Un cadre sémantique formel pour la description, sélection et composition des services web

Manel Amel Djenouhat

École Doctorale d'Informatique, Télécommunications et Electronique

Centre d'étude et de recherche en informatique et communications

Laboratoire Cédric, Équipe VESPA

Département des Technologies des Logiciels et des Systèmes d'Information

Laboratoire d'Informatique Répartie, Équipe GLSD

THÈSE présentée par :

Manel Amel DJENOUHAT

soutenue le : **23 octobre 2017**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline/ Spécialité : Informatique

Un Cadre Sémantique Formel pour la Description, Sélection et Composition des Services Web

THÈSE dirigée par :

M.BARKAOUI Kamel	Professeur des Universités, Cnam
Mme.BELALA Faïza	Professeur, Université Constantine 2

RAPPORTEURS :

M. OUSSALAH Mourad	Professeur des Universités, Universités de Nantes

---

JURY :

Mme. BOUFAIDA Zizette	Professeur, Université Constantine 2
M. BELLATRECHE Ladjel	Professeur des Universités, ENSMAA, Poitiers
M. MEZIOUD Chaker	Maître de Conférences A, Université Constantine2

Dédicace

A Mes Très Chers Parents

Remerciements

Je remercie Dieu le Tout Puissant de m'avoir donné la force de mener à terme ce travail.

Je remercie Madame Zizette Boufaida professeur à l'université de Constantine 2 de m'avoir fait honneur en acceptant de présider ma soutenance ainsi que Messieurs : Ladjel Bellatreche, Oussalah Mourad et Mezioud Chaker de juger ce modeste travail.

Je tiens à remercier mes Directeurs de thèse Madame Faiza Belala professeur à l'université de Constantine 2 et Monsieur Kamel Barkaoui professeur des universités au Conservatoire National des Arts et Métiers de Paris, pour la confiance qu'ils m'ont accordée en acceptant d'encadrer ce travail doctoral ainsi que pour l'intérêt dont ils ont fait preuve envers ma recherche.

Je tiens à leur exprimer ma gratitude pour m'avoir guidée, encouragée, conseillée et soutenue pendant toute l'élaboration de cette thèse. Je leur serai toujours reconnaissante pour la patience, la disponibilité ainsi que la gentillesse et la générosité qu'ils ont témoignées à mon égard.

Une pensée particulière à mon professeur Monsieur Hacène Sebih qui a guidé mes premiers pas vers la recherche, qu'il trouve ici l'expression de mon profond respect.

Mes remerciements vont aussi, à tous ceux qui m'ont soutenue et contribué de près ou de loin à l'élaboration de ce travail.

Enfin, ma reconnaissance va à ceux qui ont assuré le soutien affectif : mes chers parents et toute ma famille.

Résumé

L'avènement d'Internet et la maturité des architectures logicielles ont donné l'idée de coupler le monde transactionnel à celui des composants pour créer la technologie des services Web dont l'impact industriel et commercial semble très prometteur. En effet, pour faire face à la complexité croissante des logiciels, les architectures traditionnelles ont été contraintes de sortir du contexte monolithique pour migrer vers un nouveau style architectural plus adapté permettant aux composants d'évoluer dans des environnements hétérogènes.
Le principe essentiel de l'approche service est de transformer le Web en un dispositif distribué d'échange et de calcul où les services Web peuvent interagir d'une manière intelligente. L'architecture sous-jacente aux services Web, connue sous l'acronyme SOA (Service Oriented Architecture), repose sur un ensemble de standards ouverts pour décrire (WSDL), découvrir (UDDI), invoquer (SOAP) et composer (BPEL4WS) les services Web. Cependant, cette infrastructure de base ne résout pas tous les problèmes et reste impuissante face aux exigences de flexibilité imposées par le changement continu de l'environnement dans lequel évoluent ces services.
Le but de cette thèse est de dégager un cadre sémantique formel approprié supportant l'interopérabilité de différents formalismes déjà utilisés pour décrire et déployer un service Web.
En d'autres termes, nous contribuons au développement d'un formalisme mathématique rigoureux permettant de décrire un service Web complexe susceptible de changer pendant l'exécution et de coordonner avec les autres services de façon adaptative. Pour atteindre cet objectif, les étapes de description, de sélection et de composition ont constitué les trois majeures problématiques étudiées dans cette thèse.
Pour ce faire, nous proposons dans un premier temps, à travers l'utilisation du cadre sémantique formel K le langage **KWSDL**, un langage de description de services Web qui s'inspire du standard WSDL, plus riche grammaticalement et surtout doté d'une sémantique formelle. En effet, cette contribution est double ; d'une part, elle fournit un langage avec une sémantique opérationnelle en terme de règles de réécriture qui demeure absente au niveau de WSDL et d'autre part, grâce au cadre K, la description qui en découle peut être exécutable et analysable sous Maude du fait que le passage automatique

est assuré en transparence entre ces deux environnements.
Nous introduisons, dans un second temps, **WS-Sim**, une nouvelle approche basée sur la théorie des catégories visant à optimiser le processus de découverte et de sélection des services Web. En effet, en réduisant les problèmes de redondances qui surgissent au cours de l'exécution en raison des similitudes perçues entre les services, les processus d'invocation et de composition se voient avantagés.
WS-Sim évalue l'équivalence comportementale entre services en représentant chaque service par une catégorie et en établissant des liens formels (foncteur) entre elles. L'équivalence comportementale entre deux services est déduite du degré de préservation des objets et morphismes qui constituent ces catégories. En conséquence, notre approche fournit une solution formelle pour la substitution et le remplacement de services en cas de panne. Elle est également utile pour la planification du processus de composition en fournissant les combinaisons de services adéquates.
Enfin, nous consacrons la majeure partie de notre travail à l'étape de composition. Nous présentons dans cette partie le modèle **RMop-ECATNet** (Refined Meta Open ECATNet) : un modèle dédié à la spécification formelle de la composition des services Web. Il est le résultat du raffinement du modèle Mop-ECATNet proposé par [LB14]. L'intérêt d'utiliser ce modèle comme support de base revient à son pouvoir de contrôler en temps réel (dynamiquement) la composition des services Web ainsi que leurs différentes interactions. Nous avons donc étendu et enrichi ce dernier aux trois niveaux : au niveau structurel, par la mise en place de nouvelles structures compactes et abstraites réduisant considérablement le niveau de complexité du graphe de composition. Au niveau comportemental, en établissant de nouvelles stratégies de contrôle et en mettant en œuvre des patrons réutilisables pour traiter des cas particuliers. Enfin, au niveau implémentation, grâce aux notions de la méta-modélisation MDA nous avons conçu un méta-modèle pour ce formalisme qui sert de base à sa transformation automatique d'une spécification graphique à une spécification Maude textuelle soutenue par les outils ATL et Acceleo.

Toutes ces contributions constituent notre cadre sémantique formel dédié aux services Web. Ce cadre regroupe des modèles mathématiques exécutables et automatisables pouvant être évalués et vérifiés dans un environnement formel riche en outils performants tel que Maude.

Mots Clés :
Services Web, Architecture Orientée Service, Composition dynamique des services, Sélection des services, Transformation de modèles, Logique de réécriture, Cadre sémantique K, Mop-ECATNets.

Abstract

The advent of the Internet and the maturity of software architectures have given the idea of coupling the transactional and components worlds to create the Web services technology whose industrial and commercial impact looks very promising. Indeed, to cope with the increasing complexity of software, traditional architectures were forced out of the monolithic context to migrate to a new architectural style more suitable for components to operate in heterogeneous environments.
The main principle of the service approach is to transform the Web into a distributed device of exchange and computation where Web services can interact in a smart way. The Web services underlying architecture called SOA (Service Oriented Architecture is based on a set of open standards for describing (WSDL), finding (UDDI), invoking (SOAP) and composing (BPEL4WS) the Web services. However, this basic infrastructure does not solve all problems and remains powerless against the flexibility requirements of the constant change of the environment in which these services evolve.
The aim of this thesis is to provide a suitable formal semantic framework that supports interoperability of different formalisms already used to describe and deploy a Web service. In other words, we contribute to the development of a rigorous mathematical formalism to describe a complex Web service that may change during execution and coordinate with other services adaptively.
To achieve this goal, the steps of description, selection and composition constitute the three major issues studied in this thesis.
We propose so, initially, through the use of the K semantic framework the **KWSDL** language : a Web services description language inspired by the WSDL standard, rich grammatically and especially endowed with a formal semantics. Indeed, this contribution is twofold, on the one hand, it provides a language with an operational semantics in terms of rewriting rules which remains absent in WSDL and, on the other hand, thanks to the K framework, the resulting description can be executed and analyzed in Maude knowing that the automatic transition between the two environments is achieved in a transparent way.
We introduce, in a second step, **WS-Sim**, a new approach based on the category theory to automate the service selection process. Indeed, by reducing the problems of duplication

that arise during execution because of the perceived similarities between services, the process of invocation and composition are being favored.WS-Sim evaluates the behavioral equivalence between services by representing each service by a category and by establishing formal links (functor) between them. The behavioral equivalence between two services is deduced from the degree of preservation of objects and morphisms that constitute these categories.
Therefore, our approach provides a formal solution for substitution and replacement of service in case of failure. It is also helpful to plan the composition process by providing sets of adequate services.
Finally, we dedicate our major attention to the composition stage. We propose, in this section, **RMop-ECATNet** (Refined Meta Open ECATNet) : a formal model for the specification of services composition. It is the result of the refinement of the Mop-ECATNets model, introduced initially by [LB14]. The advantage of using this model amounts to its power of controlling dynamically services interactions and composition.
Our approach extends and enriches this model at three distinct levels : at the structural level, we introduce new compact and abstract structures that reduce considerably the complexity of the composition graph. At the behavioural level, we provide new control strategies and implement reusable patterns to deal with some special situations. Finally, at the implementation level, and thanks to the concepts of meta-modelling MDA, we design a meta-model to this formalism which serves as a basis to transform automatically a graph specification into a Maude textual specification (supported by ATL and Acceleo tools).
All these contributions constitute our services formal semantic framework. This framework includes executable and automatizable mathematical models that can be checked and analysed in a rich and formal environment endowed by powerful tools such as Maude.

Keywords :
Web Services, Service-Oriented Architecture, Dynamic Service Composition, Service Selection, Model Transformation, Rewrite Logic, K Framework, Meta-Modelisation, Meta Open ECATNets.

ملخص

لقد أدى ظهور شبكة الإنترنت ونضج هندسة البرامج إلى بزوغ فكرة اقران عالم المواصلات مع عالم المكونات. وهذا لتكوين ما يسمى بتكنولوجيا خدمات الويب التي لها تأثيرا جد واعد على المجال الصناعي والتجاري .
لمواجهة التعقيدات المتزايدة للبرامج، اضطرت الهندسات التقليدية للخروج من السياق الموحد والانتقال إلى نمط هندسي جديد أكثر ملائمة للمكونات للعمل والتطور في بيئات متباينة.
المبدأ الرئيسي لمنهج خدمة الويب هو تحويل الويب إلى جهاز موزع للتبادل والحساب، حيث يمكن لخدمات الويب أن تتفاعل بذكاء.
الهندسة الأساسية لخدمات الويب، المعروفة باسم SOA (خدمة الهندسة الموجهة)، تستند على مجموعة من المعايير المفتوحة لوصف (WSDL)، العثور (UDDI)، استدعاء (SOAP) وتكوين (BPEL4WS) خدمات الويب.
ومع ذلك، فإن هذه البنية الأساسية لا تحل كل المشاكل، وتبقى عاجزة أمام متطلبات المرونة التي يفرضها التغير المستمر في البيئة التي تحوي هذه الخدمات.
الهدف من هذا البحث هو تحديد إطار دلالي مناسب يتقبل التعامل مع مختلف الشكليات المستخدمة لوصف ونشر خدمة الويب.
بعبارة أخرى، نساهم في تطوير شكلية رياضية قادرة على تمثيل خدمة ويب معقدة بإمكانها التغير أثناء التنفيذ والتنسيق مع خدمات أخرى بتكيف.
ولتحقيق هذا الهدف، تمثل كل من خطوات الوصف، الاختيار والتكوين الإشكاليات الثلاث الرئيسية المدروسة في هذه الأطروحة.
لذلك، في البداية، ومن خلال استخدام الإطار الدلالي Kنقترح ***KWSDL***: لغة وصف خدمات ويب مستوحاة من *WSDL* أكثر ثراء نحويا وتمتلك بشكل خاص دلالة رسمية.
هذا الاقتراح يتكون من شقين، من جهة، يوفر لغة تحتوي على دلالات تشغيلية ترتكز على قواعد إعادة الكتابة والتي تعد غائبة في *WSDL*. ومن جهة أخرى، بفضل الإطار Kيمكن تنفيذ وتحليل المواصفة الناتجة عنها تحت *Maude*حيث أن الانتقال التلقائي يتم بشفافية بين هاتين البيئتين.
في خطوة ثانية، نقوم باقتراح ***WS-Sim***، منهج جديد يرتكز على نظرية الفئات التي تهدف لأتمتة عملية اختيار الخدمات .وبالتالي، فإن الحد من مشاكل الازدواجية والغياب التي تنشأ أثناء التنفيذ بسبب التشابه بين الخدمات، يحسن عمليات احتضار وتركيب خدمات الويب.
*WS-Sim*تقيم التكافؤ السلوكي بين الخدمات وذلك عبر تمثيل كل خدمة بفئة معينة وإنشاء روابط رسمية (Foncteur) بينها. يتم استخلاص التكافؤ السلوكي بين خدمتين من درجة الحفاظ على العناصروالشكلية التي تكون هذه الفئات.
لذا، فإن منهجنا يوفر حلا نهائيا لاستبدال خدمة وتعويضها في حالة حدوث عطب. كما أنه مفيد في التخطيط لعملية التركيب من خلال تقديم مزيج من الخدمات المناسبة.
وأخيرا، فإننا خصصنا معظم جهدنا لمرحلة التركيب. نقدم في هذا الجزء نموذج *RMop-ECATNet* : نموذج مخصص للوصف الرسمي للتركيب الديناميكي لخدمات الويب. هذا النموذج هو نتيجة صقل النموذج *Mop-ECATNet* الذي تم عرضه من قبل [LB14]. ميزة استخدام هذا النموذج كقاعدة دعم ترجع لقوته على التحكم في وقت لحظي في التركيب الديناميكي للخدمات ومختلف تفاعلاتها.
لذلك قمنا باغناء النموذج على ثلاث مستويات: على المستوى الهيكلي، بانشاء هياكل جديدة مدمجة ومجردة تحد بشكل كبير من تعقيد الرسم البياني لعملية التركيب. على المستوى السلوكي، بتقديم استراتجيات رقابة جديدة وانماط تستخدم مباشرة في حالات معينة مع امكانية اعادة استعمالها.
واخيرا، على المستوى التنفيذي، بفضل مفاهيم النمذجة الافقية قمنا بتحويل النموذج *RMop-ECATNet* تلقائيا من نموذج شكلي الى مواصفة *Maude* نصية بواسطة ادوات التحويل ATL وAcceleo.

الكلمات الرئيسية :نموذج، تركيب الخدمات ، إنتقاء خدمات الواب ، شبكات بيتري، تحويل النماذج، نموذج فوقي، منطق إعادة الكتابة

Table des figures

Liste des tableaux

Abréviations

Acronyme	*Nom complet*
BPEL4WS	Business Process Execution Langage For Web Services
COCOA	A system for doing Computations in Commutative Algebra
ECATNets	Extended Concurrent Algebraic Term Nets
HTTP	HyperText Transport Protocol
KWSDL	Web Service Description Language in K
MDA	Model Driven Architecture
Mop-ECATNets	Meta-Open- Extended Concurrent Algebraic Term Nets
OMG	Object Management Group
Op-ECATNets	Open-Extended Concurrent Algebraic Term Nets
OWL	Web Ontology Language
OWL-S	Ontology Web Language for Services
QoS	Quality of Service
RDF	Resource Description Framework
SAWSDL	Semantic Annotation for WSDL and XML Schema
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description Directory and Integration
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WSCI	Web Service Choreography Interface
WSCL	Web Services Conversation Language
WSDL	Web Services Description Language
WSFL	Web Services Flow Languages
XML	eXtended Markup Language

Table des matières

Introduction Générale

0.1 Contexte et Problématique

L'essor des Nouvelles Technologies de l'Information et des Télécommunications NTIC permet d'entrevoir de nouvelles solutions pour le développement d'architectures distribuées qui laisse présager une véritable révolution dans les domaines industriel et commercial. En effet, la maturité des architectures logicielles, l'usage des technologies du Web et la pression de l'économie mondiale mènent à une course effrénée vers la conception et la distribution de composants logiciels autonomes sur le Web.
Définis comme des «services Web», ces composants logiciels sont indépendants et interconnectés ; ils fournissent leurs fonctionnalités sous forme de services disponibles sur le Web et collaborent dans le but de réaliser un ensemble d'objectifs.
Le paradigme architectural SOA constitue le support principal des applications à base de service en dépit de l'hétérogénéité des environnements et des technologies impliqués. Cette architecture est venue contourner les inconvénients que présentent certaines technologies telles que : DCOM [HK97], Java RMI [Dow98], CORBA [SF00]. Ces technologies n'ont pas pu passer à l'échelle à cause de leur dépendance à leur contexte. En effet, leurs composants sont fortement couplés ce qui ne rend leur exploitation fructueuse que dans le contexte ambiant ; d'autant plus, que les techniques protocolaires du Web restent étrangères et peu maitrisées par ces dernières. La technologie des services Web vient controverser ces principes et s'impose comme une révolution dans le domaine industriel et commercial. Donc, pour tirer profit de ces technologies, il fallait changer d'optique en faisant un travail stratégique sur la manière de les coupler au Web. Cette initiative était la préoccupation majeure des chercheurs du domaine.
Les composants sur le Web, plus précisément les «services Web», ont vu le jour à partir des années 2000. Ils se caractérisent par une indépendance totale vis-à-vis des plateformes sur lesquelles ils sont implantés ainsi que par leur faible couplage et témoignent surtout d'une forte capacité d'intégration et de réutilisation. Les services Web sont décrits et localisés dans un cadre de travail spécifique basé sur un ensemble de standards représentés actuellement par un modèle à trois couches du processus de fonctionnement global.
Ce modèle s'articule autour de trois standards : un premier offrant un protocole de commu-

nication permettant de structurer les messages échangés entre services «SOAP»[MPD+02], un second décrivant leur Interfaces «WSDL» [CCM+01] et un troisième déterminant une spécification de leur publication et localisation «UDDI» [CDK+02]. Cependant, bien qu'ils aient toutes ces caractéristiques à leur avantage, il n'en demeure pas moins que de nouveaux problèmes ont émergé depuis leur apparition, notamment, en ce qui concerne la complexité et la dynamicité du Web qui impose le maintien d'une certaine flexibilité pas toujours facile à maitriser.
Les services Web possèdent un cycle de vie dont les phases de développement sont intrinsèquement liées. Cela suppose que l'impact d'un problème sur une étape se répercute sur les suivantes. De ce fait, il est recommandé de localiser et de traiter chaque problématique posée à temps afin d'écarter tout comportement équivoque. Parmi les problèmes qui affectent la technologie des services Web, nous pouvons citer par étapes :

Description :

- Un manque de sémantique dans les descriptions standards qui empêche la coordination entre services et crée une dichotomie dans le processus d'interaction.
- Une insuffisance d'informations concernant le service dans sa description WSDL fournie, ce qui reflète mal ses fonctionnalités et sa qualité à répondre aux requêtes.

Découverte et Sélection :

- Une mauvaise gestion du registre de stockage des services UDDI dont la localisation n'est pas publique.
- Un manque d'informations sur le service, fournies par les standards lors de l'appariement entre requête et service.
- Une absence de mécanismes d'automatisation du processus.
- Des redondances de services exposant les mêmes services avec les mêmes fonctionnalités.
- Une sélection de services souvent réalisée en se basant sur des données d'interface et non sur le comportement et la qualité du service invoqué.

Composition :

- Une mauvaise spécification du comportement du service composite : les langages de composition standards établis s'inspirent et s'appuient sur des concepts purs de programmation.
- La non maîtrise de la vulnérabilité des services face aux exigences de leur environnement dynamique.
- La non gestion de la reconfiguration dynamique des services en cas de problème au cours de l'exécution.

Vérification :

- L'emploi de méthodes semi-formelles telles que la simulation pour vérifier les modèles de services.
- La non compatibilité avec des environnements formels pour analyser et exécuter les modèles.

Toutes les phases :

- L'obligation de se connecter au Web pour invoquer un service Web.

— L'évolution constante et fulgurante du Web qui freine les techniques d'adaptation.

Dans ce travail, nous nous focalisons principalement sur les phases de description, sélection et composition de services. La problématique étudiée dans cette thèse se résume en trois principales questions chacune d'elles relève d'une des étapes sus-citées :

Comment faire coordonner des services Web sans ambiguïtés ?

WDSL joue un rôle prépondérant dans le processus de description de services Web ; en plus de donner une description de l'interface d'un service, il doit garantir que cette description assure son interopérabilité avec d'autres services vu la vocation d'universalité qui leur est associée. La description abstraite donnée par ce langage n'est pas suffisante pour spécifier des composants complexes. En effet, elle est inappropriée car ces services nécessitent des modèles d'échange plus complexes. Cette problématique impose donc la recherche d'un moyen qui soit efficace et rigoureux pour associer une sémantique formelle à ce type de langages.

Comment sélectionner le service adéquat ou planifier une éventuelle composition de services ?

Actuellement, de nombreux services Web avec des fonctionnalités similaires sont publiés par des fournisseurs concurrents ; ceci affecte et complique le processus de sélection. En effet, plusieurs services Web découverts sur le registre UDDI peuvent combler les besoins de l'utilisateur. Mais pour quel service opter ? Il devient donc nécessaire d'utiliser un mécanisme de jumelage permettant d'évaluer les services avant de les sélectionner et d'en choisir les plus pertinents.
L'optimisation de cette phase peut être utile, non seulement, pour fournir une réponse optimale à la demande du client, mais aussi, pour trouver toutes les opérations de services substituables appropriées qui peuvent être utilisées lors de la reconfiguration dynamique des services ayant échoué. Elle est également adaptée pour planifier une éventuelle composition qui puisse fournir les combinaisons adéquates des services à composer.

Comment spécifier le comportement d'un service composite et surveiller son évolution en tenant compte de la dynamicité de son environnement ?

La composition dynamique des services Web est un domaine de recherche difficile pour la diffusion d'applications orientées services. Elle suscite, de plus en plus, l'attention des chercheurs. Un intérêt particulier est accordé aux principales questions relatives à la conception et à la reconfiguration des applications basées sur les services qui évoluent dans un environnement dynamique. Une compréhension du comportement des services Web complexes et dynamiques est incontournable.
En effet, les services Web composites s'exécutent dans un environnement versatile où le nombre de services disponibles évolue très rapidement. Satisfaire les objectifs de

cette tâche particulière implique la maitrise de formalismes avancés pouvant fournir une spécification comportementale abstraite et concise avec une surveillance permanente des changements environnementaux et des éventuels besoins pouvant surgir. Ces formalismes doivent être dotés de stratégies efficaces pour la reconfiguration et le remplacement de services en cas de pannes.

0.2 Motivations et Objectifs

Le but de cette thèse est de dégager un cadre sémantique formel approprié supportant l'interopérabilité de différents formalismes déjà utilisés pour décrire et déployer un service Web.
En d'autres termes, nous contribuons au développement d'un formalisme mathématique rigoureux permettant de décrire un service Web complexe susceptible de changer pendant l'exécution et de coordonner avec les autres services de façon adaptative.
En effet, les techniques existantes qui ont montré des résultats probants, telles que : la logique de réécriture, les réseaux de Petri ont été ré-exploitées et étendues dans le but de développer ce cadre formel. Celui-ci tend à fournir :

— Une nouvelle description alternative à celle de WSDL dotée d'une sémantique formelle pour faciliter l'interopérabilité entre services Web lors du processus d'interaction.
— De nouveaux mécanismes tendant à automatiser le processus de sélection, s'appuyant sur des descriptions formelles de services et des stratégies de sélection pour trouver le service adéquat répondant à une requête précise ainsi que pour planifier des combinaisons de services pour d'éventuelles compositions.
— Une spécification comportementale formelle de services Web composites en respectant leur nature volatile et en tenant compte de la dynamicité de leur environnement.
— L'analyse et l'exécution automatique des modèles conçus à travers l'adoption des techniques de méta-transformation.

0.3 Contributions

Afin d'atteindre les objectifs sus-cités, nous proposons dans un premier temps, à travers l'utilisation du cadre sémantique formel K le langage **KWSDL** : Un langage de description de services Web qui s'inspire du standard WSDL. Plus riche grammaticalement et surtout doté d'une sémantique formelle. Cette contribution est double. D'une part, elle fournit un langage avec une sémantique opérationnelle en terme de règles de réécriture qui demeure absente au niveau de WSDL et d'autre part, grâce au cadre K, la description qui en découle peut être exécutable et analysable sous Maude du fait que le passage est assuré automatiquement et en transparence entre ces deux environnements.

Nous introduisons, dans un second temps, **WS-Sim**, une nouvelle approche basée sur la théorie des catégories visant à améliorer le processus de découverte et de sélection de services. En effet, en réduisant les problèmes de redondances qui surgissent au cours de l'exécution en raison des similitudes perçues entre les services, les processus d'invocation et de composition se voient avantagés.
WS-Sim évalue l'équivalence comportementale entre services en représentant chaque service par une catégorie et en établissant des liens formels (foncteur) entre elles. L'équivalence comportementale entre deux services est déduite du degré de préservation des objets et des morphismes qui constituent ces catégories.
En conséquence, notre approche fournit une solution formelle pour la substitution et le remplacement de services en cas de panne. Elle est également utile pour planifier le processus de composition en fournissant les combinaisons de services adéquates.
Enfin, nous consacrons la majeure partie de notre travail à l'étape de composition. Nous présentons dans cette partie le modèle **RMop-ECATNet** (Refined Meta Open ECATNet) : un modèle dédié à la spécification formelle de la composition des services Web. Il est le résultat du raffinement du modèle Mop-ECATNet proposé par [LB14]. L'intérêt d'utiliser ce modèle comme support de base revient à son pouvoir de contrôler en temps réel (dynamiquement) la composition des services Web ainsi que leurs différentes interactions. Nous avons donc étendu et enrichi ce dernier aux trois niveaux : au niveau structurel, par la mise en place de nouvelles structures compactes et abstraites réduisant considérablement le niveau de complexité du graphe de composition. Au niveau comportemental, en établissant de nouvelles stratégies de contrôle et en mettant en œuvre des patrons réutilisables pour traiter des cas particuliers. Enfin, au niveau implémentation, grâce aux notions de la méta-modélisation MDA nous avons conçu un méta-modèle pour ce formalisme servant de base à sa transformation automatique d'une spécification graphique à une spécification Maude textuelle soutenue par les outils ATL et Acceleo.
Toutes ces contributions constituent notre cadre sémantique formel dédié aux services Web. Ce cadre regroupe des modèles mathématiques exécutables et automatisables pouvant être évalués et vérifiés dans un environnement formel riche en outils performants tel que Maude.

0.4 Structure du Manuscrit

Nous introduisons, en premier lieu, les différentes problématiques soumises à l'étude dans cette thèse en les situant chacune dans son contexte. Ensuite, nous présentons les différents objectifs que nous nous sommes fixés au début de notre recherche et enfin, nous exposons les trois contributions réalisées.
Outre l'introduction, ce manuscrit comprend huit chapitres. Etant donné que les trois problématiques posées sont relatives à trois phases différentes du cycle de vie d'un service Web. Nous consacrons trois chapitres à l'état de l'art :
Le **chapitre 1** est dédié à la description des services Web. Nous faisons d'abord un rappel sur l'approche orientée service et ses principes de base puis nous donnons un bref historique sur les causes et avantages de l'apparition de la technologie des services Web.

Nous présentons par la suite, leur architecture de base SOA et son modèle ainsi que les différentes définitions qui leur sont associées dans la littérature. Nous décrivons également les différents standards qui constituent l'infrastructure de base des services Web et énumérons leurs avantages. Par ailleurs, nous consacrons la dernière section exclusivement à l'étude des langages dédiés à la description syntaxique et sémantique des services Web. Nous insistons sur la partie sémantique pour mettre en évidence la problématique posée en amont et clôturons le chapitre en dressant un tableau comparatif faisant ressortir les atouts et inconvénients de chaque approche étudiée.
Dans **le chapitre 2**, nous présentons le principe de l'approche de découverte et de sélection des services Web ainsi que les différentes approches existantes visant à optimiser ce processus. Nous étudions, d'une part, les approches basées sur la structure syntaxique ainsi que les approches basées sur le matching sémantique (qu'elles soient logiques ou pas voire hybrides) et terminons par une synthèse.
Dans **le chapitre 3** nous discutons les différents langages dédiés à la composition des services Web après l'introduction du concept de composition et la spécification de ses différents types. Enfin, les approches étudiées sont soumises à une comparaison multi-critères.
Une fois la partie état de l'art synthétisée nous présentons les différentes contributions apportées ainsi que les formalismes qui ont permis leur élaboration à travers les chapitres suivants :
Le **chapitre 4** est consacré à la présentation des formalismes utilisés, à savoir : la logique de réécriture et son environnement d'application Maude. Nous initions d'abord la logique de réécriture à travers sa théorie, ses règles de réécriture et de déduction. Le langage Maude quant à lui est défini par sa structure et son mécanisme de vérification. Enfin nous faisons un rappel sur la théorie des catégories en introduisant également les notions fondamentales relatives à cette dernière tout en spécifiant son lien avec la théorie de réécriture.
Le **chapitre 5** est dédié à notre première contribution qui a été mise en œuvre pour pallier au problème du manque de sémantique décelé dans WSDL. Nous présentons en premier lieu, le support de base de cette approche : le cadre sémantique K à travers ses modules syntaxique et sémantique. Nous donnons ensuite une vue d'ensemble de l'architecture de son outil K-Maude et de son utilisation. Dans un second temps, nous présentons notre nouveau langage sous K nommé KWSDL. Nous donnons sa grammaire ainsi que sa sémantique sous forme de règles de réécriture et motivons l'approche par un exemple.
Dans le **chapitre 6**, nous faisons part de notre seconde contribution basée sur la théorie des catégories. Nous présentons d'abord le modèle sémantique d'un service Web selon cette théorie et grâce à la notion de foncteur nous spécifions la façon d'établir un matching sémantique entre deux services représentés par leurs catégories. Cette équivalence est déduite par la préservation des objets et des morphismes constituant ces catégories.
Dans le **chapitre 7**, nous exposons la troisième et dernière contribution issue de ce travail doctoral. Nous présentons le modèle RMop-ECATNet : un modèle qui étend le modèle Mop-ECATNet dédié à la spécification de la composition des services Web complexes. Nous faisons au préalable un bref rappel sur les formalismes utilisés à savoir :

les ECATNets et les extensions de réseaux de Petri dédiées à spécifier les services Web. Le raffinement du modèle est spécifié et détaillé aux niveaux : structurel et comportemental. Il est motivé et justifié par un exemple illustratif.
Le **chapitre 8** est dédié à l'implémentation de RMop-ECATNet sous Maude. Nous faisons d'abord, un rappel sur l'architecture dirigée par les modèles et présentons à travers cette dernière les différents types de modèles existants pour représenter les systèmes informatiques à savoir : les notions de modèle, méta-modèle et méta-méta-modèle. Nous retrouvons aussi les différentes techniques de transformation de modèles ainsi que les outils qui leur sont associés. Nous illustrons l'intérêt de cette approche par la transformation du service Agence de voyage.
Nous clôturons notre manuscrit en procédant à une évaluation de notre cadre sémantique formel dégagé en démontrant l'impact des contributions apportées à chaque phase du cycle de vie du service Web par rapport à celui de la pile protocolaire et des langages qui constituent son infrastructure de base.
En guise de perspectives, nous citons quelques notions relatives à l'aspect comportemental des services Web qui n'ont pas été prises en charge dans notre travail telles que les propriétés non fonctionnelles dans le processus de sélection de services (qualité de service) et les contraintes temporelles dans la composition de services.

Chapitre 1

Description des Services Web

1.1 Introduction

L'approche orientée service vise à réduire le fort couplage entre les composants d'une application en les rendant autonomes, auto-configurables facilitant ainsi leur interopérabilité et réutilisation à travers des réseaux informatiques notamment sur le Web.
Issue de la convergence de plusieurs approches existantes, cette idée s'est affirmée comme solution idéale pour le développement d'applications complexes exigeantes en termes de maintenabilité et de flexibilité compte tenu de l'hétérogénéité des environnements dans lesquels elles évoluent. Il convient donc d'approfondir nos connaissances sur les différents concepts intervenant dans l'ascension de cette approche surtout pour démontrer en quoi la notion de service qui est le concept pivot de cette approche se révèle novatrice et avantageuse.
Dans ce chapitre, nous rappelons dans un premier temps, les principes fondamentaux de l'approche service à savoir : l'architecture de base SOA et les services Web en faisant un bref historique de leur évolution à travers le temps ; nous donnons aussi les différentes définitions qui leur sont associées dans la littérature. Nous présentons par la suite, les technologies et protocoles sur lesquels ils se basent ainsi que leur cycle de vie à travers ses phases tout en mettant en relief les différents avantages dégagés de leur mise en œuvre.
Dans un second temps, dans la section 6, nous présentons une sélection de langages dédiés à la description des services Web. Les langages étudiés sont triés en fonction de leur nature syntaxique ou sémantique. Enfin, nous clôturons notre chapitre en faisant une synthèse sur ces derniers.

1.2 Genèse

La naissance des services Web remonte à quelques années déjà ...
L'histoire commence lorsque les architectes et concepteurs se sont rendu compte de la nécessité d'uniformiser les échanges de données. Tous les axes de recherche se sont orientés

vers la notion d'interopérabilité, ce qui a généré en premier l'ancêtre des services Web l'EDI (Echange de Données Informatisé) [Cha97].
De nombreuses intentions technologiques ont émergé dans le but de faire interagir des processus métiers au travers d'un réseau informatique.
CORBA, DCOM et JavaRMI font donc leur apparition mais elles se sont malheureusement avérées de moins en moins convaincantes vu qu'aucune de ces technologies n'a pu s'imposer comme standard universel du fait de leur fort rattachement à des entités locales : système d'exploitation, éditeur ou langage spécifique.
Le Web à la rescousse : l'avènement d'Internet a bouleversé et changé l'optique des industriels en remplaçant les protocoles actuels par une approche entièrement ouverte faisant interagir des composants hétérogènes, distants et indépendants avec un protocole standard.
Les protocoles d'échange et de transfert qui ont été mis en place avec l'émergence du Web tels que XML [BPSM+97] (format d'encapsulation et d'échange de messages) et HTTP [FGM+99] (protocole de transport et de transfert de données hypertextes) concrétisent la notion d'interopérabilité.
L'un des avantages de la technologie des services Web et de son architecture est qu'elles n'imposent pas de modèles spécifiques pour l'implantation. Ceci ne limite pas leurs fournisseurs à employer diverses méthodes et langages pour assurer leur développement à l'inverse des autres technologies qui imposent l'instauration de liens stricts entre leurs entités.

1.3 Architecture SOA

1.3.1 Principe et Définition

Le principe de base de l'architecture SOA est de publier sur un réseau informatique (Intranet, Extranet, Internet) un ensemble de services réalisant une tâche bien précise qui peut être elle-même déployée comme un nouveau service, accomplissant alors une fonction plus importante dans l'architecture métier.
Il n'existe pas officiellement de définitions standards de l'architecture SOA ; néanmoins nous avons relevé de la littérature les énoncés suivants, chaque énoncé faisant référence à un ou plusieurs critères de l'architecture :

— Une architecture orientée service est une architecture d'applications au sein de laquelle toutes les fonctions sont définies en tant que services indépendants, comprenant des interfaces bien définies qui peuvent être sollicitées pour créer des processus d'entreprise. Elle permet de concevoir des systèmes logiciels fournissant des services à d'autres applications à l'aide d'interfaces publiées et découvrables, et où les services peuvent être sollicités via un réseau [CHT03].

— L'architecture SOA est un paradigme permettant d'organiser et d'utiliser des ressources distribuées pouvant être de domaines différents. Elle fournit un moyen uniforme d'offrir, de découvrir, d'interagir et d'utiliser des ressources pour produire le résultat désiré avec des préconditions et des buts mesurables [Jos07].
— Une architecture orientée service est un style d'architecture de logiciels multi-tiers qui aide les organisations à partager leurs logiques métier et leurs données entre plusieurs applications et plusieurs modèles d'usage [SN96].
— La SOA consiste à séparer l'activité métier en une série de services qui peuvent communiquer entre eux [PH07]. Ces services, leurs descriptions, et les opérations de base (publication, découverte, et liaison) que produisent ou utilisent ces descriptions, constituent la SOA de base [Pap03].

Nous déduisons à travers ces énoncés que l'architecture orientée service est une juxtaposition de plusieurs techniques qui intègre différentes pratiques dans un cadre organisé. C'est l'architecture des systèmes fortement guidée par le métier.

1.3.2 Modèle SOA

Le modèle de l'architecture SOA (Figure 1.1), est un modèle triangulaire comprenant trois entités principales : **Le fournisseur**, **le demandeur** et **l'annuaire** de services. Ces entités respectives se chargent de la description, la publication, le stockage, la localisation et l'invocation de services indépendants disponibles à travers la toile.

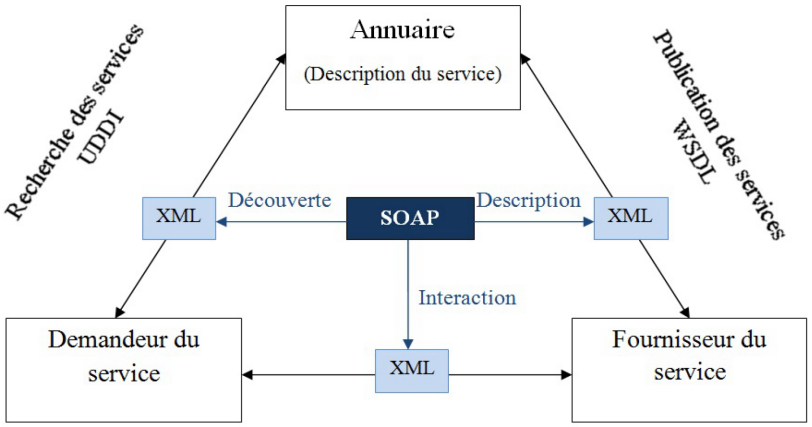


FIGURE 1.1 – Modèle conceptuel de l'architecture SOA

Fournisseur du service

C'est le détenteur du service ; il le conçoit et publie sa description dans un registre spécifique. Cette description reflète à la fois les fonctionnalités offertes par le service et leur mode d'invocation. Il représente également son environnement d'hébergement et

d'exécution.

Demandeur du service
C'est le consommateur du service, il localise le service, récupère sa description et formule ses besoins sous forme de requête et invoque les opérations souhaitées par le biais d'échanges de messages avec le fournisseur.

Annuaire du service
C'est une base de données indexée qui stocke les descriptions de services publiées par le fournisseur. Il joue le rôle d'intermédiaire entre les deux entités précédentes en acceptant, d'une part, de sauvegarder les descriptions publiées et d'autre part, en les exposant au client (demandeur de service) lors de ses éventuelles recherches.
Lorsque l'on parle d'architecture SOA, on parle forcement de services Web. Nous présentons dans ce qui suit cette technologie particulière au travers de son infrastructure de base, son cycle de vie et énumérons ses nombreux avantages.

1.4 Web services

1.4.1 Définition

La vulgarisation des services Web a depuis leur apparition, radicalement changé la vision des industriels sur leur manière d'échange et de collaboration.
Au début des années 2000, un consensus établi entre IBM et Microsoft promeut et concrétise le projet de cette technologie aux multiples avantages qui se voit vite standardisée par le W3C.
Ayant fait objet et été au centre de beaucoup de recherches et de découvertes, la notion de service Web s'est rattachée à une panoplie de définitions dans la littérature ; nous pouvons en citer les plus utilisées :
«Les services Web sont la nouvelle vague des applications Web. Ce sont des applications modulaires, auto-contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le Web. Les services Web effectuent des actions allant de simples requêtes à des processus métiers complexes. Une fois qu'un service Web est déployé, d'autres applications (y compris des services Web) peuvent le découvrir et l'invoquer» [Pon08].
Cette définition présente les services Web et met en relief leurs propriétés mais n'expose pas le contexte et les technologies qui assurent leur bon fonctionnement.

«Un service Web est un système logiciel identifié par une URI et conçu pour supporter l'interaction interopérable de machine à machine sur un réseau. Il possède une interface décrite dans un format exploitable par la machine, c.à.d. décrite en WSDL (Web Services Description Language). D'autres systèmes interagissent avec le service Web d'une façon prescrite par sa description en utilisant des messages SOAP (Simple Object Access Protocol), typiquement en utilisant HTTP avec une sérialisation XML en même temps

que d'autres normes du Web» W3C [1].

Une définition plus raffinée des services Web est donnée par [Bek12]. «La technologie des services Web offre de fortes potentialités pour surmonter les problèmes d'interopérabilité des systèmes. Elle constitue un cadre prometteur pour l'intégration des applications et pour la gestion des interactions entre divers partenaires dans un environnement distribué, hétérogène, ouvert et versatile qui est le Web».

Nous avons à travers l'étude de toutes ces définitions conclu d'un nouvel énoncé personnel regroupant les principales caractéristiques des services Web :
«Un service Web est un composant sur le Web encapsulant un ensemble de fonctionnalités pouvant être invoquées par le biais de requêtes client simples ou composées dans le but de satisfaire des besoins. Ces composants sur le Web partagent la notion d'autonomie et sont faiblement couplés, ce qui facilite leur réutilisation. Les services Web évoluent dans des contextes hétérogènes et sont soutenus par une infrastructure basée sur les standards du Web : WSDL pour leur description, SOAP pour leur communication et UDDI pour leur localisation et sauvegarde».

1.4.2 Infrastructure de base

L'infrastructure des services Web repose sur des protocoles et langages de spécification répandus sur le Web.
En effet, chaque entité active dans le triangle architectural de la SOA est liée à un standard. Nous distinguons ainsi trois standards pour chaque type d'interaction :

— **WSDL** : comme format de base pour la description des services Web.
— **SOAP** : comme protocole de communication et structuration des messages.
— **UDDI** : comme registre de stockage des services Web.

1. www.W3C.org

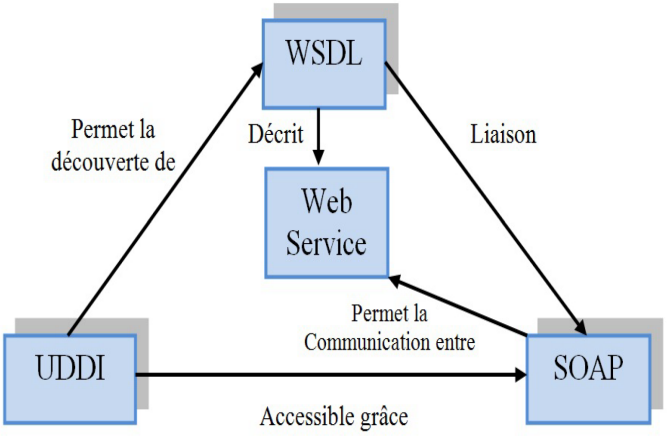


FIGURE 1.2 – Triangle protocolaire des services Web

Nous détaillons dans ce qui suit ces concepts en introduisant d'abord, le support de base de ces technologies, le langage XML.

Langage XML
Le langage XML (eXtensible Markup Language) mis au point par le W3C (World Wide Web Consortium) à la fin des années 90 est un format universel d'échange de données et d'informations. Il est largement utilisé et adapté par les entreprises et ce dans diverses disciplines.
XML peut servir de base pour exprimer tout type de données. Son emploi est polyvalent, il peut servir pour l'échange des données tout comme pour leur stockage sous forme de bases de données ou de fichiers. XML repose sur une structure à balises, chaque balise fait référence à une information intégrée au fil du texte. D'une manière générale, XML est un langage qui constitue la technologie de base des architectures basées sur le Web.
Il offre une indépendance vis-à-vis de tout langage ou plateforme et respecte les critères qu'impose la notion d'interopérabilité.
De par ces mécanismes de structurations, il garantit la flexibilité dans les échanges entres client et fournisseur.

Langage WSDL
Introduit également par le W3C, WSDL est un langage à balises basé entièrement sur XML ; il reflète la description des services Web. Cette description concerne exclusivement la partie abstraite du service. En effet, WSDL est un langage d'interface qui expose les fonctionnalités du service indépendamment de son implémentation.
WSDL est simplement un IDL (Interface Definition Language) différent des autres IDLs dans le sens où il est tout aussi neutre du point de vue du protocole de communication entre services Web que du point de vue de leur implantation. Une description WSDL est essentielle à l'invocation d'un service.

La description WSDL peut être qualifiée par le terme «contrat de service» avec un format bien précis. Ce contrat regroupe deux classes principales ou deux types de descriptions : une description abstraite et une autre concrète, les éléments qui composent ces deux parties sont :
Au niveau abstrait : types de données, messages, opérations, portType, liaisons (bindings).
Au niveau concret : ports et service.
Ces éléments sont exposés de manière détaillée dans la section 1.5.

