UNIVERSITÉ DE NANTES

ÉCOLE DOCTORALE

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DES MATÉRIAUX

Année : 2005 N° B.U. :

Thèse de Doctorat de l'Université de Nantes

Spécialité : INFORMATIQUE

Présentée et soutenue publiquement par

Denivaldo Cicero Pavão LOPES

le 11 juillet 2005

à l'UFR Sciences et Techniques, Université de Nantes

Étude et applications de l'approche MDA pour des plates-formes de Services Web

Jury

Président	:	Béatrice DAILLE, Pr.	Université de Nantes, France
Rapporteurs	:	Bernard COULETTE, Pr.	Université de Toulouse le Mirail, France
		Mohamed QUAFAFOU, Pr.	Université d'Avignon, France
Examinateur	:	Luís Ferreira PIRES, Pr.	Université de Twente, Pays Bas

Directeur de Thèse : Pr. Jean BÉZIVIN
Laboratoire : LINA – LABORATOIRE D'INFORMATIQUE DE NANTES ATLANTIQUE. 2, rue de la Houssinière, F-44322 Nantes cedex 3, France

Co-encadrant : Slimane HAMMOUDI
Laboratoire : ESEO – ÉCOLE SUPÉRIEURE D'ÉLECTRONIQUE DE L'OUEST. 4, rue Merlet de la Boulaye, BP 30926 - 49009 Angers cedex 01, France

N° ED 0366-198

ÉTUDE ET APPLICATIONS DE L'APPROCHE MDA POUR DES PLATES-FORMES DE SERVICES WEB

Study and Applications of the MDA Approach for the Web Service Platform

Denivaldo Cicero Pavão LOPES

⋈

favet neptunus eunti

Université de Nantes

Denivaldo Cicero Pavão LOPES

Étude et applications de l'approche MDA pour des plates-formes de Services Web

xxxii+264 p.

Supposons que je parle les langues des hommes et même celles des anges : si je n'ai pas d'amour, je ne suis rien de plus qu'un métal qui résonne ou qu'une cymbale bruyante. Je pourrais transmettre des messages reçus de Dieu, posséder toute la connaissance et comprendre tous les mystères, je pourrais avoir la foi capable de déplacer des montagnes, si je n'ai pas d'amour, je ne suis rien. Je pourrais distribuer tous mes biens aux affamés et même livrer mon corps aux flammes, si je n'ai pas d'amour, cela ne me sert à rien.

— Paul DE TARSE.

Résumé

Les services Web sont des technologies émergentes et prometteuses pour le développement, le déploiement et l'intégration d'applications Internet. Ces technologies, basées sur XML, fournissent une infrastructure pour décrire (WSDL), découvrir (UDDI), invoquer (SOAP) et composer (BPEL4WS) des services. Un des avantages majeurs des services Web par rapport à ses prédécesseurs tels que CORBA, DCOM et XML-RPC est l'apport de l'interopérabilité sur Internet. Cependant, les services Web ne sont pas capables de résoudre tous les problèmes. Actuellement, les systèmes d'information sur Internet sont créés en utilisant différentes technologies (par exemple, intergiciels et langages de programmation). Ils évoluent constamment et peuvent être intégrés avec des technologies anciennes. Ces facteurs rendent le développement et la maintenance de ces systèmes plus complexes qu'avant. Dans ce contexte, les services Web sont un intergiciel de plus et d'autres verront le jour. Un axe de recherche prometteur pour le développement d'applications Internet sur les plates-formes services Web consiste à séparer les aspects indépendants et dépendants de la plate-forme par une description séparée de leurs modèles. Cette tendance est mise en avant par l'approche de l'architecture dirigée par les modèles (MDA - *Model Driven Architecture*), définie par l'OMG (*Object Management Group*). Cependant, avant que ceci devienne une réalité courante, plusieurs questions nécessitent des réponses tels que la transformation de modèles, le choix entre les langages de modélisation et les méthodologies. Dans cette thèse, une démarche conceptuelle et expérimentale de l'approche MDA est utilisée sur les plates-formes de services Web afin d'identifier les problématiques autour de MDA et de proposer des solutions viables. Ainsi, nous utilisons UML (*Unified Modeling Language*) et EDOC (*Enterprise Distributed Object Computing*) comme formalisme pour décrire les aspects indépendants de la plate-forme (modèle métier), et nous proposons des métamodèles pour décrire les aspects dépendants de la plate-forme des services Web, J2EE et dotNET (modèle de plate-forme). Ensuite, nous présentons notre démarche pour passer d'un modèle métier vers un modèle de plate-forme. Les apports de cette thèse sont : la séparation explicite entre spécification de correspondances et définition de transformations ; l'étude de la plate-forme des Services Web dans le contexte de MDA ; la proposition des spécifications de correspondances entre les métamodèles de UML, de EDOC, des services Web, de J2EE et de dotNET ; les définitions de transformations en ATL (*Atlas Transformation Language*) générées à partir de ces spécifications ; un outil nommé MMT (*Mapping Modeling Tool*) pour supporter d'une part, la création de spécifications de correspondances et, d'autre part, la génération de définitions de transformations.

Mots-clés : L'architecture dirigée par les modèles, services Web, EDOC, UML, spécification de correspondances, définition de transformations et outils.

Classification ACM

Catégories et descripteurs de sujets : D.2.12 [**Software**]: Software Engineering—*Data mapping*

Termes généraux : Design, Experimentation, Theory

Abstract

Web Services are emergent and promising technologies for the development, deployment and integration of applications on the Internet. These technologies are based on XML, providing an infrastructure to support the description (WSDL), discovery (UDDI), interaction (SOAP) and composition (BPEL4WS) of services. One of the major advantages of Web Services compared to their predecessors such as CORBA, DCOM and XML-RPC is enhanced interoperability on the Internet. However, Web Services are not able to provide solutions to all problems. Currently, the software systems are created using different technologies (e.g. middleware and programming languages). They are constantly evolving, and may be integrated with old technologies. These factors make the development and maintenance of software systems more complex than before. In this context, the technologies of Web Services constitute one more middleware, and there are yet more to come. An emerging trend in the development of applications using Web Service platform is to distinguish their technology-independent and technology-specific aspects by describing them in separate models. The Model Driven Architecture (MDA), defined by Object Management Group (OMG), aims to achieve this. However, before this can become a mainstream reality, some issues in MDA approach need solutions such as model transformation, choice between modeling languages and methodologies. In this thesis, a practical experimentation of the MDA approach is used with Web Service platform in order to identify the problems linked to MDA and to propose viable solutions. Thus, we use Unified Modeling Language (UML) and Enterprise Distributed Object Computing (EDOC) as formalisms to describe the platform independent aspects (business model), and we propose metamodels to describe the platform dependent aspect of Web Services, J2EE and dotNET (platform model). We then present our approach to transforming a business model into a platform model. Finally, the contributions of this thesis are: the explicit separation of mapping specification and transformation definition; the study of Web Service platform in the MDA context; the proposition of mapping specifications between UML, EDOC, Web Services, J2EE and dotNET; the transformation definition in Atlas Transformation Language (ATL) generated from these specifications; a tool named Mapping Modeling Tool (MMT) to support, on the one hand, the creation of mapping specifications and, on the other, the generation of transformation definition.

Keywords: Model-Driven Architecture, Web Services, EDOC, UML, Mapping Specification, Transformation Definition and Tools.

ACM Classification

Subjects descriptors and Categories: D.2.12 [**Software**]: Software Engineering—*Data mapping*

General Terms: Design, Experimentation, Theory

À mes parents « Raimundo Cicero Lopes » et « Maria da Graça Pavão Lopes »
et à mon épouse « Nadircélia de Oliveira Araújo Lopes ».

Remerciements

Je souhaite, avant toute chose, remercier Dieu pour m'avoir soutenu et permis la réalisation de cette thèse.

Je remercie mon épouse, Nadircélia, pour son amour et son soutien dans les moments difficiles et de doute que j'ai rencontré durant la thèse.

Je remercie mes parents, Raimundo et Maria da Graça, pour leur soutien quant à la réalisation de cette thèse.

Je remercie mes frères, José Ribamar et Leila Maria, pour leur amitié et leurs encouragements.

Je remercie Jean Bézivin pour m'avoir encadré et assisté durant cette thèse. Ses conseils m'ont été d'une grande aide.

Je remercie Slimane Hammoudi pour m'avoir co-encadré durant cette thèse. Ses conseils m'ont été d'une grande aide.

Je tiens à exprimer ma plus grande gratitude au Pr. Bernard Coulette et au Pr. Mohamed Quafafou, rapporteurs, pour leurs précieux conseils et recommandations.

Je suis reconnaissant à la Pr. Béatrice Daille et au Pr. Luís Ferreira Pires, membres du jury, pour leurs participations et leurs conseils.

Un grand merci à l'ensemble du personnel de l'ESEO pour leur accueil chaleureux.

Je tiens à exprimer ma plus sincère gratitude à Samuel Val, Millene Val, Geoff Cawston, David Burns, Olivier Beaudoux, Olivier Camp, Daniel Schang, Patrick Albers, Joseph Mahé, Jean-Marc Percher, Guillaume Desgeorge, Jérôme Tissier, Mathieu Feuilloy et Jérôme Janvier pour avoir lu et corrigé le texte de cette thèse.

Je remercie Frédéric Jouault pour les discussions toujours utiles.

Je remercie le gouvernement de la France, et en particulier, le Conseil Général de Maine-et-Loire, pour m'avoir octroyé une bourse d'étude.

Je remercie l'Université de Nantes pour l'inscription à l'école doctorale STIM.

Enfin, un grand merci à Victor Hamon, ex-directeur de l'ESEO, et à Jacky Charruault, directeur de l'ESEO, pour m'avoir reçu dans la résidence Oxford et dans l'ESEO.

Sommaire

Introduction

Contexte de la thèse

Dans cette thèse, nous étudions l'approche de l'architecture dirigée par les modèles (MDA - *Model Driven Architecture*) et sa mise en œuvre sur la plate-forme de services Web. Ainsi, nous sommes concernés par les technologies liées à la plate-forme de services Web et à l'approche MDA.

Internet et Services Web

Depuis son apparition, l'informatique s'est développée massivement et surtout très rapidement. Si les années 80 ont vu la banalisation de l'ordinateur personnel, les années 90 ont permis d'étendre l'utilisation de l'informatique dans tous les domaines. En particulier, l'explosion de l'Internet a profondément changé nos méthodes de travail. Les logiciels informatiques sont alors devenus omniprésents dans les entreprises (par exemple, système de contrôle et de supervision, comptabilité et gestion des ressources humaines), dans les hôpitaux, dans les écoles, et sont de plus en plus utilisés dans la vie quotidienne (par exemple, téléphone, ordinateur portable, PDA, voiture, micro-onde et achat en ligne).

Les intergiciels (*middlewares*) tels que CORBA (*Common Object Request Broker Architecture*), Java RMI (*Java Remote Methode Invocation*) et DCOM (*Distributed Component Object Model*) ont été proposés pour simplifier le développement des logiciels et permettre l'interopérabilité entre les systèmes distribués et hétérogènes. Ils ont répondu de manière assez satisfaisante à l'intéropérabilité dans le contexte des systèmes distribués, mais l'apparition de l'Internet a introduit de nouvelles exigences (par exemple, interopérabilité sur Internet) auxquelles ils n'étaient pas adaptés. Dans un premier temps, ces intergiciels ont été étendus afin de faire face à ces nouvelles exigences, mais avec peu de succès. Ainsi les services Web ont été proposés afin de remplacer ou compléter les anciens intergiciels incapables de répondre efficacement à l'exigence d'interopérabilité sur l'Internet.

Les services Web ont ainsi émergé comme un ensemble de technologies prometteuses pour le développement, le déploiement et l'intégration d'applications Internet. Parmi ces technologies, nous pouvons citer :

- XML (*eXtensible Markup Language*) : le formalisme de base pour la représentation de données sur Internet.
- SOAP (*Simple Object Access Protocol*) : le protocole de communication.
- WSDL (*Web Services Description Language*) : la description de services.
- UDDI (*Universal, Description, Discovery and Integration*) : l'annuaire de services.
- BPEL4WS (*Business Process Execution Language for Web Services*) : le langage de composition de services.

Les services Web, conjointement à l'architecture orientée services (SOA - *Service Oriented Architecture*), fournissent une nouvelle manière de développer des applications conformes aux exigences de l'Internet. Dans cette architecture, l'accent est mis sur le service. Par conséquent, le développement des applications est basé sur un projet orienté service, ce qui est différent d'un projet orienté composant. Un projet orienté service se concentre sur les exigences déterminées au niveau de la stratégie et du processus métier, alors que les projets orientés composants se concentrent sur les composants du logiciel utilisé

pour livrer les services. Les services peuvent aussi être composés pour fournir une fonctionnalité plus complète tel qu'un composant.

Actuellement, les services Web semblent être la solution de l'avenir pour implémenter les systèmes distribués sur l'Internet. Cependant, l'histoire a déjà démontré qu'un seul intergiciel n'est pas capable de supporter tous les types d'application. De plus, ils ne sont pas suffisants pour assurer la pérennité des systèmes informatiques. Par conséquent, dans un futur plus lointain, de nouvelles exigences qui dépassent les capacités de services Web devraient apparaître. Ainsi, tout laisse à penser que d'autres intergiciels vont apparaître.

Architecture dirigée par les modèles (MDA)

Afin de répondre aux besoins des logiciels, toujours de plus en plus nombreux, complexes et consommateurs de ressources, différents paradigmes de programmation ont vu le jour. Le paradigme procédural a ainsi laissé la place au paradigme objet qui, à son tour, a laissé sa place au paradigme composant. Récemment, les services sont fortement réapparus surtout grâce aux services Web. Cependant, ces évolutions, bien que majeures, ne permettent plus de faire face à la complexité croissante du développement des logiciels. Les logiciels ne sont pas seulement complexes parce qu'ils doivent manipuler une grande quantité de données, mais aussi parce qu'ils sont constitués d'un volumineux code source. De plus, ils font généralement partie d'un système distribué. Une telle complexité des logiciels provient aussi d'autres facteurs majeurs :

- L'intégration de nouveaux aspects : le développement logiciel doit prendre en compte plusieurs aspects à partir du début et s'étend jusqu'à la fin du développement de manière uniforme. Parmi ces aspects, nous pouvons citer la sécurité, la disponibilité, les performances et tout ce qui est lié à la qualité de service (QoS - *Quality of Service*).
- L'évolution de la logique métier et des nouvelles technologies : l'évolution (parallèle) de la logique métier et surtout des plates-formes technologiques accroît considérablement la complexité des applications. Par exemple, CORBA, Java RMI et EJB (*Enterprise JavaBeans*) sont des technologies apparues dans les années 90 mais, aujourd'hui, ces technologies sont remplacées par les services Web ou adaptés pour interagir avec ces derniers.
- L'évolution et les anciennes technologies : malgré l'arrivée massive des nouvelles technologies, les anciennes technologies ne peuvent pas disparaître puisque elles sont encore présentes dans les systèmes du passé (*legacy systems*). Par exemple, les moniteurs transactionnels (*TP monitors*) sont apparus à la fin des années 60 et sont encore utilisés aujourd'hui. Ainsi cette évolution induit souvent un problème d'hétérogénéité.

En conséquence, le développement de logiciels informatiques ne peut plus être fait en utilisant une démarche intuitive. Il devient largement recommandé d'utiliser des méthodologies différentes pour concevoir, analyser, modéliser et produire ces logiciels. Plusieurs méthodologies ont été proposées pour supporter le développement logiciel dans les années 80, telles que OMT, Booch et Jacobson. En 1997, ces méthodologies ont convergées vers le formalisme UML (*Unified Modeling Language*). Ce langage pseudo-formel avait pour but de satisfaire au mieux les nouvelles exigences rencontrées dans le développement logiciel. Cependant, son usage s'est principalement limité à la documentation des logiciels. Parmi les langages de modélisation, nous pouvons également citer EDOC (*Enterprise Distributed Object Computing*) qui supporte le développement de composants basés sur les systèmes EDOC.

Plus récemment, l'OMG (*Object Management Group*) a proposé l'architecture dirigée par les modèles (MDA ^{TM} [1] - *Model Driven Architecture*) pour supporter le développement de systèmes distribués complexes. MDA est une nouvelle manière de spécifier les systèmes informatiques basés sur différents points de vue d'un même système. Tout d'abord, les fonctionnalités et le comportement d'un système sont décrits sans tenir compte des caractéristiques technologiques. Puis, la description des fonctionnalités et des comportements est utilisée pour créer les systèmes informatiques avec une technologie spécifique. La première phase est réalisée par un modèle indépendant d'une plate-forme technologique (PIM - *Platform Independent Model*) tandis que la deuxième phase est réalisée par un modèle dépendant d'une plate-forme technologique (PSM - *Platform Specific Model*). Dans la deuxième phase, les outils conformes à l'approche MDA doivent supporter la génération de code, les fichiers de déploiement et les fichiers de configuration. Enfin, le passage du PIM vers le PSM constitue un aspect important de cette approche.

Grâce à cette séparation explicite entre les modèles indépendants et dépendants d'une plate-forme, la logique métier peut être protégée contre les changements occasionnés par l'apparition d'une nouvelle technologie. De plus, elle apporte une plus grande réactivité quand les systèmes informatiques doivent évoluer pour utiliser une autre technologie ou atteindre d'autres exigences. L'approche MDA vise donc à libérer les systèmes informatiques de la dépendance d'une technologie spécifique, en fournissant la portabilité et l'interopérabilité au niveau des modèles. Cette approche supporte aussi la maîtrise de la complexité du développement de logiciels grâce au modèle qui devient l'élément unificateur.

Motivation

Les services Web sont des intergiciels plus adaptés à l'Internet que leurs prédécesseurs. Cependant, l'expérience a montré que la création d'un nouveau intergiciel (même avec des standards) n'est pas une solution viable à long terme pour supporter le développement logiciel. En effet, les services Web peuvent aider dans la création des logiciels, mais ils restent d'un niveau d'abstraction bas.

Un axe de recherche prometteur, pour le développement d'applications Internet, consiste en la séparation des aspects indépendants et dépendants de la plate-forme par une description séparée de leurs modèles. Cette tendance est mise en avant par l'approche MDA.

De ce fait, deux paradigmes anciens, le service et le modèle, sont réapparus avec une nouvelle présentation, les services Web et l'approche MDA, afin de supporter le développement des applications distribuées. Les services Web fournissent la plate-forme cible, les modèles contribuent à la maîtrise de la complexité, et l'approche MDA guide l'utilisation de modèles pour implémenter les applications Internet sur la plate-forme des services Web. Ainsi, l'application de l'approche MDA à la plate-forme des services Web est un challenge pour le développement et la maintenance d'applications orientées Internet du futur.

Dans le but d'étudier et d'appliquer l'approche MDA pour la plate-forme de services Web, un partenariat entre l'Université de Nantes (l'inscription à l'école doctorale), l'ESEO (École Supérieure d'Électronique de l'Ouest) et le Conseil Général de Maine-et-Loire (l'organisme de financement de la thèse) a abouti à la réalisation de cette thèse.

[1] MDA ^{TM} (*Model Driven Architecture*) est une marque déposée par l'OMG (*Object Management Group*).

Problématique

L'approche MDA propose une architecture prenant en compte le développement et la maintenance de logiciels volumineux et complexes. En utilisant cette approche, les défenseurs de MDA préconisent que les développeurs de logiciels seront capables de créer et de maintenir les logiciels avec moins d'efforts que dans le passé. Cependant, avant que cela devienne une réalité courante, plusieurs problèmes dans l'approche MDA ont besoin de solutions telles que : la définition des métamodèles indépendant de plates-formes, la définition de métamodèles dépendants de plates-formes, la transformation de modèles, la gestion de la distance sémantique entre métamodèles, la correspondance bidirectionnelle et la traçabilité.

Parmi ces différents problèmes, nous nous sommes concentrés sur :

- La plate-forme des services Web dans le contexte de MDA.
- La transformation de modèles métier en modèles de la plate-forme de services Web.
- L'impact d'un choix des langages de modélisation et des plates-formes cibles dans la modélisation de la logique métier, ainsi que dans l'approche de transformation de modèles.

Démarche de la recherche et contribution

Nous avons choisi un exemple usuel d'application Internet de type B2B (*Business-to-Business*) et B2C (*Business-to-Consumer*), l'agence de voyages. Il nous semble suffisamment illustratif pour permettre de discuter l'approche MDA dans le contexte du développement d'applications Internet sur des plates-formes services Web.

La réalisation de cette expérimentation basée sur l'approche MDA exige dans un premier temps, une méthodologie, et dans un second temps, une architecture type permettant de mieux répondre aux besoins des transformations de modèles. Parmi les différentes approches de transformation de modèles utilisées en MDA [147], nous utilisons et proposons une extension de l'approche de transformation de métamodèles[2]. Notre proposition diffère de la transformation de métamodèles par le fait de la distinction explicite entre la spécification de correspondances et la définition de transformations.

Les contributions de cette thèse sont les suivantes :

- L'étude de la plate-forme des services Web dans le contexte de l'approche MDA à travers l'étude de cas « agence de voyages ».
- Une méthodologie pour mettre en œuvre l'approche MDA. Dans cette méthodologie, la transformation de modèles est divisée en deux étapes : la spécification de correspondances et la définition de transformations.
- Une proposition de métamodèles pour la plate-forme de services Web, J2EE et dotNET.
- Des spécifications de correspondances entre PIM (UML ou EDOC) et PSM (services Web, J2EE ou dotNET). Et des définitions de transformations. Le passage du PIM vers le PSM est illustré par l'étude de cas « agence de voyages ».
- Une étude de l'impact de langages de modélisation (UML et EDOC) et de plates-formes cibles (services Web, J2EE et dotNET) dans une démarche MDA.
- Une conception et une implémentation d'un outil permettant d'une part, de créer la spécification

[2]Le terme « transformation de métamodèles » est défini dans [147] pour désigner la transformation d'un modèle source en un modèle cible dans laquelle la « spécification de transformation » met en relation un métamodèle source et un métamodèle cible. Dans cette thèse, nous utilisons simplement le terme « transformation de modèle » à la place de « transformation de métamodèles ».

de correspondances entre deux métamodèles et, d'autre part, de générer la définition de transformations à partir de cette spécification.
- Un inventaire des difficultés et une proposition de solutions pour passer d'une abstraction à une implémentation (transformation PIM vers PSM).

Organisation de la thèse

Les sujets abordés dans cette thèse nous incitent à présenter le contexte technologique dans le premier chapitre. Nous y présentons les systèmes d'informations distribués et leurs évolutions, la plate-forme de services Web, les langages de modélisation UML, EDOC, les langages de métamodélisation MOF et Ecore, ainsi que l'architecture dirigée par les modèles (MDA).

Le deuxième chapitre présente un état de l'art de l'approche dirigée par les modèles. Il se focalise sur la spécification de correspondances, la transformation de modèles et l'application de l'approche MDA à la plate-forme de services Web.

Le troisième chapitre est dédié à la modèlisation. Nous y présentons une étude de cas qui va nous aider à illustrer notre démarche pour la spécification de correspondances entre métamodèles, ainsi que pour la création de la définition de transformations entre modèles. Nous présentons également les modèles indépendants de plates-formes et les plates-formes cibles choisies pour réaliser cette étude de cas.

Dans le quatrième chapitre, nous abordons la spécification de correspondances entre métamodèles et la création de définitions de transformations en ATL (*Atlas Transformation Language*). Nous étudions en particulier la spécification de correspondances entre UML et services Web, et entre EDOC et les services Web.

Le cinquième chapitre propose un métamodèle permettant la spécification de correspondances entre métamodèles, un outil nommé MMT (*Mapping Modeling Tool*) qui réalise ce métamodèle de correspondance, et des exemples d'utilisation.

En conclusion, nous établissons un bilan de nos travaux en synthétisant la problématique étudiée, la contribution fournie, ses limitations, ainsi que leurs perspectives.

PARTIE I

Contexte de l'étude

CHAPITRE 1

Contexte technologique

1.1 Introduction

Le contexte de cette thèse nous incite à présenter les technologies étudiées lors de nos recherches sur l'architecture dirigée par les modèles (MDA ^{TM} [1] - *Model Driven Architecture*) appliquée à la plate-forme des Services Web. Ce chapitre aborde le contexte et les technologies liées aux Services Web et à MDA.

Dans cette thèse, nous sommes concernés par l'application de MDA pour la plate-forme des Services Web afin de développer des systèmes d'information orientés Internet. Nous nous intéressons particulièrement au développement des applications Internet de type B2B *e-commerce* [2].

Les Services Web représentent aujourd'hui la plate-forme la plus convoitée et adaptée pour le développement des systèmes d'information distribués sur l'Internet. Plusieurs problèmes que les Services Web visent à résoudre peuvent être compris en considérant comment les systèmes d'information distribués ont évolué dans le passé. Un aspect important de ce processus d'évolution est qu'au fur et à mesure que la technologie évolue pour répondre à de nouvelles exigences, des nouveaux problèmes apparaissent augmentant le nombre d'exigences. Ainsi, ce contexte est fortement évolutif, mais avec des problèmes bien identifiés.

Dans ce chapitre, nous présentons un panorama de l'évolution des systèmes d'information, en mettant l'accent sur la place des intergiciels (*middlewares*), et l'apparition des Services Web et de MDA.

Ensuite, nous présentons les Services Web, en analysant leurs principaux standards (XML, SOAP, WSDL et UDDI), l'architecture orientée services ou SOA (*Service Oriented Architecture*) et la composition des services.

Enfin, nous abordons les langages de modélisation UML et EDOC, les langages de métamodélisation MOF et Ecore, le format d'échange XMI, et l'approche dirigée par les modèles (MDA), en particulier la correspondance entre les métamodèles et la transformation de modèles.

1.2 Les systèmes d'information : de la distribution à l'Internet

Nous abordons les systèmes d'information distribués en ciblant principalement les aspects suivants : la conception, l'architecture et les patrons de communications [8] [69]. Ensuite, nous abordons les intergiciels (*middlewares*) qui sont un des supports indispensables pour les systèmes d'information distribués.

[1] MDA ^{TM} (*Model Driven Architecture*) est une marque déposée par l'OMG (*Object Management Group*).

[2] B2B *e-commerce* est le terme utilisé pour désigner un business qui communique avec autre business généralement dans une relation commerciale via le Web.

1.2.1 La conception d'un système d'information

La conception d'un système d'information peut en général être organisée en trois couches (*layers*) : présentation, logique d'application et gestion de ressources. La figure 1.1 présente ces couches.

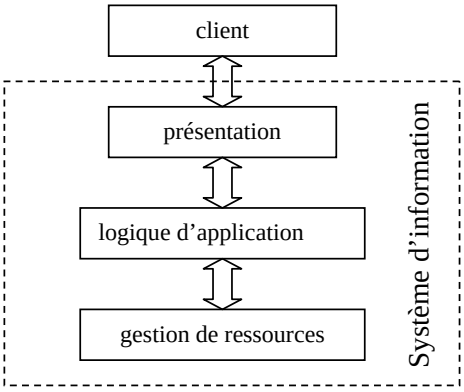


Figure 1.1 – Les différentes couches d'un système d'information

En général, les couches peuvent être définies comme suit [8] [69] :

- **Présentation** : Tout système d'information nécessite de communiquer avec des entités externes qui peuvent être des utilisateurs finaux ou d'autres ordinateurs. Une grande partie de ces communications impliquent la présentation d'informations aux entités externes et l'interaction avec elles. Cette tâche est réalisée par la couche de présentation qui peut être implémentée de différentes manières, par exemple, elle peut être implémentée par une interface utilisateur graphique ou par un module qui met en forme un ensemble de données selon une représentation spécifique. Dans le cas du Web, la couche de présentation peut être implémentée par une servlet Java qui fait la création de pages HTML (*HyperText Markup Language*).
- **Logique d'application** : En général, les systèmes d'information font le traitement de données avant de délivrer les résultats. Ce traitement utilise un programme qui implémente les opérations demandées par le client à travers la couche de présentation. La couche logique d'application est responsable du traitement de données. Ces programmes représentent souvent les services fournis par le système d'information.
- **Gestion de ressources** : Un système d'information accède et gère un ensemble de données. Ces données peuvent résider dans une base de données, un système de fichier, ou tout autre type de dépôt. Une couche de gestion de ressources gère tous les dépôts d'un système d'information.

La conception d'un système d'information peut être du type *top-down* ou *bottom-up*. La figure 1.2 illustre les deux cas et met en évidence les relations entre les étapes et les couches.

1.2.2 L'architecture d'un système d'information

Les trois couches présentées précédemment sont des constructions conceptuelles qui séparent logiquement les fonctionnalités d'un système d'information. Dans le plan réel, ces couches sont concrétisées par les *tiers*. Quatre types de systèmes d'information basés sur la notion de *tiers* peuvent être distingués : 1-*tier*, 2-*tiers*, 3-*tiers* et 3-*tiers thin client*.

La figure 1.3 présente l'architecture 1-*tier*. Cette architecture est la plus ancienne et remonte aux

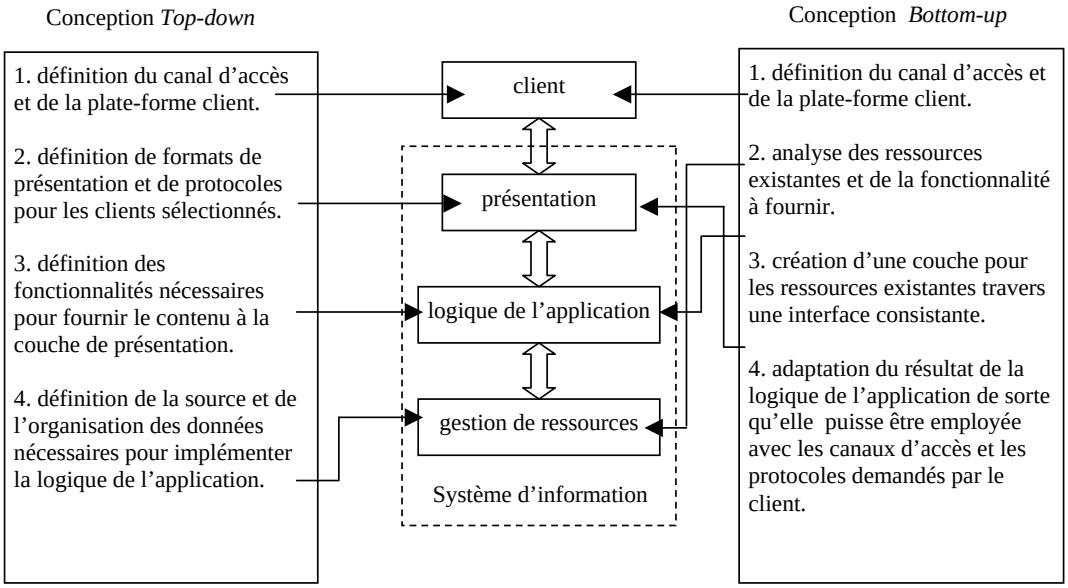


Figure 1.2 – La conception du type *top-down* et *bottom-up* [8]

années de gloire des *mainframes*. Dans ce cas, la gestion de ressources, la logique de l'application et la présentation sont faits par un *mainframe*. La présentation est préparée dans le *mainframe* et visualisée sur un terminal passif (*dumb terminal*).

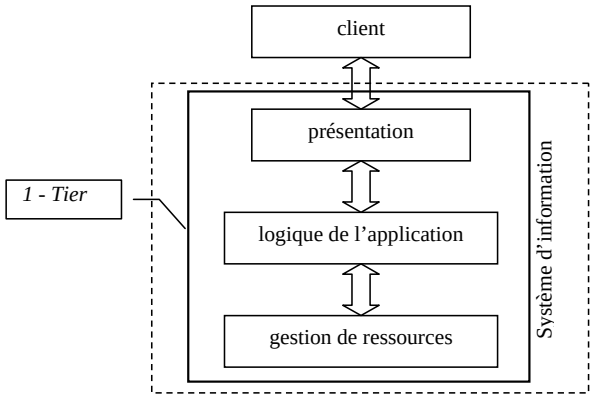


Figure 1.3 – L'architecture 1-*tier*

La figure 1.4 illustre l'architecture 2-*tiers*. Cette architecture est apparue en même temps que les ordinateurs personnels (PC - *Personal Computer*). En fait, les PC et les stations de travail ont fourni la puissance de traitement nécessaire pour séparer la présentation de la logique de l'application et de la gestion de ressources. Ainsi, la couche de présentation peut être traitée par les PC.

Le déplacement de la couche de présentation vers le PC a deux avantages importants. D'abord, la couche de présentation peut utiliser la puissance de traitement disponible dans les PC et, par conséquent, les ressources qu'elle utilisait deviennent disponibles pour les couches de logique de l'application et de gestion de ressources. Ensuite, ceci facilite la création de la présentation avec différents objectifs, sans

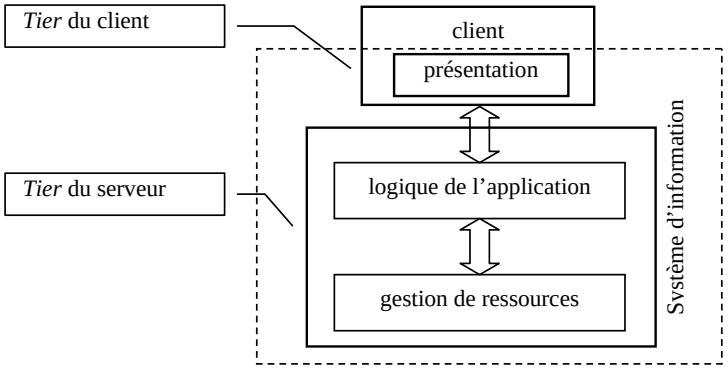


Figure 1.4 – L'architecture 2-*tiers*

augmenter la complexité du système.

L'architecture 2-*tiers* est devenue très populaire, en particulier pour les architectures client/serveur. Dans cette architecture, le client est responsable de la couche de présentation, et le serveur prend en charge la couche de logique de l'application et de gestion de ressources.

Les systèmes client/serveur ont représenté une évolution significative des systèmes informatiques. Comme les PC et les stations de travail sont devenus plus puissants (par exemple, CPU - *Central Processing Unit* plus rapides, plus d'espace mémoire vive et disque dur, et écrans couleurs), la couche de présentation a pu être de plus en plus sophistiquée. En conséquence, ces couches de présentation plus sophistiquées ont exigé plus de puissance de traitement (CPU plus puissantes) et de rapidité de communication (réseau plus rapide). Dans cette évolution, l'appel de procédure à distance (RPC - *Remote Procedure Call*) a été aussi un pas en avant.

Cependant, l'architecture 2-*tiers* a créé des îlots d'informations où un ensemble de clients pouvait communiquer avec leur serveur, mais il ne pouvait pas communiquer avec d'autres serveurs. Ainsi, l'intégration de serveurs est devenu un sujet courant dans l'informatique, et une nouvelle architecture pour répondre à ces nouvelles exigences était nécessaire.

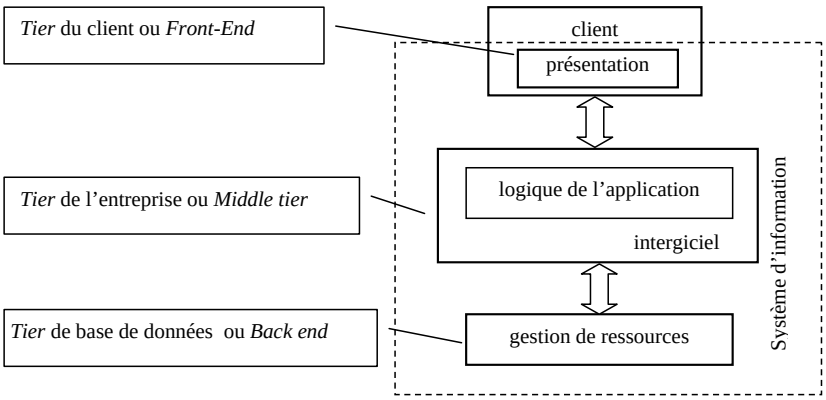


Figure 1.5 – L'Architecture 3-*tiers*

La figure 1.5 présente cette nouvelle architecture appelée 3-*tiers*. Dans cette architecture, la couche de présentation réside dans le client comme dans l'architecture 2-*tiers*. La logique de l'application ré-

side dans le *middle tier* où se situe l'intergiciel (*middleware*). Un intergiciel facilite et gère les interactions entre applications fonctionnant généralement sur des plates-formes hétérogènes. Nous pouvons citer comme exemple de intergiciels CORBA [143], Java RMI [184] et DCOM [125]. La gestion de ressources réside dans le *tier* de bases de données (ou *back end*).

