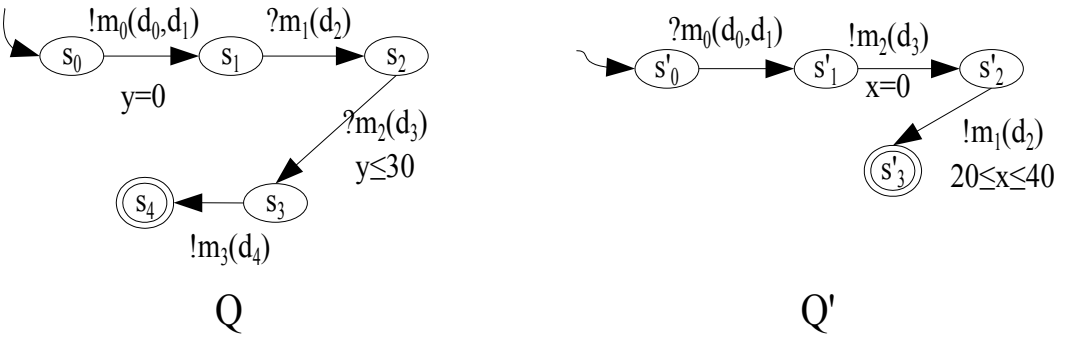


FIGURE 4.8 – Compatibilité totale non-parfaite

Compatibilité partielle

Cette deuxième classe de compatibilité est attribuée à l'ensemble des services Web qui ne sont pas totalement compatibles mais en même temps, peuvent mener quelques interactions correctes. L'hétérogénéité des services Web peut remettre en cause certaines exécutions des services Web ce qui les rend incorrectes mais en même temps, d'autres exécutions peuvent être correctes. Comme le montre la figure 4.9, les deux services peuvent mener correctement la conversation

$\Big((!m_0(d_1), t_1 = 0), !m_3(d_0), (!m_2(d_2, d_1), t_2 = 0), (?m_0(d_1), t_2 \leq 336), (?m_2(d_2, d_1), 168 \leq t_1 \leq 336), ?m_3(d_0)\Big)$

Cependant, la conversation

$\Big(!m_5(d_1), (!m_3(d_1), t_1 = 0), (?m_3(d_5), t_2 = 0), (!m_4(d_2), t_2 \geq 48), ?m_4(d_2), (?m_5(d_1), t_1 \leq 24)\Big)$

n'est pas correcte étant donné que le message $m_5(d_1)$ peut être consommé dans les 24 unités de temps qui suivent l'envoi du message $m_3(d_5)$ et en même temps $m_5(d_1)$ peut être consommé après la consommation du message $m_4(d_2)$, i.e., après 48 unités de temps après la consommation du message $m_3(d_5)$. En d'autres termes, le message $m_5(d_1)$ peut être consommé si $48 \leq t_1 \leq 24$, ce qui représente un blocage temporisé.

Puisque les deux services peuvent mener correctement au moins une conversation et en même temps peuvent en échouer en menant au moins une autre, alors on dit que les services sont *partiellement compatibles*.

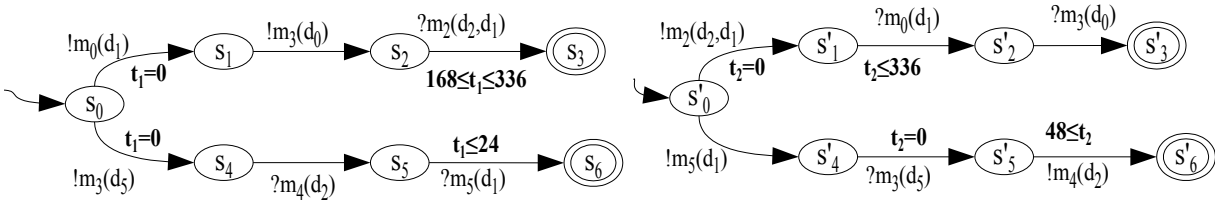


FIGURE 4.9 – Compatibilité partielle

Comme dans le cas de la compatibilité totale, on distingue deux sous classes de compatibilité partielles : (1) *compatibilité partielle parfaite* et (2) *compatibilité partielle non-parfaite*.

1. *Compatibilité partielle parfaite* : On dit qu'un ensemble de services Web est partiellement et parfaitement compatible, s'il existe au moins une conversation menant à un blocage et en même temps, ils peuvent mener d'autres conversations correctes

durant lesquelles tous les messages produits sont consommés. Par exemple, les deux services de la figure 4.9 sont partiellement et parfaitement compatibles.

2. *Compatibilité partielle non-parfaite* : La compatibilité partielle non-parfaite concerne des services Web partiellement compatibles où il existe au moins un message produit qui n'est pas consommé. Par exemple, les deux services illustrés sur la figure 4.10 sont partiellement mais non-parfaitement compatibles. En effet, les deux services échouent dans la conversation décrite dans la section 4.2.3, i.e., la conversation
$\Big(!m_5(d_1), (!m_3(d_1), t_1 = 0), (?m_3(d_5), t_2 = 0), (!m_4(d_2), t_2 \geq 48), ?m_4(d_2), (?m_5(d_1), t_1 \leq 24)\Big)$
mais en même temps, la conversation
$\Big((!m_0(d_1), t_1 = 0), !m_3(d_0), (!m_2(d_2, d_1), t_2 = 0), (?m_0(d_1), t_2 \leq 336), (?m_2(d_2, d_1), 168 \leq t_1 \leq 336), !m_5(d_0), ?m_3(d_0)\ \Big)$
ne contient aucun blocage. Mais dans cette conversation, le message $m_5(d_0)$ est un message qui ne sera pas consommé.

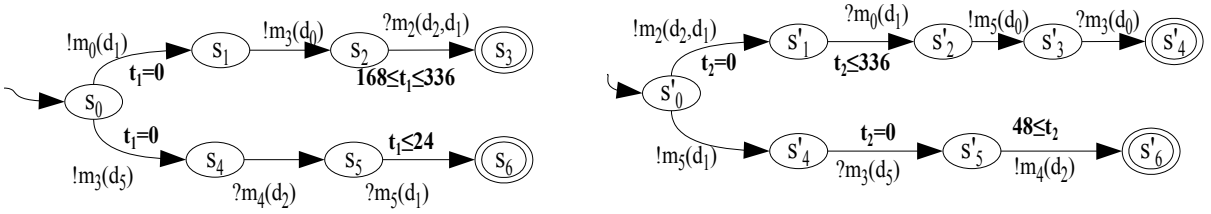


FIGURE 4.10 – Compatibilité partielle non-parfaite

Incompatibilité totale

La dernière classe que nous considérons concerne l'incompatibilité totale. Cette classe se rapporte à l'ensemble des services Web incapables de mener au moins une conversation correcte, i.e., les services échouent dans toutes leurs conversations. Comme le montre la figure 4.11, les deux services échouent toutes leurs conversations.

Quand Q' envoie le message $m_1(d_2, d_1)$, le service Q le reçoit et le consomme. Ensuite, les deux services restent bloqués en attente respectivement des deux messages $m_2(d_0)$ et $m_3(d_1)$. Cette première conversation échoue à cause d'un blocage non-temporisé.

Quant à la deuxième conversation, le même type de blocage temporisé présenté dans la section précédente apparaît.

Etant donné que les deux services échouent toutes leurs conversations, alors on dit qu'ils sont totalement incompatibles.

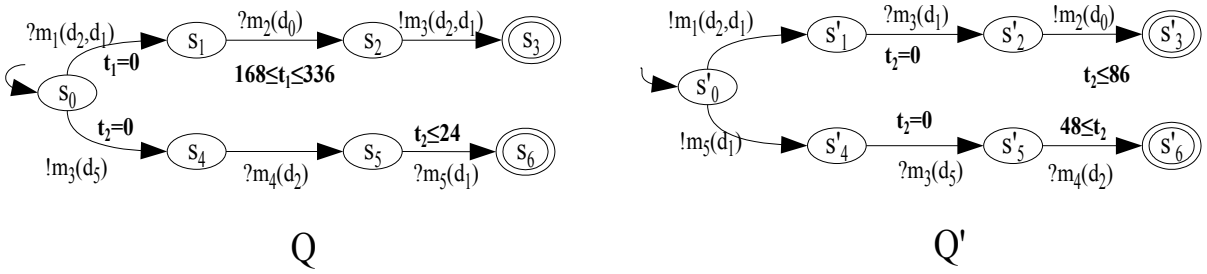


FIGURE 4.11 – Incompatibilité totale

Pour pouvoir caractériser l'interopérabilité que peuvent mener les services Web tout en tenant compte des éventuels conflits temporisés et non-temporisé, nous proposons une approche formelle de compatibilité basée sur le model checker UPPAAL. Dans ce qui suit, nous commençons tout d'abord par présenter les transformations que nous proposons afin de prendre en considération la sémantique des protocoles de conversations temporisées lors du model checking.

4.3 Transformation des protocoles de conversations

La nature des systèmes distribués et en particulier des services Web est asynchrone. Comme nous avions dit auparavant, nous proposons une approche basée sur le model checking. Particulièrement, le model checker utilisé est UPPAAL. UPPAAL repose sur un ensemble d'automates temporisés qui communiquent par une synchronisation binaire, utilisant des canaux et une syntaxe du type émission/réception. Ainsi, sur le canal *can*, un émetteur envoie un message *can*! et un récepteur se synchronise avec lui par la réception de ce message *can*?. En utilisant de tels concepts, seuls des services Web synchrones peuvent être analysés [59], ce qui est très restrictif.

En outre, la modélisation des contraintes de données dans UPPAAL par des conditions sur des variables ne tient pas compte de la sémantique des contraintes de données. En effet, les conditions sur les variables peuvent être considérées dans UPPAAL seulement si ces variables ont des valeurs. Comme les valeurs des données ne peuvent être connues et calculées que lors de l'exécution des services Web, alors les contraintes sur les variables ne peuvent pas simuler les contraintes de données. Dans le but d'appliquer le model

checking en utilisant UPPAAL, nous avons proposé une démarche de transformation et d'abstraction qui repose sur les étapes suivantes, que nous détaillons ensuite :

- *Abstraction des messages*
- *Abstraction des contraintes de données*
- *Association des invariants*
- *Association d'un canal urgent*
- *Définition des états finaux*

4.3.1 Abstraction des messages

Comme mentionné ci-dessus, UPPAAL utilise la notion de canal pour la synchronisation des processus. Dans le contexte de notre travail, cette notion correspond à celle des messages entrants et sortants ce qui permet de garder la sémantique de notre modèle. Cependant, lors de l'analyse, seuls des services synchrones peuvent être traités [59]. Etant donné que dans cette thèse nous nous intéressons à des services asynchrones temporisés, alors l'utilisation d'une telle notion est insuffisante et inadéquate.

Abstraction par variables

Afin de tenir compte des communications asynchrones, nous avons proposé d'abstraire les messages, qui correspondent à la notion de canal, par des variables. L'idée de base est de représenter chaque message par une variable dont la valeur initiale est égale à zéro. Le but est de représenter les mêmes types de messages ayant les mêmes types de données par la même variable. Ce type de variables est nommé dans la suite par *variables de messages*. Autrement dit, les messages ayant la même signature sont représentés par une variable unique, par exemple, si un service peut envoyer (resp. recevoir) le message $m_1(d_1, d_2)$ et l'autre service peut recevoir (resp. envoyer) le message $m_1(d_1, d_2)$, nous représentons ces messages, par exemple, par la variable m_1.

Abstraction par des opérations d'incrémentation et de décrémentation

Les services Web sont munis d'une queue pour stocker les messages entrants qui sont échangés d'une manière asynchrone. Lorsqu'un message est envoyé, ce message est stocké dans la queue. Donc le nombre d'occurrences de ce message s'incrémente de un. D'une manière analogue, quand un message est consommé de la queue, le nombre d'occurrences est décrémenté de un. Afin de simuler les transactions des queues d'attentes, nous proposons

de représenter la notion de messages sortants et entrants par des opérations d'incrémentation et de décrémentation des valeurs des variables de messages. En effet quand un message est envoyé alors on représente l'opération d'envoi de ce message par l'incrémentation de la variable de messages associée par un. Quand un message est consommé, alors on représente la notion de consommation du message par la décrémentation de la variable de messages correspondante. Un message peut être consommé seulement s'il existe dans la queue, ce qui revient à dire que la variable correspondante peut être décrémentée seulement si sa valeur est supérieure à zéro.

Définition 8 *Asbtraction des messages*
Soient $Q_1 = (S_1, s_{0_1}, M_1, C_1, X_1, T_1), \ldots, Q_n = (S_n, s_{0_n}, M_n, C_n, X_n, T_n)$ *n protocoles de conversations et R un ensemble de variables qui ont comme valeur initiale zéro. Nous définissons la fonction d'abstraction Abs* : $M \mapsto R$ *qui associe des variables à des messages. Pour chaque* $m(\overline{d}) \in \bigcup_{i=1}^{n} M_i$ *et* $r \in R$*, on a :*

- $(s_i, !m(\overline{d}), c, \psi, Y, s'_i) \mapsto (s_i, r++, c, \psi, Y, s'_i)$
- $(s_i, ?m(\overline{d}), c, \psi, Y, s'_i) \mapsto (s_i, r--, r > 0, c, \psi, Y, s'_i)$

Exemple 6 *Comme on peut voir sur la figure 4.12, l'ensemble des messages des deux services Q et Q' est* $\{m_0(d_1), m_3(d_0), m_2(d_2, d_1), m_1(d_1), m_2(d_3)\}$*. Nous représentons respectivement chaque type de message par une variable* $\{m_0, m_3, m_{21}, m_1, m_{22}\}$*. La transition* $(s_0, !m_0(d_1), s_1)$ *de Q permet d'envoyer le message* $m_0(d_1)$*, qui une fois envoyé, il sera stocké dans la queue de Q'. Par conséquent, nous représentons l'opération d'envoi de ce message par l'opération d'incrémentation de la valeur de la variable* m_0*. Ainsi, la transition* $(s_0, !m_0(d_1), s_1)$ *sera représentée par la transition* (s_0, m_0++, s_1)*. On applique ce processus sur toutes les transitions qui permettent d'envoyer un message. Quant aux transitions de réception de messages, comme par exemple la transition* $(s_2, ?m_2(d_2, d_1), s_3)$*, nous les représentons par l'opération de décrémentation de la variable correspondante. En effet, lors du déclenchement de la transition* $(s_2, ?m_2(d_2, d_1), s_3)$*, le message* $m_2(d_2, d_1)$ *sera retiré de la queue. Toutefois, pour que la transition* $(s_2, ?m_2(d_2, d_1), s_3)$ *puisse être déclenchée, le message* $m_2(d_2, d_1)$ *doit être disponible dans la queue. Cette condition revient à vérifier si la valeur de la variable correspondante est supérieure à zéro. En appliquant ce processus, la transition* $(s_2, ?m_2(d_2, d_1), s_3)$ *sera représentée par* $(s_2, m_{21}--, m_{21} > 0, s_3)$*.*

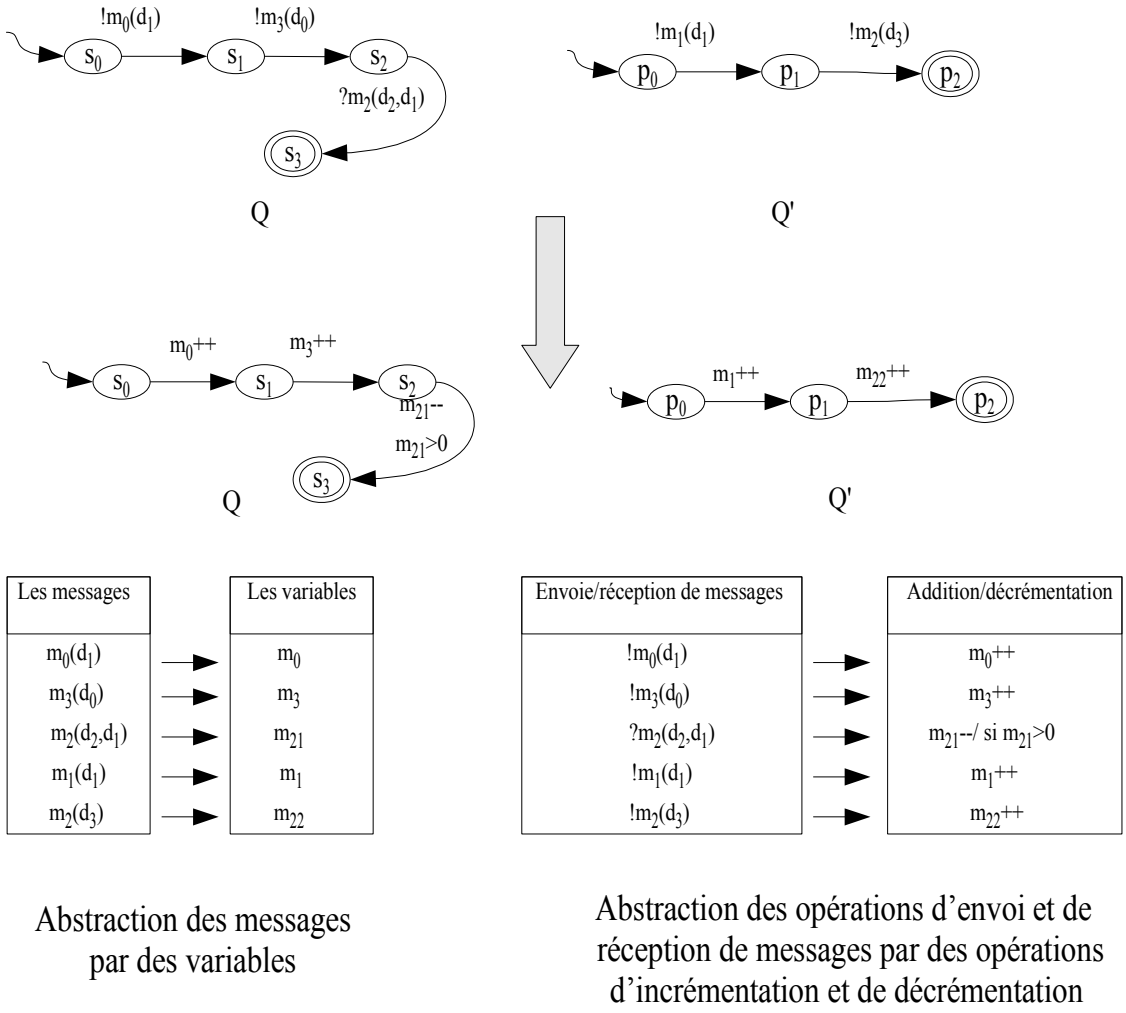


FIGURE 4.12 – Abstraction des messages

4.3.2 Abstraction des contraintes de données

Comme évoqué précédemment, avec UPPAAL on peut spécifier des contraintes de données qui peuvent être analysées seulement si les valeurs de ces données (i.e., les variables) sont connues. Cependant, les vraies valeurs peuvent être définies lors de l'exécution réelle des services Web. Etant donné que le travail présenté dans cette thèse s'inscrit dans le cadre d'analyse précédant l'exécution en temps réel, les contraintes de données ne peuvent pas être prises en considération correctement avec UPPAAL. Pour considérer les contraintes de données, nous proposons d'abstraire une deuxième fois les variables de messages, résultant du processus d'abstraction des messages décrit ci-dessus, suivant les contraintes de données. L'idée est d'abstraire différemment les mêmes variables de messages qui ont des contraintes de données disjointes.

Pour ce faire, on examine toutes les transitions qui sont étiquetées par la même variable de messages. Une fois l'ensemble des transitions communes défini, on isole les transitions dont l'ensemble des solutions des contraintes de données est disjoint. Puis, on représente les variables de messages par une autre variable de messages et on ôte les contraintes de données. Quant aux transitions dont l'ensemble des solutions des contraintes de données n'est pas disjoint, on retire les contraintes de données et on n'applique aucun changement sur leurs variables de messages.

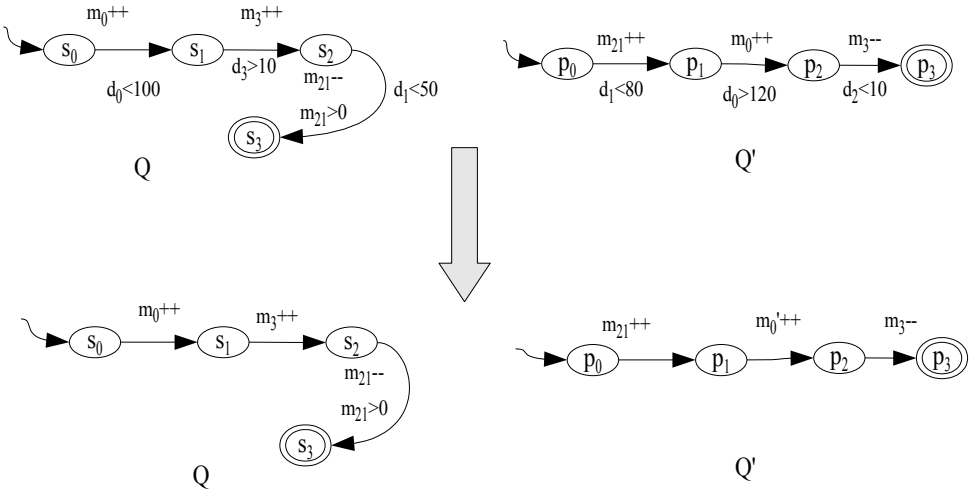


FIGURE 4.13 – Abstraction des contraintes de données

Exemple 7 *Dans cet exemple, on va expliquer le principe d'abstraction des contraintes de données. Comme on peut voir, les deux services, illustrés sur la figure 4.13, ont trois transitions communes (i.e., des transitions qui sont étiquetées par la même variable de messages) :*

- $(s_0, m_0++, d_0 < 100, s_1)$ *et* $(p_1, m_0++, d_0 > 120, p_2)$
- $(s_1, m_3++, d_3 > 10, s_2)$ *et* $(p_2, m_3--, m_3 > 0, d_2 < 10, p_3)$
- $(s_2, m_{21}--, m_{21} > 0, d_1 < 50, s_3)$ *et* $(p_0, m_{21}++, d_1 < 80, p_1)$

Commençons par le premier couple de transitions $(s_0, m0++, d_0 < 100, s_1)$ *et* $(p_1, m_0++, d_0 > 120, p_2)$. *On peut remarquer que l'ensemble des solutions (Sol()) des contraintes* $d_0 < 100$ *et* $d_0 > 120$ *est disjoint, i.e.,* $Sol(d_0 < 100) \cap Sol(d_0 > 120) = \emptyset$. *Donc, en appliquant l'étape d'abstraction des contraintes, on substitue* m_0++ *de la transition* $(p_1, m_0++, d_0 > 120, p_2)$ *par une autre variable* m'_0. *Alors la transition devient*

$(p_1, m'_0 + +, p_2)$.

Maintenant passons au deuxième couple de transitions $(s_1, m_3 + +, d_3 > 10, s_2)$ *et* $(p_2, m_3 - -, m_3 > 0,_2 < 10, p_3)$. *La première transition peut être déclenchée si la valeur de la donnée* d_3 *est supérieur à 10. La deuxième transition peut être déclenchée si la valeur de la variable* d_2 *est inférieures à 10. Etant donné que les contraintes de données sont définies sur des données différentes, alors lors de l'abstraction, on garde les mêmes variables pour les deux transitions et on fait abstraction des contraintes de données.*

Finalement, on examine le dernier couple de transitions $(s_2, m_{21} - -, m_{21} > 0, d_1 < 50, s_3)$ *et* $(p_0, m_{21}++, d_1 < 80, p_1)$. *On voit bien que l'ensemble des solutions des contraintes* $d_1 < 50$ *et* $d_1 < 80$ *n'est pas disjoint. Les deux contraintes ont un ensemble commun de solutions, i.e,* $Sol(d_1 < 50) \cap Sol(d_1 < 80) \neq \emptyset$. *Par conséquent, lors de l'application du processus d'abstraction des contraintes de données, on garde les mêmes variables de messages sans substitution.*

4.3.3 Association des invariants

La sémantique que nous considérons permet de déclencher les transitions dès que possible et sans attente. Lors de notre étude de UPPAAL, nous avons remarqué que lorsqu'une garde associée à une transition est de la forme $x \geq v$ tel que x est une horloge et v est une constante, alors le processus peut séjourner dans l'état source de la transition indifiniment. En effet, la transition peut être déclenchée à tout moment une fois que la valeur de l'horloge est supérieure à v. Par conséquent, UPPAAL signale un blocage, ce qui peut induire à des résultats d'analyse de compatibilité erronés.

Comme cité auparavant, le modèle sur lequel repose UPPAAL défini la notion d'invariant. Ce dernier permet au système de séjourner un temps limité dans un état. Afin d'éviter un blocage lors de l'analyse, nous associons à chaque état source d'une transition ayant une garde de la forme $x \geq v$ (resp. $x >$ v) un invariant de la forme $x \leq v$ (resp. $x < v$). Le but est de limiter le temps de séjours dans l'état.

Exemple 8 *La figure 4.14 illustre un protocole de conversation temporisé auquel nous associons un invariant. En l'analysant avec UPPAAL, un blocage au niveau de la transition* $(s_2, m_2 - -, m_2 > 0, 168 \leq t_1, s_3)$ *sera signalé. En effet, comme la transition a la garde* $168 \leq t_1$, *alors dans UPPAAL, le service peut séjourner indifiniment dans l'état*

s_2 puisque la transition $(s_2, m_2 - -, m_2 > 0, 168 \leq t_1, s_3)$ peut être déclenchée une fois que la valeur de l'horloge t_1 est supérieure à 168, i.e., le temps tend vers l'infini. Afin de restreindre le temps de séjour dans l'état s_2, nous lui associons l'invariant $t_1 \leq 168$.

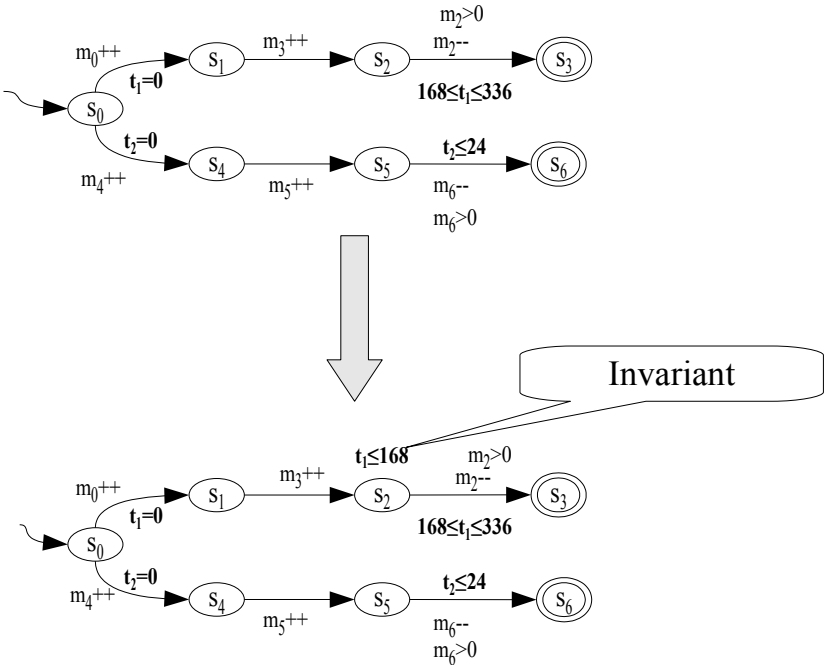


FIGURE 4.14 – Association des invariants

4.3.4 Association d'un canal urgent aux transitions

Dans la section précédente, nous avons montré comment nous traitons les transitions qui ont des gardes de la forme $x \geq v$ (resp. $x > v$). La sémantique que l'on considère permet de déclencher des transitions dès que possible. Afin de simuler cette propriété, nous associons un *canal sortant urgent* à toutes les transitions qui n'ont pas de contraintes temporelles. En effet, ces transitions n'ont pas de contraintes sur le temps de déclenchement et par conséquent, la transition peut être déclenchée à tout moment dans le temps. Le but de l'association d'un canal urgent est de forcer le déclenchement des transitions dès que possible et sans attente.

Dans UPPAAL, un canal *cv* peut être défini urgent comme suit

urgent chan cv ;

Etant donné que UPPAAL repose sur la synchronisation de messages et pour pouvoir envoyer un message sans forcément le synchroniser avec un message entrant, on définit

le canal urgent comme *broadcast*. Donc un canal urgent *cv* qui peut être envoyé sans synchronisation avec un message entrant est défini dans UPPAAL par :

urgent broadcast chan cv ;

Exemple 9 *Dans la figure 4.15, on associe aux transitions, qui n'ont pas de garde, du protocole Q le canal urgent cv afin de forcer leur déclenchement dès que possible.*

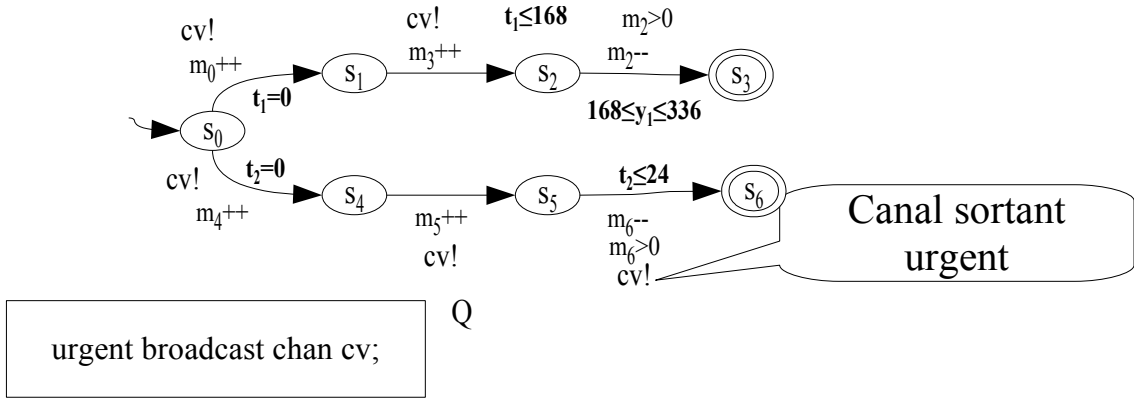


FIGURE 4.15 – Association d'un canal urgent

4.3.5 Définition des états finaux

Une interaction correcte est celle durant laquelle les services Web terminent leurs tâches. En d'autres termes les services Web atteignent leurs états finaux. Cette notion d'état final n'existe pas dans le modèle sur lequel UPPAAL repose. Afin de simuler la notion des états finaux, et comme illustré dans la figure 4.16, nous définissons un état particulier, nommé *final*, qui spécifie que les services terminent correctement leurs exécutions.