Protocole SOAP
SOAP «Simple Object Access Protocol» est un protocole d'invocation de méthodes sur des services distants. Son rôle consiste à faire circuler des informations codées en XML en appelant une méthode RPC et en envoyant des messages aux machines distantes via HTTP afin de faire dialoguer deux applications ou plus (voir Figure 1.3). Il est aussi simple et facile à implémenter dans les serveurs Web destinés à l'échange d'informations dans un environnement distribué et décentralisé.
Une des volontés du W3C vis-à-vis de SOAP est de ne pas réinventer une nouvelle technologie. SOAP a été construit pour pouvoir être aisément porté sur toutes les plates-formes et les technologies existantes.
Qualifié de simple, universel et flexible, SOAP est très bien adapté à l'utilisation des services Web car il permet de fournir au client une grande quantité d'informations récupérées sur un réseau de serveurs tiers. Il peut, entre autres, donc être utilisé dans tous les styles de communication : synchrone ou asynchrone, point à point ou multipoints, Intranet ou Internet.

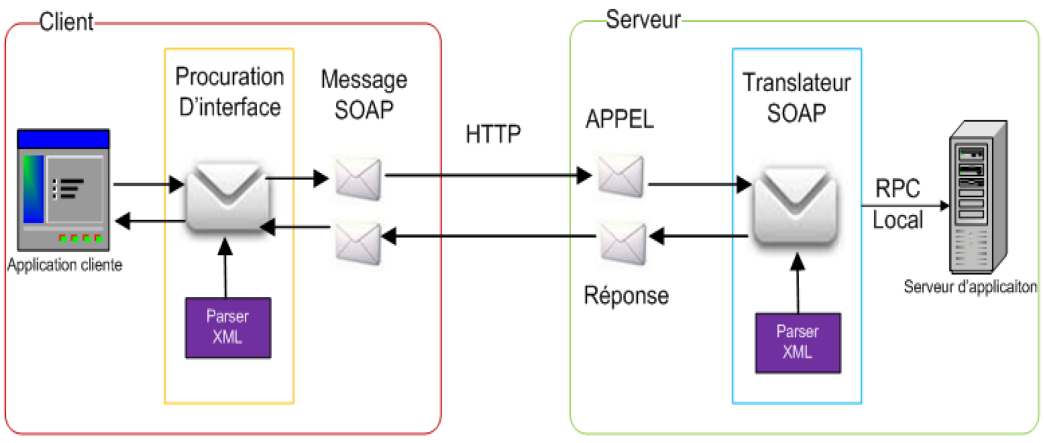


FIGURE 1.3 – Rôle du protocole SOAP

Situé à l'intérieur d'une enveloppe le message SOAP comprend deux parties : l'entête et le corps (Figure 1.4).

— **L'entête (Header)** : offre des mécanismes flexibles pour étendre un message SOAP sans aucune préalable connaissance des parties communicantes. Les extensions peuvent contenir des informations concernant l'authentification, la gestion des transactions, le payement ...etc.
— **Le corps (Body)** : offre un mécanisme simple d'échange des informations mandataires destinées au receveur du message SOAP. Cette partie contient les paramètres fonctionnels tels que le nom de l'opération à invoquer, les paramètres d'entrée et de sortie ou des rapports d'erreur.

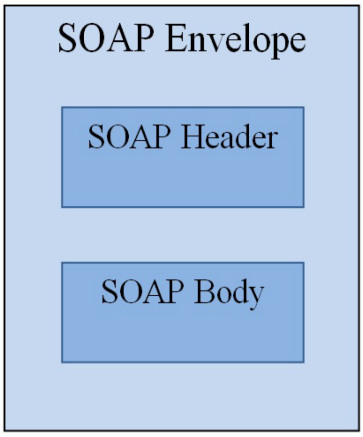


FIGURE 1.4 – Structure d'un message SOAP [ML+03]

Annuaire UDDI
UDDI (OASIS) a été conçu en 2000 à l'initiative d'un ensemble d'industriels (Ariba, IBM, Microsoft) en vue de devenir le registre standard de la technologie des services Web. L'UDDI gère l'information relative à la publication, la découverte et l'utilisation d'un service Web.
Il a été créé pour faciliter la découverte de services Web en plus de leurs publications par un API SOAP. Le demandeur de service peut interagir avec l'UDDI au moment de la conception et de l'exécution des applications afin de découvrir des données techniques et administratives sur les entreprises et leurs services Web.
L'UDDI est subdivisé en deux parties principales : une partie publication ou inscription et une partie découverte. La partie publication regroupe l'ensemble des informations relatives aux entreprises et à leurs services. Ces informations sont introduites via une API d'enregistrement. La partie découverte facilite la recherche d'information contenue dans ce registre.
L'UDDI peut être vu comme un annuaire contenant les parties suivantes :

FIGURE 1.5 – Pages de l'annuaire UDDI [TO02]

Les pages blanches (White Pages) : Ce composant permet d'avoir des informations à propos de l'organisation proposant le service. Cette description contient toutes les informations jugées pertinentes pour identifier l'organisation (telles que son nom et son adresse physique). Le futur client du service retrouve dans les pages blanches les informations fournies lors de la publication.

Les pages jaunes (Yellow Pages) : Les pages jaunes détaillent la description de l'organisation faite dans les pages blanches en répertoriant les services proposés. Dans cette section, sont décrits : la catégorie de l'entreprise, le secteur d'activité dans lequel exerce l'entreprise, les services offerts par cette organisation, le type de services et les conventions d'utilisation (prix, qualité de service ...etc.). La description des services contenue dans les pages jaunes est non technique et est donnée par les fournisseurs eux-mêmes.

Les pages vertes (Green Pages) : Les pages vertes comportent les informations techniques liées aux services Web. Elles se basent sur leurs descriptions WSDL.

1.4.3 Cycle de développement d'un service Web CV_{WS}

Les services Web possèdent un cycle de vie formé de plusieurs phases que nous avons structuré dans le schéma (Figure1.6). Nous avons en l'occurrence :

La phase de description où un fournisseur décrit en WSDL ses services et les publie dans l'annuaire UDDI.
La phase de découverte et de sélection qui se résume en la sollicitation de l'UDDI par le client via une requête dans le but de découvrir et potentiellement sélectionner les services recherchés.

La phase d'invocation est la concrétisation du choix du client (sa sélection) en invoquant directement par messages SOAP le fournisseur des services choisis.
La phase de composition est relative à l'assemblage de plusieurs services par le fournisseur afin de satisfaire une requête-client complexe.
Enfin, **la phase d'analyse** est réservée au test des modèles de services résultants des autres phases au moyen d'outils d'analyse et de vérification.

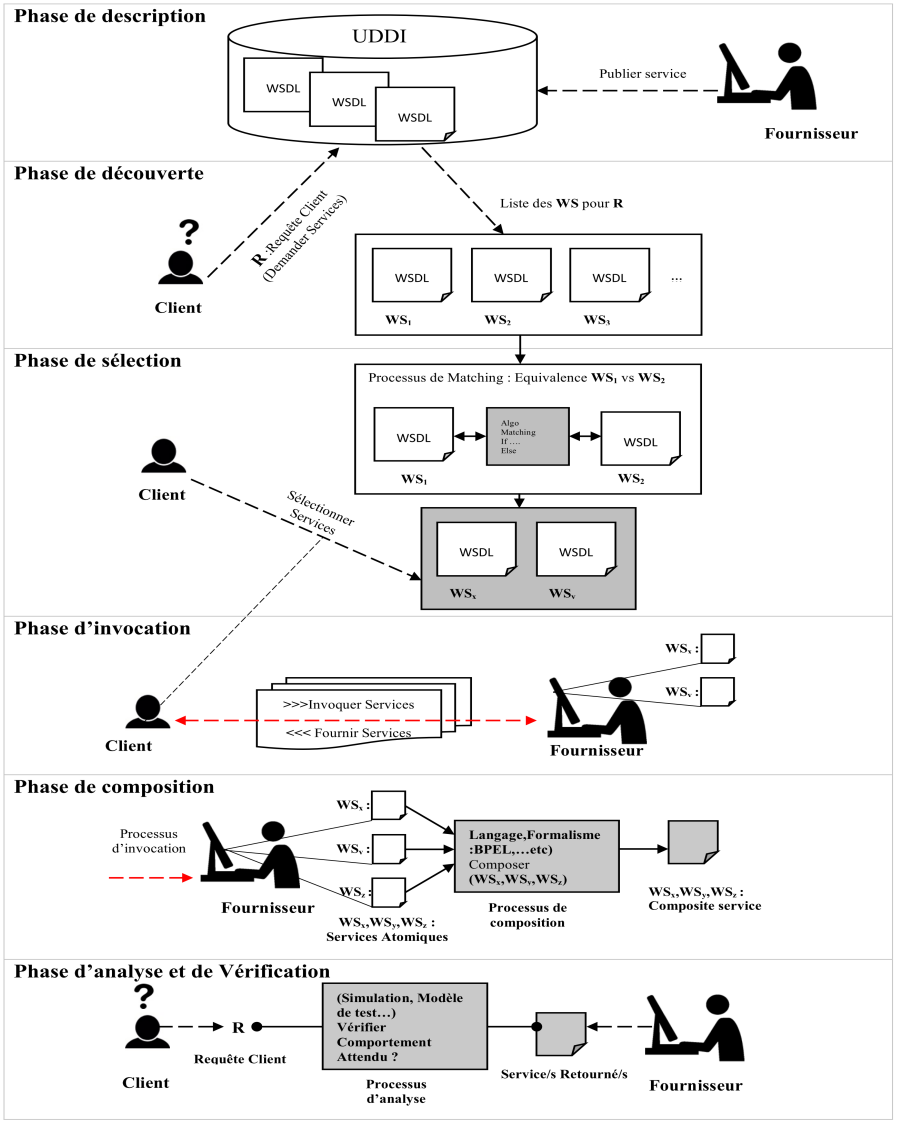


FIGURE 1.6 – Cycle de vie d'un service Web CV_{WS}

Beaucoup de bénéfices peuvent être déduits de la technologie des services Web et de leur architecture. Nous en relevons les plus communs :

— **Autonomie**
Les services Web s'autogèrent grâce au langage XML qui facilite leur autonomie puisqu'il supporte et véhicule tout type de données. En effet, il est possible d'activer une application existante pour services Web sans écrire une seule ligne de code.

— **Accessibilité**
Les services Web sont facilement accessibles puisqu'ils reposent sur des normes établies d'Internet ; ils peuvent être publiés, localisés et invoqués en transparence à travers le Web.

— **Interopérabilité**
Quel que soit le contexte ou l'environnement dans lequel évolue le service Web, il n'est pas nécessaire de changer le code existant du service afin d'en activer une opération. Le format XML le dote d'une flexibilité qui ne freine pas ses interactions avec les autres services.

— **Composabilité**
Les services Web simples peuvent être agrégés dans des services plus complexes et peuvent être associés afin d'effectuer des fonctions métier de niveau plus élevé. Ce chaînage réduit le temps de développement et permet d'optimiser les implémentations.

— **Faible couplage**
Généralement, la conception d'une application dépend d'un ensemble d'interconnexions étroites entre les deux extrémités. Les services Web requièrent un niveau plus simple de coordination qui permet une nouvelle configuration plus flexible pour une intégration des services.

— **Réutilisation**
Les applications autonomes existantes peuvent facilement être intégrées dans l'architecture orientée services ; les notions sus-citées accentuent ce pouvoir. En effet, l'autonomie et le faible couplage des services favorisent grandement leur réutilisation.

1.5 Langages de description des services Web

Beaucoup d'efforts ont été consentis pour décrire les capacités d'un service Web. Nous distinguons les descriptions syntaxiques et les descriptions sémantiques ; chacune d'elle offre des facettes décrivant soit la structure et/ou le comportement du composant.

L'un des nombreux objectifs de l'Architecture Orientée Service est que les briques de base de l'implémentation (les services) puissent être réutilisées dans d'autres systèmes. La réutilisation de services Web repose sur le fait que ces derniers soient accessibles (sur le Web) et utilisables par le plus grand nombre de clients possibles. Ainsi, ce sont des services dits universels et non propriétaires.
La description d'un service Web est nécessaire pour deux raisons : la publication du service par son fournisseur dans un registre et sa sélection ultérieure par des clients via ce registre. Nous consacrons, par conséquent, cette partie du chapitre aux moyens qui existent aujourd'hui pour décrire les services Web.
Dans cette section, nous nous préoccupons exclusivement de la description de services Web élémentaires puisqu'elle repose sur des langages spécifiques. Un service Web élémentaire est un service isolé pouvant servir de services de base à une composition de services Web. Selon sa description, un service élémentaire peut être classique ou sémantique.
L'étude du standard de description de services Web classique (WSDL) fait l'objet de la première étude tandis que les travaux proposant une description de services Web sémantiques sont exposés dans la seconde partie.

1.5.1 Langage de description syntaxique

Uniformiser la description des services Web tend à accroitre leur réutilisation. Les premiers travaux du W3C concernant un langage de description universel de services Web ont vu le jour en 2001 lors de l'émergence de cette technologie. Il a fallu trouver un langage de description utilisable et compréhensible par le plus grand nombre afin que les services Web conçus soient interopérables. Le W3C a rapidement proposé le langage WSDL (Web Service Description Language).
Par ailleurs, la description d'un service doit inclure la définition des composants nécessaires au protocole de communication afin d'assurer l'interaction avec un client ou un autre service Web. Les problématiques de réutilisation et d'interaction ont guidé le W3C afin de définir les catégories d'information à prendre en compte dans la description d'un service Web. Les éléments essentiels retenus sont :

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

Ces éléments ont été structurés dans un document XML étant donné que ce dernier facilite l'interopérabilité des systèmes distants et qu'il est facilement extensible. Ils sont aussi séparés en deux parties indépendantes, respectivement nommées partie abstraite et partie concrète (Figure 1.7). La partie abstraite regroupe les informations pouvant être réutilisées (non spécifique à un service), tandis que la partie concrète est constituée de la description des protocoles d'accès au service Web. La première est utilisée principalement lors du processus de sélection et la seconde lors de l'invocation des méthodes du service

Web. Un rôle est assigné à chaque élément :

Structure d'un document WSDL

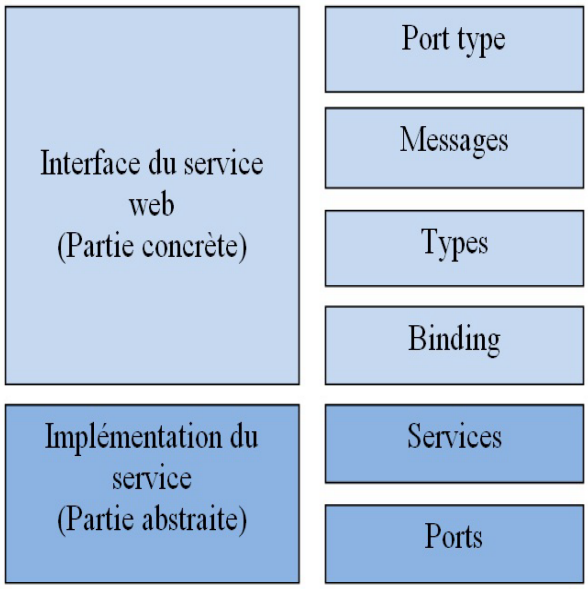


FIGURE 1.7 – Structure d'un document WSDL

Types de données : sont utilisés par le service Web lors des échanges de messages. Une fois définis, ils peuvent être référencés dans n'importe quel message.

Messages : spécifient les types d'opérations supportées par le service Web ; ils permettent d'incorporer une séquence de messages corrélés sans avoir à spécifier les caractéristiques du flux de données. Par exemple, un message Input et un message Output corrélés sont mis en correspondance dans une seule opération de type «Request/Response».

Opérations : spécifient les types d'opérations supportées par le service Web ; elles permettent d'incorporer une séquence de messages sans avoir à spécifier les caractéristiques du flux de données.

PortType : est un groupement logique ou une collection d'opérations supportées par un ou plusieurs protocoles de transport.

Liaisons (Bindings) : décrivent la façon dont un type de port est mis en œ uvre pour un protocole particulier (HTTP par exemple) et un mode d'invocation (SOAP par exemple). Cette description est faite par un ensemble donné d'opérations abstraites. Pour un type de port, on peut avoir plusieurs bindings pour différencier les modes d'invocation ou de transport de différentes opérations.

Au niveau concret, le service Web est défini grâce aux deux éléments : Port et Service. Ils décrivent des informations liées à usage contextuel du service Web. On y trouve : l'adresse du fournisseur implémentant le service et le service.

Port : spécifie une adresse URL qui correspond à l'implémentation du service Web par un fournisseur et identifie un ou plusieurs «Bindings» (ou liaisons) aux protocoles de transports (HTTP...etc.) pour un «PortType» donné. La séparation du protocole de transport de la définition du «PortType» permet à un service Web d'être valable à travers plusieurs protocoles de transport, sans avoir à redéfinir l'ensemble du fichier WSDL.

Service : spécifie l'adresse complète du service Web et permet à un point d'accès d'une application distante de choisir de multiples catégories d'opérations pour divers types d'interactions.

Tous ces éléments sont les composantes de la première version de WSDL1.0 créée en 2000 par IBM, Microsoft et Ariba. La version WSDL 1.1 a été publiée peu de temps après par le W3C comme une note [CCM+01]. La toute dernière dernière version «WSDL 2.0» a été publiée en 2007 apportant notamment quelques changements à la version WSDL 1.2. Ces changements se définissent comme suit :

— L'élément portType est devenu l'élément interface.
— L'élément port a été remplacé par l'élément endpoints.
— L'élément message à été supprimé et l'élément xs : est utilisé pour spécifier les données échangées.

Quelle que soit la version de WSDL, la description du service reste uniquement au niveau fonctionnel ; c'est-à-dire qu'elle contient la manière dont on peut utiliser le service (Quoi ?) et non ce que fait le service (Comment ?). Par conséquent, la description WSDL est jugée insuffisante pour répondre de manière précise à une requête client. Afin de pallier à cette difficulté, beaucoup d'approches s'inspirant du Web sémantique proposent des extensions sémantiques à ce type de langage. Dans la section suivante nous en présentons quelques unes.

1.5.2 Langages de description sémantiques

Les services Web sémantiques sont la combinaison de deux technologies : celle des services Web et celle du Web sémantique [MM03]. En effet, leurs descriptions sont améliorées par des langages empruntés au Web sémantique, tels que RDF [KC04] et OWL [MVH+04]. Cet emprunt permet aux services sélectionnés de répondre avec «sens» à la requête du client. Les services Web peuvent s'élever au rang des services Web sémantiques

suivant deux voies :

— La première consiste à développer un langage complet qui décrit les services Web ainsi que leur sémantique en un seul bloc.
— La deuxième consiste à annoter les langages existants avec de l'information sémantique. L'avantage principal de ce genre de solutions réside dans la facilité pour les fournisseurs de services d'adapter les descriptions existantes aux annotations proposées.

1.5.2.1 Langages à base d'ontologies

Des travaux tels que [FB02, RKL+05] proposent de représenter de manière sémantique des services Web. Malgré l'abondance de ce type de travaux, aucune approche ne s'est imposée comme une solution pour la description des services Web sémantiques. Nous avons choisi d'aborder ici seulement les travaux issus du W3C qui tentent d'apporter une solution standard en termes de description de services Web sémantiques. Dans cette section, trois langages sont présentés : DAML-S, OWL-S et WSMO ; le premier est un langage sémantique basé sur les logiques de description et les deux derniers sur des ontologies très réputées dans le domaine des services Web.

DAML-S [ABH+02] est un langage de description de haut niveau pour la description et l'invocation des services Web basé sur XML et utilisant le modèle des logiques de descriptions. Pour décrire un service Web DAML-S regroupe trois services principaux :

— ***Service Profile*** il permet la description, la promotion et la découverte des services, en décrivant non seulement les services fournis mais également des préconditions à la fourniture de ces services. Les recherches sur les services peuvent être effectuées en prenant n'importe quel élément du Service Profile comme critère de recherche.

— ***Service Model*** il présente le fonctionnement du service en décrivant la manière d'y accéder. Certains éléments du Service Model peuvent être utilisés à la manière du Service Profile afin de fournir des informations supplémentaires à un utilisateur pour lequel les opérations à effectuer seraient également un critère de choix. C'est le Service Model qui permet la composition des services si besoin est. Il permet également d'effectuer un contrôle poussé sur le déroulement de la composition.

— ***Service Grounding*** il spécifie les voies d'accès au service. Tout type abstrait déclaré dans le Service Model s'y verra attribuer une manière claire d'échanger l'information. C'est dans cette partie que le protocole et les formats des messages sont spécifiés.

OWL-S [MBH+04] : Ontology Web Language for Services (successeur de DAMLS) est une ontologie de haut niveau pour la description sémantique des services Web. Une ontologie permet de représenter un domaine par des structures interprétables par une machine. L'objectif de OWL-S est d'assurer l'automatisation du cycle de vie du service Web.
OWL-S est aussi constitué de trois concepts : Le service profile, le process model et le service grounding. Ces trois super-concepts permettent de décrire l'aspect fonctionnel et non fonctionnel du service ainsi que la composition de services. Il permet aussi de relier cette description à l'interface WSDL.

— Le «*Service Profile*» a pour objectif d'automatiser la découverte et la sélection ; il fournit une description de haut niveau d'un service et de son fournisseur. Le service profile contient trois types d'informations :

1. Une description du service et de son fournisseur (informations de contact).
2. L'aspect fonctionnel du service et en particulier : les entrées, sorties, pré-conditions et post-conditions des opérations.
3. Des propriétés non-fonctionnelles comme la réputation, la catégorie et la qualité du service de façon générale.

— Le «*Process Model*» définit le flux de contrôle et les données d'une composition de services. Trois types de processus y sont définis : atomiques, simples et composites.
— Le «*Service Grounding*» décrit les moyens d'accès au service en spécifiant le protocole de communication, le format des messages et l'encodage des paramètres. Il connecte le «Process Model» avec la description WSDL sous-jacente.

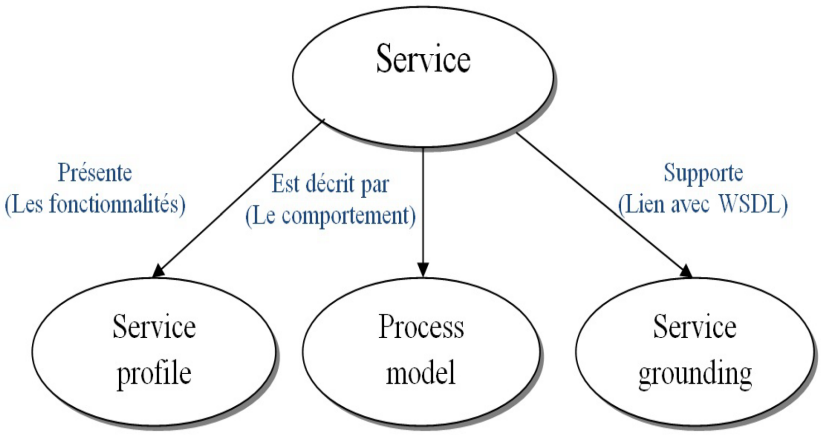


FIGURE 1.8 – Format de description d'un service par l'ontologie OWL-s

WSMO[DRS05] : Web Service Modeling Ontology est une méta-ontologie basée sur le Web Service Modeling Framework WSMF [FB02]. Le but de WSMO est d'automatiser le

cycle de vie des services Web. Il est constitué de quatre éléments : les services Web, les buts (Goal), les ontologies et les médiateurs ; chaque élément est décrit par un formalisme basé sur le langage WSML [DBLPF06].
Ce modèle réduit le couplage entre les services Web en utilisant un ensemble de médiateurs. Ces derniers assurent les tâches d'intégration d'ontologies, de découverte des services et de leur composition. En complément aux approches définies nous présentons dans la section qui suit les langages sémantiques à base d'annotations sémantiques.

1.5.2.2 Langages à base d'annotations sémantiques

L'annotation sémantique consiste à enrichir et à compléter la description d'un service. Elle établit des correspondances entre des éléments de la description et des concepts d'un ensemble d'ontologies de référence. Nous présentons dans cette section deux modèles : WSDL-S, SAWSDL. Ces modèles permettent d'annoter manuellement une description WSDL avec des éléments faisant référence à des ontologies.

WSDL-S [MVR+04] : Le but de WSDL-S est d'annoter l'interface WSDL, son méta-modèle permet l'ajout de trois éléments : <*category*>,<*precondition*>,<*effect*> et deux attributs *modelReference* et *schemaMapping*.
Les éléments ajoutés enrichissent la sémantique de WSDL et les attributs modelReference et schemaMapping pointent des concepts ontologiques.

- L'élément <*category*> est un sous-élément de <portType> qui donne la classe d'un service (la fonctionnalité) en utilisant une certaine taxonomie. Il est introduit lors de la publication du service dans l'annuaire.
- L'élément <*precondition*>, sous-élément de <operation> qui précise les préconditions devant être vérifiées avant l'exécution de l'opération.
- L'élément <*effect*>, sous-élément de <operation> qui précise les changements qui résultent de l'exécution de l'opération.
- L'attribut *ModelReference* est associé à un <xs :element>(qui fait partie d'une grammaire XML) et aux éléments <operation>,<precondition>et<effect>. Il pointe sur le concept de l'ontologie de référence.
- L'attribut *schemaMapping* est associé à un <xs :element> ; il fournit les correspondances entre la grammaire XML et les ontologies utilisées.

  SAWSDL[KVBF07] : Semantic Annotations for WSDL and XML Schema est une recommandation W3C qui étend la version WSDL2.0. Elle annote, en particulier, les éléments : opérations, input, output, type schemas, et interfaces. La spécification annote un document WSDL avec les attributs suivants : *ModelReference*, *LiftingSchemaMapping* et *LoweringSchemaMapping*.
- L'attribut *ModelReference* permet d'annoter certains éléments WSDL 2.0. Il est utilisé comme attribut dans les éléments <interface>, <operation> et <fault>. Il

indique le concept équivalent en précisant son adresse par exemple : une opération nommée op1 avec un paramètre d'entrée M1 et de sortie M2. Pour associer cette opération au concept C1 de l'ontologie Ontologie1, il suffit d'ajouter à l'élément définissant l'opération l'attribut : sawsdl :modelReference="Ontology1#C1".

— Les attributs *LiftingSchemaMapping* et *LoweringSchemaMapping* permettent d'associer à un schema type ou à un élément, un concept dans une ontologie partagée. SAWSDL n'impose pas de langages particuliers pour formaliser les ontologies.

1.6 Synthèse

Dans la section 1.5, nous avons mis en relief les différents langages proposés par le consortium W3C permettant aux fournisseurs de décrire leurs services Web. Cette étape de description est le premier processus indispensable (après l'implémentation du service) dans le cycle de vie d'une application basée sur une SOA.

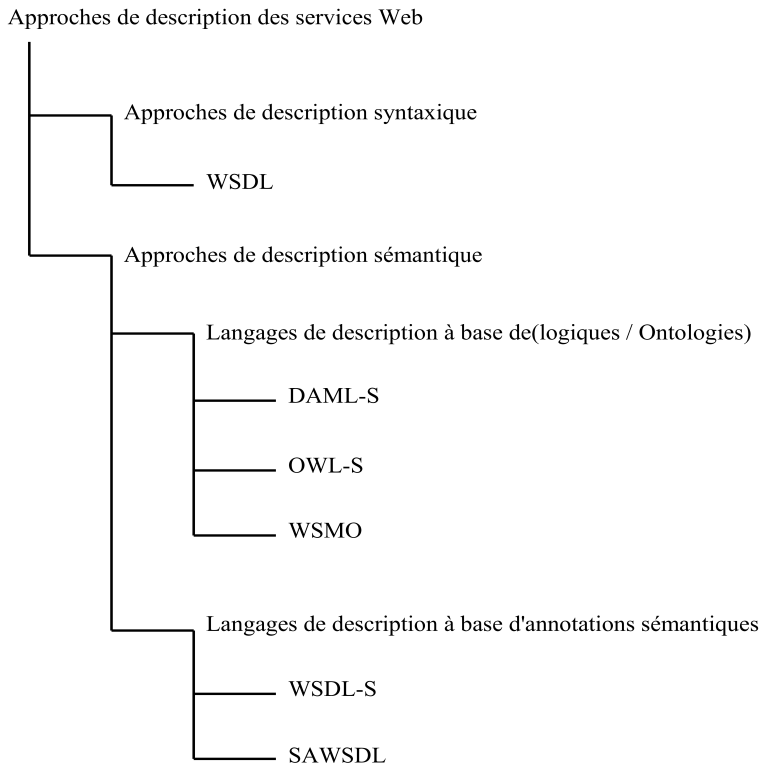


FIGURE 1.9 – Approches de description des services Web

La Figure ci-dessus résume les approches de description de services étudiées et les classe par catégorie.
Le tableau suivant dresse un comparatif entre ses dernières afin de mettre en relief leur

atouts et lacunes. Nous estimons qu'une description de services Web appropriée doit respecter au minimum les critères suivants :

Expressivité : un langage de description de services Web est expressif quand toutes les caractéristiques essentielles à un service peuvent y êtres exprimées.

Sémantique explicite : afin de garantir une interopérabilité sans faille et une découverte de services exhaustive le langage de description doit être doté d'une sémantique claire.

Réutilisation/Adaptation : les fragments d'une description de service doivent être réutilisables.

Non-Dépendance : ce critère n'est pas requis mais nous le considérons juste pour voir les dépendances que peut avoir un langage à des logiques, ontologies...etc car ces dépendances réduisent considérablement leur réutilisation.

Standardisation : c'est un critère d'ouverture qui estime la conformité de l'approche aux normes de référence.

Langage de description des SW	Expressivité	Sémantique explicite	Réutilisation Adaptation	Non-Dépendance	Standardisation
WSDL	-	-	+	+	+
DAML-S [ABH+02]	+	+	-	-	-
OWL-S [MBH+04]	+	+	+	-	+
WSMO [DRS05]	+	+	+/-	-	-
WSDL-S [MVR+04]	+	+/-	+	+	+
SAWSDL [KVB+07]	+	+/-	+	+	+/-

TABLE 1.1 – Comparaison entre les approches de description des services Web

+ : aspect pris en compte,+/ - : aspect moyennement considéré,- : aspect non-pris en compte.

De prime abord, nous constatons qu'aucune des approches étudiées ne respecte les critères escomptés. En effet, nous constatons que pour exprimer les propriétés fonction-

nelles d'un service le standard WSDL est incontournable mais malgré cela son taux d'expressivité est relativement bas du fait de son manque de sémantique évident.
Les langages à base d'ontologies tels que WSMO, OWL-s sont, quant à eux, très expressifs du fait de la sémantique octroyée par leurs ontologies de référence respectives.
Cependant, cette forte dépendance à ces ontologies réduit grandement la réutilisation de leurs descriptions.
Par ailleurs, Les langages à base d'annotations WSDL-s et SAWSDL sont très appréciés par les développeurs habitués à WSDL ; ils ne dépendent pas d'une ontologie précise et offrent beaucoup de flexibilité dans la description des services. Il n'en demeure pas moins que certaines propriétés importantes ne sont pas permises telles que les propriétés non-fonctionnelles dans SAWSDL.
Nous constatons également que ces langages sémantiques, bien que standardisés pour la plupart, restent assez proches en terme d'expression des fonctionnalités d'un service Web. Aussi, les nouvelles recommandations sont assez pauvres en terme d'expérience pour pouvoir juger de leur aptitude à spécifier correctement un service Web.
Enfin nous notons que le langage DAML-S est une des seules solutions proposant une réelle sémantique des données et pas seulement des champs prédestinés par la structure des standards ; de plus, son utilisation des logiques de descriptions réduit certes la réutilisation de ses composantes mais permet une grande puissance d'expression que ne possèdent pas tous les autres langages. C'est dans cette même optique, que nous orientons notre travail à savoir : utiliser des logiques et formalismes de haut niveau pour spécifier les services Web.

1.7 Conclusion

Le couplage entre les technologies du Web et les composants logiciels a permis de découpler leurs liens respectifs tout en augmentant leurs capacités d'interopérabilité et de réutilisation. Ces composants ont gagné grandement en autonomie en intégrant le Web et en adoptant ses technologies. La conséquence d'une telle mise en œuvre est l'instauration de la technologie des services Web.
Dans ce chapitre, nous avons dans un premier temps, fait un rappel sur les concepts fondamentaux de la technologie des services Web et de leur architecture. Nous avons également présenté les langages et les protocoles de base permettant leur déploiement, les phases constituant leur cycle de vie et énuméré leurs avantages.
Nous avons dans un second temps, établi un état de l'art sur les langages de descriptions syntaxiques et sémantiques dédiés aux services Web normalisés par le consortium W3C pour certains et en cours de standardisation pour d'autres. A l'issue de cette étude une synthèse sur ces langages a été établie.

Chapitre 2

Découverte et Sélection des Services Web

2.1 Introduction

Le processus de découverte de services Web s'appuie sur le mécanisme d'appariement entre une requête client et les services disponibles. Plus précisément, entre mots clés d'une requête formulée et la description de l'interface du service WSDL ou de son comportement. Lors d'une découverte de services plusieurs critères peuvent être considérés :

- **Critère architectural :** relatif au stockage et à la localisation des descriptions de services dans le réseau (systèmes centralisés/décentralisés)
- **Critère d'automatisation :** relatif au degré de l'intervention de l'humain dans le processus de découverte.
- **Critère de Matching :** basé sur des algorithmes faisant l'appariement entre descriptions de services Web pour mesurer leur degré d'équivalence (syntaxique ou sémantique)

Dans ce chapitre, nous discutons les approches contribuant à l'optimisation du processus de découverte et intrinsèquement celui de sélection en nous basant principalement sur le critère de mesure de similarité entre descriptions de services Web (critère de Matching). Nous étudions les approches de Matching : syntaxiques, sémantiques (logiques/non-logiques), hybrides ou encore non-fonctionnelles (en terme de QoS).

2.2 Approches de Découverte et de Sélection des services Web

Plusieurs approches relatives à la découverte fonctionnelle de services existent, nous distinguons : les approches syntaxiques, sémantiques ou hybrides (regroupant les deux aspects précités).

2.2.1 Approches Fonctionnelles

Plusieurs approches relatives à la découverte fonctionnelle de services existent, nous distinguons : les approches syntaxiques, sémantiques ou hybrides (regroupant les deux aspects précités).

2.2.1.1 Approches Syntaxiques

L'interface WSDL constitue la base de calcul du degré de similarité entre services Web dans une approche de découverte syntaxique.
La découverte de services Web à travers l'annuaire UDDI symbolise le processus de découverte syntaxique standard. Les mots clés d'une requête client sont appariées aux informations du service fourni à travers son interface. Ces informations peuvent inclure le nom du service, de son fournisseur ou de sa catégorie.
Dans le même contexte, les auteurs de [GPST04] s'appuient sur les critères de localisation du service ou du binding template. Une des approches les plus réputées est celle qui utilise le thesaurus Wordnet [Fel98] afin d'associer des synonymes aux mots clés de la requête. Query By Example (QBE) [CZC08] utilisent des exemples pour effectuer une recherche de service. Cet exemple est formé à partir des attributs suivants : La fonctionnalité du service, le nom de l'opération et les noms des paramètres. Cette approche se base sur le modèle de représentation vectoriel Vector Space Model (VSM).
[ANS+10] se base sur deux algorithmes, un algorithme comparant les éléments de la partie abstraite du document WSDL et un second comparant les éléments de la partie concrète.

2.2.1.2 Approches Sémantiques

Comme nous l'avons souligné dans le premier chapitre état de l'art, plusieurs extensions sémantiques du langage WSDL ont été établies. Ce sont ces informations sémantiques qui sont exploitées dans le processus de découverte. Les approches étudiées dans cette section sont catégorisées selon trois aspects : les approches logiques, les approches non logiques et les approches hybrides.
Les approches logiques exploitent les inférences pour vérifier la compatibilité entre la requête et l'annotation du service (subsomption, test de consistance). Les approches non logiques exploitent la sémantique implicite ou informelle des services et la traite avec d'autres techniques telles que le Datamining, le matching de graphes, la recherche d'informations. Enfin, la troisième classe regroupe les approches prenant en considération

les deux premiers aspects.

Approches Logiques
Dans [PKPS02] les auteurs présentent une approche de découverte sémantique comparant les sorties d'une requête donnée aux sorties du service offert, selon l'algorithme suivant :

matching (R.O,S.O){
Retourner
Exact : si les deux concepts ontologiques R.O et S.O sont équivalents.
Plug-in : si S.O subsume R.O
Subsume : si R.O subsume S.O
Fail : si aucune relation d'équivalence ou de subsomption n'existe entre eux}

S.O et R.O dénotent respectivement les sorties du service S et les sorties de la requête R. Ils sont exprimés en DAML, les services et la requête sont formalisés avec DAML-S. Si deux services S1, S2 obtiennent le même score, ils sont soumis à un second matching (S.I,R.I) afin de comparer leurs entrées à celles de la requête.
[HPS+03] étend l'algorithme de [PKPS02] en ajoutant un cinquième score intitulé «intersection» ; ce dernier permet de repérer les services ayant des concepts qui chevauchent la requête.
[BHL+05] propose un algorithme pour découvrir la couverture sémantique d'une requête par les concepts d'une ontologie exprimée avec la logique de description. La requête est constituée d'un ensemble de concepts d'entrées et de sorties. [Sri06] propose un mécanisme de matching faisant intervenir à la fois les entrées, les sorties, les préconditions et les effets (IOPE) des services.

Approches Non Logiques
[LDRW04] propose un système nommé Woogle qui calcule les similarités entre les descriptions textuelles des opérations de services et entre les identifiants des paramètres d'entrées/sorties.
Un algorithme de clustering est utilisé par la suite pour grouper les noms de paramètres dans des classes sémantiques utilisées ultérieurement pour calculer la similarité inter paramètres (Inputs/Outputs).
URBE [PP09] présente une approche de découverte basée sur la sémantique SAWSDL des services. Une approche de découverte basée WSMO est présentée dans [LDT07]. Les auteurs utilisent la théorie des graphes pour évaluer la similarité sémantique entre deux concepts.
LUMINA [LVM+06] est développé dans le cadre du projet de METEOR-s [POSV04] ; il convertit les descriptions WSDL-S en structures enregistrées dans l'annuaire UDDI. Le matching sémantique de LUMINA se base sur le calcul de longueurs de chemins qui relient les concepts d'une ontologie de domaine. LUMINA fait aussi une recherche syntaxique par simples mots clés.
[SSS07] propose des mesures de similarité basées sur le rappel et la précision pour apparier

les services et la requête. Les auteurs prennent en compte les paramètres suivants : les entrées, les sorties, les préconditions et les effets.

Approches Hybrides (Logiques et Non Logiques)
Les approches hybrides combinent les deux aspects évoqués ci-dessus. En effet, certaines approches logiques compensent les limites des approches non-logiques et vice-versa.
FUSION [KP08] allie le raisonnement logique à la recherche par mots-clés. La découverte à travers FUSION commence par un pré-filtrage à base de mots clés puis progresse avec un raisonnement logique.
OWLSMX [KFS06] ce système applique six filtres successifs pour chaque paire (requête-service), ces filtres sont : Exact, Plug-in, Subsumes, Subsumed-by, Hybrid-Subsumed-by et Nearest Neighbor. Les quatre premiers filtres sont purement logiques alors les deux derniers sont hybrides parce qu'ils combinent le raisonnement logique et les mesures de similarités textuelles. L'algorithme générique du système OWLSMX est donné comme suit :

Algorithme de Matching
1 : fonction MATCH(Requete R,α)
2 : Var : Resultat, DegreDeMatch, Filtres-hybrides ={*HYBRID-SUBSUMED-BY, NEAREST-NEIGHBOUR*}
3 : pour tout (S, dom) $\in$ Candidats-entrées-S(entréesR) et (S, dom') $\in$ Candidats-sorties-S(sortiesR) faire
4 : DegreDeMatch $\leftarrow$ MIN(dom, dom')
5 : si DegreDeMatch $\geq$ NEAREST-NEIGHBOUR et (DegreDeMatch Filtres-hybrides ou SIMIR(R,S) $\geq \alpha$) alors
6 : Resultat := Resultat $\cup$ {(S, DegreDeMatch, SIMIR(R,S))}
7 : fin si
8 : fin pour
9 : retourner Resultat
10 : end fonction

L'algorithme cherche des services S publiés qui s'apparient avec la requête R. Il retourne un ensemble de paires (S, degreDeMatch, SIMIR(R, S)) avec un degré d'appariement «degreeOfMatch» différent de FAIL et une similarité syntaxique dépassant le seuil de l'utilisateur α. La fonction Candidats-entrées-S (entréesR) retourne le score minimal de tous les appariements des entrées de S avec les entrées de la requête R, quant à la fonction Candidats-sorties-S (sortiesR), elle retourne le score minimal de tous les appariements des sorties de R avec celles de S.

WSMO-MX [KK06] accepte en entrée des services spécifiés en WSML. L'approche d'appariement est une combinaison d'appariement sémantique hybride de OWLS-MX, d'appariement de graphes orienté objet du DSD-Matchmaker [KKR04] et d'appariement intentionnel de service [RKL$^+$05].
Les degrés d'appariement sont calculés par l'agrégation des affectations ou «valuations»

de quatre éléments :

— l'appariement des types ontologiques (la relation hiérarchique entre concepts logiques).
— l'appariement logique (basé sur des instances) des contraintes spécifiées en F-logic.
— l'appariement des noms de relations.
— la mesure de similarité syntaxique.

SAWSDL-MX [KK08] accepte en entrée des services spécifiés en SAWSDL. Ce matchmaker est inspiré par OWLS-MX et WSMO-MX. La découverte utilise à la fois :
— une approche d'appariement logique basée sur le raisonnement par subsomption,
— une approche d'appariement syntaxique basée sur les techniques de recherche d'information.

Partant du fait qu'une description SAWSDL est une extension d'une description WSDL, l'appariement recouvre les éléments de description suivants : portType, operation, input, output. Le matchmaker SAWSDL-MX combine les deux types d'approches (similarité syntaxique ou textuelle et les filtres logiques). En effet les services ayant le même score logique seront classés avec la similarité textuelle.
iMatcher [KB08] est un matchmaker hybride qui calcule la proximité entre les descriptions OWL-s Profile, en adoptant d'une part les similarités textuelles, et d'autre part la subsomption logique.

2.2.2 Approches Non Fonctionnelles

2.2.2.1 Approches à base de Réputation et de Confiance

[YWYC09] se base sur la réputation comme moyen de découverte des services. De façon générale, les auteurs adoptent le feed-back des communautés d'utilisateurs pour sélectionner les services.
Le système proposé dans [WCL+06] prend en compte les exigences et les préférences des utilisateurs pour sélectionner les services convenables à l'aide des systèmes flous.
Dans [LCTC10], les auteurs proposent une version floue de la méthode de décision multi objective TOPSIS pour sélectionner les services. Ils considèrent cinq groupes de critères de QoS : les critères transactionnels, de performance (runtime), de coûts, de gestion de configuration et de sécurité. TOPSIS [HY81] est considérée comme l'une des méthodes de décision multicritères les plus populaires pour déterminer la meilleure solution. La méthode TOPSIS calcule la distance la plus courte de la solution idéale positive(PIS) et la distance la plus longue de la solution idéale négative (NIS).
Le PIS est une solution fictive qui maximise les critères positifs et minimise les critères de coût. Le NIS est une solution fictive qui maximise les critères de coût et réduit au minimum les critères positifs.

2.2.2.2 Extensions du registre des services Web

Certaines approches s'axent sur les méthodes améliorant l'accès à l'annuaire UDDI comme l'ajout d'un langage de requête structuré (une variante du SQL). [LDRW04] propose l'introduction d'un facilitateur de qualité (broker) qui sépare l'UDDI et l'utilisateur ; ce facilitateur gère trois critères de qualité de service : la fiabilité, la performance et le coût. Il offre trois scores de matching : Gold, Silver et Bronze. Le modèle de données de l'UDDI est étendu pour prendre en compte la QoS. L'appariement requête-service est fait à travers un calcul d'une corrélation floue.

2.3 Synthèse

Le tableau 2.1 illustre les différences entre quelques approches fonctionnelles de matching présentées dans ce chapitre. Nous réduisons le champ d'évaluation en omettant les critères de scalabilité et de performance calculées souvent par des métriques...etc. Nous cherchons principalement à dégager de cette comparaison la meilleure approche de matching qui puisse considérer les éléments reflétant le comportement réel des services Web. Cette comparaison est donc basée sur les critères suivants :

- Les paramètres d'entrées /sorties du service **IO**.
- Les préconditions et des effets produits **PE**.
- **Les propriétés non fonctionnelles** : Qualité de service.
- **Le temps de réponse** : la durée requise pour traiter une requête.
- **Le matching multi-stage** : c'est la capacité d'une approche d'agréger plusieurs scores de matching pour produire un seul score final.
- **Support de l'UDDI** : prise en charge du matching syntaxique.

Ce tableau reflète de manière assez claire l'avantage que prennent les approches hybrides par rapport aux autres approches. En effet, les approches de matching telles que URBE et [LDT07] reposent sur les langages WSMO et SAWSDL présentés dans la première partie de l'état de l'art. Ces langages sont rattrapés une fois de plus par leurs faiblesses puisque le processus de découverte est intrinsèquement lié au processus de description. Fonder un matching sur ces derniers revient à devoir faire face aux problèmes de manque de sémantique et de dépendance compte tenu des ontologies auxquelles ils sont rattachés (WSMO dans ce cas).
Quant aux approches logiques, elles sont plus avancées dans le processus de matching puisqu'elles fondent leurs raisonnements sur des logiques plus particulièrement l'approche [HPS+03] qui estime dans son matching non seulement, les entrées et sorties du service mais aussi les préconditions et effets produits par ce dernier. Ce que l'on peut reprocher à ces approches c'est leur non-considération des propriétés non-fonctionnelles d'un service et surtout leur non-compatibilité avec le standard UDDI auquel se réfère toute la

communauté des services Web dans le processus de découverte.