Par rapport à l'architecture 2-*tiers*, l'architecture 3-*tiers* présente certains avantages et inconvénients. Dans l'architecture 2-*tiers*, les couches de la logique de l'application et de la gestion de ressources sont situées au même niveau : l'avantage est que la communication entre ces deux couches est efficace, la contrepartie étant qu'un serveur plus puissant doit être utilisé pour exécuter la logique de l'application et de la gestion de ressources, ce qui implique alors un coût du matériel nécessaire plus élevé.

Les systèmes 3-*tiers* ont introduit des concepts importants tels que les interfaces uniformes pour que la logique de l'application puisse accéder à la couche de gestion de ressources, comme ODBC (*Open DataBase Connectivity*) et JDBC (*Java DataBase Connectivity*).

Les architectures 3-*tiers* sont mieux exploitées quand elles travaillent pour intégrer différentes ressources. Dans ce cas, les intergiciels modernes fournissent non seulement un lieu pour développer la logique de l'intégration qui constitue le *middle tier*, mais aussi les fonctionnalités nécessaires pour doter le *middle tier* de propriétés additionnelles telles que : la transaction entre les gestionnaires de ressources, l'équilibrage de charge, la capacité de *logging*, la réplication, la persistance, la concurrence, etc.

Une possibilité d'utiliser les systèmes 3-*tiers* avec l'Internet est présentée par la figure 1.6. Dans la littérature, cette variante de l'architecture 3-*tiers* est connue sous le nom de 3-*tiers thin client*, parce que la présentation n'est plus créée dans le client, mais par un serveur d'application Web dédié. Celui-ci génère la présentation en utilisant la création dynamique de pages HTML qui sont ensuite affichées par un navigateur Web. Nous pouvons citer comme exemples de technologies pour créer des pages Web dynamiques : Java Servlets, JSP (*Java Server Pages*), ASP.NET (*Active Server Pages dotNET*) et PHP (*Personal Home Page*).

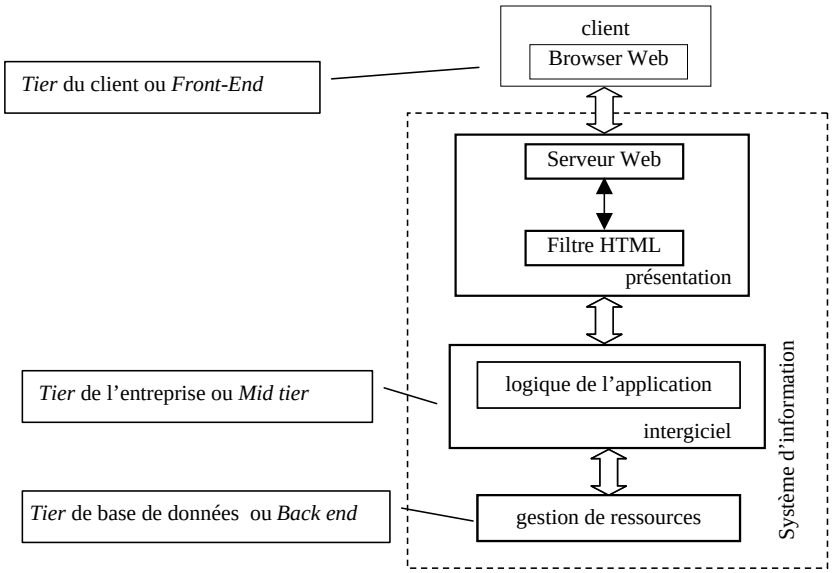


Figure 1.6 – L'Architecture 3-*tiers* avec le client léger

1.2.3 Communication dans les systèmes d'information

Les interactions dans les systèmes d'information distribués sont implémentées par les modèles de communication : appels synchrones et appels asynchrones.

Dans le cas des appels synchrones entre deux processus, celui que génère la demande doit attendre l'arrivée de la réponse pour pouvoir continuer son exécution.

Dans le cas des appels asynchrones, un processus envoie l'appel et continue son exécution jusqu'au moment où il demande le résultat de l'appel.

La figure 1.7 illustre l'appel synchrone et asynchrone.

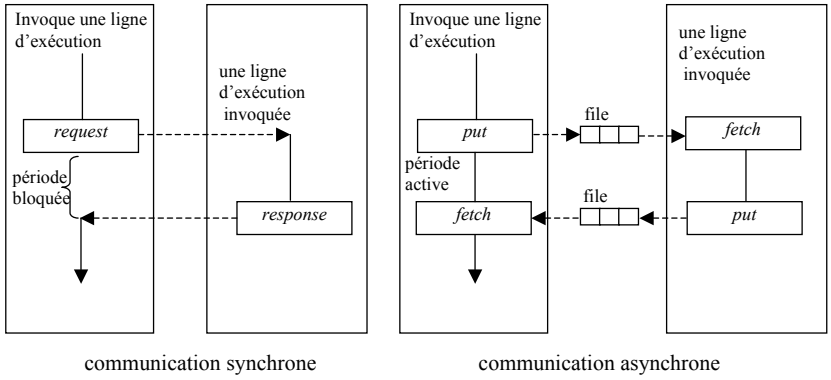


Figure 1.7 – Communications synchrone et asynchrone

1.3 Intergiciels

Les intergiciels (*middlewares*) sont largement utilisés dans les architectures des systèmes d'information [56] [175].

L'intergiciel est un logiciel de connexion formé d'un groupe de services qui permettent l'exécution de plusieurs applications sur une ou plusieurs machines connectées en réseau. Cette technologie fournit une couche d'abstraction qui permet l'interopérabilité entre différents langages de programmation, systèmes d'exploitation et architectures d'ordinateurs [195]. La figure 1.8 illustre de façon plus détaillée un intergiciel.

Les intergiciels peuvent se classer en différentes catégories :

- Systèmes basés sur RPC (*Remote Procedure Call*) : RPC est la forme la plus basique d'intergiciel. Elle fournit l'infrastructure nécessaire pour transformer les appels de procédures locales en appels de procédures distantes de façon uniforme et transparente [196]. Aujourd'hui, les systèmes RPC sont utilisés comme base pour presque toutes les autres formes d'intergiciel, par exemple CORBA [143] et Service Web [204].
- *TP monitors* (Moniteurs Transactionnels) : ils sont la forme la plus ancienne et la plus connue de intergiciel. Ils sont aussi les plus fiables, les mieux testés, et la technologie la plus stable en EAI (*Enterprise Application Integration*) [20].
- *Object brokers* : ils sont similaires à RPC, mais ont été introduits pour prendre en charge les systèmes orientés objets. L'exemple le plus populaire d'*object brokers* est CORBA de l'OMG [143].

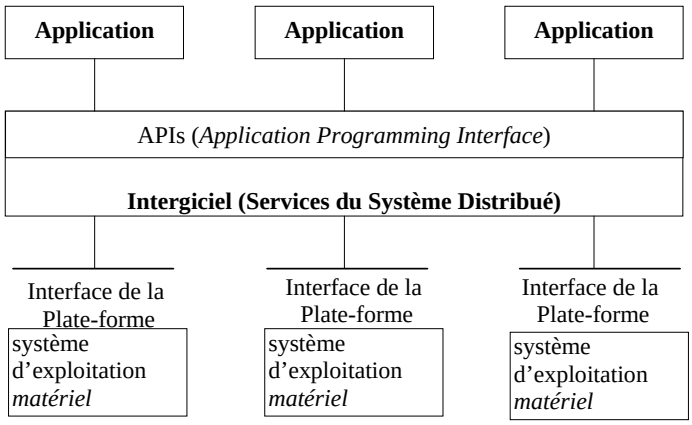


Figure 1.8 – Intergiciels [195]

- *Object monitors* : ils sont la fusion des TP monitors et des *object brokers*.
- *Message-oriented* : ce type d'intergiciel est connu comme MOM (*Message-Oriented Middleware*) basé sur les messages asynchrones. Il fournit un mécanisme d'accès transactionnel pour files, persistance pour files, et un numéro de primitives pour lire et pour écrire dans une file locale ou distante [107][194].
- WfMS (*Workflow Management Systems*) : un système de gestion de workflow permet le contrôle de l'envoi des informations entre les participants d'un processus ainsi que la définition de la logique métier nécessaire pour intégrer les systèmes hétérogènes et distribués [86].
- Services Web : c'est le type le plus récent d'intergiciel. Il a été conçu pour supporter l'interopérabilité sur l'Internet [209].

En fait, les intergiciels ont pour but premier de permettre l'interopérabilité entre systèmes informatiques. Une des premières technologies à prendre en charge l'interopérabilité a été EDI (*Electronic Data Interchange*) [56][84]. De plus, chaque intergiciel est basé sur un type spécifique de EDI. Par exemple, CORBA est basé sur CDR (*Common Data Representation*), et les Services Web sont basés sur XML (*eXtensible Markup Language*).

EDI a été pendant longtemps le support de la communication électronique pour réaliser les transactions entre les ordinateurs, comme l'ordonnancement, la facturation et les confirmations. D'abord, il a été utilisé pour réduire le transfert de documents sur papier, mais ensuite il a été utilisé en grande échelle pour automatiser les interactions. X12, EDIFAC et TRADACOMS sont des exemples des premiers standards EDI. Ils définissent la structure de documents à traiter par les ordinateurs. En d'autres termes, ils définissent le format de donnée utilisé pour la communication entre différentes plates-formes. Cependant, ces standards traditionnels dans leur état naturel ne sont pas adaptables à l'Internet.

Dans le contexte de l'Internet, XML a été utilisé comme un format de donnée pour EDI. D'autres ont adapté le standard traditionnel de EDI aux protocoles de l'Internet. Par exemple, OBI (*Open Buying on the Internet*) utilise le protocole HTTP (*HyperText Transfer Protocol*) pour transférer les messages sur l'Internet dont le contenu s'appuie sur le standard X12 [187].

Dans cette thèse, nous allons centrer la discussion sur les intergiciels en prenant comme types représentatifs RPC, *object brokers*, et les systèmes de gestion de workflow. Dans une section ultérieure, nous allons présenter les Services Web et les comparer à ces précédents intergiciels.

1.3.1 Systèmes basés sur RPC

Le mécanisme de RPC est la base sous-jacente de la majorité des intergiciels disponibles actuellement.

RPC a été introduit au début des années 80 par Birell et Nelson comme partie de leur travail sur l'environnement de programmation Cedar [23]. Dans cet environnement, le RPC fournissait une manière transparente d'appeler les procédures situées dans d'autres machines.

Les systèmes basés sur RPC ont introduit plusieurs concepts qui sont encore largement utilisés aujourd'hui, tels que IDL (*Interface Definition Language*), service de nom et service d'annuaire, liaison dynamique (*dynamic binding*), etc.

Le développement d'une application distribuée avec RPC est basé sur une méthodologie bien définie qui fournit un support très puissant pour l'architecture client/serveur. Nous allons donc présenter cette méthodologie, en considérant la création d'un serveur qui fournit des procédures qui peuvent être utilisées par un client.

La première étape consiste à définir l'interface pour la procédure, en utilisant l'IDL. L'IDL fournit une représentation abstraite pour les procédures en spécifiant les paramètres d'entrée et de sortie. En fait, l'IDL est la description des services fournis par le serveur.

La seconde étape consiste à compiler la description IDL. Le compilateur de l'IDL génère les *stubs* du client, les *stubs* du serveur, un modèle de représentation de données et les références.

La figure 1.9 illustre les étapes de la méthodologie pour utiliser RPC.

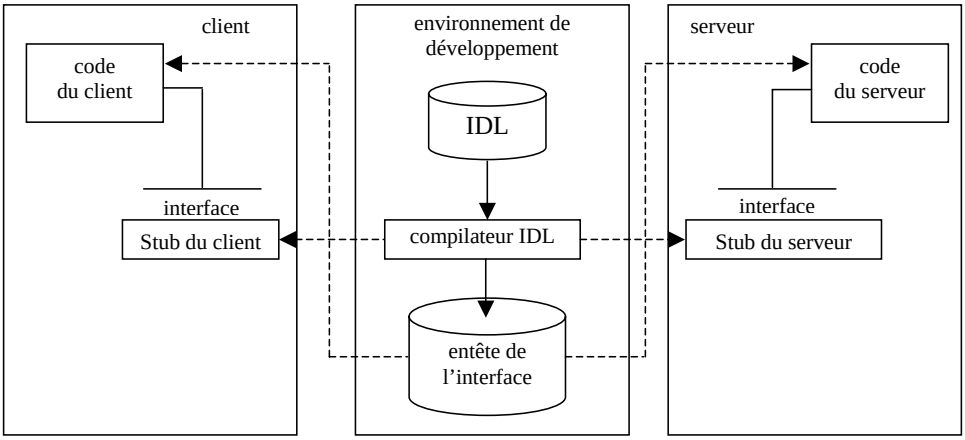


Figure 1.9 – Les étapes de la méthodologie pour travailler avec RPC

Quand un client fait un appel en RPC, il utilise les *stubs* du client pour demander l'exécution de la procédure distante. Ensuite, les *stubs* du client appellent les *stubs* du serveur, en passant le nom et les paramètres de la procédure demandée. Les stubs du serveur font l'appel à la procédure et renvoient les résultats de l'appel aux *stubs* du client. Les *stubs* du client retournent le résultat à l'application client.

Nous pouvons citer le DCE (*Distributed Computing Environment*) fournie par l'OSF (*Open Software Foundation*) [57] comme une extension significative de RPC. Une plate-forme basée sur DCE fournit un service RPC et un nombre de services additionnels qui sont très importants pour développer les systèmes d'information distribués. Par exemple, nous pouvons mentionner les services : service de synchronisation, service de fichiers distribués et service de sécurité.

1.3.2 *Object brokers*

Les *object brokers* sont des intergiciels qui supportent l'interopérabilité entre les objets. Ils sont apparus au début des années 90 comme une évolution naturelle de RPC pour s'adapter à l'orientation objet. Les objectifs des *object brokers* sont similaires à ceux de RPC, c'est à dire masquer la complexité derrière les invocations à distance. La complexité est masquée en utilisant des mécanismes qui dissimulent les invocations à distance par les invocations locales du point de vue du programmeur.

L'exemple le plus connu de *object broker* est CORBA de l'OMG [143]. Il s'agit d'une spécification et d'une architecture pour la création et la gestion des applications orientées objet distribuées sur un réseau. La spécification de CORBA est indépendante d'une implémentation et quelconque par rapport du langage de programmation et du système d'exploitation.

CORBA a connu une popularité énorme dans les années 90. Plusieurs entreprises ont développé des produits basés sur CORBA, comme Orbix de l'IONA, Visibroker de la Borland et JDK-CORBA de Sun.

Un système utilisant CORBA est basé sur trois éléments principaux :

- ORB (*Object Request Broker*) : il fournit les fonctionnalités de base pour l'interopérabilité des objets.
- *CORBA services* : il désigne un ensemble de services qui est accessible à travers une API standardisée. Ils fournissent des fonctionnalités plus spécialisées pour supporter le développement et l'exécution des objets plus complexes. Persistance, gestion de cycle de vie et sécurité sont des exemples de services CORBA.
- *CORBA facilities* : il s'agit d'un ensemble de facilités fournissant des services qui sont davantage dédiés aux applications qu'aux objets individuels. Gestion de la documentation, internationalisation et support pour les agents mobiles sont des exemples de *CORBA facilities*.

La figure 1.10 présente l'architecture de CORBA et met en évidence les relations entre chaque élément.

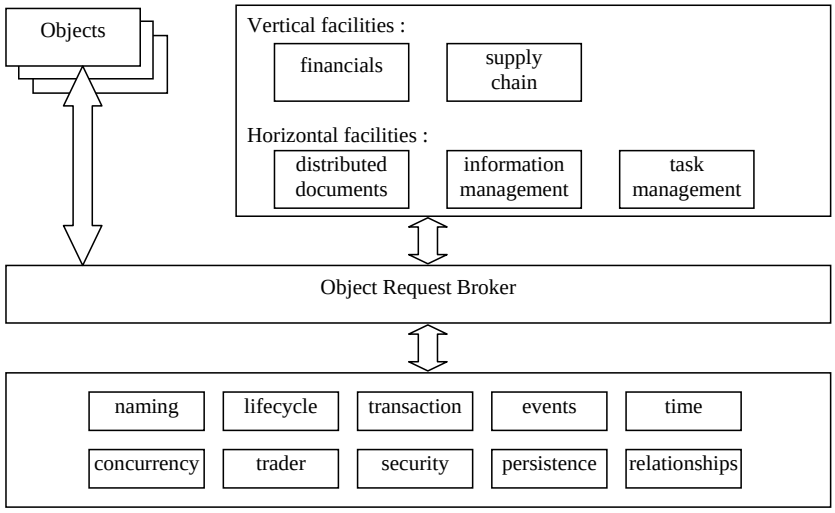


Figure 1.10 – L'architecture de CORBA simplifiée

Pour travailler avec CORBA, il faut d'abord spécifier les interfaces des objets. Les interfaces de CORBA sont définies en IDL-CORBA. Par rapport aux langages de définition d'interface de RPC, IDL est plus puissant parce qu'il supporte l'héritage et le polymorphisme.

De la même façon que RPC, un pré-compilateur IDL génère les codes nécessaires pour implémenter et invoquer un objet CORBA. Il génère du côté client les *Stubs* et du côté serveur les *Skeletons*. Ces deux éléments sont la base pour effectuer des invocations statiques en CORBA.

De plus, CORBA supporte l'invocation dynamique. Dans ce cas, le client doit trouver une interface d'un objet demandé et l'utiliser pour faire l'appel pendant l'exécution.

Pour utiliser un service fourni par un objet CORBA, le client doit connaître ce service. En CORBA, les références de services sont supportées au travers du *Naming service* [153] et *Trading Object service* [137]. Par rapport à un annuaire téléphonique, le *Naming service* est similaire aux pages blanches et le *Trading Object servic*e est similaire aux pages jaunes. Le *Naming service* permet de récupérer des informations d'un service à partir du nom du service demandé. Le *Trading Object service* permet de récupérer des informations d'un service à partir de ses propriétés. Dans ce contexte, le *trader* est un autre acteur qui aide le client à trouver un service selon les propriétés spécifiées. De plus, ces services et le *Purchasing service* peuvent être utilisés ensemble pour supporter la découverte, la vente et l'utilisation de services de façon automatisée. Cependant, la capacité d'utiliser CORBA pour faire la sélection de service et l'invocation dynamique est rarement utilisée. En fait, l'invocation dynamique, *Naming service* et *Trading Object service* résolvent seulement une partie du problème. Pour sélectionner et créer des invocations dynamiques, il faut avant tout que le client sache quelle est la signification des propriétés des services.

Un des points forts d'*object broker* est sa capacité à encapsuler les détails d'un objet de ses clients. De plus, tout ORB respectant la spécification de CORBA doit utiliser le GIOP (*General Inter-ORB Protocol*) pour faire la communication avec d'autres ORB. GIOP est un protocole standardisé et peut être implémenté au-dessus de différents protocoles de transport. Pour être conforme à la spécification CORBA, les ORB doivent aussi supporter le GIOP au dessus de TCP/IP, ce qui est connu sous le label IIOP (*Internet Inter-ORB Protocol*).

D'autres *object brokers* ont aussi été développés comme DCOM (*Distributed Component Object Model*) et son descendant, COM+, tous les deux créés par Microsoft. Cependant, DCOM et COM+ sont des *object brokers* spécifiques au système d'exploitation de Microsoft [125].

1.3.3 Systèmes de gestion de Workflow (WfMS)

Un système de gestion de *Workflow* ou WfMS (*Workflow Management System*) est un système complet qui sert à définir, gérer et exécuter des procédures en exécutant des programmes dont l'ordre d'exécution est pré-défini dans une représentation informatique de la logique de ces procédures [215].

D'abord, les systèmes de gestion de *Workflow* ont été utilisés pour automatiser des processus administratifs basés sur des versions électroniques de documents et pour prendre en charge leur distribution. Ces premiers WfMS constituent le groupe appelé *workflow* administratif et sont définis comme suit :

Workflow est l'automatisation ou semi-automatisation d'un processus métier qui utilise un ensemble de règles permettant le contrôle du flux de documents, des informations et des tâches entre les participants d'une activité [86].

Ensuite, le *Workflow* a été utilisé pour prendre en charge le contrôle du transfert d'informations entre les participants d'un processus administratif, et pour définir la logique métier nécessaire à l'intégration des systèmes hétérogènes et distribués. Dans ce cas, les WfMS qui implémentent la logique métier constituent un groupe appelé *workflow* de production.

L'effort majeur pour standardiser le *Workflow* a été et est encore fait par la WfMC (*Workflow Management Coalition*). La WfMC est un consortium international d'éditeurs de *workflow*, d'utilisateurs,

d'analystes et de groupes de recherches dont les objectifs sont la promotion des technologies de *workflow* et la définition de standards. Parmi ces standards, nous pouvons citer des langages de définition de *workflow* et API (*Application Programming Interface*) pour accéder au WfMS.

La figure 1.11 illustre le modèle de référence de *Workflow* de la WfMC [86].

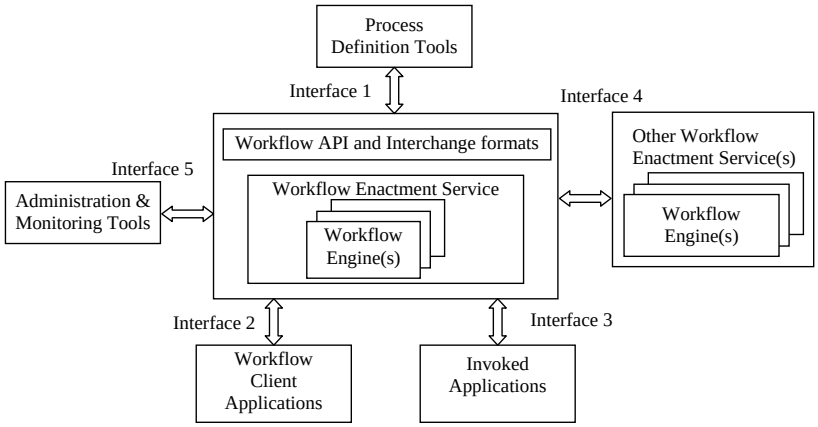


Figure 1.11 – Le modèle de référence de Workflow de la WfMC [86]

Ce modèle de référence illustre les principaux composants et interfaces de l'architecture d'un *workflow*.

Les principaux composants du modèle de référence sont les suivants :

- *Process Definition Tool* : est utilisé pour décrire les processus dans un format qui peut être traité par les ordinateurs.
- *Workflow API and Interchange formats* : les principales interfaces du modèle de référence du *workflow* sont définies comme WAPI (*Workflow Application Programming Interface*). WAPI est un ensemble de constructions permettant d'accéder aux services d'un système de *workflow* et par lequel les interactions entre le logiciel de contrôle du *workflow* et les autres composants du système sont régulés. Les *Interchange formats* supportent aussi l'interopérabilité entre différents composants d'un WfMS.
- *Workflow Enactment Service* : est un service de logiciel qui consiste en un ou plusieurs moteurs de *workflow* destinés à créer, gérer et exécuter les instances de *workflow*. Les applications doivent interagir avec ce service via WAPI.
- *Workflow Engine* : est un service logiciel qui fournit tout ou partie de l'environnement d'exécution d'un *Workflow*.
- *Administration et Monitoring Tools* : sont des outils pour gérer et suivre le déroulement des procédures *Workflow*.
- *Workflow Client Applications* : est une entité qui interagit avec l'utilisateur final.
- *Invoked Applications* : est une application automatiquement lancée par le système de gestion de *Workflow*.
- *Other Workflow Enactement Services* : le modèle de *workflow* préconise que les implémentations de *Workflow Enactement Services* doivent être interopérables.

Un *workflow* peut être spécifié par un graphe dirigé qui défini l'ordre d'exécution entre les nœuds dans un processus. Pour définir les processus en utilisant le *Workflow*, la WfMC a créé XPDL (*XML Process Definition Language*) [216]. En fait, le langage XPDL est un format d'échange entre les outils

de modélisation et les moteurs d'exécution. Dans la pratique, chaque produit commercialisé propose son propre formalisme qui peut être transformé en XPDL pour assurer la migration de la définition de processus entre WfMS.

Nous pouvons citer le WebSphere MQ Workflow de l'IBM, le BEA WebLogic Integration de BEA et BizTalk Orchestration de Microsoft comme exemples de systèmes de *Workflow*.

Le *workflow* est couramment utilisé en B2B, notamment pour l'exécution de transaction de longue durée et le monitorage.

1.4 Services Web

Dans cette section, nous décrivons plus précisément les services Web. Nous en profitons pour mettre en évidence la définition, les acteurs et les technologies des services Web.

1.4.1 Services Web : définitions et acteurs

Dans cette section, nous allons présenter les Services Web, en répondant aux questions suivantes : Qu'est-ce qu'un Service Web ? Pourquoi les Services Web ? Par qui ont-ils été conçus et normalisés ?

1.4.1.1 Qu'est-ce qu'un Service Web ?

Pour savoir ce qu'est un Service Web, nous avons besoin de définir le terme « Services Web ». Nous définirons donc les termes « service » et « Web », puis nous présenterons les définitions plus courantes utilisées pour identifier ce qu'est un Service Web.

Un service est une ressource abstraite qui représente des possibilités d'accomplir des tâches qui assurent une fonctionnalité cohérente du point de vue des entités fournisseur et demandeur. Pour être employé, un service doit être réalisé par un agent fournisseur concret [211]. La notion de service existait déjà avant l'apparition des Services Web : elle est en fait utilisée depuis des années par DCE (*Distributed Computing Environment*) de l'OSF, par CORBA de l'OMG, par Java RMI de Sun, et par DCOM (*Distributed Component Object Model*) de Microsoft. Cependant, la notion de service a pris une importance sans précédent avec l'apparition des Services Web et de l'architecture orientée services (SOA - *Service Oriented Architecture*) [209].

« Le Web ou *World Wide Web* est l'un des services majeurs de l'Internet. Le Web se compose de serveurs Web connectés à l'Internet et stockant des pages Web. Celles-ci sont des documents multimédia qui contiennent du texte, des images, des animations et des vidéos. L'Internet est le plus grand réseau du monde, et il se compose de plus de 100 millions d'ordinateurs dans plus de 100 pays et fait appel à des efforts commerciaux, académiques et gouvernementaux » (Cf. TechEncycopledia - www.techweb.com/ency-clopedia). Au fil du temps, malgré les différences techniques, les termes « Web » et « Internet » sont devenus presque synonymes, même dans le monde académique. Il est courant d'utiliser « Internet » et « Web » pour faire référence à l'un ou à l'autre. Les technologies qui supportent le Web sont HTTP, HTML, et plus récemment XML.

Le terme « Services Web » est souvent utilisé de nos jours, mais pas toujours avec la même signification. Un service Web est le plus souvent vu comme une application accessible à d'autres applications sur le Web. Ceci est une définition très ouverte, parce qu'elle permet de considérer toute chose qui utilise une

URL (*Uniform Resource Locator*) comme un service Web, par exemple les scripts CGI (*Common Gateway Interface*). Elle peut aussi se référer à un programme accessible sur le Web avec une API stable, publié avec des informations additionnelles sur un service d'annuaire, comme CORBA et le *Trading Service*.

Une définition plus spécifique est fournie par le consortium UDDI qui a défini un service Web « comme une application métier modulaire et formant un tout qui a des interfaces basées sur des standards, orientée Internet, et ouverte » [132].

Cette définition est plus précise, parce qu'elle met l'accent sur les standards de l'Internet et l'interface ouverte qui permet les invocations des services sur Internet. Cependant, cette définition n'est pas encore suffisamment précise parce qu'elle ne fournit pas la signification d'une application métier modulaire et formant un tout.

Des définitions plus précises des Services Web sont proposées par le W3C. Dans [203], le W3C définit un service Web comme « une application logicielle identifiée par une URI (*Uniform Resource Identification*), dont les interfaces et les liaisons (*bindings*) sont définies, décrites et découvertes comme des objets XML. Un service Web supporte les interactions directes avec d'autres agents logiciels en utilisant le passage de message basé sur XML via les protocoles de l'Internet ». Cette définition est plus précise que les précédentes, et elle met en évidence la façon d'identifier un service (en utilisant des URI), les éléments de base pour rendre l'interopérabilité possible (en utilisant une interface et des liaisons (*bindings*) basés sur XML), les interactions au travers de protocoles basés sur XML.

Cependant, cette définition met en évidence uniquement le standard XML, donc n'importe quelle application orientée Internet qui utilise les technologies basées sur XML est aussi un Service Web.

Dans [209], le W3C définit un service Web comme « un système logiciel 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, i.e. 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 (*HyperText Transfer Protocol*) avec une sérialisation XML en même temps que d'autres normes du Web » [209].

Dans cette dernière définition, le W3C met en évidence les technologies et leurs rôles pour mettre en œuvre un service Web. Cette définition ne mentionne pas la découverte de services Web. A priori la découverte peut être faite de différentes manières, par exemple en utilisant le moteur de recherche « Google » pour trouver un fournisseur de service Web et ainsi les services recherchés. Cependant, ceci est insuffisant et ne fournit pas une façon normalisée de recherche de services. Dans ce cas, UDDI (*Universal, Description, Discovery and Integration*) est une réponse plus adaptée pour enregistrer les informations de fournisseurs de services Web et leurs services [132].

Dans [8], « les Services Web fournissent une manière d'exposer les fonctionnalités d'un système d'information et de le rendre disponible en utilisant les technologies standard du Web. L'utilisation de technologies standard réduit les problèmes liés à l'hétérogénéité, et est la clé pour faciliter l'intégration des applications. De plus, les Services Web fournissent naturellement le support nécessaire pour de nouveaux paradigmes de calcul et pour de nouvelles architectures, tels que le traitement orienté service (SOC - *Service Oriented Computing*) et l'architecture orientée service (SOA - *Service Oriented Architecture*) ». Cette définition mentionne seulement les standards du Web, mais ne les précise pas.

Une définition plus précise des Services Web est fournie par le dictionnaire Webopedia (http://-www.webopedia.com). Il définit un service Web comme « une manière standardisée d'intégration des applications basées sur le Web en utilisant les standards ouverts XML, SOAP, WSDL, UDDI et les protocoles de transport de l'Internet. XML est utilisé pour représenter les données, SOAP pour transporter

les données, WSDL pour décrire les services disponibles, et UDDI pour lister les fournisseurs de services et les services disponibles ».

Dans cette thèse, nous retenons cette dernière définition pour identifier un service Web, car elle met l'accent sur les technologies et leur interdépendance.

Ces technologies standard sont interdépendantes, même si elles ont été conçues de façon indépendante. La figure 1.12 présente ces technologies des Services Web en mettant en évidence leur interdépendance.

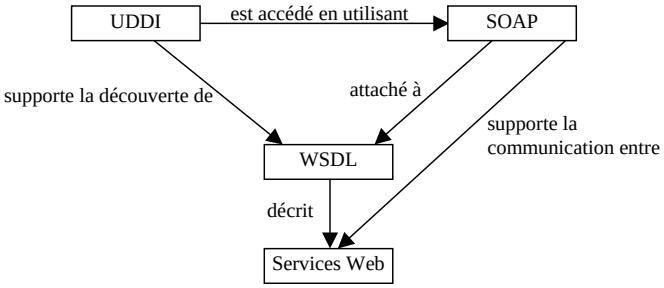


Figure 1.12 – Les principales technologies des Services Web et leurs relations

La figure 1.13 présente l'utilisation de technologies de services Web par les acteurs fournisseur et demandeur.

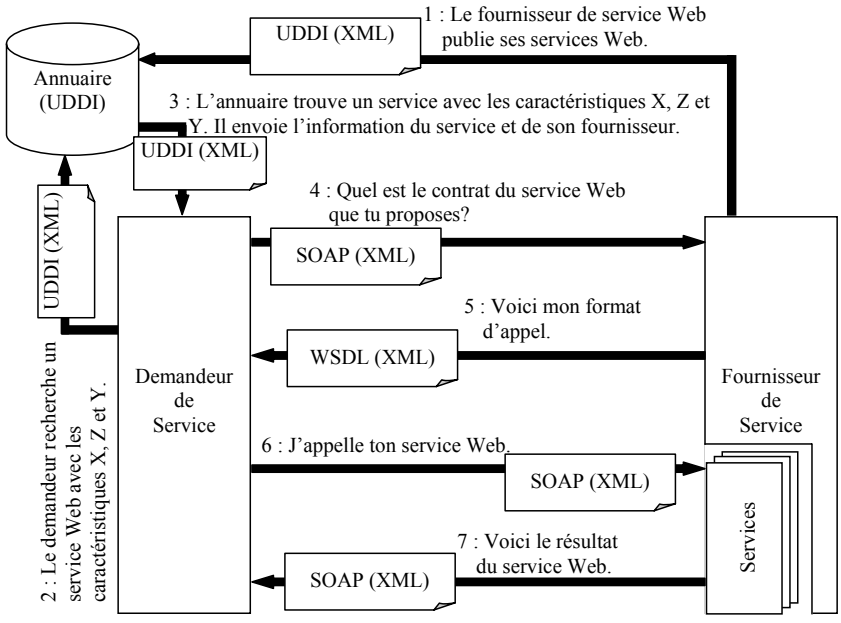


Figure 1.13 – Les technologies des Services Web et les principaux acteurs

Selon la figure 1.13, les acteurs demandeur de service, fournisseur de service et annuaire maintiennent une relation par les biais des technologies de Service Web :

1. Le fournisseur de service publie ses Services Web sur l'annuaire en utilisant UDDI.

2. Le demandeur recherche un service Web avec les caractéristiques X, Z et Y.
3. L'annuaire trouve un service avec les caractéristiques X, Z et Y, et envoie les informations sur le fournisseur du service et sur le service.
4. Le demandeur de service demande le contrat du service Web au fournisseur.
5. Le fournisseur envoie le contrat du service (WSDL).
6. Le demandeur de service appelle le service Web selon son contrat (appel en SOAP).
7. Le service Web retourne le résultat de l'appel (SOAP).

Les standards SOAP, WSDL et UDDI constituent la première génération de Services Web. D'autres standards plus récents comme *WS-Coordination* [42], *WS-Security* [136], *WS-Atomic Transaction* [41] et BPEL4WS (*Business Process Execution Language for Web Services*) [9] constituent la deuxième génération de Services Web.

1.4.1.2 Pourquoi les Services Web ?

Les Services Web ont été conçus pour répondre en premier lieu à des problèmes d'interopérabilité sur l'Internet, que les intergiciels conventionnels comme CORBA et DCOM ont été incapables de résoudre de façon satisfaisante.

Ces intergiciels conventionnels ont été développés à une époque où les systèmes distribués étaient limités à un réseau local ou privé d'une entreprise, d'un gouvernement ou d'une université. Ils ont été conçus pour répondre à des exigences spécifiques et dans un contexte bien limité. L'idée de créer un intergiciel assez robuste pour s'adapter à n'importe quelle situation ou problème a été mise en place dans la création de CORBA. Par exemple, quand l'Internet a été disponible pour l'utilisation civile, CORBA a été rapidement adapté pour travailler sur TCP/IP et le *tunneling* HTTP. Techniquement, ceci a été possible grâce au protocole GIOP/IIOP. Cependant, il s'agit d'une adaptation et non d'une solution faite pour prendre en charge le contexte de l'Internet. De plus, CORBA n'avait pas de consensus universel. Microsoft a développé et stimulé l'utilisation de DCOM. Des passerelles entre CORBA et DCOM ont été proposées, mais elles introduisaient plus de problèmes que de solutions. Ainsi, pendant les années 90, on est passé de la période *middleware* à la période *middleWar*, où une compétition s'est déroulée entre CORBA et DCOM sans avoir ni gagnant ni perdant.

A la fin des années 90, les acteurs du monde informatique se sont aperçus que ni CORBA ni DCOM ne convenaient pour supporter les systèmes distribués sur l'Internet. Plusieurs initiatives ont donc démarré pour créer une nouvelle solution plus adaptée à l'Internet. En fait les acteurs du monde informatique avaient envie de supporter l'interopérabilité de façon uniforme et extensible.

Ils se sont aussi rendus compte qu'un nouveau formalisme pour représenter l'information n'était pas suffisant pour développer les applications dans le contexte de l'Internet : il fallait aussi changer de paradigme. Ainsi le paradigme service a été davantage développé et expérimenté pour prendre en charge les exigences de l'Internet. Il fallait concevoir des solutions qui puissent être intégrées aux standards de l'Internet, comme HTTP et HTML, et/ou utilisées avec eux.