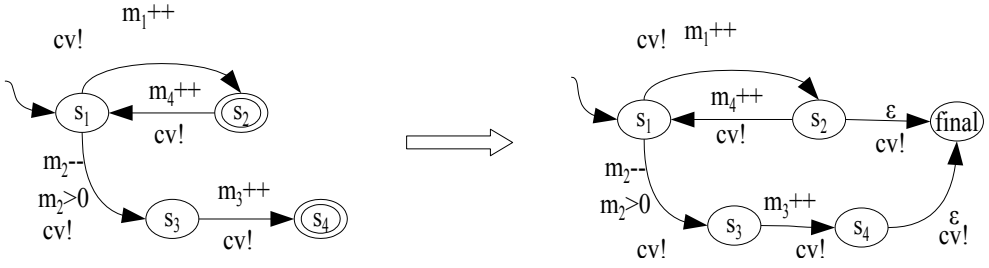


FIGURE 4.16 – Simulation des états finaux en UPPAAL

Le résultat du processus de transformation que nous avons décrit ci-dessus est un ensemble d'automates supportés par UPPAAL. Ces automates préservent la sémantique que nous considérons dans les protocoles de conversations temporisées asynchrones.

Définition 9 *(Protocole de conversations transformé)*
L'automate temporisé Q supporté par UPPAAL, résultant du processus de transformation, est défini par le tuple $(S, s_0, s_f, cu, R, X, VM, T, Inv)$ tel que :

- *S est l'ensemble des états,*
- *s_0 est l'état initial,*
- *s_f est l'état final,*
- *cu un canal urgent,*
- *R est l'ensemble des variables de messages,*
- *X est l'ensemble des horloges,*
- *VM est l'ensemble des contraintes sur les variables de messages,*
- *T est l'ensemble de transition tel que $T \subseteq S \times OP(R) \times VM \times \Psi(X) \times 2^X \times S$ qui spécifie que lors du déclenchement d'une transition, on effectue une opération OP (on incrémente la variable de messages par un $r++$: correspond à un message envoyé, décrémentation de la variable par un $r--$: correspond à un message consommé), une contrainte sur les variables VM (si OP(R)=r–, $VM = r > 0$, sinon $VM = \emptyset$), une contrainte temporelle, et l'ensemble des horloges à réinitialiser).*
- *$Inv : S \rightarrow \Psi(X)$ associe un invariant aux états*

Figure 4.17 montre les automates temporisés résultant du processus d'abstraction des protocoles de conversations des services Web introduits dans la section 1.3.

Après avoir présenté les étapes de transformation, dans ce qui suit, nous présentons la démarche formelle que nous proposons pour caractériser l'interopérabilité que peuvent mener un ensemble de services Web dans une chorégraphie.

4.4 Analyse formelle de compatibilité

Dans la section précédente, nous avons montré la nécessité et l'importance d'une approche formelle d'analyse de compatibilité de services Web. Dans la suite de cette section, nous présentons les primitives qui, en se basant sur le modèle résultant du processus de transformation, permettent de classer le type d'interopérabilité que peuvent mener un ensemble de services Web.

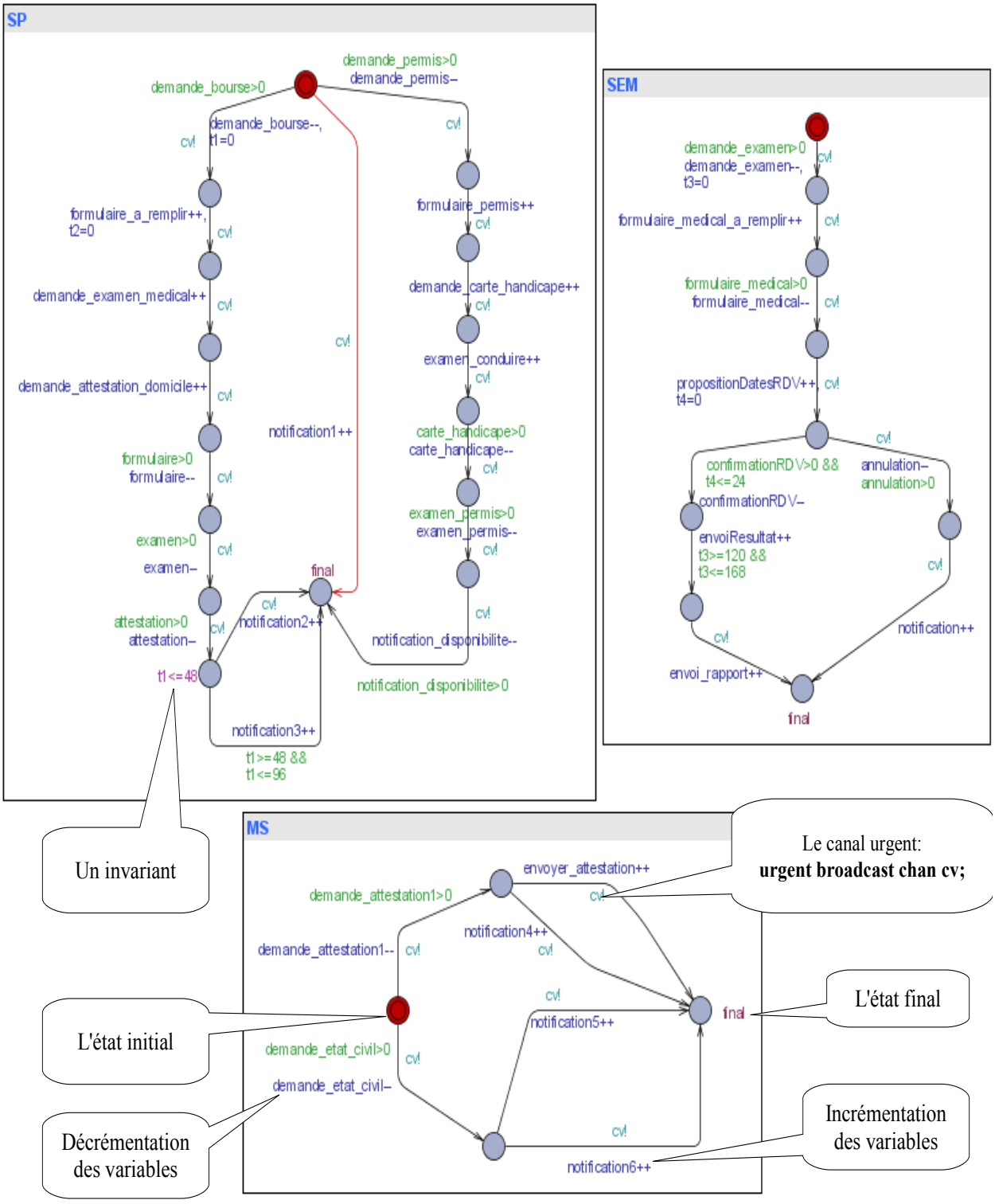


FIGURE 4.17 – Les automates temporisés supportés par UPPAAL résultants du processus de transformation

Nous distinguons cinq classes de compatibilité : (1) *compatibilité totale parfaite*, (2) *compatibilité totale non-parfaite*, (3) *compatibilité partielle parfaite*, (4) *compatibilité partielle non-parfaite* et (5) *incompatibilité totale*.

4.4.1 Compatibilité totale parfaite

En général, un ensemble de services Web est dit totalement et parfaitement compatible, si lors de la collaboration des services, aucun blocage ne peut apparaître. En outre, puisque les services Web sont de nature asynchrone, i.e., l'envoi et la réception des messages ne sont pas synchronisés, alors pour que les services soient totalement et parfaitement compatibles, on doit vérifier que tous les messages sortants sont consommés. Comme mentionné dans la section 4.2, les blocages peuvent survenir pour des raisons liés aux aspects temporisés et non-temporisés des services Web.

En récapitulant, un ensemble de services Web temporisés asynchrones est dit totalement et parfaitement compatible si tous les messages entrants peuvent être consommés (aucun blocage temporisé ou non-temporisé ne surgit) et en même temps, tous les messages sortants peuvent être tous consommés (pas de messages supplémentaires).

Formellement, ceci revient à vérifier que tous les services atteignent leurs états finaux et que, lorsque les services atteignent leurs états finaux, la valeur de toutes les variables de messages doit être égale à zéro. En effet, la valeur des variables représente le nombre d'occurrences du message envoyé. Quand un message est envoyé (resp. consommée), la variable correspondante est incrémentée (resp. décrémentée).

Soient P_1, ..., P_n, n services et $R_1, \ldots, R_n$ l'ensemble des variables de messages. L'ensemble des services est dit totalement et parfaitement compatible si la formule CTL suivante est vérifiée.

$$\begin{array}{|l} \mathbf{AF}\ P_1.final\ \wedge\ \ldots\ \wedge\ P_n.final \\ \wedge \\ \mathbf{AG}\ (P_1.final\ \wedge\ \ldots\ \wedge\ P_n.final \Rightarrow \mathbf{AF}\ r_1 == 0 \wedge \ldots \wedge r_m == 0) \\ \text{tel que } r_i \in \{R_1, \ldots R_i, \ldots, R_n\} \end{array} \tag{4.1}$$

En d'autres termes, la formule 4.1, spécifie que tous les chemins de chaque service mènent vers l'état '*final*' et en même temps, quand les services atteignent leur état final, la valeur de toutes les variables de messages est égale à zéro.

Exemple 10 *Les services SP, SM, et SEM introduits dans la section 1.3 sont dits totalement et parfaitement compatibles si*

$$\begin{array}{l}\boldsymbol{AF}\ SP.final \wedge SM.final \wedge SEM.final \wedge \\ \boldsymbol{AG}\ (SP.final \wedge SM.final \wedge SEM.final \Rightarrow \boldsymbol{AF}\ notification\_disponibilite{=}{=}0 \wedge \\ examen{=}{=}0 \wedge examen\_permis{=}{=}0 \wedge carte\_handicape{=}{=}0 \wedge examen\_conduire{=}{=}0 \\ \wedge\ formulaire{=}{=}0 \wedge demande\_carte\_handicape{=}{=}0 \wedge formulaire\_permis{=}{=}0 \wedge \\ demande\_permis{=}{=}0 \wedge notification2{=}{=}0 \wedge notification1{=}{=}0 \wedge notification3{=}{=}0 \\ \wedge\ demande\_attestation1{=}{=}0 \wedge notification4{=}{=}0 \wedge demande\_carte\_hadicape{=}{=}0 \\ \wedge\ demande\_attestation\_domicile{=}{=}0 \wedge demande\_examen\_medical{=}{=}0 \wedge \\ formulaire\_a\_remplir{=}{=}0 \wedge demande\_bourse{=}{=}0 \wedge notification6{=}{=}0 \\ \wedge\ notification5{=}{=}0 \wedge envoyer\_attestation{=}{=}0 \wedge envoiResultat{=}{=}0 \wedge \\ demande\_etat\_civil{=}{=}0 \wedge envoi\_rapport{=}{=}0 \wedge formulaire\_medical\_a\_remplir{=}{=}0 \\ \wedge\ propositionDatesRDV{=}{=}0 \wedge formulaire\_medical{=}{=}0 \wedge confirmationRDV{=}{=}0 \wedge \\ demande\_examen{=}{=}0 \wedge attestation{=}{=}0 \wedge annulation{=}{=}0 \wedge notification{=}{=}0)\end{array}$$

4.4.2 Compatibilité totale non-parfaite

Quand les services Web peuvent collaborer ensemble sans blocage mais durant leurs interactions, des messages produits ne sont pas consommés, alors on dit que ces services sont *totalement mais non-parfaitement compatibles*.

Formellement, un ensemble de services Web est dit totalement mais non-parfaitement compatible si tous les chemins de chaque service mènent vers l'état final mais en même temps, il existe au moins une variable de messages dont la valeur est supérieure à zéro. Ceci est spécifié par la formule CTL suivante :

$$\begin{array}{l}\mathbf{AF}\ P_1.final\ \wedge\ \ldots\ \wedge\ P_n.final \\ \wedge \\ \mathbf{EF}(P_1.final \wedge \ldots \wedge P_n.final \Rightarrow r_1 > 0 \vee \ldots \vee r_n > 0) \\ \text{where } r_i \in \{R_1, \ldots R_i, \ldots, R_n\}\end{array} \tag{4.2}$$

Exemple 11 *Les services SP, SM and SEM introduits dans la section 1.3 sont dits totalement mais non-parfaitement compatibles si :*

$$\boldsymbol{AF}SP.final \wedge SM.final \wedge SEM.final \wedge$$
$$\boldsymbol{EF}(SP.final \wedge SM.final \wedge SEM.final \Rightarrow notification\_disponibilite{>}0 \vee examen{>}0$$
$$\vee\ examen\_permis{>}0 \vee carte\_handicape{>}0 \vee examen\_conduire{>}0 \vee formulaire{>}0 \vee$$
$$demande\_carte\_handicape{>}0 \vee formulaire\_permis{>}0 \vee demande\_permis{>}0 \vee no-$$
$$tification2{>}0 \vee notification1{>}0 \vee notification3{>}0 \vee demande\_attestation1{>}0 \vee no-$$
$$tification4{==}0 \vee demande\_carte\_hadicape{>}0 \vee demande\_attestation\_domicile{>}0 \vee$$
$$demande\_examen\_medical{>}0 \vee formulaire\_a\_remplir{>}0 \vee demande\_bourse{>}0 \vee$$
$$notification6{>}0 \vee notification5{>}0 \vee envoyer\_attestation{>}0 \vee envoiResultat{>}0$$
$$\vee\ demande\_etat\_civil{>}0 \vee envoi\_rapport{>}0 \vee formulaire\_medical\_a\_remplir{>}0$$
$$\vee\ propositionDatesRDV{>}0 \vee formulaire\_medical{>}0 \vee confirmationRDV{>}0 \vee$$
$$demande\_examen{>}0 \vee attestation{>}0 \vee annulation{>}0 \vee notification{>}0)$$

4.4.3 Compatibilité partielle non-parfaite

Etant donnée l'hétérogénéité des services Web, dans une chorégraphie, les services peuvent mener des conversations incorrectes. Une conversation échouée, ou appelée *incorrecte*, est une conversation dans laquelle il y a au moins un service qui reste bloqué en attente d'un message qu'il ne peut pas recevoir ou envoyer. Les services Web dans une chorégraphie sont dits incompatibles dès lors que l'ensemble des conversations échangées contient au moins une conversation incorrecte.

Formellement, un ensemble de services Web n'est pas totalement compatible, s'il existe au moins un chemin qui ne mène pas vers l'état final. Cela peut être spécifié par la formule CTL suivante.

$$\mathbf{EG}\ \neg P_1.final\ \vee\ \ldots\ \vee\ \mathbf{EG}\ \neg P_n.final \vee \tag{4.3}$$

En d'autres termes, la formule 4.3, spécifie qu'il existe au moins un chemin d'un service pour lequel l'état final n'est pas accessible.

Dans une chorégraphie, les services Web peuvent mener correctement des conversations et d'un autre coté, ils peuvent en échouer d'autres. Ce cas de figure est désigné par *compatibilité partielle*.

Dans cette section, nous définissons particulièrement la classe de *compatibilité partielle non-parfaite*. Cette classe est attribuée à l'ensemble de services Web partiellement

compatibles qui, lors de leur interaction, il y a au moins une conversation correcte durant laquelle au moins un message produit n'est pas consommé. Formellement, cela est équivalent à dire qu'il existe au moins un chemin via lequel tous les services atteignent leur état final et en même temps quand ils atteignent leur état final, il y a au moins une variable de messages dont la valeur est supérieure à zéro. Ceci est spécifié par la formule CTL suivante :

$$\begin{array}{|l|}\hline \mathbf{EF}P_1.final \ \wedge \ \ldots \ \wedge \ P_n.final \ \wedge \\ \mathbf{EF}(P_1.final \ \wedge \ \ldots \ \wedge \ P_n.final \Rightarrow r_1 > 0 \ \wedge \ \ldots \ \wedge \ r_m > 0) \\ \text{where } r_i \in \{R_1, \ldots R_i, \ldots, R_n\} \\ \hline \end{array} \quad (4.4)$$

Formellement, un ensemble de services Web est dit partiellement mais non-parfaitement compatible si les formule 4.3 et 4.4 sont vérifiées.

Exemple 12 *Les services SP, SM and SEM sont dits partiellement mais non-parfaitement compatibles si :*

$$\begin{array}{|l|}\hline \boldsymbol{EG} \neg SP.final \vee \ \boldsymbol{EG} \neg SM.final \vee \ \boldsymbol{EG} \neg SEM.final \wedge \\ \boldsymbol{EF} SP.final \wedge SM.final \wedge SEM.final \wedge \\ \boldsymbol{EF} \ (SP.final \wedge SM.final \wedge SEM.final \Rightarrow notification\_disponibilite{>}0 \vee examen{>}0 \\ \vee \ examen\_permis{>}0 \vee carte\_handicape{>}0 \vee \ examen\_conduire{>}0 \vee formulaire{>}0 \vee \\ demande\_carte\_handicape{>}0 \vee formulaire\_permis{>}0 \vee \ demande\_permis{>}0 \vee \ no\text{-} \\ tification2{>}0 \vee notification1{>}0 \vee notification3{>}0 \vee demande\_attestation1{>}0 \vee \ no\text{-} \\ tification4{==}0 \vee demande\_carte\_hadicape{>}0 \vee demande\_attestation\_domicile{>}0 \vee \\ demande\_examen\_medical{>}0 \vee \ formulaire\_a\_remplir{>}0 \ \vee \ demande\_bourse{>}0 \ \vee \\ notification6{>}0 \ \vee \ notification5{>}0 \ \vee \ envoyer\_attestation{>}0 \ \vee \ envoiResultat{>}0 \\ \vee \ demande\_etat\_civil{>}0 \ \vee \ envoi\_rapport{>}0 \ \vee \ formulaire\_medical\_a\_remplir{>}0 \\ \vee \ propositionDatesRDV{>}0 \ \vee \ formulaire\_medical{>}0 \ \vee \ confirmationRDV{>}0 \ \vee \\ demande\_examen{>}0 \vee attestation{>}0 \vee annulation{>}0 \vee notification{>}0) \\ \hline \end{array}$$

4.4.4 Compatibilité partielle parfaite

Cette classe de compatiblité caractérise le fait qu'un ensemble de services Web n'est pas totalement compatible mais en même temps, les services peuvent mener correctement des conversations, et dans ces conversations tous les messages produits sont consommés.

Formellement, un ensemble de services Web peut mener correctement des conversations tel que tous les messages produits durant cette conversation sont consommés, revient à vérifier qu'il existe des chemins via lesquels tous les services atteignent leur état final et en même temps quand ils atteignent leur état final, la valeur de toutes les variables est égale à zéro. Ceci est spécifié par la formule CTL suivante :

$$\begin{array}{|l|}\hline \textbf{EF}\ P_1.final\ \wedge\ \ldots\ \wedge\ P_n.final\ \wedge \\ \textbf{EF}\ (P_1.final\ \wedge\ \ldots\ \wedge\ P_n.final \Rightarrow r_1 == 0\ \wedge\ \ldots\ \wedge\ r_m == 0) \\ \text{where } r_i \in \{R_1, \ldots R_i, \ldots, R_n\} \\ \hline \end{array} \quad (4.5)$$

L'ensemble de services Web, dont leur protocoles de conversations ne vérifient pas la formule 4.4 (i.e., la compatibilité partielle et non-parfaite n'est pas vérifiée) et en même temps vérifient les formules 4.3 et 4.5, sont dit *partiellement et parfaitement compatibles.*

Exemple 13 *Les services SP, SM and SEM sont dits partiellement et parfaitement compatibles si la formule présentée dans l'exemple 12 n'est pas vérifiée et la formule suivante est vérifiée :*

$$\begin{array}{|l|}\hline \boldsymbol{EG}\ \neg SP.final \vee\ \boldsymbol{EG}\ \neg SM.final \vee\ \boldsymbol{EG} \neg SEM.final \wedge \\ \boldsymbol{EF}\ SP.final \wedge SM.final \wedge SEM.final \wedge \\ \boldsymbol{EF}\ (SP.final \wedge SM.final \wedge SEM.final \wedge \Rightarrow notification\_disponibilite==0 \wedge examen==0 \wedge examen\_permis==0 \wedge carte\_handicape==0 \wedge examen\_conduire==0 \wedge formulaire==0 \wedge demande\_carte\_handicape==0 \wedge formulaire\_permis==0 \wedge demande\_permis==0 \wedge notification2==0 \wedge notification1==0 \wedge notification3==0 \wedge demande\_attestation1==0 \wedge notification4==0 \wedge demande\_carte\_hadicape==0 \wedge demande\_attestation\_domicile==0 \wedge demande\_examen\_medical==0 \wedge formulaire\_a\_remplir==0 \wedge demande\_bourse==0 \wedge notification6==0 \wedge notification5==0 \wedge envoyer\_attestation==0 \wedge envoiResultat==0 \wedge demande\_etat\_civil==0 \wedge envoi\_rapport==0 \wedge formulaire\_medical\_a\_remplir==0 \wedge propositionDatesRDV==0 \wedge formulaire\_medical==0 \wedge confirmationRDV==0 \wedge demande\_examen==0 \wedge attestation==0 \wedge annulation==0 \wedge notification==0) \\ \hline \end{array}$$

4.4.5 Incompatibilité totale

L'hétérogénéité des services Web peut avoir un impact sur l'interopérabilité au sein d'une chorégraphie. Cet impact peut être partiel, comme présenté dans la section précédente, ou totale. Dans le cas d'un impact total, les services Web correspondant ne peuvent

mener correctement aucune conversation. Ainsi, quand l'ensemble de services Web n'est ni totalement ni partiellement compatible, nous disons qu'il est *totalement incompatible*.

Formellement, un ensemble de services est totalement incompatible, si tous les chemins des protocoles de conversations ne mènent pas à l'état final comme spécifié par la formule CTL suivante :

$$\boxed{\mathbf{AG}\ \neg\ P_1.final\ \wedge\ \ldots\ \wedge\ \mathbf{AG}\ \neg\ P_n.final} \tag{4.6}$$

Exemple 14 *SP, SM, et SEM sont dits complètement incompatibles si :*

$$\boxed{\boldsymbol{AG}\ \neg PS.final \wedge\ \boldsymbol{AG}\ \neg MS.final \wedge\ \boldsymbol{AG}\ \neg MES.final} \tag{4.7}$$

4.5 Implantation de l'analyse de compatibilité dans UPPAAL

Après avoir présenté les primitives formelles de compatibilité en CTL, dans cette section, nous présentons leur mise en œuvre dans UPPAAL.

4.5.1 Compatibilité totale parfaite

La propriété CTL 4.1 qui caractérise la classe de compatbilité totale et parfaite présentée dans la section 4.4.1 est implémentée dans UPPAAL sous la forme des deux requêtes spécifiées comme suit :

$$\boxed{\begin{array}{l}- \ \mathbf{A{<}{>}}\ P_1.final \text{ and } \ldots \text{ and } P_n.final \\ - \ P_1.final \text{ and } \ldots \text{ and } P_n.final \rightsquigarrow r_1 == 0 \text{ and } \ldots \text{ and } \\ \quad r_m == 0, \text{ tel que } r_i \in \{R_1, \ldots R_i, \ldots, R_n\}\end{array}} \tag{4.8}$$

Exemple 15 *Les services SP, SM, et SEM sont dits totalement et parfaitement compatibles si les formules suivantes sont satisfaites dans UPPAAL.*

- ***A<>*** $SP.final$ *and* $SM.final$ *and* $SEM.final$
- *SP.final and SM.final and SEM.final* $\rightsquigarrow$ *notification_disponibilite==0 and examen==0 and examen_permis==0 and carte_handicape==0 and examen_conduire==0 and formulaire==0 and demande_carte_handicape==0 and formulaire_permis==0 and demande_permis==0 and notification2==0 and notification1==0 and notification3==0 and demande_attestation1==0 and notification4==0 and demande_carte_hadicape==0 and demande_attestation_domicile==0 and demande_examen_medical==0 and formulaire_a_remplir==0 and demande_bourse==0 and notification6==0 and notification5==0 and envoyer_attestation==0 and envoiResultat==0 and demande_etat_civil==0 and envoi_rapport==0 and formulaire_medical_a_remplir==0 and propositionDatesRDV==0 and formulaire_medical==0 and confirmationRDV==0 and demande_examen==0 and attestation==0 and annulation==0 and notification==0*

4.5.2 Compatibilité totale non-parfaite

La formule CTL 4.2 qui caractérise la classe de compatibilité totale non-parfaite est implantée sous la forme des deux requêtes UPPAAL :

- **A<>** $P_1.final$ and ... and $P_n.final$
- **E<>**($P_1.final$ and ... and $P_n.final$ *imply* $r_1 > 0$ or ... or $r_n > 0$) tel que $r_i \in \{R_1, \ldots R_i, \ldots, R_n\}$

(4.9)

Exemple 16 *Les services SP, SM and SEM sont dits totalement mais non-parfaitement compatibles si les deux formules suivantes sont valides dans UPPAAL :*

- ***A<>****SP.final and SM.final and SEM.final*
- ***E<>****(SP.final and SM.final and SEM.final imply notification_disponibilite>0 or examen>0 or examen_permis>0 or carte_handicape>0 or examen_conduire>0 or formulaire>0 or demande_carte_handicape>0 or formulaire_permis>0 or demande_permis>0 or notification2>0 or notification1>0 or notification3>0 or demande_attestation1>0 or notification4==0 or demande_carte_hadicape>0 or demande_attestation_domicile>0 or demande_examen_medical>0 or formulaire_a_remplir>0 or demande_bourse>0 or notification6>0 or notification5>0 or envoyer_attestation>0 or envoiResultat>0 or demande_etat_civil>0 or envoi_rapport>0 or formulaire_medical_a_remplir>0 or propositionDatesRDV>0 or formulaire_medical>0 or confirmationRDV>0 or demande_examen>0 or attestation>0 or annulation>0 or notification>0*

4.5.3 Compatibilité partielle non-parfaite

Cette classe de compatibilité décrite dans la section 4.4.3 est implantée dans UPPAAL comme suit :

- **E**[] not $P_1.final$
- ...
- **E**[] not $P_n.final$
- **E<>**$P_1.final$ *and* ... *and* $P_n.final$
- **E<>** $P_1.final$ and ... and $P_n.final$ *imply* $r_1 > 0$ and ... and $r_m > 0$ where $r_i \in \{R_1, \ldots R_i, \ldots, R_n\}$

(4.10)

L'ensemble de services Web dont l'une des requête ***E[]*** *not* $P_i.final$ de la formule 4.10 est satisfaite ainsi que les deux dernières requête, sont dits partiellement mais non-parfaitement compatibles.

Exemple 17 *Les services SP, SM and SEM sont dits partiellement mais non-parfaitement compatibles si au moins une des trois premières requêtes est satisfaite et en même temps les deux dernières requêtes sont satisfaites :*

- ***E[]*** *not* $SP.final$
- ***E[]*** *not* $SM.final$
- ***E[]*** *not* $SEM.final$
- ***E<>*** $SP.final$ *and* $SM.final$ *and* $SEM.final$
- ***E<>*** $SP.final$ *and* $SM.final$ *and* $SEM.final$ *imply notification_disponibilite>0 or examen>0 or examen_permis>0 or carte_handicape>0 or examen_conduire>0 or formulaire>0 or demande_carte_handicape>0 or formulaire_permis>0 or demande_permis>0 or notification2>0 or notification1>0 or notification3>0 or demande_attestation1>0 or notification4==0 or demande_carte_hadicape>0 or demande_attestation_domicile>0 or demande_examen_medical>0 or formulaire_a_remplir>0 or demande_bourse>0 or notification6>0 or notification5>0 or envoyer_attestation>0 or envoiResultat>0 or demande_etat_civil>0 or envoi_rapport>0 or formulaire_medical_a_remplir>0 or propositionDatesRDV>0 or formulaire_medical>0 or confirmationRDV>0 or demande_examen>0 or attestation>0 or annulation>0 or notification>0*

4.5.4 Compatibilité partielle parfaite

Un ensemble de services Web qui n'est pas partiellement et non-parfaitement compatible et qu'en même temps leurs protocoles de conversations satisfassent au moins l'une des requêtes **E**[] not $P_i.final$ de la formule 4.11 ainsi que les deux dernières, sont dit partiellement et parfaitement compatibles.

- **E**[] not $P_1.final$
- ...
- **E**[] not $P_n.final$
- **E<>** $P_1.final$ *and* ... *and* $P_n.final$
- **E<>** $P_1.final$ and ... and $P_n.final$ *imply* $r_1 == 0$ and ... and $r_m == 0$ where $r_i \in \{R_1, \ldots R_i, \ldots, R_n\}$

(4.11)

Exemple 18 *Les services SP, SM and SEM sont dits partiellement et parfaitement compatibles s'il ne sont pas partiellement et non-parfaitement compatibles et qu'en même temps au moins l'une des trois premières requêtes est satisfaite ainsi que les deux dernières :*

- ***E[]*** $not\ SP.final$
- ***E[]*** $not\ SM.final$
- ***E[]*** $not\ SEM.final$
- ***E<>*** $SP.final\ and\ SM.final\ and\ SEM.final$
- ***E<>*** $SP.final$ *and* $SM.final$ *and* $SEM.final$ *imply notification_disponibilite==0 and examen==0 and examen_permis==0 and carte_handicape==0 and examen_conduire==0 and formulaire==0 and demande_carte_handicape==0 and formulaire_permis==0 and demande_permis==0 and notification2==0 and notification1==0 and notification3==0 and demande_attestation1==0 and notification4==0 and demande_carte_hadicape==0 and demande_attestation_domicile==0 and demande_examen_medical==0 and formulaire_a_remplir==0 and demande_bourse==0 and notification6==0 and notification5==0 and envoyer_attestation==0 and envoiResultat==0 and demande_etat_civil==0 and envoi_rapport==0 and formulaire_medical_a_remplir==0 and propositionDatesRDV==0 and formulaire_medical==0 and confirmationRDV==0 and demande_examen==0 and attestation==0 and annulation==0 and notification==0*

4.5.5 Incompatibilité totale

La formules 4.6 décrite dans la section 4.4.5 et qui définit la classe d'incompatibilité totale est mise en œuvre dans UPPAAL comme suit :

- **A**[] not $P_1.final$
- ...
- **A**[] not $P_n.final$

(4.12)

Exemple 19 *SP, SM, et SEM sont dits complètement incompatibles si les trois requêtes suivantes sont satisfaites :*

- $\boldsymbol{A}[]\ not\ PS.final$
- $\boldsymbol{A}[]\ not\ MS.final$
- $\boldsymbol{A}[]\ not\ MES.final$

(4.13)

En faisant le model checking sur notre exemple, les services Web du sénario présenté dans la section 1.3 sont caractérisés comme totalement incompatibles.