Catégorie	Approches	Critères					
		Eléments utilisés			Matching Multi-Stage	Support de l'UDDI	Tempsd'exécution
		IO	PE	P-Non Fonctionnelles			
Logique	[HPS+03]	√	√	-	-	-	Moyen – Elevé
	[PKPS02]	√	-	-	-	-	
Non-Logique	[LDT07]	√	-	-	√	√	Faible
	URBE	√	-	-	√	√	
Hybride	WSMO-MX	√	√	-	√	-	
	OWLS-MX2	√	-	-	√	-	
	iMatcher	√	-	-	√	-	
	SAWSDL- MX	√	-	-	√	-	
	Lumina	√	-	-	-	√	Elevé
	FUSION	√	-	√	√	√	Non disponible

TABLE 2.1 – Comparaison entre les approches fonctionnelles de découverte des services Web

√ indique la satisfaisabilité du critère par l'approche et - l'inverse.

Enfin, quelques approches combinant à la fois les forces de la première et seconde catégorie ont pu satisfaire presque la majorité des critères telle que l'approche FUSION qui s'avère très performante dans le matching à la fois syntaxique et sémantique entre services.
De notre point de vue, nous estimons qu'il est possible de pousser encore le raisonnement logique d'une approche de matching hybride à considérer davantage d'aspects, notamment l'aspect comportemental d'un service Web (la dynamique du service) et ce, en terme de sémantique opérationnelle.

2.4 Conclusion

Dans ce chapitre nous avons présenté quelques approches dédiées à l'optimisation de la phase de découverte de services Web et celle de sélection. Ces approches, qu'elles soient fonctionnelles (logique, non logique, hybride) ou non fonctionnelles, ont témoigné d'un apport signifiant en terme de précision et de performance lors des recherches de services.
Nous avons dressé une comparaison multi-critères entre ses approches qui reflète les avantages et inconvénients de chacune d'elles. Cependant, bien que ces approches soient efficaces dans certains contextes, elles présentent des limites lorsqu'il s'agit de matching comportemental. En effet, très peu de solutions ont été proposées dans ce sens, ceci revient principalement à l'incompatibilité sémantique entre les bases de matching des approches proposées et à la sémantique des langages de spécification comportementaux.

Chapitre 3

Composition des Services Web

3.1 Introduction

La composition des services Web est le mécanisme qui permet l'intégration des services dans une application. Le résultat de la composition de services est un nouveau service appelé service composite (Figure 3.1). Dans ce cas, la composition est dite composition récursive ou hiérarchique.

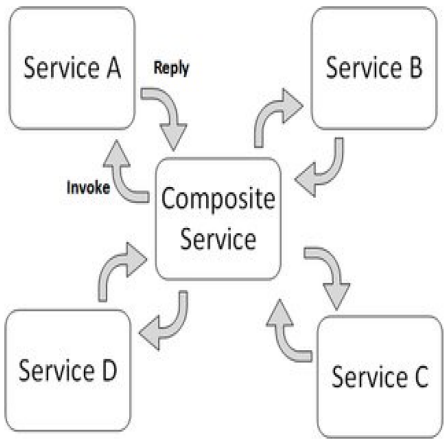


FIGURE 3.1 – Composition de services Web

La composition des services est aujourd'hui un sujet de grand intérêt autant pour le monde de la recherche que pour le monde industriel. De nombreuses recherches visent à développer des modèles de composition de services et à fournir les outils nécessaires à cette composition.
Un service Web est dit composé ou composite lorsque son exécution implique des interactions avec d'autres services Web afin de faire appel à leurs fonctionnalités. La composition

de services Web spécifie quels services ont besoin d'être invoqués, dans quel ordre et comment gérer les conditions d'exception.
Dans ce chapitre, nous discutons les différents langages dédiés à la composition de services Web après l'introduction du concept de composition et la spécification de ses différents modes et types. Nous donnons une attention particulière à la composition dynamique de services Web du fait de sa relation avec le travail que nous élaborons.Toutes les approches de composition présentées dans ce chapitre sont soumises à une comparaison multi-critères afin de mettre en évidence leurs atouts et limites.

3.2 Types de composition

3.2.1 Composition statique

La composition statique a lieu au moment de la conception d'une application. Ainsi, les composants concernés sont choisis, liés et assemblés avant d'être déployés. Une telle composition est adaptée aux environnements fermés où les composants n'évoluent pas souvent. Ce type de composition rend les applications peu flexibles et parfois inappropriées aux exigences des clients. La composition statique des services Web peut se faire de deux manières : par Orchestration ou Chorégraphie [Pel03].

3.2.1.1 Orchestration

L'orchestration de services Web exige de définir l'enchaînement des services Web selon un canevas prédéfini et de les exécuter selon un script d'orchestration. Les services Web n'ont pas de connaissance dans une composition, seul le coordinateur de l'orchestration (l'orchestrateur) a besoin de cette connaissance. Le script et le canevas décrivent les interactions entre services en identifiant les messages et en spécifiant la logique et les séquences d'invocation. Ce type de composition permet de centraliser l'invocation des services Web composants.
La Figure 3.2 illustre une orchestration de service. Le service coordinateur contrôle tous les services Web impliqués et coordonne l'exécution de leurs différentes opérations.

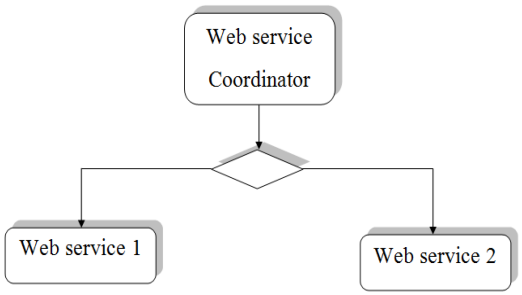


FIGURE 3.2 – Orchestration de services Web

Langages de composition par orchestration

Il existe plusieurs langages de définition pour la composition statique des services Web. Nous présentons dans cette section les plus utilisés :

— **XLANG** [Tha01] créé par Microsoft est une extension de WSDL. Ce langage fournit en même temps un modèle pour une orchestration des services et des contrats de collaboration entre ceux-ci. XLANG a été conçu avec une base explicite de théorie de calcul. Les actions sont les constituants de base d'une définition de processus de XLANG. Les quatre types d'opérations de WSDL (requête/réponse, sollicitation de la réponse et notification) peuvent être employés comme actions. XLANG ajoute deux autres genres d'action : arrêts (date-limite et durée) et exceptions.

— **WSFL** Web Service Flow Language [L+01] est un langage basé sur XML pour la description des services Web composites. WSFL considère deux types de compositions de services Web :
Le premier type indique le modèle approprié d'utilisation d'une collection de services Web de telle manière que la composition résultante décrive comment réaliser le but particulier d'un business ; le résultat est une description d'un processus métier.
Le deuxième type indique le modèle d'interaction d'une collection de services Web ; dans ce cas, le résultat est une description des interactions globales associées.

— **BPEL4WS** Business Process Exécution Language for Web Services [JMS06] appelé aussi BPEL ou BPELWS est un langage dédié à décrire les processus métier basé sur XML. Il a été conçu pour permettre de charger/partager les données distribuées, même à travers des organismes multiples, en employant une combinaison de services Web. BPEL décrit l'interaction des processus métiers basés sur les services Web à la fois au sein des entreprises et entre elles. Les entreprises utilisatrices du langage BPEL peuvent ainsi définir leurs processus métiers et en garantir l'interopérabilité non seulement à l'échelle de l'entreprise mais également avec leurs partenaires commerciaux, au sein d'un environnement de services Web. La spécification BPEL combine et remplace WSFL et XLANG.

3.2.1.2 Chorégraphie

Contrairement à l'orchestration, la chorégraphie n'a pas de coordinateur central. Chaque service Web mêlé dans la chorégraphie sait exactement quand ses opérations doivent être exécutées et avec qui l'interaction doit avoir lieu. Elle est associée à l'échange de messages entre services Web plutôt qu'à un processus métier exécuté par un seul partenaire.
La chorégraphie est un effort de collaboration dans lequel chaque participant du processus

décrit l'itération qui l'appartient. Elle trace la séquence des messages qui peut impliquer plusieurs services Web. La collaboration dans la chorégraphie des services Web peut être représentée de la manière suivante :

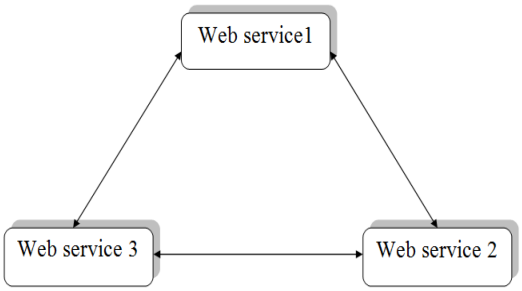


FIGURE 3.3 – Chorégraphie de services Web

Les principaux langages de chorégraphie étudiés dans cette thèse sont : WSCL et WSCI.
Langages de composition par chorégraphie

- **WSCL** [BBB+02] propose de décrire, à l'aide de documents XML, les services Web en mettant l'accent sur les *conversations* de ceux-ci. En outre, les messages à échanger sont pris en compte. WSCL a été pensé pour être déployé conjointement avec WSDL. Les définitions WSDL peuvent être manipulées par WSCL pour décrire les opérations possibles ainsi que leur chorégraphie. En retour, WSDL fournit les concrétisations vers des définitions de messages et des détails techniques pour les éléments manipulés par WSDL.
- **WSCI** [C+02] est un langage reposant sur XML. Il propose de se focaliser sur la représentation des services Web en tant qu'interfaces décrivant le flux des messages échangés (la chorégraphie des messages). Il propose ainsi de décrire le comportement externe observable du service. Pour cela, WSCI propose d'exprimer les dépendances logiques et temporelles entre les messages échangés à l'aide de contrôles de séquences, corrélation, gestion de fautes et transactions. WSDL et ses définitions abstraites sont réutilisées afin de pouvoir également décrire par la suite les modalités de concrétisation des éléments manipulés pour modéliser un service.

3.2.2 Composition dynamique

On appelle composition dynamique l'agrégation de services Web permettant de résoudre un objectif précis soumis par un utilisateur en prenant en compte ses préférences. Cette composition peut se faire avant ou pendant l'exécution des services Web. A l'heure actuelle, il n'existe pas de standard pour décrire la composition dynamique de services : les recherches sont principalement académiques.
Les différentes approches existantes pour la composition dynamique de services Web

peuvent être regroupées : les approches basées sur les workflow, les approches basées sur les techniques de l'intelligence artificielle et les approches de spécification formelles.
Les approches orientées Workflow ayant déjà été présentées dans la section précédente, nous étudions donc, dans ce qui suit les deux points restants :

3.2.2.1 Approches orientées Intelligence Artificielle

Comme nous l'avons déjà mentionné précédemment, un service Web dans une description OWL-S peut être spécifié à l'aide de ses préconditions et effets. Ces concepts sont très proches de ceux de la planification.
Plusieurs branches de l'IA intègrent le processus de composition ; nous notons notamment :

— **Calcul situationnel** : dans cette approche, le problème de la composition est abordé de la façon suivante : la requête de l'utilisateur et les contraintes des services sont représentées en terme de prédicats du premier ordre dans le langage de calcul situationnel. Les services sont transformés en actions (primitives ou complexes) dans le même langage. Puis, à l'aide de règles de déduction et de contraintes, des modèles sont ainsi générés et sont instanciés à l'exécution à partir des préférences utilisateur.

— **Composition avec SMA** «Systèmes Multi-Agents» : la composition de services peut être implémentée aussi en utilisant des SMA. Dans cette approche, chaque agent présente un service et traite une partie de la requête de l'utilisateur en utilisant ses propres capacités. Les auteurs de [MBK05]travaillent sur un système multi-agents pour la composition des services basé sur la concurrence entre coalitions de services : les agents représentant les services se contactent les uns les autres pour proposer leurs services en fonction de leurs capacités de raisonnement et ainsi former des coalitions d'agents capables de résoudre les buts fournis par l'agent utilisateur. Puis les différentes coalitions vont faire l'offre la plus compétitive possible. Chaque solution reçoit une note de l'agent utilisateur. La solution ayant le plus haut score sera choisie.
[KM08] présente deux modèles de composition de services Web sémantiques en se basant sur les SMA. Les deux modèles se distinguent par l'utilisation d'un coordinateur dans le processus de composition ou pas. Dans le premier modèle où le coordinateur est absent, la requête de l'utilisateur est décomposée par le système en activités atomiques : «task1, task2, ..., taskn». Le système sélectionne ensuite pour chaque activité atomique un agent fournisseur représentant un service Web : «AF1, AF2, ..., AFn». L'agent utilisateur négocie avec chaque agent fournisseur et lui affecte l'activité associée. Cependant dans le deuxième modèle un agent coordinateur prend la responsabilité de contrôler tout le système de composition.

— **Composition par planification** elle permet d'ordonner les services Web ayant des pré-conditions et des effets. De nombreuses études ont été faites pour implémenter la composition de services Web comme résolution d'un problème de planification.

[Pee04] propose une approche basée sur le langage PDDL dans laquelle, après la création du domaine de planification à partir de la description sémantique, un planificateur est choisi parmi plusieurs en fonction des instructions PDDL utilisées dans le domaine ou de la complexité du but à atteindre.
[MBE03] présente une technique pour générer des services composites à partir de descriptions déclaratives de haut niveau. Cette méthode utilise des règles de composabilité pour déterminer dans quelle mesure deux services sont composables.
L'approche proposée se déroule en quatre phases : premièrement, une phase de spécification offre une spécification de haut niveau de la composition désirée en utilisant le langage CSSL [AFJ+68](Composite Service Specification Langage). Ensuite, la phase de correspondance utilise des règles de composabilité pour générer des plans conformes aux spécifications du service demandé. Si plus d'un plan est généré, une sélection est effectuée par rapport à des paramètres de qualité de la composition. La dernière phase est la phase de génération : une description détaillée du service composite est automatiquement générée et présentée au demandeur. Les règles de composabilité considèrent les propriétés syntaxiques et sémantiques des services Web.

3.2.2.2 Approches Formelles

Les modèles sémantiques formels apparaissent comme l'une des meilleures solutions pour spécifier la composition de services Web.
Différents formalismes, ont été proposés à cet égard. Nous pouvons citer : les automates, les diagrammes d'activité UML, les algèbres de processus et les réseaux de Petri.

— **Automates** ce sont des modèles très utilisés pour la spécification formelle des systèmes complexes. Un automate est composé d'un ensemble de nœ uds représentant les états et un ensemble d'arcs étiquetés décrivant les transitions entre les états. Pour modéliser la composition de service à l'aide d'un automate, nous assignons généralement un état à chaque invocation de service, un autre état pour indiquer la fin du déroulement (exécution) de ce service et un événement à chaque réponse. Plusieurs contributions ont été faites dans ce sens.
Le modèle «Roman»[CDGL+08] basé sur les automates à états finis (FSA), le modèle Colombo[BCDG+05] qui est une extension du modèle Roman qui ajoute la prise en charge des données et de la communication basée sur l'échange de messages. Une autre approche nommée COCOA a été introduite dans [MGI06] l'idée principale de COCOA est de convertir les processus OWL-S en FSA. Ceci permet de transformer la composition de services en un problème d'analyse d'automates.

— **Diagrammes d'activités UML** UML[RJB04] est considéré de nos jours comme un standard pour la modélisation orientée objet des systèmes complexes. Les diagrammes UML sont divisés en trois catégories de classes : les diagrammes structu-

rels modélisent l'aspect statique du système à concevoir comme les diagrammes de classes ou les diagrammes d'objets ; les diagrammes comportementaux comme les diagrammes d'activité ou les diagrammes d'états-transitions et les diagrammes d'interaction (dynamiques) comme les diagrammes de séquence. L'un des diagrammes les plus utilisé dans le domaine de la composition des services Web est le diagramme d'activité. Pour modéliser la composition à l'aide de ce dernier, l'exécution d'un service est représentée généralement par une action. La transition vers cette action modélise l'appel au service et celle en sortie indique la fin de l'exécution du service en question.
Dans [SGS04] les auteurs présentent une approche pour la modélisation de processus métier à l'aide du diagramme d'activité et implémentent la composition à travers l'utilisation du langage BPEL. D'autres auteurs ont présenté un travail basé sur l'utilisation d'UML 1.4 pour le développement de services Web composés en suivant les principes de la méta-modélisation.

— **Algèbre des processus** les algèbres de processus sont des familles de langages formels permettant de modéliser les systèmes complexes, en particulier les systèmes distribués et concurrentiels. Plusieurs travaux de recherche ont utilisé ces formalismes pour la composition de services Web, nous citons à titre d'exemple :
[CCCV06] propose une technique de formalisation basée sur CCS pour la chorégraphie de service Web. L'algèbre CCS a été utilisée pour la modélisation et la spécification des services Web afin de raisonner sur les propriétés comportementales de la composition.
Dans [LM07] la sémantique du langage d'orchestration BPEL est cette fois-ci spécifiée en utilisant le π-calcul. Cependant, des parties de BPEL telle que la gestion des données ne sont pas traitées dans ce travail. Afin de prendre en considération les échanges de données au cours des interactions et compositions dynamiques de services Web, d'autres travaux basés sur des algèbres plus avancées ont été présentés.
[Fer04] propose une transformation bidirectionnelle pour passer du BPEL au LOTOS et vice versa. Cette translation inclut la gestion des exceptions et permet la vérification des propriétés temporelles.
Dans [SS05] les auteurs ont utilisé aussi le langage LOTOS pour modéliser la composition de services Web avant de procéder à la vérification formelle à l'aide de l'outil CADP9.

— **Réseaux de Petri** les réseaux de Petri (RdPs) [KC04] représentent un moyen de modélisation du comportement des systèmes dynamiques à événements discrets. Un des avantages de ces modèles est leur représentation graphique qui facilite grandement la modélisation conceptuelle des différents systèmes. Les RdPs sont très réputés dans le domaine de gestion des processus métiers (BPM) en raison de la variété des processus et des flux de contrôle qu'ils peuvent modéliser.
Dans [HSS05] une traduction rigoureuse de BPEL en RdPs est réalisée. Cette transformation couvre toutes les structures de contrôle et les actions de communication

de BPEL. Les modèles obtenus ont été utilisés pour vérifier les processus BPEL par le biais de l'outil WofBPEL[OVvdA+05]. Dans ce formalisme, les arcs peuvent être étiquetés par des constantes ou des variables spécifiant les paramètres de jetons. À l'instar des arcs, des conditions peuvent être associées à des transitions. Ces conditions précisent pour quelle valeur des paramètres la transition est franchissable.
[HB03] traite également ce problème par la modélisation des services Web et de leur composition via des réseaux de Petri colorés.
Dans [XFZ10] les auteurs ont utilisé les réseaux de Petri temporisés pour modéliser les flux de services Web et leur description WSDL. Ce modèle est conçu pour assister le développeur afin d'assurer l'exactitude de la spécification en termes de non-blocage et de terminaison correcte de la transaction.

3.3 Synthèse

Critères	**Approches basées Workflows**	**Approches basées Intelligence Artificielle**	**Approches basées Méthodes Formelles**
Niveau d'automatisation	-Moyen- Seule la recherche et la liaison aux services se font de manière automatique	-Elevé- Tout le processus est réalisé de manière automatique	Moyen-Elevé- Le Processus est partiellement automatisé
Niveau de dynamicité	-Moyen- les modèles sont générés statiquement seule la liaison aux services est dynamique	-Elevé- possibilité de génération de plan de manière dynamique	-Elevé- Génération de modèles de composition graphiques de manière dynamique
Formulation de la requête utilisateur	-Bas niveau- (langage de flux)	Moyen à élevé Langages logiques	-Elevé- Spécification algébriques..
Support sémantique et niveau d'abstraction	-Bas niveau- (descriptions syntaxiques : WSDL, WSFL..)	-Elevé- Formalismes logiques, DAML-s, OWL-s.	-Elevé- Logique de réécriture, Algèbre de processus
Reconfiguration dynamique	non pris en charge	Non pris en charge	Enviseagble

TABLE 3.1 – Comparaison entre les approches de composition des services Web

Les critères requis pour réaliser une composition de services de qualité peuvent se résumer en :

— Pouvoir composer de manière dynamique et automatique.
— Avoir une bonne compréhension des besoins formulés par l'utilisateur.
— Avoir un support sémantique reflétant de manière explicite le vrai comportement d'un service.
— Pouvoir reconfigurer dynamiquement l'état d'un service en cas de défaillance.

Nous constatons à travers le tableau ci-dessus que les méthodes basées Workflows sont inadaptées pour mener à bien le processus de composition de services Web, tandis que les approches basées IA et les modèles mathématiques issus des méthodes formelles offrent une spécification efficace de ce processus. Notons qu'un détail assez important échappe aux approches orientées IA, c'est la notion de reconfiguration dynamique. Cet aspect est pris en charge par les méthodes formelles du fait de leur fondement sur les logiques computationnelles.
A l'issue de cette comparaison nous estimons que les méthodes formelles sont les mieux placées pour spécifier une tâche aussi complexe qu'est la composition de services Web dynamique.

3.4 Conclusion

Dans ce chapitre, nous avons présenté la notion de composition de services Web, ses types ainsi que les différentes approches qui ont été conçues pour traiter ce processus qu'elles soient statiques ou dynamiques. Nous avons accordé une attention particulière aux langages formels de spécification de la composition. En effet, ces langages ou formalismes comparés aux autres méthodes proposées, sont les plus simples à utiliser de par leur notion graphique et assez complets pour représenter tout comportement du service. L'avantage majeur que l'on associe à ce type de langages et qu'ils sont rattachés à des environnements formels permettant leur analyse et vérification automatique.

Chapitre 4

Logique de réécriture et langage Maude

Ce chapitre a pour but de présenter les notions théoriques constituant la base de nos contributions à savoir : la logique de réécriture, son environnement d'application Maude et la théorie des catégories.
Nous introduisons d'abord la logique de réécriture en présentant ses formules, théories et règles de déduction. Le langage Maude, quant à lui, est défini par sa structure et son mécanisme de vérification. Enfin, nous présentons la théorie des catégories à travers ses notions de base et montrons comment une catégorie peut servir de modèle sémantique à une théorie de réécriture.

4.1 Logique de réécriture

La réécriture est un paradigme général d'expression du calcul dans diverses logiques computationnelles. La logique de réécriture en est une [Mes 92]. Elle a été proposée comme cadre logique dans lequel d'autres logiques peuvent être représentées et comme cadre sémantique pour spécifier les systèmes concurrents dans des domaines variés. Le calcul dans un système concurrent est exprimé à travers la déduction logique qui est intrinsèquement concurrente. La réécriture d'un terme consiste à le remplacer par un terme équivalent conformément aux lois de l'algèbre des termes. Cette logique formalise le processus de réécriture pour calculer une relation de «réécrivabilité» entre les termes algébriques. Elle permet notamment de raisonner sur des changements complexes possibles correspondant aux actions atomiques axiomatisées par les règles.
Elle offre également, des techniques d'analyse formelle permettant de prouver des propriétés du système à spécifier et de raisonner sur ses changements.
Plusieurs modèles de concurrence ont été intégrés dans la logique de réécriture, nous retrouvons : Les systèmes à transitions étiquetées [MM 93], les réseaux de Petri [MM 95], CCS [Mes 92], la Machine Chimique Abstraite [IW95], les ECATNets et leurs différentes

extensions [BMSB92], [LSB$^+$11], [LB14]...etc.
Dans ce qui suit, nous présentons les différentes notions formelles nécessaires à la compréhension de la logique de réécriture : la théorie de réécriture et les termes de preuve.

4.1.1 Théorie de réécriture

Définition 4.1. *théorie de réécriture étiquetée*
Une théorie de réécriture étiquetée est un quadruplet $\mathcal{R}$= (Σ,E,L,R) tel que :

- *Σ est un ensemble de symboles de fonctions et de sortes,*
- *E est un ensemble de Σ-équations,*
- *L est un ensemble d'étiquettes,*
- *R est un ensemble de règles de réécriture $R \subseteq L \times (T_{\Sigma,E}(X))^2$ où chaque règle est un couple d'éléments. Le premier est une étiquette, le second est une paire de classes d'équivalence de termes modulo les équations, sachant que X est un ensemble infini et dénombrable de variables.*

Etant donné une théorie de réécriture, l'ensemble des formules impliquées par cette théorie est défini par les règles de déduction de la logique de réécriture. Ces règles formalisent la notion de la réécriture des termes en définissant comment les preuves des théorèmes sont formées.
La structure statique d'un système est décrite par la spécification algébrique (Σ, E), tandis que la structure dynamique est décrite par les règles de réécriture R.

4.1.2 Règles de réécriture

Définition 4.2. *Les axiomes de base dans la logique de réécriture sont les règles de réécriture. Deux lectures complémentaires sont valables pour une règle de réécriture de la forme $t \rightarrow t'$; t et t' étant des termes algébriques :*
D'un point de vue calcul, la règle de réécriture est interprétée comme une transition locale dans un système concurrent ; c.à.d. t et t' décrivent des patrons de sous-états pouvant apparaître dans l'état distribué d'un système concurrent. La règle sert à exprimer une transition locale pouvant avoir lieu dans un tel système et provoquant le changement d'un fragment local de l'état de l'instance du patron t vers l'instance correspondante du patron t'.

D'un point de vue logique, la règle de réécriture $t \rightarrow t'$ est interprétée comme une règle d'inférence permettant d'inférer des formules de la forme t' à partir de formules de la forme t.

4.1.3 Règles de déduction

Etant donné une théorie de réécriture $\mathcal{R}$, nous disons que $\mathcal{R}$ implique une formule $[t] \rightarrow [t']$ et nous écrivons $\mathcal{R} \vdash [t] \rightarrow [t']$ si et seulement si $[t] \rightarrow [t']$ est obtenue par l'application des règles de déduction suivantes (Figure 4.1) : La règle de remplacement

identifie toutes les règles de réécriture dont le membre gauche correspond à un sous-terme de l'état global courant. Les sous-termes non identifiés évoluent en eux même par application de la règle de réflexivité.

1. Reflexivité : pour chaque $[t] \in T_{\Sigma(X)}$,

$$\frac{}{[t] \longrightarrow [t']}$$

2. Congruence : pour tout $f \in \Sigma_n$, $n \in \mathbb{N}$

$$\frac{[t_1] \longrightarrow [t'_1], ..., [t_n] \longrightarrow [t'_n]}{[f(t_1, ..., t_n)] \longrightarrow f[(t'_1, ..., t'_n)]}$$

3. Remplacement : pour chaque règle de réécriture

$$r : [t(x_1, ..., x_n)] \longrightarrow t'[(x_1, ..., x_n)] \; if u_1(\overline{x}) \longrightarrow v_1(\overline{x}) \wedge ... \wedge u_n(\overline{x}) \longrightarrow v_n(\overline{x}) \, \text{dans} \, R$$

$$\frac{[w_1] \longrightarrow [w'_1], ..., [w_n] \longrightarrow [w'_n]}{[u_1(\overline{w_1}/\overline{x})], ..., [u_n(\overline{w_n}/\overline{x})]}$$
$$[[t(\overline{w}/\overline{x})] \longrightarrow [t'(\overline{w}/\overline{x})]]$$

4. Transitivité :

$$\frac{[t_1] \longrightarrow [t_2] \;\; [t_2] \longrightarrow [t_3]}{[t_1] \longrightarrow [t_3]}$$

FIGURE 4.1 – Règles de déduction de la logique de réécriture

La règle de congruence permet de composer les effets locaux des différentes règles de base identifiées par la règle de remplacement pour construire le nouvel état du système. Elle exprime l'évolution concurrente des différents sous-termes d'un état global. Ces étapes sont réitérées jusqu'à ce qu'il n'y ait plus de règle applicable.
Enfin, la règle de transitivité construit la séquence des réécritures faites en partant du terme initial jusqu'au terme final.

Exemple 4.1.

Un exemple de système concurrent intégré au niveau de la logique de réécriture est montré à travers le réseau de Petri ordinaire suivant :

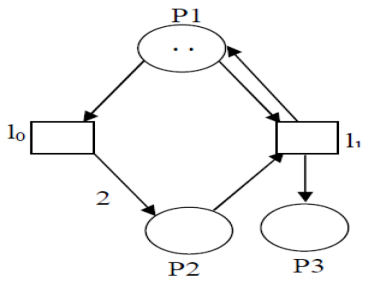


FIGURE 4.2 – Exemple d'un réseau de Petri

Ce réseau de Petri peut être représenté par la théorie de réécriture $\mathcal{R}_{RdP}$=(Σ, E, L, R), tels que :
$\sum$=($place, marking\}, p_1, p_2, p_3 :\rightarrow place, null :\rightarrow place$,_: $place \rightarrow marking$,
_._: $marking \times marking \rightarrow marking$)
$E = \{x.null = x, x.y = y.x, x.(y.z) = (x.y).z\}$
$L = \{l_0, l_1\}$
R =$\{$l$_0$: [p$_1$]$\rightarrow$[p_2. p_2], l_1 : [p_1.p_2]$\rightarrow$[p_1.p_3]$\}$
Dans cette théorie de réécriture, les places et le marquage du réseau de Petri sont spécifiées par le biais des sortes place et marking. Ainsi, l'ensemble des opérations suggérées sert à générer les places et le marquage du réseau. L'ensemble des règles de réécriture permet de décrire les transitions du réseau.
La transition l_0 permet de consommer un jeton de la place p1 et de générer deux jetons au niveau de la place p2. La transition l_1 permet de consommer un jeton de la place p1 et un autre de la place p2 et de générer un jeton au niveau de la place p1 et un autre dans la place p3.
Un comportement simple de ce réseau peut être vu par exemple, par son évolution de l'état [p1.p1] vers un autre état [p1.p3.p3].

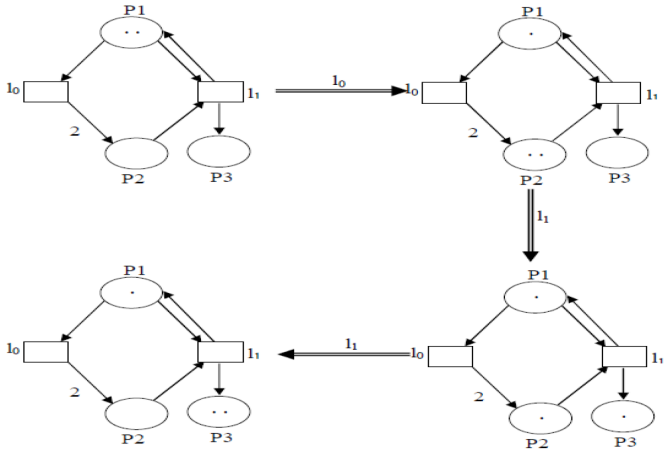


FIGURE 4.3 – Exemple d'évolution des états d'un RdP

Ce comportement est déduit par la preuve : $\mathcal{R}_{RdP} \vdash [p1.p1] \rightarrow [p1.p3.p3]$
$(p1.l_0); (l_1.p2); (l_1.p3)$ noté généralement : $l_0; l_1; l_1$ et obtenu ainsi :

$$\frac{p_1 : [p_1] \rightarrow [p_1] \qquad l_0 \rightarrow [p_2.p_2]}{p_1.l_0 : [p_1.p_1] \rightarrow [p_1.p_2.p_2]} \quad \text{Congruence}$$

$$\frac{l_1 : [p_1.p_2] \rightarrow [p_1.p_3] \qquad p_2 : [p_2] \rightarrow [p_2]}{l_1.p_2 : [p_1.p_2.p_2] \rightarrow [p_1.p_3.p_2]} \quad \text{Congruence}$$

$$\frac{p_1.l_0 : [p_1.p_1] \rightarrow [p_1.p_2.p_2] \quad l_1.p_2 : [p_1.p_2.p_2] \rightarrow [p_1.p_3.p_2]}{(p_1.l_0); (l_1.p_2) : [p_1.p_1] \rightarrow [p_1.p_3.p_2]} \quad \text{Transitivité}$$

$$\frac{p_3 : [p_3 \rightarrow [p_3] \qquad l_1 : [p_1.p_2] \rightarrow [p_1.p_3]}{l_1).p_3 : [p_1.p_2.p_3] \rightarrow [p_1.p_3.p_3]} \quad \text{Congruence}$$

$$\frac{(p_1.l_0); (l_1.p_2) : [p_1.p_1] \rightarrow [p_1.p_3.p_2] \quad l_1.p_3 : [p_1.p_2.p_3] \rightarrow [p_1.p_3.p_3]}{(p_1.l_0); (l_1.p_2); (l_1.p_3) : [p_1.p_1] \rightarrow [p_1.p_3.p_3]} \quad \text{Transitivité}$$

Nous constatons qu'à ce niveau, le comportement d'un système concurrent (modélisé dans notre cas par un réseau de Petri) est décrit formellement par des déductions dans cette logique.

4.2 Maude

La programmation déclarative a eu le mérite de faciliter considérablement les aspects conceptuels de la programmation. Cependant, la logique sur laquelle se basent généralement les langages déclaratifs ne prend pas en charge les caractéristiques inhérentes aux systèmes concurrents, notamment l'action et le changement. Afin de combler ces insuffisances Meseguer a défini un langage basé sur la logique de réécriture dénommé : Maude [CELM96].
Maude est un langage déclaratif autour duquel est construit un système performant. Dans Maude, les programmes sont des théories de réécriture et les calculs concurrents représentent des déductions dans la logique de réécriture. La finalité du projet Maude est de supporter des spécifications formelles exécutables et d'élargir le spectre de l'utilisation de la programmation déclarative. En effet, Maude permet de spécifier et de vérifier des systèmes de haute qualité dans divers secteurs tels que : le Génie Logiciel, les Réseaux de communication, l'Informatique Répartie...etc.
L'expressivité et la puissance sont ses atouts majeurs. En effet, dans Maude il est possible d'exprimer naturellement un très grand nombre d'applications partant des systèmes déterministes jusqu'aux systèmes concurrents. Il est considéré comme du cadre sémantique, non seulement pour les applications mais des formalismes entiers peuvent y être naturellement exprimés. En plus de produire des spécifications exécutables et d'assurer le prototypage des systèmes la puissance de Maude revient principalement à la collection d'outils formels qu'il englobe. Ces outils supportent différentes formes de raisonnement logique dont : un modèle pour vérifier les propriétés temporelles linéaires de la logique (LTL) «le modèle Checker», un prouveur de Théorème ITP, un Analyseur de terminaison, un Analyseur de Cohérence...etc.
Dans cette partie nous présentons le système Maude de manière générale à travers sa structure et son mécanisme de vérification.

4.2.1 Structure

Nous décrivons ici brièvement la structure d'une spécification Maude. Pour une description plus approfondie le manuel complet est disponible dans [CDE+00]. En Maude, les spécifications sont structurées en modules de tailles relativement petites pour faciliter la compréhension des gros systèmes, augmenter la réutilisabilité et localiser les effets des modifications du système. Maude définit trois types de modules :
Module fonctionnel : y sont définis les types de données et les opérations associées au moyen de théories équationnelles (signature et équations entre termes). Un tel module est déclaré avec la syntaxe *fmod* Nom-du-module... *endfm*.
Module système : permet de définir le comportement dynamique d'un système en dotant le module fonctionnel de règles de réécriture. Un tel module est déclaré avec la syntaxe *mod* Nom-du-module ... *endm*.
Module orienté-objet : offre une syntaxe orientée objet plus adaptée à la définition de

systèmes concurrents. Ces modules peuvent se réduire entièrement à des modules système. Ils sont déclarés avec la syntaxe *omod* Nom-du-module ...*endom*.

Exemple 4.2. *(Module fonctionnel)*
Ce module fonctionnel définit la structure syntaxique (place, marquage) du réseau de Petri de l'exemple 4.1.
fmod PN-SIGNATURE is
sorts PLACE MARKING .
ops $p_1 p_2 p_3$ *:->Place.*
opnill :->Place.
op_: Place->Marking.
op_._: MarkingMarking->Marking.
$vars XYZ : Marking.$
$eq X.nill = X.$
$eq X.Y = Y.X.$
$eq X.(Y.Z) = (X.Y).Z.$
endfm

Exemple 4.3. *(Module système)*
Nous rajoutons des règles de réécriture associées aux transitions du réseau de Petri pour décrire son comportement.
mod PETRINET is
Extending PN-SIGNATURE .
rl $p_1 \Rightarrow p_2.p_2$ *.*
rl $p_1.p_2 \Rightarrow p_1.p_3$ *.*
endm

Le module PN−SIGNATURE de l'exemple 4.2 est importé par le module système PETRINET.

4.2.2 Mécanisme de vérification

Les éléments principaux de Maude sont les équations et les règles. Quoiqu'utilisées à des fins différentes, elles s'utilisent de la même façon : les éléments du membre gauche sont remplacés par les éléments du membre droit. Ainsi, les ensembles d'objets évoluent en fonction des règles de réécriture. Celles-ci décrivent les interactions possibles entre les objets. Les modules Maude sont exécutables et leur exécution fait appel essentiellement à trois types de commandes :

— La commande ***reduce*** permet la simplification de termes en utilisant les équations d'un module jusqu'à l'obtention d'une forme normale, c'est-à-dire un terme qui ne

peut plus être simplifié.

— la commande ***rewrite*** (rew) permet la réécriture d'un terme en utilisant les règles de réécriture. Elle consiste à effectuer, à partir d'un état initial, une à plusieurs séquences de réécriture et ce jusqu'à ce que plus aucune réécriture ne soit possible.

— La commande ***search*** permet d'explorer toutes les séquences possibles de réécriture à partir d'un état initial donné. Elle examine chaque séquence de réécriture en vue de vérifier la satisfaction d'une certaine formule. La recherche s'achève lorsque toutes les séquences possibles ont été appliquées ou lorsqu'on a trouvé une configuration qui satisfait la formule. La vérification du modèle ne se termine pas si l'espace d'états atteignables est infini et que la configuration recherchée n'est pas accessible à partir de l'état initial.
Le mécanisme de recherche repose sur l'application des mécanismes de simplification et de réécriture. La stratégie de Maude impose que toutes les simplifications équationnelles soient appliquées avant chaque application des règles de réécriture.

4.3 Théorie des Catégories

Les catégories[EM45] peuvent être considérées, dans le domaine de l'informatique, comme une généralisation des systèmes de transitions dont les états sont des objets et les transitions sont des arcs. Ces derniers sont dotés d'une opération de composition partielle, associative qui admet des identités. Cette opération correspond intuitivement à la composition des calculs alors que les identités représentent des composants inactifs dans le système à spécifier. Dans les catégories au sens restreint, les objets sont des états et les arcs des calculs.
L'utilisation de la théorie des catégories est justifiée par l'importance des liens entre éléments d'un système ou programme par rapport aux détails d'implémentation d'un objet particulier. L'accent est mis sur les relations et connexions de l'objet à son environnement. Cette abstraction facilite l'analyse et la vérification de propriétés.

4.3.1 Notions fondamentales

4.3.1.1 Catégorie

De manière formelle une catégorie est donnée par la définition suivante :

Définition 4.3. *une catégorie C est la donnée :*

— *D'un ensemble d'objets O_C ;*
— *D'un ensemble de morphismes entre ces objets représentés par des flèches/arcs A_C ; notés (f, g, ...) et munis de la structure suivante :*
Chaque arc a un domaine de définition dom(a)constitué de l'ensemble des objets

source de l'arc et un co-domaine cod(a) constitué de l'ensemble des objets destination de celui-ci. On écrit : f : a → b

- *D'une opération de composition notée ∘ tel que pour tout couple de morphismes (f : A→B, g : B→C), f∘g : A→C est un élément de l'ensemble des morphismes qui vérifie :*
  - *L'existence de l'identité. Pour tout objet O, il existe un morphisme Id_O : O→O.*
  - *L'associativité de la composition. Pour tous les morphismes f : A→B, g : B→C et h : C→D, f∘(g∘h)= (f∘g)∘h.*

Exemple 4.4. *Nous citons quelques exemples simples de catégories.*

- *La catégorie Vide*
- *La catégorie Identité*
- *La catégorie des ensembles Set : les objets de Set sont des ensembles et les morphismes sont des fonctions munies de l'opération de composition et de la fonction identité.*
- *La catégorie monoïde. Une catégorie C est dite catégorie monoïde si elle ne contient qu'un seul objet.*

4.3.1.2 Foncteur

Etant donné que les catégories sont elles-mêmes des entités mathématiques, les morphismes entre de telles structures peuvent jouer un rôle important. En considérant les catégories comme objets, un morphisme entre deux catégories, appelé foncteur, est alors une transformation préservant les objets et les morphismes entre ces objets. Soient C et D deux catégories. un foncteur $\mathcal{F}$ de C vers D, noté $\mathcal{F}$:C→D est une application qui à tout objet de C associe un objet de D et à tout arc f : A→B de C associe un arc de $\mathcal{F}(f)$: $\mathcal{F}$(A)→ $\mathcal{F}$(B) de D, tels que :

- Préservation de l'identité : ∀ A objet de C, $\mathcal{F}(id_A)= id_{\mathcal{F}(A)}$
- Préservation de la composition : ∀ f : A→B, g : B→C, $\mathcal{F}$(g∘f)=$\mathcal{F}$(g)∘ $\mathcal{F}(f)$

Exemple 4.5. *nous citons des exemples de foncteurs connus :*

- *Le foncteur Identité : id_C : C→C, exprime l'identité sur les objets et sur les arcs.*
- *Le foncteur inclusion : pour chaque catégorie C, il existe une sous catégorie C_0 de C tel que I : C→C_0 est un foncteur.*

De manière générale, à ce stade, nous pouvons considérer la catégorie Cat comme catégorie de toutes les catégories où les objets sont des catégories et les morphismes des foncteurs entre ces catégories.

4.4 Modèle sémantique d'une théorie de réécriture

En plus de la sémantique opérationnelle associée à une théorie de réécriture et inférée par le système de déduction, la logique de réécriture définit aussi le modèle sémantique associé à cette théorie.
Les catégories sont le modèle mathématique associé aux théories de réécriture, celui ci peut supporter les termes générés par la spécification algébrique (Σ, E) et les preuves générées par la déduction. Les objets de cette catégorie sont les Σ-termes et les morphismes représentent les preuves entre eux. L'utilisation des catégories est justifiée par l'importance des morphismes qui fournissent une abstraction des détails d'implémentation des objets en se focalisant uniquement sur les relations qui existent entre ces objets. De manière plus formelle, ce modèle est défini comme suit :
Le modèle associé à une théorie de réécriture $\mathcal{R}$= (Σ, E, L, R) est une catégorie $T_{\mathcal{R}}(X)$. Les objets de $T_{\mathcal{R}}(X)$ sont des classes d'équivalence de Σ-termes [t]$\in T_{\Sigma,E}(X)$ modulo les équations E et les morphismes sont des classes d'équivalence de termes de preuves. Les arcs identités correspondent naturellement aux termes de preuve générés par la règle de réflexivité et les arcs compositions sont générés par la règle de transitivité. Les termes de preuves sont aussi dotés d'une structure algébrique identique à celle définie sur les objets de $T_{\mathcal{R}}(X)$. Cette structure est obtenue grâce à la règle de congruence de la logique de réécriture. Intuitivement, cette structure est l'une des interprétations possibles de la réécriture dans une théorie $\mathcal{R}$. La catégorie $T_{\mathcal{R}}(X)$ est juste une parmi plusieurs modèles associés à une théorie $\mathcal{R}$ appelés $\mathcal{R}$-systèmes qui sont définis dans [Mes92]

4.5 Conclusion

Dans ce chapitre, nous avons présenté les différents formalismes servant de support à nos contributions. Nous avons d'abord présenté les notions fondamentales relatives à la logique de réécriture : la théorie de réécriture et les termes de preuves. Cette logique présente un outil de raisonnement correct sur ces systèmes ayant des états et évoluant en termes de transitions. C'est alors un moyen rigoureux pour définir de manière formelle les systèmes concurrents. Les différents concepts de la logique de réécriture sont implémentés à travers l'environnement MAUDE que nous avons également présenté à travers sa structure et son mécanisme de vérification. Enfin, nous avons présenté la sémantique catégorielle d'une théorie de réécriture à travers les notions de la théorie des catégories.

Chapitre 5

Une approche basée K pour la description des services Web

5.1 Introduction

WDSL joue un rôle prépondérant dans le cycle de développement des services Web ; en plus de donner une description de l'interface d'un service, il doit garantir que cette description assure son interopérabilité avec d'autres services vu la vocation d'universalité qui leur est associée. Bien que la description abstraite soit suffisante pour les services Web dits simples, où, à ce niveau l'interopérabilité est large compte tenu de l'uniformisation de la sémantique offerte, ce n'est pas le cas pour des composants plus complexes. En effet, la description WSDL, dans ce cas, devient alors inappropriée car ces services nécessitent des modèles d'échange plus complexes. Cette problématique impose donc la recherche d'un moyen qui soit efficace et rigoureux pour associer une sémantique formelle à ce type de langages.
Dans ce chapitre, nous proposons une solution pour pallier à ce problème par la mise en place d'une méthode idoine pour la description des services Web basée sur la formalisation des concepts du standard WSDL. L'issue de cette formalisation est la production d'un langage de spécification formel de services Web doté d'une sémantique plus flexible et exploitable. La sémantique formelle définie issue de cette nouvelle approche sert à résorber les lacunes dont souffre WSDL et peut profiter de l'environnement Maude comme cadre formel et exécutable.
L'objectif principal de notre travail est donc, de fournir une spécification formelle de services Web pouvant être exploitée et formellement vérifiée. Pour ce faire, nous avons opté pour le cadre K comme support sémantique formel. Notre choix s'est porté sur cet outil pour sa simplicité d'utilisation et sa facilité d'extension. En effet, K tire cette simplicité des langages de programmation puisqu'il a été initialement conçu pour eux. Il permet entre autre, d'étendre et d'enrichir tout langage d'une sémantique formelle définie en terme de règles de réécriture. Cette sémantique permet la transformation automatique et en

transparence des spécifications dérivées en des spécifications Maude ce qui permet leur exploitation directe et leur vérification formelle.
Ce chapitre est structuré de la manière suivante :
Nous présentons dans un premier temps, le cadre sémantique K à travers sa syntaxe, sa sémantique et son outil K-Maude puis nous introduisons un exemple pour motiver notre travail et mettre en évidence les limites du langage de description standard des services Web WSDL. Nous exposons par la suite le principe de notre approche. Dans la section 4, nous présentons le langage KWSDL implanté sous K pour spécifier formellement les services Web. Nous donnons sa syntaxe et sa sémantique et concluons en montrant l'apport de ce dernier en terme d'interopérabilité sémantique.