Parmi ces initiatives, nous citons ActiveXML [3] [4], XML-RPC [220], SOAP (*Simple Object Access Protocol*) [204], WSDL (*Web Services Description Language*) [206] et UDDI (*Universal, Description, Discovery and Integration*) [192]. Toutes ces initiatives sont fondées sur XML comme norme pour représenter l'information. De plus la notion de service est présente dans l'ensemble des solutions proposées. Les Services Web sont le résultat de la convergence de ces initiatives.

Fonction	**Java RMI**	**CORBA**	**XML-RPC**	**Services Web**
Description	Interfaces en Java	IDL-CORBA	-	WSDL
Annuaire	+/- rmiregistry	CORBA Services	-	UDDI
Protocole de Communication	JRMP	GIOP	Protocole XML-RPC	SOAP Message
Représentation de donnée	Sérialisation	CDR	XML	XML
Protocole de transport	RMI	HTTP	HTTP	HTTP

Table 1.1 – Un comparatif entre les Services Web et les autres intergiciels

Au début, plusieurs mythes sont nés autour de services Web, mais il ne faut pas penser que les services Web sont la solution la plus adaptée pour n'importe quel contexte [7]. Dans le contexte de l'Internet, les Services Web présentent plusieurs avantages par rapport aux autres intergiciels. Pour dresser un comparatif plus précis, nous comparons la pile de standards de Java RMI, CORBA et XML-RPC avec la pile de Services Web conforme au tableau 1.1.

Java RMI est un intergiciel spécifique à la plate-forme Java. Les interfaces sont définies en Java. L'annuaire est très limité et se résume à un service de nom. Le protocole de communication JRMP (*Java Remote Method Protocol*), la représentation d'informations et le protocole de transfert sont spécifiques à Java.

En fait, Java RMI est une adaptation de RPC au monde Java. Dans [96], l'auteur présente une étude détaillée de performance entre Java RMI et les Services Web. Cette étude a démontré que l'utilisation de Java RMI est 8.5 plus performante que l'utilisation de services Web sur un réseau local. Cependant, l'auteur remarque que sur l'Internet, l'utilisation de Java RMI demande souvent une technique de *tunneling* comme HTTP-to-port, HTTP-to-CGI et HTTP-to-Servlet. L'effort pour déployer et configurer Java RMI et les composants de *tunneling* est plus significatif que dans le cas de services Web. De plus, Java RMI et les techniques de *tunneling* ont présenté une performance inférieure aux services Web. L'utilisation de Java RMI et HTTP-to-port a été quatre fois moins performante que les services Web. Et l'utilisation de Java RMI et HTTP-to-servlet a été trois fois moins performante que les services Web. Selon les résultats obtenus, l'auteur conseille l'utilisation de services Web au détriment de Java RMI avec la technique de *tunneling*, et il recommande l'utilisation de Java RMI (sans *tunneling*) pour les scénarios qui n'ont pas des problèmes avec les pares-feu ou qui demandent plus de performance que d'interopérabilité. Dans [199], l'auteur a montré que, dans certains contextes et avec l'utilisation de techniques d'optimisation, les services Web peuvent être plus performants que Java RMI (sans *tunneling*).

CORBA peut supporter plusieurs types d'applications distribuées, mais n'est pas efficacement adaptable à l'Internet. Il définit la notion de service, ainsi qu'une description d'interface indépendante de la plate-forme, c.-à-d. IDL-CORBA. Le protocole GIOP est un protocole conçu pour être adapté à d'autres protocoles spécifiques au domaine. CDR est une représentation binaire d'information. Ceci rend l'interopérabilité réalisable, mais sans extensibilité. IIOP est le protocole de transport défini dans GIOP pour prendre en charge le TCP/IP, qui a mis longtemps à être standardisé et utilisé comme tel. D'une part, à travers les services CORBA, un annuaire pour les services peut être fourni. Cependant, le *naming* et *trading services* ont été très peu implémentés. D'autre part, cet annuaire est très limité parce qu'il ne prend pas en charge la sémantique de propriétés et de la description du service.

Dans [75], les auteurs présentent un cas d'étude qui démontre l'intérêt et la nécessité de faire cohabi-

ter les Services Web et CORBA. De plus, ils remarquent que les services Web représentent un intergiciel pour intergiciel, c.-à-d. un intergiciel universel qui permet l'intégration de différents intergiciels afin d'assurer l'interopérabilité. Ainsi, la coexistence entre services Web et CORBA peut être justifiée dans certaines situations. Dans d'autres contextes, une seule des deux technologies peut être plus adaptée et suffisante pour supporter le développement des services.

De plus, les services Web présentent des avantages significatifs par rapport à CORBA. Le WSDL contient deux parties, l'une abstraite et l'autre concrète. La partie abstraite contient le contrat du service, et la partie concrète contient les liaisons (*bindings*) et la localisation du service. IDL-CORBA ne fournit par contre que la partie abstraite, c.-à-d. le contrat du service.

Les types de données codifiées en IDL-CORBA sont prédéfinis, bien qu'existe la possibilité de créer des types complexes (par exemple, structures, unions) et des types dynamiques avec le constructeur `any`. Par opposition, WSDL permet la description de types qui ne sont pas prédéfinis dans sa spécification, ceci grâce à l'extensibilité de XML et les schémas XML.

SOAP est un protocole de communication plus riche que IIOP. Comme SOAP est basé sur XML, il permet la création de types de données plus flexibles et non prédéfinis. De plus, SOAP peut être étendu pour répondre à des nouvelles exigences (par exemple, sécurité, transaction) sans modifier la normalisation de SOAP. En fait, SOAP est plus simple pour être adapté aux besoins d'un contexte.

Ainsi comme les autres technologies de services Web, UDDI peut être étendu pour répondre à des nouvelles exigences. Par rapport à CORBA, UDDI fournit en plus la possibilité de classer un service par ses caractéristiques techniques.

XML est un mécanisme plus riche que CDR pour représenter des données. XML a été conçu pour être extensible et compréhensible par les humains et aussi par les ordinateurs. En contrepartie, CDR est une représentation binaire qui n'est pas extensible et ni compréhensible par les humains. Il faut souligner aussi que l'utilisation de XML permet une plus grande interopérabilité et extensibilité que CDR. En contrepartie, l'utilisation de XML exige un traitement plus élaboré que CDR, ce qui occasionne un impact sensible dans la performance des services Web.

XML-RPC est seulement un RPC basé sur XML et HTTP. De plus, il est très limité pour supporter des applications complexes sur l'Internet.

Les services Web sont la plus récente technologie utilisée pour supporter le développement de systèmes d'information distribués, en particulier les applications B2B sur l'Internet. Aujourd'hui, ils semblent être la solution la plus adaptée pour assurer l'interopérabilité sur l'Internet.

Les services Web sont basés sur des technologies standardisées, ce qui réduit l'hétérogénéité et fournit un support pour l'intégration d'applications. De plus, les services Web sont le support pour de nouveaux paradigmes, comme le traitement et l'architecture orientée service. En fait, le traitement orientée service existe depuis longtemps, mais c'est la première fois qu'il atteint un niveau d'importance significatif.

Pour résumer, les Services Web ont plusieurs avantages comme l'utilisation de standards universels, l'indépendance de plate-forme, un environnement universel pour les systèmes d'information distribués, l'utilisation de plusieurs protocoles de transfert (par exemple HTTP, SMTP et FTP), le codage des messages en XML, un comportement compatible aux pare-feu, une facilité d'adaptation aux systèmes du passé (*legacy systems*), et la localisation par URI (*Uniform Resource Identification*).

Nous ne comparons pas les Services Web aux moniteurs transactionnels, MOM et *workflow*, car ils sont plutôt complémentaires que concurrents.

1.4.1.3 Qui a conçu et normalisé les Services Web ?

Les services Web ont été conçus par différentes institutions, mais nous pouvons restreindre les concepteurs en citant les organismes qui font leur standardisation et les entreprises qui les soutiennent.

Les plus importants organismes qui font la standardisation des Services Web sont le W3C (*World Wide Web Consortium*), l'OASIS (*Organization for the Advancement of Structured Information Standards*) et WS-I (*Web Services Interoperability Organization*). Le W3C est responsable du développement de plusieurs standards, parmi eux XML, WSDL, SOAP et WSA. L'OASIS est responsable du développement de standards comme UDDI, WS-Security et plus récemment WSBPEL[3] (*Web Services Business Process Execution Language*). L'organisation WS-I est un type d'organisation différente. Elle prépare des outils et des directives pour aider les développeurs à créer les logiciels en conformité avec les standards des Services Web. La WS-I délivre plusieurs produits, parmi eux des profils, des logiciels de démonstration et des outils de test. Le WS-I Basic Profile représente une avancée en Services Web [221]. Il fournit un guide essentiel pour assurer l'interopérabilité entre les implémentations des Services Web. Cette spécification est sans précédent et démontre l'importance de faire des produits vraiment interopérables, ce qui n'avait pas été vérifié à la création de CORBA.

Parmi les entreprises qui participent à la création de standards de services Web, nous pouvons citer IBM, Microsoft, Ariba, DevelopMentor, BEA Systems, Hewllet-Packard, Sun Microsystems, SAP, Canon, Xerox, Oracle et Intel.

Un des facteurs qui a contribué au succès des Services Web a été l'entente des principaux acteurs. En fait, c'est la première fois que les grandes entreprises d'informatique comme IBM, Microsoft et Sun sont d'accord pour utiliser et soutenir un standard en commun.

1.4.2 XML (*Extensible Markup Language*)

XML est un métalangage de représentation de données. Il définit un ensemble de règles de formatage pour composer des données valides. XML est une mise en forme de texte simple et très flexible dérivée de SGML (*Standard Generalized Markup Language*) (ISO 8879) [208].

XML est implémenté en utilisant un ensemble d'éléments. L'avantage de XML est que ces éléments ne sont pas prédéfinis dans la spécification. Ils peuvent être adaptés pour représenter les données d'un domaine spécifique. Un ensemble d'éléments ayant des points en commun sont regroupés en un vocabulaire. L'exemple 1.1 montre un fragment d'un document XML.

Exemple 1.1 – Un exemple de document XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE computerScience SYSTEM "computerScience.dtd">
<computerScience>
  <discipline id="ParallelDistributedProgramming">
        <professor>Denivaldo Lopes</professor>
        <language>French</language>
  </discipline>
  <discipline id="Database">
        <professor>Slimane Hammoudi</professor>
        <language>French</language>
  </discipline>
</computerScience>
```

[3] WSBPEL est le nom attribué à la même spécification BPEL4WS [9] après la normalisation par l'OASIS.

Dans cet exemple, la première ligne montre que le document est du type `xml`, la version du `xml` est `1.0`, et le type de codage utilisé pour représenter les caractères est `ISO-8859-1`.

La deuxième ligne présente le type de document `computerScience` et le langage utilisé pour formaliser le vocabulaire du document `computerScience.dtd`.

Les autres lignes présentent des éléments qui sont conformes au vocabulaire défini en `computerScience.dtd`. Dans ce cas, le DTD (*Document Type Definitions*) [208] est le langage défini pour spécifier le vocabulaire.

Rapidement, XML est devenu un standard reconnu de facto pour la représentation de données. En fait, XML est devenu une famille très large de standards variés. La figure 1.14 montre les plus importants standards de la famille XML utilisés pour supporter les Services Web.

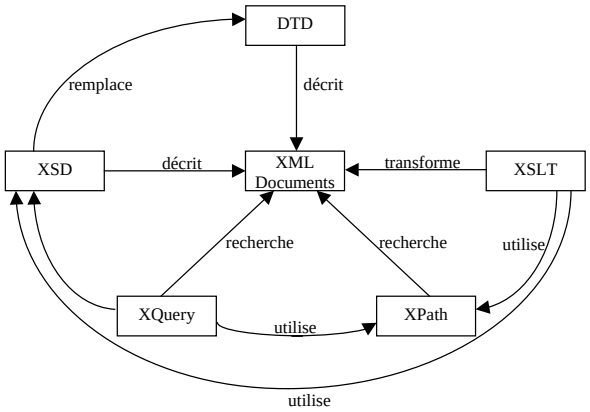


Figure 1.14 – Les spécifications XML

XML est au cœur des Services Web. Il est utilisé dans tous les standards des Services Web, soit WSDL, UDDI et SOAP. Parmi les spécifications XML, nous soulignons :

- XSD (*XML Schema*) : est un langage qui sert à décrire formellement un vocabulaire [212].
- XSLT (*Extensible Stylesheet Language Transformations*) : est utilisé pour transformer un document XML basé sur un certain schéma en un autre document XML qui peut être un document lui-même basé sur un autre schéma [207].
- XPath (*XML Path Language*) : fournit une syntaxe d'expressions utilisées pour créer des chemins de localisation [213].

1.4.3 WSDL (*Web Services Description Language*)

WSDL fournit un modèle et un format XML pour décrire des Services Web. WSDL permet de séparer la description de la fonctionnalité abstraite d'un service des détails concrets d'une description de service, c.-à-d. la séparation de « quelle » fonctionnalité est fournie de « comment » et « où » celle-ci est offerte [206].

Un document WSDL est composé essentiellement de définitions. Chaque définition est composée d'interfaces[4], messages, liaisons (*bindings*) et services. L'exemple 1.2 présente la structure fondamentale d'un document WSDL.

[4]Le `portType` a été renommé `Interface` à partir de WSDL 1.2.

Exemple 1.2 – La structure fondamentale d'un document WSDL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<definitions>
        <types>
                ...
        </types>
        <message name=>
                ...
        </message>
        <interface name=>
                ...
        </interface>
        <binding name=>
                ...
        </binding>
        <service name=>
                ...
        </service>
</definitions>
```

WSDL est composé d'une partie abstraite et d'une partie concrète. Les types, les messages et les interfaces constituent la partie abstraite, les liaisons (*bindings*) et les services constituent la partie concrète.

1.4.4 UDDI (*Universal, Description, Discovery and Integration*)

UDDI fournit la définition d'un ensemble de services qui permettent la description et la découverte (1) des entreprises, des organismes, et d'autres fournisseurs de Services Web, (2) des Services Web qu'ils rendent disponibles, et (3) des interfaces techniques qui peuvent être utilisées pour accéder à ces services [192].

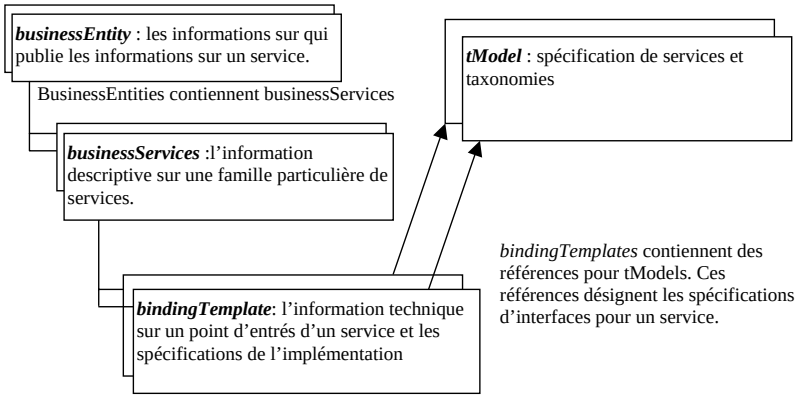


Figure 1.15 – La structure des données d'un registre de UDDI (fragment)

La figure 1.15 présente un fragment des structures de donnés de UDDI. Il est similaire à un annuaire téléphonique. Il présente donc des structures similaires aux pages blanches (i.e. `Business Entity`), pages jaunes (i.e. `Business Service`) et pages vertes (i.e. `Binding Template`). Les pages blanches sont utilisées pour trouver un service par le contact, nom et adresse du fournisseur. Les pages jaunes sont utilisées pour trouver un service par une taxonomie standardisée. Les pages vertes sont utilisées pour trouver un service par les caractéristiques techniques demandées.

1.4.5 SOAP (*Simple Object Access Protocol*)

SOAP fournit une définition des informations représentées en XML qui peuvent être utilisées pour échanger des informations structurées et typées entre les participants dans un environnement distribué et décentralisé [204]. SOAP est un protocole indépendant de toute plate-forme et de tout langage de programmation.

La spécification de SOAP établit un format de message standardisé qui consiste en un document XML capable d'être réutilisé pour travailler avec RPC ou un mécanisme de message centré sur le document (*document-centric message*). SOAP facilite l'implémentation du modèle de communication synchrone et asynchrone.

SOAP définit un protocole de communication structuré qui contient les éléments suivants :

- `Protocol headers` : les en-têtes de protocoles (e.g. HTTP, SMTP, etc).
- `Envelope` : définit un cadre général pour exprimer le contenu d'un message (i.e. *bodies*) et les en-têtes du protocole SOAP.
- `Headers` : cet élément est optionnel. C'est un mécanisme d'extension qui fournit une manière de passer les informations en messages SOAP qui ne sont pas prises en compte directement par les applications.
- `Body` : est un élément obligatoire. Il contient les informations principales transmises dans un message SOAP.

Le contenu des éléments `envelope`, `headers` et `body` n'est pas défini par les spécifications de SOAP. Ils sont dépendants de l'application. Cependant, les spécifications de SOAP précisent comment manipuler de tels éléments. La figure 1.16 illustre les trois principaux éléments de SOAP.

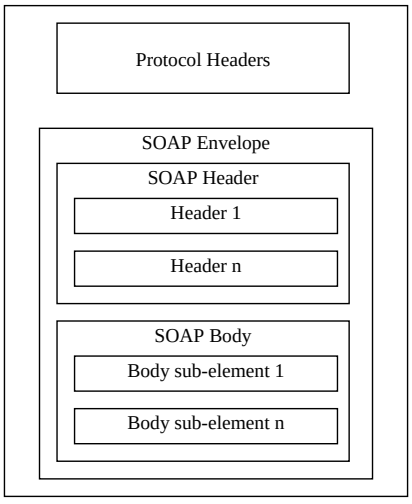


Figure 1.16 – La structure de SOAP

L'avantage de SOAP est qu'il n'est pas lié à un protocole de transfert spécifique. Il peut être utilisé avec plusieurs protocoles comme HTTP ou SMTP (*Simple Mail Transport Protocol*).

1.4.6 SOA (*Service Oriented Architecture*)

SOA est un ensemble de composants qui peuvent être appelés, et dont les descriptions d'interfaces peuvent être éditées et découvertes [211].

D'autres intergiciels ont utilisé le concept de services et ont même eu l'intention de créer une architecture orientée service. Parmi ces intergiciels, CORBA est un des premiers exemples de l'architecture orienté service.

De nos jours, les Services Web fournissent les technologies les plus adaptées pour rendre possible la création de l'architecture orientée service. Le développement des applications en utilisant SOA demande l'adoption d'un projet orienté service, ce qui est différent d'un projet orienté composant. Un projet orienté service se concentre sur les exigences déterminées au niveau de la stratégie et du processus métier, alors que les projets orientés composant se concentrent sur les composants du programme utilisé pour livrer les services. SOA est aussi une architecture de système distribué basée sur les concepts de service et caractérisée par les propriétés suivantes [209] :

- **Vue logique** : un service est une vue logique d'un système.
- **Orienté message** : la communication entre un agent fournisseur et un agent demandeur est définie en termes d'échange de messages.
- **Orienté description** : un service est décrit par des métadonnés.
- **Granularité** : les services communiquent en utilisant un nombre réduit de messages qui sont généralement grands et complexes.
- **Orienté réseau** : les services ont tendance à être utilisés sur un réseau. Cependant, ceci n'est pas une exigence absolue.
- **Plate-forme neutre** : les services communiquent en utilisant des messages codifiés dans une représentation indépendante de la plate-forme. Par exemple, CORBA utilise CDR (*Common Data Representation*) comme une représentation de données indépendante de la plate-forme. Récemment, XML est utilisé comme une représentation de donnée universelle et aussi indépendante de la plate-forme. De plus ce dernier est extensible et représente mieux les informations.

La figure 1.17 illustre les éléments fondamentaux de cette architecture. Un agent fournisseur contient les services. Ces services sont décrits à travers une représentation utilisant des métadonnées, c.-à-d. la description du service. Ensuite, l'agent fournisseur enregistre les informations de ses services dans l'annuaire. Un agent demandeur cherche dans l'annuaire des services selon les critères spécifiques. L'annuaire retourne au demandeur les informations d'un service demandé. L'agent demandeur récupère les métadonnées de ce service et les utilise pour échanger des messages avec le service.

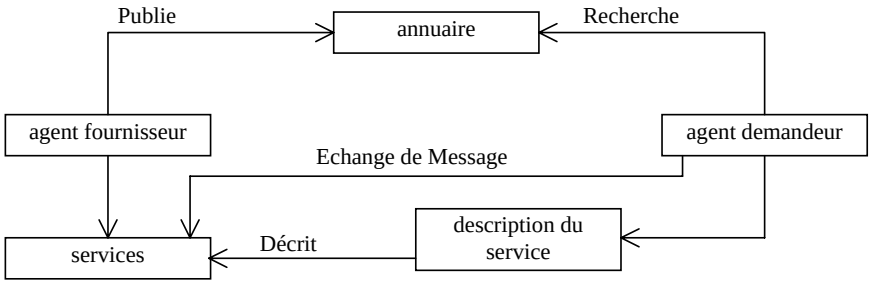


Figure 1.17 – L'architecture orientée service (SOA) (fragment)

Les technologies des services Web peuvent être utilisées pour implémenter un SOA, mais elles

doivent l'être en respectant les propriétés décrites précédemment, c.-à-d. vue logique, orienté message, etc. Dans ce cas, UDDI est utilisé pour publier ou chercher les services, WSDL pour décrire un service, SOAP est le protocole de communication.

En fait, l'ensemble XML, SOAP, WSDL et UDDI est le plus adapté pour être utilisé avec SOA. Cependant, il ne faut pas confondre les services Web avec SOA. Les services Web fournissent le support pour la description, la publication/recherche et l'infrastructure de communication pour les services, alors que SOA décrit comment un système composé de services peut être construit.

1.4.7 Composition des Services Web

La composition des Services Web est un processus par lequel un service Web est créé par l'arrangement d'autres services Web. Ce nouveau service Web est nommé service composite. En d'autres mots, un service composite est un ensemble unifié d'autres services. Dans ce cas, les services qui ont été utilisés sont cachés et réutilisés par le service composite.

La création d'une application distribuée complexe peut être obtenue par la composition de services Web. Cependant, la création d'un service à partir d'autres services est loin d'être une tâche triviale. Pour aider les développeurs à créer des services composites, l'intergiciel de composition de services Web doit fournir une abstraction et une infrastructure qui facilitent la définition et l'exécution d'un service composite.

La composition des services Web a plusieurs similarités avec la technologie de *Workflow* [197]. Les deux ont pour but de spécifier le processus métier par la composition des entités autonomes et de forte granularité. Leur différence réside dans la nature de l'entité. Dans le cas de *Workflow*, les entités sont des applications conventionnelles, dans celui des Services Web, les entités sont des services.

Tous les travaux autour de la composition des Services Web [9] [110] [186] se sont déroulés sur les concepts de base développés pendant des années dans le contexte des systèmes de gestion de *Workflow* (WfMS) [197]. D'une part, ces derniers ont eu un succès limité du fait que les systèmes de support de la composition étaient complexes, difficiles à déployer et à maintenir, composés de différentes applications, et demandaient un effort significatif de développement, surtout quand la logique d'intégration impliquait des systèmes hétérogènes et distribués. D'autre part, la composition des services Web est différente du fait que les services paraissent être des composants plus adéquats. En fait, les services Web ont des interfaces bien définies, et leurs comportements sont « spécifiés » comme partie de l'information fournie par l'annuaire de services. De plus, ils sont basés sur des standards acceptés comme universels. Ceci nous incite à penser que les services Web peuvent être bien adaptés pour la composition. Il semble donc que la composition de service ait une plus grande chance de réussir que les systèmes de gestion de *workflow*.

Un modèle de composition de service peut être relativement complexe. En général, les différentes dimensions d'un modèle de composition de service Web peuvent être groupées comme suit [8] :

- **Modèle de composant** : définit la nature des éléments qui seront utilisés dans la composition.
- **Modèle de l'orchestration** : définit l'abstraction et le langage utilisé pour définir l'ordre dans lequel les services seront appelés. Parmi les possibilités des modèles de l'orchestration, nous pouvons citer les diagrammes d'activité, les réseaux de Petri, les *statecharts*, et les hiérarchies d'activités.
- **Modèle de données et d'accès de données** : définit comment les données sont spécifiées et comment elles sont échangées entre les composants.
- **Modèle de sélection de service** : définit si les services sont reliés statiquement ou dynamiquement. Dans le cas de la composition statique, les services sont sélectionnés durant la conception. Dans le

cas de la composition dynamique, les services sont déterminés et composés en temps d'exécution.
- **Transactions** : définissent quelles sémantiques de transactions peuvent être associées à la composition, et comment cette association est faite.
- **Manipulation des exceptions** : définit comment les situations exceptionnelles qui produisent l'exécution d'un service composite peuvent être gérées, sans conduire à l'arrêt du service composite.

Afin de supporter la composition de services, plusieurs langages de composition de services ont été proposés comme XLANG [186] et WSFL (*Web Services Flow Language*) [110]. Ces langages, bien que peu exploités, ont représenté une avance dans la spécification de processus métier par la composition de services.

XLANG est un dialecte XML développé par Microsoft, dont l'objectif est de permettre la description de processus dans le cadre de services Web. XLANG est une extension de WSDL qui permet la définition de comportements. Il fournit un modèle d'orchestration de services, et de contrats de collaborations entre orchestrations. XLANG a été conçu à partir de la théorie de Pi-calcul.

WSFL est un langage XML permettant la description de composition de services Web comme faisant partie de la définition d'un processus métier. Il a été créé par IBM. WSFL est aussi une extension de WSDL. WSFL est constitué de deux types de compositions : `flowModel` et `globalModel`. Le `flowModel` spécifie le processus exécutable, alors que le `globalModel` spécifie la collaboration entre les participants (*business partners*).

La composition des services Web n'a pas encore été suffisamment développée, mais des signes positifs sont déjà observables, comme la convergence de standards concurrents tels que WSFL et XLANG en BPEL4WS (*Business Process Execution Language for Web Services*) [9].

BPEL4WS définit un modèle et une grammaire pour décrire le comportement d'un processus métier comme la composition des services Web. BPEL4WS contient plusieurs caractéristiques de XLANG et WSFL, ainsi, il est défini comme un langage structuré en blocs et graphes directs. De plus, il est basé sur XML et étend WSDL. BPEL4WS contient des éléments qui rendent la création de processus métiers abstraits et exécutables possibles. Ceci a été soumis à OASIS afin que sa standardisation soit faite [134].

Récemment, plusieurs recherches ont démontré l'intérêt de l'utilisation de sémantique Web et des ontologies pour supporter la composition de services Web, principalement dans le cas de la composition dynamique de services [130] [160] [181] [217]. Cependant, la composition dynamique de services Web est encore en développement et plusieurs problèmes restent encore ouverts.

1.4.8 Les mythes autour des Services Web

Au fil de temps, plusieurs mythes autour des services Web ont vu le jour. Parmi ces mythes [7] :
- **Services Web et les standards** : A priori, les standards de services Web devront être acceptés comme universel, mais ils n'iront pas remplacer complètement les standards conventionnels. Surtout dans le monde des applications B2B, où les applications sont construites en utilisant plusieurs technologies. D'un point de vue évolutif, les services Web sont plus une couche supplémentaire pour permettre aux applications d'interopérer sur l'Internet.
- **Services Web et les applications conventionnelles** : il y a une tendance à considérer que toutes les applications seront bâties sur la technologie de services Web, tel que la réservation de vols, applications qui font le téléchargement automatique de mises à jour et les applications qui envoient les rapports de leurs états d'exécutions. Plusieurs scénarios peuvent être préconisés pour l'utilisation de services Web, mais parmi ces scénarios, l'utilisation de technologies conventionnelles peut

être plus adaptée que l'utilisation de services Web.

- **Connectivité directe entre entreprises** : initialement, les services Web ont été conçus en utilisant le mécanisme de RPC. Dans ce cas, ils ne présentent pas d'avantages significatifs par rapport aux autres intergiciels, mis à part l'interopérabilité. Les services Web sont donc aussi fortement couplés. Ceci est difficile à accepter pour les entreprises, d'autant plus que les composants peuvent être originaires de différentes entreprises. C'est pourquoi la majorité des interactions B2B sont faites en mode asynchrone ou *batch*. Contrairement aux invocations directes, les requêtes et réponses sont envoyées par lots et conduits à travers des files. Ainsi, les éléments de l'interaction (client et serveur) deviennent au tant découplés que possible. En utilisant ce mode de communication, les éléments peuvent être conçus, maintenus, et évolués indépendamment l'un de l'autre. Dans ce scénario, SOAP utilisé conjointement avec RPC au mode asynchrone est essentiel.
- **UDDI et la liaison dynamique** (*dynamic binding*) : un registre UDDI est conceptuellement un serveur de noms et d'adresses. Il a été créé pour être un annuaire pour les services Web. Les informations stockées par ces registres sont destinées à une interprétation humaine, et non pour une interprétation par ordinateurs. De ce fait, la liaison dynamique n'est pas envisageable, parce que les ordinateurs ne sauraient pas automatiquement découvrir un service et construire l'appel en temps d'exécution. Dans ce cas, les services Web conservent le même problème que ses prédécesseurs. La sémantique des services n'est pas prise en compte par UDDI. De plus, une opération et ses paramètres peuvent avoir des significations ambiguës et rendre la liaison dynamique impossible. Dans ce contexte, l'OWL-S [118] et d'autres initiatives essaient de résoudre les problèmes liés à la sémantique. Par ailleurs, les facteurs juridiques et culturels ne sont pas à ce jour abordés, comme les responsabilités des services proposés par un fournisseur, le contrat, le payement, etc. A priori, la liaison dynamique peut être envisagée à l'intérieur d'une entreprise. Cependant, la liaison dynamique entre entreprises reste encore un sujet de recherche et de discussions.
- **Toutes les données seront codées en XML** : dans certains cas, le codage et la transmission de données en XML est indésirable. Comme dans le cas de transmission d'image, de musique, etc. Le codage binaire peut donc s'avérer souhaitable pour des raisons de performance. Ainsi, SOAP avec attachements ou d'autres solutions peuvent devenir essentiels pour des applications qui ont besoin de performance.

1.4.9 Le futur des Services Web

Initialement, les Services Web étaient considérés comme simples et très puissants. Cependant, ils étaient composés d'un ensemble très limité de technologies, XML, WSDL, UDDI et SOAP. Aujourd'hui, plusieurs spécifications ont été ajoutées, telles que WSA [209], WS-Security [136], SAML (*Security Assertions Markup Language*) [135], WS-Federation [16], WS-CDL (*Web Service Choreography Description Language*) [210], BPEL4WS [9] et WS-I Basic Profile [221]. Ces spécifications font des Services Web une technologie (ou ensemble de technologies) très complète, mais aussi très complexe.

Récemment, le nombre de propositions de standards autour des Services Web a significativement augmenté, ce qui démontre leur intérêt [5]. Cependant, le nombre de propositions de standards concurrents est aussi significatif. Dans les entreprises, l'utilisation de deux ou plusieurs standards pour résoudre un même problème rend l'implémentation des Services Web compliquée. Pour l'avenir, ces propositions concurrentes devraient converger vers une seule solution afin que les entreprises puissent développer des

[5] Pour une liste complète de spécifications de Services Web, consultez http://www-106.ibm.com/developerworks/views/webservices/standards.jsp

applications basées sur un standard universel. Le risque pour la troisième génération de Services Web est l'établissement peu probable d'un conflit entre les standards concurrents, similaire à celui des intergiciels CORBA et DCOM dans les années 90.

Au moment de l'apparition des services Web, il y avait un doute important sur leur avenir dans l'e-commerce [190]. Le temps a démontré que les services Web est la technologie la plus adaptée non seulement pour l'*e-commerce* [29] [85], mais aussi pour d'autres applications comme les applications distribuées et mobiles [52] [166]. De plus, ils sont utilisés avec succès dans le monde académique pour enseigner le développement des systèmes informatiques complexes aux étudiants [87].

La composition de services va continuer à être significative pour l'avenir de services Web. En fait, il faut également considérer la sémantique des services afin de réaliser la composition dynamique [183]. Ce qui implique l'évolution de UDDI et WSDL, ainsi que l'évolution de standards pour prendre en considération la sémantique.

L'architecture de services Web (WSA - *Web Service Architecture*) commence à changer la manière de créer des applications qui deviennent conformes à la spécification de SOA [209].

La sécurité doit être plus développée dans le cadre des Services Web pour assurer la création de services plus sécurisés qu'actuellement [136].

Les services Web pourraient devenir une référence pour assurer l'intégration des applications des entreprises (EAI - *Enterprise Application Integration*).

Même dans les domaines très exigeants en terme de performance de traitement d'information, les services Web deviennent de plus en plus utilisés. Par exemple, la recherche scientifique basée sur *grid computing* va utiliser de plus en plus les Services Web pour supporter l'interopérabilité entre les applications [170].

Du point de vue économique, l'IDC (*International Data Corporation*) a constaté que l'investissement en service Web en 2003 a été d'environ 1,1 milliard de dollars et que jusqu'à 2008 l'investissement pourrait attendre les 11 milliards de dollars en projets de logiciels basés sur les services Web. Une autre enquête de Gartner avec la participation de 110 entreprises a indiqué que 54% de ces entreprises travaillent ou ont révélé la perspective de travailler avec les services Web. En 2004, le Forrester Research a faite une enquête avec la participation de 280 grandes entreprises de l'Amérique du Nord dans laquelle il a constaté que 66 projets basés sur les Services Web sont en développement ou en production [105].

L'IDC a réalisé une autre étude sur le marché de service professionnel concernant les services Web au Japon. Selon cette étude, les services Web sont passés en 2003 d'une période d'essai à une période d'utilisation réelle. De plus, l'IDC prévoit que le marché du service professionnel concernant les services Web va atteindre 52,3 milliards de Yen (approximativement, 502,50 millions de dollars) d'investissement jusqu'à 2008 avec une taux de croissance moyenne de 57,4% entre 2003 et 2008 (source Global Information http://www.gii.co.jp/english/id24843_japan_web.html).

Aujourd'hui, nous pensons que les Services Web sont la solution de l'avenir pour implémenter les systèmes distribués sur l'Internet. Cependant, l'histoire a déjà démontré qu'un seul intergiciel n'est pas capable de supporter tous les types d'applications. De plus, ils ne sont pas suffisants pour assurer la pérennité des systèmes informatiques. Par conséquent, dans un futur plus lointain, de nouvelles exigences qui dépassent les capacités de services Web devront apparaître. Ainsi, un autre intergiciel devra être proposé.

1.5 Modèles et langages de modélisation

Avant de présenter les langages de modélisation, nous introduisons le concept de modèle. Selon le dictionnaire Hachette, un modèle est « ce qui est proposé à l'imitation ». Cette définition est assez générale. Celles qui suivent proviennent de la littérature technique de l'informatique et donnent plus d'information sur le terme modèle dans l'informatique.

Un modèle « est une abstraction d'un système physique qui distingue ce qui est pertinent de ce qui ne l'est pas dans le but de simplifier la réalité. Un modèle contient tous les éléments nécessaires à la représentation d'un système réel » [128].

Un modèle est « une représentation d'une partie de la fonctionnalité, de la structure et/ou du comportement d'un système » [138].

Un modèle est « une simplification d'un système créé avec un but spécifique. Le modèle doit être capable de répondre aux demandes à la place du système étudié. Les réponses fournies par le modèle doivent être les mêmes que celles fournies par le système, à condition qu'elles restent dans la limite du domaine défini par le but général du système » [33].

Un modèle est « une description d'un (ou d'une partie d'un) système dans un langage bien défini, c.-à-d. respectant un format précis (une syntaxe) et une signification (une sémantique). Cette description doit être convenable pour une interprétation automatisée par un ordinateur » [102].

Un modèle est « une abstraction d'un système » [69]. Une abstraction est « une description de quelque chose qui omet certains détails non pertinents pour l'objectif de l'abstraction » [138].

Un modèle est « une description d'un système physique avec un certain but, par exemple la description des aspects logiques ou comportementaux d'un système physique pour une certaine catégorie de lecteurs. Ainsi, un modèle est une abstraction d'un système physique qu'il spécifie à partir d'un point de vue (*viewpoint*), c.-à-d. pour une certaine catégorie de *stakeholders*, par exemple les concepteurs, les utilisateurs, ou les clients du système, et en respectant un certain niveau d'abstraction, selon les buts du modèle » [150].