Pour récapituler, la figure 4.18 illustre l'architecture globale de l'analyseur de compatibilité de services Web que nous proposons. Le transformateur est un composant qui, en implémentant les étapes décrites ci-dessus, permet de transformer la description des protocoles de conversations pour produire des descriptions conformes à UPPAAL ayant une sémantique équivalente à celle des protocoles de conversations. En utilisant ces description, ils génère un ensemble de formules CTL. Pour résumer, ce composant produit deux éléments : (1) la description UPPAAL des protocoles de conversations temporisées et (2) un ensemble de formules CTL qui caractérisent les différentes classes que nous considérons. Ces deux derniers sont l'entrée du model checker UPPAAL, qui, permet de vérifier si les protocoles de conversations vérifient les formules CTL produites.

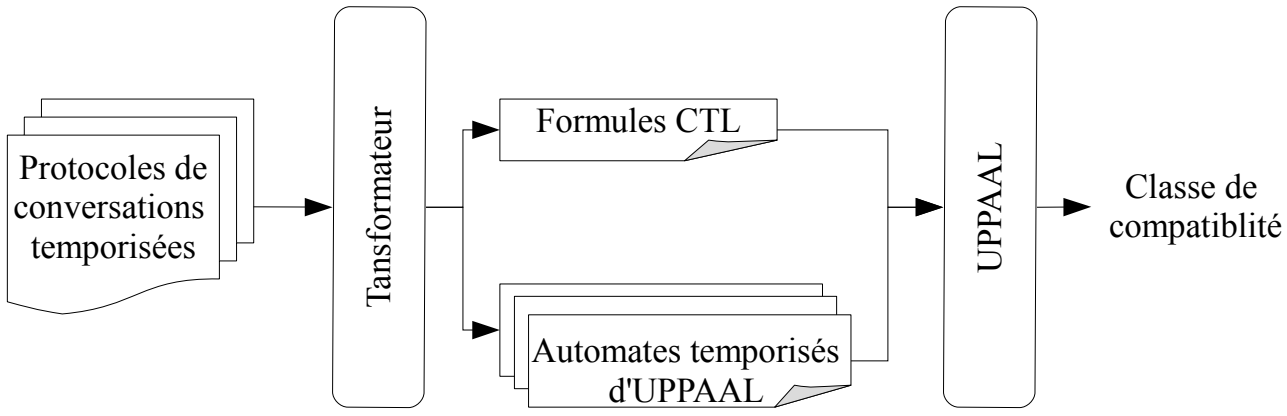


FIGURE 4.18 – Analyseur de compatibilité temporisée

4.6 Conclusion

Dans ce chapitre, nous avons présenté l'approche formelle d'analyse de compatibilité dans une chorégraphie. Cette approche repose sur une démarche basée sur le modèle checking. A la différence des approches existantes, l'approche que nous proposons tient en compte des *propriétés temporelles* et les *contraintes de données* dans des services Web *asynchrones*. Dans le cadre d'une chorégraphie, dès lors que des services collaborent ensemble, des conflits peuvent surgir. Afin de classer le type de collaboration que peuvent mener un ensemble de services Web, nous avons proposé une approche qui est basée sur le model checker UPPAAL.

Etant donné que nous considérons des *services asynchrones temporisés* munis de contraintes temporelles et de données, l'utilisation du modèle de UPPAAL tel quel est inadéquat. Pour mettre en œuvre l'analyse de compatibilité, nous avons proposé un processus de transformation, à savoir, l'abstraction des messages, des contraintes de données, l'association des invariants, du canal urgent, et la simulation de la notion des états finaux. En utilisant le résultat de ces abstractions, nous avons présenté un ensemble de formules CTL qui caractérisent les différentes classes de la chorégraphie de compatibilité. Par la suite, nous avons présenté comment ces formules CTL peuvent être implantées dans UPPAAL afin de les valider.

Il est à noter que les deux critères de compatibilité parfaite et non-parfaite sont utiles étant donné qu'en étendant ce travail par les aspects de sécurité, il est nécessaire de savoir l'enchaînement des messages.

Chapitre 5

Composition de services Web temporisés

Sommaire

5.1 Introduction 110

5.2 Le problème de composition temporisée 110

5.2.1 Nécessité de mécanismes de composition temporisée 111

5.2.2 Le rôle du médiateur 112

5.3 Les élements clés du cadre de composition 115

5.4 Démarche formelle de composition 116

5.4.1 Construction des connexions P2P temporisées 117

5.4.2 Expliciter les dépendances entre les contraintes temporelles . . 123

5.4.3 Génération du médiateur temporisé 129

5.5 Exemple récapitulatif 130

5.5.1 Composition sans l'intervention temporisée du médiateur . . . 131

5.5.2 Composition avec l'intervention temporisée du médiateur . . . 133

5.6 Conclusion 135

5.1 Introduction

Le concept de composition est l'un des concepts clé du paradigme SOC. La composition permet de créer de nouvelles fonctionnalités produisant un service à valeur ajoutée qui peut, lui aussi, contribuer à la création d'autres compositions.

Dans le chapitre précédent, nous avons présenté l'approche d'analyse de compatibilité de services Web temporisés qui supportent des communications asynchrones. L'approche proposée précédemment permet d'affirmer si un ensemble de services peuvent collaborer correctement, et ce en caractérisant le type de collaboration qu'ils peuvent mener (i.e., la classe de compatibilité). Dans le contexte de la composition, cette analyse permet de savoir à quel degré les services Web sont composables et ce via les différentes classes que nous avons définies. D'une manière générale, la classe de compatibilité totale reflète le fait que les services correspondants peuvent être composés correctement et pour cela, il suffit de créer des connexions entre les différents services. Quant à la compatibilité partielle, elle révèle que seulement un sous ensemble des traces des services Web sont composables et d'autres ne le sont pas. La dernière classe qui consiste en l'incompatibilité totale indique que l'ensemble des services Web n'est pas composables. Etant donnée la prolifération des services hétérogènes sur le Web, il est devenu primordial d'étudier des alternatives pour essayer de contourner ou de masquer les incompatibilités qui peuvent porter atteinte au succès de la composition.

Dans ce chapitre, nous adressons le problème de la composition de services Web. Dans un premier temps, nous présentons informellement le problème de composition de services Web supportant des communications asynchrones temporisées. Ensuite, nous exposons les éléments clé du cadre que nous proposons. Puis, nous fournissons les étapes et les primitives formelles de composition de services sous la forme d'algorithmes. Finalement, nous présentons un exemple récapitulatif.

5.2 Le problème de composition temporisée

Dans cette section, nous introduisons informellement le problème de composition temporisée de services Web. En général, ce problème est défini comme suit : étant donné un ensemble de services Web temporisés et une description qui spécifie le besoin du client, appelée *service client*, le problème consiste à construire une composition qui soit correcte et qui réponde au besoin du client. Une composition est dite *correcte* si elle ne contient aucun blocage et en même temps assure les fonctionnalités attendues par le client.

Comme mentionné précédemment, les éléments considérés dans la description des services Web jouent un rôle très important dans la composition de services. Particulièrement, nous avons discuté la nécessité de considérer les contraintes temporelles associés aux échanges asynchrones de messages qui impliquent des données. Dans la section 4.2 du chapitre 4, nous avons présenté les blocages dûs aux conflits temporisés et non temporisés que peut entraîner l'hétérogénéité des services Web. Ces problèmes peuvent porter atteinte à la composition de services Web. Une solution qui semble porteuse est de générer un service intermédiaire, appelé *médiateur*, dont le but est de masquer d'une manière transparente les conflits quand cela est possible. Ces concepts sont discutés dans la section suivante.

5.2.1 Nécessité de mécanismes de composition temporisée

La composition en général et la composition temporisée en particulier nécessite encore beaucoup d'investigations. Elle permet de répondre à des exigences et à des besoins qui nécessitent la collaboration de plusieurs services qui sont contraints de respecter mutuellement leurs contraintes temporelles définies de façons complètement indépendantes. Nous trouvons ce besoin dans plusieurs domaines, tels que :

1. *E-commerce* : Dans le e-commerce, pour satisfaire un besoin particulier, différents services peuvent être requis. Par exemple, pour un besoin d'achat de livres sur Internet, plusieurs services peuvent être impliqués, tels que le gestionnaire de stocks, le gestionnaire des commandes, la banque pour la vérification de la validité de la carte bancaire. Les services peuvent avoir leurs propres contraintes temporelles qui sont indépendantes les unes des autres et qui doivent être respectées pour assurer les fonctionnalités requises.
2. *E-healthcare* : Le e-healthcare nécessite évidement la collaboration de plusieurs services de différentes organisations, telles que les cliniques, les laboratoires d'analyses médicales, la sécurité sociale, la banque. Dans ce domaine, la notion de temps est très importante. Un médecin peut exiger des analyses dans des délais bien précis. De même, les analyses médicales ne peuvent être fournies qu'après un délai donné.

Dans ce chapitre, nous portons notre attention au problème de composition de services Web en présence de contraintes temporelles. Principalement, l'approche que nous considérons repose sur la génération d'un médiateur qui, quand c'est possible, permet de créer les connexions nécessaires pour accomplir correctement une composition.

Dans ce qui suit, nous présentons des exemples de situations dans lesquelles la génération de médiateur peut être intéressante.

5.2.2 Le rôle du médiateur

Dans ce qui suit, nous allons reprendre des problèmes discutés dans la section 4.2 et montrer des exemples dans lesquels la génération du médiateur peut masquer certains problèmes d'hétérogénéité.

Hétérogénéité non-temporisée

Prenons un exemple simple pour lequel le médiateur peut essayer de créer les connexions manquantes et nécessaires au succès de la composition. Comme on peut voir sur la figure 5.1, la conversation des deux services Q et Q' échoue quand Q attend le message $m_3(d_3)$ et le service Q' attend $m_4(d_4)$. En effet, Q peut commencer par envoyer $m_1(d_1)$ puis $m_2(d_2)$ puis attend $m_3(d_3)$. Quant au service Q', il commence par envoyer $m_5(d_3, d_6)$ puis attend $m_4(d_4)$. En essayant de créer des connexions, les deux services échouent. Pour que la conversation réussisse, il faut générer les deux messages manquants $m_3(d_3)$ et $m_4(d_4)$. Pour pouvoir produire ces deux messages, les données d_3 et d_4 sont requises. On peut remarquer que la donnée d_3 a été envoyée par le message $m_5(d_3, d_6)$. En d'autres termes, cette donnée est disponible et peut être réutilisée pour générer le message $m_3(d_3)$. Une fois que le médiateur génère ce message et l'envoie à Q, ce dernier envoie ensuite $m_4(d_4)$. Une fois envoyée, la donnée d_4 devient disponible. De la même manière, le médiateur génère le message $m_4(d_4)$ et l'envoie à Q'. De la même façon, le médiateur utilise les données d_1 et d_2 qui sont déjà envoyées pour générer le message $m_6(d_1, d_2)$.

Aspects temporisés

Comme le médiateur peut intervenir pour contourner un problème d'hétérogénéité non-temporisée, il peut aussi intervenir pour masquer une hétérogénéité temporisée. Dans cette section, nous montrons un exemple de ce genre de situation.

Comme on peut voir sur la Figure 5.2, le service Q peut commencer par envoyer le message $m_0(d_0, d_1)$ que Q' consomme. Puis Q' envoie $m_2(d_2, d_3)$ ensuite envoie $m_1(d_2)$ après 20 unités de temps à partir de l'envoi de $m_2(d_2, d_3)$. Q ne peut consommer $m_1(d_2)$ qu'après 20 unités de temps à partir de l'envoi de $m_2(d_2, d_3)$, et reste ensuite bloqué dans l'attente de $m_2(d_2, d_3)$. En effet, Q ne peut consommer le message $m_2(d_2, d_3)$ qu'après la

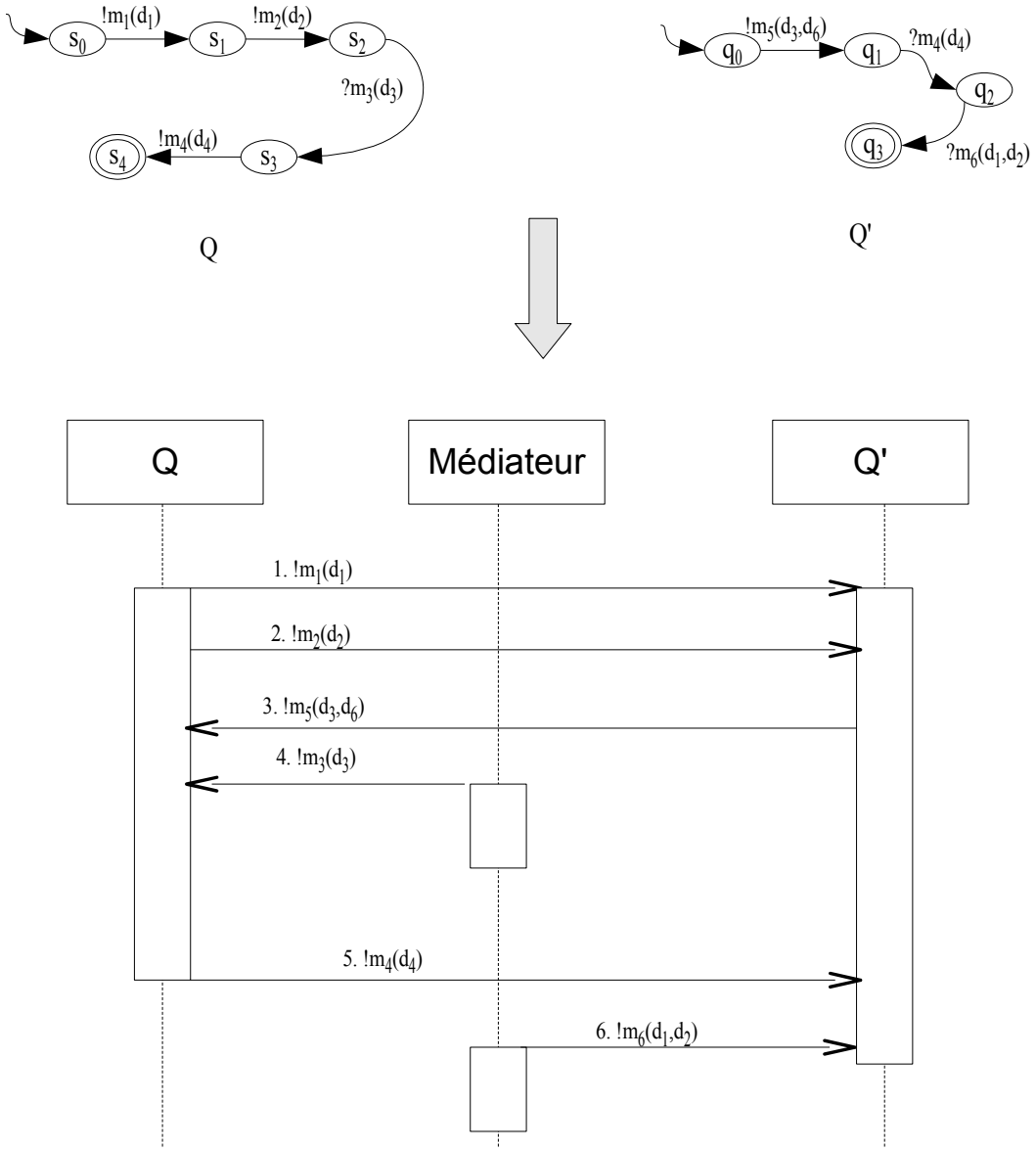


FIGURE 5.1 – Intervention du médiateur

consommation de $m_1(d_2)$, i.e., après 20 unités de temps mais en même temps et suivant sa propre contrainte, il doit le consommer dans les 10 unités de temps qui suivent l'envoi de $m_0(d_0, d_1)$. Donc si on récapitule, si on essaye de composer ces deux services, un conflit au niveau de la consommation du message $m_2(d_2, d_3)$ sera détecté et la composition échoue.

Cependant, pour essayer de réussir la composition, une alternative qui consiste à faire appel à un médiateur est envisageable. Si on examine la situation, quand le service Q' envoie $m_2(d_2, d_3)$, les données d_2 et d_3 deviennent disponibles. Alors le médiateur (voir Figure 5.2) peut utiliser ces données pour générer les messages $m_1(d_2)$ et $m_2(d_2, d_3)$ et de

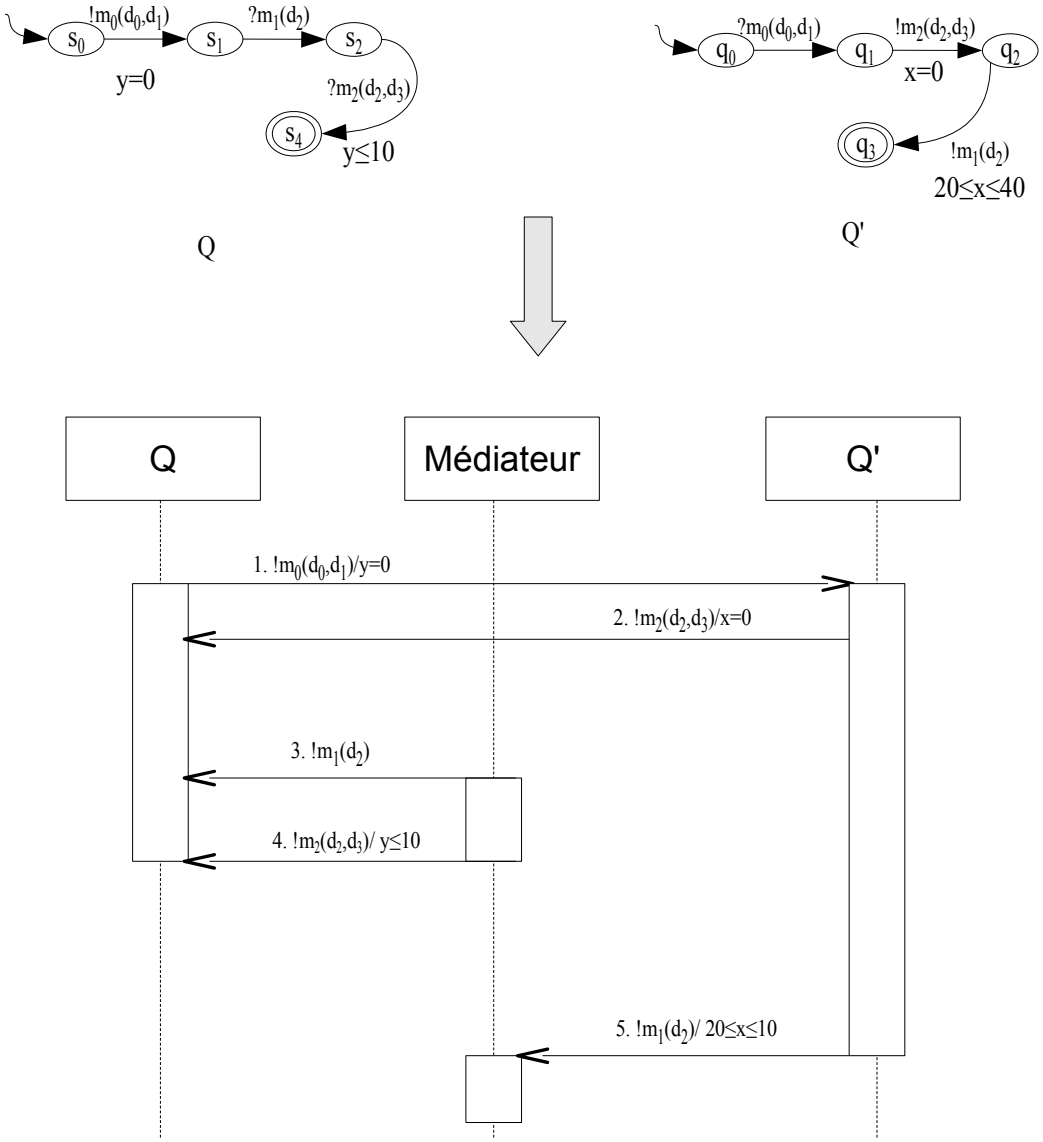


FIGURE 5.2 – Intervention du médiateur

cette manière, les connexions manquantes seront établies. A la fin, le médiateur consomme le message $m_1(d_2)$ et ce 20 unités de temps après l'envoi de $m_0(d_0, d_1)$.

Nous venons de montrer via un exemple simple, le rôle important que le médiateur joue dans la composition de services Web temporisée. A cause de certains conflits, les connexions peuvent échouer et par conséquent, la composition échoue. Par contre, avec l'intervention du médiateur, dans certains cas, les conflits peuvent être contournés et la composition réussit.

Dans ce qui suit, nous présentons les principaux éléments du cadre de composition que nous proposons.

5.3 Les élements clés du cadre de composition

Le cadre de composition que nous proposons est principalement constitué de deux élements. D'un coté, un ensemble de services Web et de l'autre coté, un service client, via lequel une description du besoin pour lequel on souhaite construire une composition est donnée.

- *Les services Web découverts* : Le travail proposé dans cette thèse repose sur l'hypothèse que des services Web sont déjà découverts. Ces services sont décrits par le biais de leurs protocoles de conversation temporisés. Comme décrit dans le chapitre 3, pour modéliser ces protocoles, nous adoptons les machines à états finis, dotées d'horloges telles que définies dans les automates temporisés [6].
- *Le service client* : Le deuxième élément de notre cadre de composition est la description du besoin du client. Nous supposons que pour chaque besoin, une description est fournie, appelée dans le reste de ce document *service client*. Le service client décrit le flux de données sans faire référence aux opérations des services Web découverts comme c'est le cas dans par exemple [24, 27, 66, 101, 7, 29, 10, 23, 26]. Le client spécifie les séquences de données qu'il fourni et les données qu'il requiert.
- *Service médiateur* : Comme cité auparavant, notre approche se base sur la génération d'un médiateur. Ce médiateur a accès aux données échangées durant les conversations de services Web qu'il médiatise pour les réutiliser dans la génération des messages requis. Pour cela, l'historique des données échangées est stocké. On note que nous supposons que les données qui ont le même nom ont la même valeur. Quand la valeur d'une donnée change, son nom change aussi.

Exemple 20 *Revenons à l'exemple introduit dans la section 1.3 du chapitre 1. Dans cet exemple, nous supposons que l'on souhaite construire une composition des trois services : service d'entité médicale, service de la préfecture, et service de la mairie pour essayer de construire un service complexe qui puisse répondre aux requêtes des demandeurs de bourses pour handicapés. Dans l'application que l'on considère, le service client, illustré sur la figure 5.3, permet à des handicapés d'interagir avec les différentes entités nécessaires pour l'attribution de bourse pour handicapés.*

Le demandeur envoie sa requête de demande de bourse en fournissant son numéro de sécurité social, son âge, et son type d'handicap via le message demande_bourse(ns,age,handicap) et ce si l'âge du demandeur est d'au moins 18 *ans. Ensuite, il reçoit soit une notification de rejet via le message requete_rejetée(motifRej), soit une proposition de contrôle*

médical via le message examen_med(motif) puis reçoit un formulaire médical à remplir via le message formulaire_med(formulaireMédical). Le demandeur remplit le formulaire et l'envoie via le message formulaire_médical(formulaireMedRempli). Suite à cela, le demandeur reçoit une proposition de date de Rendez-vous propos_DV(dates). Puis soit le demandeur annule la procédure en envoyant le message annuler_visite(motifA) ou précise une date de RDV en envoyant le message date_RDV(date) et ce après les 48 *heures qui suivent la réception du message propos_RDV(dates). Finalement, le demandeur doit recevoir la notification finale de la préfecture dans les* 96 *heures qui suivent sa demande de bourse, i.e., de l'envoi du message demande_bourse(ns,age,handicap).*

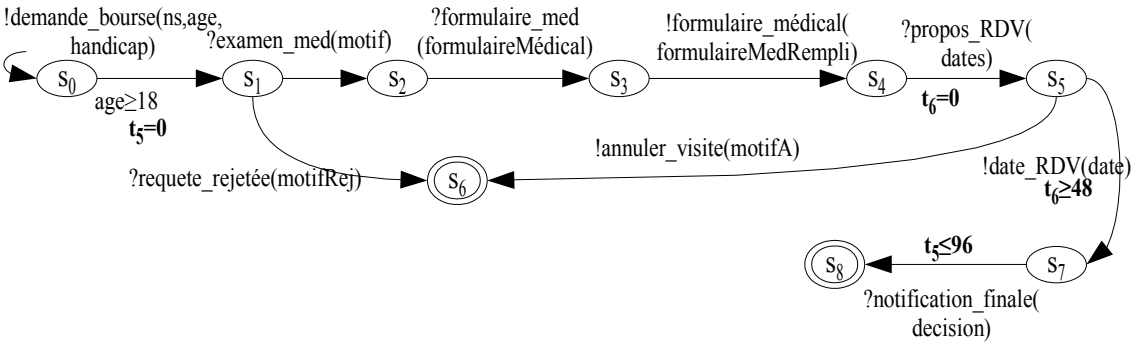


FIGURE 5.3 – Le service client.

Le premier problème auquel nous nous intéressons est comment composer les services Web de telle sorte qu'aucun conflit temporel ne surgisse. Le deuxième aspect que nous considérons est comment contourner les éventuels conflits pour essayer de réussir la composition en péril à cause de ces conflits. Dans la suite, nous présentons les primitives que nous proposons pour atteindre ces objectifs.

5.4 Démarche formelle de composition

Après avoir introduit le problème auquel nous nous intéressons, dans la suite, nous présentons la démarche sur laquelle notre approche se base. L'idée de base consiste tout d'abord à essayer de créer des connexions pair-à-pair (P2P) via la définition de canaux de communication entre les différents services. Une connexion peut être créée si aucun conflit (temporisé ou non-temporisé) ne surgit. En particulier, les conflits temporisés peuvent apparaître lors de la collaboration de services Web en propageant leur effet mutuellement. En effet, lors d'une conversation, les contraintes temporelles de chaque service peuvent

avoir un effet sur les autres services Web et créer des dépendances implicites entre eux. Ces dépendances peuvent donner lieu à des conflits temporisés. Pour pouvoir détecter ces problèmes lors de la construction de la composition, nous avons proposé le processus d'ordonnancement d'horloges (voir Section 5.4.2).

Etant donné que le flux des messages peut être interrompu à cause des conflits (temporisés et non-temporisés), alors les connections P2P peuvent échouer. C'est pour remédier à ce problème que nous proposons une approche basée sur la génération d'un médiateur. Le rôle du médiateur est d'essayer, quand c'est possible, de créer les connexions manquantes et requises pour accomplir correctement la composition.

Dans la suite, nous commençons par expliquer chaque étape pour montrer à la fin un exemple illustratif.

5.4.1 Construction des connexions P2P temporisées

Comme mentionné précédemment, le service client spécifie les séquences de données qu'il peut fournir et les séquences de données qu'il requiert. Pour cela, il spécifie les délais qu'il faut respecter sous forme de contraintes temporelles. Dans le but de satisfaire le service client, nous essayons tout d'abord de combiner le service client avec les services découverts. La combinaison consiste en la création de canaux de communication. Intuitivement, un *canal* définit le service expéditeur, le service récepteur et le message échangé qui transite via le canal. L'ensemble ordonné des canaux constitue ce que l'on appelle *un schéma de composition temporisé*. Pour récapituler, en ayant un service client et un ensemble de services découverts, notre but est de construire un automate global qui caractérise le schéma temporisé de composition. L'automate global est appelé *Automate du Schéma de Composition Temporisée (ASCT)*.

Pour construire le ASCT, nous introduisons le concept de *configuration* qui représente les états du ASCT à un instant donné. Une configuration définit l'évolution des états des services lors de leurs interactions. La configuration initiale spécifie que tous les services sont dans leur état initial.

Etant donnée une configuration source, le ASCT atteint une nouvelle configuration quand il existe deux services qui changent leur état en échangeant un message de telle sorte qu'aucun conflit ne surgisse. En particulier, les conflits temporisés apparaissent du fait que les différentes horloges utilisées pour spécifier des contraintes temporelles ne sont pas partagées. Par conséquent, lors de la combinaison des services, des dépendances implicites peuvent être créées ce qui peut donner lieu à des conflits temporisés. Le défi est de

pouvoir découvrir les conflits temporisés implicites lors de la construction de la composition (ce processus est présenté dans la section 5.4.2) et d'essayer de les contourner (voir section 5.4.3).

Définition 10 *(Automate de Schéma de Composition Temporisée ASCT)*
Un ASCT est défini par le tuple (S, Q, M, C, X, T) *tels que*

- S *est l'ensemble des configurations*
- Q *est l'ensemble des services*
- M *est l'ensemble des messages*
- C *est l'ensemble des contraintes de données*
- X *est l'ensemble des horloges*
- $T \subseteq S \times M \times C \times \Psi(X) \times S$ *est l'ensemble des transitions globales du ASCT. Une transition spécifie, qu'à partir d'une configuration source, le ASCT atteint une nouvelle configuration et ce en créant un canal d'échange de message qui interconnecte deux services de telle sorte que les contraintes de données et les contraintes temporelles sont satisfaites.*
- *Comme le médiateur peut être impliqué, alors l'ensemble des services* Q *est défini par* $\{R, P, Med\}$*, tel que* R *est le client service,* P *est l'ensemble des services disponibles, et* Med *est le médiateur.*

Parmi l'ensemble des transitions temporisées, nous distinguons les transitions dites *transitions passives* et celles dites *non-passives*.

- *Une transition passive* est une transition temporisée (ou non-temporisée) qui a des contraintes temporelles de la forme $x \leq v$ (resp. $x < v$). En effet, ces transitions sont considérées comme des transitions passives étant donné qu'elles ne génèrent pas de conflits si elles précédent des contraintes de la forme $x \leq v$ ou même des contraintes de la forme $x \geq v$.
- *Une transition non-passive* est une transition temporisée qui a une contrainte temporelle de la forme $x \geq v$ (resp. $x > v$). En effet, les conflits temporisés apparaissent principalement à cause des contraintes de la forme $x \geq v$ (resp. $x > v$) quand elles précèdent les transitions ayant des contraintes de la forme $x \leq v$ (resp. $x < v$).

L'approche de composition repose sur l'algorithme 5.1. Le principe de cet algorithme consiste à créer des connexions entre les différents services pour essayer de satisfaire le service client. Le but est d'essayer tout d'abord de créer des connexions entre le service

client et les différents services. Il est à noter que la connexion des transitions passives est prioritaire sur celle des transitions non-passives. Quand la connexion échoue, il fait appel à l'algorithme 5.4 pour générer le médiateur. Quand une connexion peut être créées, l'algorithme 5.3 intervient afin de vérifier si la nouvelle connexion ne fait pas apparaître un conflit temporisé.