5.2 Le cadre sémantique K

Introduit par le premier auteur en 2003 pour l'enseignement des langages de programmation et continuellement raffiné et développé depuis, le cadre K [R10] est un cadre sémantique formel basé sur la réécriture initialement conçu pour associer une sémantique formelle aux langages de programmation. Plusieurs langages ont été définis dans K tels que Java [BR15], C [ER12], Scheme [MR07]...etc. Son principal atout est qu'il regroupe les forces des cadres existants (expressivité, modularité, concurrence et simplicité) tout en évitant leurs faiblesses. Pour donner la sémantique aux constructions d'un langage, le cadre K repose sur des structures de calcul, des configurations et des règles. Ce sont ces structures informatiques ou plus simplement appelées calculs qui ont inspiré le nom "K" du cadre. Les calculs sont généralement utilisés pour gérer le fragment séquentiel du langage défini et les stratégies d'évaluation des différentes constructions syntaxiques. Dans cette section nous donnons un aperçu du cadre sémantique formel K par la présentation des modules syntaxique et sémantique (configuration + règles k) qui constituent la base de tout langage défini dans K.
Toutes les fonctionnalités d'un langage définit dans K sont regroupées dans des modules. Un module K prend la forme générale :
$Module{<}NAME{>}...Endmodule$ où $<NAME>$ est l'identifiant unique du langage écrit en lettres majuscules.
La définition K d'un langage doit contenir au moins un module mais il est souhaitable de séparer la syntaxe de la sémantique afin de minimiser les ambiguïtés. Le module syntaxique est ainsi étendu du suffixe SYNTAX après son nom :$Module{<}NAME{>}-SYNTAX...Endmodule$. Le module sémantique peut importer la syntaxe en insérant à son code le mot-clé imports $<NAME>-SYNTAX$.

5.2.1 Syntaxe K

```
module EXP-SYNTAX

  //@ Arithmetics Syntax
  syntax Exp ::= #Int
               | Exp "+" Exp [strict] //addition
               | Exp "*" Exp [strict] //multiplication
               | Exp "/" Exp [strict] //division
               | Exp "?" Exp ":" Exp [strict(1)]
               | Exp ";" Exp [seqstrict]

  //@ Input / Output Syntax

  syntax Exp ::= "read"
               | "print" Exp [strict]


  //@ Concurrency features
  syntax Exp ::= "spawn" Exp
               | "rendezvous" Exp [strict]
end module

module EXP
  imports EXP-SYNTAX
  syntax KResult ::= #Int
  configuration
    ⟨k color="green" multiplicity="*"⟩ $PGM:K ⟨/k⟩
    ⟨streams⟩
      ⟨in color="magenta" stream="stdin"⟩ .List ⟨/in⟩
      ⟨out color="Fuchsia" stream="stdout"⟩ .List ⟨/out⟩
    ⟨/streams⟩

 //@ Arithmetics Semantics

  rule I1:#Int + I2:#Int  =>  I1 +Int I2
  rule I1:#Int * I2:#Int  =>  I1 *Int I2

  rule I1:#Int / I2:#Int  =>  I1 /Int I2
                              when I2 =/=Bool 0

  rule 0 ? _ : E:Exp  =>  E
  rule I:#Int ? E:Exp : _  =>  E  when I =/=Bool 0

  rule _:#Int ; I2:#Int  =>  I2


  //@ Input / Output Semantics

  rule ⟨k⟩ read  =>  I:#Int ⋯⟨/k⟩
       ⟨in⟩ ListItem(I)  =>  . ⋯⟨/in⟩


  rule ⟨k⟩ print I:#Int  =>  I ⋯⟨/k⟩
       ⟨out⟩⋯ .  =>  ListItem(I) ⟨/out⟩


  //@ Concurrency Semantics

  rule ⟨k⟩ spawn E  =>  0 ⋯⟨/k⟩
       (.  =>  ⟨k⟩ E ⟨/k⟩)


  rule ⟨k⟩ rendezvous I  =>  0 ⋯⟨/k⟩
       ⟨k⟩ rendezvous I  =>  0 ⋯⟨/k⟩

end module
```

MODULE EXP-SYNTAX

Arithmetics Syntax

SYNTAX *Exp* ::= *#Int*
| *Exp* + *Exp* [strict]
| *Exp* * *Exp* [strict]
| *Exp* / *Exp* [strict]
| *Exp* ? *Exp* : *Exp* [strict(1)]
| *Exp* ; *Exp* [seqstrict]

Input / Output Syntax

SYNTAX *Exp* ::= read
| print *Exp* [strict]

Concurrency features

SYNTAX *Exp* ::= spawn *Exp*
| rendezvous *Exp* [strict]

END MODULE

MODULE EXP
IMPORTS EXP-SYNTAX
SYNTAX *KResult* ::= *#Int*
CONFIGURATION:

Arithmetics Semantics

RULE $I_1 + I_2 \Rightarrow I_1 +_{Int} I_2$

RULE $I_1 * I_2 \Rightarrow I_1 *_{Int} I_2$

RULE $I_1 / I_2 \Rightarrow I_1 \div_{Int} I_2$ when $I_2 \neq_{Bool} 0$

RULE $0 ? \_ : E \Rightarrow E$

RULE $I ? E : \_ \Rightarrow E$ when $I \neq_{Bool} 0$

RULE $I_1 ; I_2 \Rightarrow I_2$

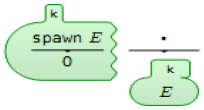
Input / Output Semantics

RULE

RULE

Concurrency Semantics

RULE



RULE

END MODULE

FIGURE 5.1 – Définition K du langage EXP

K fournit une notation pratique pour définir de manière modulaire la syntaxe et la sémantique d'un langage. La syntaxe d'un langage dans K est donnée en utilisant la notation BNF (Backus-Naur-Form) [Knu64] où les terminaux sont entre guillemets et les non-terminaux commencent par des lettres majuscules. La Figure 5.1 présente la définition K du langage Exp. La première partie (lignes 1-20) reflète sa syntaxe. Par exemple, le code suivant définit une catégorie syntaxique Exp qui contient un type entier et des opérations arithmétiques.

```
//@ Arithmetics Syntax
syntax Exp ::= #Int
             | Exp "+" Exp [strict] //addition
             | Exp "*" Exp [strict] //multiplication
             | Exp "/" Exp [strict] //division
             | Exp "?" Exp ":" Exp [strict(1)]
             | Exp ";" Exp [seqstrict]
```

K fournit des types de données prédéfinis pour les différencier de ceux définis par l'utilisateur ; ils sont préfixés par le symbole "#". Outre le type des entiers, on retrouve les booléens "$\#Bool$", les flottants "$\#float$", les chaines de caractères "$\#String$" et les identifiants "$\#Id$". A ces déclarations syntaxiques sont associés des attributs servant à l'évaluation de cette dernière en fonction du calcul employé.

5.2.2 Sémantique K

La spécification de la sémantique dans K consiste en trois parties : les stratégies d'évaluation, la configuration et les règles K.
Les stratégies d'évaluation traduisent les liens qui existent entre la syntaxe d'un langage et sa sémantique. Elles spécifient l'ordre dans lequel les arguments doivent être évalués. Par exemple, les deux arguments de l'opérateur d'addition " + " doivent être évalués avant le calcul de leur somme, tandis que pour l'opérateur conditionnel "_ ?_ :_", seul le premier argument doit être évalué.
Strict et *Seqstrict* sont deux attributs d'évaluation qui sont paramétrés par des nombres indiquant les positions sur lesquelles les arguments sont stricts (1 étant la position de gauche). Si aucun paramètre n'est fourni, toutes les positions sont considérées comme strictes. La différence entre les attributs strict et seqstrict est que ce dernier inflige l'évaluation des arguments dans l'ordre donné. par exemple, strict (ligne 8) indique que le premier argument de l'expression conditionnelle doit être évalué avant de donner la sémantique. Tandis que, seqstrict (ligne 9) indique que les deux sous-expressions doivent être évaluées, mais que l'expression de gauche doit être évaluée en premier.

Configuration

Une configuration dans K représente l'état du système en cours d'exécution. La configuration est donnée sous forme de cellules imbriquées et étiquetées contenant des calculs basés sur les structures de données. Ces cellules sont représentées par des balises semblables à celles du langage XML.

K suppose que ces configurations soient définies avant que les règles sémantiques ne soient données. Le mécanisme d'abstraction d'une configuration permet aux définitions d'être à la fois compactes et modulaires ne nécessitant souvent pas de modifications aux règles existantes lorsque la configuration change.

La Figure 5.2 décrit la structure générale d'une configuration EXP (en Latex). La cellule k contient le programme en cours d'exécution (noté par la variable $\$PGM$), la cellule streams contient les cellules in et out qui modélisent les flux d'entrée/sortie du programme.

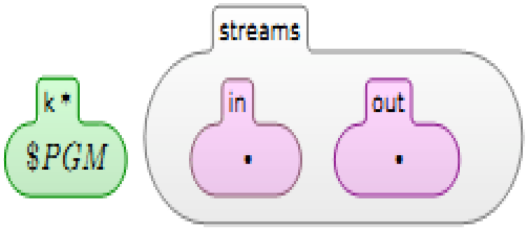


FIGURE 5.2 – Configuration initiale du langage EXP

K permet plusieurs manipulations sur ces cellules, par exemple :

— color : pour colorier les cellules et permettre une meilleure visibilité.
— stream : pour la spécification des flux d'entrée et de sortie.
— multiplicity : pour dupliquer les cellules (allant de 0 à plusieurs (*) copies).

Une cellule peut contenir des calculs dans des listes et des ensembles, respectivement :(List, Bag, Set, et Map) qui sont déclarés par .List, .Bag, .Set, et .Map. Le point « . » signifie dans K une liste ou un ensemble vide. Les éléments d'une liste ou d'un ensemble sont séparés par un espace, par exemple : BagItem(1) BagItem(2).

Les Règles K

Les règles K se distinguent en spécifiant uniquement ce qui est nécessaire à partir d'une configuration en identifiant clairement les changements à opérer ; Ainsi, elles sont plus concises, plus modulaires et plus concurrentes que les règles de réécriture régulières. Plus précisément, les règles de réécriture K généralisent les règles de réécritures classiques en rendant explicites les parties du terme qui sont en lecture ou en écriture seule, ou même ignorées. Ce qui fait de K le cadre adéquat pour la définition des langages concurrents ou des calculs. Il existe deux types de règles dans K : les règles structurelles et les règles de calcul.

— **Règles structurelles K**
Les règles structurelles décomposent et éventuellement poussent les tâches qui sont prêtes à être traitées par les règles de calcul. Elles ne comptent pas comme étape de calcul et ne sont pas observables.
Les règles structurelles K sont semblables aux équations dans la logique de réécriture : elles précisent le comportement déterministe par la définition des classes d'états sur lesquelles les transitions se produisent. Cependant, à la différence des règles structurelles, les équations sont appliquées dans les deux sens pour ainsi définir une classe d'équivalence d'états.

— **Règles de calcul K**
Les règles de calcul déterminent le mécanisme de réécriture, elles spécifient comment une configuration évolue et éventuellement modifie l'état d'un programme. Une règle K de calcul décrit comment un terme ou un sous-terme peut se transformer en un autre, d'une manière similaire à une règle de réécriture : chaque terme qui est équivalant au coté gauche sera remplacé par le terme du coté droit. Une règle sémantique est introduite par le mot-clé rule et le symbole " ⇒ " délimitant respectivement la position du terme gauche (remplacé) et du terme droit (remplaçant). Ce symbole est représenté en LATEX par une ligne horizontale. Par exemple, la règle print (ligne 54) effectue deux changements sur la configuration :
L'expression **print** est remplacée par son paramètre de type entier et une liste contenant cet entier remplace la liste vide dans la cellule du flux de sortie **out**.
Afin de mettre en pratique toutes ces définitions K utilise l'outil K-Maude. Dans ce qui suit nous donnons une vue d'ensemble sur l'architecture de cet outil et de ces composantes principales.

5.2.3 L'outil K-Maude

K-Maude est un outil intégré au sommet du système Maude. La Figure 5.3 montre son architecture de base. Les flèches grises représentent les traducteurs de modules intégrés à l'outil.

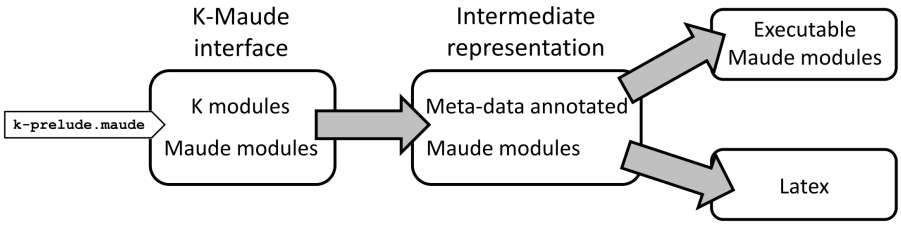


FIGURE 5.3 – Architecture K-Maude

L'outil K-Maude se compose d'un noyau et d'une interface qui sert d'intermédiaire entre l'outil et l'utilisateur. Les modules K sont convertis en transparence, en représentations

intermédiaires Maude ; ces représentations peuvent être exécutées et analysées sous Maude ou transformées en documentation officielle de la sémantique du langage en Latex. Noyau La syntaxe principale de l'outil se trouve dans k-prelude.maude. Ce fichier représente le noyau de K-Maude, il fournit les briques de base pour la construction des calculs, des configurations...etc. Il est très important que ce fichier soit inclus dans toutes les définitions K avec ou sans l'utilisation de l'outil K-Maude.

Interface K-Maude l'interface est ce que l'utilisateur voit habituellement : en plus des modules Maude, K-Maude inclut des fichiers K (portant l'extension.kmaude ou .k) contenant des modules écrit en K.

Représentation intermédiaire un premier composant traduit les modules K en modules Maude. Les modules Maude résultants servent comme représentation intermédiaire des définitions K mais ne sont pas destinés à être exécutés. Par ailleurs, ils sont fortement annotés avec des méta-données. Cette représentation intermédiaire peut être traduite en plusieurs versions :

De K à Maude : les Modules issus de la première transformation subissent une deuxième transformation afin qu'ils soient exécutables et analysables sous Maude.
De K au Latex : bien que l'interface textuelle K-Maude lui-même soit un bon moyen de communication, K a été principalement introduit par une notation très intuitive et visuelle plus appropriée pour la conception des langages de programmation. Par conséquent, pour faciliter l'inclusion des définitions K dans les documents de recherche et des présentations K-Maude permet des annotations (comme des attributs spéciaux) spécifiant comment différentes constructions doivent être représentées dans LATEX, et fournit un outil (écrit en Maude) qui génère automatiquement un document LATEX de la définition intermédiaire "*.maude*" résultante.
Dans cette partie nous avons donné un bref aperçu de l'outil K-Maude, plus de détails techniques concernant son exploitation sont exposées dans [SR10].

5.3 Exemple de motivation

Afin d'illustrer la faisabilité de notre approche, nous donnons en exemple, un service exposant en ligne des formations proposées par l'université de Constantine 2. Nous nommons ce service «UC2.dz» (Figure 5.4), sa description WSDL est disponible sur le Web ce qui permet son invocation directe.

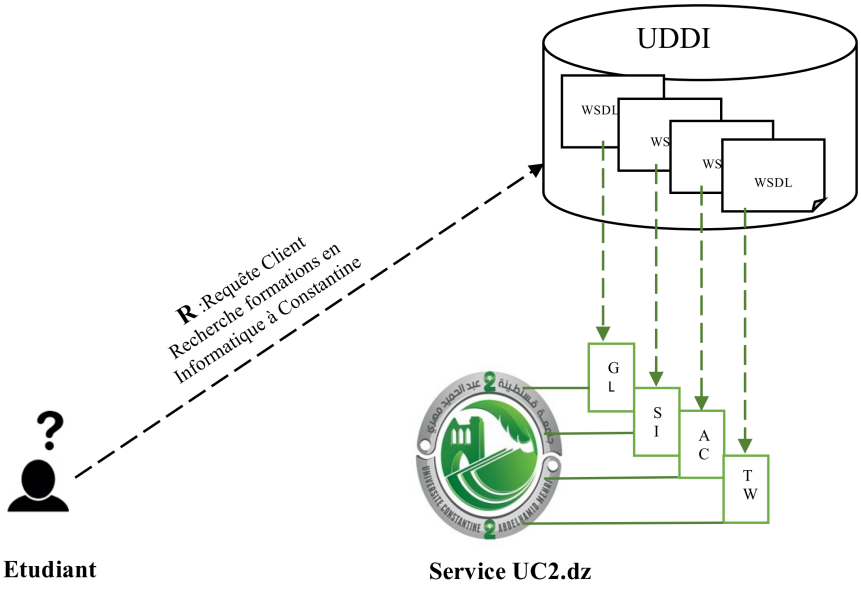


FIGURE 5.4 – Exemple Inscription Etudiant

Scénario standard

Les formations étant destinées à des étudiants, nous pouvons déduire le scénario suivant : l'étudiant formule une requête demandant la liste des différentes formations en informatique offertes par l'université de Constantine 2, une liste de descriptions WSDL de services lui est renvoyée, l'étudiant choisit la formation qui l'intéresse et invoque le service adéquat.

Problématique soulevée

Nous notons, que la requête de l'étudiant n'est pas interprétée de manière exhaustive. En effet, l'étudiant peut à travers ce service, uniquement, obtenir des informations sur les formations sans pouvoir s'y inscrire. Plus précisément, cela peut s'expliquer par :
Syntaxiquement, le terme «recherche» est différent du terme «inscription», cependant, les deux concepts sont sémantiquement liés, le premier étant une condition préalable de la seconde. L'étudiant n'est donc pas informé de tous les services potentiellement pertinents. Un tel résultat est la conséquence du manque de sémantique décelé dans les descriptions WSDL et ce genre de subtilité ne peut être contourné que lorsque le service est doté d'une description sémantique plus élaborée.
Dans ce même contexte, notre approche à travers l'utilisation du cadre sémantique K vise à remédier à ces problèmes en proposant une description alternative aux services Web dotée d'une sémantique formelle et opérationnelle pouvant être analysée dans un environnement formel adapté. Le principe de cette approche est exposé en détails dans la section suivante.

5.4 Principe de notre approche

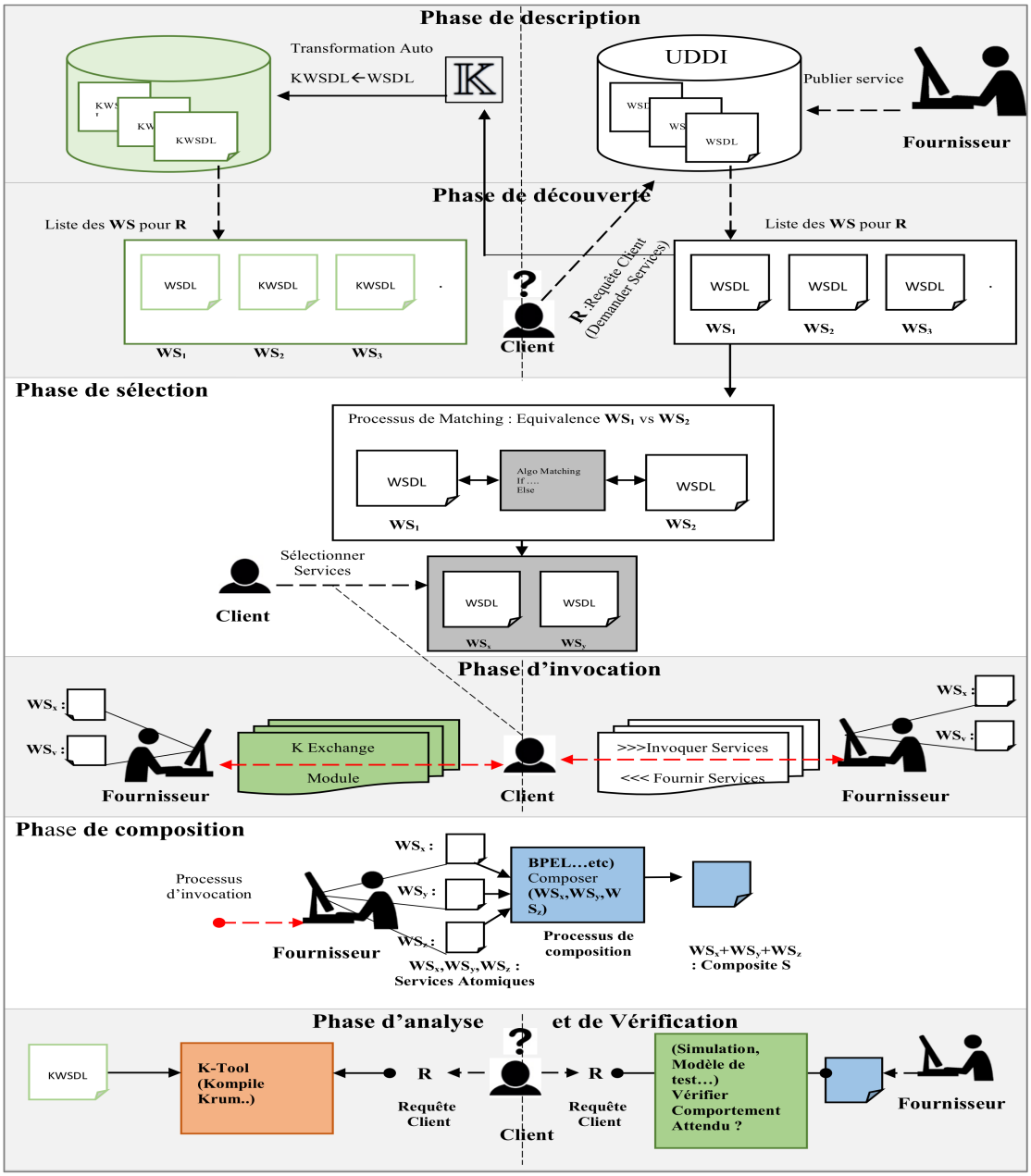


FIGURE 5.5 – Description de services Web : K vs Techniques standards

Ce schéma (Figure 5.5) fait référence au schéma représentant le cycle de vie d'un service Web à travers ses phases de développement présenté dans la partie état de l'art. Ce dernier fera l'objet de référence standard, au fil de la présentation de nos contributions,

afin de pouvoir mesurer et évaluer l'impact des solutions apportées à chaque phase. Dans notre contribution, nous intervenons de manière concrète au niveau des phases de description, d'invocation et d'analyse.

Phase de description :
Notre étude se porte sur le langage standard de description des services Web WSDL afin de s'en inspirer et concevoir un langage de réécriture équivalent.
La définition des descriptions KWSDL s'effectue à travers deux étapes distinctes, la première étant de formaliser et transformer la syntaxe extraite de la description WSDL de façon à ce qu'une sémantique puisse lui être associée. La seconde étape consiste à y inférer cette sémantique (règles de réécriture) au moyen des techniques K.

Phase de découverte : La phase de découverte est implicitement affectée par le changement de la nature des descriptions de WSDL à KWSDL (mapping automatique).

Phase d'invocation : En KWSDL, le protocole de communication entre services n'est plus nécessaire vu que sa sémantique opérationnelle permet de spécifier le processus d'interaction entre services à travers le module exchange.

Phase d'analyse : Cette étape est automatiquement accessible grâce aux outils de vérification formelle intégrés dans l'environnement Maude. Les spécifications KWSDL sont facilement analysables.

5.5 Langage KWSDL

Le langage KWSDL comparé au standard WSDL est bien plus qu'un langage d'interface. Il permet de spécifier, non seulement, de manière simple un service Web mais aussi, de donner un sens à cette spécification. En outre, son rôle ne se réduit pas qu'à cela, il admet l'aspect communicatif relatif aux services Web en spécifiant, entre autres, les échanges effectués entre eux. Enfin, il demeure extensible et tous les modèles qui en sont dérivés sont automatiquement convertibles et analysables sous Maude.
Comme tout langage, le langage KWSDL possède une syntaxe et une sémantique. Il intègre également une configuration qui est une notion propre à K. Nous présentons dans ce qui suit ce langage à travers ces composantes :

5.5.1 Grammaire KWSDL

La syntaxe du langage KWSDL est intégrée sous forme d'un module faisant office d'un méta-modèle formel prenant en considération tous les aspects du standard WSDL.

Description WSDL	Grammaire KWSDL
	Module KWSDL-SYNTAX
<definitions name="ServiceName" targetNamespace="uri" xmlns="uri" xmlns:soap="uri xmlns:tns="uri" xmlns:xsd="uri">	Syntax **KWSDL**::"KWSDL" "Service Name" NameService ":" "{" DescriptionPart "}" Syntax DescriptionPart ::= "**definitions**" "Service Name" "=" NameService ":" "{"Def"}" >DescriptionPart DescriptionPart [left] Syntax Def ::= "TargeNameSpace" NameSpace "=" Serviceurl |"DefaultNameSpace" DNameSpace "="DServiceuri |"DefaultUsedProtocol" DProtocol "=" DProtocoluri |"DefaultFormat" DFormat "=" DAFormaturi
<message name="nmtoken"> <part name="nmtoken" type="qname"/> </message>	|"**message**" MsgN TypeMsg ":" Msg Syntax TypeMsg ::= "request" |"response"|"notification"
<portType name="nmtoken_PortType"> <operation name="nmtoken"><input message="qname"/><output message="qname"/></operation></portType>	|"**operation**" Type OpN Msg Syntax Type ::= "Input" |"Output"
<binding name="nmtoken" type="qname"> <operation name="nmtoken"><input>...</input><output> ...</output></operation></binding>	|"**binding**" Binding BindingPort Protocol Style "=" Body Syntax DProtocol ::="SOAP" |"SMPT"
<service name="nmtoken"> <port binding="qname" name="nmtoken"> <soap:addresslocation="url" /></port></service>	|"**service**" Service Binding BindingPort Location |"port" PortId "=" Port |"**exchange**" ServiceName "." PortName "to" BindingPortName"." Msg > Def Def [left] ....
</definitions>	

TABLE 5.1 – Définition de la grammaire KWSDL

KWSDL n'est pas destiné à spécifier l'interface d'un service. Notre inspiration de WSDL est strictement relative à son aspect standard. Nous jugeons que tous les concepts qui le constituent sont essentiels à la représentation d'un service Web. Nous faisons donc à travers le Tableau 5.1 une correspondance entre les grammaires des deux langages afin de mettre en évidence les changements opérés.
La description WSDL représentée dans le tableau a été réduite afin de mettre en évidence essentiellement les éléments qui font partie du mapping. Dans WSDL la racine du document est représentée par le concept definitions, les fils directs de la racine sont dans l'ordre :
Les types, les messages, les PortType (+opérations), les bindings et le service(+ports).
Cette structuration a été respectée dans KWSDL du moment que l'on retrouve tous ces concepts à quelques exceptions près. En effet, KWSDL exclut certaines déclarations comme par exemple celle des types de données puisqu'ils sont, par défaut, intégrés à K (built-in).
Une déclaration simple du concept message est donnée, par exemple, par :
"message" MsgN TypeMsg " :" Msg
Syntax TypeMsg ::= "request" |"response"|"notification"
Un message est spécifié par son nom, son type (requête, réponse, notification) ainsi que les données qu'il véhicule.
De la même manière sont déclarées les opérations du service, ses ports et connecteurs.
left
est un attribut souvent associé à des déclarations syntaxiques portant une information sémantique décrivant la stratégie d'évaluation de la construction correspondante. Dans notre cas par exemple :
>Def Def [left] veut dire que la définition qui est à gauche doit être évaluée en premier. Il existe d'autres attributs dédiés à l'évaluation des déclarations syntaxiques tels que Strict, SeqStrict...etc.
Comme nous l'avons mentionné au préalable, K fournit des modules pour regrouper les fonctionnalités du langage. Bien qu'il n y ait aucune exigence rigide ou même des directives sur la façon de regrouper ces fonctionnalités. La syntaxe du langage est toujours séparée par un module distinct afin de pouvoir l'analyser avant même de lui associer une sémantique.

```
tc@box:~/workspace$
tc@box:~/workspace$ kompile KWSDL-SYNTAX.k
tc@box:~/workspace$ _
```

FIGURE 5.6 – Compilation de la syntaxe KWSDL

La Figure 5.6 illustre la compilation de la grammaire KWSDL sous K, l'absence de renvoi de message d'erreur par le compilateur après compilation confirme le succès de celle-ci.
Nous constatons qu'à ce niveau prématuré de la définition du langage, l'intérêt de l'utilisation de K est déjà évident. En effet, la possibilité d'isoler et de séparer la syntaxe

de sa sémantique minimise les ambiguïtés d'analyse du programme. Les erreurs peuvent être identifiées plus tôt dans le processus de développement et sont, par conséquent, faciles à résorber. Par ailleurs, l'absence de sémantique formelle décelée dans le langage WSDL est à l'origine de conflits gênant l'interopérabilité entre services. Ces lacunes sont liées au niveau d'expressivité relativement bas de la description syntaxique. Cette description ne caractérise pas la sémantique de la fonctionnalité réalisée par le service. Pour remédier à ce problème, nous proposons dans la section suivante, la sémantique formelle du langage KWSDL qui lève toute ambigüité.

5.5.2 Sémantique de KWSDL

Les langages sémantiques et ontologies tels que OWL, SAWSDL présentés dans la partie état de l'art, ont contribué à l'amélioration des descriptions du standard WSDL en terme de sémantique. Cependant, aucune sémantique opérationnelle n'a encore été définie. Dans notre approche, nous définissons non seulement la structure syntaxique d'un service mais aussi son comportement. La sémantique du langage KWSDL est écrite dans le module global K qui importe le module syntaxique précèdent, ce module comprend la configuration (Tableau 5.2) et les règles structurelles et de calcul K (Tableau 5.3).
Les stratégies d'évaluation servent de lien entre la syntaxe et la sémantique, en spécifiant comment les arguments d'une construction du langage doivent être évaluées. Par exemple, l'attribut left mentionné plus haut.

5.5.2.1 Configuration KWSDL

Dans K, l'état d'un programme/système en cours d'exécution est représenté par une *configuration*.
Les configurations sont structurées sous forme de cellules étiquetées contenant diverses structures de données.
Les balises du Tableau 5.2 décrivent à la fois la configuration initiale et la structure générale d'une configuration pour KWSDL qui doit être incluse dans le module KWSDL. La déclaration d'une configuration est introduite par le mot-clé «configuration» et se compose d'une cellule nommée k censée contenir le programme en cours d'exécution (désignés ici par la variable $\$PGM$ de type K) et un ensemble de cellules qui modélisent tous les concepts représentant l'état d'un service.

```
Configuration<state color="yellow">

<k color="green"> $PGM:KWSDL </k>

<definitionscolor="cyan" >

<portType color="red" multiplicity ="*">

<PortName> "port" </PortName>

<operation>.Map </operation>

<messagecolor="orange">.Map</message>

</portType>

<bindingcolor="Orchid" multiplicity ="*">

<bindingName>"binding"</bindingName><bindingPortName>"bindingPortName"</bindingPortName><protocol>"protocol"
</protocol><style>"style"</style><inputOperations>.Map</inputOperations><outputOperations>.Map</outputOperations>

</binding>

<servicecolor="green" multiplicity ="*"><serviceName>"nameservice"</serviceName>

<bindingName>"binding"</bindingName><bindingPortName>"bindingPortName"

</bindingPortName><location>.Map </location>

</service>

<Exchangecolor="red">.Map </Exchange>

</definitions>

...</state>
```

TABLE 5.2 – Configuration initiale d'une description KWSDL

Dans la configuration initiale KWSDL, la balise <**state**> comprend toutes les balises K définissant l'état global du système. La balise <**k**> contient la description KWSDL, le point à partir duquel le programme entame son exécution. Les balises <**définitions**>, <**message**>, <**portType**>,<**Binding**> et <**service**> représentent respectivement l'ensemble des définitions liées au service (espace de nom, protocole utilisé par défaut, etc), messages (nom, type, message), port types (nom, opérations), connecteurs (nom, nom du type de port, le corps d'encodage).

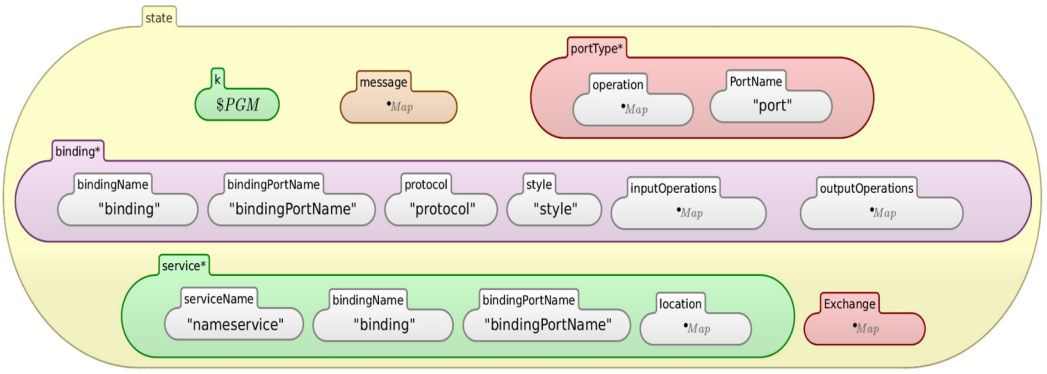


FIGURE 5.7 – Configuration initiale KWSDL en Latex

Les seuls types de contenu cellulaire actuellement autorisés par l'outil K sont les Bag, Set et List. Les éléments de ces structures sont obtenus en injectant des calculs à travers les constructeurs ListItem, BagItem, SetItem. La déclaration de configuration sert de squelette au processus d'abstraction qui permet aux utilisateurs de mentionner seulement les cellules pertinentes dans chaque règle sémantique, le reste du contexte de configuration étant automatiquement déduit.

5.5.2.2 Règles structurelles et opérationnelles pour KWSDL

La définition de la sémantique KWSDL consiste à fournir un ensemble de règles de réécriture qui modélisent le changement d'état du système après avoir été spécifiquement transformé en structures internes. Ces règles précisent l'échange de messages entre les services via les ports et les connecteurs.
Avant de spécifier cette sémantique, K impose un réarrangement structurel afin de préparer l'application des calculs. La sémantique étant donnée en terme de règles de réécriture, la restructuration est donc aussi établie par des règles dites structurelles.
En effet, deux types de règles existent dans la sémantique K : les règles opérationnelles (de calcul), qui comptent comme étapes de calcul et les règles structurelles qui ne comptent pas comme étapes de calcul.
Le rôle des règles structurelles est de réorganiser les termes déclarés de telle sorte que les règles de calcul puissent s'appliquer directement. Dans notre cas, et comme étudié dans la partie état de l'art, la structure d'un fichier WSDL est subdivisée en deux parties distinctes : une partie abstraite et une partie concrète.
La partie abstraite englobe les données utilisées et la partie concrète traduit comment ces données sont utilisées et propose une ou plusieurs réalisations de la partie abstraite. En KWSDL les règles structurelles jouent ce rôle en traduisant le processus d'échange de messages via les ports et connecteurs du service..etc.
Par ailleurs, la sémantique opérationnelle associée à KWSDL est donnée dans le Tableau 5.3.

Module KWSDL

Imports Module KWSDL-SYNTAX

Rule<k>...portX={O}=>X ~>O...</k>

(.=><portType><portName>X</portName><operation>.</operation></portType>)

rule<k>...X~>TP:Type T:TypeMsg:MS:Msg =>X ~>....</k>

<portType><portName>X</portName><operation>Rho:Map (.=> TP T |->MS)</operation>

</portType>

rule<k>...service S~>U:Loc:AS:AdressService =>S~>. ...</k>

<service>...<serviceName>S</serviceName>

<bindingName>B</bindingName><bindingPortName>BP</bindingPortName>

<location>Rho:Map(.=>U|->AS) </location></service>

rule<services>...<service> ....

<serviceName>X</serviceName>

<portType>...Out P :port|->(RequestMsg :String=>.)...</portType>

<serviceName>Y</serviceName>

<Binding>... In B:Binding |->(Request:String=>Resquest+ResquestMsg)...</Binding>

<exchange > Rho:Map</exchange>

When $hasMapping(Rho,X.P)

andBool (Y.B==K Rho:Map (X.P))

andBooL Request==String ""

.....

TABLE 5.3 – Sous ensemble de règles opérationnelles KWSDL

A titre d'exemple, la règle liée au port type définit un portType par l'ensemble des opérations d'entrée/sortie qu'il comprend. Une opération prend la forme «type typeMsg Msg» avec : type : «input ou output», typeMsg :«request ou response» et Msg comprenant le message véhiculé.
L'application de cette règle est traduite par le résultat qui se trouve dans la structure interne décrite par la balise <portType> composée de deux sous-balises :
<portName> pour le nom du port et <operation> pour les différentes opérations d'entrée/sortie effectuées.

5.6 Exemple de spécification KWSDL

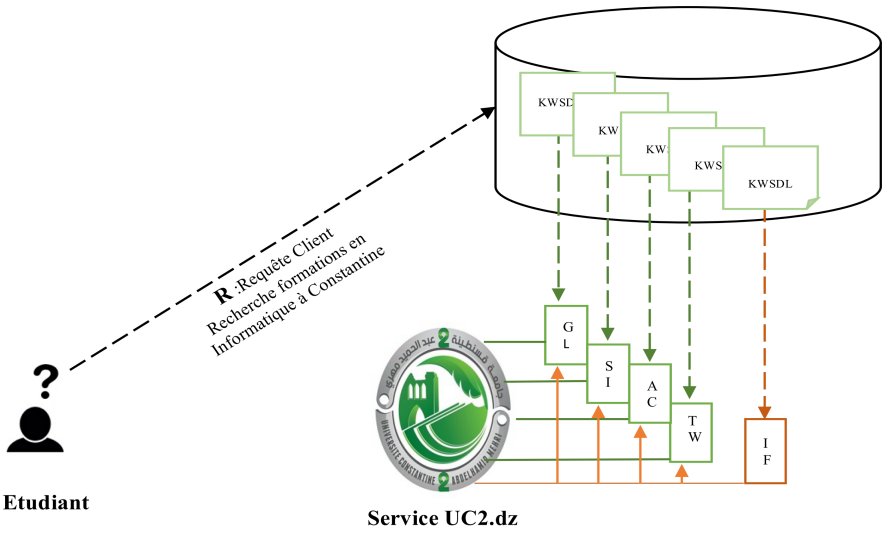


FIGURE 5.8 – Exemple Inscription Etudiant 2

Nous reprenons dans cette section l'exemple du service UC2.dz et supposons que les services publiés relatifs aux formations sont décrits en KWSDL. Nous notons à travers la Figure 5.8 qu'un service a été rajouté à la liste retournée (service IF «Inscription Formation»). Ce service est rattaché à toutes les formations publiées ce qui induit que l'étudiant est en mesure de s'inscrire.
Nous constatons à travers l'application de cette approche que le fait de doter une description d'un sens augmente considérablement la pertinence des résultats de la phase de découverte. Nous pouvons pousser cette extension sémantique plus loin par son analyse. En effet KWSDL bénéficie des avantages de l'environnement Maude du fait que K soit implanté dessus. Toute spécification dans K peut être automatiquement traduite en spécification Maude et exploitée dans son environnement.

```
KWSDL Service Name Student:
{définitions Service Name = Student:
{TargeNameSpace
Tns="http://www.UC2.dz/kwsdl/Student.kwsdl"
DefaultNameSpace Dns="http://schemasxmlsoap.org/kwsdl"
message NumCardSearch request : "NumCardStudentRequest"
message NumCardSearch response: "NumCardStudentResponse"
port SdudentPort = Input request: "NumCardStudentRequest"
port SdudentPort = Output request: "NumCardStudentResponse"
        EncodingStyle =
"http://schemasxmlsoap.org/soap/encoding/"
        Tns = ""http://www.UC2.dz/kwsdl/
 use = encoded
 service Sdudent
 StudBinding
 StudentPort
 ServiceLocation : "http://www.UC2.dz/"
exchange UC2 . SdudentPort to StudBinding .
SendnumCardStudentRequest
exchange Student.receiveNumCardStudentResponse to
StudBinding .StudentoPort
}
}
```

FIGURE 5.9 – Service UC2dz.kwsdl

La Figure 5.9 décrit le service UC2.dz suivant la syntaxe donnée en amont. Après la compilation de la grammaire (Figure 5.6) le résultat de l'exécution de ce service (Figure 5.10) reflète son état à travers la configuration donnée. L'interaction Etudiant/Université (confirmant l'inscription de ce dernier par la délivrance de sa carte d'étudiant) est exprimée par la balise exchange.

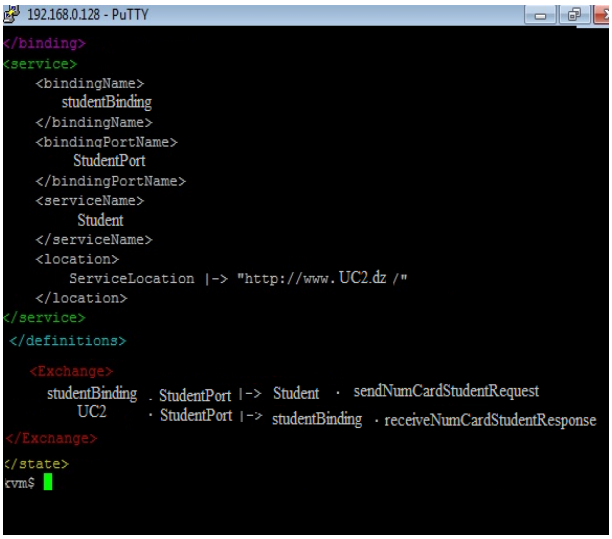


FIGURE 5.10 – Résultat de l'exécution du service UC2dz.kwsdl

L'exécution du modèle d'échange traduite par cet exemple suppose que des modèles plus complexes peuvent être, aussi, pris en charge par K et par conséquent, prouve l'apport signifiant de cette approche.
L'interopérabilité constitue un point important à considérer, compte tenu de sa relation intrinsèque avec le processus d'interaction d'un service Web. Notre approche a démontré à travers l'exploitation du langage KWSDL que la sémantique inférée garantit une interopérabilité sans failles.
En effet, la sémantique de base que l'on pouvait apparenter au standard, bien qu'il n'en ait pas est une sémantique fonctionnelle : c'est à dire que les messages s'échangeaient correctement mais sont non assimilés par le destinataire. Aussi, d'autres vraies sémantiques ont été inférées à WSDL par annotations ou ontologies montraient leurs limites dès que les modèles d'échange devenaient plus complexes. Par exemple, la différence entre les ontologies représentant deux services communicants.
Enfin, dans le cadre K, deux services communicant ont une compréhension commune de la signification des données qu'ils échangent. Cette interopérabilité sémantique reflète la qualité de communication.
Notons aussi que l'utilisation d'un tel langage permet d'éviter les redondances lors de leur découverte et rend le service alors plus efficace et plus sécurisé.

5.7 Conclusion

Dans cette approche, nous avons présenté un langage de description de services Web alternatif au standard WSDL nommé KWSDL assurant une interopérabilité sémantique entre services lors de leurs échanges. Ce langage a été mis en place au moyen de l'outil K. Nous avons défini sa syntaxe et sa sémantique opérationnelle en terme de règles de réécriture. Les résultats de l'exploitation de cette approche ont permis de soulever certaines confusions liées au problème de l'interopérabilité causé principalement par le manque de sémantique formelle inhérente à WSDL.

Chapitre 6

WS-SIM : Une approche de Découverte et de Sélection des Services Web

6.1 Introduction

Le processus de découverte de services est une étape cruciale dans le développement et le déploiement des services Web. Actuellement, de nombreux services Web aux fonctionnalités similaires sont publiés par des fournisseurs concurrents, ce qui complique davantage la tâche au client quant au choix du service à invoquer.
Plusieurs services Web découverts sur le registre UDDI peuvent combler les besoins du client, mais quel est le service adéquat ?. Il devient donc nécessaire, afin de faire face à ce problème, d'utiliser un mécanisme de jumelage pouvant évaluer le degré de similarité entre les descriptions des services découverts afin d'en relever les plus pertinents.

En effet, une découverte efficace de services Web consiste à trouver les services les plus pertinents parmi ceux fonctionnellement équivalents et ce par rapport à une demande spécifique du client. Automatiser ce processus tend, non seulement, à faciliter et à accélérer l'exécution des processus qui lui sont intrinsèquement liés tels que la sélection, la composition ou l'invocation. Mais aussi à réduire les redondances de services décelées.

Dans ce chapitre, nous présentons WS-Sim : une approche basée sur la logique de réécriture permettant de mesurer l'équivalence comportementale entre services Web. Notre travail repose sur une technique de matching faisant correspondre des descriptions de services écrites en KWSDL. L'usage des concepts formels pour identifier des relations sémantiques entre services privilégie une recherche fondée sur les fonctionnalités exécutées par le service qui sont considérées plus importantes que l'aspect structurel que reflètent les définitions standards.

En conséquence, notre contribution fournit une solution formelle pour l'optimisation du processus de découverte et de sélection des services Web. Elle est dotée aussi d'un mécanisme de substitution permettant la réutilisation et le remplacement des services similaires dans certains cas (par exemple, lors de la reconfiguration du système en cas de panne de services).