Un modèle est « une simplification de quelque chose qui nous permet de voir, manipuler, et raisonner sur le sujet étudié, et qui nous aide à en comprendre la complexité inhérente » [122].

Dans la littérature, nous trouvons ainsi différentes définitions du terme modèle. Cependant, elles transmettent la même idée, imitation du réel, représentation, simplification, abstraction, et description limitée d'un système. Nous remarquons aussi qu'un modèle est créé avec un but spécifique et dans un contexte particulier. Ainsi, le but et le contexte sont les éléments qui vont diriger les choix faits pendant la création d'un modèle. Un modèle n'est pas une description complète d'un système, mais il doit permettre l'élaboration de raisonnements sur le système étudié dans un contexte limité. Les modèles sont nécessaires justement parce qu'ils nous permettent de simuler les systèmes physiques avant leur construction, en fournissant une manière simple, manipulable et économique de prévoir les éléments d'un système et leurs relations. Le modèle peut aussi être déduit d'un système existant ou d'un système en cours de développement. Ils sont également essentiels pour établir une communication efficace entre les membres d'une équipe et entre les équipes impliquées dans un projet, et pour assurer la robustesse architecturelle du système en conception ou en analyse.

Il y a aussi des différents types de modèles, par exemple les modèles physiques, les modèles mathématiques et les modèles informatiques. Les modèles informatiques sont ceux qui nous intéressent dans cette étude, en particulier les modèles dans le contexte du MDA.

La création de modèles est faite en utilisant un langage bien défini, c.-à-d. avec une syntaxe et une

sémantique spécifiées pour réglementer la création des éléments et leurs relations. Ainsi, un modèle peut être par exemple conçu à travers un langage mathématique ou un langage graphique.

Un langage de modélisation est une spécification formelle bien définie qui contient les éléments de base pour construire des modèles. De plus, il est conçu dans un domaine limité et avec des buts spécifiques, de sorte que nous pouvons considérer l'existence de plusieurs langages de modélisation, chacun étant adapté à un domaine spécifique [51]. Le langage conçu pour créer des modèles est souvent défini comme un métamodèle, c.-à-d. ce qui précède le modèle.

1.5.1 UML (*Unified Modeling Language*)

Dans les années 90, une multitude de méthodes ont été proposées afin de supporter la spécification et la documentation de logiciel (Booch, OMT, OOSE, méthodes orientées objet, et autres). Cette diversité a engendré un problème d'interopérabilité entre les outils de modélisation qui a ralenti l'adoption des méthodes de développement de logiciels basés sur les modèles. Dans ce contexte troublé par différents langages de modélisation, Grady Booch, Ivar Jacobson et Jim Rumbaugh ont proposé le langage unifié de modélisation (UML - *Unified Modeling Language*) [149]. UML a été conçu pour unifier différents langages de modélisation, être le plus général possible et être extensible afin de prendre en charge des contextes non prévus. UML est basé sur les fondements de l'orientation objets, la notion de classe, l'objet, les attributs, les méthodes et les relations entre les classes ou les objets. UML est un langage visuel dédié à la modélisation et non à la programmation, dans le sens d'avoir tous les supports visuels et sémantiques nécessaires pour remplacer les langages de programmation. Cependant, il est un support fondamental pour générer des programmes.

La première version de UML a été publiée en 1997 par l'OMG. Depuis, UML est devenu la référence pour la création de modèles pour l'analyse et la conception de logiciels.

La spécification de UML définit un langage graphique pour visualiser, spécifier, construire et documenter les artefacts des systèmes distribués orientés objets. UML représente une collection des meilleures pratiques d'ingénierie qui ont réussi dans la modélisation de systèmes grands et complexes [149]. La spécification d'UML comporte :

- une définition formelle du métamodèle d'UML, c.-à-d. le langage abstrait pour spécifier les modèles UML.
- Une notation graphique concrète (non normative) pour représenter les modèles UML.
- Un ensemble d'interfaces CORBA pour représenter et gérer les modèles UML. Les outils d'analyse et de conception basés sur UML peuvent utiliser les interfaces CORBA pour créer, accéder et manipuler les modèles UML. De plus, ce format permet l'échange de modèles entre différents outils.
- Un format XMI pour échanger des modèles UML (XMI sera présenté dans la section 1.7).

Dans cette présentation d'UML, nous n'abordons que le métamodèle UML et ses diagrammes.

Le métamodèle d'UML (figure 1.18) est assez riche pour représenter différents points de vue à travers ses diagrammes, lesquels reflètent les caractéristiques statiques ou dynamiques d'un système.

Les diagrammes d'UML représentant les caractéristiques statiques sont :

- **les diagrammes de classes** : expriment de manière générale la structure statique d'un système, en termes de classes et de relations entre ces classes. De plus, ils présentent un ensemble d'interfaces et de paquetages, ainsi que leurs relations.
- **les diagrammes d'objets** : sont des graphes d'instances qui incluent les objets et les valeurs de données. Un diagramme d'objets statiques est une instance d'un diagramme de classes, présente

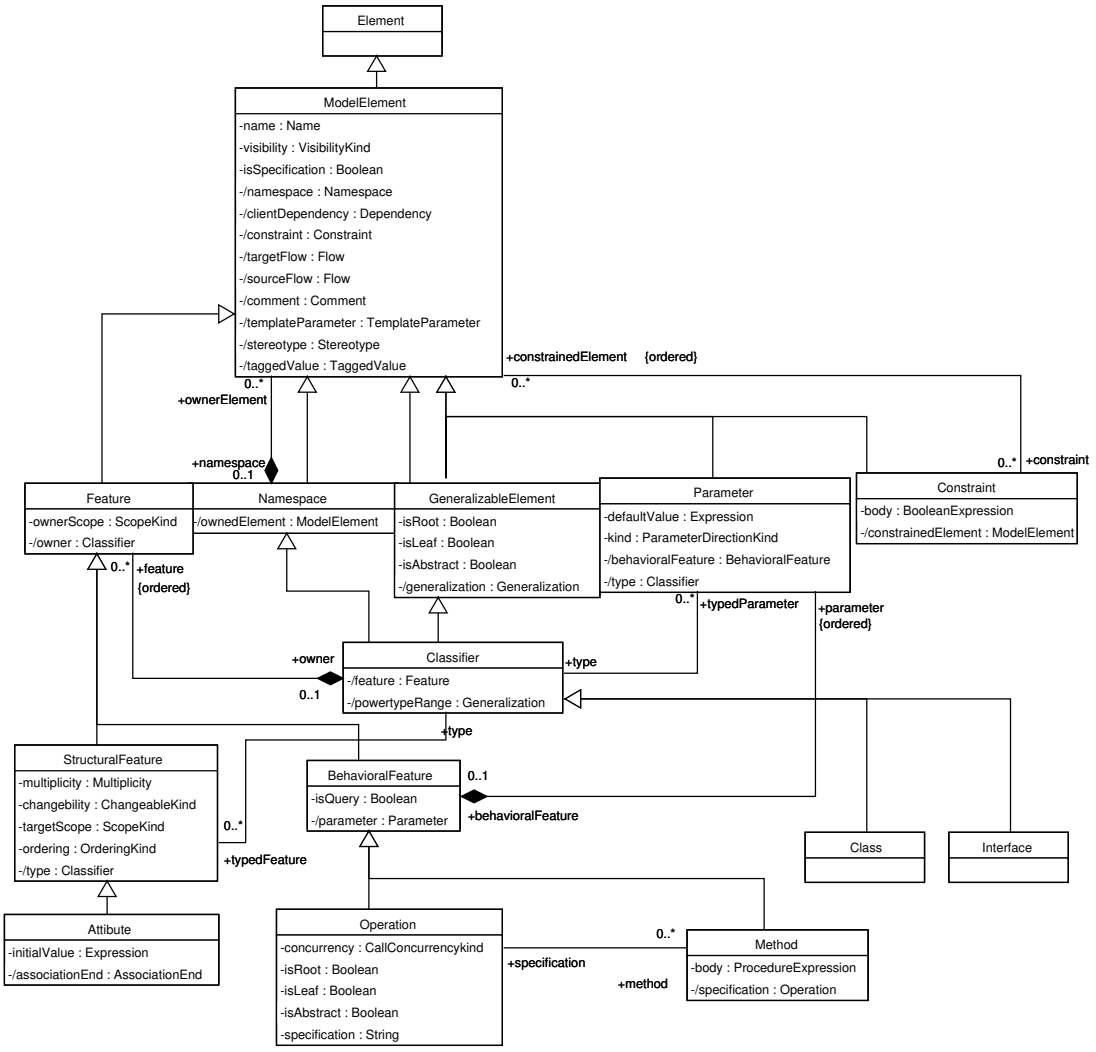


Figure 1.18 – Le métamodèle de UML (fragment)

un *snapshot* de l'état détaillé d'un système à un instant donné.

- **les diagrammes de cas d'utilisation** : représentent les cas d'utilisation, les acteurs, et les relations entre le cas d'utilisation et les acteurs. Ils décrivent, sous forme d'actions et de réactions, le comportement d'un système du point de vue d'un utilisateur.
- **les diagrammes de composants** : présentent les dépendances entre les composants logiciels. Ils incluent les classificateurs (i.e. classes) qui spécifient les composants, et les artefacts qui les implémentent, tels que les fichiers de code source, de code binaire, exécutables ou scripts.
- **les diagrammes de déploiements** : présentent la configuration des éléments de traitement en temps d'exécution, ainsi que les composants logiciels, les processus et les objets qui les exécutent.

Les diagrammes de UML représentant les caractéristiques dynamiques sont :

- **les diagrammes de séquence** : montrent les interactions entre les objets dont la représentation se concentre sur la séquence des interactions selon un point de vue temporel.
- **les diagrammes de collaborations** : présentent l'ensemble des rôles joués par des objets dans un

contexte particulier, ainsi que les liens entre ces objets.

- **les diagrammes d'états-transitions** : représentent les automates d'états finis du point de vue des états et des transitions.
- **les diagrammes d'activités** : sont une variation des diagrammes d'états-transitions dans laquelle les états représentent à la fois la réalisation des actions ou sous activités et à la fois les transitions déclenchées par la finalisation des actions ou sous activités.

Un métamodèle est une spécification utilisée pour créer des modèles conformes à cette spécification, par exemple, le métamodèle de UML. La figure 1.18 présente un fragment du métamodèle de UML.

Parmi les éléments de ce métamodèle, nous citons :

- `Class` : décrit un ensemble d'objets ayant les mêmes attributs, les mêmes opérations et les mêmes relations.
- `Interface` : décrit le comportement visible d'une classe.
- `Attribute` : représente une caractéristique ou un état.
- `Operation` : est un service appartenant à un objet. Elle est décrite par une signature et peut posséder des paramètres et renvoyer un résultat.

UML est un langage général, et a donc été conçu pour prendre en charge une grande variété de contextes. Cependant, même avec cette intention d'être général, UML ne peut pas couvrir tous les contextes et offre ainsi un mécanisme d'extensibilité basé sur les profils, qui est une caractéristique fondamentale.

Un profil permet la personnalisation d'UML pour prendre en charge des domaines spécifiques qui ne peuvent pas être représentés avec UML dans son état original. Le fragment du métamodèle UML correspondant à ce mécanisme d'extension est présenté par la figure 1.19. Le profil est constitué de stéréotypes, de contraintes et de valeurs marquées.

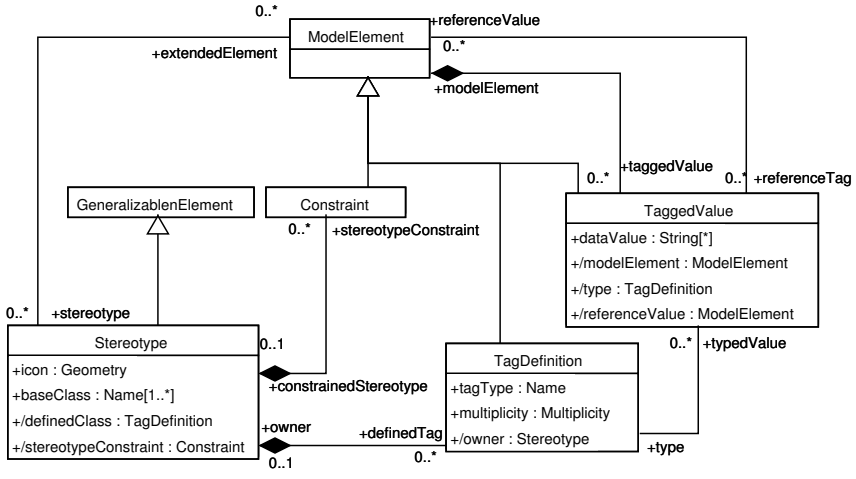


Figure 1.19 – Le mécanisme d'extension de UML

Un stéréotype définit une sous-classe d'un ou plusieurs éléments du métamodèle UML. Cette sous-classe a la même structure que ses éléments de base, mais le stéréotype peut spécifier des contraintes et des propriétés supplémentaires. Les contraintes peuvent être spécifiées non formellement, mais l'utilisation d'OCL (*Object Constraint Language*) [148] est préférable pour créer les contraintes de façon formelle et standardisée. Les propriétés supplémentaires sont définies par des valeurs marquées.

L'OMG a standardisé certains profils :

- le profil CORBA : permet la spécification de sémantiques CORBA IDL en utilisant la notation UML.
- le profil EAI (*Enterprise Application Integration*) : fournit une norme d'échange de métadonnées pour les informations sur les interfaces d'application. Le but est de simplifier l'intégration d'applications [144].
- le profil EDOC : a pour objectif de permettre la conception orientée objet d'un système d'information d'entreprise et d'en dériver une implantation à l'aide de composants métier distribués [145].
- le profil *Testing* : permet la spécification de tests, aussi bien au niveau structurel que comportemental d'un système.
- le profil *Scheduling, Performance and Time* : fournit les concepts nécessaires à la modélisation des systèmes temps réel.

Un grand nombre d'outils permettent de définir et de manipuler des modèles basés sur UML. Parmi les plus connus, nous pouvons citer :

- Poseidon for UML (http://www.gentleware.com).
- Together (http://www.borland.com/together).
- Rational Rose (http://www-306.ibm.com/software/rational).
- Objecteering (http://www.objecteering.com).

1.5.2 EDOC (*Enterprise Distributed Object Computing*)

EDOC (*Enterprise Distributed Object Computing*) est une spécification pour le développement de composants basés sur les systèmes EDOC à travers un *framework* de modélisation [145]. La spécification EDOC est indépendante d'une plate-forme technologique, ce qui permet de réaliser son implémentation sur différents intergiciels, comme Services Web, CORBA, EJB ou DCOM/COM.

Cette spécification présente les métamodèles et les profils UML pour modéliser les applications des entreprises. Elle est composée de l'ECA (*Enterprise Collaboration Architecture*), les patrons, les modèles spécifiques aux technologies et les correspondances vers des technologies.

L'ECA fournit cinq métamodèles et profils :

- CCA (*Component Collaboration Architecture*) : utilise des classes ainsi que des graphes d'activité et de collaboration pour modéliser la structure et le comportement des composants.
- *Entity* : utilise un ensemble d'extensions UML pour modéliser les entités.
- *Events* : est un ensemble d'extensions UML pour modéliser les événements d'un système.
- *Business Process* : compléments de CCA et du modèle de comportement du système.
- *Relationships* : étend les facilités du noyau d'UML pour répondre aux exigences des relations en général, la modélisation du métier et la modélisation du logiciel.

Les patrons fournissent les modèles standards qui peuvent être utilisés pour élaborer les modèles bien définis pour les systèmes EDOC.

D'une part, l'ECA est centré sur la spécification de parties structurelles et comportementales des systèmes EDOC. D'autre part, les modèles spécifiques de la technologie et les correspondances de technologies font partie de la spécification des systèmes EDOC laquelle prend en charge la correspondance entre la spécification d'ECA et un modèle spécifique de la technologie. Cette spécification définit un métamodèle EDOC et un profil UML pour les EJB (*Enterprise JavaBeans*), et un métamodèle pour FCM (*Flow Composition Model*). FCM est un métamodèle d'objectif général qui fait l'abstraction d'un intergiciel fourni par le groupe de produits MQ-Series [145].

Nous nous intéressons particulièrement à la partie CCA de la spécification EDOC. En effet, cette

partie de la spécification présente un métamodèle de composition générale qui résume l'état de l'art sur les composants, la récursivité dans la composition, la spécification de ports de communication entre composants et la notion de protocole pour représenter les dialogues complexes entre composants. La figure 1.20 présente un fragment du métamodèle de EDOC.

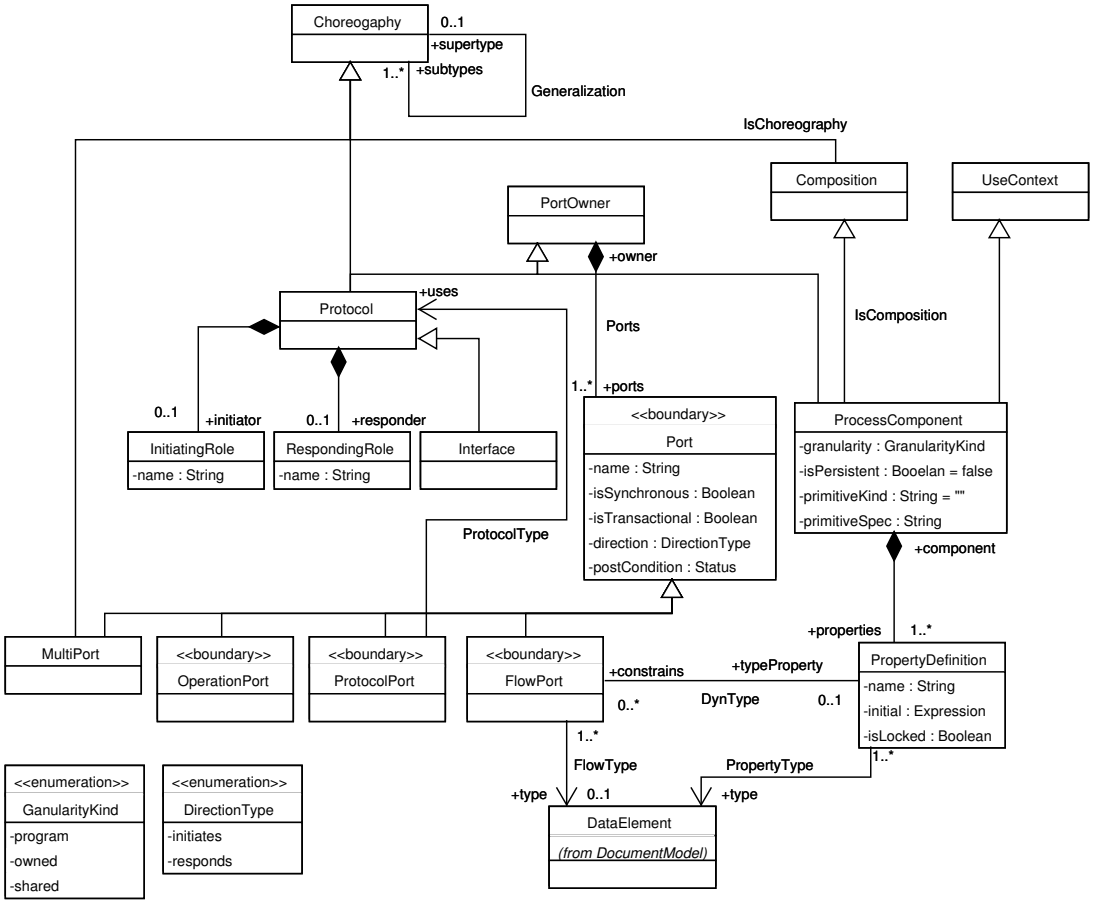


Figure 1.20 – Le métamodèle de la spécification structurelle de EDOC (fragment)

Parmi les éléments de ce fragment du métamodèle EDOC-CCA, nous distinguons :

- `ProcessComponent` : représente le contrat pour un composant qui réalise des actions. Un `ProcessComponent` peut définir un ensemble de `Ports` pour l'interaction avec d'autres `ProcessComponent`. Le `ProcessComponent` définit le contrat externe du composant en termes de ports et de chorégraphie d'activités de ports (qui envoient ou reçoivent des messages ou initialisent sous-protocoles).
- `PortOwner` : contient l'ensemble des `Ports` qui réalisent le contrat du `ProcessComponent`.
- `Port` : définit un point d'interaction entre les `ProcessComponents`. La forme la plus simple d'un port est le `FlowPort`, qui peut produire ou consommer un type de données simples. Les interactions plus complexes entre les composants utilisent un `ProtocolPort`, lequel fait référence à un `Protocol`, c.-à-d. à une conversation complète entre composants.
- `PropertyDefinition` : définit un paramètre de configuration de composants qui peut être un ensemble.

- `InitiatingRole` : représente le rôle du protocole qui envoie un premier message.
- `RespondingRole` : représente le rôle du protocole qui reçoit un premier message.
- `Interface` : représente une interface objet simple. Elle peut contenir des `OperationPorts` représentant les sémantiques de `call-return` ainsi que des `FlowPorts` représentant des opérations `one-way`.

Un certain nombre d'outils permettent de définir et de manipuler des métamodèles basés sur EDOC. Par exemple, Component-X fournit une interface graphique pour créer et éditer des modèles en utilisant la notation CCA (http://www.enterprise-component.com)

1.6 Langages de métamodélisation

EDOC et UML sont des langages de modélisation. Ils ont une syntaxe et une sémantique qui dirigent la création de modèles, ces derniers étant spécifiées par un métamodèle.

Dans la littérature, nous trouvons plusieurs définitions de métamodèle.

Un métamodèle « est un modèle d'un langage de modèles » [63].

Un métamodèle « est un modèle qui définit le langage pour exprimer un modèle » [150].

Un métamodèle utilise d'autres langages connus comme langages de métamodélisation. Chaque langage de métamodélisation correspond à un métamétamodèle, c.-à-d. ce qui précède le métamodèle.

Un métamétamodèle « est un modèle qui définit le langage pour exprimer un métamodèle. La relation entre un métamétamodèle et un métamodèle est analogue à la relation entre un métamodèle et un modèle » [150].

La relation entre un métamétamodèle, un métamodèle, un modèle et une information est bien définie dans l'architecture de métamodélisation (*framework* de métamodèlisation). Cette relation est basée sur l'architecture à quatre niveaux présentée dans la figure 1.21.

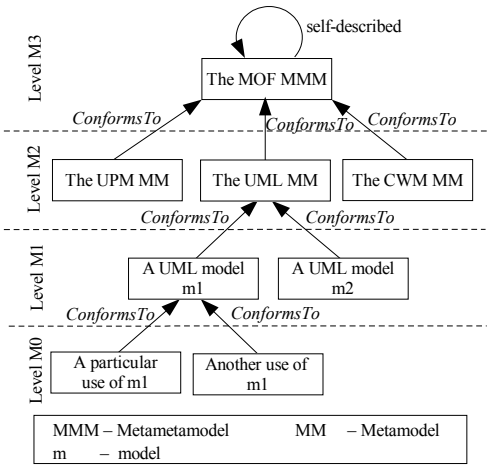


Figure 1.21 – L'architecture à quatre niveaux

Dans ce document, nous définissons que les relations entre le niveau `M3` et `M2`, le niveau `M2` et `M1`, et le niveau `M1` et `M0` sont de type `ConformsTo` (conforme à). Nous pouvons citer comme exemples de langages de métamodélisation (c.-à-d. métamétamodèle) MOF (*Meta Object Facility*) [141] et Ecore [30] [60]. Ils sont tous les deux basés sur les notions de l'orientation objet. Dans ce cas, un objet est

une instance de classe. Ainsi, le terme instance (c'est-à-dire, une occurrence de quelque chose) met en relation une classe et un objet.

Cette architecture présente les niveaux suivants [141] [149] :

- `M3` (métamétamodèle) : le niveau `M3` constitue la base de l'architecture de métamodélisation. La fonction première de ce niveau est de définir le langage pour spécifier un métamodèle. Un métamétamodèle définit un modèle de plus haut niveau d'abstraction que le métamodèle, et il est typiquement plus compact que le métamodèle qu'il décrit.
- `M2` (métamodèle) : un métamodèle est conforme à un métamétamodèle. La fonction première du niveau de métamodèle (c.-à-d. `M2`) est de définir un langage pour spécifier des modèles. Les métamodèles sont typiquement plus élaborés que les métamétamodèles qui les décrivent.
- `M1` (modèle) : un modèle est conforme à un métamodèle. La fonction première du niveau de modèle est de définir un langage pour décrire un domaine d'information (c.-à-d. le niveau des objets).
- `M0` (information) : les objets d'utilisateur sont représentés par un modèle. La première responsabilité des objets d'utilisateur est de décrire un domaine d'information spécifique.

Dans cette architecture, nous remarquons qu'il y a peu de métamétamodèles (par exemple, MOF et Ecore), plusieurs métamodèles (par exemple, UML, EDOC et UEML [193]), un grand nombre de modèles et une infinité d'informations.

1.6.1 Langage de métamodélisation : MOF (*Meta Object Facility*)

Le MOF (*Meta-Object Facility*) a été adopté en 1997 par l'OMG. La spécification de MOF définit un langage abstrait et un *framework* pour la spécification, la construction, et la gestion de métamodèles neutres de technologie. De plus, le MOF définit un *framework* pour l'implémentation d'entrepôts qui contiennent les métadonnées (c.-à-d. modèles) décrites par les métamodèles. Ce *framework* utilise les correspondances de technologies normalisées pour transformer les métamodèles MOF dans des API de métadonnées [141].

La spécification de MOF comporte :

- une définition formelle du métamétamodèle MOF.
- une correspondance de métamodèles MOF vers CORBA IDL qui produit des interfaces IDL pour gérer n'importe quelle métadonnée.
- un ensemble des interfaces IDL-CORBA pour représenter et manipuler métamodèles MOF.
- un format XMI pour l'échange de métamodèles MOF.

À ce jour, les versions stables et normalisées sont la version 1.4 de MOF [141] et la version 1.5 d'UML [149]. Dans ce cas, le métamétamodèle MOF ressemble beaucoup au noyau du métamodèle d'UML, sauf la partie réflexive qui a été renforcée dans MOF. Le métamétamodèle MOF est présenté par la figure 1.22.

Nous pouvons aussi remarquer des petites différences entre le MOF et le noyau d'UML. Par exemple :

- En MOF, `ModelElement` dépend de `ModelElement` (c.-à-d. la réflexivité), alors qu'en UML, ceci n'existe pas. Ainsi, MOF est décrit par lui-même.
- En MOF, `GeneralisableElement` hérite de `Namespace`. Dans ce cas, la relation entre `Classifier` et `Feature` est implicite, parce que le `Classifier` hérite de la relation `Contains` de `Namespace`. Il contient donc un ensemble de `Feature`. En UML, la relation entre `Classifier` et `Feature` est explicite.
- En MOF, l'héritage est possible au travers de la relation `Generalizes`. Ceci met en relation le `GeneralizableElement` avec ses ancêtres (`supertypes`) et ses enfants (`subtypes`) dans un

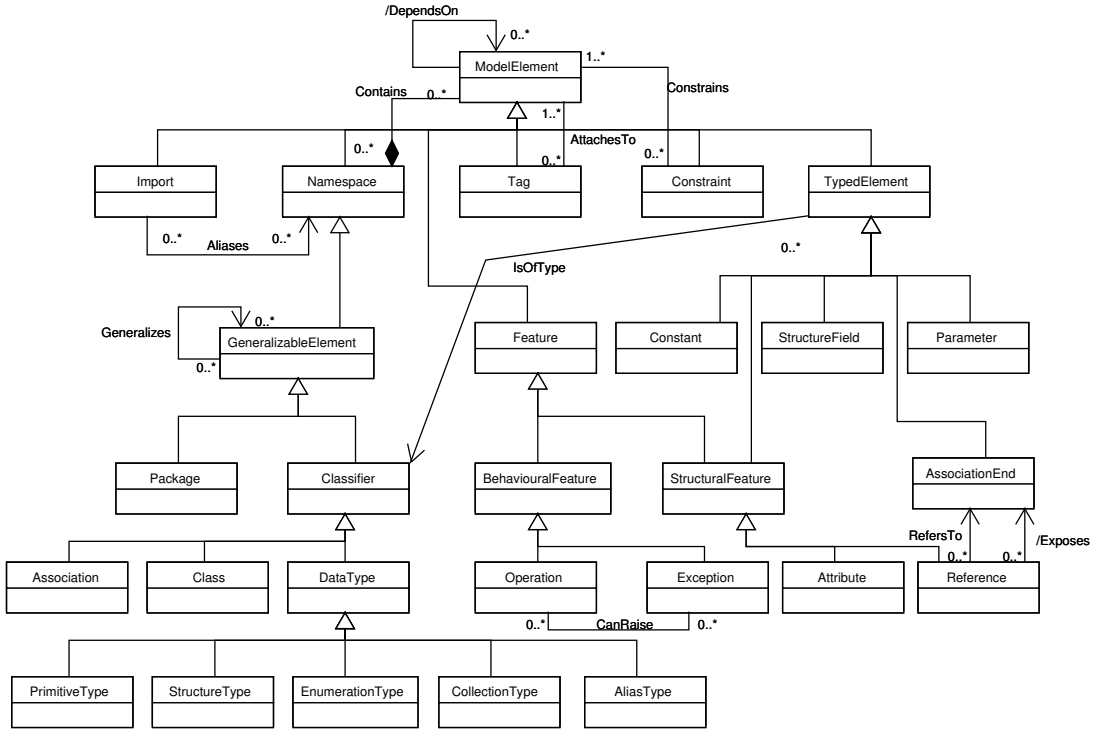


Figure 1.22 – Le MOF *(model package)*

graphe d'héritage d'éléments de modèle. Nous remarquons aussi que le `GeneralizableElement` peut uniquement connaître ses ancêtres : il ne connaît pas ses enfants (la relation est donc unidirectionnelle).

- En UML, l'élément `Reference` n'existe pas alors qu'en MOF, l'élément `Reference` appartient à un `classifier` (`Class`). Ce premier permet à un client `Class` de naviguer directement d'une instance de la `Class` à une autre instance.

Un certain nombre d'outils permettent de définir et de manipuler des métamodèles basés sur le MOF, nous citons par exemple UML2MOF. C'est un outil qui permet de convertir un modèle UML en un métamodèle MOF (http://mdr.netbeans.org/uml2mof/).

1.6.2 Langage de métamodélisation : Ecore

Ecore est un langage de métamodélisation qui fait partie d'EMF (*Eclipse Modeling Framework*) qui est le résultat des efforts du projet Eclipse (*Eclipse Tools Project*). EMF est un *framework* de modélisation et génération de code pour supporter la création d'outils et d'applications dirigées par les modèles [30].

La figure 1.23 présente un fragment du métamétamodèle Ecore [30].

Ce fragment d'Ecore présente les éléments suivants :

- `ENamedElement` : est une généralisation pour `EClassifier` et `ETypedElement`.
- `EPackage` : groupe les classes (c.-à-d. `EClass`) et les types de données (c.-à-d. `EDataType`).
- `EClassifier` : est un élément commun qui est spécialisé en un `EClass` ou `EDataType`.
- `EClass` : est identifié par un nom et il contient des attributs, des références et des opérations.
- `EAttribute` : contient des données qui appartiennent à une classe.

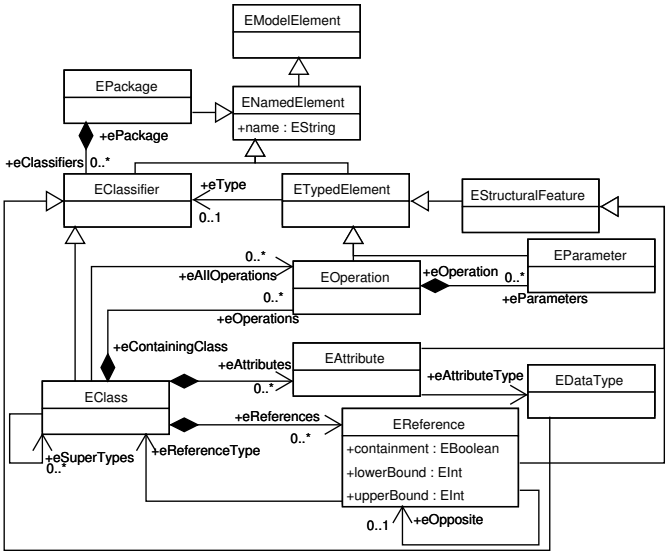


Figure 1.23 – Le métamétamodèle Ecore (fragment)

- `EReference` : permet la création d'associations entre classes.
- `EOperation` : contient le comportement d'une classe.

Un certain nombre d'outils permettent de définir et de manipuler des métamodèles basés sur Ecore, nous pouvons citer par exemple :

- Le plug-in de EMF qui contient un éditeur pour la création de métamodèles en utilisant Ecore (http://www.eclipse.org/emf).
- L'outil Omondo (http://www.omondo.com).

1.7 Le format d'échange normalisé : XMI

XMI (*XML Metadata Interchange*) permet l'échange de modèles sérialisés en XML. XMI si focalise sur les échanges de métadonnées conformes au MOF [151].

L'objectif de XMI est de permettre de sérialiser et d'échanger des métamodèles MOF et des modèles basés sur ces métamodèles sous forme de fichiers en utilisant des dialectes XML. Il permet aussi de sérialiser des métamodèles qui sont créés avec Ecore. Ceci est possible parce que XMI ne définit pas un dialecte XML unique, mais un ensemble de règles qui permettent de créer une DTD ou schéma XML pour différents métamodèles.

Ainsi, les métamodèles conformes à MOF ou à Ecore et leurs modèles respectifs peuvent être portables en utilisant XMI.

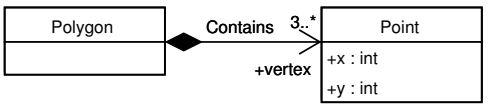


Figure 1.24 – Un simple modèle de polygone

La figure 1.24 présente un simple modèle UML, et l'exemple 1.3 présente sa représentation corres-

pondante en XMI.