Les différentes étapes que nous utilisons pour créer un ASCT, et qui correspondent à l'algorithme exponentiel 5.1, sont comme suit :

1. Pour chaque transition de chaque trace du service client,
2. À partir des états courants des services Web, on isole les transitions passives,
3. De même, on isole les transitions non-passives. En effet, cet isolement a pour but de définir un ordre dans le choix des transitions candidates pour la création du schéma de composition.
4. Si la transition courante du service client est une transition passive et si elle peut être satisfaite, i.e., connectée (voir Algorithme 5.2), alors,
5. Mettre à jour l'état courant du service client,
6. Sinon, si la connexion échoue, alors,
7. On vérifie parmi l'ensemble des transitions passives de l'ensemble des services Web si il y a une transition qu'on peut connecter,
8. Si aucune transition ne peut être connectée, alors,
9. On vérifie parmi l'ensemble des transitions non-passives de l'ensemble des services Web s'il y a une transition qu'on peut connecter,
10. Si aucune transition ne peut être connectée, alors,
11. La composition échoue.
12. Sinon, si la transition du service client est une transition non-passive, alors,
13. Avant d'essayer de connecter la transition courante du service client, nous vérifions parmi l'ensemble des transitions passives de l'ensemble des services Web, s'il y a une transition qu'on peut connecter
14. Si aucune transition ne peut être connectée, alors,
15. Si la transition non-passive du service client peut être connectée, alors,
16. On met à jour l'état courant du service client,
17. Sinon, si la connexion de la transition non-passive du service client échoue,

Algorithm 5.1: Composition

Entrées: Le service client $Q_g = (S_c, s_{0_c}, F_c, M_c, C_c, X_c, T_c)$, l'ensemble des services $Q_i = (S_i, s_{0_i}, F_i, M_i, C_i, X_i, T_i)$, pour $i = \{1, .., n\}$
, la configuration initiale de ASCT $\overline{s} = (s_{0_1}, \ldots, s_{0_n})$, l'état courant du service client $s_c = s_{0_c}$
Sorties: ASCT $= (S, s_0, F, M, C, X, T)$, et le médiateur $Med = (S_{med}, s_{0_{med}}, F_{med}, M_{med}, C_{med}, X_{med}, T_{med})$

début
  compositionReussie=vrai
1 **pour** *chaque transition* $t_c = (s_c, \alpha_c, c_c, \psi_c, Y_c, s'_c)$ *de chaque trace du service client et* ***si*** *compositionReussie* **faire**
2 $T_{sp} = \{t_i\}$, pour $i = \{1..n\}$ où $t_i = (s_i, \alpha_i, c_i, \psi_i, Y_i, s_{i+1})$ tel que $s_i \in (s_1 \ldots s_n)$ et $\psi_i = x \leq c$ ou $\psi_i = x < c$
3 $T_{snp} = \{t_j\}$, pour $j = \{1..n\}$ où $t_j = (s_j, \alpha_j, c_j, \psi_j, Y_j, s_{j+1})$ tel que $s_j \in (s_1 \ldots s_n)$ et $\psi_j = x \geq c$ ou $\psi_j = x > c$
   $T_{visited} = null$
4 **si** *((*$\psi_c = x < c$*) ou (*$\psi_c = x \leq c$*)) et (satisfaction(*t_c*,*$\overline{s}$*))* **alors**
5 $s_c = s'_c$
   sinon
6 **si** *((*$\psi_c = x < c$*) ou (*$\psi_c = x \leq c$*)) et (¬satisfaction(*t_c*,*$\overline{s}$*))* **alors**
      choisir une transition t_{sp} de T_{sp}
      $T_{visited} = T_{visited} \cup t_{sp}$
7 **tant que** $T_{sp} \nsubseteq T_{visited}$ *and* ¬ *satisfaction(*$t_{sp}, \overline{s}$*)* **faire**
        choisir une autre transition t_{sp} de T_{sp}
        $T_{visited} = T_{visited} \cup t_{sp}$
8 **si** ¬ *satisfaction(*t_{sp}*,*$\overline{s}$*)* **alors**
        Choisir une transition t_{snp} de T_{snp}
        $T_{visited} = T_{visited} \cup t_{snp}$
9 **tant que** $T_{snp} \nsubseteq T_{visited}$ *and* ¬ *satisfaction(*$T_{snp}, \overline{s}$*)* **faire**
          choisir une autre transition t_{snp} de T_{snp}
          $T_{visited} = T_{visited} \cup t_{snp}$
10 **si** ¬ *satisfaction(*t_{snp}*,*$\overline{s}$*)* **alors**
11 compositionReussie=faux
      sinon
12 **si** *(*$\psi_c = x > v$*)ou(*$\psi_c \geq v$*)* **alors**
        choisir une transition t_{sp} de T_{sp}
        $T_{visited} = T_{visited} \cup t_{sp}$
13 **tant que** $T_{sp} \neq \emptyset$ *and* $\neg satisfaction(T_{sp}, \overline{s})$ **faire**
          choisir une autre transition t_{sp} de T_{sp}
          $T_{visited} = T_{visited} \cup t_{sp}$
14 **si** ¬ *satisfaction(*t_{sp}*,*$\overline{s}$*)* **alors**
15 **si** *satisfaction(*$t_c, \overline{s}$*)* **alors**
16 $s_c := s'_c$
          sinon
17 choisir une transition t_{snp} de T_{snp}
              $T_{visited} = T_{visited} \cup t_{snp}$
18 **tant que** $T_{snp} \neq \emptyset$ *and* $\neg satisfaction(T_{snp}, \overline{s})$ **faire**
                choisir une autre transition t_{snp} de T_{snp}
                $T_{visited} = T_{visited} \cup t_{snp}$
19 **si** ¬ *satisfaction(*t_{snp}*,*$\overline{s}$*)* **alors**
20 compositionReussie=faux
21 **si** *compositionReussie* **alors**
22 **si** $\overline{s} \in F$ **alors**
23 Retourner ASCT et le médiateur
      sinon
24 'La composition échoue car les services n'atteignent pas leurs états finaux'
   sinon
    'La composition échoue car le service client ne peut pas être satisfait'
fin

18. On essaye de connecter une transition parmi les transitions non-passives des services Web,
19. Si la connexion échoue,
20. Alors la composition échoue.
21. Si la composition n'a pas échoué, alors,
22. Si tous les services atteignent leur état final, alors
23. On retourne le ASCT et le médiateur généré,
24. Sinon, la composition échoue

La fonction *satisfaction()* (voir l'algorithme 5.2) est utilisée pour créer des connexions P2P entre deux services. Cette fonction a deux entrées, une transition $(s, \alpha(\overline{d}), c, \psi, Y, s')$ à connecter et la configuration courante des services $\overline{s}$. Les différentes étapes et qui correspondent à l'algorithme 5.2 sont présentées comme suit :

1. On commence tout d'abord par vérifier si la transition d'entrée n'a pas été déjà visitée deux fois, alors
2. Nous vérifions si il y au moins un service qui a une transition passive $(s_j, \alpha_j(\overline{d}), c_j, \psi_j, Y_j, s'_j)$ telle que les contraintes de données c_i et c_j ne sont pas disjointes et les messages $\alpha_i(\overline{d})$ et $\alpha_j(\overline{d})$ sont égaux mais qui ont des polarités différentes [2]. Autrement dit, si la transition $(s_i, \alpha_i(\overline{d}), c_i, \psi_i, Y_i, s'_i)$ permet de déclencher un message entrant (resp. message sortant), la transition $(s_j, \alpha_j(\overline{d}), c_j, \psi_j, Y_j, s'_j)$ doit permettre d'envoyer (resp. recevoir) le message sortant (resp. entrant) correspondant.
3. Etant donné qu'un message peut être envoyé après un délai, quand on génère un médiateur pour le consommer, alors la consommation peut être faite après ce délai. Donc pour chaque remise à zéro d'horloges, Y_i et Y_j, on génère un médiateur vide qui permet de remettre à zéro les horloges (Y_i, Y_j). En effet, ces horloges peuvent être utilisées par le médiateur pour spécifier des contraintes temporelles associées à la consommation des messages.
4. Si un médiateur vide est généré $(s_{med}, \epsilon, Y_i, Y_j, s'_{med})$ pour remettre à zéro des horloges, alors nous construisons une transition ASCT candidate qui connecte deux transitions $(s_i, \alpha_i(\overline{d}), c_i, \psi_i, Y_i, s'_i)$ et $(s_j, \alpha_j(\overline{d}), c_j, \psi_j, Y_j, s'_j)$ avec la transition vide générée du médiateur $(s_{med}, \epsilon, Y_i, Y_j, s'_{med})$.

2. La polarité d'un message définie si un message est entrant ou sortant. $Polarity(m) =?$ si m est un message entrant. $Polarity(m) =!$ si m est un message sortant

Algorithm 5.2: Satisfaction

Entrées: Une transition $t_i = (s_i, \alpha_i(\overline{d}), c_i, \psi_i, Y_i, s'_i)$, une configuration $\overline{s}$
Sorties: boolean
P2PConnection($(s_i, \alpha_i(\overline{d}), c_i, \psi_i, Y_i, s'_i)$***,*** $\overline{s}$***)***
début
  echec=faux
1 **si** $\neg cycle(s_i, \alpha_i, c_i, \psi_i, Y_i, s'_i)$ **alors**
2 **si** *il existe* $t_j = (s_j, \alpha_j, c_j, \psi_j, Y_j, s'_j) \in T_{sp}$ *tel que* $\alpha_i(\overline{d}) = \overline{\alpha_j(\overline{d})}$ *and* $c_i \cap c_j \neq \emptyset$ **alors**
3 **si** $Y_i \neq \emptyset$ *ou* $Y_j \neq \emptyset$ **alors**
     $Y_{asct} = Y_i \cup Y_j$
     $T_{med} = T_{med} \cup (s_m, \epsilon, Y_{asct}, s'_m)$
     $\psi_{asct} = \psi_i \cup \psi_j$
4 $t_{asct} = (s_0 s_1 \dots s_i \dots s_j \dots s_n s_m, \alpha(\overline{d}), \psi_{asct}, Y_{asct}, s_0 s_1 \dots s'_i \dots s'_j s_n s'_m)$
    sinon
5 $t_{asct} = (s_0 s_1 \dots s_i \dots s_j \dots s_n s_m, \alpha(\overline{d}), \psi_i, \psi_j, Y_i, Y_j, s_0 s_1 \dots s'_i \dots s'_j s_n s_m)$
6 **si** $clock\_order(t_{asct})$ **alors**
7 $T_{asct} = T_{asct} \cup t_{asct}$
8 $\overline{s} = s_0 s_1 \dots s'_i \dots s'_j s_n s'_m$, retourner vrai
    sinon
9 $t_{med} = mediateur(t_i, D)$
     si $t_{med} \neq null$ **alors**
10 $t_{asct} = (s_0 s_1 \dots s_i \dots s_n s_m, \alpha(\overline{d}), \psi_i, Y_i, s_0 s_1 \dots s'_i \dots s_n s'_m)$
11 **si** $clock\_order(t_{asct})$ **alors**
12 $T_{asct} = T_{asct} \cup t_{asct}$
13 $\overline{s} = s_0 s_1 \dots s'_i \dots s_n s'_m$, retourner vrai
          sinon
           echec=vrai
     sinon
      echec=vrai
   sinon
    echec=vrai
14 **si** *echec* **alors**
    si *il existe* $t_j = (s_j, \alpha_j, c_j, \psi_j, Y_j, s'_j) \in T_{snp}$ *tel que* $\alpha_i(\overline{d}) = \overline{\alpha_j(\overline{d})}$ **alors**
     si $Y_i \neq \emptyset$ *ou* $Y_j \neq \emptyset$ **alors**
       $Y_{asct} = Y_i \cup Y_j$
       $T_{med} = T_{med} \cup (s_m, \epsilon, Y_{asct}, s'_m)$
       $\psi_{asct} = \psi_i \cup \psi_j$
       $t_{asct} = (s_0 s_1 \dots s_i \dots s_j \dots s_n s_m, \alpha(\overline{d}), \psi_{asct}, Y_{asct}, s_0 s_1 \dots s'_i \dots s'_j s_n s'_m)$
     sinon
      $t_{asct} = (s_0 s_1 \dots s_i \dots s_j \dots s_n s_m, \alpha(\overline{d}), \psi_i, \psi_j, Y_i, Y_j, s_0 s_1 \dots s'_i \dots s'_j s_n s_m)$
     si $clock\_order(t_{asct})$ **alors**
       $T_{asct} = T_{asct} \cup t_{asct}$
       $\overline{s} = s_0 s_1 \dots s'_i \dots s'_j s_n s'_m$, retourner vrai
     sinon
       $t_{med} = mediateur(t_i, D)$
       si $t_{med} \neq null$ **alors**
        $t_{asct} = (s_0 s_1 \dots s_i \dots s_n s_m, \alpha(\overline{d}), \psi_i, Y_i, s_0 s_1 \dots s'_i \dots s_n s'_m)$ **si**
        $clock\_order(t_{asct})$ **alors**
          $T_{asct} = T_{asct} \cup t_{asct}$
          $\overline{s} = s_0 s_1 \dots s'_i \dots s_n s'_m$, retourner vrai
        sinon
          retourner faux
       sinon
        retourner faux
    sinon
      $t_{med} = mediateur(t_i, D)$
      si $t_{med} \neq null$ **alors**
       $t_{asct} = (s_0 s_1 \dots s_i \dots s_n s_m, \alpha(\overline{d}), \psi_i, Y_i, s_0 s_1 \dots s'_i \dots s_n s'_m)$
       si $clock\_order(t_{asct})$ **alors**
         $T_{asct} = T_{asct} \cup t_{asct}$
         $s_{asct} = s_0 s_1 \dots s'_i \dots s_n s'_m$, retourner vrai
       sinon
        retourner faux
      sinon
15 retourner faux
fin

5. Sinon, si il y a pas de remise à zéro des horloges, le médiateur reste dans le même état. Nous construisons la transition ASCT candidate qui connecte les deux transitions $(s_i, \alpha_i(\overline{d}), c_i, \psi_i, Y_i, s'_i)$ et $(s_j, \alpha_j(\overline{d}), c_j, \psi_j, Y_j, s'_j)$.
6. Nous appliquons le processus d'ordonnancement d'horloges pour détecter les éventuels conflits temporels (Via la fonction *clock_order()* présentée dans l'algorithme 5.3). Si aucun conflit n'est détecté, alors,
7. La transition ASCT candidate construite est acceptée,
8. On met à jour la configuration courante des services Web,
9. Sinon si, lors de l'application du processus d'ordonnancement, des conflits surgissent, alors on fait appel au médiateur (voir l'algorithme 5.4),
10. Si le médiateur est généré, alors on génère une transition ASCT candidate qui connecte la transition courante avec celle du médiateur,
11. On applique le processus d'ordonnancement d'horloge. Si aucun conflit n'est détecté, alors
12. On accepte la transition candidate,
13. Puis, on met à jour la configuration courante des services,
14. Si, on échoue dans la connexion de la transition $(s_i, \alpha_i, c_i, \psi_i, Y_i, s'_i)$, alors on applique les même étapes, mais cette fois ci en exploitant des transitions non-passives des services Web,
15. Si en appliquant les étapes précédentes, la connexion échoue, alors retourner faux.

Dans la suite, nous présentons la démarche de détection des conflits temporels.

5.4.2 Expliciter les dépendances entre les contraintes temporelles

Lors de la création des transitions de ASCT, les contraintes de données associées à l'envoi et à la réception de messages peuvent être vérifiées directement. Cependant, pour assurer une composition correcte, les contraintes temporisées des services ne peuvent pas être vérifiées aussi simplement que les contraintes de données. En effet, les horloges utilisées pour définir les contraintes temporelles ne sont pas partagées, i.e., elles sont complètement indépendantes. Par conséquent, lors de la création des transitions ASCT, des dépendances implicites peuvent être crées. Dans ce cas, des conflits temporels peuvent

apparaître. Afin de découvrir les conflits temporisés lors de la combinaison de services, nous avons besoin de mécanismes qui permettent de rendre explicite les dépendances temporisées. Pour ce faire, nous proposons le processus d'*ordonnancement d'horloges*. L'idée de base consiste à définir un ordre partiel entre les différentes horloges des différents services et ce pour chaque transition du ASCT.

La définition de l'ordre entre les différents horloges des services Web est assurée par l'algorithme 5.3. En ayant comme entrée la transition candidate $(s_i, m_i(\overline{d}), c_i, \psi_i, Y_i, s'_i)$, cet algorithme permet d'affirmer que la nouvelle transition fait apparaître ou non un conflit temporel. Les étapes que nous utilisons pour la détection des conflits temporels lors de la création du ASCT sont comme suit :

1. Si l'état source de la transition d'entrée est un état initial, i.e., la transition candidate du ASCT est la première transition, alors
2. Retourner vrai,
3. Sinon, on propage toutes les contraintes temporelles de la forme $x > v$ (resp. $x \geq v$) de la transition précédente,
4. Une horloge z initialisée au niveau d'une transition prédécesseur a une valeur supérieure à une horloge y initialisée dans la transition courante. Alors, on définit l'ordre $y \leq z$,
5. En outre, nous propageons l'ordre du type $z_1 \leq \ldots \leq z_n$ défini au niveau des transitions prédécesseurs,
6. Egalement, si il existe une contrainte de la forme $x \leq v$ et en même temps, une horloge y est initialisée, alors on peut savoir que la différence entre les deux horloges x et y est toujours inférieure à v. A cet effet, nous définissons l'ordre $x - y \leq v$,
7. De même, on applique la même opération pour les contraintes de la forme $x \geq v$,
8. Si dans l'ensemble des contraintes et des ordres définis (resp. propagés), il existe deux contraintes non passives $x \geq v$ et $x' \geq v'$ et en même temps, il y a un ordre de la forme $x - x' \geq v$ alors
9. On définit l'ordre $x \geq v + v'$. En effet, cet ordre permet de prendre en considération le cumul des valeurs des horloges.
10. En appliquant ces étapes de définition et de propagation, on commence par tester si il existe un ordre de la forme $v \leq x_1 \leq \ldots \leq x_n \leq v'$ tel que $v' < v$. Dans ce cas, il existe un conflit.

Algorithm 5.3: Clock_Order

Entrées: Une transition $(s_i, m_i(\overline{d}), \psi_i, Y_i, s'_i)$
Sorties: boolean
début

```
1   si s_i est l'état initial alors
2       retourner vrai
    sinon
        pour chaque ϱ_{i-1} ∈ ψ_{i-1}, tel que ϱ_{i-1} = x ≥ v ou ϱ_{i-1} = x > v de
        (s_{i-1}, m_{i-1}(d), ψ_{i-1}, Y_{i-1}, s'_{i-1}) faire
3           ψ_i = ψ_i ∪ ϱ_{i-1}
            pour chaque y = 0 ∈ Y_i et z = 0 ∈ Y_{i-1} faire
4               ψ_i = ψ_i ∪ y ≤ z
            pour chaque z_1 ≤ z_2 ∈ ψ_{i-1} faire
5               ψ_i = ψ_i ∪ z_1 ≤ z_2
            pour chaque y ∈ Y_i et ϱ_i ∈ ψ_i faire
6               si ϱ_i = x ≤ v alors
                    ψ' = x − y ≤ v
                    ψ_i = ψ_i ∪ ψ'
                sinon
                    si ϱ_i = x ≥ v alors
7                       ψ' = x − y ≥ v
                        ψ_i = ψ_i ∪ ψ'
8           si ∃ϱ_i = x ≥ v and ϱ'_i = x' ≥ v' and ϱ''_i = x − x' ≥ v alors
9               ψ' = x ≥ v + v'   ψ = ψ ∪ ψ'
10      si ∃ v ≤ y_0 ≤ ... ≤ y_n ≤ v' ∈ ψ_i tel que v' < v alors
            retourner faux
        sinon
11          si ∃ x ≥ v' ∈ ψ_i et y ≤ v'' ∈ ψ_i et x − y ≤ v ∈ ψ_i tel que v' − v'' > v alors
                retourner faux
            sinon
12              si ∃ x ≤ v' ∈ ψ_i et y ≤ v'' ∈ ψ_i et x − y ≥ v ∈ ψ_i tel que v' ≥ v'' et v' < v alors
                    retourner faux
                sinon
13                  si ∃ x ≤ v' ∈ ψ_i et y ≥ v'' ∈ ψ_i et x − y ≥ v ∈ ψ_i tel que v' − v'' < v alors
                        retourner faux
                    sinon
                        retourner vrai
```

fin

11. Sinon, si il existe trois contraintes $x \geq v'$ et $y \leq v''$ et $x - y \leq v$ mais $v' - v'' > v$ (i.e., suivant les contraintes de $x \geq v'$ et $y \geq v''$ et si $v' - v'' > v$, alors la différence $x - y \leq v$ est violée), alors retourner faux
12. Sinon, si il existe trois contraintes $x \leq v'$, $y \leq v''$ et $x - y \geq v$ tel que $v' < v$ alors la contrainte $x - y \geq v$ est violée
13. Sinon, si il existe trois contraintes $x \leq v'$, $y \geq v''$, et $x - y \geq v$ tel que $v' - v'' < v$

alors la contrainte $x - y \geq v$ est violée.

14. Sinon, retourner vrai.

Dans ce qui suit, nous donnons des exemples d'application du processus d'ordonnancement d'horloges pour détecter les conflits temporels.

Exemple 21 *Dans cet exemple, on va essayer de combiner d'une manière asynchrone les deux services illustrés sur la Figure 5.4 tout en appliquant le processus d'ordonnancement. Comme le service Q peut envoyer le message $m_0(d_0, d_1)$ en initialisant l'horloge y, alors on crèe la transition globale $(s_0q_0, !m_0(d_0, d_1), y = 0, s_1q_0)$. Ensuite, on crée la transition globale $(s_1q_0, ?m_0(d_0, d_1), s_1q_1)$ qui spécifie que Q' consomme le message $m_0(d_0, d_1)$. A partir de l'état q_1, le service Q' envoie le message $m_2(d_3)$ et initialise l'horloge x. Etant donné que l'horloge y a été initialisée avant l'horloge x alors on peut définir l'ordre $0 \leq x \leq y$ qu'on associe à la transition globale $(s_1q_1, !m_2(d_3), 0 \leq x \leq y, s_1q_2)$. Une fois que le message $m_2(d_3)$ est envoyé, Q envoie le message $m_1(d_2)$ tel que $20 \leq x \leq 40$. En propageant l'ordre $0 \leq x \leq y$ précédemment défini, on crée la transition globale $(s_1q_2, !m_1(d_2), 0 \leq x \leq y \wedge 20 \leq x \leq 40, s_1q_3)$. Etant donné que le message $m_1(d_2)$ est déjà envoyé par Q', alors Q peut le consommer et donc on peut créer la transition globale $(s_1q_3, ?m_1(d_2), s_2q_3)$. Finalement, Q doit consommer le message $m_2(d_3)$ dans les 10 unités de temps suivant l'émission de $m_0(d_0, d_1)$. En propageant l'ordre $20 \leq x \leq y$, on construit la transition globale $(s_2q_3, ?m_2(d_3), 20 \leq x \leq y \leq 10, s_3q_3)$. Le conflit $20 \leq 10$ parait au niveau de l'ordre $20 \leq x \leq y \leq 10$.*

Dans l'exemple suivant, nous présentons un autre exemple de détection de conflit temporisé.

Exemple 22 *Via l'exemple présenté dans la figure 5.5, nous présentons un autre exemple de conflit temporel que l'on peut détecter via le processus d'ordonnancement d'horloges. On va essayer de combiner d'une manière synchrone les deux services. Le service Q' commence par envoyer le message $m_0(d_0)$ et initialise l'horloge y. En même temps, le service Q peut le consommer, i.e., les deux services peuvent se synchroniser sur le message $m_0(d_0)$. Donc, lors de la combinaison synchrone, on crée la transition globale $(s_0q_0, m_0(d_0), y = 0, s_1q_1)$. Ensuite, les deux services se synchronisent sur le message $m_1(d_1)$. Quand Q envoie $m_1(d_1)$, l'horloge x est initialisée et Q' doit le consommer dans les 10 unités de temps qui suivent l'envoi de $m_0(d_0)$. On peut remarquer que lorsque l'horloge x est initialisée, la valeur de l'horloge y est inférieure ou égale à 10. Donc, on peut définir l'ordre $y - x \leq 10$. D'un autre coté, comme l'horloge y est initialisée avant l'horloge x, alors on peut définir*

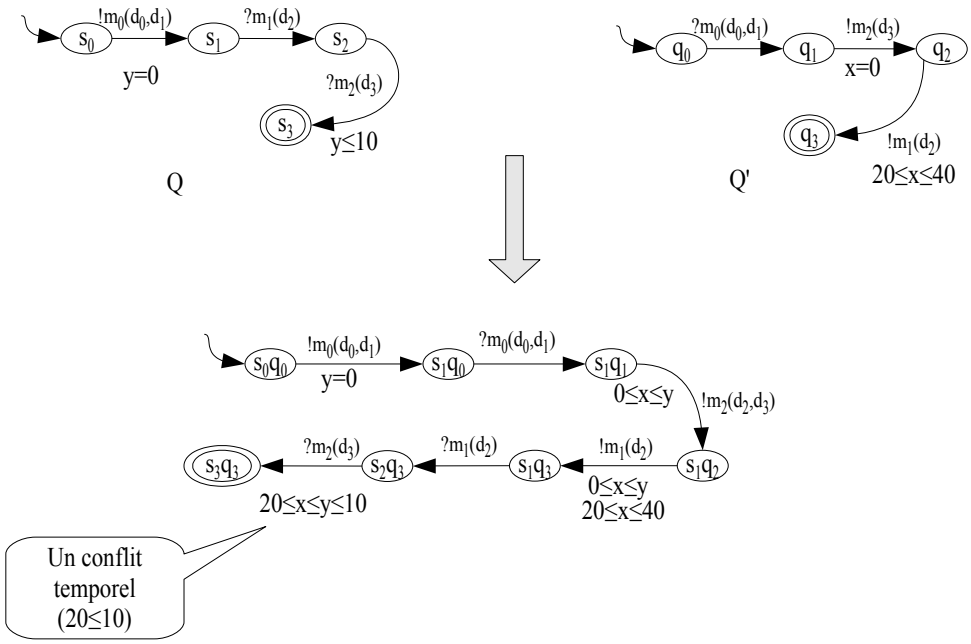


FIGURE 5.4 – Détection de conflits temporels via le processus d'ordannancement d'horloges

l'ordre $x \leq y$. *Avec ces nouvelles contraintes (ordres) définies, on crée la transition globale* $(s_1q_1, m_1(d_1), 0 \leq x \leq y \wedge y - x \leq 10 \wedge y \leq 10, s_2q_2)$. *Finalement,* Q' *doit envoyer le message* $m_2(d_1, d_0)$ *après au moins* 20 *unités de temps qui suivent l'envoi de* $m_0(d_0)$ *et en même temps,* Q *doit le consommer dans les* 5 *unités de temps qui suivent l'envoi de* $m_1(d_1)$. *En propageant l'ordre* $y - x \leq 10$ *et l'ordre* $0 \leq x \leq y$, *nous déduisons que les contraintes* $y \geq 20$, $x \leq 5$, *et* $x - y \leq 10$ *sont conflictuelles étant donné que* $20 - 5 > 10$ *(*$x - y > 10$ *viole la contrainte* $x - y \leq 10$*).*

Exemple 23 *Maintenant considérons les deux services* Q *et* Q' *illustrés sur la figure 5.6. Le service* Q *envoie le message* $m_0(d_0, d_1)$ *et initialise l'horloge* x. *Donc, on construit la transition* $(s_0q_0, !m_0(d_0, d_1), x = 0, s_1q_0)$. *Puis,* Q' *consomme le même message et initialise l'horloge* z. *Etant donné que l'horloge* z *est initialisée après l'horloge* x, *alors on définit l'ordre* $z \leq x$ *et on l'associe à la transition globale* $(s_1q_0, ?m_0(d_0, d_1), 0 \leq z \leq x, s_1q_1)$. *Une fois le message* $m_0(d_0, d_1)$ *est consommé,* Q *peut envoyer* $m_2(d_2, d_3)$. *Par conséquent, on crée la transition* $(s_1q_1, !m_2(d_2, d_3), s_2q_1)$. *De même,* Q' *peut envoyer le message* $m_1(d_2)$ *et ce après* 10 *unités de temps de la consommation de* $m_0(d_0, d_1)$, *alors en propageant l'ordre* $z \leq x$, *on crée la transition globales* $(s_2q_1, !m_1(d_2), 10 \leq z \leq x, s_2q_2)$. *De la même manière, on crée la transition globale* $(s_2q_2, ?m_2(d_2, d_3), s_2q_3)$. *Etant donné que le*