Ce chapitre est structuré comme suit :
Nous présentons en premier lieu le principe de notre approche à travers ses différentes étapes tout en motivant ces dernières par l'application d'un exemple concret. Afin de raisonner de manière exhaustive, nous réétudions le modèle comportemental des services Web de manière à atteindre un niveau d'abstraction plus élevé en considérant exclusivement les informations sémantiques jugées pertinentes. La sémantique du modèle étant basée sur la logique de réécriture, nous spécifions donc un service comme une théorie de réécriture. De cette théorie découle un modèle sémantique permettant la catégorisation d'un service. L'estimation de l'équivalence comportementale est rendu possible grâce à cette catégorisation par le biais de foncteurs. Enfin, nous concluons en citant les apports de cette approche.

6.2 Principe de WS-SIM

Notre approche contribue progressivement à la mise en place d'un support formel permettant d'estimer l'équivalence entre services Web. Cette action graduée est instaurée de façon à respecter l'enchainement des phases constituant respectivement le cycle de développement d'un service Web standard.
Plus particulièrement et si l'on se réfère au schéma global du cycle de vie du service Web. Cette approche traite les deux phases de découverte et de sélection. Afin d'effectuer une mesure de similarité entre services Web, nous contribuons à l'élaboration d'une approche hybride combinant à la fois la mesure de l'équivalence structurelle et celle de l'équivalence comportementale. Notre approche est illustrée par la Figure 6.1 et se compose des étapes suivantes :

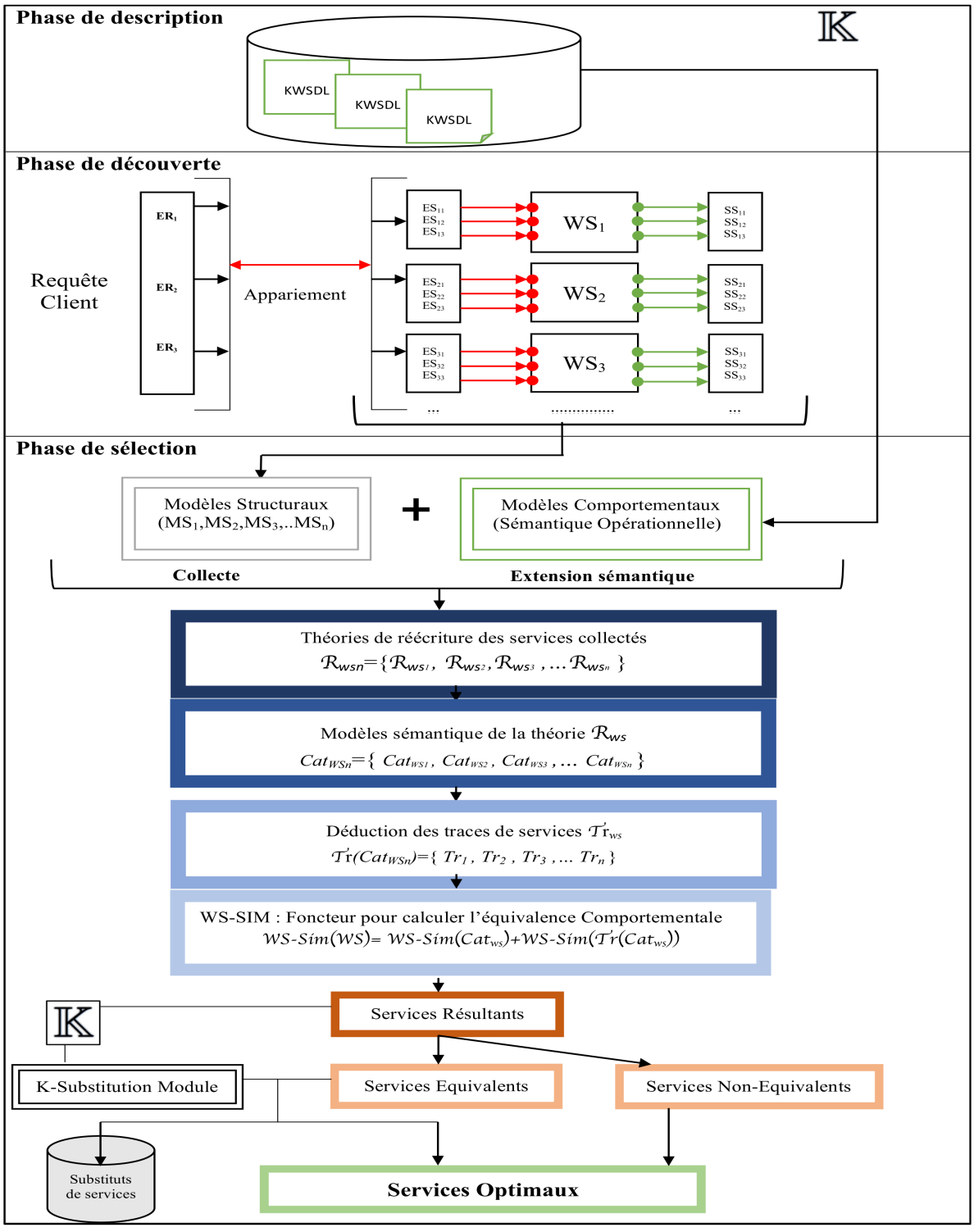


FIGURE 6.1 – Approche de mesure de similarité WS-SIM

Phase de découverte :
Une requête utilisateur formée d'un ensemble de mots clés (Entrées Requête ER) est d'abord comparée automatiquement aux mots clés (Entrées Services ES) des descriptions de services se trouvant dans le registre via un algorithme d'appariement, une liste de services jugés susceptibles de répondre favorablement aux besoins formulés est retournée.
Phase de sélection :

— La collecte issue de l'étape de découverte constitue le support structurel de l'approche. Nous complétons ce dernier en générant automatiquement son extension sémantique en nous référant au langage KWSDL (**phase de description**).

— Afin de pouvoir estimer le degré d'équivalence entre les services Web représentés respectivement par les modèles sus-citées et étant donné que la sémantique inférée à ces derniers est définie en termes de règles de réécriture, nous donnons, à ce niveau, une nouvelle définition générique et formelle à ces services. En effet, nous considérons un service comme une théorie de réécriture $\mathcal{R}_{WS}$ en faisant davantage abstraction de tous les détails d'implémentation et de structure inutiles.

— De cette théorie émane le modèle sémantique des services Web Cat_{WS} qui est utilisé pour catégoriser chaque service en vue de la sémantique qui lui est associée.

— Cette sémantique catégorielle permet, d'une part, de déduire les traces possibles d'exécution de services $\mathcal{T}r_{WS}$ et d'autre part, au moyen de la théorie des catégories de définir des liens formels nommés foncteurs entre catégories de services pour estimer le degré d'équivalence entre elles.

— Nous nommons notre approche de mesure de l'équivalence entre services **WS-Sim.** Les artefacts produits par l'application de cette dernière sont automatiquement analysés ; les services jugés équivalents sont soumis au processus de substitution en utilisant le module «K-Substituion» tiré de l'outil **K** pour évaluer les possibilités de substitution ou/et de remplacement de ces derniers.
(Raisonnement K-substitution : deux règles sont considérées comme substituables si et seulement si, leurs applications conduisent aux mêmes configurations).

— La concaténation des résultats de substitution et de la liste des services jugés non-équivalents constitue l'ensemble des services optimaux pour la requête formulée en amont.

— Enfin, nous consacrons dans notre approche une base de donnée pour stocker les substituts de services générés pour une réutilisation ultérieure.

6.3 Exemple de motivation

Afin de démontrer l'efficacité de notre approche, nous présentons dans cette section l'exemple du service composite «Achat livre». Ce service est destiné à effectuer des recherches sur des livres, d'informer le client de leurs éventuelles disponibilités et de permettre leur achat.
L'expérience est dirigée à travers les étapes énoncées en amont :
Phase de découverte (Modèle structurel) :
Le modèle abstrait du service est extrait de l'annuaire des services UDDI ; une partie de sa description est exposée dans la Figure qui suit :

```
.....

<message name="SearchBookRequest">
  <part name="Book" type="xs:string"/>
</message>

<message name="SearchBookResponse">
  <part name="BookTitle" type="xs:string"/>
</message>

<portType name="BookSearch">
  <operation name="getBookTitle">
    <input message="SearchBookRequest"/>
    <output message="SearchBookResponse"/>
  </operation>
</portType>

...
```

FIGURE 6.2 – Partie de la description WSDL du service achat livre

Phase de sélection (Modèle Comportemental) :
Le module AchatLivre (Figure 6.3) est le résultat du mapping et de l'extension sémantique du modèle de la Figure 6.2. L'extension sémantique impose la restructuration de la syntaxe WSDL. Seuls les concepts WSDL reflétant le processus d'interaction du service sont formalisés (messages et types de message).
A titre d'exemple, l'application de la règle portant le nom **BookSearch** à la configuration <**available book**> traduit la reception de la requête par le **port d'entrée** P_1 du service et son traitement qui renvoie par **le port de sortie le nom du livre recherché** à travers le message de réponse.
Les trois points figurant aux extrémités de la cellule de la configuration représentent les parties en amont et aval de la configuration qui ne sont pas affectées par la règle BookSearch.

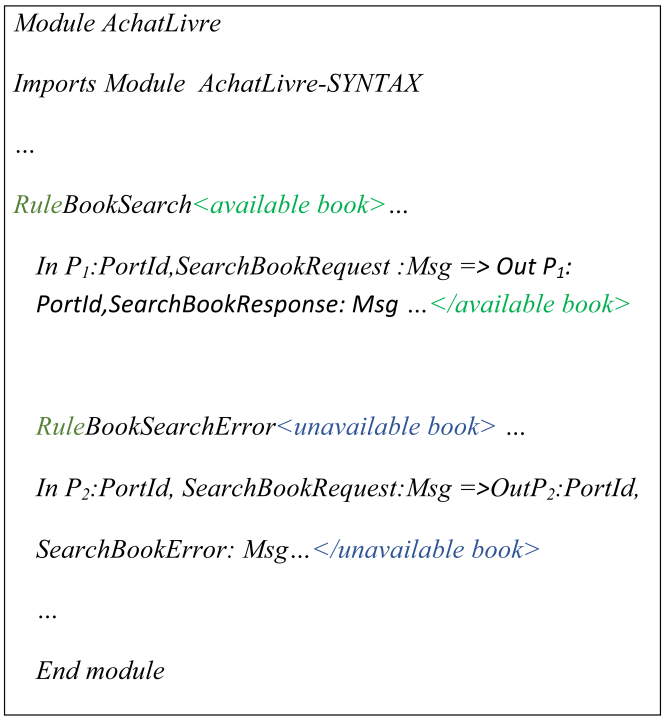


FIGURE 6.3 – Sémantique K inférée à la description WSDL

Nous définissons dans ce qui suit la théorie de réécriture d'un service Web et déduisons de cette dernière la théorie du service Achat Livre. Les services représentés en tant que système de réécriture par une théorie de réécriture ont l'avantage d'exprimer la sémantique opérationnelle définie. L'intérêt de notre approche est d'augmenter principalement la réutilisation des services en réduisant au maximum le nombre de redondances. Les services issus de l'étape de sélection fournissent un modèle initial abstrait de composition.

6.4 Théorie de réécriture d'un service Web

Formellement un service Web peut être défini de manière alternative par une théorie de réécriture étiquetée. Nous détaillerons dans ce qui suit cette théorie. L'avantage d'une telle définition est la prise en charge de tous les aspects structurels et comportementaux d'un tel service. En effet, le modèle sémantique associé à une théorie de réécriture [Mes 92] associe à un service Web une sémantique catégorielle où les objets définissent les configurations (états) possibles d'un service Web. Les morphismes (flèches de la catégorie) représentent une évolution possible du service d'un état à un autre. Dans la théorie des catégories, nous retrouvons les concepts appropriés pour modéliser la dynamique comportementale d'un service. Entre autres, nous identifions :

Les Configurations (ou états) qui représentent dans un modèle de service Web les situations possibles qui peuvent se produire avant et après l'invocation d'un service. Une configuration est définie par un *Nom*, un *Type* et le *Message reçu*.

Les Types de Configurations(initiale c_0, intermédiaire c_x et finale c_f). La configuration initiale c_0 représente le début de l'invocation du service, la configuration intermédiaire c_x représente le service à un instant t de son processus d'exécution et la configuration finale c_f est le dernier point atteint du processus d'exécution.

Le Message dans notre cas sert à contenir les informations véhiculées à travers les configurations du système. Il est caractérisé par un *Nom* et un *Type*.

Les Types de Messages : il existe dans le contexte des services Web deux types de messages :
Des messages pour exprimer les besoins nommés messages **Requête** et des messages répondant à ces requêtes : les messages **Réponse**.

Evolution : l'évolution de l'état des configurations du service se traduit par les transformations successives des configurations existantes en de nouvelles configurations par l'application d'un ensemble d'opérations relatives aux échanges de messages effectués (requête, réponse).

Comportement/Trace : le comportement global du service peut être résumé par les séquences de transformations effectuées (**configuration**, **messages échangés**) ; ces séquences reflètent les traces d'exécution du service partant d'une configuration initiale.

Définition 6.1. *Soit $\mathcal{R}_{WS}$=(Σ_{WS},E_{WS},L_{WS},R_{WS}) la théorie de réécriture définissant un service Web noté WS, où :*

- *La signature du service est donnée par le couple :(Σ_{WS}, E_{WS}), elle sert à définir la syntaxe algébrique des éléments constituant un service Web.*
- *L_{WS} est l'ensemble des étiquettes utilisées pour repérer les règles de réécriture.*
- *R_{WS} est l'ensemble des règles de réécriture locales représentant l'évolution ou la dynamique des états d'un service WS.*

Dans la Figure ci-dessous (Figure 6.4), nous identifions une sous théorie de réécriture générique valable quelque soit le service WS à laquelle nous rajoutons les éléments syntaxiques propres à un service donné ainsi que ses règles de réécriture pour former $\mathcal{R}_{WS}$. L'exemple ci-dessous explique cette remarque.

Sortes *Msg, MesName, MesType, ListMsg* *Conf,IdConf, ListConf* *Soussortes* *RequestMsgResponseMsgNotificationMsg<* *MesType* *Conf<ListConf* *Msg<ListMsg*	*Opérations :* *isEmpty : ListConf→Bool* *left, right : ListConf→Conf* *initialConf, finalConf: →Conf* *Nill :→ListConf* *<_,_,_> : IdConf,Msg→Conf* *_;_: ListConf,ListConf→ListConf* *Succ : Conf→Conf* *_and_ :ListMsg,ListMsg→ListMsg* *[_,_] : MesName,MesType→Msg*	*Variables* *C: Conf, L:ListConf* *Equations :* *isEmpty(L)=if L=Nill then True else false;* *left (Nill)= Nill ;* *left (C;L)= C ;* *right(Nill)= Nilll ;* *right(C;L)= right(L) ;* *right(C)=C;*

FIGURE 6.4 – Définition de base de la théorie $\mathcal{R}_{WS}$

Exemple 6.1.

Reprenons l'exemple de l'achat du livre énoncé précédemment ; la Figure 6.5 représente la sémantique considérée mettant en relief les différentes configurations et règles opérationnelles agissant sur ces configurations.

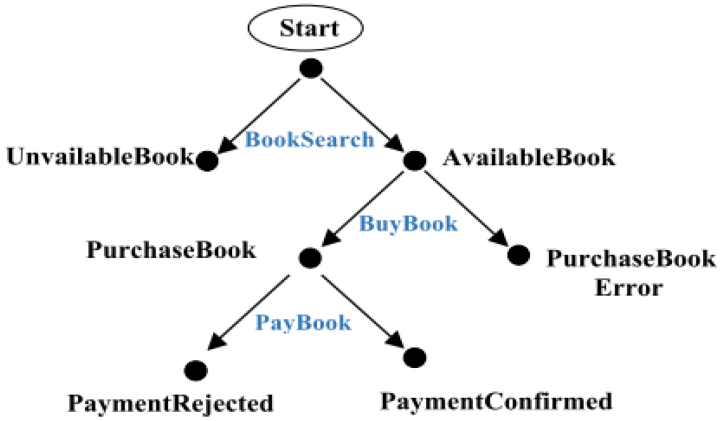


FIGURE 6.5 – Représentation sémantique du service Achat livre

Pour former les opérations (respectivement les équations) de la théorie $\mathcal{R}_{achatLivre}$, nous enrichissons celles présentées dans la Figure 6.4 par ces opérations spécifiques (constructeurs des deux sortes Msg et Conf) :

Opérations :

Start, AvailableBook, UnavailableBook, PurchaseBook, PurchaseBookError, PaymentConfirmed, PaymentRejected : $\rightarrow$ *IdConf*

SearchBookRequest, SearchBookResponse, SearchBookError,CommandBookRequest, CommandBookResponse,CommandBookError,PayBookRequest,PayBookResponse,PayBookError : $\rightarrow$ *MesName*
Equations :
initialConfig =<Start, [SearchBookRequest, ResquestMsg]> ;
finalConfig=<PaymentRejected, [PayBookResponse, ResponseMsg]> ;
finalConfig=<PaymentRejected, [PayBookError, NotificationMsg]> ;

Les règles locales étiquetées $\mathcal{R}_{achatLivre}$ dans ce cas sont énoncées comme suit :

L= {*BookSearch,BookSearchError,BuyBook,BuyBookError,PayBook,PayBookError*}
[**BookSearch**] :*initialConfig* $\rightarrow$ <*AvailableBook,[SearchBookResponse, ResponseMsg]*>,
[**BookSearchError**] : *initialConfig* $\rightarrow$ <*UnvailableBook,[SearchBookError, NotificationMsg]*>,
[**BuyBook**] : <*AvailableBook, [CommandBookRequest, RequestMsg]*> $\rightarrow$
<*PurchaseBook,[CommandBookResponse, ResponseMsg]*>,
[**BuyBookError**] : <*AvailableBook,[CommandBookRequest, RequestMsg]*> $\rightarrow$
<*PurchaseBookError,[CommandBookError, ResponseMsg]*>,
[**PayBook**] : <*PurchaseBook,[PayBookRequest, RequestMsg]*> $\rightarrow$
<*PaymentConfirmed,[PayBookResponse, ResponseMsg]*>,
[**PayBookError**] : <*PurchaseBook,[PayBookRequest, RequestMsg]*> $\rightarrow$
<*PaymentRejected, [PayBookError, NotificationMsg]*>.

Un calcul pour ce service Web peut correspondre au lancement d'une requête à la recherche d'un livre. Cela se traduit bien sûr par la séquence des règles de réécriture (règles locales telles qu'elles sont définies et les règles de déduction de la logique de réécriture). En effet, une trace de calcul peut être déduite par la séquence (configuration-règle locale- nouvelle configuration),ie, $\mathcal{T}r_{1achatlivre}$=*Start-**SearchBook**-AvailableBook-**BuyBook**-PurchaseBook ...*

6.5 Sémantique catégorielle d'un service Web

Evidement, le modèle mathématique qui reflète le comportement d'une théorie de réécriture doit pouvoir véhiculer deux types d'identités : les termes générés par la spécification algébrique (Σ,*E)* et les preuves générées par la déduction. Nous présentons alors le modèle comme étant une catégorie où les objets sont les Σ-termes (configurations ou états d'un WS) et les morphismes représentent les preuves entre eux (évolutions des états du WS).
Nous montrons dans cette section que les services Web héritent du modèle sémantique d'une théorie de réécriture, leurs modèles forment des catégories. Nous définissons pour ce fait, des morphismes entre leurs états, ensuite nous établissons un foncteur entre deux catégories de services (WS1 et WS2) pour formaliser la similarité qui peut exister entre eux.
Le modèle sémantique d'un service Web défini par $\mathcal{R}_{WS}$= (Σ, E, L, R) est une catégorie Cat_{WS}= (C_{WS},M_{WS}) donnée par :

— C_{WS} un ensemble de configurations représentant les objets de la catégorie, chaque configuration (état) est représentée par un nom (IdConf).

— M_{WS} un ensemble de morphismes entre ces objets caractérisés par leur nom, domaine et codomaine,

— $\circ$ une opération de composition des morphismes où pour tout couple de morphismes (f : A→B, g : B→C), $f \circ$ g : A→C est un élément de l'ensemble des morphismes qui vérifie :

— L'existence de l'identité : pour toute Configuration, il existe un morphisme id_C : C→C.

— L'associativité de la composition : pour tout morphisme f :A→B, g :B→C et h :C→D, $f \circ$ (g $\circ$ h) = ($f \circ$ g)$\circ$ h.
Les différentes configurations d'un WS peuvent se répartir en classes d'équivalence [C] selon la signature adoptée (Σ_{WS},E_{WS}) et forment ainsi les objets de la catégorie Cat_{WS}. Les morphismes M_{WS} de cette catégorie sont définis récursivement par :

— le morphisme identité id_{C_1} :$[C_1]$→$[C_1]$ exprimant la règle réflexivité qui ne fait aucun changement sur une configuration C_1.
— la règle locale R_i : $[C_1] \rightarrow [C_2]$ faisant évoluer une configuration C_1 représentante de toute une classe d'équivalence vers une autre configuration C_2.
— la composition de deux ou plusieurs morphismes (preuves de calcul) définissant une trace d'exécution d'un WS (application de plusieurs règles locales et les règles de déduction de la logique de réécriture).

Il est important de noter que les morphismes peuvent être aussi définis par des classes d'équivalence de preuves. Les exécutions équivalentes sont réunies dans une même classe.

6.5.1 Equivalence sémantique entre services Web

Une conséquence assez attrayante de la définition précédente est déduite pour mesurer le degré de similarité sémantique entre les services Web ; pour cela, nous établissons un foncteur entre les modèles sémantiques définis pour les services Web en question. Cette section donne plus de détails concernant cette approche.

Définition 6.2. *Soient Cat_{WS1}=(C_{WS1},M_{WS1}) et Cat_{WS2}=(C_{WS1},M_{WS2}) deux catégories associées à deux services WS_1 et WS_2, le foncteur SIM de Cat_{WS1} vers Cat_{WS2} noté : SIM : $Cat_{WS1} \rightarrow Cat_{WS2}$ est la paire (SIM_C,SIM_R) où SIM_C est une application qui à toute configuration de C_{WS1} associe une configuration de C_{WS2} et SIM_R est aussi une application qui à tout morphisme de M_{WS1} $f : A \rightarrow B$ associe une flèche de M_{WS2},$SIM_M(f)$: $SIM_C(A) \rightarrow SIM_C(B)$ tels que :*

— *Le morphisme identité est préservé : $SIM_M(id_A) = id_{SIM_C(A)}$;*

— *La composition est aussi préservée,* $SIM_M(g \circ f) = SIM_M(g) \circ SIM(f)$, $\forall$ *f et g flèches de* M_{WS2}.

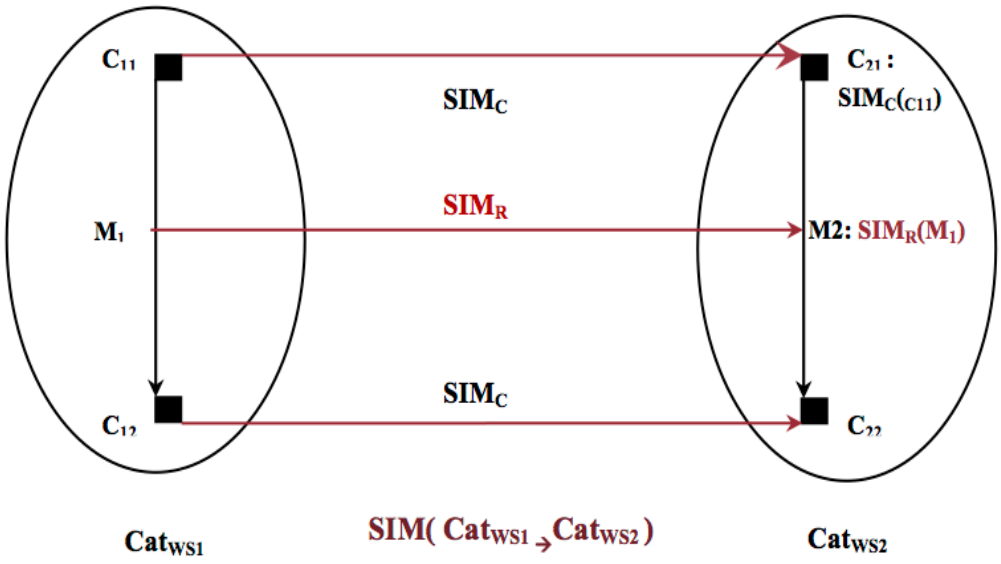


FIGURE 6.6 – Le Foncteur SIM=(SIM_C,SIM_R) Entre Cat_{WS1} et Cat_{WS2}

En terme de service, l'équivalence comportementale est déduite respectivement par l'équivalence entre leurs configurations et celle des règles qui agissent sur ces configurations ; nous donnons dans ce qui suit l'interprétation de cette équivalence :
Le foncteur SIM : $Cat_{WS1} \rightarrow Cat_{WS2}$ (Figure 6.6) ou SIM : $(C,R)_{WS1} \rightarrow (C,R)_{WS2}$ est basé sur les équivalences suivantes :
Equivalence entre configurations de services SIM_C :
Le calcul de l'équivalence entre les configurations de deux catégories de services est traduit par :

Proposition 6.1. *Deux configurations de services sont dites équivalentes, si et seulement si les messages qu'elles comportent sont des conséquences directes et équivalentes par rapport à une requête donnée. La formule qui calcule l'équivalence entre les configurations de services prend la forme suivante :*
$SIM_C \in <SIM_{C1}, SIM_{C2}>$ *Tels que :*

— $SIM_{C1}(NC1)=NC2$, *NC1 et NC2 sont les noms ou identificateurs des configurations de services.*
— $SIM_{C1}(TM1)=TM2$, *TM1,TM2 représentent les types de messages échangés*
— $SIM_{C1}(M1)=M2$, *M1,M2 sont les messages véhiculés entre services.*

Equivalence entre règles SIM_R :

Proposition 6.2. *Nous considérons que deux règles opérationnelles sont équivalentes, si et seulement si elles produisent les mêmes effets à partir de deux configurations jugées similaires.*
En d'autres termes, si la relation left(r)-right(r) (configuration) des règles r_1 et r_2 amène à deux configurations similaires, r_1 et r_2 sont dites similaires.
La similarité entre les règles est évaluée en calculant la similarité entre les configurations du coté gauche et droit régies par les deux règles respectives.
(left(r_1), r1) → right(r_1)
(left(r_2), r2) → right(r_2)
l'équivalence entre r1 et r2 est traduite par : SIM_R(r1)=r2 où les conditions suivantes doivent être satisfaites :

- *Sim_C (left(r_1))= left(r_2).*
- *Sim_C (right(r_1))= right(r_2).*

Ainsi, l'équivalence entre trace d'exécution est déduite de l'équivalence entre les séquences de règles appliquées aux configurations de services (ensemble des termes de preuve). Une relation de trace d'équivalence entre deux services WS1 et WS2, s'écrit WS1 $\simeq$ tr WS2.

Exemple 6.2.

Afin de vérifier l'efficacité de l'approche, une étude de cas est menée pour évaluer la similarité entre services Web, nous reprenons l'exemple AchatLivre. L'expérience est dirigée à travers les étapes énoncées en amont.
Après découverte des modèles structuraux WSDL, nous procédons à l'extraction des données sémantiques de leurs extensions KWSDL (Figure 6.7).
Nous supposons dans ce cas qu'il existe un second service AchatLivre2 retourné qui répond à la même requête.

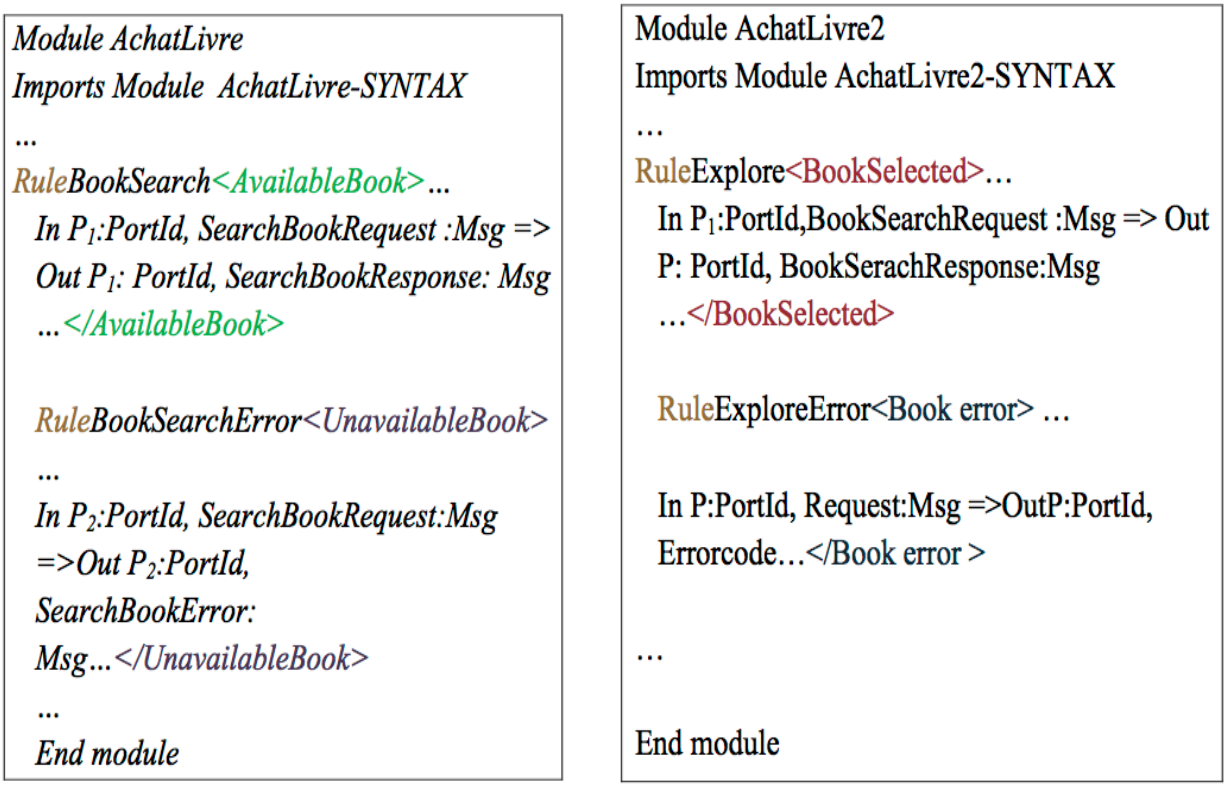


FIGURE 6.7 – Modèles sémantiques des services AchatLivre, AchatLivre2

Nous procédons, par la suite, à la spécification de la théorie du service AchatLivre2 de la même manière qu'elle a été définie pour le service AchatLivre. Les tableaux 6.1 et 6.2 représentent respectivement $\mathcal{R}_{achatlivre}$ et $\mathcal{R}_{achatlivre2}$.

Configuration left(r)	***Règles Locales***	***Configuration right(r)***	***Traces d'exécution***
Start, SearchBookRequest: initialConfig	*BookSearch*	*AvailableBook,SearchBookRequest*	*Start-booksearch-Available Book*
	BookSearch Error	*UnvailableBook,SearchBookResponse*	*Start-BooSearchError-UnvailableBook*
AvailableBook,Command BookRequest	*BuyBook*	*PurchaseBook,CommandBookResponse*	*Start-Booksearch-AvailableBook-BuyBook-PurchaseBook*
	BuyBookError	*PurchaseBookError,CommandBookError*	*Start-BookSearch-AvailableBook-BuyBookError-PurchaseBookError*
PurchaseBook,PayBookRequest	*PayBook*	*PaymentConfirmed,PayBookResponse*	*Start-Booksearch-AvailableBook-BuyBook-PurchaseBook-PayBook-PaymentConfirmed*
	PayBookError	*PaymentRejected,PayBookError*	*Start-Booksearch-AvailableBook-BuyBook-PurchaseBook-PayBookError-PaymentRejected*

TABLE 6.1 – Eléments de base de la théorie $\mathcal{R}_{achatlivre}$

Ponfiguration left(r)	***Règles Locales***	***Configuration right(r)***	***Traces d'exécution***
Start, BookSearchRequest: initialConfig	*Explore*	*BookSelected,SelectBookResponse*	*Start-explore-BookSelected*
	Explore Error	*BookError, SelectBookError*	*Start-ExploreError-BookError*
BookSelected,OrderBookRequest	*OrderBook*	*BookPurchase,OrderBookResponse*	*Start-Explore-SelectedBook-OrderBook-BookPurchase*
	OrderBookError	*BookPurchaseError,OrderBookError*	*Start-Explore-SelectedBook-OrderBookError-BookPurchaseError*
BookPurchase,PaymentRequest	*Payment*	*PaymentConfirmed,PaymentResponse*	*Start-Explore-SelectedBook-OrderBook-BookPurchase Payment-PaymentConfirmed*
	PaymentError	*PaymentRejected,PaymentNotification*	*Start-Explore-SelectedBook-OrderBook-BookPurchasePaymentError-PaymentRejected*

TABLE 6.2 – Eléments de base de la théorie $\mathcal{R}_{achatlivre2}$

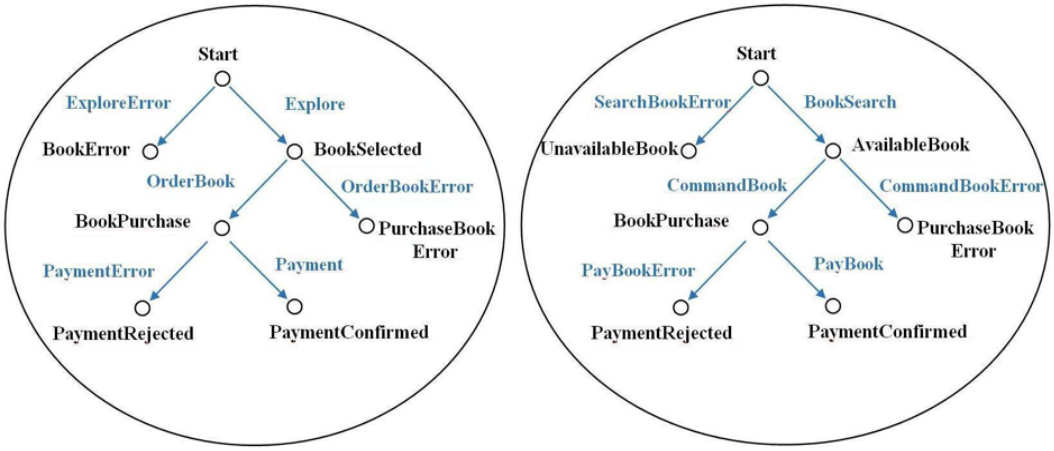


FIGURE 6.8 – Catégories $Cat_{achatlivre}$,$Cat_{achatlivre2}$

La Figure 6.8 illustre les deux catégories $Cat_{achatlivre}$ et $Cat_{achatlivre2}$.
A présent, nous déduisons de cette représentation la préservation des éléments de la catégorie $Cat_{achatlivre}$ par l'application du foncteur de similarité SIM. En effet, on retrouve l'équivalent de l'ensemble des configurations de la catégorie ainsi que des flèches qui les relient dans la catégorie $Cat_{achatlivre2}$.

6.6 Conclusion

Dans ce chapitre, nous avons proposé une approche formelle pour la mesure de l'équivalence comportementale entre services Web en comparant leurs comportements en fonction de la sémantique extraite de leurs descriptions KWSDL.
La technique d'appariement WS-Sim interprète la similarité sémantique entre un couple de services à travers leurs modèles sémantiques représentés par leurs catégories respectives. La préservation des objets et morphismes respectivement configurations et règles de réécriture agissant sur ses configurations induit à l'équivalence entre ces services. Les résultats de l'application de notre approche démontrent que WS-Sim est complète et plus appropriée par rapport aux approches étudiées dans la partie état de l'art en termes d'équivalence comportementale. En effet, nous avons prouvé par la faisabilité de cette approche, que la meilleure façon de mesurer la similarité entre services est de comparer leurs comportements réels.

Chapitre 7

R-Mop-ECATNet : Un modèle pour la Spécification et le Contrôle de la Composition Dynamique des Services Web

7.1 Introduction

La notion de service est un paradigme en cours d'évolution qui trouve ses racines dans les technologies existantes. Le modèle à base de composants apparaît comme l'architecture naturelle pour les services Web.
Ces services ont besoin de communiquer et leur environnement est dynamique et évolutif. En effet, de nouveaux services peuvent être ajoutés. Les services existants sont constamment modifiés, temporairement suspendus, ou définitivement supprimés. Ainsi, cette volatilité, variabilité et ce changement dynamique constant rendent difficile voire impossible la prédiction statique de tous les scénarios qui peuvent survenir au cours du processus de composition. Dans ce chapitre, nous présentons RMop-ECATNet (Refined-Meta Open ECATNet) : une définition alternative au modèle Mop-ECATNet dédiée à la spécification formelle de la composition de services Web. La spécification RMop-ECATNet est graphiquement plus riche et plus abstraite. De plus, ce modèle a la capacité d'organiser et de contrôler dynamiquement les éventuelles reconfigurations de services en prenant en compte leur aspect transactionnel tout en veillant à maintenir la flexibilité de la composition.
Nous montrons à travers cette restructuration du modèle Mop-ECATNet comment assurer la spécification formelle des services Web et contrôler leurs interactions (en terme de propriétés transactionnelles) ainsi que leur composition dynamique de manière plus simple et automatique.

Ce chapitre comporte quatre sections principales. Dans la section 2, nous faisons un bref rappel sur les réseaux de Petri algébriques de haut niveau, en particulier, sur leurs extensions dédiées à spécifier les services Web. Dans la section 3, nous illustrons à travers un exemple les limites du modèle Mop-ECATNet. La quatrième section est dédiée au principe de notre approche où nous présentons nos extensions du modèle Mop-ECATNet aux trois niveaux : structurel, comportemental et implémentation. Nous donnons, par la suite, la définition complète du modèle RMop-ECATNet par son niveau structurel (notations graphiques et structures proposées) et son niveau comportemental en spécifiant l'impact des propriétés transactionnelles sur la spécification du comportement. Nous présentons les stratégies de contrôle mises en œuvre pour la gestion de la reconfiguration dynamique des services et les patrons élaborés pour traiter des cas particuliers dans ce même processus.
Comme pour chaque contribution présentée, nous situons et comparons les avantages de cette dernière par rapport aux techniques de base déployées dans le cadre du CV_{WS}. La phase de composition est au cœur de cette comparaison dans ce chapitre.
Dans la phase de composition standard plusieurs langages de composition peuvent être utilisés tels que BPEL,OWL...etc. Ces langages ne sont pas standardisés ; ils exploitent les descriptions WSDL des services à composer (pas de sémantique). Ils sont difficiles à mettre en œuvre et à vérifier formellement. Par opposition à ce type de langages nous proposons, dans notre cadre formel, en dressant un parallèle à la phase de composition standard, un modèle formel de spécification du processus de composition RMop-ECATNet simple par sa représentation graphique, riche en terme d'informations par sa source (KWSDL) et adapté à la nature versatile des services Web, d'autant plus que ce formalisme peut être automatiquement traduit en code Maude exécutable et vérifié formellement (point discuté dans le chapitre suivant).

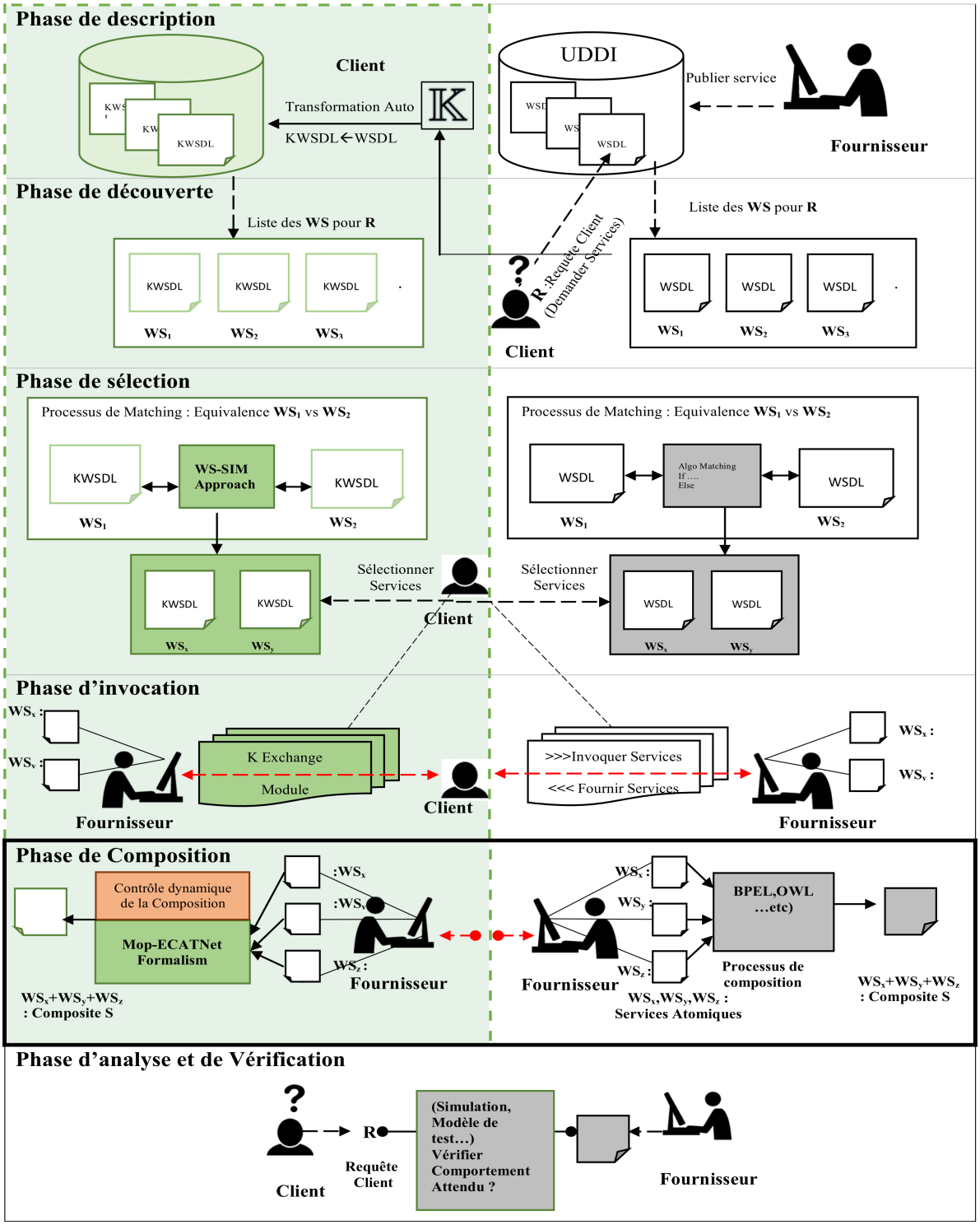


FIGURE 7.1 – RMopECATNet dans le CV_{WS}

7.2 Réseaux de Petri Algébriques de haut niveau

Les réseaux de Petri modélisent de façon relativement simple l'aspect opérationnel des systèmes du fait de leur représentation graphique. Beaucoup de notions peuvent être exprimées par ces modèles, notamment, le parallélisme telles que la synchronisation et la vraie concurrence. Cependant, cette simplicité est considérablement réduite dès que le nombre de composants du système accroit. La difficulté de décrire les structures de données manipulées y participe aussi grandement.
Les spécifications algébriques permettent de décrire ces données. Elles permettent, entre autres, de construire des spécifications de façon modulaire et hiérarchique. Plus précisément, les spécifications algébriques sont orientées vers le changement d'états de données et non vers la structure de contrôle du système. On peut dire qu'elles agissent plus d'un point de vue statique que dynamique.
Les RdPs et les spécifications algébriques sont donc deux méthodes formelles de spécification qui procèdent de manières différentes mais qui peuvent être complémentaires. En effet, leur combinaison a donné lieu aux Réseaux de Petri à termes algébriques qui sont considérés comme des réseaux de Petri de haut niveau. Ces derniers permettent de décrire, de manière cohérente et unifiée à la fois, les aspects statiques et dynamiques des systèmes concurrents sans pour autant changer de modèle de spécification.
Dans ce qui suit nous présentons un de ces modèles hybrides : le modèle des ECATNets à travers sa structure et son comportement dynamique. Nous accordons un interêt particulier aux extensions de ce modèle qui cadrent avec le contexte de l'approche orientée service. Pour ce faire, nous procédons de manière incrémentale en termes des extensions faites, nous présentons d'abord le formalisme des ECATNets puis son extension dédiée à représenter les systèmes ouverts : les Op-ECATNets et enfin nous terminons par leur ultime abstraction à deux niveaux les Mop-ECATNets. Chaque formalisme est présenté à travers un exemple succinct illustrant son rôle.

7.2.1 Modèle des ECATNets

Les ECATNets sont une catégorie de réseaux de Petri algébriques basés sur une combinaison saine du formalisme des types abstraits algébriques et de celui des réseaux de Petri de haut niveau ayant pour objectif la modélisation, la validation et la simulation des systèmes parallèles. Ce modèle utilise une sémantique exprimée en termes de logique de réécriture qui permet d'appréhender de manière correcte et élégante la sémantique du vrai parallélisme. La définition formelle du modèle des ECATNets est donnée par :

Définition 7.1. *Un ECATNet est une paire E= (Spec,R) où :*
Spec = (Σ, E) est une spécification algébrique tels que :
Σ= (S,Op) est une signature
***S** est un ensemble de sortes.*
***Op** est un ensemble d'opérations sur S.*
***E** est un ensemble de Σ-équations.*

R *est un uplet tel que R = (P,T,σ,Cap,IC,DT,CT,TC)*
P *est un ensemble fini de places.*
T *est un ensemble fini de transitions (P ⋂T=∅)*
σ est l'application de typage des places.
Cap *est la fonction de capacité des places.*
IC *est la condition d'entrée (Input Condition).*
DT *est fonction des jetons détruits (DestroyedTokens).*
CT *est la fonction des jetons créés (CreatedTokens).*
TC *est la condition de transition (Transition Condition).*

La spécification algébrique Spec définit les structures de données manipulées par le système. La structure de ***R*** décrit le comportement du système. L'ensemble des places ***P*** symbolise les conditions ou ressources du système (variables d'état). Les transitions ***T*** correspondent aux évènements qui modifient les informations de contrôle.