Exemple 1.3 – La représentation correspondante en XMI du modèle de Polygone

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML' timestamp = 'Tue
    Nov 23 21:00:33 CET 2004'>
  ***
  <XMI.content>
    <UML:Model xmi.id = 'a1' name = 'model 1' isSpecification = 'false'
 isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
     ***
        <UML:Class xmi.id = 'a4' name = 'Polygon' visibility = 'public'
isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
isAbstract = 'false' isActive = 'false'>
        </UML:Class>
        <UML:Class xmi.id = 'a6' name = 'Point' visibility = 'public'
isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
isAbstract = 'false' isActive = 'false'>
          ***
          <UML:Classifier.feature>
            <UML:Attribute xmi.id = 'a8' name = 'x' visibility = 'public'
isSpecification = 'false' ownerScope = 'instance'>
              ***
            </UML:Attribute>
            <UML:Attribute xmi.id = 'a12' name = 'y' visibility = 'public'
isSpecification = 'false' ownerScope = 'instance'>
              ***
            </UML:Attribute>
          </UML:Classifier.feature>
        </UML:Class>
        <UML:Association xmi.id = 'a15' name = 'Contains'
isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
isAbstract = 'false'>
          ***
        </UML:Association>
        ***
  </XMI.content>
</XMI>
```

1.8 Le développement traditionnel de logiciel et XP (*eXtreme Programming*)

Le développement traditionnel de logiciel (c.-à-d. avant MDA) est souvent dirigé par une approche orientée codage. Un processus typique de l'approche traditionnelle peut être divisé selon les étapes suivantes :

1. La conceptualisation et le rassemblement des exigences (*requirements*).
2. L'analyse et la description fonctionnelle (*analysis*).
3. La conception (*design*).
4. Le codage.
5. Les tests.

6. Le déploiement.

Le problème de l'approche traditionnelle est que les documents et diagrammes créés lors des trois premières étapes perdent rapidement leur valeur quand le codage commence. De plus, quand le système est modifié plusieurs fois, l'écart entre le code, la documentation et les diagrammes augmente [102].

XP (*eXtreme Programming*) est une approche délibérée et disciplinée pour le développement de logiciels. Cette approche est basée sur des règles simples et pratiques [53].

Une caractéristique de XP est l'interaction intensive entre les clients et les développeurs pendant un projet, le client devenant un partenaire actif dans le développement. La figure 1.25 présente le cycle de vie d'un projet XP.

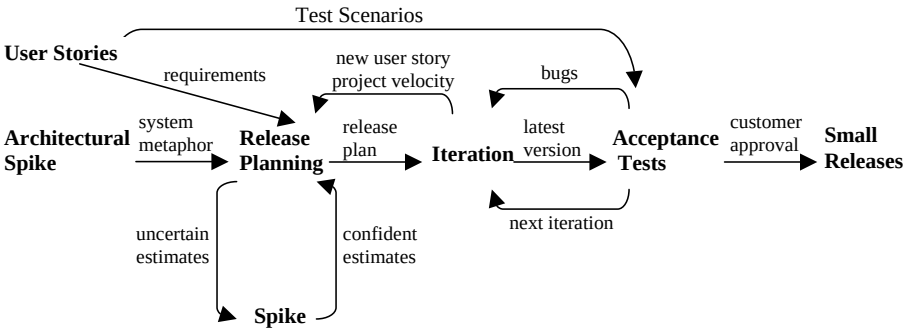


Figure 1.25 – L'exemple d'un projet XP [214]

XP est basée sur l'utilisation du code pour diriger le développement de logiciel. Les principales étapes sont le codage et les tests. Cependant, le codage et les tests rendent la maintenance des systèmes informatiques très difficile. De plus, l'utilisation de XP est conseillée pour réaliser des projets de taille raisonnable, et ayant un cycle de vie court. Au final, l'extension ou la maintenance d'un projet réalisé avec XP pose toujours problème, puisque les développeurs qui détiennent l'expertise sont souvent en changement de secteur ou d'entreprise.

1.9 L'Architecture dirigée par les modèles (MDA)

Dans cette section, nous décrivons plus précisément l'architecture dirigée par les modèles ou MDA (*Model Driven Architecture*), basée sur les technologies standard présentées précédemment (UML, MOF et XMI). Nous en profitons pour mettre en évidence la définition, les acteurs et les techniques de développement. Cependant, il faut comprendre qu'il n'y a pas une spécification précise de MDA, mais un ensemble de spécifications qui forment le MDA.

1.9.1 MDA : définitions et acteurs

Dans cette section, nous décrivons MDA en abordant les questions suivantes : Qu'est-ce que MDA ? Pourquoi MDA ? Quels sont les bénéfices de l'approche MDA ? Par qui a-t-elle été conçue et normalisée ?

1.9.1.1 Qu'est-ce que MDA ?

Dans la littérature, nous trouvons plusieurs définitions de MDA. Nous en présentons les plus courantes dans les paragraphes suivants.

MDA est « une évolution de l'OMA qui assure l'intégration et l'interopérabilité pour couvrir le cycle de vie d'un système depuis le modèle métier (*business modeling*) et sa conception, jusqu'à la construction de composants, l'assemblage, l'intégration, le déploiement, la gestion et l'évolution » [138]. Elle est bâtie sur l'expérience acquise par la création de standards comme UML, MOF, CWM, XMI et IDL. MDA définit une architecture pour structurer les modèles afin de permettre l'intégration, l'interopérabilité et la portabilité.

MDA est une approche qui augmente la puissance des modèles pour le développement de systèmes. Elle est dirigée par les modèles « parce qu'elle fournit une approche de l'utilisation de modèles pour diriger la compréhension, la conception, la construction, le déploiement, l'opération, la maintenance et la modification de systèmes » [147].

MDA est « une approche pour la spécification de systèmes des technologies de l'information (IT - *Information Technology*). Elle sépare la spécification des fonctionnalités de la spécification de l'implémentation de ces fonctionnalités sur une plate-forme technologique spécifique » [147].

Le standard MDA de l'OMG est « un cas particulier de l'approche MDE (*Model Driven Engineering*) » [63].

En résumé, nous définissons MDA comme une démarche de développement basée sur les modèles et un ensemble de standards de l'OMG. Cette démarche permet de séparer les spécifications fonctionnelles d'un système (le PIM - *Platform Independent Model*) des spécifications de son implémentation (le PSM - *Platform Specific Model*). Les standards fondamentaux de MDA sont IDL, UML, MOF, CWM et XMI. Dans le cas de MDA, l'OMG préconise la transformation comme opérateur fondamental devant être appliqué entre PIM et PSM ou entre PIMs ou entre PSMs.

MDE est une approche plus générale que MDA : elle considère l'existence de plusieurs modèles dans le sens large (c.-à-d. les métamétamodèles et les métamodèles sont aussi des modèles) sur lesquels des opérations spécifiques peuvent être réalisées. Les opérations peuvent être élémentaires (par exemple, *create*, *update*, *delete*, *select*, *enumeration*) ou complexes (par exemple, *matching*, *differencing*, *merge*, *transformation*) [21] [115] [168]. De plus, MDE est une approche ouverte qui prend en charge plusieurs autres espaces technologiques afin de les harmoniser [63] [104].

Dans la démarche MDA, tout est considéré comme modèle, aussi bien les schémas que le code source ou binaire. Le modèle indépendant de plate-forme (PIM) et le modèle dépendant de plate-forme (PSM) constituent les principaux types de modèles dans le cadre de MDA. Les PIM sont des modèles qui n'ont pas de dépendance avec les plates-formes technologiques (ils sont donc indépendants de CORBA, EJB, Services Web, dotNET). Les PSM sont des modèles dépendants des plates-formes technologiques et servent de base à la génération de code exécutable.

Une plate-forme est définie comme « *a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented* » [147]. Cependant, cette définition de plate-forme est ambiguë.

Dans [147], les auteurs essaient de clarifier le terme plate-forme en citant des exemples :

- types de plates-formes génériques :
  - *Object* : plate-forme qui supporte le style architectural des objets avec interfaces, requêtes individuelles de services, réalisation de services déclenchée par ces requêtes, et réponse à ces

requêtes.
  - *Batch* : plate-forme qui supporte une série de programmes indépendants dont l'exécution est séquentielle.
  - *Dataflow* : plate-forme qui supporte un flux continu de données entre les parties des logiciels.
- types de plates-formes spécifiques à la technologie :
  - CORBA : plate-forme objet qui permet l'invocation à distance et les styles d'architectures basés sur les événements.
  - CORBA Components : plate-forme objet qui permet le style architectural basé sur les composants et les entrepôts.
- types de plates-formes spécifiques au vendeur :
  - CORBA : Iona Orbix, Borland Visbroker, JDK-CORBA.

Ces exemples aident à comprendre ce qu'est une plate-forme. Cependant, un autre problème se pose. Dans ces exemples, les auteurs mentionnent les types de plates-formes génériques, parmi eux les *objects*. Dans ce cas, un modèle métier en UML ou EDOC peut être indépendant de plates-formes spécifiques à la technologie ou au vendeur, mais n'est pas indépendant de l'objet, puisque UML et EDOC sont basés sur la notion d'objet. En effet, le fait qu'un modèle soit indépendant ou non d'une plate-forme est relatif. En réalité, un modèle doit toujours être dépendant d'une plate-forme, quelle soit l'objet ou une autre plate-forme. D'autres propositions pour classer les modèles paraissent plus intéressantes et cohérentes, comme celle présentée dans [6]. Dans les chapitres qui suivent, nous approfondissons ces propositions.

Les opérations de transformation sont aussi à la base de la démarche MDA [142]. Dans ce cas, les transformations peuvent être groupées en :

- PIM vers PIM : ces transformations s'effectuent pour ajouter ou soustraire des informations aux modèles. Le passage de la phase d'analyse à la phase de conception est la plus naturelle des transformations de ce genre. De telles transformations ne sont pas toujours automatiques, et demandent l'intervention du développeur pour ajouter ou soustraire des informations. La plus naturelle des transformations est l'enrichissement de PIM en étapes successives afin de le rendre plus complet.
- PIM vers PSM : ces transformations s'effectuent lorsque les PIM sont suffisamment enrichis avec des informations pour pouvoir être transformés sur une plate-forme technologique. Un exemple de ce genre est la transformation d'un modèle métier en UML vers un modèle UML qui prend en compte la logique métier en utilisant une plate-forme comme les services Web.
- PSM vers PIM : ces transformations s'effectuent pour séparer la logique métier de la plate-forme technologique. Par exemple, une transformation d'un PSM (par exemple, un modèle UML qui utilise des profils CORBA) qui soustrait la plate-forme (par exemple, services Web) et retourne que la logique métier. Particulièrement, ce genre de transformation est le plus demandé dans les processus de *reverse engineering* [40].
- PSM vers PSM : ces transformations s'effectuent lors des phases de raffinement de plates-formes technologiques, de déploiement, d'optimisation ou de reconfiguration.

Les relations [6] entre PIM, PSM, métamodèles, infrastructure (i.e. plate-forme) et autres technologies sont décrites par le métamodèle MDA (voir figure 1.26) [138].

En analysant la figure 1.26, nous pouvons remarquer que MDA fournit le processus par lequel un modèle est transformé dans un autre modèle. La figure 1.27 présente une autre illustration pour comprendre les éléments d'une transformation et les relations entre eux [35].

[6] Aujourd'hui la relation `basedOn` a été redéfinit en `ConformsTo`.

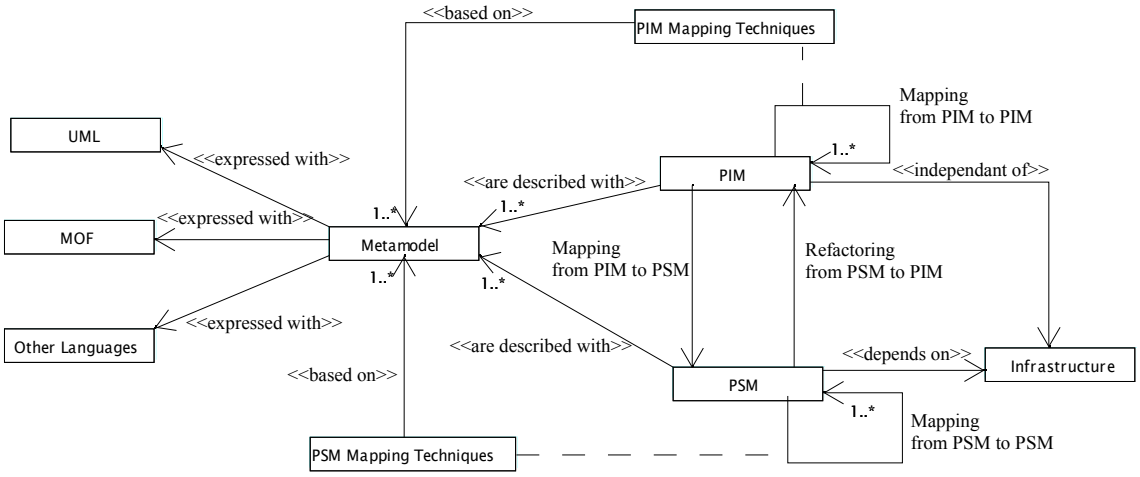


Figure 1.26 – La description du métamodèle MDA[138]

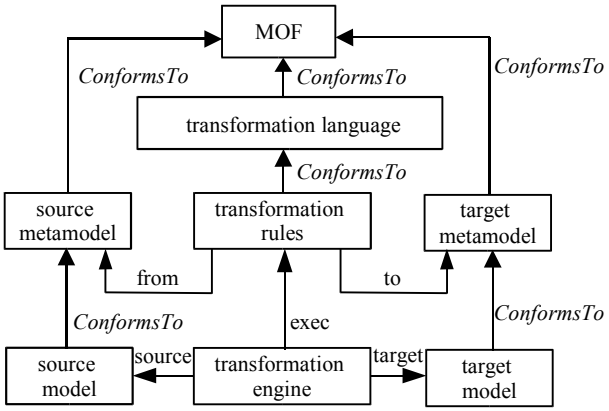


Figure 1.27 – La transformation de modèles en MDA [35]

1.9.1.2 Pourquoi MDA ?

Dans les années 80, la notion d'objet était considérée comme le noyau autour duquel s'articule le développement du logiciel pour les systèmes d'information. Elle a aussi donné naissance aux méthodes d'analyse et de conception par objets, comme Booch. L'adoption de la technologie objets a permis des avances incontestables dans le développement de logiciels, mais elle n'a pas résolu tous les problèmes.

Plus tard, la notion d'objets ne pouvait plus répondre à elle seule à toutes les exigences de développement et maintenance des systèmes d'information, car ils sont devenus beaucoup plus complexes. D'autres technologies sont alors apparues, comme les objets distribués (par exemple, CORBA), les patrons de conception et les composants (par exemple, CCM [140]).

Dans ce contexte, les intergiciels ont pris de l'importance pour assurer la coopération des objets ou des composants produits par des organisations différentes. Ainsi, au milieu des années 90, l'OMG a cru qu'il était possible de standardiser un seul et unique intergiciel capable de résoudre tous les problèmes d'interopérabilité et de portabilité. Cependant, d'autres intergiciels, comme DCOM, sont aussi apparus. Des passerelles entre ces deux intergiciels ont été proposées, mais elles ne représentaient pas une solution

satisfaisante, et apportaient plus de problèmes qu'elles n'en résolvaient. Ensuite, l'apparition d'autres intergiciels comme EJBs, CCM, dotNET et Services Web a rendu encore plus compliquée cette situation.

Ainsi le concept unique et simplificateur du « tout est objet » du début des années 80, a évolué vers un « tout est composant » dans les années 90. Dans ce contexte, les intergiciels jouaient un rôle fondamental pour résoudre les problèmes d'interopérabilité et de portabilité. Ces solutions étaient satisfaisantes tant qu'elles étaient capables de répondre aux exigences des systèmes informatiques, en particulier celle de la gestion de la complexité.

Parallèlement à cette évolution, d'autres propositions ont pris de l'importance comme la programmation par aspects.

D'une part toutes ces technologies ont été développées pour répondre à des exigences spécifiques et, d'autre part, au fur et à mesure qu'une technologie répond à une exigence, d'autres exigences apparaissent. Nous avons donc plusieurs solutions, ainsi que des intergiciels, mais elles ne sont pas capables de répondre aux nouvelles exigences. En considérant l'Internet, les Services Web ne sont pas une solution complète, mais un intergiciel de plus pour répondre aux exigences de développement des applications pour l'Internet.

En même temps, l'utilisation de modèles orientés objets a commencé à être plus courant pour supporter l'analyse et la conception de systèmes (par exemple les méthodes Booch, OMT, et OOSE). L'unification de ces différentes méthodes dans UML a ensuite démontré l'importance et la viabilité de la modélisation comme support fondamental pour développer et maintenir les systèmes informatiques.

L'OMG a compris qu'un intergiciel tel que CORBA n'était pas la solution la plus adaptée pour faire face à ce nouveau contexte dans lequel les systèmes informatiques deviennent plus complexes pour diverses raisons, à savoir :

- **l'intégration de nouveaux aspects** : la sécurité, la disponibilité et la fiabilité doivent être prises en compte dès le début (analyse et conception).

- **l'arrivée de technologies nouvelles** : aujourd'hui, les systèmes évoluent plus rapidement que dans le passé, pas seulement parce que les exigences du métier et des applications changent, mais aussi parce que les plates-formes techniques sont en constante et rapide évolution. Donc, la protection des investissements en logiciels contre l'obsolescence due à l'intégration de nouvelles plates-formes techniques est un but important.

- **la compatibilité avec les technologies anciennes** : les nouvelles technologies arrivent à grande vitesse mais les technologies anciennes ne disparaissent pas. Ces technologies sont agrégées dans les systèmes du passé (*legacy systems*), ce qui pose un problème d'hétérogénéité.

- **le grand nombre de technologies** : les systèmes informatiques distribués sont généralement construits avec plusieurs technologies. De plus ces systèmes doivent toujours communiquer avec d'autres systèmes d'une manière transparente.

Ces facteurs réunis rendent la productivité et la qualité des systèmes logiciels difficile à assurer. De plus, le temps de livraison des systèmes informatiques devient de plus en plus court, par conséquent les entreprises sont tenues de répondre aux exigences nouvelles le plus vite possible.

Ainsi, l'OMG a pris la décision d'investir dans l'approche dirigée par les modèles pour faire face à la complexité de développement, de maintenance et d'évolution des systèmes informatiques.

L'idée principale de MDA est représentée par son *logotype* (cf. figure 1.28). Les modèles sont les entités capables d'unifier et de supporter le développement de systèmes informatiques en assurant l'interopérabilité et la portabilité. Les modèles représentent le noyau (par exemple, MOF, UML et CWM), les intergiciels représentent le niveau d'exécution des applications (par exemple, CORBA, Services Web

et dotNET), et les services normalisés fournissent le support à l'exécution de ces applications (par exemple, la sécurité, transactions et événements). Tout cet ensemble de technologies est utilisé pour supporter différents domaines tels que la finance, le commerce électronique ou les télécommunications.

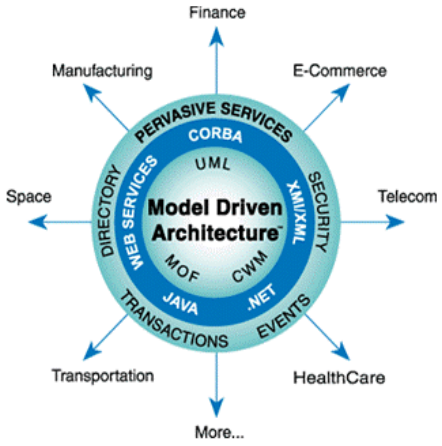


Figure 1.28 – L'Architecture Dirigée par les Modèles

MDA est apparu pour supporter l'évolution et gérer la complexité des systèmes distribués, mais aussi pour harmoniser les technologies. Nous ne considérons pas que les technologies utilisées dans le passé (par exemple, objet, composant et intergiciels) aient échoué, mais qu'elles n'étaient pas prévues pour résoudre ce genre de problèmes. L'apparition de l'Internet et l'utilisation en masse des systèmes informatiques ont également créé de nouvelles exigences jamais imaginées. Ces technologies sont ainsi arrivées à leurs limites. En particulier, leur niveau d'abstraction qui n'est plus capable d'aider les développeurs dans le contexte actuel. Dans ce cas, les modèles constituent la couche nécessaire pour fournir le niveau d'abstraction demandé aujourd'hui.

L'approche dirigée par les modèles (MDA ou MDE) n'est pas une solution qui va résoudre tous les problèmes, mais elle paraît être la seule aujourd'hui capable de fournir des réponses satisfaisantes à toutes ces nouvelles exigences. Ses limites ne sont pas encore connues, mais nous sommes conscients que toute approche a ses limites.

1.9.1.3 Quels sont les bénéfices de l'approche MDA ?

Certains bénéfices de l'approche MDA n'ont pas encore été vérifiés, mais les résultats actuels sont encourageants. Les bénéfices suivant ont été identifiés :

- Le même PIM peut être utilisé plusieurs fois pour générer les modèles sur des plates-formes différentes (PSMs) [73].
- Les différentes vues d'un même système, c'est-à-dire plusieurs niveaux d'abstraction[7] ou détails d'implémentation [138].
- L'augmentation de la portabilité et de l'interopérabilité des systèmes à travers les modèles.
- La protection de la logique métier contre les changements ou l'évolution des technologies [138].
- L'évolution simultanée des modèles métier et des technologies.

[7]Nous définissons les niveaux d'abstraction comme la possibilité de voir un système en plusieurs niveaux différents et liés entre eux, chaque niveau révélant une caractéristique importante du système.

- La protection contre les facteurs inhérents au développement manuel de systèmes [126].
- L'augmentation du retour des investissements en technologies.
- Le support pour le *reengineering* et *reverse engineering* [40] qui rend possible la récupération de la logique métier à partir de codes sources ou des environnements d'implémentation.
- L'augmentation des niveaux d'abstraction par lesquels les humains peuvent communiquer avec d'autres humains de manière plus productive.
- La réutilisation et la composition de modèles rendant possible la construction de systèmes complexes.
- L'augmentation de l'interaction et de la migration entre les espaces technologiques différents [104].

De plus, nous pouvons mentionner qu'une approche dirigée par les modèles oblige les architectes et les développeurs à focaliser l'architecture et le modèle du système en développement. Par opposition, une approche centrée sur le codage focalise l'attention des architectes et des développeurs sur le codage, par conséquent, négligent les propriétés principales du système. Dans [179], les auteurs mettent en évidence le support d'analyse plus élaboré fourni par MDA et leur utilisation dans les systèmes d'information d'entreprises (EIS - *Enterprise Information System*).

Des exemples d'utilisation de MDA ont déjà démontré plusieurs de ces bénéfices. Par exemple, dans [126], les auteurs ont montré que le développement d'une étude de cas (J2EE PetStore) est 35% plus rapide en utilisant un outil basé sur MDA plutôt qu'une approche centrée sur le code.

MDA est une solution plus puissante que l'approche traditionnelle ou XP, parce que le modèle et le code sont toujours en synchronisation, les informations sur les projets sont sauvegardées de façon formelle et bien documentée. De plus, il assure aussi le développement rapide des systèmes informatiques [126].

1.9.1.4 Par qui a-t-elle été conçue et normalisée ?

MDA est une initiative de l'OMG partie du constat que les nouvelles exigences ne pouvaient pas être résolues en utilisant CORBA. Cette proposition a été renforcée par les organisations académiques, industrielles et gouvernementales qui ont vu dans l'approche dirigée par les modèles une solution nécessaire et viable dans le contexte actuel.

À ce jour, il n'y a pas une spécification MDA mais un ensemble de standards (UML, CWM, XMI, MOF) et documents qui établissent la base et la direction pour la création et pour l'utilisation de la démarche MDA [138] [142] [147].

1.9.2 Correspondance

Les concepts de correspondance (*mapping*) et de transformation sont souvent confondus dans la littérature sur MDA. Plusieurs documents utilisent les termes *mapping* et transformation de manière interchangeable. Les paragraphes suivants illustrent quelques définitions.

Mapping est « un ensemble de règles et techniques utilisées pour modifier un modèle afin d'obtenir un autre modèle. Les *mappings* sont utilisés pour la transformation de PIM vers PIM, PIM vers PSM, PSM vers PSM et PSM vers PIM » [138].

Un *mapping* en MDA fournit « les spécifications pour la transformation d'un PIM en un PSM sur une plate-forme particulière. Le modèle de la plate-forme va déterminer la nature du *mapping*. Un *mapping* est spécifié en utilisant un langage permettant de décrire la transformation d'un modèle en un autre

modèle. La description peut être en langage naturel, en langage algorithme, en langage d'action, ou en langage de transformation de modèle » [147].

Un *mapping* est « l'application ou l'exécution d'une fonction de *mapping* afin de transformer un modèle en un autre. Une fonction de *mapping* est une collection de règles ou d'algorithmes qui définissent comment un *mapping* spécifique est réalisé » [122].

Alors que nous ne partageons pas l'avis de la littérature sur MDA, nous sommes en accord avec la littérature sur les bases de données, dans laquelle la distinction entre *mapping* et *transformation* est bien explicite. Dans ce domaine, le terme *mapping* est utilisé dans le sens de correspondance, et le terme *transformation* est utilisé pour faire référence à l'action de transformer un modèle en un autre [21] [115] [168]. Ainsi, nous utilisons le terme correspondance (traduction de *mapping*) pour référencer les inter-relations existant entre les éléments d'un modèle (ou métamodèle) et ceux d'un autre modèle (ou métamodèle).

D'une part, la correspondance décrit quels sont les éléments équivalents ou similaires entre deux métamodèles et d'autre part, la transformation utilise la correspondance pour achever la création d'un modèle cible à partir d'un modèle source.

1.9.3 La transformation de modèles

La transformation de modèles est « le processus de conversion d'un modèle en un autre modèle du même système » [147].

Durant ces dernières années, la transformation de modèles, en particulier la création d'un langage de transformation normalisé, est devenue le centre des réflexions autour de MDA. L'OMG a lancé un appel à travers son RFP QVT [142] pour que des propositions d'un langage de transformation soient faites. À ce jour, plusieurs d'entre elles ont été reçues [59] [91] [171], mais elles sont encore en analyse.

D'autres langages de transformation ont aussi démontré la viabilité des concepts autour de MDA, par exemple ATL (*Atlas Transformation Language*) [32], Mtrans [164], YATL (*Yet Another Transformation Language*) [163], BOTL (*Bidirectional Object oriented Transformation Language*) [117], TPL (*Template Pattern Transformation Language*) [45], UMLx [62] et MOLA (*MOdel transformation LAnguage*) [98].

Ces langages de transformation sont basés sur différents paradigmes : impératives ou déclaratives, non-typés ou typés syntaxiquement ou typés sémantiquement, textuelles ou graphiques [54]. Il est alors possible qu'un langage de transformation soit plus adapté qu'un autre dans un contexte spécifique. Cependant, en général, la tendance est de proposer des langages mixtes intégrants différents paradigmes. Dans le chapitre suivant, nous présentons plus des détails sur les langages de transformations et leurs caractéristiques, en particulier, les langages ATL, YATL et BOTL.

Quoi qu'il en soit, ces langages de transformation représentent une évolution significative pour MDA. De plus, le langage de transformation normalisé sera un élément fondamental pour l'avenir de MDA [12] [13] [73] [103].

Un outil de transformation de modèles prend comme entrée un modèle et le transforme en un autre modèle. Au niveau actuel de MDA, le modèle d'entrée est souvent le PIM et le modèle cible est souvent le PSM. Cependant, une transformation peut avoir un PSM comme source et un autre PSM comme cible. Une transformation peut aussi avoir un PSM comme le modèle source et un PIM comme le modèle cible. De plus, une transformation de PIM vers PIM est tout à fait possible, même si elle nécessite l'intervention humaine.

Nous voulons faire ressortir trois concepts reliés au terme « transformation » :

- d'abord, une **transformation** est la génération manuelle, semi-automatique ou automatique d'un modèle cible à partir d'un modèle source, en conformité avec une définition de transformation.
- une **définition de transformation** (ou *transformation specification*) est une collection de règles de transformation qui décrivent ensemble comment un modèle spécifié dans un langage source peut être transformé en un autre modèle spécifié dans un langage cible.
- une **règle de transformation** est une description de la manière dont une ou plusieurs constructions dans un langage source peuvent être transformées en autre construction dans un langage cible.

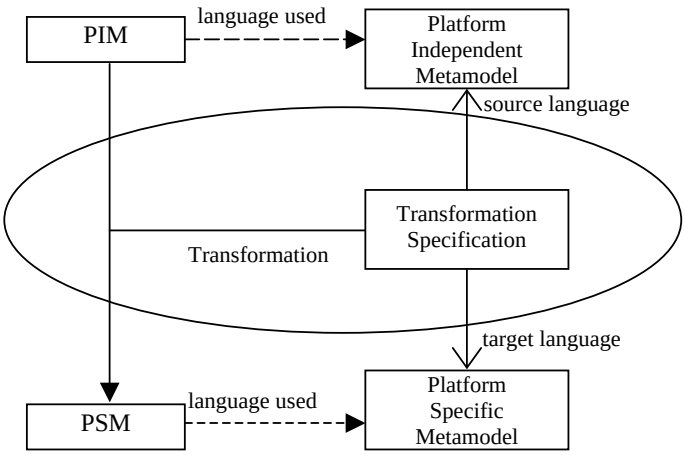


Figure 1.29 – La transformation de métamodèle [147]

Dans [147], les auteurs présentent une classification intéressante des approches pour transformer les modèles : *marking*, transformation de métamodèle, transformation de modèle, application de patrons et fusion de modèles (*model merging*). Dans cette thèse, nous nous intéressons plutôt à la transformation de métamodèles [8] qui est présentée par la figure 1.29. Cependant, nous distinguons explicitement l'étape de *transformation specification* en deux phases que nous appelons respectivement la spécification de correspondances et la définition de transformations. Dans les chapitres suivants, nous détaillons notre démarche.

En plus de ces approches, aujourd'hui, l'approche de *weaving* est proposée dans le cadre de MDA [22] [38] [198] [219]. La figure 1.30 présente la technique de *weaving* en comparaison avec la technique de *marking*.

La technique de marquage (*marking*) consiste à ajouter des marques (*marks*) dans la transformation d'un PIM en un PSM. Les marques sont des extensions d'un modèle initial et non intrusives qui spécifient les informations nécessaires pour la transformation de modèle sans polluer le modèle original [122]. Elles sont définies par les modèles de marquage, lesquels décrivent la structure et la sémantique d'un ensemble de types de marques.

Dans le cadre de MDA, les marques sont faites grâce à l'utilisation de profils UML. Ainsi, le méta-modèle d'une plate-forme spécifique est représenté comme une extension d'UML à travers le profil de cette plate-forme. Les marques peuvent aussi être utilisées pour raffiner un PIM afin d'ajouter d'autres informations comme la sécurité, mais sans spécifier la plate-forme qui va l'implémenter.

[8] Dans cette thèse, nous utilisons le terme « transformation de modèle » avec le même sens de « transformation de métamodèle » présenté dans [147].

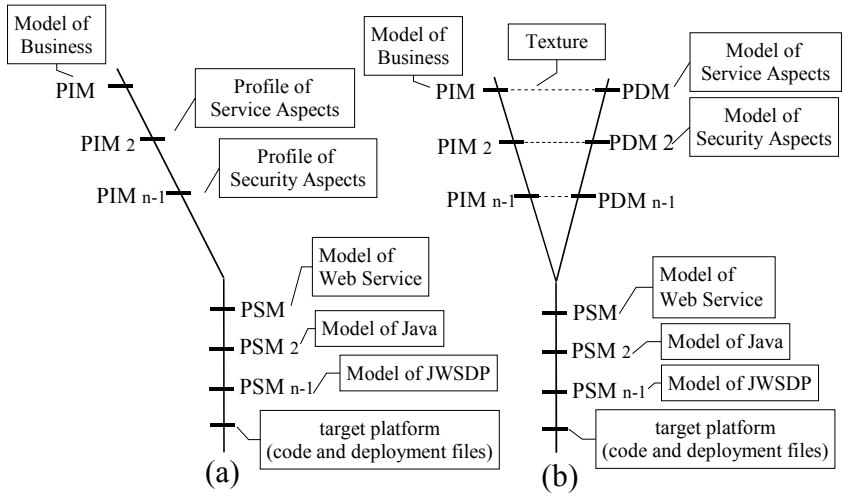


Figure 1.30 – Les techniques (a) *marking* et (b) *weaving*

Weaving est la technique la plus récente. Elle est basée sur l'idée du tissage (*texture mapping*) entre le modèle indépendant de la plate-forme (PIM) et le modèle de description de la plate-forme (PDM - *Platform Description Model*), puis sa transformation dans un modèle spécifique de plate-forme (PSM).

1.9.4 Les mythes autour de MDA

Plusieurs mythes autour de MDA peuvent être énumérés :

- **L'indépendance totale de plate-forme** : la notion de plate-forme est relative, et non absolue. Par exemple, IDL-CORBA peut être considéré comme indépendant de plate-forme parce que il a été conçu indépendant des systèmes d'exploitation et des langages de programmation. Dans ce cas, la plate-forme de référence est le système d'exploitation et les langages de programmation. Dans le cas d'un modèle métier basé sur UML, il est indépendant de plate-forme si nous considérons un type d'intergiciel comme référentiel (par exemple, CORBA). Cependant, ce modèle UML est spécifique à la plate-forme orientée objet, puis que UML est orienté objet.
- **L'utilisation d'UML pour modéliser tout type de système** : UML est un langage pour les ingénieurs et les architectes de logiciels. Cependant, UML n'est raisonnablement pas le meilleur langage pour modéliser certains aspects des systèmes particuliers. D'autres langages peuvent être développés spécifiquement pour prendre en charge la modélisation de tels systèmes, ceci est mis en avant par la tendance DSL (*Domain Specific Language*) [51]. Dans le *framework* basé sur les quatre niveaux de modèles, un aspect important est la présence de plusieurs langages de modélisation (métamodèles) comme préconisée par MDE. Ce dernier vise à élargir l'espace des formalismes de modélisation adapté au domaine.
- **La génération automatique et complète du code à partir de modèles** : en règle générale, il est possible de générer un code à partir d'un modèle seulement si l'information nécessaire à sa génération est présente dans le modèle. Dans l'état actuel de l'ingénierie de modèles et de MDA,

il n'est pas possible de générer l'intégralité du code, puis les modèles ne définissent pas la sémantique et la logique avec un niveau de détail suffisant. Pour résoudre ce problème, l'OMG propose l'utilisation de *action semantics* de UML 2.0. L'*action semantics* est une proposition qui présente des aspects positifs, mais elle est encore très récente et peu expérimentée pour assurer que son utilisation va permettre la génération de l'intégralité du code dans une approche MDA [177].

- **Les problèmes de l'approche MDA seront résolus avec un langage de transformation normalisé** : la transformation est sans doute la partie importante de MDA, qui occupe aujourd'hui un large espace de la recherche autour de MDA. Cependant, d'autres parties vont voir le jour. Une vision plus futuriste de MDA peut être liée à gestion de modèles [21] [159].
- **L'utilisation de modèles va simplifier les systèmes complexes** : s'il est vrai que les modèles peuvent faciliter la gestion de la complexité, ils ne peuvent cependant pas transformer un système complexe en un système simple. La différence réside dans le fait que l'utilisation de modèles nous permet d'étudier un aspect spécifique de ce système, de le comprendre et de le dominer. Un modèle ne peut représenter qu'un nombre limité d'aspects, mais plusieurs modèles peuvent conjointement représenter un système dans son ensemble. L'avantage principal de l'utilisation de modèles est qu'ils nous permettent de traiter un problème par fragments afin de mieux l'appréhender et de fournir de solutions. Ainsi, la complexité d'un système est divisée afin que l'être humain, ou que les machines, puissent comprendre les fragments et présenter des solutions viables. Par conséquent, l'ensemble des solutions permet de résoudre la complexité globale du système.

1.9.5 Le futur de MDA

Faire des prédictions au sujet des technologies est très difficile, surtout si elles sont liées aux systèmes d'information et à leur évolution.

Nous pensons que le futur de MDA va dépendre de son utilisation (pour construire et maintenir des systèmes informatiques) et des résultats obtenus. Cependant, il faut savoir que MDA n'est plus simplement une théorie, mais une réalité présentant déjà des résultats positifs.

D'autres travaux ont déjà démontré la viabilité de l'application de MDA, par exemple l'application de MDA aux services Web [35] [68], aux systèmes embarqués [74] [77], à CORBA [11], aux applications militaires [188] [189], et aux systèmes d'information [177].

Plusieurs entreprises proclament avoir implémenté une approche MDA. Cependant, l'aproche MDA n'est pas encore complètement expérimentée et plusieurs problèmes ne sont pas encore résolus. Par exemple, le langage de transformation autour de MDA n'est pas normalisé, mais les propositions soumises à l'OMG devraient offrir un avenir sûr pour MDA. De plus, méthodologies autour de MDA sont aussi nécessaires.

L'approche dirigée par les modèles a déjà prouvée sa viabilité sur plusieurs études de cas mais elle doit aussi changer la culture des architectes et des développeurs qui ont en charge la création et la maintenance de systèmes informatiques. En d'autres termes, MDA doit encore faire face à des résistances culturelles à l'instar du passage de MERISE à UML. Ils doivent donner plus de place aux modèles au détriment des connaissances et pratiques du codage des applications.

Dans ce nouveau contexte, les architectes et les développeurs doivent acquérir de nouveaux savoir-faire pour exprimer les fonctionnalités des systèmes à travers les modèles, métamodèles, les correspondances entre métamodèles et les transformations de modèles. Les outils de conception et de développement des applications doivent aussi évoluer pour supporter MDA (modélisation, transformation et génération de code exécutable).

Les réponses à MOF/QVT [142] et la standardisation d'un langage de transformation représenteront le franchissement d'une étape cruciale pour l'avenir de MDA, mais ce ne sera pas la seule. D'autres standards seront demandés au fur et à mesure que MDA évoluera. Parmi eux, la normalisation de métamodèles spécifiques au domaine [51], la création de processus MDA plus sophistiqués [63], la création d'outils permettant de spécifier des métamodèles et d'éditer les modèles basés sur eux, la normalisation de métamodèles de plates-formes et l'interopérabilité entre les outils.

Du point de vue économique, plusieurs entreprises ont déjà montré leur intérêt et le chiffre des investissements pour adopter une approche MDA ou, dans un sens plus large, MDE. Par exemple, THALES (http://www.thalesgroup.com) a révélé qu'elle a déjà investi 5 millions d'euros et prétend investir plus 20 millions d'euros en partenariat avec la commission européenne pour développer l'approche dirigée par les modèles [174].

1.10 Synthèse

Les systèmes d'information ont évolué d'un contexte centralisé et monolithique à des environnements distribués et hétérogènes. Dans ce nouveau contexte, les intergiciels ont pris une place importante à fin d'assurer l'interopérabilité entre différents systèmes d'information.

Étant donné que les intergiciels sont devenus primordiaux à la vie d'une entreprise, plusieurs constructeurs et organismes de normalisation ont proposé leur propre intergiciel sans consensus universel. Plusieurs intergiciels sont apparus comme DCE, Java RMI, XML-RPC, DCOM et CORBA. Parmi ces intergiciels, DCOM et CORBA ont été les plus répandus, mais l'interopérabilité entre eux n'était pas assurée, alors qu'ils étaient eux-même conçus pour assurer le problème d'interopérabilité. Ainsi, la période d'euphorie, où on pensait que les intergiciels allaient résoudre tous les problèmes de développement de systèmes d'information, s'est terminée par une période de conflit entre les intergiciels (*middleWar*). À la même période, l'Internet est arrivé aux entreprises avec des nouvelles exigences pour lesquelles les intergiciels traditionnels ne pouvaient pas répondre de manière satisfaisante. Afin d'assurer une interopérabilité universelle, les grands constructeurs et les organismes de normalisation se sont mis d'accord sur un ensemble de technologies, formant aujourd'hui les services Web.

Parallèlement aux apparitions des intergiciels tel que DCOM, CORBA et Services Web, les techniques et langages de modélisation ont pris une place importante avec la naissance de UML comme langage standard. Ainsi, l'idée que l'interopérabilité ne devait pas être résolue au niveau du code, mais à un niveau d'abstraction plus élevé commençait à émerger. Ainsi, en novembre 2000, l'OMG a proposé l'approche dirigée par les modèles pour le développement et la maintenance des systèmes à prépondérance logicielle.

CHAPITRE 2

État de l'art : MDA pour les services Web

2.1 Introduction

Dans ce chapitre, nous proposons un état de l'art sur les services Web, MDA, et plus précisément sur l'application de l'approche MDA aux plates-formes de services Web.

Les services Web ont évolué à partir d'un ensemble limité de technologies (SOAP, WSDL, UDDI et XML) vers un grand nombre de technologies complémentaires dédiées aux applications sur l'Internet. Ces technologies complémentaires ont pour but de fournir des services essentiels pour le fonctionnement des services Web ou des recommandations pour la création d'applications orientées services. Parmi ces services, nous pouvons évoquer la sécurité [136], l'aspect transactionnel [41], la composition de services [9] et les évènements. Parmi ces recommandations, nous pouvons mentionner le WSA (*Web Service Architecture*) [209] et le WS-*Basic Profile* [221].

Les axes de recherche qui abordent les services Web sont également nombreux. Nous pouvons citer :

- La composition de services Web basée sur la technologie de Workflow [48] [83] [92] [173] [197] et sur les réseaux de Petri [80].
- La composition de services Web basée sur la sémantique des services Web [116], l'ontologie [118] [218] et la planification [46].
- La sécurité des services Web [29] [108] [109] [135] [136].
- L'aspect transactionnel et les services Web [111] [157].
- Le traitement orienté services (*service oriented computing*) [158] et le *grid computing* [170] [191].
- L'architecture orientée services [17] [127] [209].
- Les applications mobiles basées sur les services Web [52] [166].
- Les applications Internet (B2B *e-commerce*, B2C) supportées par les services Web [29] [55] [85] [121] [176].
- La gestion de systèmes informatiques basée sur les services Web (*Web Services Management*) [31].
- L'application de services Web et XML pour faciliter la gestion de données distribuées et semi structurées [2].
- Les *frameworks* basés sur les services Web [19] [66] [83] [93] [100] [160] [222].

A partir de 2003, l'approche MDA a commencé à évoluer rapidement : elle est passée d'un ensemble de standards déjà connus et utilisés (par exemple, UML, MOF et XMI) à des langages de transformation de modèle et des propositions de méthodologies, démarches et processus de développement de logiciel. De plus, les premiers outils basés sur l'approche dirigée par les modèles sont apparus.

Parmi les axes de recherche qui abordent l'approche MDA, nous pouvons citer :

- La spécification de correspondances (*mappings* [43] [82] [115] [168] [172] [219] et la définition de transformation de modèles [32] [59] [71] [117] [163] [171] [200].
- La définition de méthodologies [47] [67] [72] [103] [165] et de processus MDA [65].
- L'application de MDA pour guider le développement des applications basées sur les plates-formes CORBA [202], CCM (*CORBA Component Model*) [18] et Services Web [15] [27] [35] [68] [78] [99] [156] [161] [180].
- L'application de MDA pour développer les systèmes embarqués [77].
- Les outils de développement basés sur l'approche MDA :
  - outils réalisés selon la philosophie de logiciel libre : *ATL Development Tools* [14], Jamda [182], AndroMDA [1] et JET [169].
  - outils réalisés comme produit commercial : OptimalJ [50], ArcStyler [90], XDE [88], Codagen Architect [49], MIA (*Model-in-Action*) [123].

Dans cette thèse, nous nous intéressons à l'application de MDA dans l'élaboration de logiciels basés sur la plate-forme de services Web.

Dans la littérature sur l'approche MDA et, plus généralement, dans la modélisation des systèmes, il y a deux tendances complémentaires et non exclusives concernant les techniques de transformation. La première est d'étudier les problèmes liés à la création de correspondances entre métamodèles (ou modèles). La deuxième se concentre sur la conception et la réalisation d'un langage de transformation selon les recommandations de l'OMG. Actuellement, dans la littérature MDA, ces deux tendances sont souvent confondues et rarement distinguées de manière explicite. Dans cette thèse, nous proposons une séparation explicite entre la spécification de correspondance et la définition de transformation.

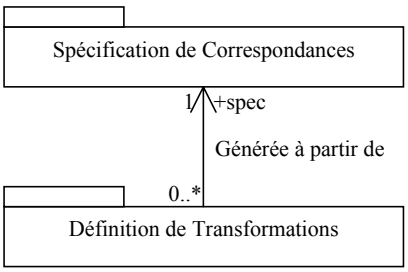


Figure 2.1 – Relation entre spécification de correspondances et définition de transformations

La figure 2.1 illustre notre proposition et met en relation la spécification de correspondances et la définition de transformations. D'une part, la spécification de correspondances présente la logique utilisée pour établir les relations entre deux métamodèles. D'autre part, la définition de transformations contient des règles précises pour transformer un modèle en un autre modèle, en utilisant un langage de transformation exécutable. En effet, la spécification de correspondances peut être considérée comme un PIM, et la définition de transformations comme un PSM. D'une part, la spécification de correspondances nous permet de faire des analyses et de prendre des décisions concernant les correspondances entre deux métamodèles. D'autre part, la définition de transformations est écrite dans un langage de transformations qui est une plate-forme spécifique à la réalisation et l'exécution de transformations. Nous reviendrons plus en détails par la suite sur l'intérêt d'une telle démarche.

Les sections suivantes sont organisées dans cette perspective : séparer l'étude de la spécification de correspondances de la création d'un langage de transformation. Cette séparation entre spécification de correspondances et définition de transformations est utilisée dans notre démarche.

Dans cet état de l'art, les travaux que nous présentons utilisent le terme « *mapping* » selon trois interprétations : correspondance, transformation, ou correspondance et transformation. Nous précisons à chaque fois l'interprétation mise en avant.

2.2 Correspondance entre métamodèles

Dans cette section, nous présentons plusieurs travaux qui ont abordé la spécification de correspondances dans la perspective d'expliciter la problématique, d'établir une formalisation, et de trouver une théorie pour prendre en charge cette problématique.

G. Caplat et J. Sourrouille définissent le *mapping* dans le sens de la transformation de modèles de la façon suivante [43] :