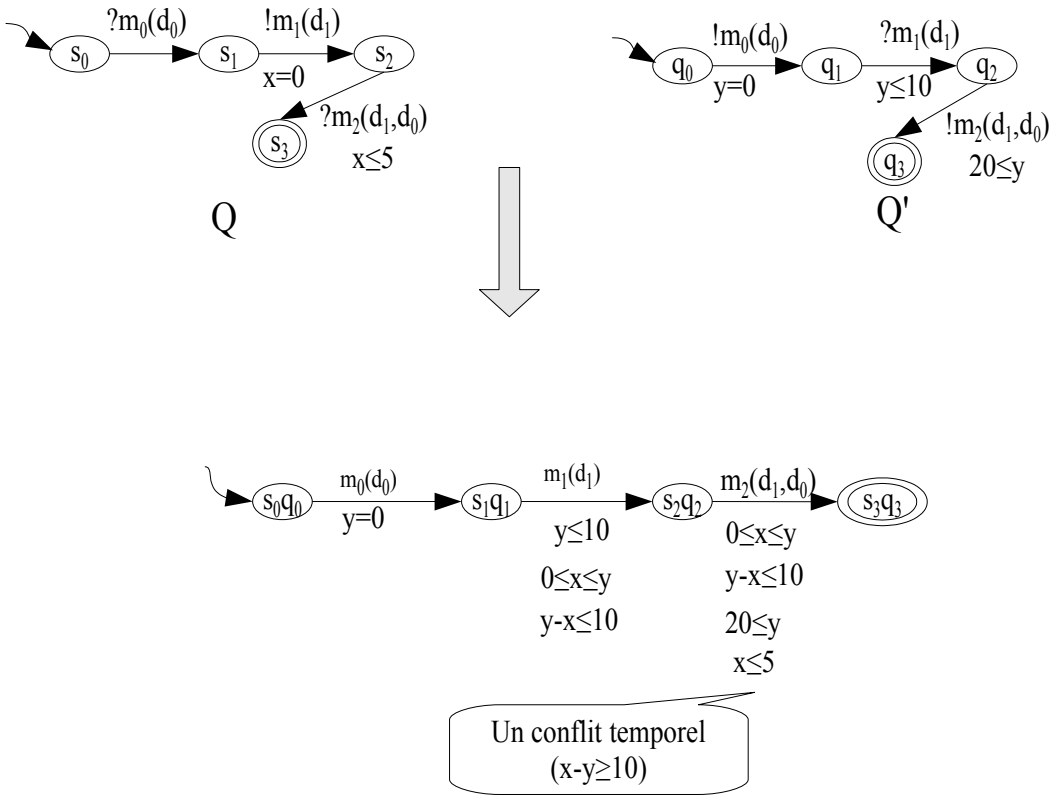


FIGURE 5.5 – Détection de conflits temporels via le processus d'ordonnancement d'horloges

message $m_1(d_2)$ *est déjà envoyé, alors le service* Q *peut le consommer et initialise l'horloge* y*. Comme l'horloge* y *est initialisée après les horloges* z *et* x *et en propageant l'ordre* $z \leq x$*, nous définissons l'odre* $0 \leq y \leq z \leq x$*. Egalement, nous propageons la contrainte* $10 \leq z$*. Etant donné qu'on a la contrainte* $z \geq 10$ *et* $y = 0$ *alors on définit la contrainte* $z - y \geq 10$*. Avec ceci, on crée la transition* $(s_2q_3, ?m_1(d_2), z - y \geq 10, 0 \leq y \leq z \leq x, s_3q_3)$*. Ensuite nous construisons la transition* $(s_3q_3, !m_4(d_5), s_3q_4)$*. Le service* Q *doit envoyer le message* $m_3(d_4)$ *après* 20 *unités de temps de la réception de* $m_1(d_2)$*. On propage l'ordre* $0 \leq y \leq z \leq x$ *et* $z - y \geq 10$ *précédemment défini. Etant donné qu'on a* $z \geq 10$*,* $y \geq 20$*, et* $z - y \geq 10$*, alors on définit* $z \geq 20 + 10$*. En propageons ce dernier ordre, nous créons la transition* $(s_3q_4, !m_3(d_4), 20 \leq y \leq z \leq x \wedge 30 \leq z, s_4q_4)$*. Une fois que le message* $m_3(d_4)$ *est envoyé,* Q *doit consommer* $m_4(d_5)$ *dans les* 25 *unités de temps qui suivent l'envoi de* $m_0(d_0, d_1)$*. De la même manière, on propage les contraintes* $20 \leq y \leq z \leq x$ *et* $30 \leq z$ *et on crée la transition* $(s_4q_4, ?m_4(d_5), 20 \leq y \leq z \leq x \leq 25 \wedge 30 \leq z, s_5q_4)$*. Les deux contraintes sont conflictuelles étant donné que* $30 \leq z \leq x \leq 25$*, i.e.,* $30 \leq 25$*.*

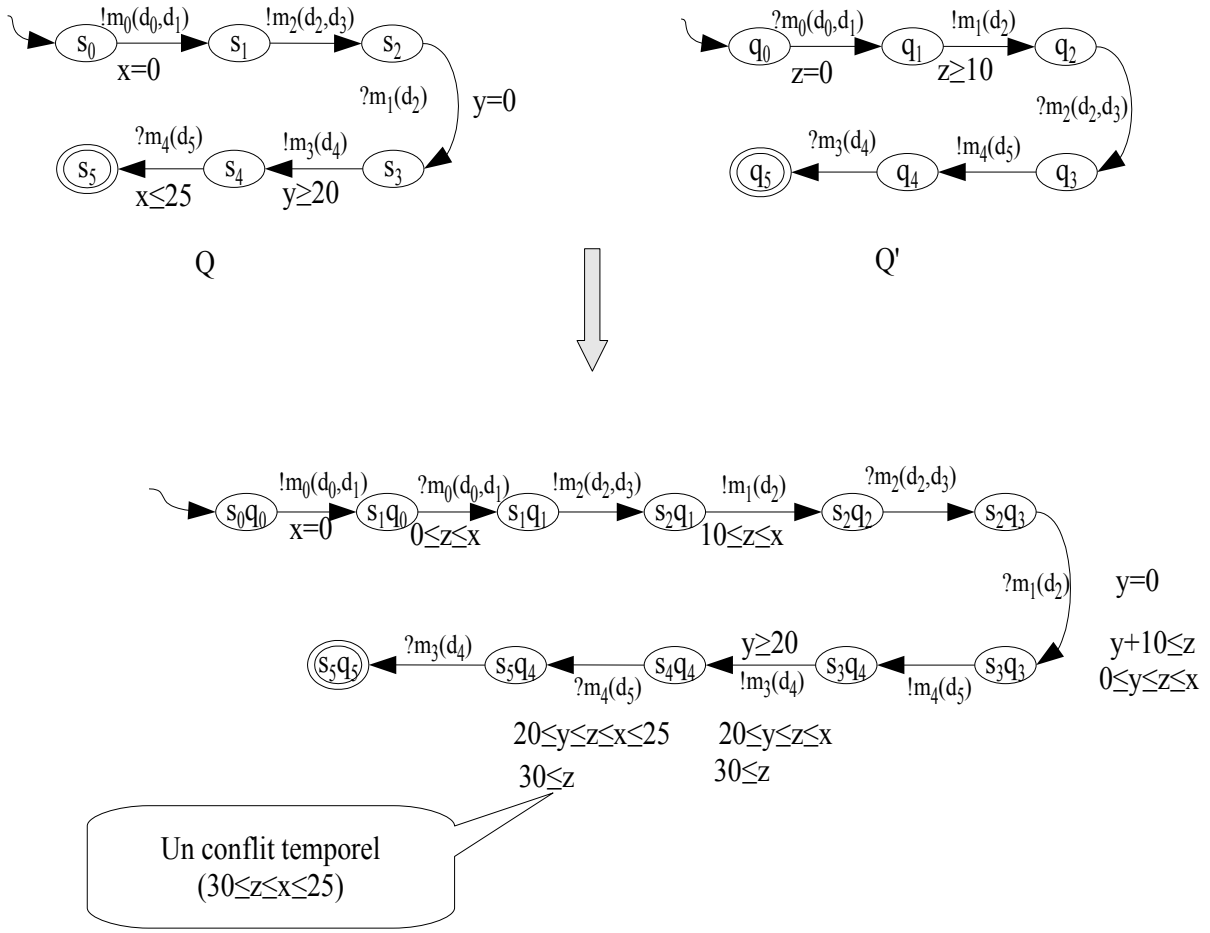


FIGURE 5.6 – Détection de conflits temporels via le processus d'ordannancement d'horloges

Dans les sections précédentes, nous avons présenté le mécanisme de création de connections entre les services Web ainsi que la découverte des conflits temporels. Dans la section suivante, nous présentons les primitives de génération du médiateur.

5.4.3 Génération du médiateur temporisé

Comme évoqué précédemment, les connexions P2P peuvent échouer. Le médiateur vise à créer les messages requis pour établir les connexions entre les services Web. Dans ce chapitre, nous proposons de générer les messages manquants en utilisant l'historique des données échangées précédemment, i.e., les données disponibles en se basant sur l'hypothèse que les données ayant les même noms ont les même valeurs.

Pour produire un message requis, nous vérifions si les données impliquées sont dis-

ponibles, i.e., elles ont été préalablement échangées. En d'autres termes, le médiateur réutilise l'historique des données pour produire les messages requis.

Le médiateur est défini en utilisant le ASCT calculé, et ce en ajoutant des messages entrants, sortants et des messages vides.

Tant que le ASCT peut être exécuté, le médiateur ne fait rien. Quand deux services sont connectés et qu'au moins un service permet d'initialiser une horloge, le médiateur génère un message vide et remet à zéro les mêmes horloges. Les transitions vides permettent de remettre à zéro les horloges qui peuvent être utilisées ultérieurement lors de la consommation d'un message. En effet, quand un service produit un message après un certain délai et que ce message ne peut pas être consommé par un autre service, i.e., c'est un message supplémentaire, le médiateur peut le consommer après ce délai. Tandis que, quand un service attend un message qui ne peut pas être envoyé par un autre service (i.e., il y a un message manquant), alors, si c'est possible, le médiateur génère le message manquant et ce en utilisant les données déjà disponibles. L'algorithme de génération du médiateur a comme entrées une transition $(s_i, \alpha, c_i, \psi_i, Y_i, s'_i)$ pour laquelle le médiateur est appelé, l'ensemble des données précédemment échangées D_a, et la configuration initiale des services $\overline{s}$. Les étapes que nous utilisons sont :

1. Si le médiateur doit être généré pour consommer un message supplémentaire envoyé lors du déclenchement de la transition $(s_i, !m(\overline{d}), c_i, \psi_i, Y_i, s'_i)$, alors
2. Nous générons la transition du médiateur $(s_m, ?m(\overline{d}), c_i, \psi_i, Y_i, s'_m)$ pour consommer le message $m(\overline{d})$
3. Si le médiateur doit être généré pour produire un message manquant, alors
4. Si les données impliquées dans le message manquant sont disponibles (i.e., elles ont été déjà échangées), alors
5. Le médiateur génère le message manquant,
6. Sinon, si au moins il y a une donnée manquante qui n'a pas été déjà échangée, alors le médiateur ne peut pas générer le message manquant.
7. Retourner null

5.5 Exemple récapitulatif

Dans le but de montrer via un exemple simple l'application de notre approche, nous utilisons le scénario introduit dans la section 1.3du chapitre 1. Tout d'abord, on va essayer

Algorithm 5.4: Génération de médiateur

Entrées: Une transition $t_i = (s_i, \alpha(\overline{d}), c, \psi, Y, s'_i)$, l'ensemble des données préalablement échangées D_a
Sorties: Une transition
mediator$((s_i, \alpha_i, c_i, \psi_i, Y_i, s'_i), D_a, \overline{s})$
début

1 **si** $\alpha = !m(\overline{d_m})$ **alors**
  $M_{med} := M_{med} \cup m(\overline{d})$
  $S_{med} := S_{med} \cup s'_m$
2 $T_{med} := T_{med} \cup (s_m, ?m(\overline{d}), c_i, \psi_i, Y_i, s'_m)$
  retourner $(s_m, ?m(\overline{d}), c_i, \psi_i, Y_i, s'_m)$;
sinon
3 **si** $\alpha = ?m(\overline{d_m}))$ **alors**
4 **si** $\overline{d_m} \subseteq D_a$ **alors**
  $M_{med} := M_{med} \cup m(\overline{d_m})$
  $S_{med} := S_{med} \cup s'_m$
5 $T_{med} := T_{med} \cup (s_m, !m(\overline{d_m}), c_i, Y_i, s'_m)$
  retourner $(s_m, ?m(\overline{d}), c_i, Y_i, s'_m)$;
sinon
6 Les données requises ne sont pas toutes disponibles, alors le message manquant ne peut pas être généré
7 retourner null

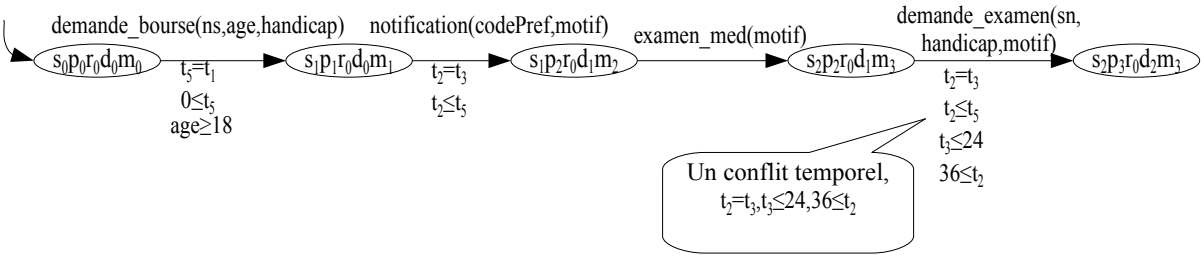
fin

de créer un ASCT sans l'intervention temporisée du médiateur.

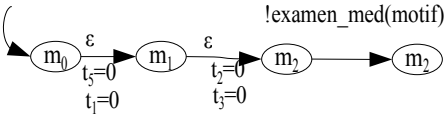
5.5.1 Composition sans l'intervention temporisée du médiateur

Etant donné que le médiateur peut être généré, alors initialement, on génère un médiateur vide qui a comme état initial m_0.

Le processus qui génère le ASCT (Figure 5.7.a) et le médiateur associé (Figure 5.7.b), sans l'intervention temporisée du médiateur, peut être illustré comme suit. La configuration initiale du ASCT est $s_0p_0r_0d_0m_0$. L'état courant du service client est s_0. A partir de cet état, on peut déclencher la transition *demande_bourse(ns,age,handicap)*. On vérifie si parmi les transitions des services, il existe une transition qui permet de recevoir le même message. On peut remarquer que le service SP attend le même message. Etant donné que les contraintes de données ne sont pas disjointes, alors on peut connecter les deux transitions $(s_0, !demande\_bourse(ns, age, handicap), age \geq 18, t_5 = 0, s_1)$ et $(p_0, ?demande\_bourse(ns, age, handicap), age > 16, t_1 = 0, p_1)$. Lors du déclenchement des deux transitions, les horloges t_1 et t_5 sont initialisées. Donc, on génère une transition vide du médiateur qui permet d'initialiser ces deux horloges. En effet, cette initialisation



a)- Un ASCT conflictuel



b)- Le médiateur associé

FIGURE 5.7 – Génération de ASCT sans l'intervention temporisée de médiateur

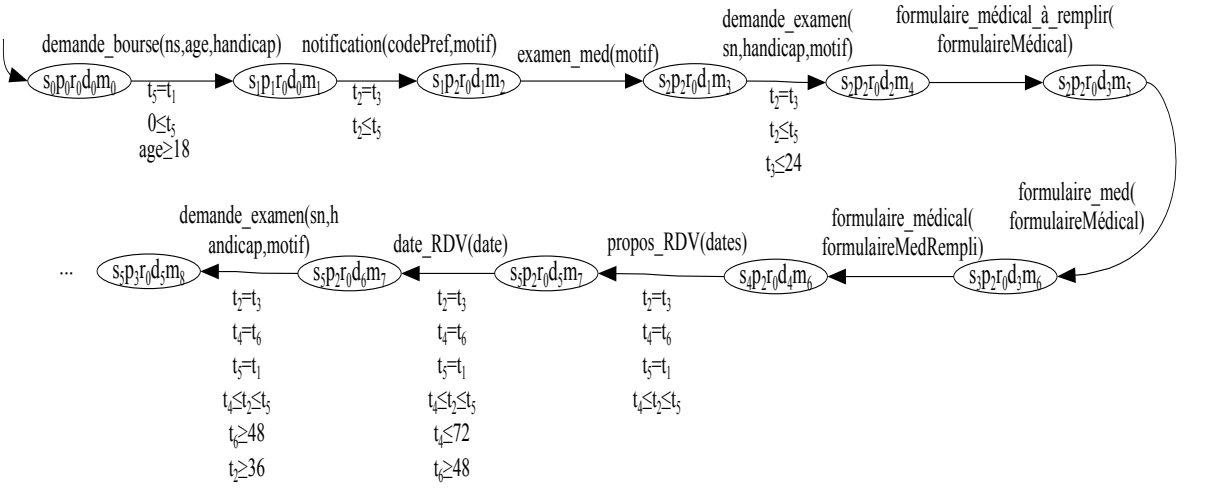
sert ultérieurement à utiliser ces horloges pour spécifier des contraintes quand c'est nécessaire. On crée la transition globale qui connecte les deux transitions des deux services ainsi que la transition vide du médiateur $(s_0p_0r_0d_0m_0, demande\_bourse(ns, age, handicap), age \geq 18, age > 16, 0 = t_5 = t_1, s_1p_1r_0d_0m_1)$. La nouvelle configuration devient $s_1p_1r_0d_0m_1$ et le nouvel état du service client devient s_1. A partir de la nouvelle configuration, la transition courante du service client devient $(s_1, ?examen\_med(motif), s_2)$. Aucune transition permettant l'envoi du message $examen\_med(motif)$ n'existe. Alors, on vérifie si la génération du médiateur peut produire ce message. Etant donné que la donnée *motif* n'a pas été déjà calculée, alors la génération du message *examen_ med(motif)* échoue. De ce fait, on teste les transitions des services. Parmi l'ensemble des transitions qui peuvent être déclenchées, on a la transition $(p_1, !notification(codePref, motif), t_2 = 0, p_2)$. Etant donné qu'aucune transition permettant la réception de ce message n'existe, alors on génère la transition $(m_1, ?notification(codePred, motif), t_2 = 0, m_2)$ du médiateur pour consommer ce message supplémentaire. Une fois la transition du médiateur générée, on génère la transition globale ASCT $(s_1p_1r_0d_0m_1, notification(codePref, motif), t_2 \leq t_5, t_5 = t_1, s_1p_2r_0d_1m_2)$. La nouvelle configuration devient $s_1p_2r_0d_1m_2$. A partir de cette configuration, le service client peut déclencher la transition $(s_2, ?formulaire\_med(formulaireMédical), s_3)$. Comme il n'existe aucune transition permettant d'envoyer le message *formulaire_ med(formulaireMédical)*, alors on vérifie si on peut connecter une transition des services Web. A

partir de la configuration courante, le service SEM attend le message *demande_examen(sn, handicap,motif)* et doit le consommer dans les 24 heures qui suivent la réception du message *notification(codePref,motif)*. Le message *demande_examen(sn,handicap,motif)* peut être envoyé par le service SP après 36 unités de temps de l'envoi du message *notification(codePref,motif)*. En propageant l'ordre précédemment défini, on construit la transition globale ASCT $(s_1p_2r_0d_1m_3, demande\_examen(sn, handicap, motif), t_2 = t_3, t_2 \leq t_5, t_3 \leq 24, t_2 \geq 36, s_2p_3r_0d_2m_3)$. Cette dernière transition est conflictuelle, étant donné que $t_2 = t_3$, $t_3 \leq 24$ et $t_2 \geq 36$. On voit bien avec l'intervention non-temporisée du médiateur pour produire (resp. consommer) les messages manquants (resp. supplémentaires), un conlit temporel apparait et la composition échoue.

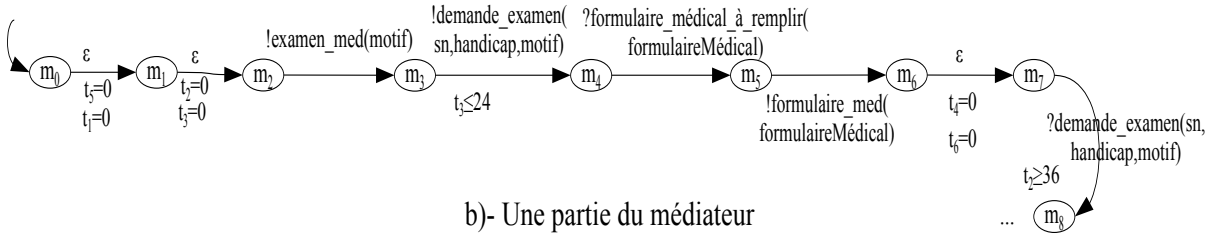
Dans la section suivante, on va construire une composition et ce en appliquant notre algorithme qui fait appel à l'intervention temporisée du médiateur.

5.5.2 Composition avec l'intervention temporisée du médiateur

Pour générer le ASCT (voir Figure 5.8.a) ainsi que le médiateur temporisé associé (voir Figure 5.8.b), nous suivons les étapes suivante. Tout d'abord, nous appliquons les mêmes étapes que celles décrites dans la section 5.5.1 jusqu'à atteindre la configuration $s_2p_2r_0d_1m_3$. A partir de cette configuration, le service SEM peut déclencher la transition passive $(d_1, ?demande\_examen(sn, handicap, motif), t_3 \leq 24, d_2)$. Etant donnée que la transition du service SP $(p_2, !demande\_examen(sn, handicap, motif), t_2 \geq 36, p_3)$ est une transition non-passive, alors on vérifie si le médiateur peut générer le message $demande\_examen(sn, handicap, motif)$. Comme on peut remarquer, les données sn, handicap, et $motif$ ont été précédemment échangées. Alors le médiateur peut générer le message requis $demande\_examen(sn, handicap, motif)$ via la transition $(m_3, !demande\_examen(sn, handicap, motif), m_4)$. En générant le message requis, on construit la transition globale $(s_2p_2r_0d_1m_3, demande\_examen(sn, handicap, motif), t_2 = t_3, t_2 \leq t_5, t_3 \leq 24, s_2p_2r_0d_2m_4)$. A partir de la nouvelle configuration $s_2p_2r_0d_2m_4$, on choisi la transition passive $(d_2, !formulaire\_médical\_à\_remplir(formulaireMédical), d_3)$ du service SEM. Aucune transition passive ne peut être connectée avec cette transition. Donc, on génère le médiateur qui consomme le message *formulaire_médical_à_remplir(formulaireMédical)* via la transition $(m_4, ?formulaire\_médical\_à\_remplir(formulaireMédical), m_5)$, puis on la connecte avec la transition $(d_2, !formulaire\_médical\_à\_remplir(formulaireMédical), d_3)$ pour créer la transition globale $(s_2p_2r_0d_2m_4, formulaire\_médical\_à\_remplir(formulaireMédical), s_2p_2r_0d_3m_5)$. La configuration courante devient $s_2p_2r_0d_3m_5$. A partir de



a)- Une partie de ASCT généré



b)- Une partie du médiateur associé

FIGURE 5.8 – Génération de ASCT avec l'intervention temporisée du médiateur

cette configuration, le service client peut déclencher la transition $(s_2, ?formulaire\_med(formulaireMédical), s_3)$. Comme cette transition ne peut pas être connectée avec une autre transition des services Web, alors on vérifie si le médiateur peut générer le message $formulaire\_med(formulaireMédical)$. La donnée $formulaireMédical$ a déjà été envoyée par le service SEM via la transition globale $(s_2p_2r_0d_2m_4, formulaire\_médical\_à\_remplir(formulaireMédical), s_2p_2r_0d_3m_5)$. Donc, le médiateur peut générer le message manquant $formulaire\_med(formulaireMédical)$ et ce en générant la transition $(m_5, !formulaire\_med(formulaireMédical), m_6)$. Une fois la transition du médiateur générée, on la connecte avec la transition du service client en créant la transition ASCT $(s_2p_2r_0d_3m_5, formulaire\_med(formulaireMédical), s_3p_2r_0d_3m_6)$. De la même manière, on connecte les deux services SEM et le service client en créant les transitions globales ASCT $(s_3p_2r_0d_3m_6, formulaire\_médical(formulaireMedRempli), s_4p_2r_0d_4m_6)$ et

$(s_4p_2r_0d_4m_6, propos\_RDV(dates), t_2 = t_3, t_4 = t_6, t_5 = t_1, 0 \leq t_4 \leq t_2 \leq t_5, s_5p_2r_0d_5m_7)$. A partir de la nouvelle configuration $s_5p_2r_0d_5m_7$, la transition courante du service client devient $(s_5, !date\_RDV(date), t_6 \geq 48, s_7)$. Cette transition est une transition non-passive, alors avant d'essayer de la connecter, on va vérifier si il y a des transitions passives des services que l'on peut connecter. Comme il n'existe aucune transition passive qui peut être déclenchée, alors on essaye de connecter la transition courante $(s_5, !date\_RDV(date), t_6 \geq 48, s_7)$ du service client. Cette transition peut être connectée avec la transition $(d_4, ?date\_RDV(date), t_4 = 0, d_5)$. En propageant l'ordre précédemment défini, on construit la transition $(s_4p_2r_0d_4m_6, propos\_RDV(dates), t_2 = t_3, t_4 = t_6, t_5 = t_1, 0 \leq t_4 \leq t_2 \leq t_5, s_5p_2r_0d_5m_7)$.

On vient de montrer que, dans certains cas, l'intervention temporisée du médiateur peut contourner des conflits temporels. Par exemple le conflit décrit dans la section 5.5.1 a pu être contourné via l'intervention temporisée du médiateur, permettant ainsi d'éviter l'échec de la composition.

En appliquant les même étapes, soit nous construisons un automate global qui caractérise le schéma de composition temporisé, soit on détecte un conflit qui ne peut pas être contourné.

5.6 Conclusion

Dans ce chapitre, nous avons présenté une approche formelle de composition de services Web asynchrones et temporisés. Afin de construire une composition, nous commençons tout d'abord par créer des connexions P2P entre les différents services. Etant donné que les contraintes temporelles des services Web sont locales et mutuellement indépendantes, des conflits peuvent apparaître. Dans le but de détecter ces conflits lors de la création de la composition, nous avons proposé le processus d'*ordonnancement d'horloges*.

A cause de la nature hétérogène des services Web, les connexions P2P peuvent échouer et par conséquent, la composition échoue. Pour essayer de résoudre ce problème, nous avons proposé une approche basée sur la génération d'un médiateur. Son rôle est de consommer les messages supplémentaires (i.e., les messages produits et qui ne sont pas consommés) et de générer les messages via lesquels, le médiateur peut contourner les éventuels conflits. Nous avons présenté des algorithmes pour mettre en œuvre les différents aspects et des exemples de fonctionnement.

Chapitre 6

Validation et mise en œuvre

Sommaire

6.1 Introduction 137

6.2 Architecture 138

6.2.1 Le vérificateur de la compatibilité temporisée asynchrone d'une chorégraphie 138

6.2.2 Composition de services temporisée 139

6.3 Les détails d'implantation 140

6.3.1 Représentation des protocoles de conversations temporisés . . . 141

6.3.2 Implantation du vérificateur 142

6.3.3 Implantation de la composition 145

6.4 Conclusion 148

6.1 Introduction

Dans le but de tester et de valider notre travail, nous avons implanté les primitives décrites dans les chapitres précédents. Dans ce chapitre, nous illustrons les différents aspects de la mise en œuvre. Dans la section 6.2, nous présentons les éléments conceptuels de l'architecture proposée. La section 6.3 expose les détails techniques de la mise en œuvre des différents éléments de l'approche proposée.

6.2 Architecture

Notre plateforme, illustrée sur la Figure 6.1, est constituée de deux éléments : (1) *le vérificateur*, et (2) *le composeur*.

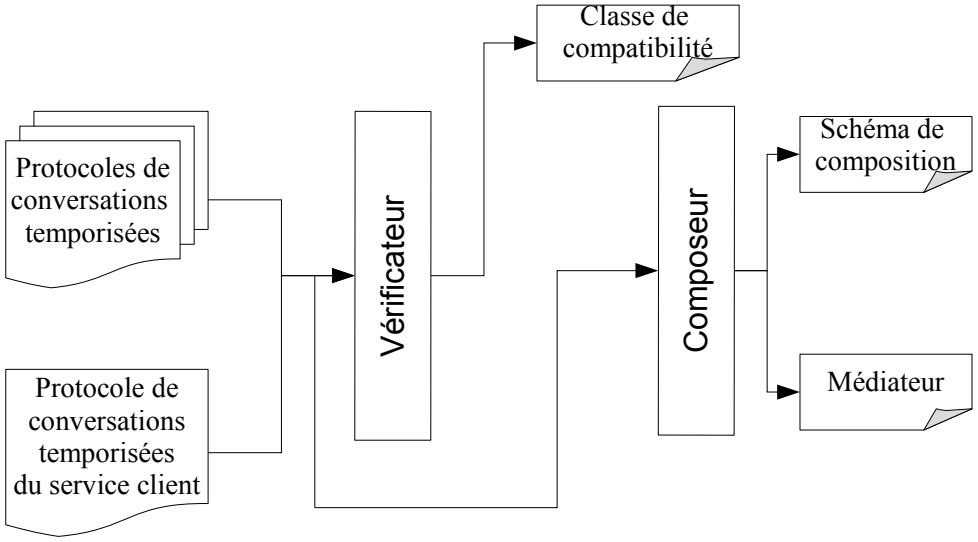


FIGURE 6.1 – L'architecture de l'approche proposée

Le composant *Vérificateur* implémente les primitives de vérification de la compatibilité des services Web temporisés présentées dans le chapitre 4. Quant au deuxième composant, il met en œuvre l'approche de composition de services Web temporisés présentée dans le chapitre 5. Dans la suite, nous présentons les éléments de chaque composant.

6.2.1 Le vérificateur de la compatibilité temporisée asynchrone d'une chorégraphie

La figure 6.2 illustre l'architecture globale de l'analyseur de compatibilité de services Web que nous proposons. Ce composant repose principalement sur trois modules : (1) le module *CTP2UTA (Conversational Timed Protocol to UPPAAL Timed Automata)*, (2) le module *UTA2CTL (UPPAAL Timed Automata to CTL)*, et (3) le model checker UPPAAL.

Le module *CTP2UTA* implémente les étapes décrites dans la section 4.3. Il a comme entrée un ensemble de protocoles de conversations à vérifier et a comme sortie un ensemble d'automates UPPAAL. En effet, son but est de transformer la description des protocoles de conversations pour produire des descriptions UPPAAL qui préservent la sémantique des protocoles de conversations temporisés asynchrones.

La spécification des automates UPPAAL produite par *CTP2UTA* est l'entrée de *UTA2CTL*. Ce dernier est chargé de construire les formules CTL qui caractérisent les différentes classes de compatibilité décrites dans la section 4.4.

La description des automates UPPAAL ainsi que les formules CTL sont ensuite incorporées au model checker UPPAAL pour déterminer la classe de compatibilité de l'ensemble des protocoles de conversations.