7.2.1.1 Représentation Graphique

Comme dans les autres types de réseaux de Petri, le réseau d'un ECATNet est représenté par un graphe orienté biparti dont l'ensemble des sommets est P⋃T (les places étant représentées par des cercles ou des éllipses et les transitions par des rectangles ou des barres). La représentation graphique de la cellule d'un réseau d'ECATNet a la forme suivante :

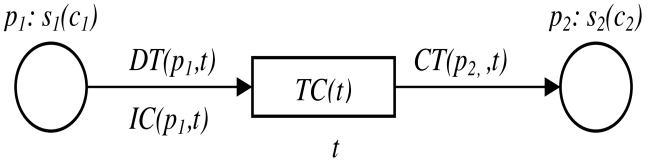


FIGURE 7.2 – Représentation graphique de la cellule d'un réseau ECATNet

Dans la Figure 7.2, les places p_1 et p_2 sont respectivement de sorte s_1 et s_2 ($i.e.\sigma(p_1)$=s_1 et $\sigma(p_2)$=s_2) et de capacité c_1 et c_2 ($i.e.Cap(p_1)$=c_1 et Cap (p_2)=c_2). IC(p_1,t) est écrite à gauche de l'arc en entrée à la transition t (pour un observateur situé au niveau de la transition). Les jetons à détruire DT(p_1,t) sont écrits à droite de l'arc d'entrée à la transition t. Les jetons créés CT(p_2,t) constituent la valeur de l'arc sortant de la transition t vers la place p_2. Dans le cas où IC(p_1,t) = DT(p_1,t), une seule de ces inscriptions suffit dans le réseau.

7.2.1.2 Aspect Comportemental d'un ECATNet

Un ECATNet peut être vu comme un modèle de système concurrent. En effet, si plusieurs transitions sont sensibilisées en même temps et qu'il n'y a pas de conflits entre

elles, elles peuvent être tirées en parallèle. Ce comportement peut être décrit de manière naturelle par la logique de réécriture. L'intérêt majeur de cette logique réside dans sa capacité d'exprimer tout comportement parallèle, au sens de la sémantique du vrai parallélisme, par des déductions dans cette logique. Le système évoluant passe d'un état s à un état s'. Cette transformation est le résultat d'une composition parallèle d'un ensemble de changements élémentaires. Ces changements sont décrits à l'aide d'axiomes modélisant le comportement dynamique d'un ECATNet. Ainsi l'axiome, $u \rightarrow v$ if c, exprime que l'état partiel extrait de l'état global s est à remplacer par l'état partiel v si la condition c est vérifiée ; ce qui fait passer l'état s vers s'.
De manière plus concrète, l'état global d'un système est aussi décrit par une structure de classes d'équivalence formées autour des multi-ensembles de (P×CATdas(E,X)) modulo les axiomes d'associativité, de commutativité et de l'identité de l'opération⊗. Mais pour simplifier, cet état est généralement noté :<pi, M(pi)>$\bigotimes_{i=1}^{n}\langle p_i, M(p_i)\rangle$. De plus, chaque transition t d'un ECATNet est traduite en un axiome (règle de réécriture) dont la forme dépend essentiellement de la relation entre IC(p,t) et DT(p,t). Nous distinguons ici les trois situations les plus concrètes dans le cas où IC(p,t) est définie positivement.

Cas 1 : IC(p,t) = DT(p,t)
La règle de réécriture relative à ce cas consiste à exprimer le retrait du terme <p, IC(p,t)> de l'état global et le remplacer par le terme <p', CT(p',t)>. Cette situation correspond au cas usuel des réseaux de Petri.
t : <p, IC(p,t)>→<p', CT(p',t)>

cas 2 : IC (p,t)⋂ DT(p,t)= ∅
La règle décrivant le comportement parallèle dans ce cas s'exprime par :
t : (p, DT(p,t))→(p', CT(p',t) if (IC(p,t)⋂M(p))→IC(p,t)

Cas 3 : IC(p,t)⋂DT(p,t)≠∅
Ce cas peut se ramener à une combinaison judicieuse des cas 1 et 2.

Exemple 7.1.

L'exemple présente une cellule de production qui fabrique des pièces forgées de métal à l'aide d'une pression. Cette cellule se compose d'une table A qui sert à alimenter la cellule en pièces brutes d'un robot de manutention, d'une presse et d'une table B qui sert au stockage des pièces forgées. Le robot inclut deux bras disposés perpendiculairement sur un même plan horizontal interdépendant d'un même axe de rotation et sans possibilité verticale de mobilité.
La Figure représente la disposition spatiale des éléments de la cellule. Le robot peut prendre une pièce brute de la table A et la déposer dans la presse à l'aide du bras 1. Il peut également prendre une pièce forgée de la presse et la déposer sur la table du stockage B à l'aide du bras 2. En bref, le robot peut faire deux mouvements de rotation. Le premier lui permet de passer de sa position initiale à sa position secondaire. Ce mouvement permet au robot de déposer une pièce brute dans la pression et probablement celui d'une pièce forgée sur

la table du stockage B. Le second lui permet de passer de sa position secondaire vers sa position initiale et de poursuivre le cycle de la rotation.

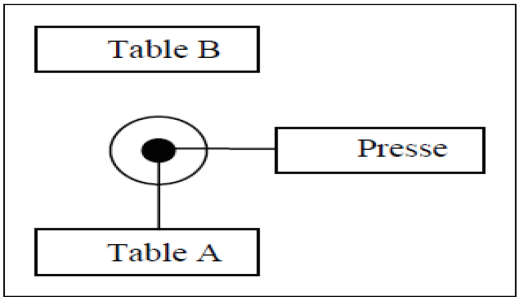


FIGURE 7.3 – Modèle ECATNet de la cellule de production

Les places
Ta : *table A ; ensemble, potentiellement vide, des pièces brutes.*
Tb : *table B ; ensemble, potentiellement vide, des pièces forgées.*
Ar1 : *bras 1 du robot ; au plus une pièce brute.*
Ar2 : *bras 2 du robot ; au plus une pièce forgée.*
Pr : *presse au plus une pièce brute ou une pièce forgée.*
PosI : *position initiale du robot ; elle est marquée "ok" si elle est la position courante du robot.*
PosS : *position secondaire du robot ; elle est marquée "ok" si elle est la position courante du robot.*
EA : *cette place a été ajoutée pour tester si les deux bras du robot sont vides.*

Les transitions
T1 : *prise d'une pièce brute par le bras 1 du robot.*
T2 : *prise d'une pièce forgée par le bras 2 du robot.*
D1 : *dépôt d'une pièce brute dans la presse.*
D2 : *dépôt d'une pièce forgée sur la table B.*
TS1, ***TS2*** : *rotation du robot de la position initiale vers la position secondaire.*
TI : *rotation du robot de la position secondaire vers la position initiale.*
F : *forge de la pièce brute introduite dans la presse.*
E : *dépôt d'une pièce brute sur la table A.*
R : *retire des pièces forgées de la table B.*
La Figure 7.4 représente le modèle ECATNet de la cellule de production. Le symbole ϕ est utilisé pour dénoter le multi-ensemble vide dans les inscriptions des arcs. Les multi-ensembles sur les arcs **r** dénotent "raw" (brute) et **f** dénotent "$forge$"(forgée).

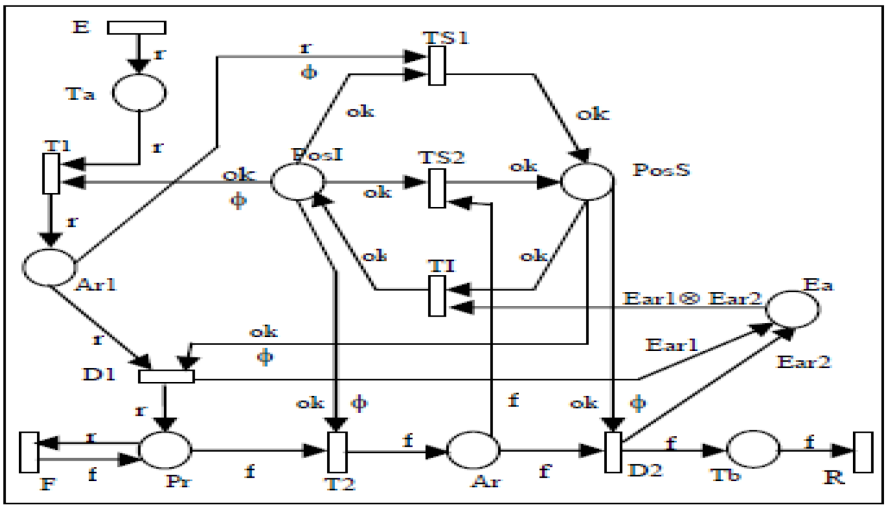


FIGURE 7.4 – Modèle ECATNet de la cellule de production

Bien qu'ils soient appropriés pour spécifier le comportement des systèmes complexes, les ECATNets ne modélisent pas l'aspect interactif entre composants principalement, lorsqu'il s'agit de spécifier le comportement de systèmes communicants tels que les services Web. Beaucoup de travaux étendent par divers aspects ce modèle. Nous retenons dans notre étude seulement deux modèles : le modèle Op-ECATNet qui prend en considération l'aspect «communicatif» des services Web et le modèle Mop-ECATNet qui est une abstraction de ce dernier ; ce modèle spécifie et contrôle à la fois dynamiquement le comportement des services Web dans une composition.

7.2.2 Extensions des Réseaux de Petri Orientées Service : Op-ECATNets, Mop-ECATNets

Modèle des Op-ECATNets
Le modèle Op-ECATNet est un modèle ECATNet ouvert qui modélise à la fois le comportement opérationnel du système mais aussi son processus d'interaction.

Définition 7.2. *Un Open-ECATNet est défini selon la structure suivante : Open ECATNet (E, IO, ST,M_0, M_f) tels que :*
E est un ECATNet.
IO est un ensemble d'interfaces d'entrées/sorties.
ST est un ensemble d'états.
M_0 et M_f sont respectivement le marquage initial et le marquage final du réseau.

Comportement d'un Op-ECATNet
Le comportement d'un Op-ECATNet est semblable au comportement d'un ECATNet étendu par deux opérations : «send» et «receive» qui assurent la communication entre

services. Ces deux opérations sont représentées par des places interfaces d'entrée/sortie «IO».

Exemple 7.2.

L'objectif de ce service Web composite est d'assurer la réservation d'un billet d'avion à des voyageurs.
Dans cette composition, l'utilisateur «user» invoque le service en envoyant une demande de reservation «Flight Request» en indiquant sa destination à l'«Agent» de service. Ce dernier en fonction des disponibilités renvoie une offre «offer» ou une notification d'indisponibilité «na». Si l'utilisateur reçoit une offre il peut l'accepter «ack» ou la rejeter «nack».
Si l'offre convient au client le vol est réservé et le billet est remis «Flight Ticket» ; dans le cas contraire l'utilisateur peut renouveler sa demande.

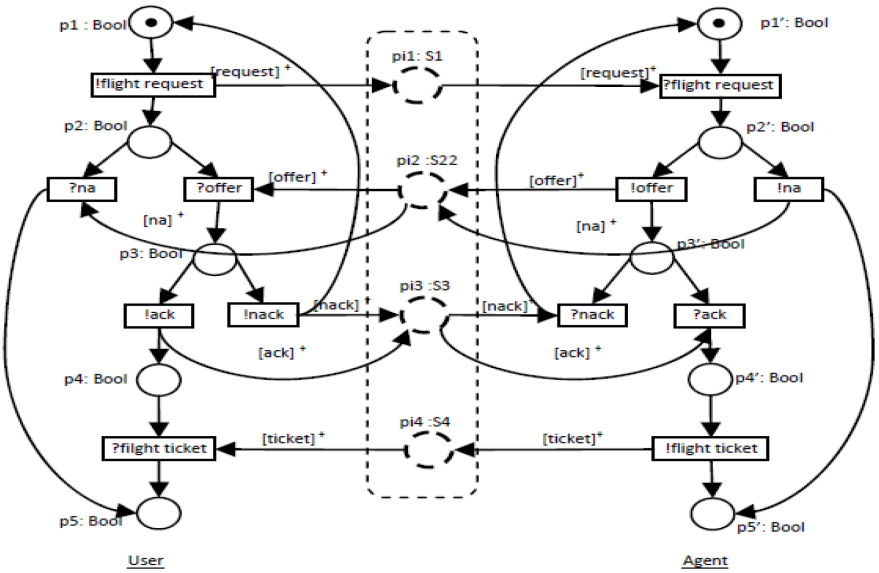


FIGURE 7.5 – Exemple d'un Op-ECATNet [LSB11]

Modèle des Mop-ECATNets
Le modèle des Méta-Open-ECATNets est une abstraction du modèle précédent. C'est l'extension d'un réseau à deux niveaux ; un niveau dit «bas» qui gère la configuration du système et qui est représenté par des Open-ECATNets et un second «méta» niveau qui gère la reconfiguration du système représenté par un ECATNet.

Un Mop-ECATNet est représenté formellement comme suit :

Définition 7.3. *Un Mop-ECATNet est un réseau à deux niveaux ME = <O, E, Q,τ> où :*

O : est un Op-ECATNet
E : est un ECATNet
Q : est le Méta-arc qui relie les deux niveaux du modèle (la place d'un ECATNet à la transition d'un Op-ECATNet)
τ : est le temps de tirage alloué à chaque transition.
Le modèle des Mop-ECATNets comprend à travers ses deux niveaux respectifs :

Trois types de places :
*Les méta-places **mp**, les places dynamiques **p** et les places interfaces **pi**.*
Deux types de transitions :
*Les méta-transitions **mt** et les transitions dynamiques **t***
Deux types d'arcs :
Les arcs ordinaires utilisés par les deux niveaux et les méta-arcs d'information qui relient les deux niveaux méta et dynamique.
Représentation graphique d'un Mop-ECATNet
La Figure ci-dessous illustre la structure générale d'un Mop-ECATNet.

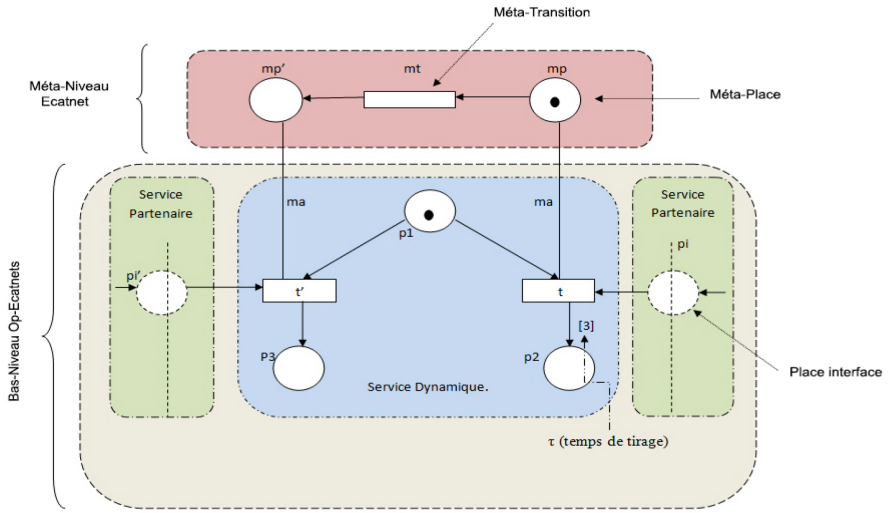


FIGURE 7.6 – Structure générale d'un Méta-Open-ECATNet

Cette structure est la représentation la plus simple d'un Mop-ECATNet ; elle est traduite ainsi :
P = {mp, mp'},
T = { mt },
Q = {(MT, t), (MT', t')}
MI'= {(1 0)},

IO = {pi, pi'},
ST= {p1, p2, p3},
M_0 = {(1 0 0)}, M_f = {(0 1 0), (0 0 1)}
et τ =(p3, 3).

Exemple 7.3.

Nous reprenons l'exemple de la réservation de vol afin de le modéliser par un Mop-ECATNet.

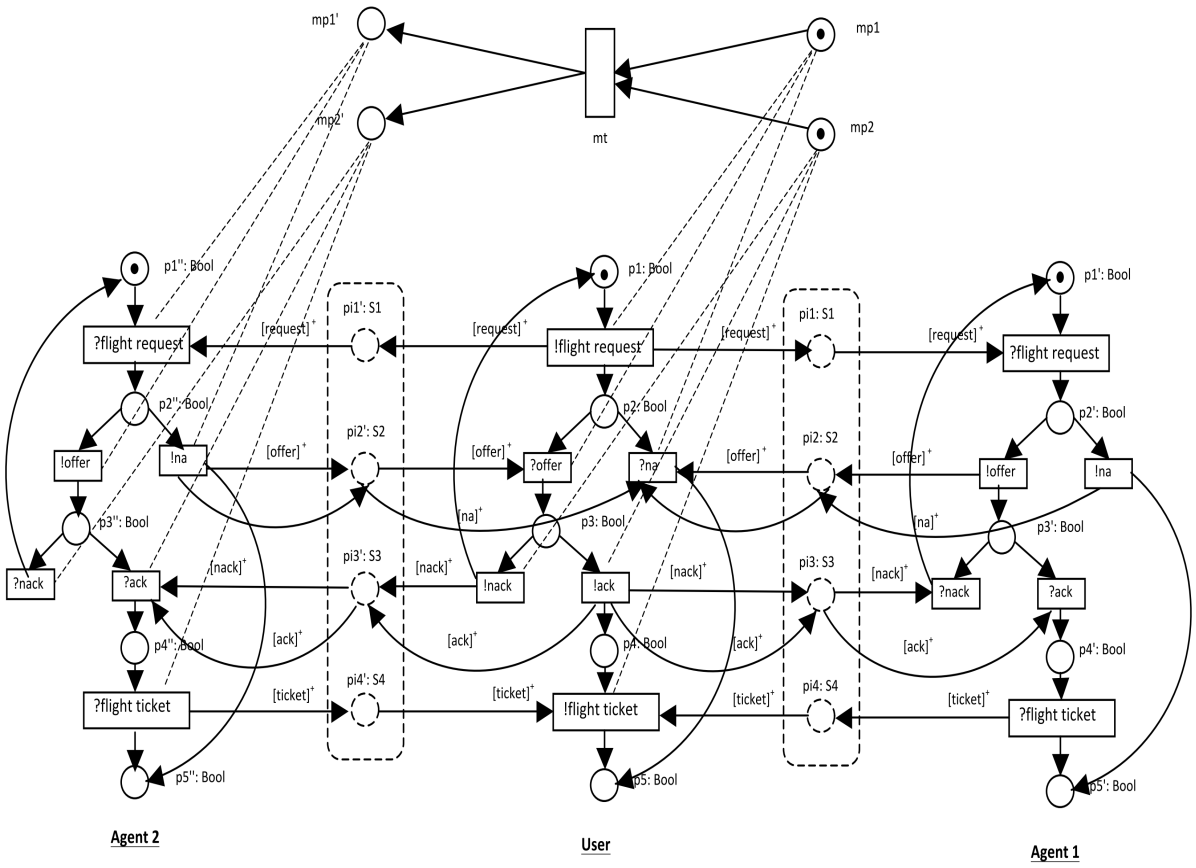


FIGURE 7.7 – Service réservation basé Mop-ECATNet

Nous rajoutons par conséquent, à sa spécification Op-ECATNet deux réseaux. Un ECATNet basique pour jouer le rôle de contrôleur (méta-niveau) et un Op-ECATNet «Agent2» comme alternative de remplacement à l'agent1 dans le cas où ce dernier échoue dans sa mission.
En effet, le méta-niveau contrôle les échanges «User»/«Agent» et reconfigure en temps

réel le système en cas de détection d'anomalies. Les informations circulant à travers les méta-arcs avisent continuellement le contrôleur de l'état du système. Lorsqu'une panne survient (par exemple déconnexion de l'agent1, réponse négative..) le méta-niveau réagit en sensibilisant la méta-place afin de trouver des alternatives de compensation. L'Agent2, dans cet exemple, est l'alternative équivalente de l'agent1 ; le processus d'exécution est donc immédiatement repris par ce dernier à partir du point où s'est arrêté le premier.
Dans cette section, nous avons présenté les réseaux de Petri algébriques de manière générale et de manière particulière une de leurs instances : le modèle des ECATNets ainsi que ses différentes extensions orientées service. A chaque formalisme nous avons associé une définition formelle et un exemple d'illustration.
Nous déduisons de cette étude que le modèle Mop-ECATNet est le plus approprié pour notre travail car il permet à la fois de spécifier les services Web de manière graphique à travers des Open-ECATNets mais aussi et surtout de contrôler dynamiquement leur interactions et composition. Dans ce qui suit nous présentons et motivons notre approche de modélisation basée RMop-ECATNet.

7.3 Exemple de Motivation

Afin de montrer l'efficacité de l'idée proposée, nous présentons dans cette section l'exemple de l'Agence de voyage. Cet exemple est assez commun et est fréquemment utilisé pour la spécification du comportement des services Web dans le processus de composition. Avant de présenter le modèle raffiné proposé, nous mettons en relief les limites du modèle Mop-ECATNet traditionnel à travers la modélisation de l'Agence de voyage.

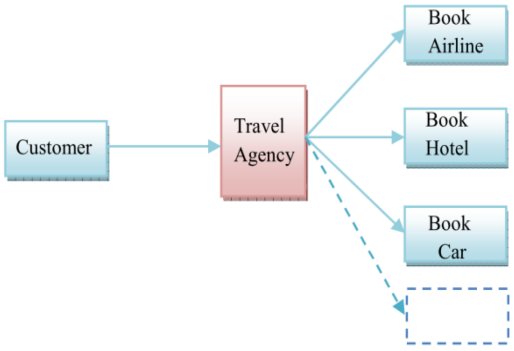


FIGURE 7.8 – Service Agence de voyage

Si l'on se réfère au modèle Mop-ECATNet pour spécifier cet exemple, l'Agence de voyage constitue le service dynamique, les services partenaires : la réservation d'un billet d'avion «BookAirline», la réservation d'une chambre d'hôtel «Book Hotel» et la location d'une voiture «Book Car».
La liste de services suivante constitue la base de la spécification graphique Mop-ECATNet

de l'Agence de voyage :

1. D : le service Web dynamique qui requiert la réservation d'un billet d'avion, d'une chambre d'hôtel et la location d'une voiture.
2. P1 : le service partenaire chargé de la réservation d'un billet d'avion.
3. P2 : un second service partenaire chargé de la réservation d'une chambre d'hôtel.
4. P3 : le troisième service partenaire chargé de la location d'une voiture pour assurer le transfert (aéroport-hôtel).

Une fois les services composés, certains problèmes peuvent survenir au cours de la phase d'exécution du service composite. A titre d'exemple, si un service partenaire ne parvient pas à réaliser sa mission pendant la composition, il est nécessaire d'en prévoir un ou de nouveaux services afin de le remplacer. Cette situation peut se produire dans notre exemple dans les situations suivantes :

— Lors de la réservation d'une chambre d'hôtel, un problème de connexion peut être observé (défaillance du service),
— Quand un service partenaire prend beaucoup de temps à émettre une réponse (Blocage)
— Quand un service partenaire renvoie une réponse qui ne correspond pas à la demande du client (réponse négative).

Pour faire face à ces situations, le modèle Mop-ECATNet est doté d'un mécanisme de contrôle géré par son Méta-niveau. Ce contrôle consiste à surveiller les anomalies qui peuvent survenir et à reconfigurer dynamiquement le système en remplaçant les services défaillants. A titre d'exemple, en cas d'échec du service partenaire P1, P1' prend la relève. P1' est donc le service alternatif de P1 et permet la réservation alternative d'un billet d'avion.

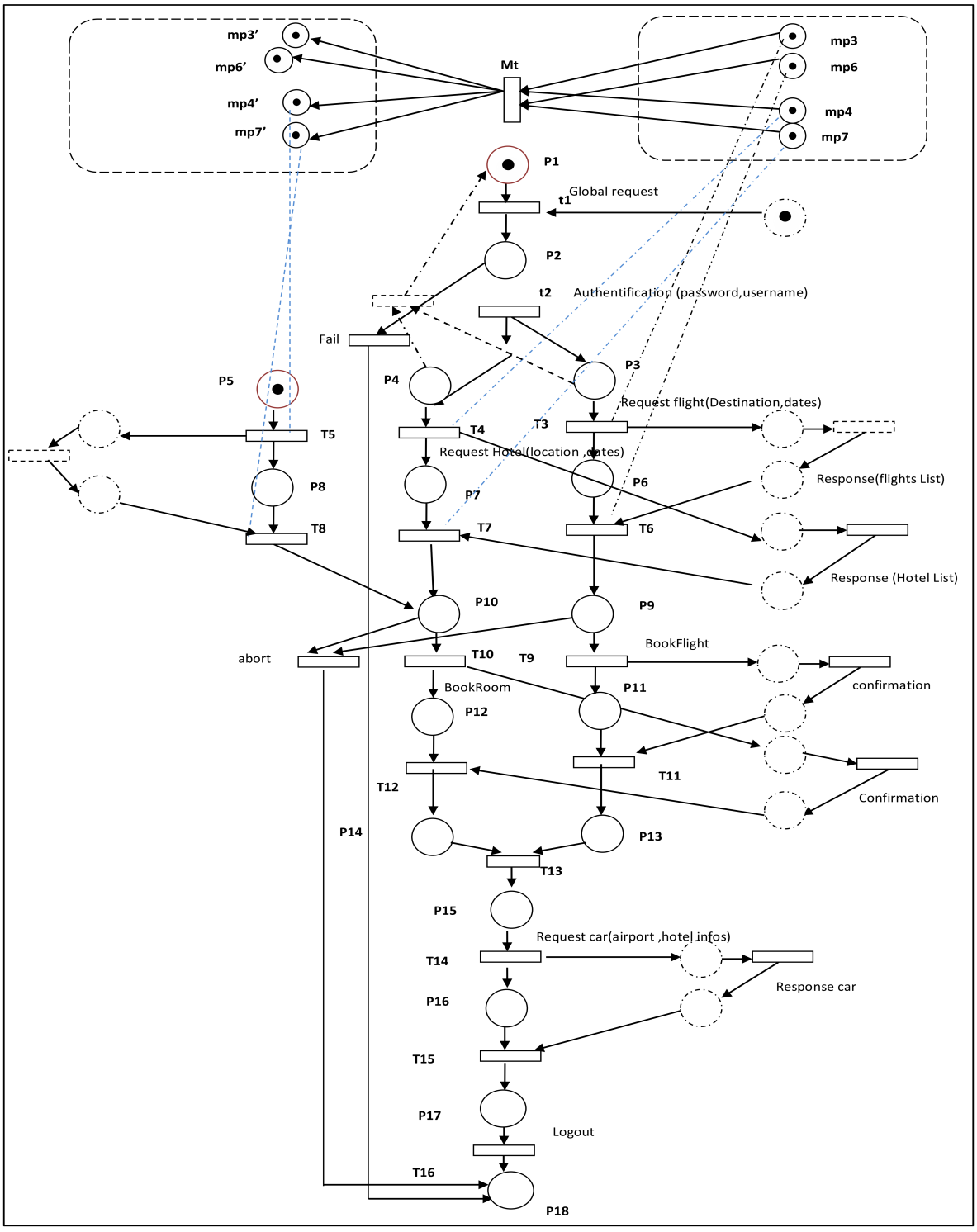


FIGURE 7.9 – Modélisation basée Mop-ECATNet traditionnel pour le service Agence de voyage

La Figure 7.9 représente la spécification Mop-ECATNet du service composite «Agence de voyage». Nous notons de prime abord la complexité du graphe. Cette complexité ne demeure pas que graphique. Nous résumons les limites du modèle Mop-ECATNet par ce qui suit :

1. Le graphe de composition des services Web est assez complexe du fait que les services (dynamique et partenaires) sont représentés par des Op-ECATNets. La distinction entre ces derniers est difficile compte tenu du nombre de places et de transitions présentes dans le graphe.
2. Les liens et dépendances qui peuvent exister entre services partenaires sont dissimulés dans le graphe, ce qui complique le choix quant aux stratégies de contrôle à employer et qui limite aussi grandement la réutilisation du modèle.
3. Les actions ou opérations effectuées par les services ne sont pas typées et sont contrôlées de manière machinale par le méta niveau.
4. La stratégie de contrôle adoptée par l'auteur estime que chaque transition dynamique doit être reliée par un méta-arc au méta-niveau. Si l'on suppose simplement qu'il existe des actions ou transactions non reconfigurables dans un service, le rôle de l'arc devient stérile et sa présence vaine ne fait qu'accroitre le niveau de complexité du graphe.
5. La reconfiguration se fait dynamiquement mais pas de manière ciblée ; en effet, à chaque défaillance décelée le niveau contrôle réagit en reconfigurant de manière globale le système. Ce type de reconfiguration est parfois inutile dans certaines situations.
6. Le système de contrôle du Mop-ECATNet n'est pas conçu pour être profitable. Mis à part la possibilité de réutilisation du service composite, le système ne prévoit pas de template pour spécifier des situations précises qui peuvent se répéter.
7. La spécification graphique des services basée Mop-ECATNet n'est pas automatisée ; il n'existe pas d'éditeur dédié à ce formalisme bien qu'héritant des concepts des réseaux de Petri qui ont de nombreux outils de conception à leur actif.
8. La spécification Maude des services relative au formalisme est spécifiée par l'auteur de manière ad-hoc, ce qui ne facilite pas d'une part l'écriture de cette dernière en terme temps et d'autre part, la tâche aux utilisateurs non familiers avec ce langage.

7.4 Principe de l'approche

Afin de faciliter la résorption des problèmes soulevés lors de l'exemple présenté précédemment, notre approche vise à agir à trois niveaux catégorisés ainsi :

Au niveau structurel

1. Raffinement du modèle Mop-ECATNet en RMop-ECATNet en introduisant de nouvelles notations génériques qui offrent une vue plus abstraite identifiant, en amont, tous les éléments clés de la composition des services Web.
2. Séparation entre les types de services partenaires et mise en évidence de leurs relations de dépendance.

Au niveau comportemental

1. Classification des services participants selon leur degré d'atomicité et de vitalité. (franchissement et reconfiguration du système).
2. Amélioration des stratégies de contrôle (par la suppression de tous les méta-arcs liés inutilement à des transitions dynamiques).
3. Introduction du concept de parallélisme au niveau Méta pour permettre une meilleure flexibilité lors de la reconfiguration du système.
4. Proposition de nouveaux patrons réutilisables pour traiter des cas de figure particuliers (par exemple : le problème de synchronisation).
5. Instauration de nouvelles règles pour chaque point de contrôle, par exemple, la reconfiguration basée sur le degré d'atomicité des services et la validation de la composition de services par rapport à la notion de vitalité (l'omission de certains services partenaires peut ne pas influencer la véracité du processus de composition).

Au niveau implémentation

1. Conception d'un méta-modèle générique pour représenter les Mop-ECATNet afin de le déployer dans plusieurs environnements en utilisant les notions de Méta-modélisation.
2. Extension de ce même méta-modèle afin de l'adapter aux nouvelles notions du RMop-ECATNet.
3. Transformation automatique de la spécification RMop-ECATNet en une spécification Maude «Transformation RMop2Maude».

7.5 Définition des RMop-ECATNets

7.5.1 Aspect structurel

Comme nous l'avons constaté précédemment, bien que le modèle Mop-ECATNet soit graphique, la croissance du nombre de participants dans une composition rend sa spécification plus complexe.
Afin de réduire cette complexité, nous proposons à travers le tableau suivant, une notation condensée du modèle pour spécifier la composition de services faisant la distinction entre les services utilisés au cours de leur processus d'exécution : le service composite dit «service

dynamique» et les services partenaires qui interagissent avec lui, tout en mettant en relief les dépendances qui peuvent les lier.

Concepts des Mop-ECATNets	*Concepts des Services Web*	*Notation RMop-ECATNets*	*Rôle*
Open-ECATNet D	ServiceDynamique**D**		Configuration du système (formulation des besoins du client)
Open-ECATNet P	Service Partenaire**P**		Configuration des services susceptibles de répondre favorablement aux besoins formulés par le client
Open-ECATNet P'	ServicePartenaire Alternatif **P'**		Configuration des services partenaires alternatifs
ECATNet E	Contrôleur de Service **C**		Service de contrôle et de la gestion de la reconfiguration du système
Meta-arc *Q*	Points de Contrôle		Arc d'information bidirectionnel pour la gestion du contrôle
Places interface Pi	Interfaces d'entrée/Sortie		Points de communication entre le service dynamique et ses partenaires

TABLE 7.1 – Notations du modèle RMop-ECATNet

Dans ce tableau chaque concept du modèle Mop-ECATNet est identifié en terme de service Web. Nous associons à chaque concept une notation graphique RMop-ECATNet et déterminons son rôle.
Par exemple, le méta-arc ***Q*** dans les Mop-ECATNets représente la fonction qui relie les deux niveaux (Meta et basique). En terme de service, cet élément spécifie les points de contrôle du service et est représenté graphiquement en RMop-ECATNet par une flèche verticale marquée aux extrémités par deux points qui le lient respectivement à chaque

niveau.
Nous définissons, par ailleurs, formellement, ce modèle comme suit :

Définition 7.4. *Un RMop-ECATNet RME_{WS} modélisant un service Web WS est un tuple <C, S, A> où :*

- *C est le service contrôleur*
- *S est le service composite, tel que : S = (D, R, P, P', CP, Pi) :*
  D est le service Web dynamique ayant un ensemble de requêtes R ;
  P et P' sont respectivement des ensembles finis de services partenaires et de services partenaires alternatifs ;
  CP est un ensemble de points de contrôle du service ;
  Pi est un ensemble fini de places interfaces.
- *A est l'ensemble des arcs reliant les deux niveaux RMop-ECATNet aux points de contrôle du service CP.*

Nous reprenons donc notre exemple de l'Agence de voyage en appliquant les nouvelles notations. La Figure 7.10 illustre le nouveau modèle.

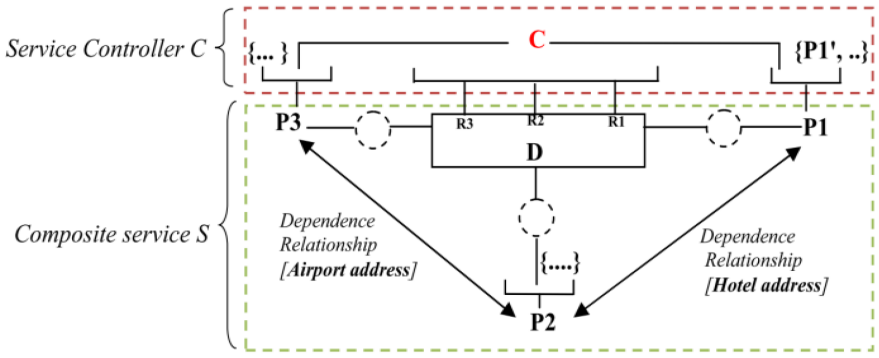


FIGURE 7.10 – Représentation abstraite du service «Agence de voyage» basée RMop-ECATNet

Nous retrouvons la liste établie lors de la présentation de l'exemple :
Le service dynamique D, les services partenaires P1, P2, P3 et le service P1' service alternatif à P1. Le service P1' dans les RMop-ECATNets est contenu dans une liste provisionnelle de services équivalents établie par le contrôleur pour assurer la reconfiguration dynamique du système en cas de pannes.
La liste de services identifiée en amont est représentée dans le modèle RMop-ECATNet

par un seul service S. S est le service Web composite spécifiant l'interaction de D avec P1, P2 et P3.
En conséquence à cette application, une séparation claire est naturellement déduite entre les différents niveaux de la structure du service Web composite. Le modèle est plus compact et nettement plus compréhensible et les liens de dépendances sont mis en évidence entre chaque paire de services partenaires.
Jusqu'à présent, nous avons montré comment utiliser les RMop-ECATNets comme support de spécification formelle pour définir l'aspect structurel de la composition des services Web complexes. Dans ce qui suit, nous définissons la façon d'optimiser le niveau contrôle du modèle Mop-ECATNet afin de veiller à ce que l'aspect comportemental de la configuration et la reconfiguration soit conforme au comportement attendu par les services dynamiques dans leur composition.

7.5.2 Aspect comportemental

Face au comportement imprévisible des services et aux conséquences relativement négatives qui peuvent en découler, nous procédons au raffinement du niveau méta du modèle Mop-ECATNet. Les propriétés transactionnelles et de vitalité d'un service sont des notions fondamentales qui reflètent de manière concrète son comportement. Cette extension intégrée au modèle RMop-ECATNet offre un support de contrôle dynamique et transactionnel intervenant de manière judicieuse dans la reconfiguration des services Web. Nous présentons cette extension sémantique graduellement à travers la définition du comportement d'un RMop-ECATNet. Afin de spécifier le comportement ou la sémantique d'un RMop-ECATNet il est impératif de connaître sa syntaxe concrète. Nous définissons dans ce qui suit la syntaxe concrète de ce modèle et déduisons par la suite sa sémantique.

Syntaxe d'un RMop-ECATNet

La syntaxe d'un RMop-ECATNet combine respectivement la syntaxe d'un ECATNet pour son méta-niveau et celle des Open-ECATNets pour le bas niveau. Ces syntaxes ont été précédemment introduites dans la section 2. Pour définir la syntaxe concrète d'un RMop-ECATNet nous reprenons la définition abstraite modélisant un service Web WS donnée dans la section 5.1.

Définition 7.5. *Un RME_{WS} est un tuple <C, S, A> où :*

Au Niveau Meta :
Le service contrôleur C est représenté par un ECATNet (Spec, MR)
Spec =(Σ, E) est une spécification algébrique tels que :
Σ=(S,Op) est une signature
S est l'ensemble des sortes
Op est l'ensemble des opérations sur S
E est l'ensemble des Σ-équations

MR est un uplet tel que MR = (MP,MT,σ,Cap, MIC,MDT,MCT,MTC) :
MP est un ensemble fini de Méta-places
MT est un ensemble fini de Méta-transitions (MP ⋂MT=∅)
σ est l'application de typage des Méta-places
Cap est la fonction de capacité des Méta-places
MIC est la condition d'entrée du méta-niveau (MetaInput Condition)
MDT est le nombre de jetons détruits au méta-niveau (MetaDestroyedTokens)
MCT est le nombre de jetons créés au méta-niveau (Meta CreatedTokens)
MTC est la condition de la méta-transition (MetaTransition Condition)

Au Niveau Service (bas niveau) :

*Le service composite **S** = (D, R, P, P', CP, Pi) où : D, P et P' sont représentés par des Open-ECATNets **O**(E, Pi, M_0, M_f) :*
*Les services P et P' communiquent avec le service dynamique **D** et le service contrôleur agit pricipalement sur le service dynamique. La syntaxe d'un service dynamique est donc donnée par un Open-ECATNet (E,Pi,M_0, M_f) où : E est un ECATNet (Spec, DR) :*
Spec=(Σ, E) est une spécification algébrique telle que :
(S,Op) est une signature
S est l'ensemble des sortes
Op est l'ensemble des opérations sur S
E est l'ensemble des Σ-équations

DR est un uplet tel que DR=(DP,DT,σ,Cap, DIC,DDT,DCT,DTC,DTr)
DP est un ensemble fini de places dynamiques
DT est un ensemble fini de transitions dynamiques (DP ⋂DT=∅)
σ est l'application de typage des places dynamiques
Cap est la fonction de capacité des places dynamiques
DIC est la condition d'entrée du niveau dynamique (DynamicInput Condition)
DDT est le nombre de jetons détruits au niveau dynamique (DynamicDestroyedTokens)
DCT est le nombre de jetons créés au niveau dynamique(DynamicCreatedTokens)
DTC est la condition de la transition dynamique (DynamicTransition Condition)
DTr est le type de transaction effectuée lors des échanges entre services.
*τ est le temps de tirage alloué à chaque transition **dynamique**.*
M_0 et M_f sont respectivement le marquage initial et le marquage final du RME.

A est un méta arc reliant les deux niveaux (une méta-place ECATNet à une transition dynamique d'un Op-ECATNet).

Représentation graphique
La Figure 7.11 présente la structure générale du modèle RMop-ECATNet

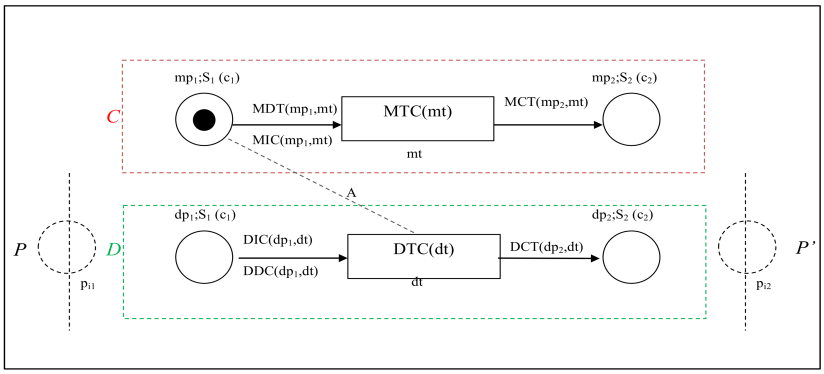


FIGURE 7.11 – Structure générale d'un RMop-ECATNet

Au niveau Méta :
Les méta places *mp1* et *mp2* sont respectivement de sortes *S1* et *S2* et de capacités *C1* et *C2*. La condition d'entrée méta *MTC* et la liste de jetons détruits *MDT* sont placées à gauche de l'entrée de la méta transition mt. Ce positionnement est relatif à l'application des règles de réécriture qui traduisent la sémantique de franchissement (terme gauche⇒terme droit). On retrouve la liste de jetons créés *MCT* après franchissement à droite de la transition mt.
Au niveau dynamique :
On retrouve les mêmes fonctions et conditions qu'au niveau méta *(dp1, dp2, S1, S2, C1, C2, DTC, DDT, DCT)* étendues par des places interfaces (*Pi*) qui permettent la communication entre services et le type de transaction (*DTr*).

Sémantique d'un RMop-ECATNet :
Le comportement d'un RMop-ECATNet peut être spécifié en précisant les règles de franchissement des transitions. Ces règles sont très importantes vu qu'elles décrivent le fonctionnement du schéma de contrôle du système spécifié.
Les règles de franchissement pour les ECATNets et Open-ECATNets ont déjà été introduites. Dans ce chapitre, nous nous axons sur l'aspect comportemental lors de la reconfiguration des services. En instaurant de nouvelles formes de contrôle, ces règles sont partiellement modifiées. Ainsi, avant de donner la sémantique de reconfiguration selon le modèle RMop-ECATNet, nous présentons d'abord les stratégies de contrôle déployées dans ce modèle.

Stratégies de contrôle selon le modèle RMop-ECATNet

Parallélisme
La reconfiguration dynamique basée Mop-ECATNet est un processus qui se déclenche

lors de la détection d'une anomalie par la couche contrôle. Le système de contrôle est doté d'un mécanisme de compensation afin de maintenir la stabilité de la composition des services. La sémantique d'une reconfiguration Mop-ECATNet se traduit par la recherche instantanée d'un service de remplacement équivalent au service défaillant pouvant poursuivre l'exécution et compléter les tâches inachevées.

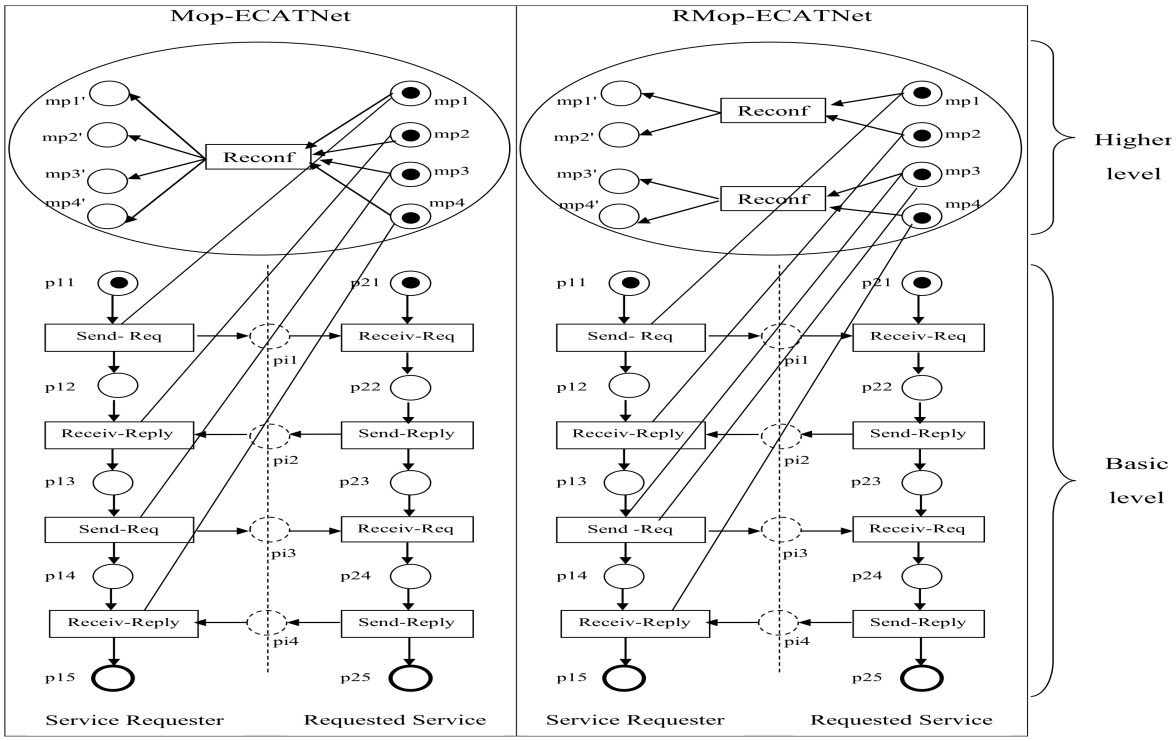


FIGURE 7.12 – Reconfiguration dynamique Mop-ECATNet vs RMop-ECATNet

Graphiquement les informations qui circulent entre les deux niveaux (dynamique et contrôle) sont modélisées par le biais des méta arcs où chaque méta place (place contrôle) est reliée à une transition dynamique. Toutes ces places de contrôle sont liées à une seule méta transition du méta-niveau (Figure 7.12-partie gauche) [LB14].
Ainsi, si l'on se réfère à l'exemple de l'Agence de voyage (Figure 7.9) nous remarquons que, par exemple, lors de l'échec du service P1 dans le processus de composition, le contrôleur se charge de le remplacer par le service P1'. Nous jugeons que cette représentation est inadéquate du moment que cela engendre automatiquement et inutilement la reconfiguration des autres services qui ne présentent aucune défection.
Nous proposons donc d'introduire le concept du parallélisme au niveau du méta-niveau. Ceci est illustré dans la (Figure 7.12-partie droite). Cette extension permet une reconfiguration à la fois ciblée et qui peut être partielle.