« *A **mapping** is a set of rules and techniques used to modify one model in order to get another model* ».

« *Let $m(s)/f$ the model of the system s in the formalism f. A mapping is a general transformation $m_1(s)/f_1 \rightarrow m_2(s)/f_2$* ».

« *The model of a formalism is called a metamodel* ».

G. Caplat et J. Sourrouille

Ils traitent le problème général du *mapping* dans lequel $f_1 = UML + Profile$, $m(f_1)/MOF$, où f_1 est décrit en MOF, et où $m_1(s)/f_1$ est un PIM et $m_2(s)/f_2$ est un PIM enrichi avec les éléments du PSM. De plus, ils présentent deux types de *mappings* : directs (ou endogènes) et indirects (ou exogènes). Le *mapping* direct est réalisé quand les modèles sont décrits par un même formalisme. Le *mapping* est indirect quand les modèles sont décrits par des formalismes différents. La figure 2.2 illustre les deux types de *mappings*.

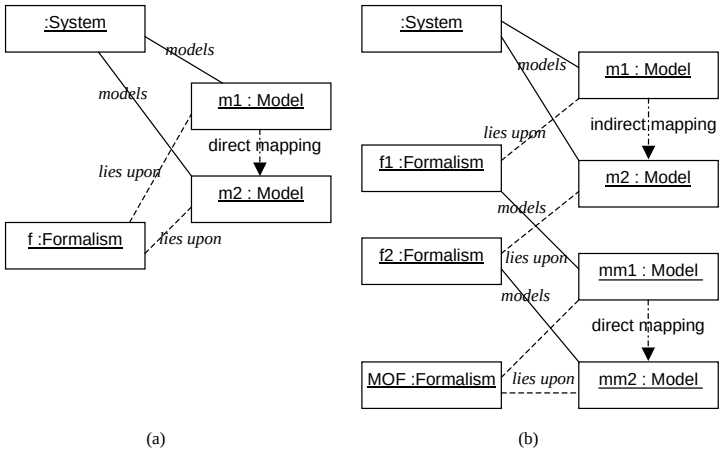


Figure 2.2 – (a) Le *mapping* direct et (b) le *mapping* indirect [43]

Un *mapping* direct est une transformation simple dans un processus de raffinement de modèles. Dans le cas du *mapping* indirect, une comparaison entre les formalismes f_1 et f_2 est nécessaire et une relation doit être définie de manière à ce que chaque expression primitive et abstraite en f_1 soit traduite dans

une expression primitive et abstraite en f_2. Dans la figure 2.2, le métalangage de formalisme permettant l'expression de telles traductions est naturellement le MOF.

Par rapport au *mapping* indirect, G. Caplat et J. Sourrouille soulignent que :

« *In general case, the indirect mapping of models* $m_1(s)/f_1 \rightarrow m_2(s)/f_2$ *requires the direct mapping of formalisms :* $mm_1(f_1)/MOF \rightarrow mm_2(f_2)/MOF$*. This mapping can be very expensive and even not possible. Fortunately, the situation is better when one of the formalisms provides an extension mechanism such as the UML. Either* f_1 *or* f_2 *(or both) may provide an extension mechanism* ».

G. Caplat et J. Sourrouille

Ce point de vue limite l'approche MDA puisque le seul métamodèle utilisable serait l'UML et son mécanisme d'extension (les *profiles*). Ceci est en accord avec la volonté de l'OMG d'établir l'UML comme seul langage de modélisation. Cependant, le *framework* de métamodélisation utilisé par l'OMG pour justifier l'approche MDA prévoit plusieurs langages de modélisation. Il est vrai que l'extension d'UML peut être vue comme un autre langage de modélisation, mais elle comporte des inconvénients. De plus, elle va à l'encontre de la proposition de langages de type DSL (*Domain Specific Language*) [51].

Par rapport aux techniques de transformation, G. Caplat et J. Sourrouille suggèrent que l'utilisation de différents formalismes (métamodèles) peut rendre la création de *mapping* difficile, voire impossible. Cette suggestion privilégie la technique de marquage, laquelle est basée sur les profils UML. Cependant, dans la technique de transformation de métamodèles et dans notre démarche, l'utilisation des différents métamodèles pour la création de modèles est indispensable pour la réalisation de transformations [35].

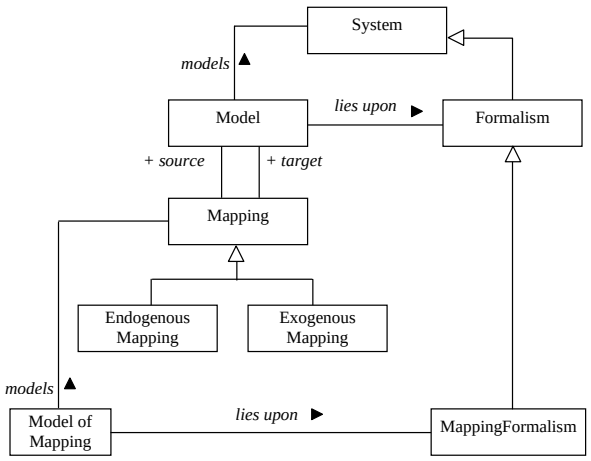


Figure 2.3 – Les relations entre *mapping* et formalismes [44]

La figure 2.3 présente la proposition de G. Caplat et J. Sourrouille pour décrire les rôles de chaque élément de l'approche MDA [44]. Elle illustre le fait qu'un *mapping* (au sens de transformation) met en relation un modèle source et un modèle cible. Un *mapping* est aussi basé sur un modèle (`Model of Mapping`) qui est lié à un formalisme (c.-à-d. `MappingFormalism`). Les *mappings* endogènes et exogènes sont des extensions d'un même *mapping* de base. Un formalisme hérite des caractéristiques d'un système.

Dans [63], J. M. Favre reprend cette figure et l'appelle mégamodèle qui est défini comme une théorie pour les concepts autour du MDE (*Model Driven Engineering*). Dans ce cas, il est important de

rappeler que MDA est un cas particulier de MDE. Selon J. M. Favre, ce mégamodèle (cf. figure 2.3) n'est pas suffisant pour répondre à toutes les questions au sujet du MDE [63]. Dans ce mégamodèle, un « `Model of Mapping` » n'est pas un modèle. Il ne peut donc pas être transformé en un autre « `Model of Mapping` ». De plus, ce mégamodèle considère UML comme l'unique formalisme, contrairement à MDE qui préconise plusieurs langages de modélisation. D'autres mégamodèles ont aussi été proposés [35] [102] [122] [147] [161]. Cependant, la recherche pour un mégamodèle précis de MDE est encore en cours [39].

G. Caplat et J. Sourrouille soulignent aussi les différences syntaxiques et sémantiques entre les métamodèles au moment de la création de *mappings* [44]. La figure 2.4 présente l'exemple qu'ils utilisent pour exposer les différences syntaxiques, et la figure 2.5 présente l'exemple qu'ils utilisent pour exposer les différences sémantiques.

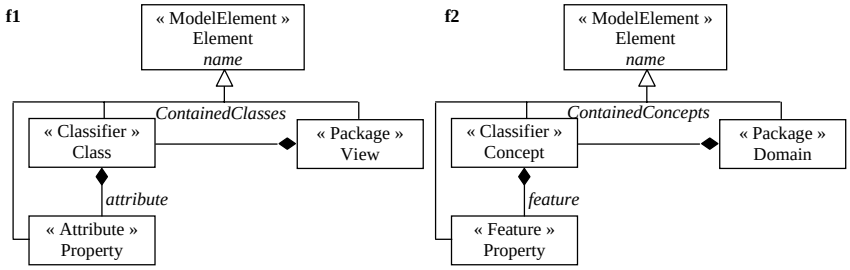


Figure 2.4 – Deux métamodèles simples (multiplicité non spécifiée) proposés en [44]

Dans la figure 2.4, les classes stéréotypées (notation à la façon UML) avec `ModelElement`, `Classifier`, `Package`, `Attribute` et `Feature` font référence à une méta-instance de MOF. Cette figure démontre que les éléments de métamodèles et modèles peuvent avoir la même sémantique, mais possède une syntaxe différente. L'élément `Classe` dans `f1` est l'équivalent de `Concept` dans `f2`. Le `View` dans `f1` est l'équivalent de `Domain` dans `f2`. La métaclasse `Attribute` utilisée pour créer `Property` dans `f1` est l'équivalent de la métaclasse `Feature` utilisée pour créer `Property` dans `f2`.

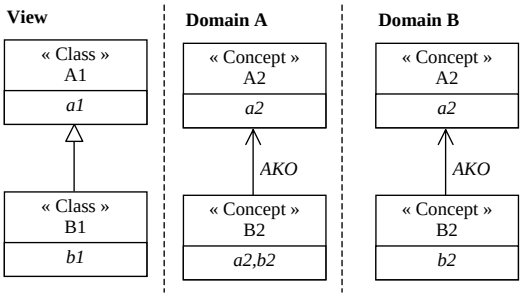


Figure 2.5 – Traduction de la relation d'héritage proposée en [44]

La figure 2.5 présente trois modèles basés sur les métamodèles de la figure 2.4 ainsi que la notion d'héritage en `f1` et la notion d'association binaire en `f2`. La notion d'héritage de `f1` doit être traduite en une expression en `f2` avec une sémantique équivalente. Contrairement à la figure 2.4 où le *mapping* ne concerne que la forme, la traduction d'un modèle `View` en `Domain` doit prendre en charge la sémantique. Dans ce cas, l'héritage peut être traduit en une association dénotée `AKO`. Si `B1` hérite de `A1`, les propriétés

de `A2` doivent être ajoutées aux propriétés de `B2`. Une première solution est d'ajouter toutes les propriétés de `A2` à `B2` (le modèle `View` et `Domain A`). Le modèle est facilement construit, mais une modification de la valeur de `A2.a2` n'est pas propagée à `B2.a2`. Si une telle modification est faite dynamiquement, `m1` et `m2` n'ont pas le même comportement, c'est-à-dire, qu'ils ne sont pas sémantiquement équivalents. Une deuxième solution est la transformation de l'accesseur des propriétés de `Concept` dans une opération qui prend en charge la propagation de l'altération des propriétés. Concrètement, l'opération `ref_value(x)` de MOF peut être redéfinie en `f2` de la manière suivante [44] :

$ref\_value(x)$
$if\ x\ \in\ self.properties()\quad //\ self\ is\ the\ current\ concept$
$\ result = self.ref\_value(x)$
$else\ if\ x\ \in\ self.associations(''AKO'').properties()$
$\ result = self.associations(''AKO'').ref\_value(x)$

J. H. Hausmann et S. Kent définissent le *mapping* (dans le sens de correspondance) de la façon suivante [82] :

« *... we use the term* ***mapping*** *to address the general understanding of 'connection between models' and refer to relation and pair whenever the distinction between definition and instance is suitable* ».

J. H. Hausmann et S. Kent

Les *mappings* doivent être conformes aux exigences suivantes [82] :
- Ils doivent représenter la connexion implicite entre leurs parties.
- Une notation pour *mapping* doit être facilement compréhensible pour permettre sa définition et son adoption par les utilisateurs humains.
- Ils ne doivent pas interférer dans la définition de modèles.
- Ils doivent avoir une instance persistante.

La figure 2.6 illustre le patron du modèle pour spécifier les relations entre modèles proposées dans [82].

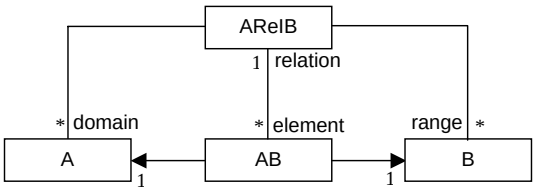


Figure 2.6 – Le patron du modèle pour les relations [82]

Pour spécifier les relations entre modèles, ce patron établit une relation (`ARelB`) et une paire de classes (`AB`) pour chaque correspondance entre les éléments des modèles. Les expressions OCL peuvent être utilisées pour définir le domaine (*domain*), le co-domaine (*range*) et les contraintes additionnelles de la relation.

La figure 2.7 illustre une syntaxe à la UML pour formaliser les relations entre les éléments des modèles [82].

Cette figure montre que chaque langage de modélisation (ici UML et Java) est défini séparément dans son *package*. Grâce à l'utilisation des relations, chaque connexion peut être exprimée sans influencer les contenus de ces *packages*. Les relations sont représentées par les lignes pointillées. En fait, le *mapping* est aussi un modèle.

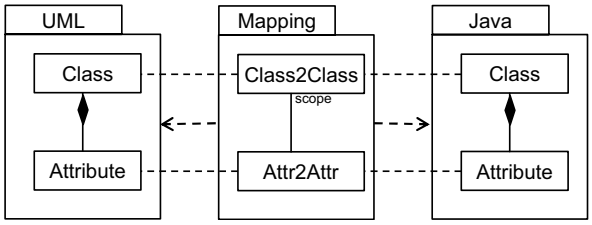


Figure 2.7 – Relations entre modèles [82]

Une syntaxe concrète pour définir les correspondances est proposée par la figure 2.8 [82]. Une relation est représentée par une ligne pointillée entre les classes qu'elle relie. Ces classes définissent le type des objets du domaine et co-domaine qui participent aux relations. La figure en forme de losange au milieu de la relation contient le nom de la relation. Les associations entre relations sont représentées par un lien entre ces losanges. Les n-uplets sont visualisés par un point noir à l'extrémité d'une relation (par exemple, `x` et `y` à l'extrémité de `B3`). Les noms et les cardinalités des extrémités des relations suivent la notation UML. Les relations imbriquées (relations qui sont définies dans le contexte d'une autre relation) sont mises en place par une association composite qui est représentée par un losange noir à l'extrémité de la relation conteneur.

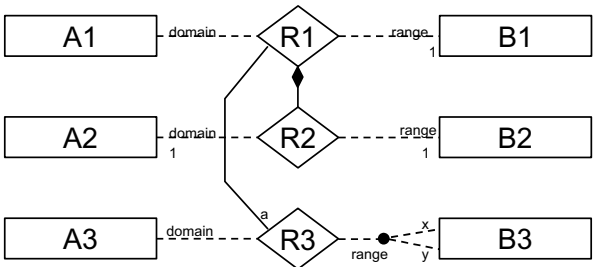


Figure 2.8 – Exemple de syntaxe de relations [82]

Dans la figure 2.8, la relation `R1` lie `A1` à `B1`, et contient une relation spécifiée par la relation `R2` qui lie `A2` à `B2`. De plus, `R1` est associée à `R3` qui lie `A3` à `B3`. Dans ce dernier cas, `R3` contient une cardinalité du type `1:n`, le domaine contient un élément, et le co-domaine contient deux éléments représentés par le n-uple `x` et `y`.

J. H. Hausmann et S. Kent présentent aussi un exemple intéressant du *mapping* entre UML et Java et proposent ensuite des définitions de transformation en OCL [82].

Dans [219], E. D. Willink propose un formalisme graphique nommé UMLx pour spécifier le *mapping*[1] entre modèles. Ce langage peut être divisé en deux parties : une syntaxe de schéma et une syntaxe de transformation. La syntaxe de schéma est basée sur un sous-ensemble du diagramme de classe UML. Par conséquent, UMLx présente aussi le concept de composition, d'association et d'héritage. La syntaxe de transformation contient les éléments : `from`, `to`, `preservation`, `evolution` et `removal`. Un élément `from` définit une association entre un élément d'un modèle source avec un contexte interne LHS (*Left Hand Side*), et `to` définit une association entre un élément d'un modèle cible avec un contexte

[1] E. D. Willinkdans [219] utilise le terme *mapping* dans le sens de correspondance et de transformation, mais il ne donne pas de définition explicite.

interne RHS (*Right Hand Side*). Les éléments `preservation`, `evolution` et `removal` spécifient la transformation d'une construction LHS dans une construction RHS.

Notre approche [34] [35], qui sera présentée dans les chapitres suivants, a des similarités avec les approches de J. H. Hausmann et S. Kent [82], et E. D. Willink [219], car nous définissons aussi des liens entre les éléments de correspondances, tel que la composition.

Dans le domaine de base de donnée et plus récement dans le projet gestion de modèles [21], P. A. Bernstein définit le *mapping* de la façon suivante :

« *...transformations between models, which in turn requires an explicit representation of* ***mappings****, which describe how two models are related to each other* ».

P. A. Bernstein

P. A. Bernstein apporte une contribution significative avec sa théorie de gestion de modèles et leur application dans l'intégration et l'évolution de schémas [21]. Cette théorie est basée sur des opérateurs pour gérer les modèles où les techniques de correspondances et de transformation prennent une place importante. Parmi ces opérateurs, nous citons :

- `Match` : prend deux modèles comme entrée et retourne un *mapping* entre eux comme résultat [172].
- `Compose` : prend un *mapping* entre les modèles `A` et `B` et un *mapping* entre les modèles `B` et `C`, et retourne un *mapping* entre `A` et `C` comme résultat.
- `Diff` : prend un modèle `A` et un *mapping* entre `A` et un modèle `B`, et retourne un sous modèle de `A` qui ne participe pas du *mapping*.
- `ModelGen` : prend un modèle `A`, et retourne un nouveau modèle `B` basé sur `A` et un *mapping* entre `A` et `B`.
- `Merge` : prend un modèle `A` et `B` et un *mapping* entre eux, et retourne l'union `C` de `A` et `B` [168].
- `Apply` : prend un modèle et une fonction abstraite `f` comme entrée et applique `f` pour chaque objet de ce modèle.
- `Copy` : prend un modèle comme entrée et retourne une copie de ce modèle.
- `ModelGen` : prend un modèle conforme à un métamodèle et génère un autre modèle conforme à un autre métamodèle.
- `Enumerate` : prend un modèle comme entrée et retourne un « curseur » comme résultat. Quand l'opérateur `Next` est appliqué au curseur, il retourne un élément du même modèle d'entrée ou `null` si le curseur est arrivé à la fin de l'énumération.

Dans le cadre de notre étude, nous nous intéressons principalement à l'opérateur `ModelGen` qui est implémenté par le moteur de transformation de ATL (*Atlas Transformation Language*) [32].

La création d'un *mapping* est liée au problème de *schema matching* qui est déjà très connu dans plusieurs domaines d'application des bases des données, tels que l'intégration de bases de données hétérogènes, le commerce électronique et les entrepôts de données (*data warehousing*). Ainsi, l'opérateur `Match` implémente un algorithme qui prend en charge le *schema matching* [172].

Typiquement, le *schema matching* est réalisé manuellement, et peut éventuellement être supporté via une interface graphique. La spécification manuelle de *schema matches* reste un processus fastidieux, source d'erreurs, lent et coûteux. Dans le contexte de MDA, ceci est aussi un problème courant, et la définition de transformations dépend du *mapping* qui est lié à une approche de *schema matching*.

Avant de définir un opérateur `match`, la représentation pour les schémas (ou métamodèles) d'entrées et le *mapping* de sortie doivent être déterminés. Dans le cas de MDA, nous devons déterminer le métamodèle source et cible, ainsi qu'un métamodèle pour le *mapping*.

Dans [172], E. Rahm et P. A. Bernstein définissent le *mapping* de la manière suivante :

« *We define a* ***mapping*** *to be a set of mapping elements, each of which indicates that certain elements of schema S_1 are mapped to certain elements in S_2. Furthermore, each mapping element can have a mapping expression which specifies how the S_1 and S_2 elements are related* ».

E. Rahm et P. A. Bernstein

En général, il est impossible de déterminer un *mapping* complet entre deux modèles de manière automatique, d'abord parce que les modèles ont des sémantiques différentes qui affectent les critères de correspondance entre les modèles. La réalisation d'un opérateur `Match` doit déterminer les candidats correspondants, que l'utilisateur peut accepter, refuser ou modifier. Dans ce cas, une approche semi automatique est plus réaliste. Malheureusement, les critères utilisés pour établir les correspondances entre les éléments de S_1 et de S_2 sont basés sur des heuristiques qui sont difficilement transposables en une forme mathématique précise qui guiderait l'implémentation de l'opérateur `Match` [172]. Les heuristiques permettent l'identification des similarités structurales et des noms entre les schémas. D'autres approches utilisent les techniques d'apprentissage (*machine learning*) pour acquérir la connaissance sur les *mappings* créés et les réutiliser pour déduire d'autres *mappings* [58].

Aujourd'hui, MDA en est encore à ses débuts. Plus tard, quand les systèmes seront développés de façon dirigée par les modèles à grande échelle, plusieurs définitions de transformations entre modèles seront nécessaires et des solutions pour le problème de *schema matching* seront donc requises. Ainsi, dans le contexte de MDA, une approche pour l'automatisation de *schema matching* est nécessaire.

Dans [168], R. A. Pottinger et P. A. Berstein présentent le *mapping* comme un des éléments fondamentaux pour réaliser entre autre la fusion (*merging*) des modèles. De plus, ils présentent différents types de relations entre les modèles et leurs implications, ce qui est très important pour trouver des éléments équivalents ou similaires dans la manipulation de modèles.

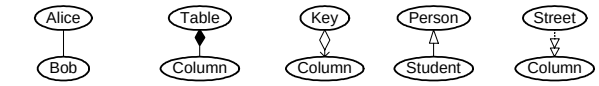


Figure 2.9 – Différents types de relations entre modèles [168]

Les types de relations sont illustrés par la figure 2.9. Ces relations sont très connues en modélisation de systèmes. Les types de relations sont [168] :

- `Associates` : $A(x, y)$ signifie que x est associé à y (association simple au sens UML).
- `Contains` : $C(x, y)$ signifie que x contient y (composition au sens UML).
- `Has-a` : $H(x, y)$ signifie que x a un sous composant y (agrégation au sens UML).
- `Is-a` : $I(x, y)$ signifie que x est une spécialisation de y (héritage au sens UML).
- `Type-of` : $T(x, y)$ signifie que x est du type y (typage d'attribut en UML).

Les implications entre les types de relations sont présentées comme suit [168] :

i) Si $T(q, r)$ et $I(r, s)$ donc $T(q, s)$.
ii) Si $I(p, q)$ et $H(q, r)$ donc $H(p, r)$.
iii) Si $I(p, q)$ et $C(q, r)$ donc $C(p, r)$.
iv) Si $C(p, q)$ et $I(q, r)$ donc $C(p, r)$.
v) Si $H(p, q)$ et $I(q, r)$ donc $H(p, r)$.

La figure 2.10 illustre de façon graphique les implications entre les types de relations. Ceci donne une vision de la façon dont les modèles peuvent être manipulés.

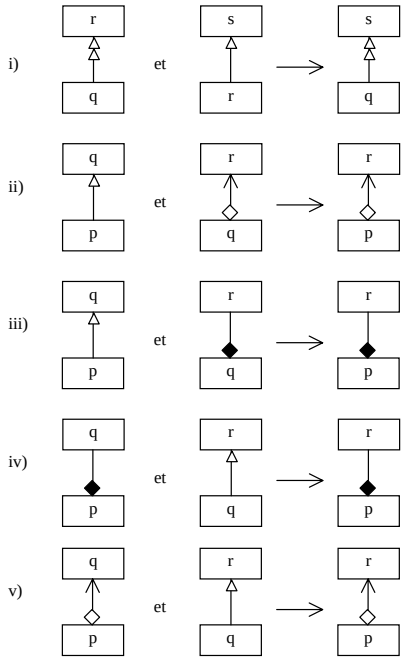


Figure 2.10 – Les implications entre les types de relations

Pour appliquer ces relations et ces implications (dans le but de trouver des similarités ou des équivalences entre les modèles), un modèle L peut être considéré comme un triplet $(EL, RootL, ReL)$. EL est l'ensemble des éléments de L, $RootL \in EL$ est l'élément racine de L, et ReL est l'ensemble des relations dans L.

R. A. Pottinger et P. A. Berstein définissent que deux modèles sont équivalents ou similaire comme suit :

« *We define two models to be **equivalent** if they are identical after all implied relationships are added to each of them until a fixpoint is reached (i.e., applying each rule results in no new relationship)* » [168].

R. A. Pottinger et P. A. Berstein

« *By **similar**, we mean that they are related but we do not express exactly how* » [21].

P. A. Bernstein

Y. Velegrakis, R. J. Miller et L. Popa ont étudié l'adaptation et l'évolution des *mappings* entre schémas quand ces derniers changent. Ils définissent le *mapping* comme suit [201] :

« *A **mapping** specifies how data instances of one schema correspond to data instances of another. **Mappings** are often specified in a declarative, data-independent way (for example, as queries or view definitions). However, they necessarily depend on schemas they relate* » [201].

Y. Velegrakis, R. J. Miller et L. Popa

Dans le travail de Y. Velegrakis, R. J. Miller et L. Popa, le *mapping* est formalisé comme suit [201] :

- Un *mapping* m d'un schéma S (schéma source) en un schéma T (schéma cible) est une assertion de la forme : $Q^S \rightsquigarrow Q^T$, où Q^S est une requête sur S et Q^T est une requête sur T.
- Un système de *mapping* est une triple $< S, T, M >$, où S est le schéma source, T est le schéma cible, et M est un ensemble de *mappings* entre S et T.
- Les types de *mapping* peuvent être classés en *sound*, *complete* et *exact*. En *sound mappings*, les requêtes retournent un ensemble de n-uplets, et la relation $\rightsquigarrow$ est la relation $\subseteq$ (sous-ensemble ou égal). En *complete mappings*, les requêtes retournent un ensemble de n-uplets, et la relation $\rightsquigarrow$ est la relation $\supseteq$ (super-ensemble ou égal). En *exact mappings*, la relation $\rightsquigarrow$ est la relation $=$ (égal).

Pour valider leur approche pour la maintenance et l'évolution des *mappings*, Y. Velegrakis et al. ont créé un outil appelé ToMAS (*Toronto Mapping Adaptation System*). De plus, ils proposent un autre outil appelé Clio [10] [167] qui permet la génération de *mappings* (cet outil illustre la mise en œuvre de l'opérateur `Matching` définit par P. A. Bernstein [21]). La figure 2.11 illustre l'outil Clio [167]. Cette figure montre un schéma source (`Source Schema`), le schéma cible (`Target Schema`) et les requêtes (`Query`). Cet outil est basé sur des standards ouverts et des standards commerciaux comme Java, DOM, XSLT et SQL. Un des modules fondamentaux de Clio est le *mapping engine* qui génère les *mappings* entre le schéma source et le schéma cible. Dans cet outil, le *mapping* est stocké dans une structure interne.

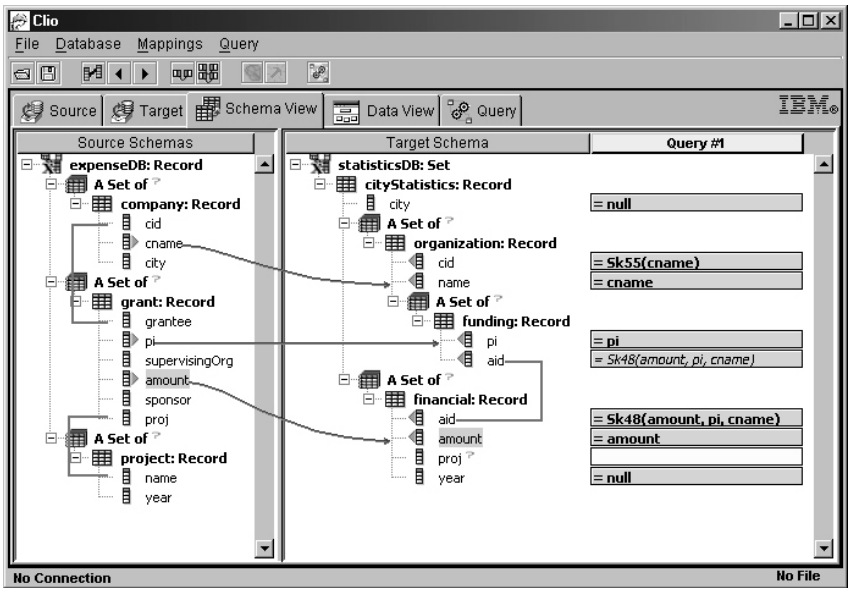


Figure 2.11 – L'outil Clio (génération semi-automatique de *mappings*) [167]

Un des problèmes auquel le *mapping*[2] doit faire face est la distance sémantique entre métamodèles. La notion de distance sémantique existe dans plusieurs domaines de l'informatique [24] [76] [95]. Elle est, d'une part, une propriété importante entre les différents métamodèles[3] (par exemple, un métamodèles utilisé pour capturer les notions d'un domaine et un autre métamodèle pour représenter la plate-forme technologique) et, d'autre part, elle est un paramètre important dans le processus de transformation de modèles [177].

[2]Dans le sens de spécification de correspondances.

[3]Dans un sens plus large, nous pouvons considérer formalismes, spécifications, etc.

Dans [44], G. Caplat et J. L. Sourrouille confirment l'importance de la distance sémantique dans la création de *mappings* :

« *The **mapping** effort depends on the **semantic distance** between notions that are involved within the patterns. This distance cannot be assessed by the comparison of metamodels because they only reflect the abstract syntaxes of the formalismes* » [44].

G. Caplat et J. L. Sourrouille

Dans [24], G. Booch, A. Brown, S. Iyengar, J. Rumbaugh et Bran Selic mettent l'accent sur l'importance de la prise en compte de la distance sémantique dans l'ingénierie de logiciel :

« *Reducing the **semantic distance** between problem domain and representation allows a more direct coupling of solutions to problems, leading to more accurate designs and increased productivity* » [24].

G. Booch, A. Brown, S. Iyengar, J. Rumbaugh et Bran Selic

Ainsi, une approche MDA et son automatisation devrait permettre la réduction de la distance sémantique entre les concepts d'un domaine et les technologies d'implémentation :

« *One of the primary purposes of automation in MDA is to bridge the **semantic gap** between domain concepts and implementation technology by explicitly modeling both domain and technology choices in frameworks and then exploiting the knowledge built into a particular application framework. (...) Perhaps the greatest difficulty associated with software development is the enormous **semantic gap** that exists between domain-specific concepts encountered in modern software applications, such as business process management or telephone call processing, and standard programming technologies used to implement them* » [24].

G. Booch, A. Brown, S. Iyengar, J. Rumbaugh et Bran Selic

Dans [177], O. Sims remarque que le problème de la distance sémantique dans l'ingénierie de logiciel est souvent résolu de façon informelle et en se basant sur les experts du domaine (heuristiques) :

« *The **gap** is often informally filled either by all business application developers learning a great deal about software technology, or by a few expert software technologists within a project team or shared across project teams* » [177].

O. Sims

Dans [21], P. A. Bernstein rappelle que la distance sémantique entre métamodèles peut être appréhendée par :

- la construction de métamodèles plus expressifs avec l'utilisation de relations tels que *is-a*, *data-types* et *keys*.
- l'utilisation d'expressions (tel que les requêtes - *queries*) dans les éléments de *mappings*.
- la création d'outils qui permettent l'ajout de sémantiques dans le mapping afin de réduire la distance sémantique entre les métamodèles.

L'objectif de cette section était de montrer que le problème de la spécification de *mapping* (dans le sens de correspondance) n'est pas un problème nouveau, mais qu'il a déjà été étudié il y a longtemps dans d'autres domaines comme les bases de données. En ce qui nous concerne, nous pensons que la définition de transformations doit être précédée d'une proposition qui prend en compte la spécification de correspondances dans le contexte de MDA.

2.3 La transformation de modèles

Dans cette section, nous présentons les démarches proposées pour fournir un langage de transformation selon les recommandations de l'OMG [142].

Depuis l'apparition de MDA, la transformation de modèles a pris une importance majeure. En fait, MDA est basée sur la définition de modèles indépendants de plates-formes (PIM) qui sont transformés en modèles spécifiques de plates-formes (PSM). MOF et UML fournissent une base bien établie pour définir des métamodèles pour PIM et PSM. Cependant, à ce jour, aucune proposition bien établie pour la définition de transformations dans le contexte de MDA n'est définitivement normalisée et utilisée en grande échelle. En 2002, l'OMG a démarré un processus de normalisation à travers son RFP MOF 2.0 *Query/-Views/Transformations* (QVT) [142] pour stimuler et créer un langage de transformation normalisé. Ce RFP n'est pas seulement destiné à la transformation, mais aussi à l'expression des requêtes (*Query*) et la définition de vue (*View*).

Dans [71], T. Gardner et al. définissent les éléments principaux du RFP QVT comme suit :

- *Query* (requête) : Une requête est une expression qui est évaluée sur un modèle. Le résultat d'une requête est une ou plusieurs instances de types définies dans le modèle source, ou définies par le langage de requête. Dans le contexte de MDA, l'OCL (*Object Constraint Langage*) est le langage le plus utilisé. D'autres langages peuvent aussi être utilisés comme UML *Action Semantics* [150].
- *View* (Vue) : Une vue est un modèle qui est complètement dérivé d'un autre modèle (le modèle de base). Une vue ne peut pas être modifiée séparément du modèle duquel elle est dérivée. Les changements dans le modèle de base seront obligatoirement reproduits dans la vue. Dans le cas où la vue peut être éditée, les changements dans la vue doivent être reproduits dans le modèle de base. Le métamodèle de la vue n'est pas exactement le même métamodèle que celui de la source. Une vue doit être complète et avoir le même contenu d'information que la source, mais réorganisée pour une tâche ou un utilisateur en particulier. Une requête est un type restreint de vue. Les vues sont générées via des transformations.
- *Transformation* (Transformation) : Une transformation génère un modèle cible à partir d'un modèle source. Les transformations peuvent mener à des modèles dépendants ou indépendants. Dans le premier cas, la transformation couple le modèle source au modèle cible. Dans le second cas, il n'y a pas de relations entre le modèle source et le modèle cible une fois que celui-ci est généré.

Parmi les termes utilisés dans le RFP QVT, nous citons :

- Langage déclaratif : terme général pour un langage relationnel ou fonctionnel, opposé à un langage impératif. Les langages impératifs spécifient explicitement les séquences et les étapes à suivre pour produire un résultat. En contrepartie, les langages déclaratifs décrivent les relations entre variables en termes de fonctions ou règles d'inférence, et un exécuteur de langage (interpréteur ou compilateur) applique un algorithme déterminé à ces relations pour produire le résultat.
- Langage impératif : tout langage de programmation qui spécifie la manipulation de l'état du système informatique.
- Langage hybride : combinaison de constructions déclaratives et impératives. Typiquement, l'approche déclarative est utilisée pour sélectionner les règles et les appliquer, et l'approche impérative est utilisée pour implémenter les détails des règles qui ne sont pas complètement exprimées de manière déclarative.
- Règles : unités dans lesquelles les transformations sont définies. Une règle est responsable de la transformation d'une sélection spécifique du modèle source en éléments correspondants du modèle cible. Une transformation est spécifiée par un ensemble de règles.

- Déclaration : spécification d'une relation entre les éléments du modèle de gauche et ceux du modèle de droite.
- Implémentation : spécification détaillée précisant comment créer les éléments cible à partir des éléments source.
- *Match* : se produit lors de l'application d'une transformation, plus précisément quand les éléments du modèle de gauche et du modèle de droite sont conformes aux spécifications de contraintes fournies par une règle.

Après la divulgation du RFP QVT, plusieurs langages de transformation ont été proposées dans la littérature [32] [117] [163] ou soumis directement à l'OMG [59] [91] [171].

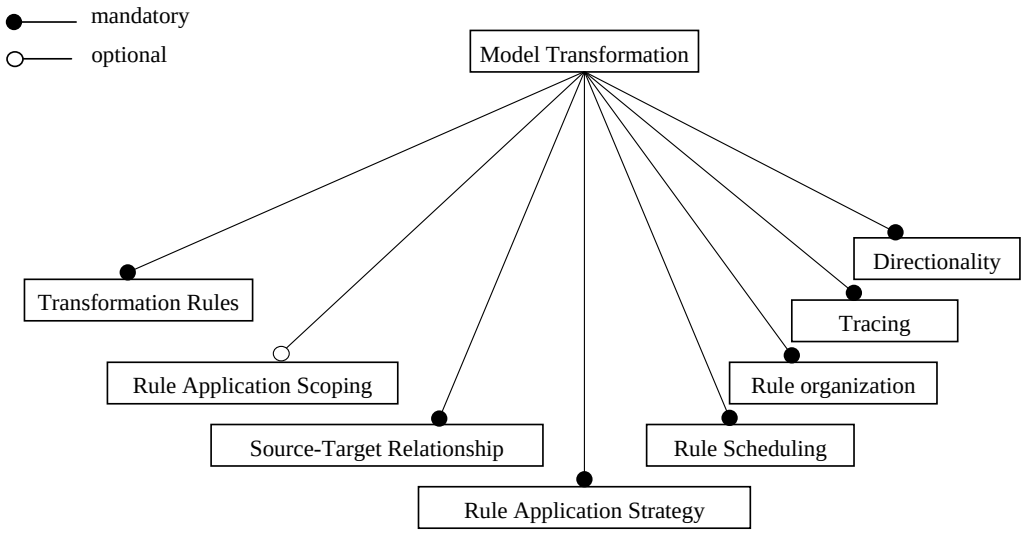


Figure 2.12 – Le diagramme des caractéristiques de langages de transformation de modèle [54]

Ces langages présentent des approches différentes pour résoudre le problème de définition de transformation de modèles. Dans [54], K. Czarnecki et S. Helsen proposent une taxonomie basée sur les caractéristiques présentes dans la majorité des propositions de langages de transformation. La figure 2.12 contient un diagramme de caractéristiques qui définit cette taxonomie pour les langages de transformation de modèles.

Selon la figure 2.12, les langages de transformation peuvent être distinguées en fonction de plusieurs paramètres :

- `Transformation Rules` (Règles de transformation) : une règle de transformation a obligatoirement deux parties distinctes, un LHS (*Left Hand Side*) et un RHS (*Right-Hand Side*). Le LHS permet d'accéder au modèle source, et le RHS permet d'accéder au modèle cible. Les LHS et RHS peuvent être une combinaison de :
  - `Variables` (Variables) : les variables sont utilisées pour manipuler les éléments du modèle source et/ou cible. Les variables peuvent être non typées, syntaxiquement typées ou sémantiquement typées.
  - `Patterns` (Patrons) : les patrons peuvent être classés suivant la forme, la syntaxe et le typage. La forme d'un patron peut être *strings*, termes ou graphes. Les patrons *string* sont utilisés en modèles textuels (*textual templates*). Les termes et les graphes sont utilisés dans la transformation modèle-vers-modèle. Quant à la syntaxe, elle peut être abstraite ou concrète. Un patron peut

être non typé, syntaxiquement typé ou sémantiquement typé.
  - `Logic` (Logique) : la logique peut être non exécutable ou exécutable. La logique non exécutable est utilisée pour spécifier une relation entre modèles. L'exécutable peut être impérative ou déclarative. La forme déclarative comprend les requêtes OCL pour la récupération des éléments d'un modèle et la création implicite des éléments cibles par le biais des contraintes. La forme impérative contient un type de langage de programmation qui fait appel à une API pour manipuler les modèles directement, par exemple JMI (*Java Metadata Interface*) [94] fournit une API pour accéder à un dépôt MOF (*repository* MOF).

  De plus, une règle de transformation peut avoir les caractéristiques suivantes :
  - `LHS/RHS Syntatic Separation` (Séparation Syntaxique) : la syntaxe d'une règle peut faire la distinction explicite entre un RHS et un LHS ou cette distinction peut ne pas avoir lieu.
  - `Bidirectionality` (Bidirectionnalité) : une règle peut être exécutable dans une seule direction ou dans les deux directions (bidirectionnelle). La règle bidirectionnelle est plus complexe et plus difficile à implémenter [101].
  - `Parameterization` (Paramétrage) : les règles de transformation peuvent avoir des paramètres de contrôle additionnels pour permettre la configuration ou l'adaptation.
  - `Intermediate Structures` (Structures intermédiaires) : quelques approches demandent la construction de structures de modèles intermédiaires.
- `Rule Application Scoping` (Portée de l'application de la règle) : permet à une transformation de restreindre les parties d'un modèle qui participe à la transformation.
  - `Source` (Source) : dans ce cas, nous pouvons établir une portée moindre que celle du modèle source entier.
  - `Target` (Cible) : dans ce cas, nous pouvons augmenter la portée du modèle cible.
- `Source-Target Relationship` (Relation Source-Cible) : plusieurs approches imposent la création d'un nouveau modèle cible qui doit être séparé du modèle source. D'autres approches imposent que le modèle source et le modèle cible soient le même modèle (elles permettent uniquement la mise à jour sur place, ou *in-place update*).
- `Rule Application Strategy` (Stratégie d'application de la règle) : une règle doit être appliquée depuis une localisation spécifique à l'intérieur de la portée du modèle source. Dans le cas où elle est activée depuis plus d'une localisation, la stratégie de l'application de la règle doit être spécifiée. Elle peut être :
  - `Deterministic` (Déterministe) : la stratégie suit un protocole déterminé et connu d'avance.
  - `Non-deterministic` (Non déterministe) : une localisation peut être choisie ou plusieurs peuvent être appliquées de façon concurrente.
  - `Interactive` (Itérative) : l'utilisateur doit dire quelle localisation doit être appliquée à la règle.
- `Rule scheduling` (Ordonnancement d'une Règle) : détermine l'ordre dans lequel les règles individuelles sont appliquées. Le mécanisme de *scheduling* peut être classé en quatre groupes :
  - `Form` : l'aspect de l'ordonnancement peut être implicite et explicite. L'ordonnancement implicite implique que l'utilisateur n'a pas un contrôle explicite dans l'algorithme d'ordonnancement. L'ordonnancement explicite fournit des constructions dédiées au contrôle de l'ordre d'exécution.
  - `rule selection` : les règles peuvent être sélectionnées par une condition spécifique, ou choisies de façon non déterministe, ou sélectionnées par un mécanisme de priorité, ou encore sélectionnées de manière interactive.
  - `rule iteration` : la récursivité, la boucle et le point fixe (*fixpoint*) se trouvent parmi les mécanismes d'itération de règles.

  - `phasing` : cet aspect fait référence à l'organisation du processus de transformation en plusieurs étapes.
- `Rule organization` (Organisation de règles) : détermine la composition et la structuration de plusieurs règles de transformation. Elle peut être classée en :
  - `Modularity mechanism` (mécanisme de modularité) : permet l'organisation de règles en modules. La création de modules permet à un module d'importer les éléments d'un autre module.
  - `Reuse mechanism` (mécanisme de réutilisation) : il fournit une manière de définir une règle basée sur une ou plusieurs autres règles. Parmi les mécanismes de réutilisation, nous citons : composition logique, héritage entre règles, dérivation, extension, spécialisation, héritage entre modules.
  - `Organizational structure` (structure d'organisation) : les règles peuvent être organisées en conformité avec la structure du métamodèle source, ou du métamodèle cible, ou encore avoir une organisation spécifique.
- `Tracing` (`Traceability`) (Trace) : les transformations peuvent enregistrer des liens entre les éléments source et cible. Ces liens peuvent être utiles pour l'analyse de la performance, la synchronisation entre modèles, les débuggeurs orientés modèle, et la détermination de la cible d'une transformation.
- `Directionality` (Directionnalité) : indique la direction d'une transformation.
  - `Unidirectional` (unidirectionnelle) : la transformation est exécutée dans une seule direction, dans laquelle un modèle cible est déduit à partir d'un modèle source.
  - `Bidirectional` (bidirectionnelle) : la transformation peut être exécutée dans les deux directions. Elle fournit également le support nécessaire pour la synchronisation entre modèles.

Dans la création de cette taxonomie, K. Czarnecki et S. Helsen ont considéré les approches modèle-vers-modèle et modèle-vers-code de façon uniforme. La première approche est encore en évolution et en expérimentation alors que la dernière est la plus diffusée et utilisée. Cependant, elles présentent plusieurs similarités. En général, la transformation de modèle vers du code peut être vue comme une transformation modèle vers modèle, puisque le code est aussi un modèle. Cependant, pour des raisons pratiques de réutilisation de la technologie de compilateur existante, le code est souvent généré comme texte simple, qui peut être compilé.

L'approche modèle-vers-code peut être :

- `Visitor-based` (orientée visiteur) : cette approche est la plus basique et consiste à fournir un mécanisme de visiteur pour s'affranchir de la représentation interne d'un modèle et écrire le code comme un texte.
- `Template-based` (orientée modèle) : un *template* consiste habituellement en un texte cible qui contient des métacodes pour récupérer l'information d'un modèle source et les utiliser pour remplir le texte cible.

L'approche modèle-vers-modèle peut être :

- `direct-manipulation` (manipulation directe) : cette approche fournit une représentation interne de la représentation du modèle ainsi qu'une API pour la manipuler. Elle est habituellement réalisée comme un *framework* orienté objet.
- `relational` (relationnelle) : cette catégorie groupe les approches déclaratives dans lesquelles les relations mathématiques constituent le concept principal. L'idée sous-jacente est d'établir un type de relation entre un élément source et cible et de la spécifier en utilisant des contraintes.
- `graph-transformation-based` (orientée transformation de graphe) : cette catégorie utilise la théorie des graphes.
- `structure-based` (orientée structure) : dans cette catégorie, deux étapes peuvent être distin-

guées : l'une concerne la création de l'architecture hiérarchique du modèle cible, et l'autre l'établissement des valeurs des attributs et des références dans le modèle cible.
- `hybrid-based` (hybride) : est une composition des approches déclarative et impérative.
- approches spécifiques : Dans la spécification CWM (*Common Warehouse Metamodel*) de l'OMG un framework de transformation est proposé. Ce *framework* fournit un mécanisme pour lier les éléments source et cible, mais la dérivation des éléments cibles doit être réalisée dans un langage concret, lequel n'est pas défini par CWM. Une autre approche significative est XSLT. Depuis que les modèles peuvent être sérialisés à la façon de XML (par exemple, XMI), XSLT peut être utilisé pour réaliser la transformation. Cependant, cette approche n'est pas adaptée à la transformation de modèles car elle est source d'erreurs dans la définition, présente une forte verbosité, et offre une visibilité réduite due à la faible compacité de XMI.

Dans ce document, nous nous intéressons aux langages de transformation de modèles ATL [32], YATL [163] et BOTL [117]. Nous présenterons ces langages, puis les analyserons selon la taxonomie mentionnée précédemment.

2.3.1 ATL

ATL[4] (*Atlas Transformation Language*) est un langage pour la réalisation de transformations de modèles dans le contexte de MDA [32]. La première réalisation d'ATL est basée sur MDR (*MetaData Repository*) [131] et Java [32], tandis que la deuxième est basée sur MDR, EMF (*Eclipse Modeling Framework*), Java et Eclipse [14]. Elles présentent globalement les mêmes caractéristiques que nous exposons dans cette section de manière non distincte.

Une requête en ATL est une expression en OCL qui peut retourner des types primitifs, des éléments d'un modèle, des collections, des n-uplets, ou une combinaison de ces types (par exemple, collections de n-uplets). La version actuelle d'ATL ne supporte ni la transformation incrémentale, ni la bidirectionnalité. Cependant, les concepteurs d'ATL préconisent l'utilisation du support de traçabilité (*traceability*) pour réaliser cette caractéristique en ATL. La transformation en ATL est unidirectionnelle. ATL est un langage hybride (déclaratif et impératif). L'approche déclarative d'ATL est basée sur OCL. L'approche impérative en ATL contient des instructions qui explicitent les étapes d'exécution dans une procédure (*helpers*). La figure 2.13 présente la syntaxe abstraite de règles de transformation en ATL. Selon la syntaxe abstraite, une règle est définie par un nom et peut contenir les éléments `ActionBlock` et `OutPattern`. Une `MatchedRule` spécialise une `Rule` et contient un `InPattern`. Une `CalledRule` spécialise une `Rule` ou une `Operation` en OCL.

Une règle est explicitement appelée en utilisant son nom et ses paramètres (une `CalledRule`), ou exécutée comme un résultat de la reconnaissance d'un `inPattern` dans le modèle source (une `MatchedRule`). Le résultat de l'exécution d'une règle peut être déclaré en utilisant un `outPattern`, de manière déclarative, impérative ou les deux.

Une règle formée d'un `inPattern`, d'un `outPattern` et sans section impérative est définie comme une règle déclarative. Une règle formée avec un nom, des paramètres et une section impérative, mais sans un `outPattern`, est définie comme une règle impérative. La combinaison des deux formes est simplement appelée règle hybride.

La figure 2.14 présente les détails de la relation entre un `InPattern` et un `OutPattern`. En ATL, un `MatchedRule` spécifie un patron source (`inPattern`) comme un ensemble de types d'origine du métamodèle source associés à l'ensemble des noms de variables et optionnellement filtrés en utilisant

[4] ATL a été conçu, développé et implémenté par J. Bézivin et F. Jouault.

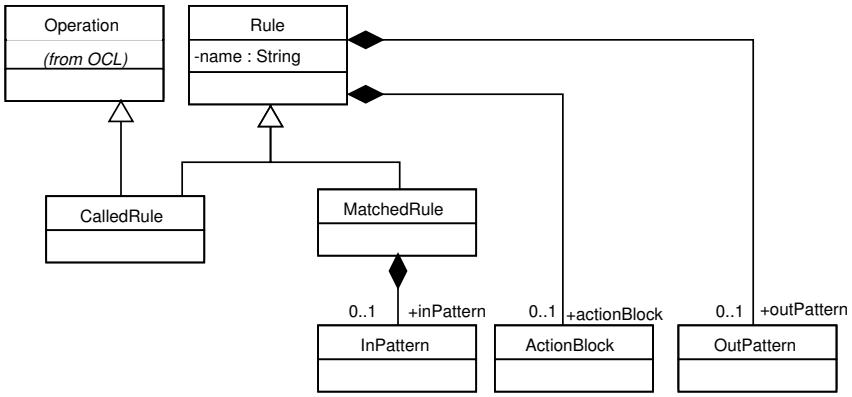


Figure 2.13 – Règles en ATL (syntaxe abstraite) [32]

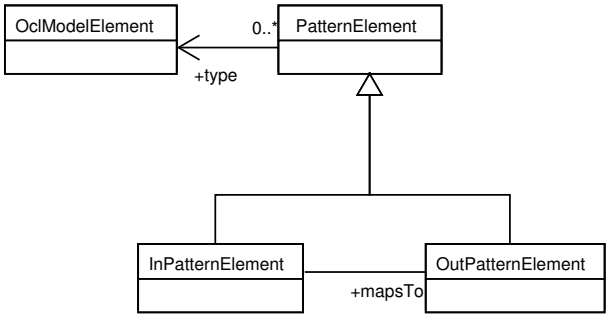


Figure 2.14 – Patrons d'éléments (syntaxe abstraite) [32]

une expression logique en OCL. Un patron cible (`outPattern`) est un ensemble de types d'origine du métamodèle cible associé à des noms de variables et des liaisons (*bindings*). Quand une règle contenant un patron cible est exécutée, les éléments cibles sont créés. Une liaison spécifie la valeur utilisée pour initialiser une propriété d'une instance. En fait, un patron cible est un ensemble de nœuds qui peuvent être liés par des liaisons.

En ATL, les constructions déclaratives sont constituées d'un `InPattern` représenté par le mot clé `from`, d'un `outPattern` représenté par le mot clé `to`, et d'un ensemble de liaisons représentées par le symbole `<-`. L'exemple 2.1 présente une règle complètement déclarative en ATL.

Exemple 2.1 – Une règle complètement déclarative en ATL