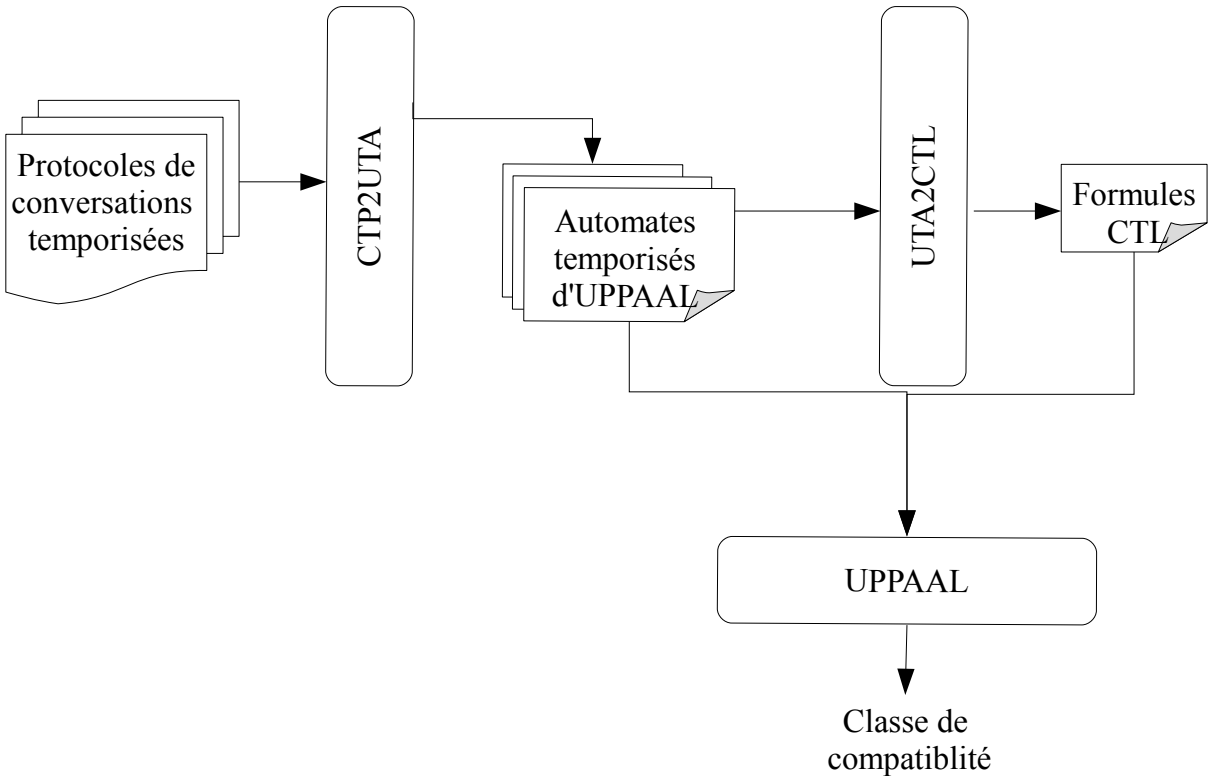


FIGURE 6.2 – Analyseur de compatibilité temporisée

6.2.2 Composition de services temporisée

Le deuxième composant de notre cadre consiste en l'outil *composeur* dont l'architecture est illustrée sur la Figure 6.3. Ce composant dispose de trois modules : (1) le module *P2PCoord (P2P coordinator)*, (2) le module *ClockOrd (clock ordering)*, et (3) le module *Med (Mediator)*.

Le point d'entrée du composeur est un ensemble de protocoles de conversations ainsi que le protocole de conversations du service client. Le module *P2PCoord* implante l'algorithme 5.1 du chapitre 5. Lors de la création d'une connexion entre deux services Web,

ce composant fait appel à *clockOrd* pour la détection des éventuels conflits temporels. Quand la création de la connexion est réussie, *P2Pcoord* met à jour le ASCT ainsi que l'historique des données échangées. *CoordP2P* peut aussi faire appel au composant *Med* qui se charge de la génération du médiateur pour consommer un message ou en produire un. Quand ce composant génère un médiateur, la description du médiateur est mise à jour.

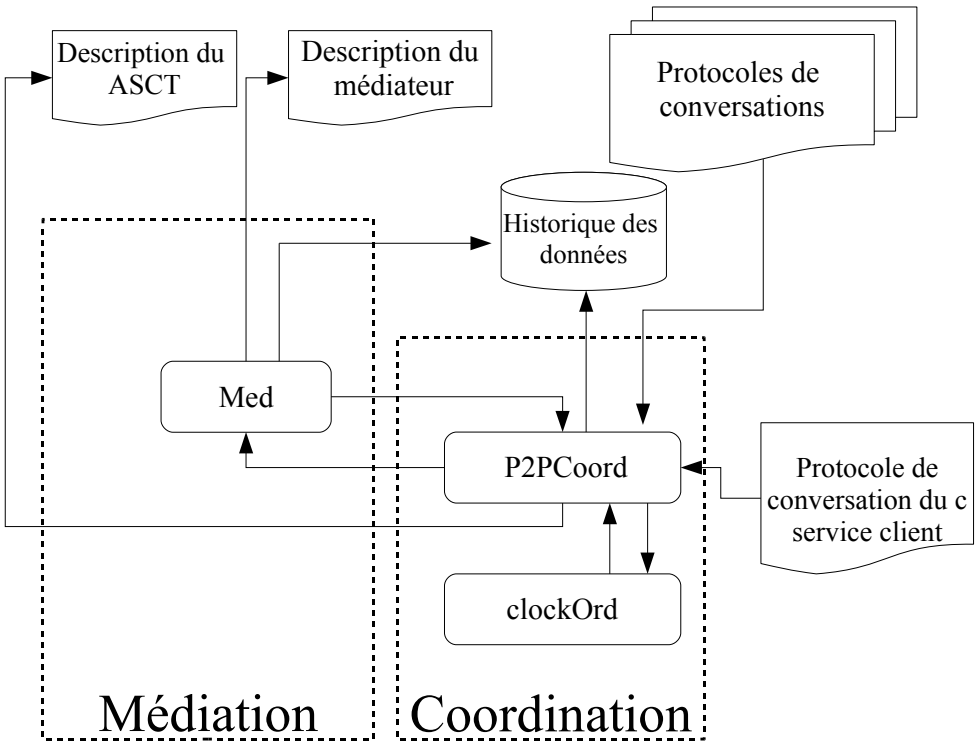


FIGURE 6.3 – L'architecture de l'approche proposée

Dans la suite, nous présentons les détails techniques de la mise en œuvre des différents composants.

6.3 Les détails d'implantation

Dans cette section, nous commençons tout d'abord par décrire la spécification utilisée pour représenter les différentes entrées et sorties des composants. Puis, nous exposons les éléments relatifs à l'implantation du vérificateur ainsi que le composeur.

6.3.1 Représentation des protocoles de conversations temporisés

L'entrée de notre plateforme est un ensemble de protocoles de conversations temporisés. Nous avons opté pour le langage de description XML pour la représentation de la description des protocoles de conversations en automates. Un protocole de conversation est un document XML qui a comme racine l'élément <*services*>. Cet élément regroupe un ensemble d'éléments <*service*> qui décrivent les composants de chaque service Web. Chaque élément <*service*> est défini par les éléments suivants :

- <*states*> : spécifie l'ensemble des états du service
- <*finalStates*> : décrit l'ensemble des états finaux d'un service
- <*initialState*> : représente l'état initial du service
- <*transitions*> : spécifie l'ensemble des transitions du service. Cet élément contient un ensemble d'éléments <*transition*>. Chaque élément définit les composants suivants :
  - <*name*> : spécifie l'identifiant de la transition
  - <*from*> : représente l'état source de la transition
  - <*to*> : décrit l'état cible de la transition
  - <*message*> : décrit le message échangé lors du déclenchement de la transition. Cet élément spécifie :
    - <*messageName*> : spécifie le nom du message
    - <*polarity*> : identifie la polarité du message. La polarité décrit si le message est un message entrant ou sortant.
    - <*parameters*> : Cet élément décrit les paramètres du message échangé. Il englobe un ensemble d'éléments <*parameter*> qui décrivent les noms des paramètres.
  - <*dataConstraints*> : regroupe un ensemble d'élements <*dataConstraint*>. Chaque élément spécifie une contrainte de données :
    - <*dataName*> : la donnée
    - <*operator*> : l'opérateur utilisé pour spécifier la contrainte temporelle
    - <*value*> : la valeur que doit respecter la contrainte de donnée.
  - <*clocksReset*> : designe l'ensemble des éléments <*clockName*> qui correspondent aux noms des horloges remises à zéro lors du déclenchement de la transition
  - <*timedConstraints*> : symbolise l'ensemble des contraintes temporelles. Cet élément comprend un ensemble d'éléments <*timedConstraint*> qui décrivent les éle-

ments suivants :

- <*clockName*> :décrit le nom d'une horloge
- <*operator*> : spécifie l'opérateur utilisé pour spécifier la contrainte temporelle
- <*value*> : indique la valeur que doit respecter la contrainte temporelle.

La figure 6.4 montre un extrait de document XML du protocole SP.

Dans la section suivante, nous allons décrire le vérificateur de compatibilité de services Web temporisés.

6.3.2 Implantation du vérificateur

Le vérificateur de la compatibilité a été implémenté en JAVA sous forme d'un outil nommé CTP2UTA. Cet outil a été implémenté sous forme d'un parseur XML, qui a été développé en utilisant l'API JDOM (Java Document Object Model). Ce parseur a comme sortie un fichier XML décrivant l'ensemble des automates UPPAAL ayant une sémantique équivalente aux protocoles de conversations. Un automate UPPAAL préservant la sémantique des protocoles de conversations temporisés est un fichier XML ayant comme racine l'élément <*nta*> qui regroupe les éléments suivants :

- <*declaration*> : cet élément regroupe les déclarations globales des services. Dans le contexte de notre travail, il décrit les variables des messages de l'ensemble des services
- <*template*> : Cet élément décrit un service Web. Il comprend les éléments suivants :
  - <*name*> : désigne le nom du service
  - un ensemble d'éléments <*location*> spécifiant les états.
  - un ensemble d'éléments <*transition*>. Chaque élément spécifie :
    - <*source*> : désigne l'état source de la transition
    - <*target*> : spécifie l'état cible de la transition
    - un ensemble d'éléments <*label*> qui décrivent le type d'étiquette associée à la transition. Plusieurs types sont considérés :
      - *guard* : cet élément correspond à une contrainte
      - *synchronisation* : correspond à une étiquette de type synchronisation, i.e., envoi ou reception de message
      - *assignment* : correspond à une étiquette de type affectation, i.e., affectation d'une valeur (opération) à une variable.
- <*system*> : cet élément correspond à l'instanciation des services.

```
<service>
    <title>SP</title>
    <states>
        <state> p0 </state>
        <state> p1 </state>
    </states>
    <finalStates>
        <finalState>p7</finalState>
    </finalStates>
    <initialState>p0</initialState>
    <transitions>
        <transition>
            <name>T1</name>
            <from>p0</from>
            <to>p1</to>
            <message>
                <messageName>demande_bourse</messageName>
                <polarity>?</polarity>
                <parameters>
                    <parameter>sn</parameter>
                    <parameter>age</parameter>
                    <parameter>handicap</parameter>
                </parameters>
            </message>
            <dataConstraints>
                <dataConstraint>
                    <dataName>age</dataName>
                    <operator> &gt; </operator>
                    <value>16</value>
                </dataConstraint>
            </dataConstraints>
            <clocksReset>
                <clockName>t1</clockName>
            </clocksReset>
            <timedConstraints/>
        </transition>
        <transition>
            <name>T3</name>
            <from>p2</from>
            <to>p3</to>
            <message>
                <messageName>demande_examen</messageName>
                <polarity>!</polarity>
                <parameters>
                    <parameter>sn</parameter>
                    <parameter>handicap</parameter>
                    <parameter>motif</parameter>
                </parameters>
            </message>
            <timedConstraints>
                <timedConstraint>
                    <clockName>t2</clockName>
                    <predicat> &gt;</predicat>
                    <value>36</value>
                </timedConstraint>
            </timedConstraints>
        </transition>
    </transitions>
</service>
```

FIGURE 6.4 – Extrait de la description XML du protocole de conversations du service SP

La figure 6.5 montre un extrait d'un document XML permettant de spécifier les automates UPPAAL générées.

```
<?xml version="1.0" encoding="utf-8"?>
    <nta>
        <declaration>
            int
notification_disponibilite=0,examen_permis=0,carte_handicape=0,examen_conduire=0,demande_carte_hadicape=0;
            urgent broadcast chan cv;
        </declaration>
        <template>
            <name>PService</name>
            <declaration>
            clock t1,t2;
            </declaration>
            <location id="id0">
                <name>p0</name>
            </location>
            <location id="id1">
                <name>p1</name>
            </location>
            <location id="id2">
                <name>p2</name>
            </location>
            <location id="id6">
                <name>final</name>
            </location>
            <transition>
                <source ref="id0"/>
                <target ref="id6"/>
                <label kind="guard">notification_disponibilite>0</label>
                <label kind="synchronisation">cv!</label>
                <label kind="assignment">notification_disponibilite--</label>
            </transition>
        </template>
        <system>
            SP = PService();
            system SP;
        </system>
    </nta>
```

FIGURE 6.5 – Description XML de l'automate UPPAAL généré du service SP

En utilisant la spécification des automates UPPAAL générée, le composant *UTA2CTL* génère les formules CTL qui caractérisent l'ensemble des classes de compatibilité. Ces formules sont décrites dans un fichier ayant une extension *.q*.

La formule 6.1 montre une partie de la propriété de la compatibilité totale parfaite générée.

```
A<>SP.final and SM.final and SEM.final
/*

*/
SP.final and SM.final and SEM.final --   > formulaire_permis==0
and carte_handicape==0 and notification_disponibilite==0
and demande_permis==0 and formulaire==0 and exa-
men==0 and examen_permis==0 and examen_conduire==0 and
demande_carte_handicape==0
```
(6.1)

La figure 6.6 montre une capture d'écran de l'outil CTP2UTA. Le menu *File* dispose de deux sous éléments : (1) *Open* et *Exit*. Le bouton Open permet de choisir le fichier XML contenant la description des protocoles de conversations. Le bouton *Exit* permet de fermer l'outil CTP2UTA.

Une fois la spécification XML des services sélectionnée, l'arborescence des traces peut être visualisée grâce à l'onglet gauche. En outre, une description textuelle des protocoles de conversations se charge dans la partie droite.

Par le biais du bouton *Transform* (voir Figure 6.7), notre parseur génère une spécification XML des automates UPPAAL qui préservent la sémantique des protocoles de conversations temporisés asynchrones. La figure 6.7 montre un exemple d'exécution de l'outils en utilisant le scénario introduit dans la section 1.3 du chapitre 1.

En outre, comme le montre la Figure 6.8, l'outil produit en sortie un autre fichier .q dans lequel sont spécifiées les propriétés CTL qui caractérisent les différentes classes de compatibilité.

Les fichiers générés par l'outil CTP2UTA sont ensuite introduits dans le model checker UPPAAL, comme le montre la Figure 6.9.

6.3.3 Implantation de la composition

Les modules du composeur ont également été implémentés en JAVA. Ce composant a comme entrée la spécification XML des services Web. Ce composant distingue entre les services Web et le service client. En utilisant la spécification des services Web et celle du service client, soit le composant produit deux spécifications : (1) un fichier XML décrivant

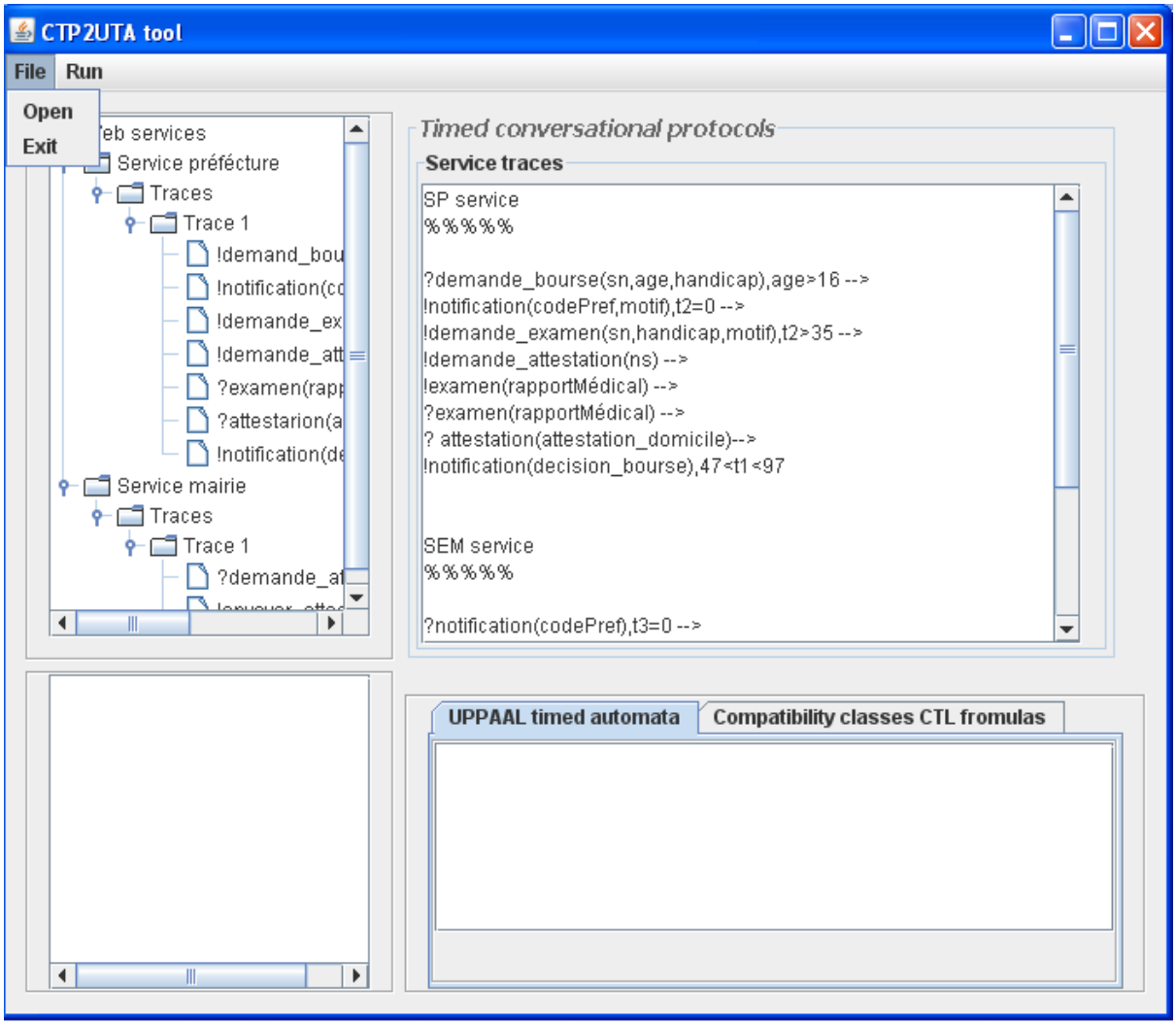


FIGURE 6.6 – Chargement des protocoles de conversations

le schéma de composition temporisé, et (2) un fichier XML spécifiant le médiateur, soit ne retourne aucun fichier et affirme que la composition ne peut pas réussir.

La figure 6.10 illustre un extrait de la spécification XML du ASCT généré. Quant à la Figure 6.11, elle montre un extrait de la spécification XML du médiateur généré.

Afin d'évaluer l'approche proposée, nous avons mené des tests préliminaires. Ces expérimentations ont été faites sur une machine de 1,66 GHz CPU, et 1 Go de mémoire. Nous avons remarqué qu'en générale le temps d'execution de l'algorithme proposé est proportionnel à la complexité du service client. En effet, plus le nombre de transitions et des traces du service client est important, plus le temps d'execution de l'algorithme augmente. Figure 6.12 montre l'évaluation du temps d'exécution (en milliseconde) du calcul de composition par rapport au nombre de transitions du service client.

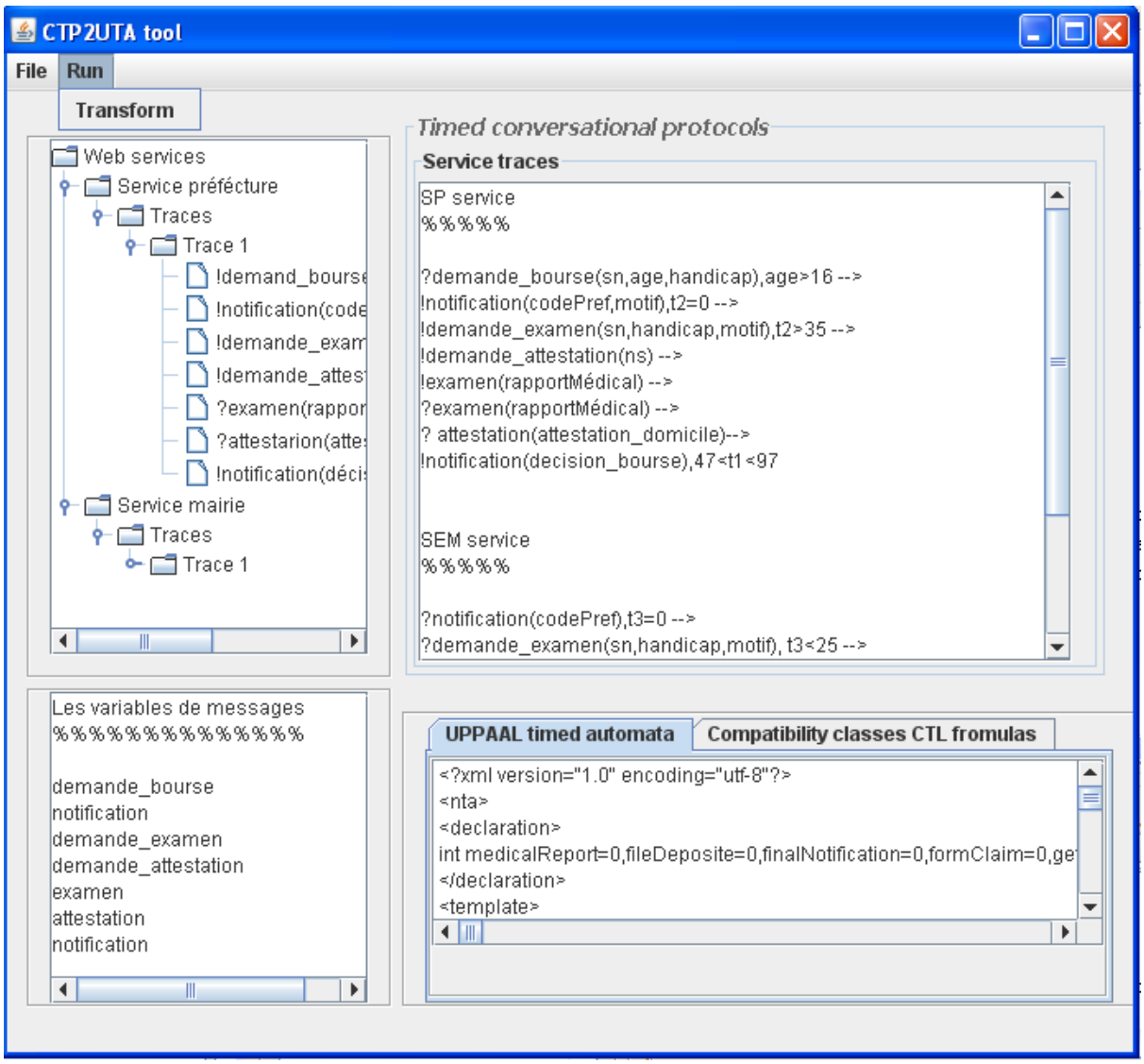


FIGURE 6.7 – Une capture d'écran de la spécification XML des automates UPPAAL générés

En particulier, parmi les transitions qui ont un impact considérable sur le temps d'exécution de l'algorithme est le nombre de transitions de réception de messages, et surtout les messages manquants (i.e., requis). Ceci est dû au fait que les services Web sont parcourus à chaque fois que la connexion de la transition du service client échoue. Ainsi, plus le nombre de messages requis est important et la complexité des services Web est importante, plus la complexité du calcul augmente. Figure 6.13 montre l'évaluation du temps d'exécution (en milliseconde) du calcul de composition par rapport au nombre de transitions de réception de messages manquants du service client.

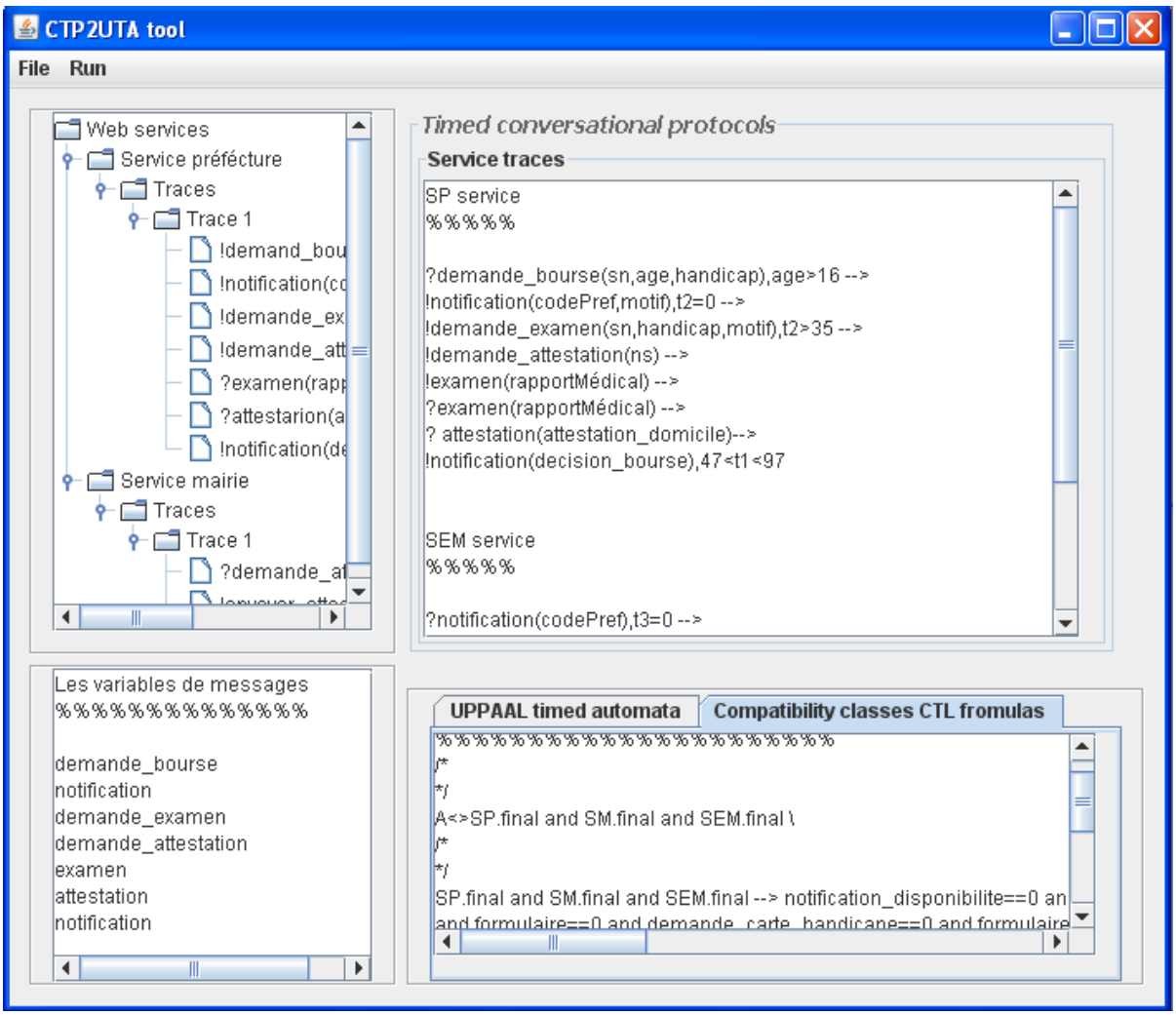


FIGURE 6.8 – Une capture d'écran des propriétés CTL générées

Le module *P2PCoord* a été intégralement implémenté ainsi que le module *Med*. Quant à la partie fonctionnelle du module *clockOrd*, sa version actuelle permet de détecter les conflits temporels de la forme $v \leq x_0 \leq \ldots \leq x_n \leq v'$ tels que $x_0, \ldots x_n$ sont des horloges, v et v' sont des valeurs entières, où $v > v'$. Cependant, les conflits du type $x - x' \leq v$ (resp. $x - x' \geq v$) sont en cours de développement. Un outil doté d'une interface graphique reposant sur les différents modules du composeur est également en cours d'implantation.

6.4 Conclusion

Dans ce chapitre, nous avons décrit les aspects expérimentaux de la mise en œuvre de l'approche proposée dans ce manuscrit. Nous avons présenté les différents éléments

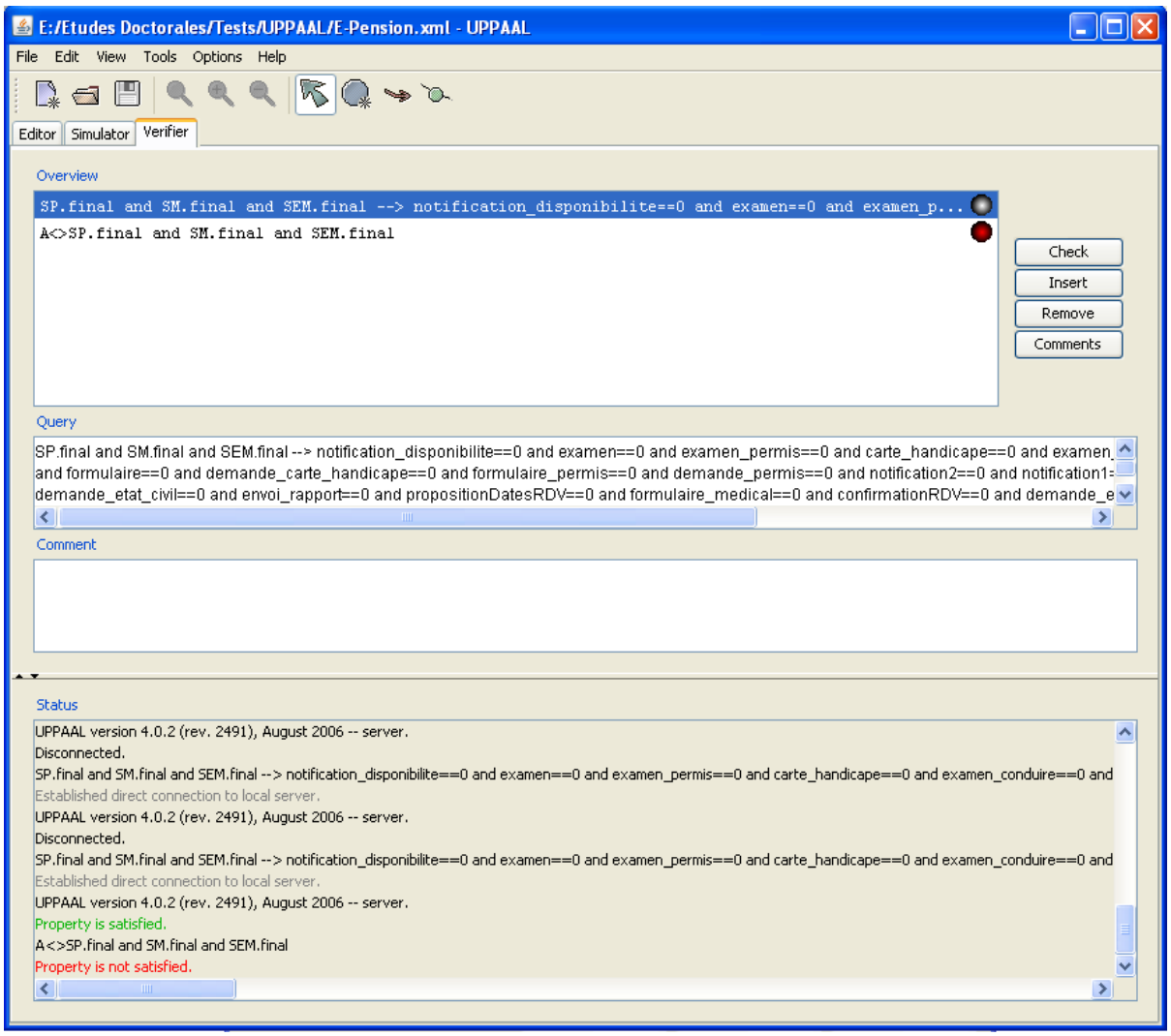


FIGURE 6.9 – Une capture d'écran du vérificateur de UPPAAL

conceptuels qui consistent en deux composants :

- *Le vérificateur de compatibilité* : ce composant dispose des éléments supportant l'analyse de la compatibilité. Il implante les étapes de transformation de la description des protocoles de conversations ainsi que les étapes de génération des propriétés CTL.
- *Le composeur* : Ce composant est constitué de plusieurs modules permettant la création d'une composition temporisée.