Propriétés transactionnelles
Les propriétés transactionnelles sont l'une des principales caractéristiques que possèdent les services Web. Dans notre étude, nous nous intéressons principalement à deux types d'entre elles : les transactions atomiques et les transactions compensables. Un service est dit atomique (avec la sémantique du "tout ou rien") quand il devient une composante indivisible et ses transactions (ensemble d'opérations) ne peuvent être soumises à la décomposition, contrairement à un service remplaçable qui offre une flexibilité permettant sa substitution par son équivalent. En effet, le service peut recourir à un système de compensation pour retourner à son état d'origine lorsqu'une erreur survient ou dans la situation où ses transactions ne peuvent être compensées. Afin de respecter ces propriétés les structures du modèle Mop-ECATNet ont été ajustées :

Reconfiguration dynamique avec support transactionnel
En prenant en compte les deux aspects évoqués ci-dessus, nous déduisons qu'il est impossible de remplacer une transaction atomique. Ces transactions sont des opérations indivisibles ne pouvant être substituées ni remplacées. Nous proposons donc de supprimer tous les méta-arcs liés à des transitions dynamiques supportant ce type de transaction. Les places contrôles sont par conséquent, elles aussi éliminées ; ce qui réduit considérablement la complexité du graphe. Nous proposons, par ailleurs, de typer les transitions dynamiques en terme des transactions qui les franchissent.

Proposition 7.1. *Une transition dynamique peut être contrôlée, si et seulement si les transactions qu'elle englobe sont compensables. (ie, MTC= true if DTr == compensable)*

Franchissabilité et Franchissement d'une Méta Transition
Le processus de reconfiguration dynamique est intrinsèquement lié au franchissement des méta-transitions. Le franchissement d'une transition et les conditions de ce franchissement sont formellement exprimés par des règles de la logique de réécriture où les axiomes sont en réalité des règles de réécriture conditionnelles décrivant les effets des transitions comme des types élémentaires de changement. Les règles de déduction nous permettent d'avoir des conclusions valides de la reconfiguration à partir des changements.
Nous proposons donc, dans ce qui suit, de spécifier les règles de franchissabilité et de franchissement relatives au méta-niveau.

Franchissabilité d'une méta-transition
Le franchissement d'une méta-transition est contraint par des conditions de franchissabilité (enabling).

Définition 7.6. *Une méta transition mt de C est franchissable pour un marquage M,si et seulement si, il existe au moins une affectation aff_{mt} de ctx(mt) telles que les conditions suivantes soient vérifiées simultanément :*
La méta-condition d'entrée MIC est satisfaite :
$\forall$ *mp*$\in$ *MP*$\forall$*(mp,mt)*$\in$ *dom(MIC) on a :*
- *M(mp)*$==\sigma$*(mp)*$\emptyset_\sigma$*(mp) si MIC(mp,mt)= empty*

— $\alpha \subseteq (mp)\ M(mp)$ *si* $[MIC(mp,mt)]_{aff_{mt}} =_E \alpha$
— $\neg(\alpha \subseteq (mp)\ M(mp))$ *si* $[MIC(mp,mt)]_{aff_{mt}} =_E \sim\alpha$

Remarque *: cette propriété est évaluée initialement à vraie pour toutes les places de contrôle.*

Les jetons à détruire sont disponibles :

— $\forall\ mp \in P\ \forall\ (mp,mt) \in dom(MDT)$ *on a :*
— $\alpha \subseteq \sigma(mp)\ M(mp)$ *si* $([MDT(mp,mt)]_{aff_{mt}E}\alpha\ \Lambda\ (MDT(mp,mt)) \neq \forall$

La condition de transition MTC est vraie :

$MTC(mt)_{\ aff_{mt}} =_E$ *true si* $mt \in dom(MTC)$ *et DTr == compensable.*

— ***Le non dépassement des capacités des places de sortie de mt :***
— $\forall mp \in dom(Cap)/M(mp) \oplus \sigma(mp)\ M_{cr}\ (mp) — \sigma(mp)\ M_{dt}(mp) / \leqslant Cap\ (mp)$

Franchissabilité concurrente (parallèle)

Un sous-ensemble de méta-transitions MT'⊆MT de C est franchissable de façon concurrente à un marquage M, si et seulement s'il existe au moins une famille d'affectations consistantes $(aff_{mt})_{mt}$∈MT' telles que toutes les conditions d'un franchissement simple soient vérifiées simultanément.

Franchissement d'une méta-transition

Le franchissement d'une meta-transition *mt* consiste à ôter des multi-ensembles de termes (précisés par *MDT*(mp,mt)) de chaque place en entrée de *mt* et à ajouter les multi-ensemble *MCT*(mp,mt) dans chaque place de sortie de *mt*.

La règle de réécriture relative à ce cas consiste à exprimer le retrait du terme <mp,MIC(mp,mt)> de l'état global et le remplacer par le terme <mp',MCT(mp',mt)>.

Cette situation correspond au cas usuel des réseaux de Petri : mt :<mp,MIC(mp,mt)>→<mp',MCT(mp',mt)>.

La règle décrivant le comportement parallèle s'exprime par :

Mt : (mp,MDT(mp,mt))→(mp',MCT(mp',mt) if (MIC(mp,mt)∩M(mp))→MIC(mp,mt).

Pour une meilleure assimilation des stratégies proposées, nous proposons à travers le tableau suivant un résumé de toutes règles de transformation établies jusqu'ici pour le modèle RMop-ECATNet.

Patrons de synchronisation IDP et DDP

Afin d'étendre davantage cette couche, nous proposons deux patrons réutilisables pouvant être appliqués directement lors du processus de synchronisation des services Web.

En effet, les services s'exécutent d'une manière parallèle et peuvent dépendre les uns des autres. Cette dépendance oblige parfois le contrôleur à intervenir afin de les synchroniser à un moment donné. Nous distinguons deux types de dépendance (directe et indirecte), les patrons IDP et DDP ont été mis en œuvre pour être appliqués pour traiter chacune des deux situations.

Le patron **IDP** est appliqué en cas de dépendance indirecte (synchronisation non-bloquante). Les deux services peuvent s'exécuter de manière parallèle, mais à un moment donné leur synchronisation est gérée par le méta-niveau afin de s'assurer que les services

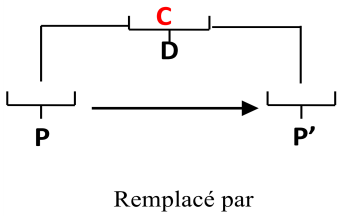
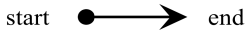
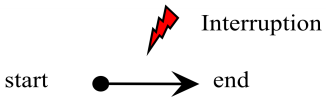
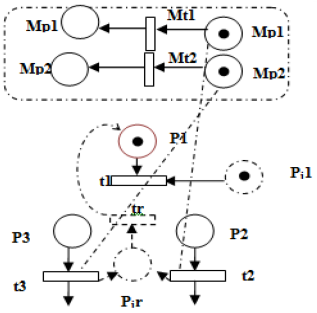
Comportement du service dynamique controlé	*Formalisation basée RMop-ECATNets*	*Notation*
Reconfigurationdu système	Règles de réécriture	
TransactionAtomique	Etat non contrôlé (pas de lien avec les transitions du bas niveau)	
TransactionCompensable	Etat contrôlé (meta-arclié aux transitions du bas niveau)	
Reconfiguration totale du système non nécessaire	Transitions parallèles Du méta niveau	

TABLE 7.2 – Extension de la couche contrôle du modèle Mop-ECATNet

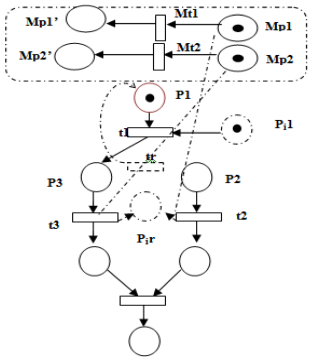
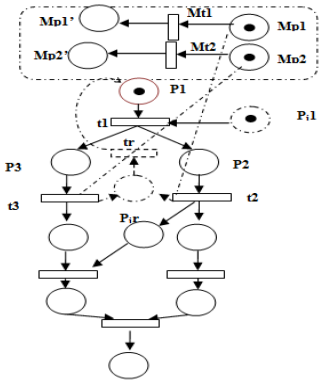
Comportement de la composition dynamique contrôlée	*Nom du Patron*	*Notation du patron dans le modèle RMop-ECATNet*
Le contrôleur Synchronise par rendez-vous / mutuellement les opérations effectuées par les services dans la composition dynamique.	Patron de Dépendance Indirecte (**IDP**)	
Le contrôleur utilise le concept de sémaphores pour gérer les processus de synchronisation des services.	Patron de Dépendance Directe (**DDP**)	

TABLE 7.3 – Patrons de contrôle RMopECATNet

achèvent leurs processus respectifs pour être synchronisés. D'autre part, pour éviter les blocages qui peuvent survenir en cas de dépendance directe entre les services, nous utilisons le concept de sémaphores (**DDP** Pattern). Les opérations liées aux services Web dépendants ne peuvent s'exécuter jusqu'à ce que le processus **1** atteigne un certain niveau dans l'exécution de ses opérations.
D'autres propriétés fonctionnelles importantes ont été omises par le modèle Mop-ECATNet. Parmi ces propriétés nous retrouvons :

— **Ré-exécutablité d'un service** Tout service est ré-exécutable. Certaines ré-exécutions sont conditionnées par un nombre fini de tentatives autorisées et un temps d'attente entre ces tentatives.
— **Vitalité d'un service** Un service est qualifié de **vital** si en son absence, le processus de composition est interrompu. A l'inverse, un service est dit **non-vital** si son

absence n'a pas de répercussion sur le processus de composition auquel il participe. Après avoir identifié les services partenaires, leurs alternatives et les relations qu'ils peuvent avoir entre eux ainsi que les différentes propriétés transactionnelles qui les qualifient. La spécification du comportement d'un service composite peut être réalisée en associant les stratégies suivantes :

Définition 7.7. *Étant donné $RME_{WS'}$ un RMop-ECATNet modélisant un service composite, les stratégies définissant son comportement sont données par $S_{RME-WS'}$, récursivement formé comme suit :*

- *Une stratégie $S \in S_{RME-WS'}$ est une simple règle de réécriture représentant une transaction atomique, ou*
- *Une séquence finie de règles de réécriture définissant une transaction compensable, ou*
- *Un ensemble de règles de réécriture exprimant des opérations de service exécutables,ou*
- *Un ensemble de règles de réécriture conditionnelles coordonnées représentant les patrons de synchronisation de dépendance directe ou indirecte, ou*
- *Une combinaison des règles de réécriture ci-dessus. Comme nous l'avons indiqué précédemment, un modèle de contrôle doit présenter toutes les branches et scénarios possibles explorés lors de l'exécution afin de pouvoir déployer les stratégies adéquates et parvenir à une reconfiguration du système fiable.*

Le Tableau 7.4 résume toutes les propriétés transactionnelles des services constituant le service composite Agence de Voyage. En se basant sur ces informations nous exprimons la logique de déploiement des stratégies de contrôle proposées pour le service composite «Agence de voyage» :
Atomicité : l'authentification est qualifiée par une transaction atomique du moment où deux réponses sont renvoyées à l'issue de l'exécution. Soit un accès à la plateforme de réservation ou un rejet. Notons aussi que les confirmations de réservation ou de paiement sont des transactions atomiques du moment que leurs réponses sont naturellement des notifications de confirmation.
Compensabilité (Reconfiguration) : plusieurs transactions sont compensables dans la composition de l'agence de voyage. En outre, si par exemple, le service de la réservation de la chambre d'hôtel P échoue, il est possible de le remplacer par un autre service partenaire P'. Ainsi, le service P est dit compensable et remplaçable et P' est son alternative de remplacement.
Ré-exéutabilité : tous les services de l'agence ainsi que leurs transactions respectives sont, par défaut, ré-exécutables. Une particularité est distinguée, par exemple, pour le processus d'authentification : la réexécution de ce processus est restreinte à trois tentatives.
Vitalité : les trois services partenaires constituant l'Agence de voyage ne sont pas d'égale

Transaction	*Type de transaction*	*Service Partenaire*
Service Réservation d'hôtel		
Authentification	Atomique	Accès ou rejet
(Identifiant, mot de passe) ou abandon	Ré-exécutable [3 fois]	
Requête (indication du lieu d'hébergement)	Compensable	Réponse (liste des chambres disponibles) ou (Indisponibilité, blocage (temps de réponse trop long),défaillance du service)
Sélection d'une chambre parmi celles proposées ou abandon	Atomique	Confirmation du choix
Service Réservation Billet d'avion		
Authentification	Atomique	Accès ou rejet
(Identifiant, mot	Ré-exécutable[3 fois]	
de passe) ou abandon		
Requête(Indication d'itinéraire)	Compensable	Réponse (liste des vols disponibles) ou (Indisponibilité, blocage (temps de réponse trop long), défaillance du service)
Sélection d'un vol disponible ou abandon	Atomique	Confirmation du choix
Service location de voiture		
Authentification	Atomique	Accès ou rejet
(Identifiant, mot de passe) ou abandon	Ré-exécutable [3 fois]	
Requête (Indication de l'itinéraire)	Compensable	Réponse (liste des vols disponibles) ou (Indisponibilité, blocage (temps de réponse trop long), défaillance du service)
Sélection d'une voiture ou abandonner	Atomique	Confirmation du choix

TABLE 7.4 – Propriétés transactionnelles du service Agence de voyage

importance. En effet, nous notons que la composition des services peut être valide même si la voiture n'est pas louée. le service de location de voiture est qualifié donc de **non-vital** pour cette composition. Cependant, la composition ne peut être valide si la réservation de la chambre d'hôtel échoue. Dans ce cas le service partenaire de la réservation de la chambre d'hôtel est **vital**.

Synchronisation : les relations de dépendance entre services peuvent être déduites du modèle abstrait (voir Figure 7.10). Par exemple, pour assurer le transfert du client de l'aéroport à l'hôtel, les adresses de l'hôtel et de l'aéroport sont nécessaires pour invoquer le service de location de voiture. Le Méta-niveau doit donc gérer la synchronisation des deux services au point t13 (Figure 7.9) et veiller à ce que leurs processus respectifs soient correctement terminés pour effectuer cette synchronisation. Le patron IDP peut être appliqué pour traiter ce cas vu que c'est une synchronisation non-bloquante.

Dans d'autres circonstances, le service de réservation de chambre d'hôtel peut avoir une dépendance directe du service de réservation du billet d'avion ; par exemple, le billet d'avion peut être réservé sans contrainte alors que la date d'arrivée est nécessaire pour réserver la chambre d'hôtel. Il est de la responsabilité du contrôleur du service composite d'interrompre le processus de service de la réservation de la chambre l'hôtel jusqu'à ce que la date d'arrivée soit connue. Le patron DDP est approprié pour traiter ce cas.

Nous jugeons, à travers cette étude, que le raffinement du modèle Mop-ECATNet en RMop-ECATNet était essentiel et a été bénéfique tant sur le plan structurel que comportemental.

7.6 Conclusion

Dans cette contribution, nous avons défini le modèle RMop-ECATNet pour la spécification formelle du processus de composition en terme des propriétés transactionnelles des services Web. Ce modèle profite des puissances théoriques du formalisme Mop-ECATNet et l'étend sur différents plans.

En effet, pour aboutir au modèle RMop-ECATNet nous avons, au niveau structurel, proposé de nouvelles structures (plus compactes) pour réduire la complexité du graphe de composition. Au niveau comportemental, nous avons étendu la couche de contrôle du modèle en définissant de nouvelles stratégies de contrôle et en mettant en œuvre des patrons réutilisables qui rendent la composition récursive. Enfin, nous avons particulièrement consacré le chapitre 8 à l'implémentation du modèle sous Maude afin de mieux souligner son rôle.

Chapitre 8

Implémentation de RMop-ECATNet dans Maude

8.1 Introduction

Dans ce chapitre, nous contribuons à l'implémentation du modèle RMop-ECATNet proposé dans le chapitre précédent dans Maude. Cette implémentation est réalisée de manière totalement automatique et transparente au moyen des techniques de méta-modélisation et de méta-transformation relatives à l'approche MDA.
L'intérêt de l'utilisation de l'approche MDA revient à son pouvoir de transformer une application en une autre en se basant uniquement sur des modèles assurant ainsi un niveau d'abstraction élevé, totalement indépendant des contraintes d'implémentation. Ce chapitre est structuré comme suit :
Nous faisons d'abord, un rappel sur l'architecture dirigée par les modèles et présentons à travers cette dernière les différents types de modèles existants pour représenter les systèmes informatiques, à savoir : les notions de modèle, méta-modèle et méta-méta-modèle. Nous donnons également, une vue d'ensemble sur les différentes techniques de transformation de modèles ainsi que les outils qui leurs sont associés. Nous illustrons l'intérêt de notre approche par la transformation RMop2Maude du service Agence de voyage.

8.2 Pré-requis

Dans cette section nous faisons un bref rappel sur l'ingénierie des modèles et plus particulièrement sur son support technologique : l'approche MDA.

8.2.1 Ingénierie Dirigée par les Modèles

L'IDM (en anglais Model Driven Engineering) est une discipline particulière du génie logiciel. Elle vise, non seulement, à favoriser un «génie» logiciel plus proche des métiers

en autorisant une appréhension des applications selon différents points de vues (modèles) exprimés séparément mais intègre également comme fondamentales la composition et la mise en cohérence de ces perspectives. De plus elle se veut productive en automatisant la prise en charge des outils relatifs à la validation des modèles, les transformations et les générations de code [FEBF06].
Un processus IDM peut être subdivisé en deux parties distinctes : la première est dédiée à la spécification de modèles à l'aide de formalismes de modélisation définis eux-mêmes par des méta-modèles ; la seconde, à leur transformation en d'autres modèles, en code, ou encore en d'autres artefacts grâce à des langages et outils de transformation.
Enfin, l'intérêt essentiel de raisonner exclusivement en terme de modèles implique la non-considération du code comme élément clé du processus de développement mais comme conséquence d'une ou plusieurs transformations de modèles. Dans ce qui suit nous présentons le support de base de l'ingénierie des modèles à savoir : le modèle et ses différentes abstractions.
Dans l'IDM tout est modèle et tout raisonnement est bâti sur la notion de modèle.
Modèle
Le modèle est la représentation abstraite d'une entité active. En d'autres terme, le modèle était initialement considéré comme une représentation figée d'un système actif. Grâce aux progrès de l'IDM, le modèle est passé du stade «contemplatif» au stade «productif» au sein d'outils spécifiques [BB02]. En effet, le modèle constitue désormais le noyau actif du processus de développement. L'une des conséquences les plus révolutionnaires de cette ascension est la génération automatique de code à partir de modèles.
Méta-Modèle
Tout formalisme ou langage de modélisation dans lequel est exprimé un modèle est décrit par un méta-modèle. Le méta-modèle est donc le langage de description des modèles, il contient par conséquent tous les concepts nécessaires à la création de modèles, bien sûr, dans un contexte particulier. Plus précisément, le rôle d'un méta-modèle est de définir, au minimum, la syntaxe abstraite d'un formalisme en définissant des concepts et des relations entre ces derniers. Par exemple, le méta-modèle d'un langage de programmation représente sa grammaire.
De nombreux langages de modélisation sont nés d'un processus IDM et donc décrits par des méta-modèles. Il s'agit de méta-modèles représentant des Domain Specific Languages [VDKV+00] (DSL) ou bien des langages plus universels couvrant un large spectre d'applications tel que le méta-modèle d'UML.
Un méta-modèle est, donc lui aussi, un modèle décrit à son tour par un autre formalisme : le méta-méta-modèle. **Méta-méta-modèle**
Un méta-méta-modèle sert à décrire un méta-formalisme. En IDM, on retrouve des méta-formalismes reposant sur des concepts différents (objet, XML, entités-relations...). Par exemple, un des méta- formalismes les plus utilisés en IDM provient de MDA : il s'agit de MOF [Béz04].
Afin de ne pas recourir à la définition d'une infinité de méta-modèles, le méta-méta-modèle a été conçu de telle sorte à être en mesure de s'auto-décrire.

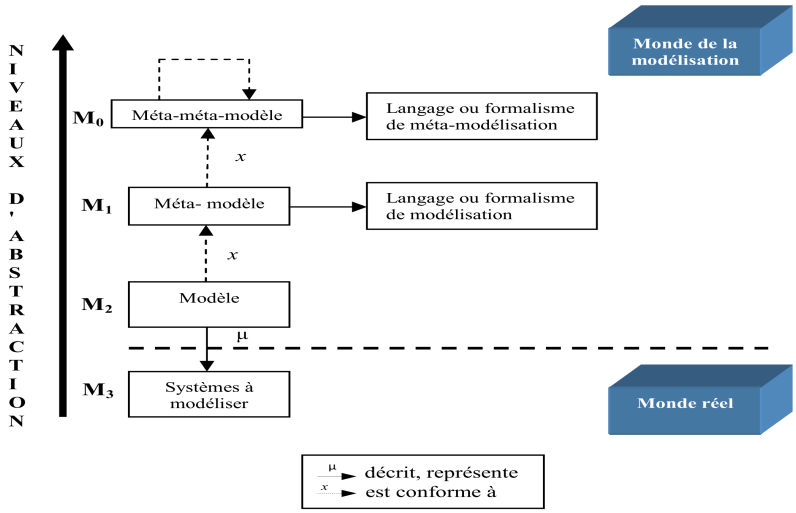


FIGURE 8.1 – Niveaux méta de l'IDM

Les modèles présentés ci-dessus sont hiérarchisés dans l'IDM (Figure 8.1) en quatre niveaux distincts dits «niveaux-méta» et ce, relativement à leur niveau d'abstraction (par ordre croissant) où :

— M0 est le niveau qui correspond au monde réel,

— M1 est le niveau des modèles,

— M2 est le niveau des méta-modèles,

— M3 est le niveau des méta-méta-modèles.

8.2.2 Approche MDA

L'approche MDA (Model Driven Architecture) ou encore (en français, Architecture Dirigée par les Modèles) définie par l'OMG en 2000 est certainement l'incarnation la plus célèbre de l'IDM. MDA représente l'espace technologique de l'IDM. MDA inclut la définition de plusieurs standards, notamment UML et MOF.
La distinction de la MDA peut être mesurée par rapport aux avantages qu'elle présente. En effet, elle assure la pérennisation de «la logique métier de l'entreprise» grâce à l'élaboration de modèles. La productivité de cette logique métier grâce à l'automatisation des transformations de modèles et la prise en compte des plates-formes d'exécution grâce à l'intégration de celles-ci dans les transformations de modèles.

8.2.2.1 Pyramide MDA

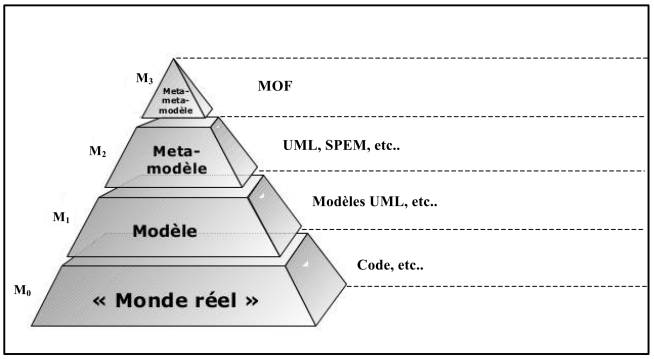


FIGURE 8.2 – Pyramide MDA

La Figure 8.2 illustre la pyramide MDA à quatre niveaux d'abstraction qui représente les concepts spécifiques à l'approche MDA de l'OMG : c'est une variante particulière de la Figure 8.1.
Utiliser une forme pyramidale permet de donner une indication sur le nombre d'entités pouvant exister sur chaque niveau : le plus bas niveau, M0, comporte une infinité de systèmes ; certains d'entre eux ont fait l'objet de processus de modélisation et sont représentés sous forme de modèles (M1). Ces modèles sont spécifiés au moyen de méta-modèles généraux situés en (M2) pouvant être des standards de l'OMG (UML en est l'exemple le plus connu) ou créés pour l'occasion (méta-modèles représentant des Domain Specific Modeling Languages ou DSMLs). Enfin, le standard le plus important de l'approche MDA est certainement le méta-formalisme MOF : il se situe de par sa nature au sommet de la pyramide : le niveau M3.
La totalité des méta-modèles standardisés par l'OMG se conforment à MOF et tout méta-modèle créé dans une optique MDA doit s'y conformer aussi. Ce méta-formalisme est auto-descriptif en vertu des principes de l'IDM. Il se subdivise en deux sous-formalismes EMOF (Essential MOF) et CMOF (Complete MOF). Le premier est une restriction, un sous-ensemble, du second. MOF contient tous les concepts nécessaires à la description de la syntaxe abstraite de méta-modèles au niveau M2, notamment les notions de classes, d'attributs, d'héritages et d'associations. On retrouve ces concepts dans la spécification du méta-modèle d'UML. Même si elles ne sont pas indiquées, les relations de conformation et de représentation entre les niveaux sont analogues à celles de la Figure 8.1.

8.2.2.2 Modèles MDA

Il existe trois types de modèles au centre de l'architecture MDA liés entre eux par des transformations de modèles :

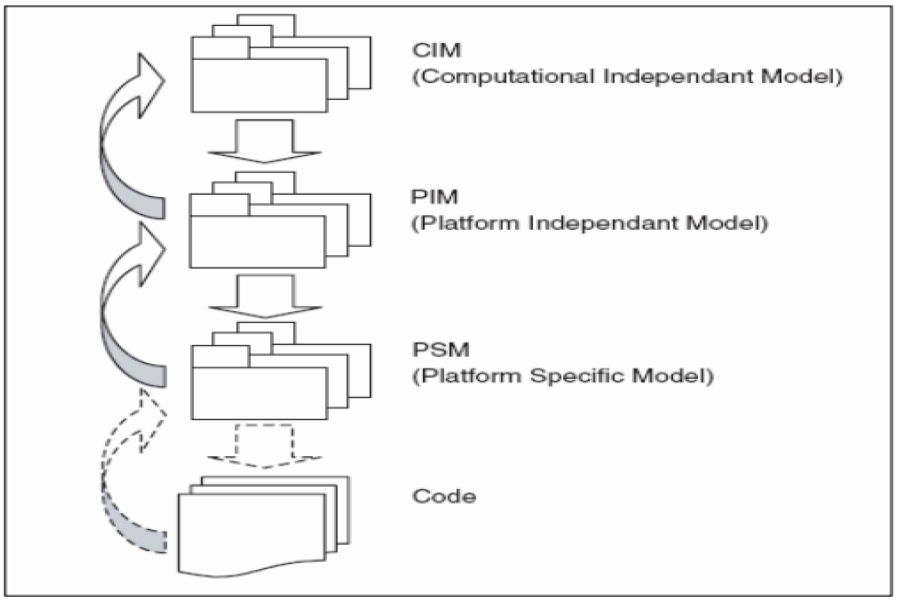


FIGURE 8.3 – Modèles MDA

— **CIM**(*Computation Independent Model*) : est un modèle indépendant de tout ce qui touche à l'implémentation et même à l'informatique. Les CIM sont des modèles métier (de haut niveau) qui n'ont pas été raffinés (par exemple, des diagrammes de cas d'utilisation UML) ;
— **PIM** (*Platform Independent Model*) : est un modèle de conception indépendant des plateformes et des technologies (ce qui lui confère une grande durée de vie). Il représente la logique métier à un niveau plus bas que les CIM. Un PIM est spécifié, usuellement, au moyen de diagrammes de classes UML enrichis par des contraintes OCL ;
— **PSM** (*Platform Specific Model*) : est le modèle le plus raffiné en MDA. Il est utilisé pour générer le code qui sera exécuté sur une ou plusieurs plateformes spécifiques. Souvent, on associe la notion de PSM à la notion de code (le code est assimilé à un PSM particulier).

8.2.2.3 Transformation de modèles

Un modèle ne peut passer du stade contemplatif au stade productif qu'à condition qu'on lui applique une ou plusieurs transformations : la transformation de modèles est un élément important de l'IDM [SK03].
En effet, une transformation est la génération automatique d'un modèle cible à partir d'un modèle source, suivant une définition de transformation. Une définition de transformation est un assortiment de règles de transformation qui décrivent ensemble comment un modèle du langage source peut être transformé en un modèle dans le langage cible. Une règle de transformation est la description du processus de transformation d'un ou plusieurs

concepts du langage source qui peuvent être transformés en un ou plusieurs concepts du langage cible.
Dans un contexte IDM, on qualifie de transformation de modèles tout programme dont les entrées et les sorties sont des modèles. On parle également de «modèle source» et de «modèle cible». Selon qu'une transformation produise en sortie un modèle ou du code, elle sera qualifiée de «Model To Model» («M2M») ou «Model To Text» («M2T»).

Mécanisme de transformation

Une transformation se compose de règles : chacune d'elles établit une correspondance entre un élément du modèle source et un élément concept du modèle cible. La somme de ces correspondances constitue la définition de la transformation. Chaque règle de transformation comporte un membre gauche et un membre droit que l'on abrège souvent en «LHS» (Left-Hand Side) et «RHS» (Right-Hand Side). Schématiquement, le LHS accède au modèle source tandis que le RHS agit dans le modèle cible.

Transformation MDA

Les transformations de modèles préconisées par MDA sont essentiellement les transformations CIM vers PIM et PIM vers PSM. La génération de code à partir des PSM n'est, quant à elle, pas considérée comme une transformation de modèle à part entière.
MDA envisage aussi les transformations inverses : code vers PSM, PSM vers PIM et PIM vers CIM comme l'exprime la Figure 8.4.

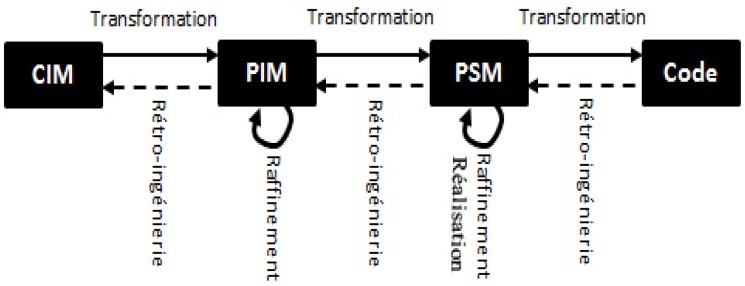


FIGURE 8.4 – Relation entre les niveaux de modèles MDA

— **CIM** vers **PIM** Les modèles CIM expriment les besoins des utilisateurs. Cette étape consiste à construire partiellement des modèles PIM à partir des CIM. Le but est de retranscrire les informations contenues dans les CIM vers les modèles PIM. Ceci permet de s'assurer que les besoins de l'utilisateur sont véhiculés et respectés tout au long du processus MDA.
— **PIM** vers **PIM** (**raffinement**) : Cette transformation constitue un enrichissement et une spécialisation du modèle en y apportant des informations indépendantes des spécificités d'une technologie. La description d'un modèle de répartition, de persistance des données ou de composants peut être vue comme un raffinement.
— **PIM** vers **PSM** (projection) : C'est la traduction du modèle générique vers une plateforme d'exécution.

— **PSM** vers **PSM** (réalisation) : C'est la mise en œuvre concrète d'un modèle générique sur une plate-forme d'exécution. Elle consiste en l'ensemble des phases qui mènent à un logiciel exécutable tel que la génération de code source, la compilation, le déploiement, l'instanciation et l'initialisation des composants logiciels.
— **PSM** vers **PIM** (Transformation inverse) : Cette transformation permet l'élaboration du modèle générique à partir de l'implémentation existante d'un logiciel. En théorie, cette transformation est censée fournir un modèle générique décrivant l'application à partir d'une base de code accessible. En pratique, il est très complexe d'automatiser entièrement ce processus pour tout modèle PSM. On peut citer à titre d'exemple les transformations suivantes :
— Transformation d'un automate à états finis non déterministe (AFN) en un automate à états finis déterministe (AFD).
— Transformation d'un diagramme UML en un réseau de Petri.
— Transformation d'un processus métier en un réseau de Petri.

Types de transformation

Une transformation de modèles, outre le fait qu'elle puisse intervenir entre deux méta-modèles distincts, elle peut également mettre en jeu des niveaux d'abstraction différents. Si c'est le cas, on parlera de transformation verticale. En l'absence de changement de niveau d'abstraction, on qualifiera au contraire la transformation*d'horizontale.*

Cette notion de niveaux d'abstraction ne doit surtout pas être confondue avec celle des «niveaux méta» de l'IDM : une transformation de modèles verticale est toujours définie en M2 et est exécutée sur un modèle de M1 pour produire un autre modèle de M1 !. Mais deux modèles situés au niveau M1 et impliqués dans une transformation peuvent appartenir à des niveaux d'abstraction très différents. D'un point de vue de processus de développement. Par exemple, le modèle source est un modèle générique, le modèle de destination est un modèle de code spécifique à la plateforme Java : la génération de code est une transformation *verticale.*

Transformation Obliques : une transformation oblique combine une transformation horizontale et une autre verticale. Ce type de transformation est notamment utilisé par les compilateurs qui effectuent des optimisations du code source avant de générer le code exécutable.

De manière orthogonale à cette catégorisation, on peut distinguer les transformations de type modèle vers code (exogène) et celles de type modèle vers modèle (endogène) même si les premières peuvent être considérées comme un cas particulier des dernières (il suffit de fournir un méta-modèle pour le langage de programmation cible).

Transformation endogène

Une transformation endogène est une transformation entre les modèles exprimés dans le même langage, par exemple transformation depuis UML vers UML.

Inversement, une transformation dans laquelle le modèle source et le modèle cible possèdent le même méta-modèle est qualifiée d'*endogène.* Les règles de transformation sont définies sur un unique méta-modèle. La Figure 8.5 illustre une transformation endogène : un modèle *M1* est transformé en un modèle *M2* et qui sont tous les deux conformes au méta-modèle

MM.

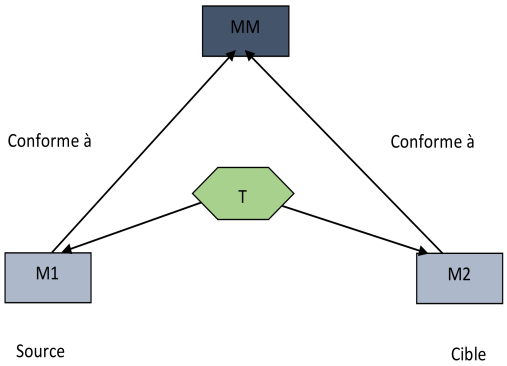


FIGURE 8.5 – Transformation endogène

On a recours à des transformations endogènes dans le cadre, par exemple, d'une optimisation de modèles, ou bien d'une simplification ou encore de *refactoring*.

Transformation exogène

On qualifie une transformation de modèles d'*exogène* lorsque les méta-modèles source et cible sont différents. En d'autres termes, lorsque l'on veut produire un modèle cible exprimé dans un formalisme différent de celui dans lequel est exprimé le modèle source. Par exemple, la Figure 8.6 montre une transformation exogène.

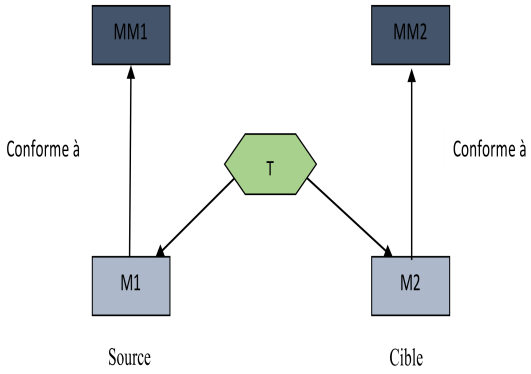


FIGURE 8.6 – Transformation exogène

On utilise des transformations exogènes pour toute transformation de modèle impliquant des méta-modèles différents ; l'exemple le plus significatif étant sans aucun doute la génération de code. On fait également appel aux transformations exogènes pour effectuer des migrations de modèles (depuis une plateforme vers une autre) tout en restant au même niveau d'abstraction ou encore dans des opérations de *reverse engineering* (rétro-ingénierie ou rétro-conception).

La Figure 7.7 récapitule les différents types de transformation MDA et leur rôles respectifs.

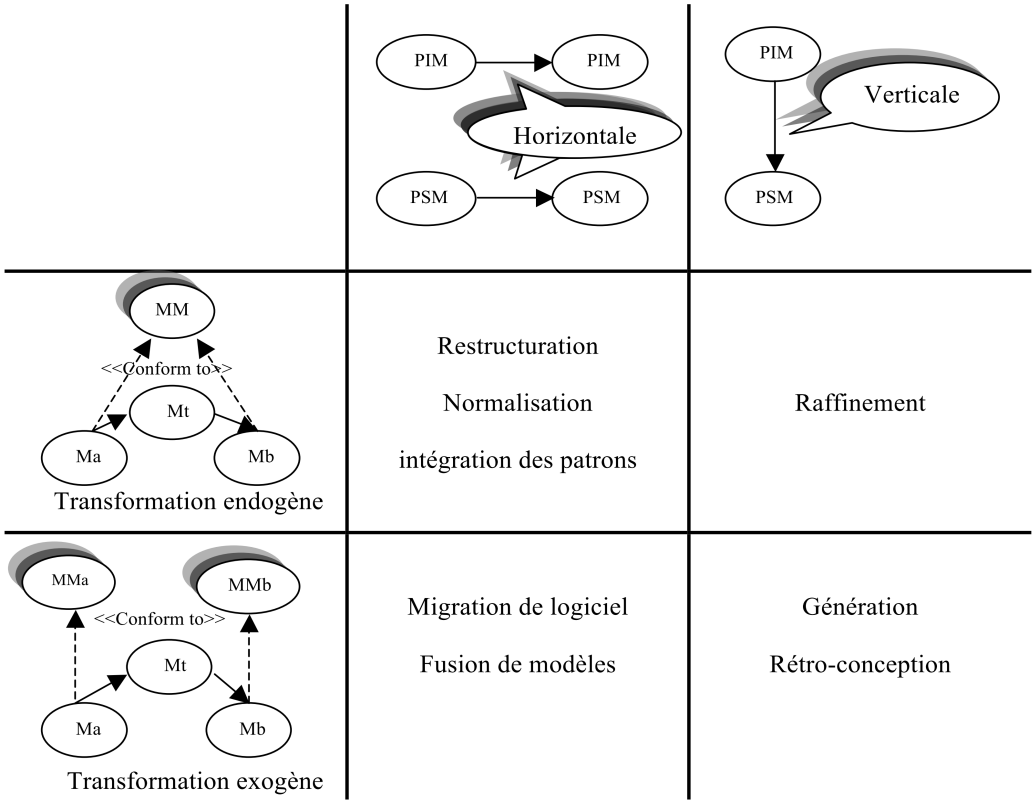


FIGURE 8.7 – Taxonomie des types de transformation MDA

A travers ce rappel nous déduisons que l'approche **MDA** présente plusieurs avantages dont :

- La liberté de travail qu'elle offre par la possibilité de choisir notre propre modèle.
- Le temps de réalisation relativement court pour la réalisation du modèle.
- La possibilité de réutiliser le modèle en cas de modification par le développeur ou l'utilisateur.
- La vision abstraite que l'on peut obtenir du problème à résoudre.
- Le passage d'une programmation basique à une programmation évoluée (méta) grâce à la transformation de modèles.

Enfin, ce chapitre nous a permis de comprendre la philosophie de l'IDM, à savoir la place qu'occupe le modèle dans cette discipline, les différentes approches de transformation de modèles existantes, les différents méta-formalismes et méta-modèles que l'on peut utiliser.Le standard MDA de l'OMG a occupé une place particulière dans ce chapitre, il est une réincarnation particulière de l'IDM qui bénéficie des méta-modèles et méta-formalismes déjà standardisés par l'OMG.

8.3 Transformation RMop2Maude

Afin de vérifier formellement et exécuter les spécifications des systèmes à base de services Web, nous réalisons une transformation du modèle RMop-ECATNet en une spécification Maude exécutable utilisant une approche de Méta-modélisation. Cette section a pour but de montrer comment générer automatiquement une spécification Maude (modèle cible) à partir d'un modèle RMop-ECATNet (modèle source) par le biais d'une méta-transformation. Ceci est réalisé au moyen des outils de transformation ATL et Acceleo.

8.3.1 Principe de notre approche

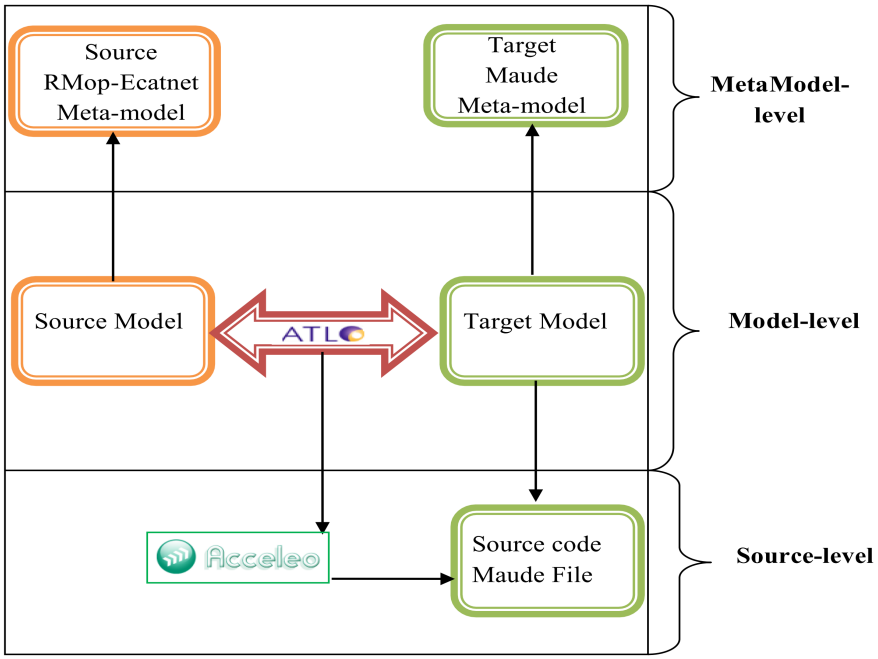


FIGURE 8.8 – Approche de transformation RMop-ECATNet2Maude

La Figure 8.8 présente une vue globale de notre approche de transformation ; elle décrit l'approche avec ses différents niveaux d'abstraction : le méta-niveau, le niveau modèle, le niveau transformation et le niveau code source.
Nous réalisons, d'abord, une transformation «modèle à modèle» à l'aide de l'outil ATL puis une seconde «modèle à texte» au moyen de l'outil Acceleo.
Le processus de transformation produit des modules Maude à partir des spécifications contenues dans le modèle RMop-ECATNet. Des règles de transformation sont définies pour les deux Méta-modèles conçus ; les modèles source et cible sont conformes à leurs méta-modèles et constituent respectivement les modèles d'entrée et de sortie du processus

de transformation.
La transformation proposée est exogène et horizontale sachant que les modèles source et cible ont le même niveau d'abstraction selon la classification présentée précédemment.

8.3.2 Méta-modèles

Méta modèle source :
Afin de pouvoir assurer la transformation RMop2Maude, nous avons dans un premier temps conçu un méta-modèle pour le modèle traditionnel des Mop-ECATNet. Ce modèle a été réajusté et réadapté en fonction du raffinement établi pour aboutir au méta-modèle RMop-ECATNet.