```
module UML2JAVA ;
        create OUT : JAVA from IN : UML ;
        rule Class2JClass{
                from uclass : UML !Class
                to jclass : JAVA !JClass(
                   uclass.name <- jclass.name
                )
}
```

Les instructions impératives en ATL peuvent être regroupées en :

- Expressions : Les expressions sont basées sur OCL.
- Variables : Une déclaration de variable est faite par le mots clé `let`. Par exemple :

```
let VarName : varType = initialValue ;
```

- Attribution : L'opérateur de *binding* <- peut être utilisé à cette fin. De plus, ATL fournit l'opérateur := pouvant être utilisé quand une résolution automatique n'est pas nécessaire. Ceci permet à l'opérande de gauche de prendre la valeur de celle de droite.
- Manipulation d'instances : Les instances peuvent être créées de manière implicite grâce à l'approche déclarative. De plus, il est possible d'expliciter la création ou la destruction d'une instance en utilisant l'opérateur `new` et `delete`. Par exemple :
  - Création d'une instance :
    ```
    let jClass : Java !JClass = new Java !JClass() ;
    ```
  - Suppression d'une instance :
    ```
    delete jClass ;
    ```
- Déclarations conditionnelles :
  - `if ... then ... else ... endif`
    à la manière OCL.
  -
    ```
    if (condition) {
    -- statement 1
    }else if (condition 2){
    -- statement 2
    }else{
    -- statement 3
    }
    ```
  -
    ```
    switch (expression){
    case expression 1:
      --statement 1
    break;
    case expression 2:
      --statement 2
    break;
    default:
      --statement 3
    break;
    }
    ```
- déclarations de boucle : ATL permet la déclaration de boucles `while`, `do ... while(condition)` et `foreach`. Par exemple :
  -
    ```
    while (condition) {
    -- statement
    }
    ```
  -
    ```
    do{
    -statement
    }while(condition);
    ```
  -
    ```
    foreach element in collection{
    --statement
    }
    ```

De plus, ATL fournit le mécanisme de *Helpers* pour éviter la redondance de code et la création de grandes expressions OCL dans une règle. Ceci induit aussi une meilleure lisibilité des programmes ATL.

Un *helper* en ATL est une fonction qui peut recevoir des paramètres et retourner une valeur ou une instance d'un élément. Les *helpers* sont toujours utilisés dans le contexte d'une transformation ou pour d'autres *helpers*. L'exemple 2.2 présente l'utilisation d'un *helper*.

Exemple 2.2 – Déclaration et utilisation d'un helper en ATL

```
module UML2JAVA ;
create OUT : JAVA from IN :UML ;

helper existsAttributes (s:Sequence(UML!Attribute)):
Integer=
        if s-> size()>0 then
                1
        else
                0
        endif;

rule Class2JClass{
        from c: UML!Class
        to jclass:JAVA!JClass(
                jclass.name <- c.name,
                [...] if (existsAttributes(c.getFeature()) then
                        statement1
                    else
                        statement 2
                    endif
        )
}
```

Selon la taxonomie créée par K. Czarnecki et S. Helsen [54], ATL a les caractéristiques suivantes :

- *transformation rule* (règles de transformation) : ATL fait bien la distinction entre LHS et RHS. Une règle en ATL est une combinaison de variables syntaxiquement typées et d'une logique exécutable basée sur OCL.
  - `LHS/RHS Syntatic Separation` (Séparation Syntaxique) : LHS/RHS ont la même syntaxe.
  - `Bidirectionality` (Bidirectionnalité) : ATL est unidirectionnel.
  - `Parameterization` (Paramétrage) : ATL ne supporte pas le paramétrage.
  - `Intermediate Structures` (Structures intermédiaires) : ATL ne nécessite pas une structure intermédiaire explicite.
  - `Rule Application Scoping` (Portée de l'application de la règle) : ATL ne supporte pas cette caractéristique.
  - `Source-Target Relationship` (Relation Souce-Cible) : en ATL, la source et la cible sont des modèles distincts.
- `Rule Application Strategy` (Stratégie d'application de la règle) : en ATL, la stratégie est déterministe.
- `Rule scheduling` (Ordonnancement de règles) : non connu.
- `Rule organization` (Organisation de règles) : ATL supporte la modularité.
- `Tracing` (`Traceability`) (Trace) : ATL supporte la traçabilité (traceability) par le biais de l'enregistrement des étapes de transformation réalisées par le moteur de transformation.
- `Directionality` (Directionnalité) : une transformation en ATL est unidirectionnelle.

Par rapport à la classification générale, ATL est un langage permettant d'effectuer la transformation modèle-vers-modèle avec les caractéristiques suivantes :

- `direct-manipulation` (manipulation directe) : ATL n'est pas un langage permettant la manipulation directe.
- `relational` (relationnelle) : ATL étant en principe déclaratif, il est aussi relationnel (les contraintes de ces relations sont spécifiées en OCL).
- `graph-transformation-based` (orienté transformation de graphe) : ATL n'est pas directement basé sur la théorie des graphes.
- `structure-based` (orienté structure) : ATL n'est pas orienté structure.
- `hybrid-based` (hybride) : ATL contient des caractéristiques déclaratives et impératives.

2.3.2 YATL

YATL (*Yet Another Transformation Language*) est un langage de transformation développé à l'intérieur de KMF (*Kent Modelling Framework*) [162] [163]. Le compilateur et l'interprétateur pour YATL sont réalisés en Java. Certaines parties de ces processeurs de langage ont été construites en utilisant :

- SDTS (*Syntax-Driven Translation Scheme*), analyseurs lexiquaux et générateurs de parseurs, pour générer automatiquement un analyseur lexiqual, le parseur et le translateur.
- KMF *Studio* et un modèle MOF de la syntaxe abstraite pour générer le code Java associé à la syntaxe abstraite de YATL.

De plus, YATL a été conçu avec les objectifs suivants :

- Le moteur de transformation doit être capable de réaliser des transformations efficaces de systèmes à grande échelle.
- Le processus d'application des règles de transformation doit être déterministe.
- La syntaxe et la sémantique de YATL doivent être bien définies.
- YATL doit être un langage hybride (il doit contenir l'approche déclarative et impérative).
- *Queries*, *Views* et transformations sont organisées en *namespaces* pour permettre la réutilisation et éviter la collision de noms.

Chaque transformation en YATL contient une ou plusieurs règles de transformation. Une règle de transformation est constituée de deux parties : un LHS (*Left Hand Side*) et un RHS (*Right-Hand Side*). Le LHS d'une transformation en YATL est spécifié par une expression de filtrage écrite en OCL ou un code natif (par exemple, Java). Cette approche permet à YATL de décrire des filtres en utilisant les informations du modèle et les propriétés de la plate-forme (par exemple, les fonctions de conversion). Le LHS et RHS en YATL sont décrits avec la même construction syntaxique.

La figure 2.15 présente la syntaxe abstraite de YATL [161].

La syntaxe abstraite de YATL est constituée de :

- `NamedElement` : Élément de base pour `Model`, `Namespace`, `Transformation` et `Rules`.
- `Namespace` : La création de règles de transformation est organisée en espace de noms (`namespaces`). Un `namespace` contient les transformations entre un `sourceModel` et `targetModel`.
- `Model` : Il représente le modèle source ou cible.
- `Query` : Identique à une requête OCL.
- `Transformation` : Transformation contenant les règles.
- `Rule` : Règle contenant un `CompoundStm` et un `filter`. Le premier élément établit la relation entre le LHS et le RHS, le second élément est une expression OCL qui fait le filtrage de l'exécution des règles.
- `Filter` : Il est appliqué à une règle pour contrôler son exécution.
- `CompoundStm` : Il contient la déclaration des parties LHS et RHS et le corps de la règle.

- `Unit` : Il met en relation un `namespace`, un `import` et une `Rule`.
- `Import` : Il réalise la réutilisation d'autres règles de transformations dans un autre `namespace`.

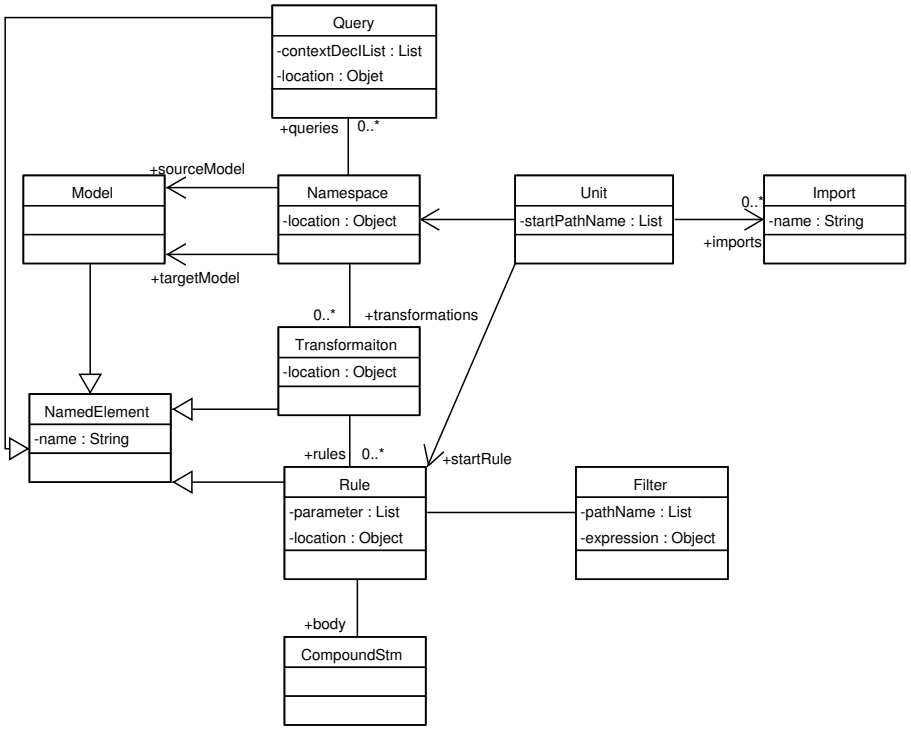


Figure 2.15 – Syntaxe Abstraite de YATL [161]

La partie déclarative de YATL est basée sur les expressions OCL et la description de la partie LHS des règles de transformation. L'approche déclarative en YATL permet la réalisation de requêtes sur les modèles, tandis que la partie impérative permet la manipulation des informations obtenues par les requêtes.

La partie impérative de YATL est constituée d'opérations de contrôle de cycle de vie telles que la création et la destruction, d'opérations de changement des valeurs de propriétés, de déclarations, d'opérations de décision, de déclarations d'itérations et de déclarations natives. Cette dernière permet l'interaction avec la plate-forme (par exemple, Java). De plus, YATL utilise la syntaxe des expressions OCL pour spécifier des opérations de base telles que l'addition de deux valeurs entières. Une déclaration complète en YATL contient une séquence d'instructions qui sont exécutées dans un ordre défini. La syntaxe abstraite de déclarations de YATL est décrite par la figure 2.16.

La syntaxe de YATL :

- Déclaration de variables : Une déclaration de variable en YATL suit la syntaxe d'OCL :
  `let varName : oclType = oclExpression ;`
- Déclaration d'attribution : réalise l'affectation de la valeur de l'opérande de droite par la variable ou une propriété donnée par l'opérande de gauche.
  `oclExpression1 := oclExpression2 ;`
- Instructions de cycle de vie : YATL supporte la création et la destruction explicite des objets de manière impérative. La création est faite par la déclaration `new` et la destruction par la déclaration `delete`.

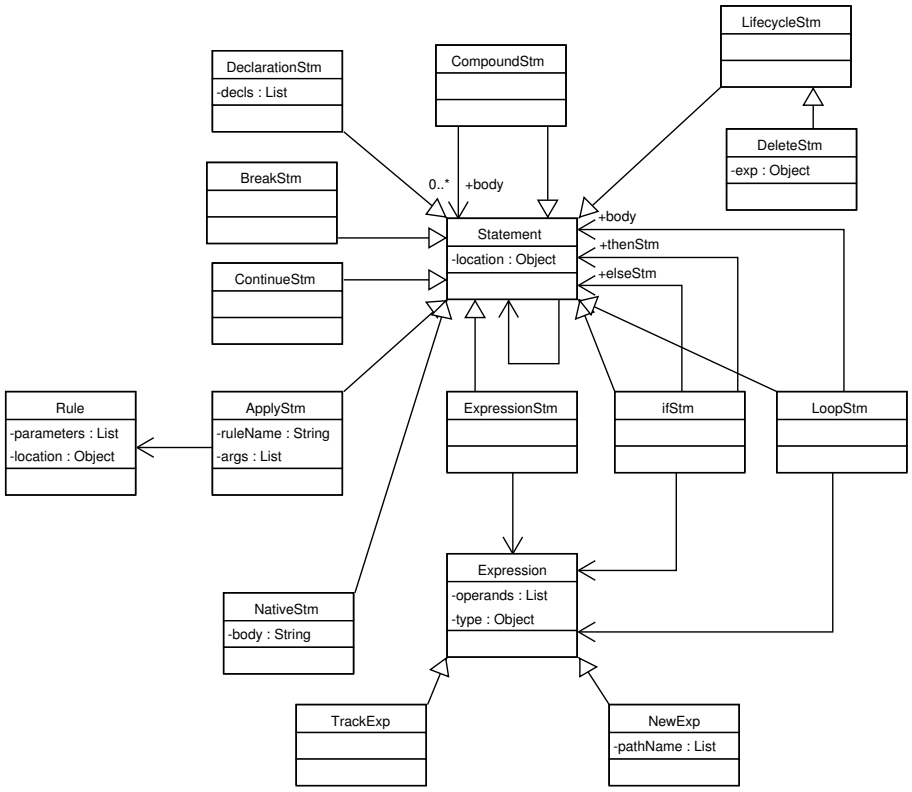


Figure 2.16 – La syntaxe abstraite de déclaration en YATL [163]

```
varName := new pathname ;
delete oclExpression ;
```

- Déclaration conditionnelle :
  `if condition then statement else statement endif`
- Déclarations de boucle : YATL supporte les boucles
  `while ...do, do...until et foreach...in ...do`
  comme suit :

```
while condition do statement

do statement until condition

foreach variableDeclaration in oclExpression do statement
```

- Déclaration native : le support de calcul de YATL est basé sur OCL, mais il n'est pas capable à lui seul de traiter les systèmes à grande échelle. Cette limitation est minimisée grâce à un mécanisme de communication qui permet l'interaction entre YATL et la plate-forme (langage) dans laquelle l'exécution de YATL est achevée, par exemple, Java. Les déclarations natives en YATL ont la syntaxe suivante :

```
native {
code for the host platform (Java, C#, etc.) }
```

- Patron de construction (*building pattern*) : crée un objet complexe, par exemple une instance d'une

classe `Book` ayant deux propriétés, `name` et `pages` :
`build Book { name= 'test', pages = 5}`

- Déclarations d'expressions : les expressions en YATL sont écrites en OCL. De plus, YATL contient deux nouveaux types d'expressions : `new` et `track`. L'expression `new` est utilisée pour créer une instance de type spécifié. L'expression `track` est utilisée pour stocker et récupérer les correspondances (*mappings*) en temps d'exécution.