Nous avons exposé les détails techniques d'implémentation du prototype que nous avons exploité afin de mener des expérimentations préliminaires.

```
<?xml version="1.0" encoding="UTF-8"?>
    <ASCT>
        <states>
            <state>s0p0r0d0m0</state>
            <state>s1p1r0d0m1</state>
            <state>s5p3r0d5m8</state>
        </states>
        <finalStates>
            <finalState>s8p7r2d8m12</finalState>
        </finalStates>
        <transitions>
            <transition>
                T0
                <from>s0p0r0d0m0</from>
                <to>s1p1r0d0m1</to>
                <message>
                    <messageName>demande_bourse</messageName>
                    <parameters>
                        <parameter>ns</parameter>
                        <parameter>age</parameter>
                        <parameter>handicap</parameter>
                    </parameters>
                </message>
                <dataConstraints>
                    <dataConstraint>
                        <dataName>age</dataName>
                        <predicat> &gt; </predicat>
                        <value>16</value>
                    </dataConstraint>
                </dataConstraints>
                <clocksOrders>
                    <clocksOrder>
                        <clockName>t1</clockName>
                        <clockName>t5</clockName>
                        <predicat> = </predicat>
                    </clocksOrder>
                </clocksOrders>
            </transition>
        </transitions>
    </ASCT>
```

FIGURE 6.10 – Description XML du ASCT généré

Actuellement, nos efforts de développement se portent sur l'outil *composeur* ainsi que l'amélioration de l'outil CTP2UTA. Le but étant de fournir une plateforme intégrant le vérificateur ainsi que le composeur. Ceci permettra de fournir un support visuel à l'utilisateur facilitant l'interaction et la compréhension des résultats retournés. Cela nous permettra également de mener des tests plus approfondis.

```
<?xml version="1.0" encoding="UTF-8"?>
    <mediator>
        <states>
            <state>m0</state>
            <state>m1</state>
            <state>m2</state>
        </states>
        <initialState>m0</initialState>
        <finalStates>
            <finalState>m0</finalState>
            <finalState>m3</finalState>
        </finalStates>
        <transitions>
            <transition>
                <from>m1</from>
                <to>m0</to>
                <message>
                    <messageName>empty</messageName>
                    <polarity/>
                    <parameters/>
                </message>
                <dataConstraints/>
                <clocksReset>
                    <clockName>t1</clockName>
                    <clockName>t5</clockName>
                </cocksReset>
                <timedConstraints/>
            </transition>
```

FIGURE 6.11 – Description XML du médiateur généré

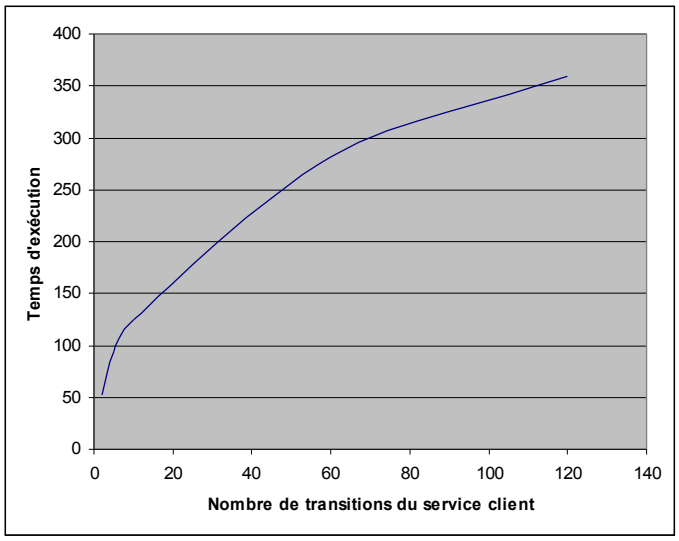


FIGURE 6.12 – Evaluation de la performance de l'approche proposée par rapport au nombre des transitions du service client

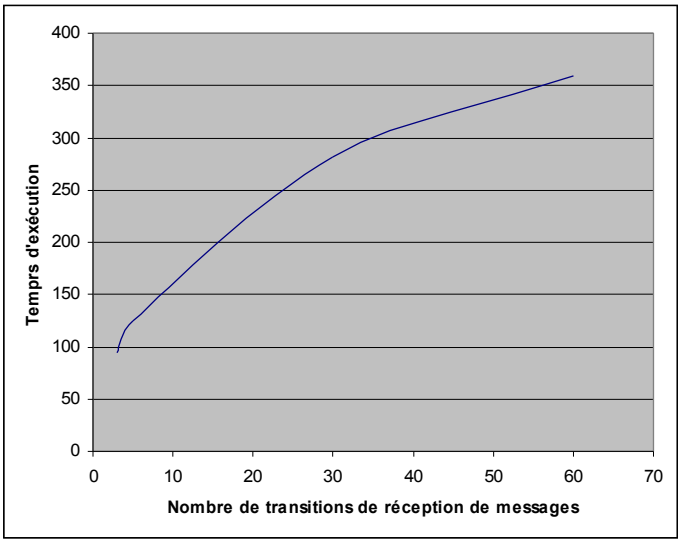


FIGURE 6.13 – Evaluation de la performance de l'approche proposée par rapport au nombre des transitions de réception de messages du service client

Chapitre 7

Conclusion générale et perspectives

Les efforts de développement technologique du Web et la standardisation des spécifications sous-jacentes a incité l'adoption du Web comme infrastructure pour la mise en œuvre des collaborations à échelle mondiale. Ainsi, les paradigmes a base de composants ont évolué vers le paradigme orienté service qui s'articule autour du concept de service Web. La caractéristique importante des services Web est sa propriété de *composabilité* qui permet de créer des services plus complexes en combinant des services plus élémentaires en fonction de propriétés fonctionnelles et non-fonctionnelles. Durant cette thèse, nous avons porté notre attention en particulier aux propriétés temporelles qui spécifient les délais nécessaires pour échanger des messages. Comme démontré par des travaux préliminaires, les aspects temporels sont d'une très grande importance dans les interactions des services Web (exemple [15, 79]).

Etant donné l'hétérogénéité des services Web, dès qu'ils interagissent ensemble, des conflits temporisés (et/ou non-temporisés) peuvent surgir et peuvent porter atteinte à leur composabilité. De ce fait, il s'avère nécessaire de s'appuyer sur des mécanismes formels automatiques pour la vérification et la composition de ces services. C'est dans cette optique que nous avons mené l'étude rapportée dans ce manuscrit et qui a abouti à la proposition d'un cadre formel d'analyse de compatibilité et de composition de services Web temporisés asynchrones que nous synthétisons dans les sections suivantes.

7.1 Synthèse des travaux et résultats

Les contributions de cette thèse peuvent se résumer sous la forme des trois points que nous détaillons par la suite.

1. La proposition d'un modèle formel des protocoles de conversations temporisées
2. La proposition d'une approche d'analyse de compatibilité de services Web temporisés asynchrones
3. La proposition d'une approche de composition de services Web temporisés asynchrones

Modélisation des protocoles de conversations temporisées

Dans un premier temps, nous avons proposé un modèle formel qui fournit une sémantique opérationnelle pour la prise en compte des aspects temporels dans les protocoles de conversations de services Web asynchrones. Le modèle de représentation des protocoles de conversations temporisées utilisé dans ce travail est celui des machines à états finis dotés d'horloges. Dans le contexte de ce travail, un protocole de conversations est caractérisé par : (1) les *séquences des messages* supportées, (2) les *données*, (3) les *contraintes de données*, (4) les *contraintes temporelles*, et (5) l'*aspect conversationnel asynchrones* des services Web. Ainsi, notre modèle vient en complément aux modèles proposés dans [24, 79, 78, 115]. En effet, Dans [24], Berardi et al. prennent en considération les messages, les données, et les opérations sans prendre en considération les contraintes temporelles. Dans [15, 115, 79, 78] les auteurs considèrent les messages ainsi que des contraintes temporelles mais sans prendre en compte les données et les contraintes de données.

Ce modèle formel est l'ingrédient de base pour le cadre d'analyse et de composition de services Web présenté ensuite.

Analyse de compatibilité des services Web

En utilisant le formalisme des protocoles de conversations temporisées, nous avons abordé le problème d'analyse de compatibilité de services Web. En général, les approches existantes d'analyse de compatibilité de services Web reposent principalement sur l'hypothèse que les services Web supportent que des communications synchrones (e.g., [15, 113, 115, 88]). Ainsi, dès qu'il s'agit de services asynchrones, les approches existantes sont incapables de caractériser leur compatibilité. D'autre part, le cadre présenté dans [15, 113, 115, 88] permet la détection d'un type particulier de conflits temporels. Par exemple, le conflit du type $x - x' \leq v$ où x et x' sont des horloges et v une valeur entière, présenté dans la section 5.4.2 du chapitre 5, n'est pas considéré. Suite à ces limites, nous avons proposé une approche basée sur le model checking. Le model checker que nous avons choisi est

celui d'UPPAAL, qui repose sur les automates temporisés et qui est doté d'une interface facilitant la simulation et la vérification des processus. Les étapes suivies pour mettre en œuvre cette analyse sont comme suit :

1. *Transformation des protocoles de conversations* : la notion d'envoi et de réception de messages de notre modèle de protocoles de conversations temporisées correspond à la notion de canal de communication dans le modèle de UPPAAL. Cependant, cette notion de canal ne permet de considérer que des services synchrones. Afin de tenir compte de notre sémantique des protocoles de conversations, nous avons proposé un ensembles de règles de transformation. Le but de ces règles est de produire des automates UPPAAL équivalents sémantiquement aux protocoles de conversations temporisées asynchrones.
2. *Caractérisation des classes de compatibilité* : la deuxième étape consiste à caractériser un ensemble de classes de compatibilité de services Web asynchrones temporisés. En utilisant la logique CTL, nous avons défini cinq classes de compatibilité : (a) *compatibilité totale parfaite*, (b) *compatibilité totale non-parfaite*, (c) *compatibilité partielle parfaite*, (d) *compatibilité partielle non-parfaite*, et finalement, (e) *incompatibilité totale*. Ces différentes classes peuvent aider à savoir au préalable le niveau de composabilité d'un ensemble de services Web.
3. *Validation par UPPAAL* : La dernière étape consiste à examiner si les automates résultants du processus de transformation vérifient les formules CTL afin de caractériser la classe de compatibilité des services Web

Les différentes étapes ont été implantées sous la forme d'un prototype, ce qui nous a permis de mener des tests préliminaires.

Composition de services Web

Parallèlement à l'analyse de compatibilité, nous nous sommes également intéressés au problème de composition. Le but est de créer une composition de services Web temporisés qui satisfasse un besoin donné (le service client). Le défi est de pouvoir détecter lors de la création de la composition les conflits et essayer quand cela est possible de les masquer. Pour ce faire, nous avons proposé une approche qui repose sur les étapes suivantes :

1. *Création des connexion P2P* : cette première étape consiste à essayer de créer des connexions entre les services de telle sorte que le service client soit satisfait

2. *Détection des conflits temporels* : lors de la création des connexions P2P, il est important de fournir un mécanisme de détection des conflits temporels. Dans ce contexte, nous avons proposé le *processus d'ordonnancement d'horloges*. Le but est de définir un ordre entre les différentes horloges des différents services Web afin de mettre en avant les éventuels conflits que nous avons présenté dans la section 5.4.2 du chapitre 5.
3. *Génération d'un médiateur* : les conflits en général et en particulier les conflits temporels provoquent l'échec d'une composition. Dans l'objectif de contourner ces conflits, nous avons adopté une démarche basée sur la génération d'un médiateur. Ce médiateur joue le rôle de producteur et/ou de consommateur de messages. L'activité de production est déclenchée principalement quand un message est considéré comme *requis* afin de contourner un eventuel conflit. Quant à la consommation d'un message, elle a lieu quand un message est considéré comme *supplémentaire*.

L'approche de composition de services Web a été implantée sous la forme de plusieurs modules. Par le biais de ces modules, nous avons mené quelques expérimentations initiales pour évaluer la performance des algorithmes proposés.

7.2 Limites et perspectives

Les travaux de recherche qui ont été menés dans le cadre de cette thèse ont pu donner des réponses à certaines questions que nous nous sommes posées, mais ont aussi dégagé un ensemble de questions et de perspectives.

Achever et améliorer l'implantation

Nos efforts actuels de développement se concentrent sur la finalisation du prototype en cours. Le but est de fournir un outil de spécification, de vérification et de composition de protocoles de conversations temporisées asynchrones. Cet outil facilitera l'interaction et la compréhension des résultats fournis. En utilisant cet outil, il serait intéressant de mener des tests plus approfondis en générant des protocoles plus complexes en injectant des erreurs variés et aléatoires afin d'expérimenter davantage l'approche proposée.

Egalement, la mise en œuvre de la génération des messages requis par le médiateur peut être étendue par des mécanismes de transformation pour tenir compte de la structure et de la sémantique des données manipulées [100].

Dynamicité de l'instanciation

Les approches d'analyse et de composition de services Web que nous avons proposées permettent de considérer une seule instance de chaque service. En d'autres termes, lors de l'analyse ou la composition, on suppose qu'une seule instance de chaque service est requise pour réussir la chorégraphie (resp. la composition). Cette hypothèse est restrictive étant donné que pour réussir une chorégraphie (composition), une ou plusieurs instances de chaque service peuvent être requises. Comme le montre la figure 7.1, on ne peut pas réussir la collaboration des deux services Q et Q', avec une seule instance de Q et une instance de Q'. Dans cet exemple, on a besoin de deux instances du service Q et une seule instance du service Q'.

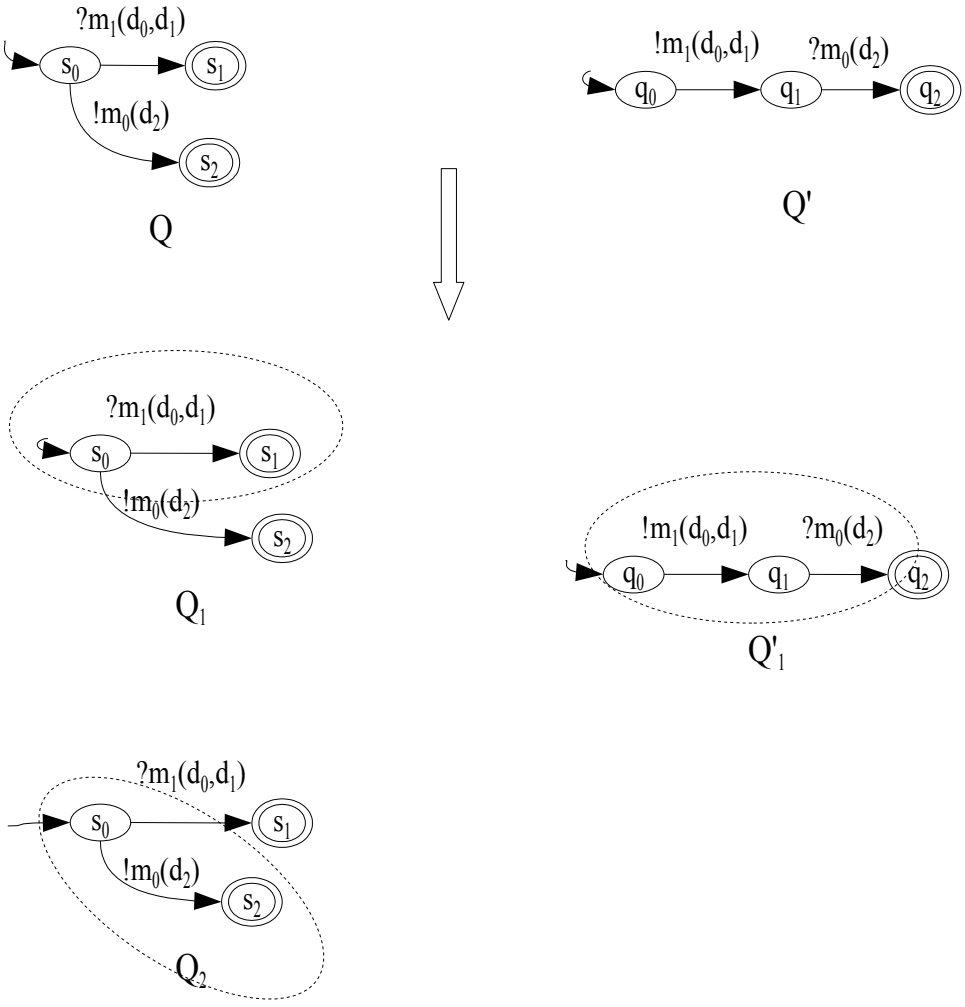


FIGURE 7.1 – Nécessité de considérer l'instanciation dynamique dans les interactions des services Web

L'exemple présenté dans la figure 7.1 montre un cas très simple et non-temporisé de services Web. Comme travaux futurs, nous envisageons d'étendre les primitives présentées dans cette thèse pour considérer cet aspect dynamique d'instanciation lors de l'étude des interactions temporisées des services Web asynchrones et temporisés.

Etude d'autres formes temporelles

Le modèle temporel que nous considérons permet de spécifier les délais nécessaires pour échanger des messages. Cependant, le comportement des services Web peut être contraint par d'autres formes de propriétés temporelles, telles que les compteurs, des contraintes utilisant le temps absolu et la durée de validité de données. En outre, des contraintes globales plus complexes qui portent sur l'ensemble de la composition (chorégraphie) et qui définissent des relations entre les différentes contraintes temporelles des services Web, peuvent être investiguées.

D'autre part, l'approche proposée dans ce manuscrit est applicable avant l'exécution en temps réel de la composition (chorégraphie). De ce fait, on a supposé que le temps de communication est négligeable. Cette hypothèse nous a permis d'étudier l'impact des propriétés temporelles dans la compatibilité et la composition de services Web en ne regardant que leurs propres contraintes temporelles. Il serait intéressant d'étendre les primitives proposées pour tenir compte du temps de communication qui peut avoir dans certains cas un effet considérable sur la réussite de la composition (chorégraphie).

Interactions avec d'autres dimensions

A part les aspects fonctionnels que nous avons pris en considération dans cette thèse, d'autres aspects non-fonctionnels jouent encore un rôle très important, tels que les aspects de sécurité. Etant donné que le modèle formel que nous avons proposé repose sur l'échange de messages impliquant des données, alors il est crucial de se baser sur des protocoles sécurisés assurant ainsi la sécurité et la confidentialité des données via par exemple l'introduction des politique privées (privacy policies) [58, 69].

Annexe A

Propriètés de l'algorithme de composition temporisée

Dans cette section, nous étudions les propriétés de nos algorithmes. Particulièrement, nous étudions les propriétés de terminaison et de sûreté.

A.1 Propriété de terminaison

Dans cette section, nous discutons la terminaison de l'algorithme que nous proposons. Supposons que l'algorithme ne termine pas. Dans ce cas, il doit exister au moins une trace dans le ASCT qui a un nombre infini de transitions. Ceci peut arriver si les services ont des états qui peuvent être visités indéfiniment (i.e., il y a un cycle). L'algorithme que nous proposons prend en considération les cycles pour éviter de boucler indéfiniment. Comme on a un nombre fini d'états et toutes les transitions sont visitées un nombre fini de fois, alors notre algorithme ne boucle pas. De ce fait, a chaque parcours, notre algorithme soit construit une transition ASCT quand deux services peuvent être connectées, ou échoue. Alors on conclut que l'algorithme termine.

A.2 Propriété de sûreté

Supposons que notre algorithme ne soit pas sûr. Dans ce cas, l'algorithme soit construit un ASCT bien que deux services ne puissent pas être connectés, ou l'algorithme échoue bien qu'il y ait des services qui puissent être connectés.

Nous commençons par le premier cas dans lequel l'algorithme construit un ASCT.

Dans notre travail, une transition ASCT $(\overline{s}, m(\overline{d}), \psi, c, \overline{s'})$ spécifie qu'un message $m(\overline{d})$ transite via un canal, i.e., il existe un service qui envoie le message et un autre service qui le reçoit tel qu'aucun conflit ne surgisse. Donc, notre algorithme génère une transition ASCT seulement si il y a deux services qui peuvent être connectés.

Maintenant, supposons qu'il y ait deux services qui puissent échanger un message. Dans ce cas, en définissant un ordre entre les horloges, l'algorithme crée une transition ASCT, i.e., si il y a deux services qui peuvent être connectés, l'algorithme génère une transition ASCT. Donc, notre algorithme génère une transition ASCT quand des services peuvent être connectés correctement, soit ne construit pas le ASCT.

Annexe B

Descritpion en WSDL d'un service Web

L'interface d'un service Web est décrite par une spécification WSDL en plusieurs parties :

- types : décrit tous les types de données utilisés,
- message : décrit le nom du message ainsi que les différentes parties qui le constituent,
- portType : définit les signatures des opérations, les paramètres des opérations et leurs types,
- binding : établit le mécanisme de communication du service Web.

La structure d'une description WSDL est comme suit :

<definitions>

<types>

définition des types

</types>

<message>

définition d'un message

</message>

<portTypes>

définition des opérations du service

</portTypes>

<binding>

définition du protocol des interactions

</binding>

</definitions>

<definitions>

Le fichier suivant présente un exemple d'un extrait d'une description WSDL du service SP :

<definitions>

<message name="demande_bourse">

<part name="ns" type="xs :string"/>

<part name="age" type="xs :int"/>

<part name="handicap" type="xs :string"/>

</message>

<message name="notification">

<part name="codePref" type="xs :int"/>

<part name="motif" type="xs :int"/>

</message>

<portType name="SPItems">

<operation name="recevoirDemande">

<input message ="demande_bourse">

<output message ="notification">

</operation>

</portTypes>

<binding type="SPItems" name ="B1">

<soap :binding style="document" transport="http ://schemas.xmlsoap.org/soap/http"/>

<operation>

<soap : operation soapAction ="http ://example.com/getItem"/>

<input>

<soap : body use =”literal”/>

</input>

<output>

<soap : body use =”literal”/>

</output>

</operation>

</binding>

</definitions>

<definitions>

Dans cet exemple, les messages décrits sont des messages en entrée et en sortie de type chaîne de caractères. L'opération de réception des demandes de bourse est *recevoir-Demande* qui possède deux paramètres : un premier paramètre en entrée et un deuxième paramètre en sortie. Les messages échangés sont dans le format SOAP et le protocol de communication est HTTP.

Annexe C

Génération du protocole de conversations temporisées de la spécification OWL-S étendue

Comme cité aupavaravant, les protocoles de conversations peuvent être spécifiés en utilisant les standards proposées ces dernières années tels que, entre autres, BPEL, OWL-S, et WSCL. Dans cette section, nous présentons un exemple d'extraction du protocole de conversations temporisées de la spéicification OWL-S annotée par les propriétés temporelles que nous considérons.

OWL-S est un langage basé sur une ontologie pour la description des propriétés des services Web. Dans ce langage, chaque service Web est spécifié par trois parties XML : *service profile*, qui décrit ce qu'un service fait ; *service model*, qui décrit comment le service fonctionne (se comporte) ; et *service grounding*, qui fournit les détails du comment invoquer le service via des messages [92].

OWL-S permet la description du comportement externe d'un service Web en se basant sur un modèle sémantique, dans lequel chaque *processus atomique* est décrit sémantiquement par des entrées, pré-conditions, sorties, et effets. Un processus atomique correspond à une opération WSDL qu'un service peut exécuter.

Un processus atomique avec des entrées mais qui n'a pas de sortie correspond à une opération WSDL de réception de message. Un processus atomique avec des entrées et des sorties correspond à une opération WSDL de réception-émission. Un processus composite avec des entrées et des sorties où l'envoi des sorties précède la réception des entrées, correspond à une opération WSDL de *émission-retour*. Finalement, un processus atomique

avec des sorties sans entrées correspond à une opération WSDL de *notification*.

Comme le montre la Figure C.1, la sortie du processus atomique *PropositionRDV* du service SEM est la données *dates* qui correspond au message WSDL $propos\_RDV(dates)$ et l'entrée *date* correspond au message sortant *date_RDV(date)*. Le fait que le service SEM doive recevoir la donnée *date* dans les 72 heures de l'envoi de *dates*, est modélisé par une séquence de deux transitions $(s_0, !propos\_RDV(dates), x = 0, s_1)$ et $(s_1, ?date\_RDV(date), x \leq 72, s_2)$.

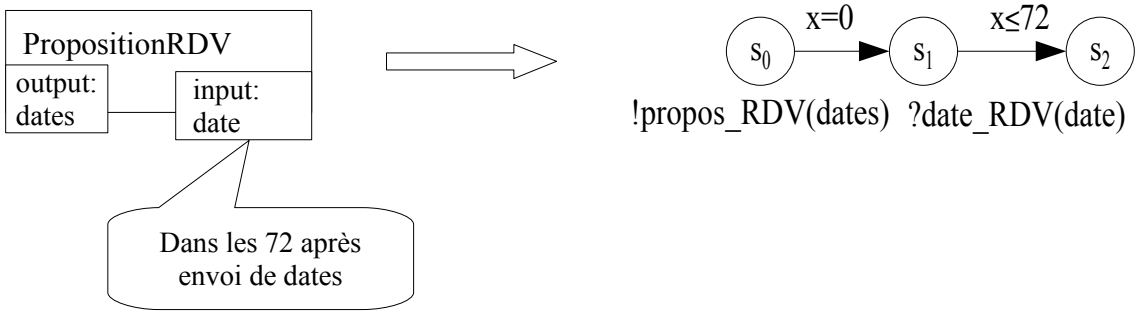


FIGURE C.1 – Représentation de la description du processus atomique étendue en automate

Afin de décrire le flux de contrôle entre les processus atomiques, OWL-S définit les structures suivantes :

1. *Sequence* : tous les processus sont exécutés séquentiellement
2. *If-then-else* : le choix de l'exécution dépend de certaines conditions
3. *Choice* : l'exécution dépend de certaines informations
4. *Repeat-while* : les processus sont exécutés tant que certaines conditions sont satisfaites
5. *Repeat-until* : les processus sont exécutés jusqu'à ce que certaines conditions soient satisfaites
6. *Anyorder* : les processus sont exécutés dans un ordre qui est décidé à l'execution en temps réel.
7. *Split* : spécifie que les processus sont exécutés en parallèle.
8. *Split+Join* : spécifie que les composants ne sont pas activés jusqu'à ce qu'une execution parallèle des processus soit achevée.

Figure C.2 illustre la correspondance entre les opérateurs de OWL-S et leur représentation en automate.

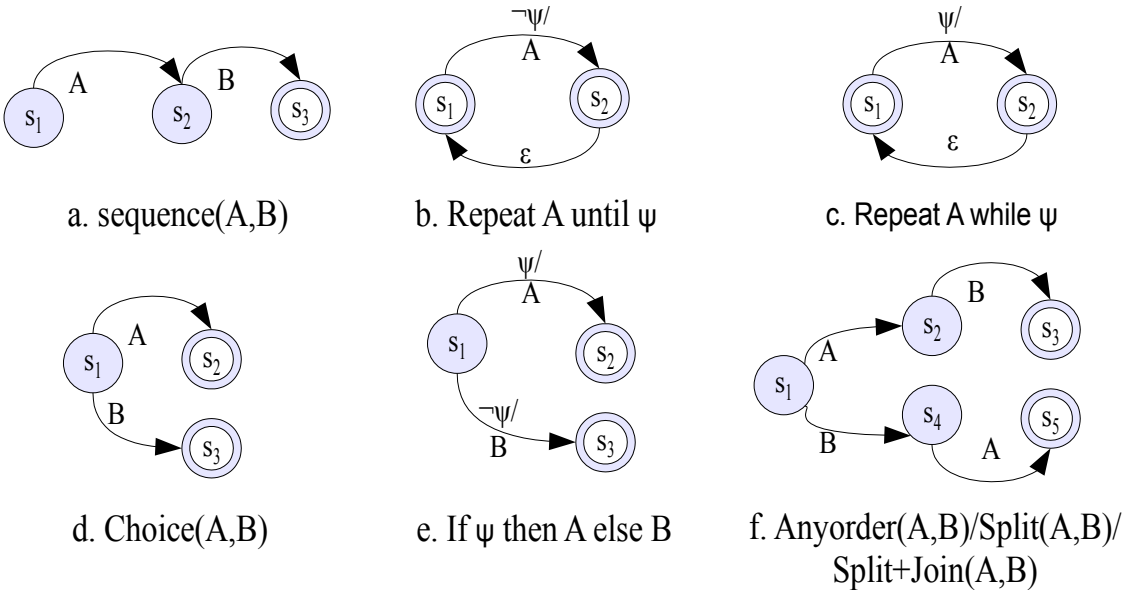


FIGURE C.2 – Représentation des opérateurs de OWL-S en automate

Bibliographie

[1] Oasis. web services business process execution language version 2.0. *http ://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html*, April, 2007.

[2] Marco Aiello, Mike P. Papazoglou, Jian Yang, Mark James Carman, Marco Pistore, Luciano Serafini, and Paolo Traverso. A request language for web-services based on planning and constraint satisfaction. In *Proceedings of the* 3^{rd} *International Workshop on Technologies for E-Services (TES'02)*, pages 76–85, August 23-24, Hong Kong, China, 2002.

[3] Ali Ait-Bachir. *ArchiMed : un canevas pour la détection et la résolution des incompatibilités des conversations entre services web.* PhD thesis, Université Joseph Fourier. Grenoble 1, 2008.