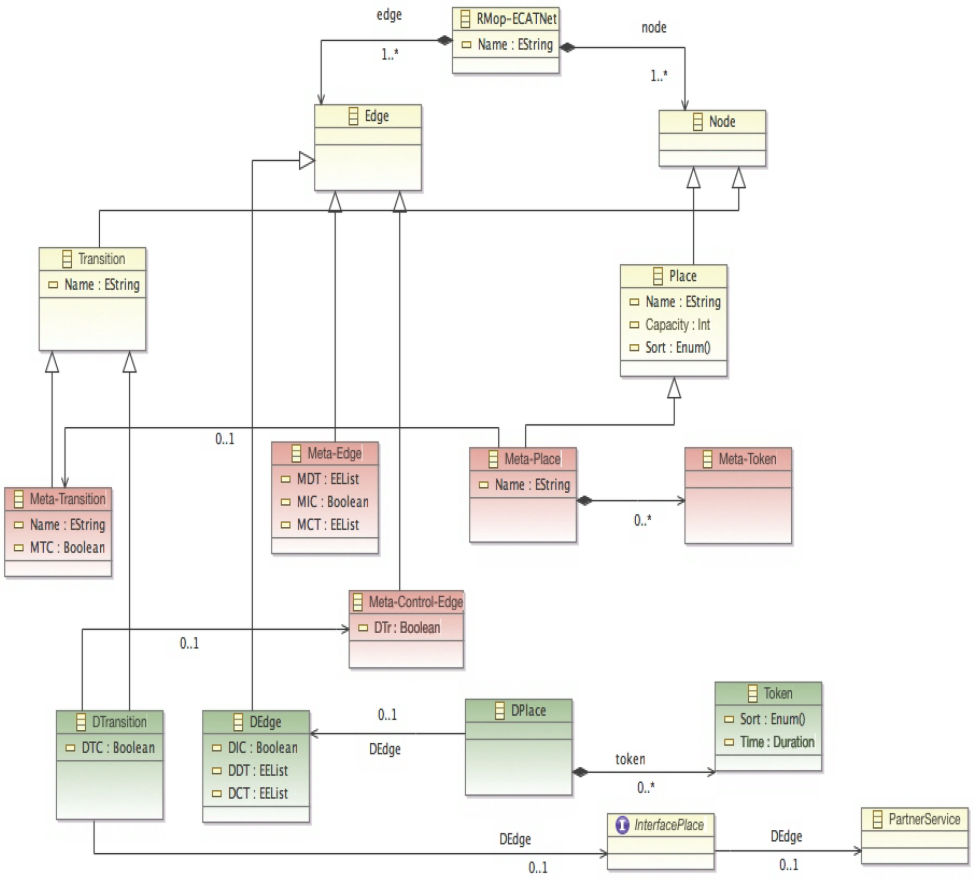


FIGURE 8.9 – Méta-modèle RMop-ECATNet

La Figure 8.9 illustre le méta-modèle RMop-ECATNet ; les classes dérivées de ce dernier sont par niveau :
Au Méta-niveau : Méta-Place, Méta-Transition, Meta-Edge, Meta-Control-Edge, Meta-Token.

Au Bas-niveau : DPlace, DTransition, Dedge, Interface Place, Token.
Méta modèle cible :
En parallèle au modèle source, le méta-modèle cible est inspiré du méta-modèle Maude publié dans [RDV08]. Plusieurs classes y figurant ont été dissimulées afin mettre en relief particulièrement les classes utiles à cette transformation. Le méta-modèle Maude réduit est donné dans la Figure 8.10.

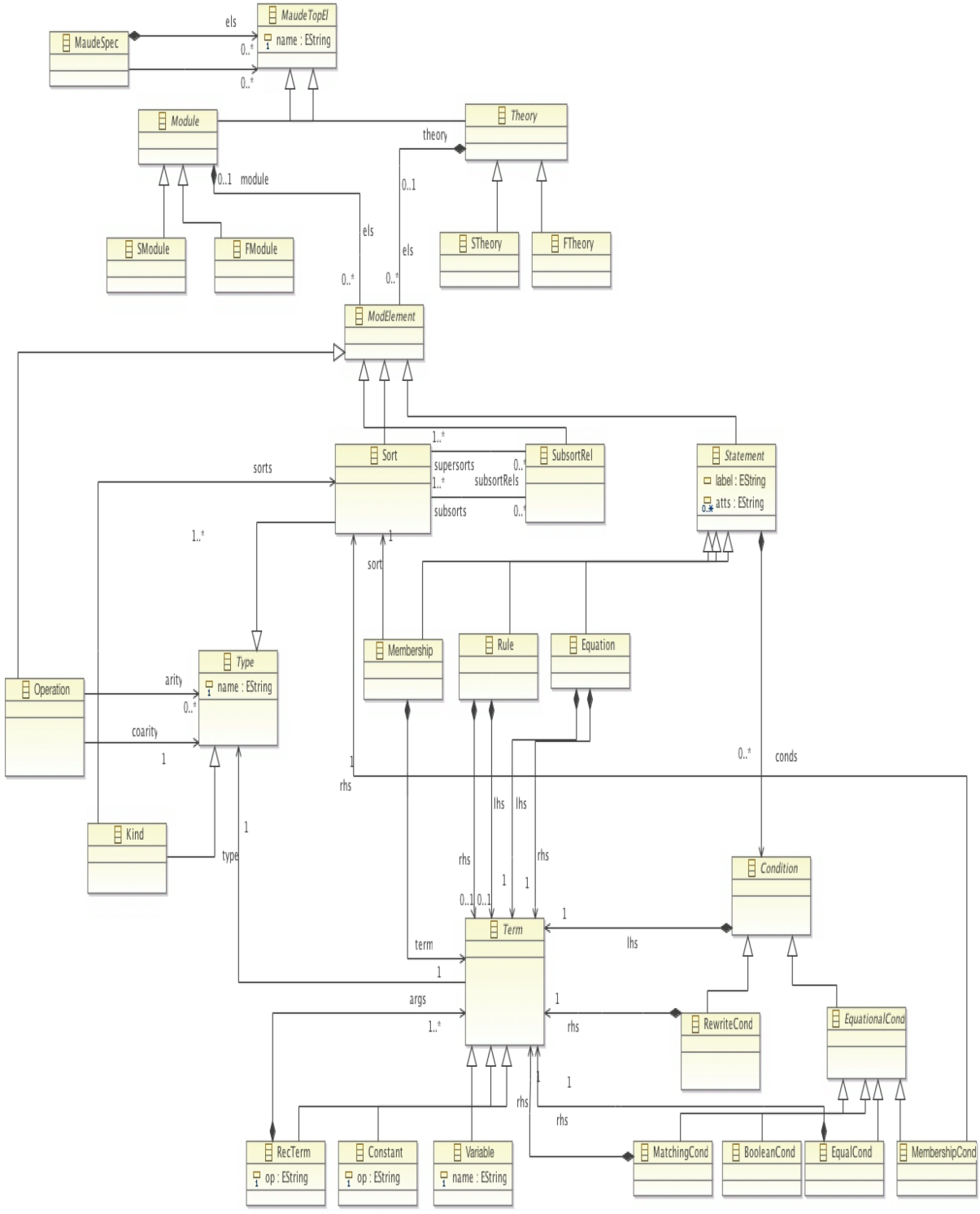


FIGURE 8.10 – Méta-modèle MAUDE

Pour notre approche de transformation, nous utilisons les classes suivantes : Maude FModule, SModule, Sort, Statement, Equation, Condition, Operation, Rule et Term.

8.3.3 Règles de Transformation

Dans cette section, nous établissons l'ensemble des correspondances entre les deux méta-modèles respectant la syntaxe dictée par l'outil ATL. Le Tableau 8.1 énumère quelques règles décrivant la correspondance entre les éléments du méta-modèle des RMop-ECATNets et ceux de Maude. Par exemple, la règle **Place2Module** permet la

Règles	*Méta-modèle des RMop-ECATNets*	*Méta-modèle Maude*
Place2Module	Place	Module Place
	Name	Supersort :Place
	Sort	Opn:_
	Capacity	Arity:String
		Coarity:Place
Transition2Module	Transition	Module Transition
	Name	Supersort :Transition
	TC	Arity:Transition
		Coarity:Boolean
		Opn:_
Edge2Module	Edge	Module Edge
		Supersort:Edge
MetaTransition2 Subsort	MetaTransition	Subsort of Transition
Meta Place 2Subsort	MetaPlace	Subsort of Place
…		

TABLE 8.1 – Principales règles de transformation RMop2Maude

transformation d'une place d'un RMop-ECATNet en un module fonctionnel Maude.
Rule Place2Module {

From
Source : RMopECATNetMM !Place
To
Target : MaudeMM !ModulePlace (
Name<- Source.Name)}

Elle est composée de deux parties (le From et le To) et est identifiée par un nom unique (**Place2module**). Cette règle précise les éléments du modèle de source qui doivent être liés (Place). Le nombre et le type des éléments du modèle cible générés (Module place) ainsi que la façon dont les éléments du modèle cible doivent être initialisées à partir des éléments source (Nom).

8.3.4 Exemple de transformation RMop2Maude

Afin d'illustrer notre proposition, nous reprenons le service de l'Agence de voyage (Figure 7.8). Pour atteindre la spécification Maude du service via l'approche de transformation, nous spécifions le modèle source du service à partir du méta-modèle RMop-ECATNet (RMop.ecore). Après la création des plugins des méta-modèles : RMop.ecore, Maude.ecore et l'établissement de toutes les règles de transformation correspondantes (Figure 8.12). Nous générons, via l'outil Acceleo, le modèle de sortie sous forme de fichier Maude.
De manière plus détaillée, afin de parvenir à une spécification textuelle Maude à partir de la représentation graphique du modèle de composition de l'Agence de voyage, nous procédons par étapes comme suit :

— Nous donnons d'abord, une esquisse du modèle de l'Agence de voyage basée sur des structures prédéfinies dans le tableau 7.1 et nous obtenons la vue abstraite du modèle (Figure 7.10).
— Une fois la structure validée, nous utilisons les tableaux(Tableau 7.2 et Tableau 7.3) pour réadapter le modèle selon les stratégies de contrôle définies (où nous identifions les transactions, les points de contrôle et les patrons qui peuvent être utilisés, ..etc). Nous obtenons ainsi le modèle global RMop-ECATnet de l'Agence de voyage Figure 8.11.

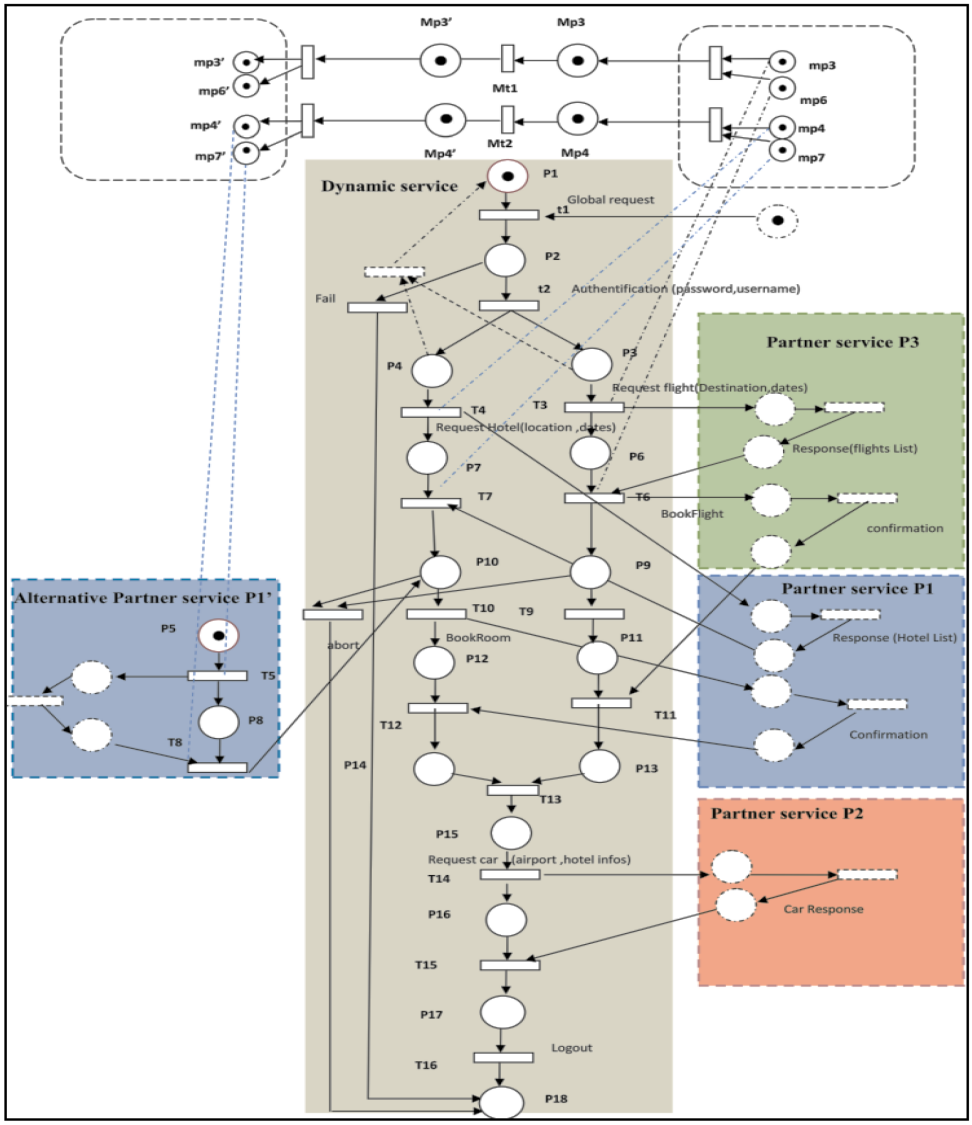


FIGURE 8.11 – Modèle global RMop-ECATNet du service Agence de Voyage

Nous réalisons la première transformation (modèle-à-modèle) à l'aide de l'outil ATL.
Tranformation Modèle à Modèle par le biais d'ATL

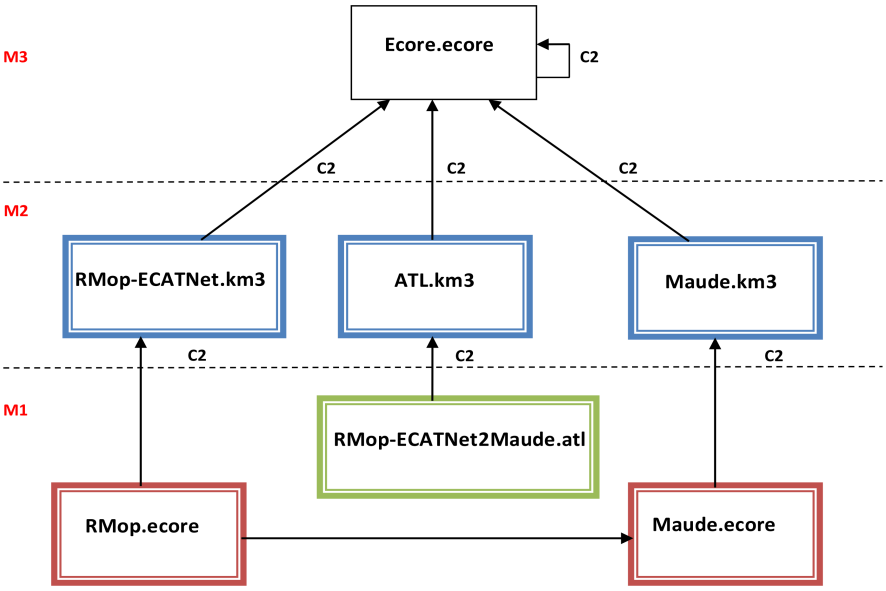


FIGURE 8.12 – Hiérarchie des modèles EMF dans la transformation modèle à modèle RMop2Maude

Pour réaliser cette transformation nous fournissons davantage ce qui suit :

Le méta-modèle source des RMop-ECATNets (Figure 8.9).

Le méta-modèle cible Maude (Figure 8.10).

Lemodèle source de l'Agence de voyage conforme au méta-modèle Mop-ECATNet.

Le modèle de transformation ATL RMop-ECATNet2Maude. Le modèle XMI résultant est conservé pour la seconde étape du processus de transformation.

Tranformation Modèle à Texte par le biais d'Acceleo

Pour atteindre une spécification textuelle Maude, nous procédons à une deuxième transformation Modèle à Texte (voir Figure 8.13)

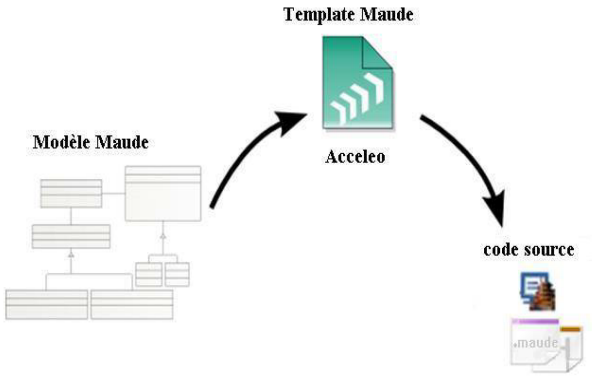


FIGURE 8.13 – Transformation «Modèle à Texte» sous Acceleo

Afin de réaliser cette transformation nous :

Spécifions le template Maude selon les caractéristiques de l'outil Acceleo.

Fournissons le modèle résultant de la première transformation (spécification en XMI). Nous aboutissons, enfin, automatiquement à une spécification textuelle Maude(voir Figure 8.14).

```
mod AgenceVoyageWs is

including Interface .
op AVWS : -> Markings [ctor] .
ops p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14
p15 p16 p17 p18 : -> DPlace [ctor] .
ops pi1 pi2 pi3 pi4 pi5 pi6 pi7 pi8 pi9 pi10 pi11
pi12 : -> lnrerfacePlace [ctor] .
ops Mp3 mp3 mp6 Mp3' mp3' mp6' MP4 mp4 mp7 Mp4' mp4'
mp7': -> Meta-Place[ctor] .
ops Mt1 Mt2 : -> Meta-Transition [ctor] .
ops t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14
t15 t16 : -> DTtransition [ctor] .
op AgenceVoyage : -> RMopECATNet [ctor].
var M : Markings .
var x : Bool .
eq FTWS = <p3,true > <p6,false> <p9.false> <p11,false>
<p13,false> .
rl [?flight request]: <p3,true> <pt,x> <pi1,request>
M => <p3,false> <pt,true> <pi1 ,none> M .
rl [!Available ticket]:<(pt.true> <pi2,none> <p9,x>
M => <pt,false> <pi2,FIights list> <p9,true> M .
rl [BookFlight]: <p9,true> <pi5,x> <pt4'.Bookflight>
M => <p9,false> <pt4,true> <pi2,none> M .
..
```

FIGURE 8.14 – Partie de la spécification Maude résultante du service Agence de voyage

8.4 Conclusion

Dans ce chapitre, nous avons procédé à la transformation du modèle des RMop-ECATNet en une spécification Maude en nous basant sur les techniques de la méta-transformation.
Les outils ATL et Acceleo ont servi comme support de base à cette transformation (modèle à modèle puis modèle à texte) et nous ont permis d'atteindre automatiquement une spécification textuelle Maude à partir du modèle de composition graphique initial.
Cette transformation a rendu plus facile la spécification aux utilisateurs peu familiers avec le langage Maude car ils peuvent obtenir automatiquement le code source à partir d'une représentation graphique. Il demeure également un gain avantageux en temps pour les experts du domaine en leur permettant de passer directement à d'autres phases telles que la vérification des propriétés du système en utilisant des outils de l'environnement Maude (par exemple : le modèle de vérification «modèle Checker»).

Conclusion Générale

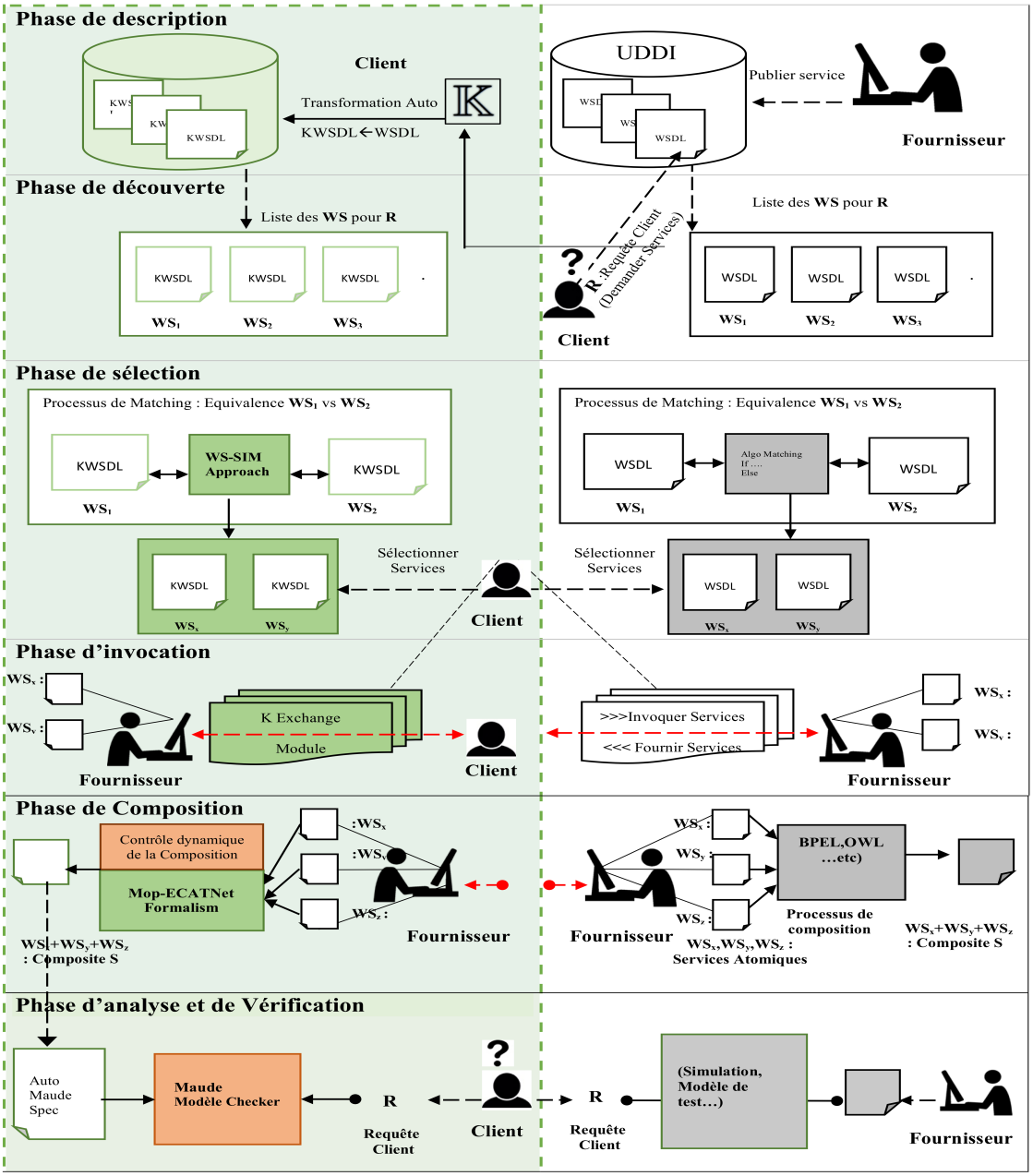


FIGURE 8.15 – Cadre sémantique formel vs standards pour le CV_{WS}

A la manière d'un puzzle la Figure 8.15 illustre l'aboutissement au cadre sémantique formel envisagé au départ. L'assemblage de ses pièces s'est fait progressivement et dans un ordre précis respectant l'enchainement des phases du CV_{WS}.
En effet, chaque phase s'est vu attribuer un traitement personnalisé. Pour y parvenir, certaines méthodes ou approches jugées obsolètes ont été réadaptées et actualisées ; d'autres nécessitaient un changement radical par transformation ou remplacement et enfin certaines ont subi un léger raffinement afin d'augmenter leur performance.
Nous estimons à ce niveau que la prérogative de travailler et de ré-exploiter l'existant a été pleinement respectée.
Nous notons également, comme conséquences positives le fait d'avoir traité des problèmes de fond qui persistent toujours dans le domaine des services Web en dépit du développement accru de cette technologie dont nous citons :

Le problème de sémantique : ce problème affecte toutes les phases ; sa résorption a contribué grandement à réconcilier les services et à améliorer leur interopérabilité.

La dynamicité des services Web : la dynamicité des environnements dans lesquels évoluent les services Web les rend imprévisibles et difficiles à maitriser. Les mécanismes de contrôle et stratégies mises en œuvre ont dompté ces services et neutralisé leur comportement imprévisible.

Le processus d'automatisation : cette étape est cruciale et est généralement liée à l'implémentation. Nous avons pu, grâce aux notions de méta-modélisation et aux théories étudiées, l'intégrer à chaque phase en procédant à un niveau assez abstrait.
Enfin, toutes nos contributions sont extensibles et exploitables dans tout environnement formel supportant la réécriture. En guise de perspectives, nous citons quelques notions relatives à l'aspect comportemental des services Web qui n'ont pas été prises en charge dans notre travail telles que les propriétés non fonctionnelles dans le processus de sélection de services (qualité de service) et les contraintes temporelles dans la composition de services.

Bibliographie

[ABH+02] : Anupriya Ankolekar, Mark Burstein, Jerry R Hobbs, Ora Lassila, David Martin, Drew McDermott, Sheila A McIlraith, Srini Narayanan, Massimo Paolucci, Terry Payne, et al. Daml-s : Web service description for the semantic web. In *International Semantic Web Conference*, pages 348–363. Springer, 2002.

[AFJ+68] : Donald C Augustin, Mark S Fineberg, Bruce B Johnson, Robert N Linebarger, F John Sansom, and Jon C Strauss. The sci continuous system simulation language/cssl/. 1968.

[AMM+04] : Anupriya Ankolekar, David Martin, Deborah McGuinness, Sheila McIlraith, Massimo Paolucci, and Bijan Parsia. Owl-s'relationship to selected other technologies. *W3C Member Submission*, 2004.

[ANS+10] : Alsayed Algergawy, Richi Nayak, Norbert Siegmund, Veit Köppen, and Gunter Saake. Combining schema and level-based matching for web service discovery. In *International Conference on Web Engineering*, pages 114–128. Springer, 2010.

[BB02] : Jean Bézivin and Xavier Blanc. Mda : Vers un important changement de paradigme en génie logiciel. *Développeur référence v2*, 16 :15, 2002.

[BBB+02] : Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, Kannan Govindarajan, Alan Karp, Harumi Kuno, Mike Lemon, Gregory Pogossiants, Shamik Sharma, et al. Web services conversation language (wscl) 1.0. *W3C Note*, 14, 2002.

[BCDG+05] : Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic composition of web services in colombo. In *SEBD*, pages 8–15, 2005.

[Bek12] : Amina Bekkouche. Composition des Services Web Sémantiques A base

d'Algorithmes Génétiques. PhD thesis, 2012.

[Béz04] : Jean Bézivin. In *search of a basic principle for model driven engineering.* Novatica Journal, Special Issue, 5(2) :21–24, 2004.

[BHL+05] : Boualem Benatallah, Mohand-Said Hacid, Alain Leger, Christophe Rey, and Farouk Toumani. On automating web services discovery. *The VLDB Journal The International Journal on Very Large Data Bases*, 14(1) :84–96, 2005.

[BMSB92] : Mohamed Bettaz, Mourad Maouche, Moussa Soualmi, and Madani Boukebeche. Using ecatnets for specifying communication software in the osi framework. In Computing and Information, 1992. Proceedings. ICCI'92., *Fourth International Conference on, pages* 410–413. IEEE, 1992.

[BPSM+97] : Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml). *World Wide Web Journal*, 2(4) :27–66, 1997.

[BR15] : Denis Bogdanas and Grigore Rosu. K-java : a complete semantics of java. *ACM SIGPLAN Notices*, 50(1) :445–456, 2015.

[C+02] : World Wide Web Consortium et al. Web service choreography interface (wsci) 1.0. http ://www. w3. org/TR/wsci/, 2002.

[CCCV06] : Javier Cmara, Carlos Canal, Javier Cubo, and Antonio Vallecillo. Formalizing wsbpel business processes using process algebra. *Electronic Notes in Theoretical Computer Science*, 154(1) :159–173, 2006.

[CCM+01] : Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, et al. Web services description language (wsdl) 1.1, 2001.

[CDE+00] : Manuel Clavel, Francisco Duran, Steven Eker, Patrick Lincoln, Narciso Marti-Oliet, José Meseguer, and Carolyn Talcott. Maude manual (version 2.6)(january 2011), 2000.

[CDGL+08] : Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella, and Fabio Patrizi. Automatic service composition and synthesis : the roman model. *IEEE Data Eng. Bull.*, 31(3) :18–22, 2008.

[CDK+02] : Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web : an introduction to soap, wsdl, and uddi. *IEEE Internet computing*, 6(2) :86–93, 2002.

[CELM96] : Manuel Clavel, Steven Eker, Patrick Lincoln, and José Meseguer. Principles of maude. *Electronic Notes in Theoretical Computer Science*, 4 :65–89, 1996.

[Cha97] : Claude Charmot. L'échange de données informatisé (EDI) : l'échange de données du commerce électronique, volume 3321. Ed. Techniques Ingénieur, 1997.

[CHT03] : Kishore Channabasavaiah, Kerrie Holley, and Edward Tuggle. Migrating to a service oriented architecture. I*BM DeveloperWorks*, 16, 2003.

[CZC08] : Marco Crasso, Alejandro Zunino, and Marcelo Campo. Easy web service discovery : A query-by-example approach. *Science of Computer Programming*, 71(2) :144–164, 2008.

[DBLPF06] : Jos De Bruijn, Holger Lausen, Axel Polleres, and Dieter Fensel. The web service modeling language wsml : an overview. In *European Semantic Web Conference*, pages 590–604. Springer, 2006.

[Dow98] : Troy Bryan Downing. Java RMI : remote method invocation. *IDG Books Worldwide*, Inc., 1998.

[DRS05] : John Domingue, Dumitru Roman, and Michael Stollberg. Web service modeling ontology (wsmo)-an ontology for semantic web services, 2005.

[EM45] : Samuel Eilenberg and Saunders MacLane. General theory of natural equivalences. *Transactions of the American Mathematical Society*, 58(2) :231–294, 1945.

[ER12] : Chucky Ellison and Grigore Rosu. An executable formal semantics of c with applications. In *ACM SIGPLAN Notices*, volume 47, pages 533–544. ACM, 2012.

[FB02] : Dieter Fensel and Christoph Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2) :113–137, 2002.

[FEBF06] : Jean-Marie Favre, Jacky Estublier, and Mireille Blay-Fornarino. L'ingénierie dirigée par les modèles : au delà du mda, traité ic2, série informatique et systèmes d'information, 2006.

[Fel98] : Christiane Fellbaum. WordNet. *Wiley Online Library*, 1998.

[Fer04] : Andrea Ferrara. Web services : a process algebra approach. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251. ACM, 2004.

[FGM+99] : Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul

Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. *Technical report*, 1999.

[GPST04] : John Garofalakis, Yannis Panagis, Evangelos Sakkopoulos, and Athanasios Tsakalidis. Web service discovery mechanisms : Looking for a needle in a haystack. In *International Workshop on Web Engineering*, volume 38, 2004.

[HB03] : Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *Proceedings of the 14th Australasian database conference*-Volume 17, pages 191–200. Australian Computer Society, Inc., 2003.

[HK97] : Markus Horstmann and Mary Kirtland. Dcom architecture. *Microsoft white paper*, 1997.

[HPS+03] : Ian Horrocks, Peter F Patel-Schneider, and Frank Van Harmelen. From shiq and rdf to owl : The making of a web ontology language. *Web semantics : science, services and agents on the World Wide Web*, 1(1) :7–26, 2003.

[HSS05] : Sebastian Hinz, Karsten Schmidt, and Christian Stahl. Transforming bpel to petri nets. In International conference on business process management, pages 220–235. *Springer*, 2005.

[HY81] : CL Hwang and K Yoon. Multiple criteria decision making. *Lecture Notes in Economics and Mathematical Systems*, 186, 1981.

[IW95] : Paola Inverardi and Alexander L. Wolf. Formal specification and analysis of software architectures using the chemical abstract machine model. *IEEE transactions on Software Engineering,* 21(4) :373–386, 1995.

[JMS06] : Matjaz B Juric, Benny Mathew, and Poornachandra G Sarang. Business Process Execution Language for Web Services : An Architect and Developer's Guide to Orchestrating Web Services Using BPEL4WS. *Packt Publishing Ltd*, 2006.

[Jos07] : Nicolai M Josuttis. SOA in practice : the art of distributed system design. " O'Reilly Media, Inc.", 2007.

[KB08] : Christoph Kiefer and Abraham Bernstein. The creation and evaluation of isparql strategies for matchmaking. In *European Semantic Web Conference*, pages 463–477. Springer, 2008.

[KC04] : Graham Klyne and Jeremy J Carroll. Resource description framework (rdf) : Concepts and abstract syntax. w3c recommendation, 2004. *World Wide Web Consortium*, http ://w3c. org/TR/rdf-concepts, 2004.

[KFS06] : Matthias Klusch, Benedikt Fries, and Katia Sycara. Automated semantic web service discovery with owls-mx. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922. ACM, 2006.

[KK06] : Frank Kaufer and Matthias Klusch. Wsmo-mx : A logic programming based hybrid service matchmaker. In *Web Services, 2006. ECOWS'06. 4th European Conference* on, pages 161–170. IEEE, 2006.

[KK08] : Matthias Klusch and Patrick Kapahnke. Semantic web service selection with sawsdlmx. In *Proceedings of the Second International Conference on Service Matchmaking and Resource Retrieval in the Semantic Web*-Volume 416, pages 2–16. CEUR-WS. org, 2008.

[KKR04] : Michael Klein and Birgitta König-Ries. Coupled signature and specification matching for automatic service binding. In *Web Services*, pages 183–197. Springer, 2004. [KM08] : Sandeep Kumar and Ravi Bhushan Mishra. A framework towards semantic web service composition based on multi-agent system. I*nternational Journal of Information Technology and Web Engineering (IJITWE)*, 3(4) :59–81, 2008.

[Knu64] : Donald E Knuth. Backus normal form vs. backus naur form. *Communications of the ACM*, 7(12) :735–736, 1964.

[KP08] : Dimitrios Kourtesis and Iraklis Paraskakis. Combining sawsdl, owl-dl and uddi for semantically enhanced web service discovery. *The Semantic Web : Research and Applications*, pages 614–628, 2008.

[KVBF07] : Jacek Kopecky, Tomas Vitvar, Carine Bournez, and Joel Farrell. Sawsdl : Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, 11(6), 2007. [L$^+$01] : Frank Leymann et al. Web services flow language (wsfl 1.0). 2001.

[LB14] : Fateh Latreche and Faiza Belala. Mop-ecatnets for formal modeling dynamic web services. In *ICAASE*, pages 27–34, 2014.

[LCTC10] : Chi-Chun Lo, Ding-Yuan Chen, Chen-Fang Tsai, and Kuo-Ming Chao. Service selection based on fuzzy topsis method. In *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference* on, pages 367–372. IEEE, 2010.

[LDRW04] : Jinyan Li, Guozhu Dong, Kotagiri Ramamohanarao, and Limsoon Wong. Deeps : A new instance-based lazy discovery and classification system. Machine learning, 54(2) :99–124, 2004.

[LDT07] : Haihua Li, Xiaoyong Du, and Xuan Tian. A wsmo-based semantic web

services discovery framework in heterogeneous ontologies environment. In *International Conference on Knowledge Science, Engineering and Management*, pages 617–622. Springer, 2007.

[LM07] : Roberto Lucchi and Manuel Mazzara. A pi-calculus based semantics for ws-bpel. *The Journal of logic and algebraic programming*, 70(1) :96–118, 2007. [LSB+11] : Fateh Latreche, Hacene Sebih, Faiza Belala, et al. Analyzing web services interaction using open ecatnets. 2011.

[LVM+06] : Ke Li, Kunal Verma, Ranjit Mulye, Reiman Rabbani, John Miller, and Amit Sheth. Designing semantic web processes : The wsdl-s approach. Semantic Web Services, Processes and Applications, pages 161–193, 2006.

[MBE03] : Brahim Medjahed, Athman Bouguettaya, and Ahmed K Elmagarmid. Composing web services on the semantic web. *The VLDB JournalThe International Journal on Very Large Data Bases*, 12(4) :333–351, 2003.

[MBH+04] : David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, et al. Owls : Semantic markup for web services. *W3C member submission*, 22 :2007–04, 2004.

[MBK05] : Ingo Muller, Peter Braun, and Ryszard Kowalczyk. Design patterns for agent-based service composition in the web. In *Quality Software, 2005.(QSIC 2005). Fifth International Conference* on, pages 425–430. IEEE, 2005.

[Mes92] : José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical computer science*, 96(1) :73–155, 1992.

[MGI06] : Sonia Ben Mokhtar, Nikolaos Georgantas, and Valérie Issarny. Cocoa : Conversation based service composition for pervasive computing environments. In *ICPS*, pages 29–38, 2006.

[MH05] : Jan Mendling and Michael Hafner. From inter-organizational workflows to process execution : Generating bpel from ws-cdl. In *On the Move to Meaningful Internet Systems 2005 : OTM 2005 Workshops*, pages 506–515. Springer, 2005.

[ML+03] : Nilo Mitra, Yves Lafon, et al. Soap version 1.2 part 0 : Primer. *W3C recommendation*, 24 :12, 2003. [MM03] : Sheila A McIlraith and David L Martin. Bringing semantics to web services. *IEEE Intelligent systems*, 18(1) :90–93, 2003.

[MPD+02] : Gunnar Mein, Shankar Pal, Govinda Dhondu, Thulusalamatom Krishnamurthi Anand, Alexander Stojanovic, Mohsen Al-Ghosein, and Paul M Oeuvray. Simple object access protocol, September 24 2002.

[MR07] : Patrick Meredith and Mark Hills Grigore Rosu. An executable rewriting logic semantics of k-scheme. In *Workshop on Scheme and Functional Programming*, volume 1, page 10, 2007.

[MVH+04] : Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10) :2004, 2004.

[MVR+04] : John Miller, Kunal Verma, Preeda Rajasekaran, Amit Sheth, Rohit Aggarwal, and Kaarthik Sivashanmugam. Wsdl-s : Adding semantics to wsdl-white paper. LSDIS Lab, University of Georgia, Georgia, USA, 2004.

[OVvdA+05] : Chun Ouyang, Eric Verbeek, Wil MP van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur HM ter Hofstede. Wofbpel : A tool for automated analysis of bpel processes. In *International Conference on Service-Oriented Computing*, pages 484–489. Springer, 2005.

[Pap03] : Mike P Papazoglou. Service-oriented computing : Concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference* on, pages 3–12. IEEE, 2003.

[Pee04] : Joachim Peer. A pddl based tool for automatic web service composition. In *International Workshop on Principles and Practice of Semantic Web Reasoning*, pages 149–163. Springer, 2004.

[Pel03] : Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10) :46–52, 2003.

[PH07] : Mike P Papazoglou and Willem-Jan Heuvel. Service oriented architectures : approaches, technologies and research issues. *The VLDB JournalThe International Journal on Very Large Data Bases*, 16(3) :389–415, 2007.

[PKPS02] : Massimo Paolucci, Takahiro Kawamura, Terry R Payne, and Katia Sycara. Semantic matching of web services capabilities. In *International Semantic Web Conference*, pages 333–347. Springer, 2002.

[Pon08] : Julien Ponge. Model based analysis of Time-aware Web service interactions. PhD thesis, Université Blaise Pascal-Clermont-Ferrand II, 2008.

[POSV04] : Abhijit A Patil, Swapna A Oundhakar, Amit P Sheth, and Kunal Verma. Meteor-s web service annotation framework. In *Proceedings of the 13th international conference on World Wide Web*, pages 553–562. ACM, 2004.

[PP09] : Pierluigi Plebani and Barbara Pernici. Urbe : Web service retrieval based on similarity evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 21(11) :1629–1642, 2009.

[R10] : Grigore Rosu and Traian Florin Serbãnutã. An overview of the k semantic framework. *The Journal of Logic and Algebraic Programming*, 79(6) :397–434, 2010.
[RDV08] : José Eduardo Rivera, Francisco Durán, and Antonio Vallecillo. A metamodel for maude. *Technical report*, 2008.

[RJB04] : James Rumbaugh, Ivar Jacobson, and Grady Booch. Unified modeling language reference manual, *the. Pearson Higher Education*, 2004.

[RKL+05] : Dumitru Roman, Uwe Keller, Holger Lausen, Jos De Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web service modeling ontology. *Applied ontology*, 1(1) :77–106, 2005.

[SF00] : Jon Siegel and Dan Frantz. CORBA 3 fundamentals and programming, volume 2.John Wiley Sons New York, NY, USA, 2000.

[SGS04] : David Skogan, Roy Grønmo, and Ida Solheim. Web service composition in uml. In Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. *Proceedings. Eighth IEEE International*, pages 47–57. IEEE, 2004.

[SK03] : Shane Sendall and Wojtek Kozaczynski. Model transformation : The heart and soul of model-driven software development. *IEEE software*, 20(5) :42–45, 2003.
[SN96] : Roy W Schulte and Yefim V Natis. Service oriented. Architectures, part, 1, 1996.

[SR10] : Traian Serbanut and Grigore Rosu. K-maude : A rewriting based tool for semantics of programming languages. Rewriting Logic and Its Applications, pages 104–122, 2010.

[SS05] : Gwen Salaün and Wendelin Serwe. Translating hardware process algebras into standard process algebras : illustration with chp and lotos. In *International Conference on Integrated Formal Methods*, pages 287–306. Springer, 2005.

[SSS07] : Dimitrios Skoutas, Alkis Simitsis, and Timos Sellis. A ranking mechanism for semantic web service discovery. In *Services, 2007 IEEE Congress* on, pages 41–48. IEEE, 2007.

[Tha01] : Satish Thatte. Xlang : Web services for business process design. *Microsoft Corporation*, 2001.

[TO02] : Brian E Travis and Mae Ozkan. Web services implementation guide, volume 1. *Architag Press*, 2002.

[TSPB02] : Rajesh K Thiagarajan, Amit K Srivastava, Ashis K Pujari, and Viswes-war K Bulusu. Bpml : A process modeling language for dynamic business models. In *Advanced Issues of E-Commerce and Web-Based Information Systems, 2002.(WECWIS 2002). Proceedings. Fourth IEEE International Workshop* on, pages 222–224. IEEE, 2002.

[VDKV+00] : Arie Van Deursen, Paul Klint, Joost Visser, et al. Domain-specific languages : An annotated bibliography. *Sigplan Notices*, 35(6) :26–36, 2000.

[WCL+06] : Ping Wang, Kuo-Ming Chao, Chi-Chun Lo, Chun-Lung Huang, and Yinsheng Li. A fuzzy model for selection of qos-aware web services. In *e-Business En-gineering, 2006. ICEBE'06. IEEE International Conference* on, pages 585–593. IEEE, 2006.

[XFZ10] : PengCheng Xiong, YuShun Fan, and MengChu Zhou. A petri net ap-proach to analysis and composition of web services. *IEEE Transactions on Systems, Man, and CyberneticsPart A : Systems and Humans*, 40(2) :376–387, 2010.

[YWYC09] : Gang Ye, Chanle Wu, Jun Yue, and Shi Cheng. A qos-aware model for web services discovery. In *Education Technology and Computer Science, 2009. ETCS'09. First International Workshop* on, volume 3, pages 740–744. IEEE, 2009.

Manel Amel DJENOUHAT

Un Cadre Sémantique Formel pour la Description, Sélection et Composition des Services Web

Résumé

Le but de cette thèse est de dégager un cadre sémantique formel approprié supportant l'interopérabilité de différents formalismes déjà utilisés pour décrire et déployer un service Web. En d'autres termes, nous contribuons au développement d'un formalisme mathématique rigoureux permettant de décrire un service Web complexe susceptible de changer pendant l'exécution et de coordonner avec les autres services de façon adaptative. Pour atteindre cet objectif, les étapes de description, de sélection et de composition ont constitué les trois majeures problématiques étudiées dans cette thèse.

Pour ce faire, nous avons proposé dans un premier temps, à travers l'utilisation du cadre sémantique formel K le langage K-WSDL; un langage de description de services Web doté d'une sémantique opérationnelle en terme de règles de réécriture qui peut être exécutable et analysable sous Maude. Nous avons introduit, dans un second temps, l'approche WS-Sim basée sur la théorie des catégories qui évalue l'équivalence comportementale entre services en représentant chaque service par une catégorie et en établissant des liens formels (foncteur) entre elles. Enfin, nous avons présenté le modèle RMop-ECATNet (Refined Meta Open ECATNet) : un modèle dédié à la spécification formelle de la composition des services Web et fruit du raffinement du modèle Mop-ECATNet proposé par [LB14]. Nous avons étendu et enrichi ce dernier aux trois niveaux : structurel, comportemental et implémentation.

Mots-clés : Services Web, Architecture Orientée Services, Composition dynamique de services, Sélection de services, Transformation de modèles, Logique de réécriture, Cadre sémantique K, Mop-ECATNets.

Résumé en anglais

The aim of this thesis is to provide a suitable formal semantic framework that supports interoperability of different formalisms already used to describe and deploy a Web service. In other words, we contribute to the development of a rigorous mathematical formalism to describe a complex Web service that may change during execution and coordinate with other services adaptively. To achieve this goal, the steps of description, selection and composition constitute the three major issues studied in this thesis.

We proposed so, initially, through the use of the K semantic framework the K-WSDL : a Web services description language endowed with an operational semantics in terms of rewriting rules which can be executed and analyzed in Maude. We introduced, in a second step, WS-Sim, a new approach based on the category theory which evaluates the behavioral equivalence between services by representing each service by a category and by establishing formal links (functor) between them. Finally, we present RMop-ECATNet (Refined Meta Open ECATNet): a formal model for the specification of services composition. product of the refinement of the Mop-ECATNets model, introduced initially by [LB14]. We extended and enriched this model at three distinct levels: at the structural, behavioural level and implementation levels.

Keywords: Web Services, Service-Oriented Architecture, Dynamic Service Composition, Service Selection, Model Transformation, Rewrite Logic, K Framework, Met a-Modelisation, Meta Open ECATNets.