[4] G. Alonso, F. Casati, H.A. Kuno, and V. Machiraju. Web services - concepts, architectures and applications. *Springer Verlag*, Heidelberg, 2004.

[5] Michael Altenhofen, Egon Börger, and Jens Lemcke. An execution semantics for mediation patterns. In *In Proceedings of the BPM 2005 Workshops : Workshop on Choreography and Orchestration for Business Process Managament*, 2005.

[6] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[7] Muscholl Anca and Walukiewicz Igor. A lower bound on web services composition. In *Proceedings of the* 10^{th} *International Conference on Foundations of Software Science and Computation Structures (FOSSACS'07)*, volume 4423, pages 274–287, March 24-April 1, Braga, Portugal ? 2007.

[8] Ferrara Andrea. Web services : a process algebra approach. In *Proceedings of the* 2^{nd} *international conference on Service oriented computing (ICSOC'04)*, pages 242–251, November 15-19, New York, NY, USA, 2004.

[9] Karim Baïna, Boualem Benatallah, Fabio Casati, and Farouk Toumani. Model-driven web service development. In *Proceedings of the* 16^{th} *International conference on Advanced Information Systems Engineering (CAiSE'04).*

[10] Philippe Balbiani, Fahima Cheikh, , and Guillaume Feuillade. Composition of interactive web services based on controller synthesis. In *Proceedings of the IEEE Congress on Services (Services'08) - Part I*, pages 521–528, Washington, DC, USA, 2008.

[11] Matteo Baldoni, Cristina Baroglio, Alberto Martelli, Viviana Patti, and Claudio Schifanella. Verifying the conformance of web services to global interaction protocols : A first step. In *Proceedings of the International Workshop on Web Services and Formal Methods (WS-FM'05)*, pages 257–271, September 1-3, Versailles, France, 2005.

[12] Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, Kannan Govindarajan, Alan Karp, Harumi Kuno, Mike Lemon, Gregory Pogossiants, Shamik Sharma, and Scott Williams. Web services conversation language (wscl) 1.0. *http ://www.w3.org/TR/wscl10/*, 14 March 2002.

[13] B Benatallah, F Casati, and F Toumani. Analysis and management of web service protocols. *23rd International Conference on Conceptual Modeling*, November 2004.

[14] Boualem Benatallah, Fabio Casati, Daniela Grigori, Hamid R. Motahari Nezhad, and Farouk Toumani. Developing adapters for web services integration. In *Proceedings of the* 17^{th} *International Conference on Advanced Information Systems (CAiSE'05)*, pages 415–429, June 13-17, Porto, Portugal, 2005.

[15] Boualem Benatallah, Fabio Casati, Julien Ponge, and Farouk Toumani. On temporal abstractions of web service protocols. In *The 17th Conference on Advanced Information Systems Engineering (CAiSE '05). Short Paper Proceedings*, June 13-17, Porto, Portugal, 2005.

[16] Boualem Benatallah, Fabio Casati, Julien Ponge, and Farouk Toumani. Compatibility and replaceability analysis for timed web service protocols. In *21èmes Journées Bases de Données Avancées (BDA'05)*, October 17-20, Saint Malo, 2005.

[17] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Web service conversation modeling : A cornerstone for e-business automation. *IEEE Internet Computing*, 8(1) :46–54, 2004.

[18] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Representing, analysing and managing web service protocols. *Data Knowedge Engineering*, 58(3) :46–54, 2006.

[19] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Analysis and management of web service protocols. In *Proceedings of the* 23^{rd} *International Conference on Conceptual Modeling (ER'04)*, pages 524–541, November, Shanghai, China, 2004.

[20] Boualem Benatallah, Fabio Casati, Farouk Toumani, and Rachid Hamadi. Conceptual modeling of web service conversations. In *Proceedings of the* 15^{th} *International conference on Advanced Information Systems Engineering (CAiSE'03)*, pages 449–467, Klagenfurt, Austria, June 16-18, 2003.

[21] Boualem Benatallah and Hamid R. Motahari Nezhad. Service oriented architecture : Overview and directions. In *Lipari Summer School*, pages 116–130, July 8-21, Lipari Island, Italy, 2007.

[22] Boualem Benatallah, Quan Z. Sheng, Anne H. H. Ngu, and Marlon Dumas. Declarative composition and peer-to-peer provisioning of dynamic web services. In *Proceedings of the* 18^{th} *International Conference on Data Engineering (CAICDE'02)*, pages 297–308, 26 February-1 March 2002, San Jose, 2002.

[23] Daniela Berardi. *Automatic Service Composition. Models, techniques and tools.* PhD thesis, La Sapienza University, Roma, 2005.

[24] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 613–624. ACM, August 30 - September2, Trondheim, Norway, 2005.

[25] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic composition of e-services that export their behavior. In *Service-Oriented Computing - ICSOC 2003, First International Conference, Trento,*

Italy, December 15-18, 2003, Proceedings, volume 2910 of *Lecture Notes in Computer Science*, pages 43–58. Springer, 2003.

[26] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic composition of e-services that export their behavior. In *Proceedings of the* 1^{st} *International Conference on Service-Oriented Computing (ICSOC'03)*, volume 2910, pages 43–58, December 15-18, Trento, Italy, 2003.

[27] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, and Massimo Mecella. Composition of services with nondeterministic observable behavior. In *Service-Oriented Computing - ICSOC 2005, Third International Conference (ICSOC)*, pages 520–526, 2005.

[28] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, and Massimo Mecella. Composition of services with nondeterministic observable behavior. In *Proceedings of the* 3^{rd} *International Conference on Service-Oriented Computing (ICSOC'05)*, pages 520–526, December 12-15, Amsterdam, The Netherlands, 2005.

[29] Daniela Berardi, Fahima Cheikh, Giuseppe De Giacomo, and Fabio Patrizi. Automatic service composition via simulation. *Int. J. Found. Comput. Sci.*, 19(2) :429–451, 2008.

[30] Piergiorgio Bertoli, Marco Pistore, and Paolo Traverso. Automated web service composition by on-the-fly belief space search. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS'06)*, pages 358–361, June 6-10, Cumbria, UK, 2006.

[31] Aysu Betin-Can, Tevfik Bultan, and Xiang Fu. Design for verification for asynchronously communicating web services. In *Proceedings of the 14th international conference on World Wide Web (WWW'05), year = May 10-14, Chiba, Japan, 2005, pages = 750-759.*

[32] Lucas Bordeaux and Gwen Salaün. Using process algebra for web services : Early results and perspectives. In *Proceedings of the* 5^{th} *International Workshop on Technologies for E-Services (TES'04)*, pages 54–68, August 29-30, Toronto, Canada, 2004.

[33] Patricia Bouyer and François Laroussinie. Vérification par automates temporisés. In *In Nicolas Navet, editor, Systèmes temps-réel 1 : techniques de description et de vérification*, pages 121–150, 2006.

[34] Maroua Bouzid and Antoni Ligeza. Algebraic temporal specifications with extended tus. hierarchical granular terms and their applications. In *Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*, pages 249–253, Hong Kong, China, 14-16 November 2005,.

[35] Antonio Brogi, Carlos Canal, Ernesto Pimentel, and Antonio Vallecillo. Formalizing web service choreographies. *Electr. Notes Theor. Comput. Sci.*, 105 :73–94, 2004.

[36] Antonio Brogi and Razvan Popescu. Towards semi-automated workflow-based aggregation of web services. In *Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC'05)*, pages 214–227, December 12-15, Amsterdam, The Netherlands, 2005.

[37] Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation specification : a new approach to design and analysis of e-service composition. In *Proceedings of the international conference on World Wide Web (WWW'03)*, pages 403–410, May 20-24, Budapest, Hungary, 2003.

[38] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Theoretical aspects of communication-centred programming. *Electron. Notes Theor. Comput. Sci.*, 209 :125–133, 2008.

[39] Fabio Casati and Ming-Chien Shan. Models and languages for describing and discovering e-services. In *SIGMOD Conference*, page 626, 2001.

[40] Piotr Chrzastowski-Wachtel, Boualem Benatallah, Rachid Hamadi, Milton O'Dell, and Adi Susanto. A top-down petri net-based approach for dynamic workflow modeling. In *Proceedings of the International Conference on Business Process Management (BPM'03)*, pages 336–353, Eindhoven, The Netherlands, June 26-27, 2003.

[41] Juan Carlos Corrales, Daniela Grigori, and Mokrane Bouzeghoub. Découverte de services basée sur leurs protocoles de conversation. *Ingénierie des Systèmes d'Information*, 12(1) :9–32, 2007.

[42] Alin Deutsch, Liying Sui, and Victor Vianu. Specification and verification of data-driven web services. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'04)*, pages 71–82, June 14-16, Paris, France, 2004.

[43] Gregorio Díaz, María-Emilia Cambronero, Juan José Pardo, Valentin Valero, and Fernando Cuartero. Automatic generation of correct web services choreographies and orchestrations with model checking techniques. In *Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW'06)*, page 186, 19-25 February, Guadeloupe, French Caribbean, 2006.

[44] Gregorio Díaz, María-Emilia Cambronero, M. Llanos Tobarra, Valentin Valero, and Fernando Cuartero. Analysis and verification of time requirements applied to the web services composition. In *Proceedings of the International Workshop on Web Services and Formal Methods (WS-FM'06)*, pages 178–192, September 8-9, Vienna, Austria, 2006.

[45] Gregorio Díaz, Juan José Pardo, María-Emilia Cambronero, Valentin Valero, and Fernando Cuartero. Automatic translation of ws-cdl choreographies to timed automata. In *Proceedings of the International Workshop on Web Services and Formal Methods (WS-FM'05)*, pages 230–242, September 1-3, Versailles, France, 2005.

[46] Remco Dijkman and Marlon Dumas. Service-oriented design : A multi-viewpoint approach. *Technical Report CTIT Technical Report Series No. 04-09, Centre for Telematics and Information Technology, University of Twente*, The Netherlands, February 2004.

[47] Marlon Dumas, Murray Spork, and Kenneth Wang 0002. Adapt or perish : Algebra and visual notation for service interface adaptation. In *Proceedings of the 4^{th} International Conference on Business Process Management (BPM'06)*, pages 65–80, Vienna, Austria, September 5-7, 2006.

[48] Johann Eder and Amirreza Tahamtan. Temporal conformance of federated choreographies. In *Proceedings of the* 19^{th} *International Conference on Database and Expert Systems Applications(DEXA'08)*, Turin, Italy, September 1-5, 2008.

[49] Wolfgang Emmerich. Software engineering and middleware : a roadmap. In *22nd International Conference on Software Engineering (ICSE'00), Future of Software Engineering Track*, pages 117–129, June 4-11, Limerick Ireland 2000.

[50] Marie-Christine Fauvet, , Marlon Dumas, , Boualem Benatallah, and Hye-Young Paik. Peer-to-peer traced execution of composite services. In *Proceedings of the* 2^{nd} *International Workshop on Technologies for E-Services (TES'01)*, pages 103–117, London, UK, 2001.

[51] Dieter Fensel. Semantic web enabled web services. In *Proceedings of the* 25^{th} *Annual German Conference on Advances in Artificial Intelligence (KI'02)*, pages 319–322, September 16-20, Aachen, Germany, 2002.

[52] Howard Foster. *A Rigorous Approach To Engineering Web Service Compositions*. PhD thesis, Imperial College London, 2006.

[53] Tim Bray Jean Paoli C. M. Sperberg-McQueen François Yergeau, John Cowan and Eve Maler. Extensible markup language (xml) 1.1. *http ://www.w3.org/TR/2004/REC-xml11-20040204/*, 4th February 2004.

[54] Xiang Fu, Tevfik Bultan, and Jianwen Su. Realizability of conversation protocols with message contents. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 96–, June 6-9, San Diego, California, USA, 2004.

[55] Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting bpel web services. In *Proceedings of the 13th international conference on World Wide Web (WWW'04)*, pages 621–630, May 17-20, New York, NY, USA, 2004.

[56] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artif. Intell.*, 121(1-2) :109–169, 2000.

[57] W3C Working Group. Simple object access protocol (soap) 1.1. *http ://www.w3.org/TR/SOAP/*, 2000.

[58] Nawal Guermouche, Salima Benbernou, Emmanuel Coquery, and Mohand-Said Hacid. Privacy-aware web service protocol replaceability. In *IEEE International Conference on Web Services (ICWS), July 9-13, 2007, Salt Lake City, Utah, USA*, pages 1048–1055, 2007.

[59] Nawal Guermouche and Claude Godart. Timed model checking based approach for compatibility analysis of synchronous web services. *INRIA Research report*, 2008.

[60] Nawal Guermouche and Claude Godart. Timed model checking based approach for web services analysis. In *Proceedings of the IEEE International Conference on Web Service (ICWS'09)*, pages 213–221, July 6-10, Los Angeles, CA, USA, 2009.

[61] Nawal Guermouche and Claude Godart. Asynchronous timed web service-aware choreography analysis. In *Proceedings of the* 21^{th} *International Conference on Advanced Information Systems (CAiSE'09)*, pages 364–378, June 8-12, Amsterdam, The Netherlands, 2009.

[62] Nawal Guermouche and Claude Godart. Timed properties-aware asynchronous web service composition. In *Proceedings of the* 16^{th} *International Conference on Cooperative Information Systems (CoopIS'08)*, pages 44–61, November 9-14, Monterrey, Mexico, , 2008.

[63] Nawal Guermouche and Claude Godart. Toward data flow oriented services composition. In *Proceedings of the* 12^{th} *International IEEE Enterprise Distributed Object Computing Conference (EDOC'08)*, pages 379–385, September 15-19, Munich, Germany, 2008.

[64] Nawal Guermouche, Claude Godart, and Boualem Benatallah. Data messaging based approach for web service composition. In *Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE'08)*, pages 449–454, October 22-24, Xi'an, China, 2008.

[65] Nawal Guermouche, Olivier Perrin, and Christophe Ringeissen. Timed specification for web services compatibility analysis. *Electr. Notes Theor. Comput. Sci.*, 200(3) :155–170, 2008.

[66] Nawal Guermouche, Olivier Perrin, and Christophe Ringeissen. Mediation based approach for services composition. In *In* 6^{th} *International Conference on Software Engineering Research, Management and Applications (SERA'08)*, pages 273–280, August 20-22, August 20-22, 2008.

[67] Andrea Ferrara Gwen Salaün and Antonella Chirichiello. Negotiation among web services using lotos/cadp. In *Proceedings of the European Conference on Web Services (ECOWS'04)*, pages 198–212, September 27-30, Erfurt, Germany, 2004.

[68] Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *Proceedings of the* 14^{th} *Australasian Database Conference (ADC'03)*, February, Adelaide, South Australia, 2003.

[69] Rachid Hamadi, Hye-Young Paik, and Boualem Benatallah. Conceptual modeling of privacy-aware web service protocols. In *Proceedings of the* 19^{th} *International Conference on Advanced Information Systems Engineering (CAiSE'07)*, pages 233–248, June 11-15, Trondheim, Norway, 2007.

[70] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech : A model checker for hybrid systems. In *Proceedings of the* 9^{th} *International Conference on Computer Aided Verification (CAV'97)*, pages 460–463, Haifa, Israel, June 22-25, 1997.

[71] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2) :193–244, 1994.

[72] Sebastian Hinz, Karsten Schmidt, and Christian Stahl. Transforming bpel to petri nets. In *Proceedings of the International Conference on Business Process Management (BPM'05)*, pages 220–235, Nancy, France, 2005.

[73] W.M.P. van der Aalst H.M.W. Verbeek. Analyzing bpel processes using petri nets. In *Proceedings of* 2^{nd} *International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 59–78, June 2005.

[74] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison Wesley, November 2000.

[75] Richard Hull, Michael Benedikt, Vassilis Christophides, and Jianwen Su. E-services : a look behind the curtain. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'03)*, pages 1–14, June 9-12, San Diego, CA, USA, 2003.

[76] Matjaz Juric, Poornachandra Sarang, and Benny Mathew. *Business Process Execution Language for Web Services*. Packt Publishing, 2006.

[77] Slim Kallel, Anis Charfi, Tom Dinkelaker, Mira Mezini, and Mohamed Jmaiel. Specifying and monitoring temporal properties in web services compositions. In *Proceedings of the 7th IEEE European Conference on Web Services (ECOWS'09)*, pages 148–157, 9-11 November 2009, Eindhoven, The Netherlands.

[78] Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. Timed modelling and analysis in web service compositions. In *Proceedings of the The First International Conference on Availability, Reliability and Security (ARES'06)*, pages 840–846, April 20-22, Vienna University of Technology, Austria, 2006.

[79] Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. Representation, verification, and computation of timed properties in web service compositions. In *Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 497–504, September 18-22, Chicago, Illinois, USA 2006.

[80] Martin Keen, JinYoung Byun, Mark Grottoli, Li Hui, Ajay Mathur, Robert Skaer, Guru Vasudeva, Gerd Watmann, and Peter Xu. Patterns : Serial and parallel processes for process choreography and workflow. *IBM*, 2004.

[81] Patrick Kellert and Farouk Toumani. Les web services sémantiques. *Web sémantique, Action spécifique 32 CNRS/STIC*, October, 2003.

[82] Mariya Koshkina and Franck van Breugel. Modelling and verifying web service orchestration by means of the concurrency workbench. *SIGSOFT Softw. Eng. Notes*, 29(5) :1–10, 2004.

[83] Ugo Dal Lago, Marco Pistore, and Paolo Traverso. Planning with a language for extended goals. In *AAAI/IAAI*, pages 447–454, 2002.

[84] Cosimo Laneve and Gianluigi Zavattaro. Foundations of web transactions. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS'05)*, pages 282–298, Edinburgh, UK, April 4-8, 2005.

[85] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. In *International Journal on Software Tools for Technology Transfer*, 1997.

[86] Alexander Lazovik, Marco Aiello, and Mike P. Papazoglou. Planning and monitoring the execution of web service requests. In *Proceedings of the* 1st *International*

Conference on Service-Oriented Computing (ICSOC'03), pages 335–350, December 15-18, Trento, Italy, 2003.

[87] Alon Y. Levy. logic-based techniques in data integration. pages 575–591, 2001.

[88] Michele Mancioppi, Manuel Carro, Willem-Jan van den Heuvel, and Mike P. Papazoglou. Sound multi-party business protocols for service networks. In *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC'08)*, pages 302–316, Sydney, Australia, December 1-5, 2008.

[89] Axel Martens. Consistency between executable and abstract processes. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 60–67, 2005.

[90] Axel Martens, Simon Moser, Achim Gerhardt, and Karoline Funk. Analyzing compatibility of bpel processes. In *Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW'06)*, page 147, uadeloupe, French Caribbean, 19-25 February, 2006.

[91] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. Owl-s : Semantic markup for web services. *http ://www.w3.org/Submission/OWL-S/*, 22 November 2004.

[92] David Martin, Massimo Paolucci, Sheila Mcilraith, Mark Burstein, Drew Mcdermott, Deborah Mcguinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara. Bringing semantics to web services : The owl-s approach. In *SWSWPC*, pages 26–24. Springer Berlin Heidelber, 2005.

[93] Manuel Mazzara. Timing issues in web services composition. In *Proceedings of the Formal Techniques for Computer Systems and Business Processes, European Performance Engineering Workshop, and International Workshop on Web Services and Formal Methods, (EPEW/WS-FM'05)*, pages 287–302, Versailles, France, September 1-3, 2005.

[94] Manuel Mazzara and Sergio Govoni. A case study of web services orchestration. In *Proceedings of the 7th International Conference on Coordination Models and Languages (COORDINATION'05)*, pages 1–16, Namur, Belgium, April 20-23, 2005.

[95] Sheila A. McIlraith and Tran Cao Son. Adapting golog for composition of semantic web services. In *Proceedings of the* 8^{th} *International Conference on Principles and Knowledge Representation and Reasoning (KR'02)*, pages 482–496, April 22-25, Toulouse, France,2002.

[96] Kenneth L. McMillan. Symbolic model checking. In *Boston, Kluwer Academic Publishers*, 1993.

[97] Massimo Mecella and Carlo Batini. Enabling italian e-government through a cooperative architecture. *IEEE Computer*, 34 :200–1, 2001.

[98] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing web services on the semantic web. *VLDB J.*, 12 :333–351, 2003.

[99] Robin Milner. *Communication and Concurrency.*

[100] Sonia Ben Mokhtar, Anupam Kaul, Nikolaos Georgantas, and Valérie Issarny. Efficient semantic service discovery in pervasive computing environments. In *7th International Middleware Conference, (Middleware)*, pages 240–259, Melbourne, Australia, November 27-December 1, 2006.

[101] Anca Muscholl and Igor Walukiewicz. A lower bound on web services composition. *CoRR*, abs/0804.3105, 2008.

[102] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the international conference on World Wide Web (WWW'02)*, pages 77–88, May 7-11, Honolulu, Hawaii, USA, 2002.

[103] Hamid R. Motahari Nezhad, Boualem Benatallah, Axel Martens, Francisco Curbera, and Fabio Casati. Semi-automated adaptation of service interactions. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*, pages 993–1002, May 8-12, Banff, Alberta, Canada, 2007.

[104] Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata : Closing a decidability gap. In *Proceedings of the* 19^{th} *IEEE Symposium on Logic in Computer Science (LICS'04)*, pages 54–63, 14-17 July, Turku, Finland, 2004.

[105] Chun Ouyang, Eric Verbeek, Wil M. P. van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H. M. ter Hofstede. Formal semantics and analysis of control flow in ws-bpel. *Sci. Comput. Program.*, 67(2-3) :162–198, 2007.

[106] Chun Ouyang, Eric Verbeek, Wil M. P. van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H. M. ter Hofstede. Wofbpel : A tool for automated analysis of bpel processes. In *Proceedings of the 3^{rd} International Conference on Service-Oriented Computing (ICSOC'05)*, pages 484–489, Amsterdam, The Netherlands, December 12-15.

[107] Michael Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann, and Bernd J. Krämer. Service-oriented computing : A research roadmap. In *Service Oriented Computing (SOC)*, number 05462, 2006.

[108] Chris Peltz. Web services orchestration and choreography. *Web services journal*, 2003.

[109] Marco Pistore, Fabio Barbon, Piergiorgio Bertoli, Dmitry Shaparau, and Paolo Traverso. Planning and monitoring web service composition. In *Proceedings of the 11^{th} International Conference on Artificial Intelligence : Methodology, Systems, and Applications (AIMSA'04)*, pages 106–115, September 2-4, Varna, Bulgaria, 2004.

[110] Marco Pistore, Annapaola Marconi, Piergiorgio Bertoli, and Paolo Traverso. Automated composition of web services by planning at the knowledge leve. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 1252–1259, July 30-August 5, Edinburgh, Scotland, UK, 2005.

[111] Marco Pistore, Paolo Traverso, Piergiorgio Bertoli, and Annapaola Marconi. Automated synthesis of composite bpel4ws web services. In *IEEE International Conference on Web Services (ICWS'05)*, pages 293–301, July 11-15, Orlando, FL, USA, 2005.

[112] Amir Pnueli. The temporal logic of programs. In *18^{th} Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57, Providence, Rhode Island, USA, 31 October-2 November,1977.

[113] Julien Ponge. A new model for web services timed business protocols. In *Atelier Conception des systèmes d'information et services Web (SIWS-Inforsid)*, 2006.

[114] Julien Ponge. *Model based Analysis of Time-aware Web Services Interaction*. PhD thesis, Université Blaise Pascal-Clermont-Ferrand, 2008.

[115] Julien Ponge, Boualem Benatallah, Fabio Casati, and Farouk Toumani. Fine-grained compatibility and replaceability analysis of timed web service protocols. In *the 26th International Conference on Conceptual Modeling (ER'07)*, pages 599–614, November 5-9, Auckland, New Zealand, 2007.

[116] Raymond Reiter. *Knowledge in action : logical foundations for specifying and implementing dynamical systems*. MIT Press, Cambridge, Mass., 2001. The frame problem and the situation calculus.

[117] Gwen Salaün, Lucas Bordeaux, and Marco Schaerf. Describing and reasoning on web services using process algebra. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 43–, June 6-9, San Diego, California, USA, 2004.

[118] R. Schantz and D. Schmidt. Middleware for distributed systems : Evolving the common structure for network-centric applications. *Encyclopedia of Software Engineering*, Wiley and Sons, 2002.

[119] Karsten Schmidt. Lola : A low level analyser. In *Proceeding of the 21st International Conference on Application and Theory of Petri Nets (ICATPN'00)*, pages 465–474, Aarhus, Denmark, June 26-30, 2000.

[120] Tran Cao Son Sheila A. McIlraith and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16 :46–53, 2001.

[121] Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing : Semantics, Processes, Agents*. Wiley Computer Publishing, New york, 2005.

[122] Craig A. Knoblock Snehal Thakkar and José Luis Ambite. A view integration approach to dynamic composition of web services. 2003.

[123] Shirin Sohrabi, Nataliya Prokoshyna, and Sheila A. McIlraith. Web service composition via generic procedures and customizing user preferences. In *Proceedings of the 5th International Semantic Web Conference (ISWC'06)*, pages 597–611, November 5-9, Athens, GA, USA, 2006.

[124] Biplav Srivastava and Jana Koehler. Web service composition - current solutions and open problems. In *Proceedings of the ICAPS Workshop on Planning for Web Services*, pages 28–35, 2003.

[125] OASIS Standard. Simple object access protocol (soap) 1.1. *http ://www.oasis-open.org/committees/uddi-spec*, 2008.

[126] Yu Tang, Luo Chen, Kai-Tao He, and Ning Jing. Srn : An extended petri-net-based workflow model for web service composition. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 591–599, San Diego, California, USA, June 6-9, 2004.

[127] Snehal Thakkar, José Luis Ambite, and Craig A. Knoblock. A data integration approach to automatically composing and optimizing web services. *Proceedings of the 2nd ICAPS International Workshop on Planning and Scheduling for Web and Grid Services*, 2004.

[128] Snehal Thakkar, José Luis Ambite, Craig A. Knoblock, and C Shahabi. Dynamically composing web services from on-line sources. *Proceeding of the AAAI Workshop on Intelligent Service Integration*, pages 1–7, 2002.

[129] Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. *Inf. Process. Lett.*, 99(6) :222–226, 2006.

[130] Wil M. P. van der Aalst. Challenges in business process management : Verification of business processing using petri nets. *Bulletin of the EATCS*, 80, 2003.

[131] Steve Vinoski. An overview of middleware. In *Proceedings of the* 9^{th} *International Conference on Reliable Software Technologies (Ada-Europe'04)*, pages 35–51, June 14-18, Palma de Mallorca, Spain, 2004.

[132] W3C. Web service description language (wsdl. *http ://www.w3.org/TR/WSDL/*, 2001.

[133] Petia Wohed, Wil M.P. van der Aalst, Marlon Dumas, and Arthur H.M. ter Hofstede. Analysis of web services composition languages : The case of BPEL4WS. In *Conceptual Modeling - ER 2003*, volume 2813, pages 200–215, 2003.

[134] Yanping Yang, QingPing Tan, Yong Xiao, Jinshan Yu, and Feng Liu. Exploiting hierarchical cp-nets to increase the reliability of web services workflow. In *International Symposium on Applications and the Internet (SAINT'06)*, pages 116–122, January 23-27, Phoenix, Arizona, USA, 2006.

[135] Xiaochuan Yi and Krys Kochut. A cp-nets-based design and verification framework for web services composition. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 756–760, June 6-9, San Diego, California, USA, 2004.

[136] Sergio Yovine. Kronos : A verification tool for real-time systems. (kronos user's manual release 2.2). *International Journal on Software Tools for Technology Transfer*, 1 :123–133, 1997.

[137] Jia Zhang, Jen-Yao Chung, Carl K. Chang, and Seongwoon Kim. Ws-net : A petri-net based specification model for web services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 420–427, San Diego, California, USA, June 6-9, 2004.

Résumé

L'avantage majeur qu'offrent les services Web est le fait qu'ils reposent sur des standards et les technologies du Web pour interagir en s'échangeant des messages. A part les séquences de messages, d'autres facteurs affectent l'interopérabilité des services Web, telles que les contraintes temporelles qui spécifient les délais nécessaires pour échanger des messages.

La thèse rapportée dans ce manuscrit étudie l'impact de ces propriétés dans la composition de services Web. La considération de telles propriétés soulève plusieurs problèmes auxquels on a essayé d'apporter une solution. Le premier aspect consiste à définir un modèle qui tienne compte des abstractions nécessaires afin de pouvoir analyser et synthétiser une composition, à savoir les messages, les données, les contraintes de données, les propriétés temporelles et l'aspect asynchrone des communications des services. En se basant sur ce modèle, le deuxième problème consiste à proposer une approche d'analyse de compatibilité. Cette analyse vise à caractériser la compatibilité ou la non-compatibilité des services Web et ce en prenant en considération les abstractions précédemment citées. Nous étudions particulièrement l'impact des propriétés temporelles dans une chorégraphie dans laquelle les services Web supportent des communications asynchrones. Nous proposons une démarche basée sur le model checking qui permet de détecter les éventuels conflits temporisés qui peuvent surgir dans une chorégraphie. Finalement, le dernier problème auquel nous nous intéressons est celui de la construction d'une composition qui essaie de répondre au besoin du client et ce en prenant en compte les aspects temporels. L'approche que l'on propose est basée sur la génération d'un médiateur pour essayer, quand c'est possible, de contourner les incompatibilités temporisées et non-temporisées qui peuvent surgir lors d'une collaboration. Des mécanismes et des algorithmes ont été développés afin de mettre en oeuvre ces objectifs.

Mots-clés: Services Web temporisés, composition de services Web, propriétés temporelles, protocoles de conversations temporisées asynchrones, analyse de compatibilité temporisée, Chorégraphie de services Web

Abstract

Web services are based on standrards and Web technologies to interact by exchanging messages. Apart from the sequences of messages, other factors affect the interoperability of Web services, such as temporal constraints that specify the required delay to exchange messages. This thesis studies the impact of these properties in the composition of Web services. The consideration of such properties raises several problems that need several investigations. The first aspect is to define a formal model that takes into account the necessary abstractions in order to analyze and synthesize a composition. The abstractions we have considered are : messages, data, data constraints, temporal properties and the asynchronous nature of services communications. Based on this model, the second problem we handled is the compatibility analysis. This later aims at characterizing the compatibility or incompatibility of Web services by taking into account the abstractions mentioned above. We propose an approach based on model checking to detect the conflicts that may arise in timed choreography. Finally, the last problem we dealt with is the construction of a composition which attempts to satisfy the client need. The approach that we propose is based on the generation of a mediator to try, whenever possible, to resolve the incompatibilities that may arise during a collaboration. Mechanisms and algorithms have been developed to implement these primitives.

Keywords: Web services, Web services composition, timed properties, asynchronous timed conversations protocol, timed compatibility analysis