L'exemple 2.3 présente une règle de transformation en YATL qui transforme une classe UML en une classe Java.

Exemple 2.3 – La transformation en YATL d'une classe UML en une classe Java

```
start kmf::uml2java::main;
namespace kmf(uml, javaModel) {
transformation uml2java {
        -- 1-1 Mappings
        -- Map a UML class to a Java class
        rule umlClass2JavaClass match uml::Foundation::Core::Class () {
                -- Create Java class
        let jClass: javaModel::JavaClass;
        jClass := new javaModel::JavaClass;
        -- Set name
        jClass.name := self.name.body_;
        -- Store mapping
        track(self, class2class, jClass);
}
```

Selon la taxonomie créée par K. Czarnecki et S. Helsen [54], YATL a les caractéristiques suivantes :

- `Transformation Rule` (Règles de Transformation) : YATL fait bien la distinction entre LHS et RHS. Une règle en YATL est une combinaison de variables syntaxiquement typées et d'une logique exécutable basée sur OCL et/ou sur la plate-forme native (par exemple, Java ou C#).
  - `LHS/RHS Syntatic Separation`(Séparation Syntaxique) : LHS/RHS ont la même syntaxe.
  - `Bidirectionality` (Bidirectionnalité) : YATL est unidirectionnel.
  - `Parameterization` (Paramétrage) : YATL supporte le paramétrage.
  - `Intermediate Structures` (Structures intermédiaires) : il ne demande pas une structure intermédiaire explicite.
  - `Rule Application Scoping` (Portée de l'application de la règle) : YATL ne fournit pas cette caractéristique.
  - `Source-Target Relationship` (Relation Souce-Cible) : en YATL, la source et la cible sont des modèles distincts.
- `Rule Application Strategy` (Stratégie d'application de la règle) : YATL est déterministe.
- `Rule scheduling` (Ordonnancement de règles) : non connu.
- `Rule organization` (Organisation de règles) : YATL fournit la modularité.
- `Tracing` (Traceability) : YATL supporte la traçabilité grâce à l'opérateur `track`.
- `Directionality` (Directionnalité) : une transformation en YATL est unidirectionelle.

Par rapport à la classification générale, YATL est un langage permettant d'effectuer la transformation modèle-vers-modèle avec les caractéristiques suivantes :

- `direct-manipulation` (manipulation directe) : YATL n'est pas un langage permettant la manipulation directe, mais il fournit cette caractéristique par le biais des déclarations natives.
- `relational` (relationnelle) : YATL étant en principe déclaratif, il est aussi relationnel (les contraintes de ces relations sont spécifiées en OCL).

- `graph-transformation-based` (orienté transformation de graphe) : YATL n'est pas directement basé sur la théorie des graphes.
- `structure-based` (orienté structure) : YATL n'est pas orienté structure.
- `hybrid-based` (hybride) : YATL contient des caractéristiques déclaratives et impératives.

2.3.3 BOTL

Le BOTL (*Basic Object-oriented Transformation Language*) est un langage de transformation qui a été défini mathématiquement et qui utilise une notation à la UML pour la définition de règles de correspondances [28] [117]. Une des réalisations de BOTL utilise MOF et est une extension d'ArgoUML écrite en Java.

La définition de transformation en BOTL est mathématiquement formalisée, ce qui lui donne une base rigoureuse et précise. Ce langage permet d'utiliser des techniques de description graphique et de description d'algorithmes pour définir un ensemble de règles de correspondances. En BOTL, chaque règle définit une correspondance entre un fragment d'un modèle source et un fragment d'un modèle cible. Chaque règle est constituée d'un LHS et d'un RHS qui sont représentées comme des variables de modèles. Le formalisme de BOTL présenté par P. Braun et F. Marschall, peut être résumé par les définitions suivantes [117] :

Définition 1 : Un métamodèle mm est un n-uplet d'un ensemble de $types$, $class\ allocation$, et un ensemble de $class\ associations$. Toutes les classes ont un nom unique. Les noms des attributs dans une classe sont aussi uniques. Toutes les $class\ associations$ connectent des éléments $class\ allocation$. Les types de tous les attributs sont des éléments de l'ensemble de $types$.

Définition 2 : similaire à la définition d'un métamodèle, un modèle est un n-uplet $(mm,\ OS,\ OA)$ consistant en un métamodèle mm, une allocation d'objet OS, et un ensemble des associations d'objets OA.

Définition 3 : une variable de modèle mv est un n-uplet $(mm,\ OVS,\ OVA)$ constitué d'un métamodèle mm, d'une allocation de variable d'objet OVS, et d'un ensemble d'associations de variables d'objet OVA. Les variables font référence à des classes, et les associations de variables d'objets font référence à des associations de classes du métamodèle mm. Toutes les associations de variables d'objets connectent des variables d'objets OVS.

Définition 4 : Une règle de transformation r_i est un n-uplet $(mv_0,\ mv_1)$ composé de deux variables d'objet. Un ensemble de règles de transformation est une séquence finie de règles.

Définition 5 : Un fragment de modèle mf est un n-uplet $(OS,\ OA)$ constitué d'une allocation d'objet OS et d'un ensemble d'associations d'objets OA. Toutes les associations d'objets connectent des objets OS.

Définition 6 : Un *match* mfm_i d'un fragment de modèle est un n-uplet $(mv, mf, match_o, match_a)$ composé d'une variable de modèle mv, un fragment de modèle mf, et deux fonctions bijectives $match_o : OVS \rightarrow OS$ et $match_a : OVA \rightarrow OA$. Un $match_o$ prend une variable d'objet et retourne un objet, toute variable d'objet OVS est donc transformée bijectivement en un objet OS de même type. Un $match_a$ prend une association de variables d'objets et retourne une association d'objets. Toutes les associations de variables d'objet OVA sont donc transformées bijectivement dans une association d'objet OA avec les mêmes terminaisons d'associations et la même cardinalité.

Définition 7 : Une application de règle est une fonction
$apply(m, r_i, mf_o) \mapsto mf_1$
Une application de règle applique une règle r_i à un modèle source m et fusionne le fragment de modèle résultant avec le fragment existant mf_o.
Définition 8 : Une application d'un ensemble de règles est une fonction
$transform(m, r) \mapsto mf_r$ où
$mf_0 = (\emptyset, \emptyset)$ et
$mf_i = apply(m, r_{i-1}, mf_{i-1})$ pour chaque $i \in \{1..|r|\}$
$transform$ est une transformation de modèle si $mf|r|$ est un modèle valide écrit $r|mv_1|_{mm}$.

Selon la taxonomie créée par K. Czarnecki et S. Helsen [54], BOTL possède les caractéristiques suivantes :

- `Transformation Rule` (Règles de Transformation) : BOTL fait bien la distinction entre LHS et RHS.
  - `LHS/RHS Syntatic Separation` (Séparation Syntaxique) : LHS/RHS ont la même syntaxe.
  - `Bidirectionality` (Bidirectionnalité) : BOTL est bidirectionnel.
  - `Parameterization` (Paramétrage) : BOTL ne supporte pas le paramétrage.
  - `Intermediate Structures` (Structures intermédiaires) : Il ne demande pas une structure intermédiaire explicite.
  - `Rule Application Scoping` (Portée de l'application de la règle) : BOTL ne fournit pas cette caractéristique.
  - `Source-Target Relationship` (Relation Souce-Cible) : En BOTL, la source et la cible sont un seul modèle.
- `Rule Application Strategy` (Stratégie d'application de la règle) : BOTL est non déterministe.
- `Rule scheduling` (Ordonnancement de règles) : Il est non déterministe. Cependant, le résultat de l'exécution de règles de transformation est toujours déterministe.
- `Rule organization` (Organisation de règles) : il ne fournit pas cette caractéristique.
- `Tracing` (Traceability) : Il supporte la traçabilité de façon indirecte en utilisant la création de règles inverses pour trouver les étapes de la transformation.
- `Directionality` (Directionnalité) : Une transformation est bidirectionnelle comme son nom le suggère.

Par rapport à la classification générale, BOTL est un langage permettant d'effectuer la transformation modèle-vers-modèle avec les caractéristiques suivantes :

- `direct-manipulation` (manipulation directe) : BOTL n'est pas un langage permettant la manipulation directe de modèles.
- `relational` (relationnelle) : BOTL étant déclaratif, il est aussi relationnel.
- `graph-transformation-based` (orienté transformation de graphe) : BOTL n'est pas directement basé sur la théorie des graphes. Cependant, comme il est représenté à la UML, il a une relation indirecte avec la théorie des graphes.
- `structure-based` (orienté structure) : BOTL n'est pas orienté structure.
- `hybrid-based` (hybride) : BOTL ne contient que des caractéristiques déclaratives.

2.3.4 Une évaluation des langages de transformation

Le tableau 2.1 présente un comparatif des langages de transformation ATL [32], YATL [162] et BOTL [117], ainsi que leurs caractéristiques selon la taxonomie présentée précédemment.

Caractéristique	**ATL**	**YATL**	**BOTL**
Transformation rule :	distinction entre LHS et RHS	distinction entre LHS et RHS	distinction entre LHS et RHS
- LHS/RHS Syntatic Separation	syntaxe identique	syntaxe identique	syntaxe identique
- Bidirectionality	non explicitement	unidirectionnelle	bidirectionnelle
- Parameterization	non	oui	non
- Intermediate Structures	non	non	non
- Rule Application Scoping	non	non	non
- Source-Target Relationship	modèles distincts	modèles distincts	modèles identiques
Rule Application Strategy	déterministe	déterministe	non déterministe
Rule scheduling	non connu	non connu	non connu
Rule organization	modularité	modularité	aucune
Tracing (Traceability)	indirectement	oui	indirectement
Directionality	unidirectionnelle	unidirectionnelle	bidirectionnelle
Direct-manipulation	non	possibilité avec `native`	non
Relational	oui	oui	oui
Graph-transformation-based	non	non	partiellement
Structure-based	non	non	non
Hybrid-based	oui	oui	non, uniquement déclaratif

Table 2.1 – Les caractéristiques de ATL, YATL et BOTL

Les langages de transformation ATL et YATL présentent plusieurs similarités. Par exemple, ils sont hybrides (déclaratifs et impératifs), basés sur OCL et fournissent un mécanisme de modularité.

En ATL, une règle de transformation n'est pas explicitement bidirectionnelle, mais ATL fournit des supports pour permettre la création des règles bidirectionnelles. En YATL, une règle de transformation est unidirectionnelle. La traçabilité est assurée en YATL. Par contre, en ATL, l'historique des transformations peut être enregistré pour supporter la traçabilité. YATL permet l'interaction avec la plate-forme de base, par exemple C ou Java, par le biais de l'instruction `native`. ATL ne fournit pas un mécanisme d'interaction avec la plate-forme de base, afin d'assurer la portabilité des définitions de transformations.

Le langage BOTL présente un ensemble de caractéristiques différentes de ATL et YATL. Par exemple, BOTL est complètement déclaratif. En BOTL, une transformation est bidirectionnelle. Dans une transformation définie en BOTL, le modèle source et le modèle cible sont les mêmes. BOTL est partiellement basé sur la transformation de graphes. De plus, ATL et YATL sont des langages textuels, alors que BOTL est un langage graphique.

2.4 Application de MDA pour la plate-forme des services Web

L'application de MDA pour créer des modèles dépendants de plates-formes à partir de modèles indépendants de plates-formes n'est pas une tâche simple. Plusieurs questions sont encore sans réponses, mais l'apparition de plusieurs langages de transformation [32] [161] a servi à valider en partie l'approche MDA. Cependant, nous percevons qu'il manque encore une théorie bien établie pour supporter MDA [63].

Pour répondre à plusieurs des questions encore ouvertes, l'expérimentation de MDA avec plusieurs langages de modélisation et plates-formes cibles doit permettre d'en apprendre plus sur ces problèmes existants et de proposer des solutions viables. Parmi les langages de modélisation, EDOC et UML sont les plus connus et les plus adaptés pour modéliser les systèmes distribués. Parmi les plates-formes cibles, les services Web, CORBA, J2EE, dotNET et le *grid computing* sont les plus envisageables pour appliquer l'approche MDA aux systèmes futurs.

Le service Web est une plate-forme très importante aujourd'hui. C'est la plate-forme cible qui paraît être la plus étudiée et la plus convoitée pour construire les nouvelles applications Internet et pour permettre aux applications du passé (*legacy applications*) d'être adaptées à l'Internet.

Nous présentons dans une première partie les approches privilégiant les aspects statiques des services Web (structure) et, dans une seconde partie, nous discutons les approches privilégiant leurs aspects dynamiques (comportement).

2.4.1 Aspects statiques

D. Frankel et J. Parodi font partie des premiers à discuter l'approche MDA pour développer des applications ayant les services Web comme plate-forme cible [68]. Ils proposent en particulier :

1. Une architecture *four-tiers* pour les systèmes distribués dans laquelle les services Web ont un rôle fondamental pour les applications orientées Internet.
2. La création des modèles formels pour supporter l'approche MDA.
3. Une démarche pour la correspondance entre un modèle métier en UML et les services Web.

La figure 2.17 présente l'architecture *four-tiers* proposée par D. Frankel et J. Parodi.

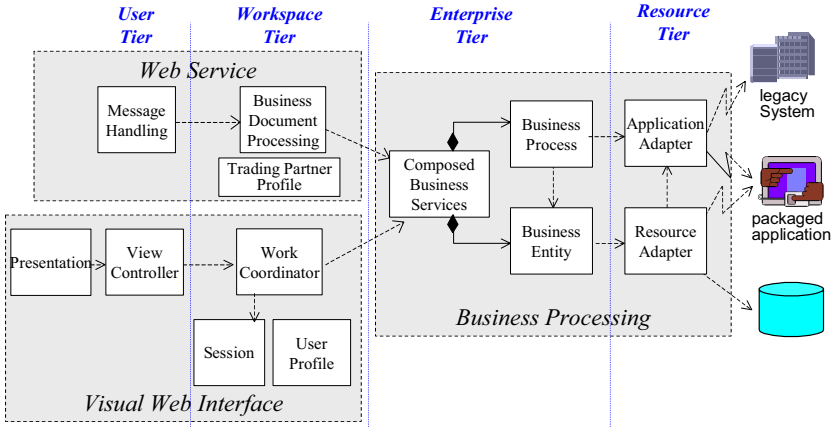
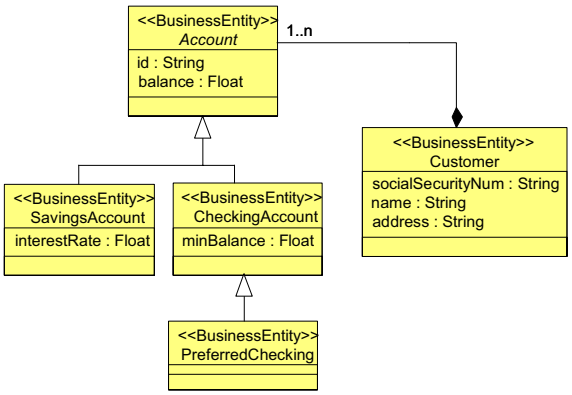
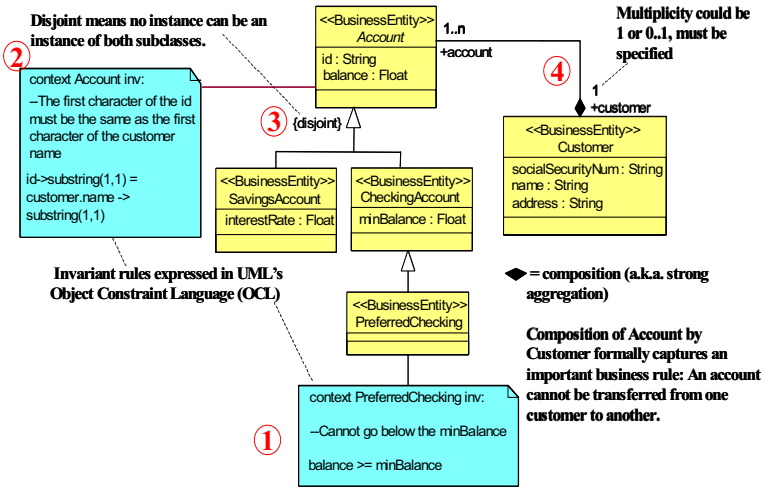


Figure 2.17 – L'architecture *four-tiers* [68]

Selon la figure 2.17, le `Resource Tier` est responsable de l'adaptation de la logique métier (`Business Process`) aux systèmes du passé (*legacy systems*) et de l'adaptation des données (`Resource Adapter`) aux bases de données. L'`Enterprise Tier` contient les services métier sous forme de services Web composés ayant des associations avec les processus métier (`Business Process`) et les données (`Business Entity`). Le `Workspace Tier` peut être organisé de deux manières : comme un traitement de documents (`Business Document Processing`) et comme un coordinateur de travail (`Work Coordinator`). Le `User Tier` peut être organisé de deux manières : comme un manipulateur de messages (`Message Handling`) et comme un *browser* (`Presentation + View Controller`). Par exemple, un navigateur Web qui fait le contrôle et la présentation de données (i.e. interaction avec l'utilisateur). Dans [119], une autre architecture *four-tiers* très similaire à celle de la figure 2.17 est aussi proposée.



(a)



(b)

Figure 2.18 – (a) un modèle métier imprécis et incomplet ; (b) un modèle métier plus précis et complet[68]

La figure 2.18 présente (a) un modèle UML imprécis et incomplet et (b) un autre modèle UML plus précis et complet. Cette figure vise à illustrer qu'une approche MDA peut réussir si, et seulement si, les modèles indépendants de la plate-forme sont assez précis et complets pour être transformés en un modèle dépendant de la plate-forme. Cela ne signifie pas que le modèle doit avoir tous les concepts de la plate-forme cible, mais il doit avoir une description abstraite de la solution du problème. Par conséquence, la description de la logique métier doit contenir la structure et la sémantique de la résolution du problème. Par exemple, la figure 2.18 (a) ne contient pas d'informations précises sur les cardinalités et contraintes. Ceci complique la réalisation de ce modèle dans un langage tel que Java et demande au programmeur d'être toujours en contact avec le concepteur. Par contre, la figure 2.18 (b) est plus précise : elle permet d'en connaître plus sur la logique métier de résolution du problème et demande moins d'interaction entre le programmeur et le concepteur. Par exemple, un modèle peut être précis par l'utilisation d'*Action Semantics* d'UML [177].

La figure 2.19 présente un fragment de correspondance entre un modèle UML et WSDL. Une classe stereotypée `BusinessService` en UML correspond au `PortType` en WSDL. Une méthode en UML correspond à une `Operation` en WSDL. Un paramètre d'entrée ou de sortie en UML correspond à un `Message` d'entrée ou de sortie en WSDL.

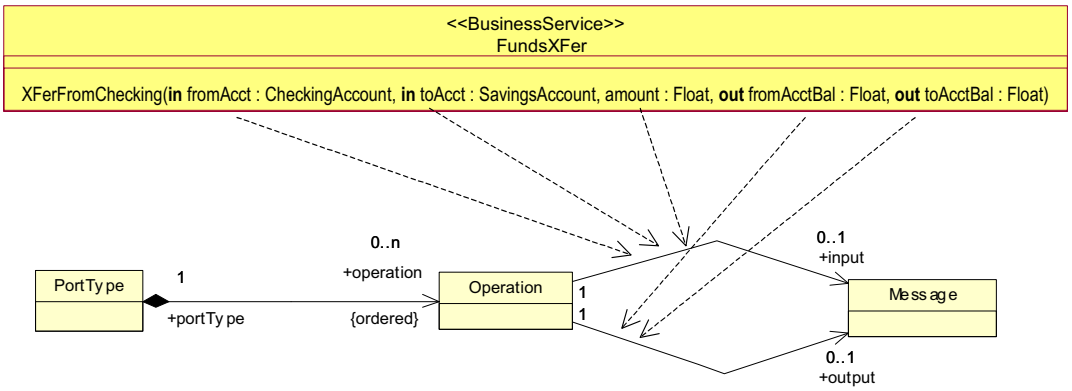


Figure 2.19 – Une correspondance entre un modèle métier en UML et WSDL [68]

D'autres travaux ont été lancés pour étudier la création de métamodèles pour des plates-formes cibles comme les services Web. Parmi ces travaux, B. Bordbar et A. Staikopoulos [25] étudient la création d'un métamodèle le plus précis possible pour les services Web, à partir de schémas ou d'autres spécifications existantes. Par exemple, ils proposent un métamodèle pour WSDL à partir de son schéma [206]. La figure 2.20 présente ce métamodèle généré à partir du schéma de WSDL. Ce métamodèle contient `Definitions` qui a une référence pour zéro ou plusieurs éléments `anyTopLevelOperationalElement` (avec le stéréotype `<XSDgroup>`). Ceci contient les éléments `Import`, `Message`, `PortType`, `Binding` et `Service`. L'élément `Message` contient des éléments `Part`. L'élément `PortType` contient des éléments `Operation`. L'élément `Binding` contient des éléments `BindingOperation`. L'élément `Service` contient des éléments `Port`. Dans ce métamodèle, les différents types d'opérations sont aussi modélisés tels que « *request-response* » et « *solicit-response* ».

De plus, B. Bordbar et A. Staikopoulos comparent leur proposition au métamodèle proposé dans nos travaux [34], afin de faire ressortir les similarités et les différences. Ils soulignent entre autres que la détermination du métamodèle le plus adapté à la plate-forme peut avoir des impacts à la fois dans la modélisation et la transformation. Dans ce cas, une approche basée sur la qualité de modèle (QoM -

Quality of Model), similairement aux travaux sur la qualité de service (QoS - *Quality of Service*), serait très bénéfique. Cependant, nous n'avons pas l'intention d'aborder cette problématique ici, étant donnée son ampleur.

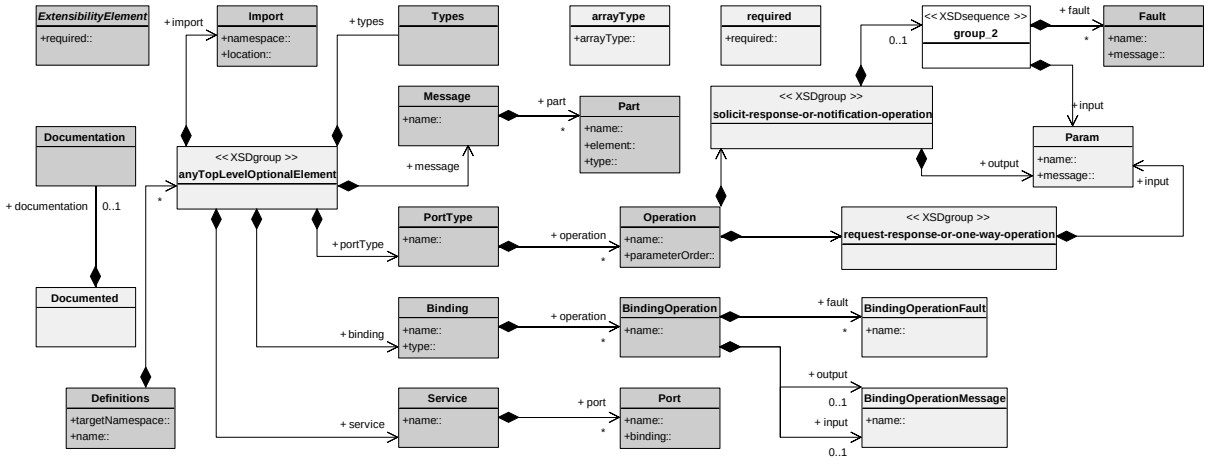


Figure 2.20 – Un métamodèle pour WSDL proposé en [25]

Parmi les travaux d'O. Patrascoiu, une validation du langage YATL est réalisée par l'application de MDA à la plate-forme de services Web [161]. Il part d'un modèle conforme à EDOC et le développe dans la plate-forme de services Web. Il fournit un métamodèle pour WSDL (figure 2.21), la transformation d'EDOC-CCA en WSDL, et utilise un cas d'étude pour démontrer la validité de YATL. Ce travail présente des similarités avec notre proposition pour l'application de MDA pour la plate-forme de services Web [34].

Le métamodèle de WSDL de la figure 2.20 et celui de la figure 2.21 présentent plusieurs différences. Par exemple, dans le premier métamodèle, l'élément `Message` contient des éléments `Part`, alors que dans le deuxième métamodèle, l'élément `Message` a une référence pour les éléments `Part`. Une différence plus importante peut être vérifié dans l'élément `Operation`. Dans le premier métamodèle, l'élément `Operation` contient des éléments pour chaque type d'opération, par exemple « *oneway* » et « *request-reponse* ». Dans le deuxième métamodèle, l'élément `Operation` contient des éléments `Input`, `Output` et `Fault`, il ne explicite pas le type d'opération.

Une correspondance entre WSDL et EDOC proposée dans [161] peut être résumée par le tableau 2.2.

EDOC	**WSDL**
DataType	XML Schema SimpleType
ComplexType	XML Schema ComplexType
Attribute	XML Schema Attribute
CCA FlowPort	Message
CCA OperationPort	Operation
CCA ProtocolPort	PortType
CCA ProcessComponent	Service

Table 2.2 – Une correspondance entre EDOC et WSDL (fragment) [161]

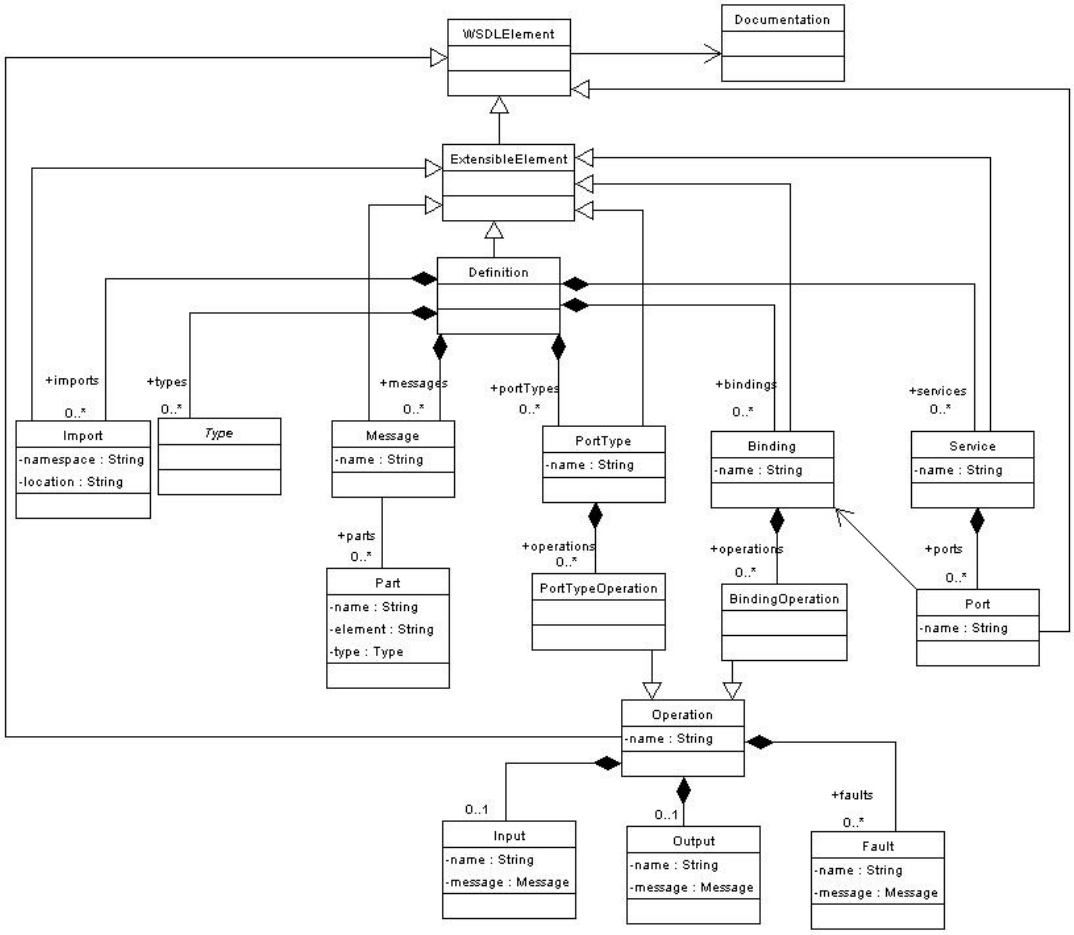


Figure 2.21 – Un métamodèle de WSDL proposé en [161]

Étant donné les correspondances (cf. table 2.2) entre le métamodèle de EDOC et WSDL, l'exemple 2.4 présente des fragments de la définition de transformation en YATL proposée par O. Patrascoiu pour transformer un modèle EDOC-CCA en WSDL.

Exemple 2.4 – Fragments de règles de transformation en YATL [161].

```
start kmf::edoc2ws::main;
        namespace kmf(sd, ocl) {
                transformation edoc2ws {
                -- EDOC.ECA.DocumentModel to WS.XSD

                -- Map an EDOC DataType to an XSD SimpleType
                rule dt2st match edoc::ECA::DocumentModel::DataType () {
                        -- Create SimpleType and store mapping
                        let st: ws::xsd::SimpleType;
                        st := new ws::xsd::SimpleType;
                        st.name := self.name;
                        track(self, type2type, st);
                }
                ***
                --- Map CCA to WSDL
```

```
16                  rule cca2wsdl() {
                            -- Create a WSDL Message for each EDOC FlowPort
18                          apply flowPort2message();
                            -- Map Operation Ports
20                          apply operationPort2operation();
                            -- Map Protocol Ports
22                          apply protocolPort2portType();
                            -- Map ProcessComponent
24                          apply processComponent2service();
                            -- Link Definition to types, messages, and portTypes
26                          apply linkDefinition2X();
                    }
28                  ***
                    -- main rule
30                  rule main () {
                    -- Map DocumentModel to XSD
32                  apply documentModel2xsd();
                    -- ECA to WSLD
34                  apply cca2wsdl();
                    }
36          }
    }
```

Pour valider son langage de transformation, O. Patrascoiu s'est basé sur l'étude de cas de l'agence de voyages. La figure 2.22 présente l'agence de voyages modélisée en EDOC. De plus, il relate l'utilisation des règles de transformation en YATL pour obtenir le WSDL correspondant au modèle de l'agence de voyages en EDOC.

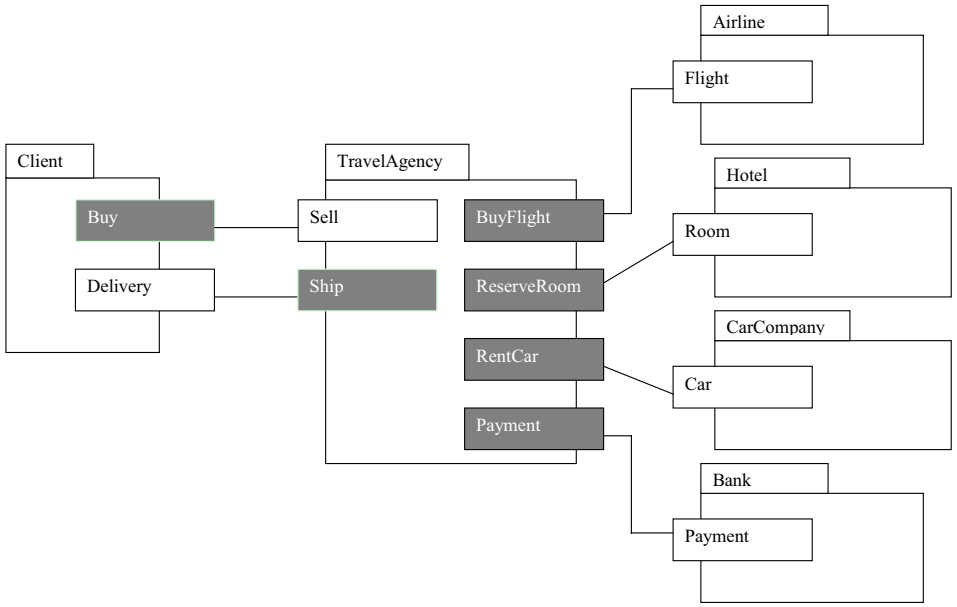


Figure 2.22 – L'étude de cas de l'agence de voyages en EDOC proposé dans [161]

2.4.2 Aspects dynamiques

D'autres études sont allées plus loin et ont également considéré les aspects dynamiques pour créer des modèles indépendants de la plate-forme et les développer en services Web.

R. Gronmo et al. proposent une méthodologie pour développer les services Web et une correspondance entre un modèle UML et WSDL [78]. La figure 2.23 illustre la méthodologie pour le développement de services Web dirigé par les modèles.

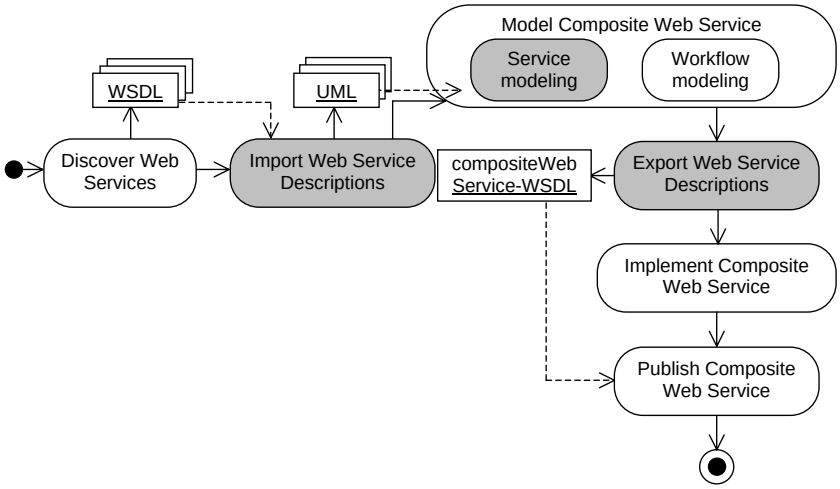


Figure 2.23 – Les étapes de développement de services Web dirigé par les modèles : méthodologie [78]

Cette méthodologie peut être organisée en suivant ces étapes :

- `Discover Web Services` : Le développeur utilise un navigateur Web et un client d'annuaire pour chercher un service Web candidat pour la composition. Il obtient une liste de services Web et les localisations de leur description (représentée en WSDL).
- `Import Web Service Descriptions` : Le développeur importe la description du service et la traduit en UML. Le résultat est un modèle UML conforme au WSDL.
- `Model Composite Web Service` : Le développeur utilise un outil UML pour réviser et intégrer les modèles importés sous forme de modèle d'un service composé. Cette étape est composée de la modélisation de la structure du service (`Service modeling`) et de la modélisation du comportement du service composé (`Workflow modeling`).
- `Export Web Services Description` : Le modèle du service Web composé est exporté afin d'obtenir un document WSDL du service Web composé.
- `Implement Composite Web Service` : Le service Web est réalisé à partir d'un document WSDL.
- `Publish Composite Web Service` : Finalement, le service Web composé est publié dans un annuaire.

La figure 2.24 présente un modèle UML et son équivalent en WSDL. Selon cette figure, une `Class` en UML est l'équivalent de `Types` en WSDL. Une `Interface` en UML est l'équivalent de `PortType` et `Binding` en WSDL. Une méthode en UML est l'équivalent d'`Operation` en WSDL. Une classe stéréotypée par `BusinessServices` est l'équivalent de `Service` en WSDL. Ensuite, R. Gronmo et al.

utilisent UMT (*UML Transformation Tool*) pour créer les règles de transformation de UML en WSDL [178].

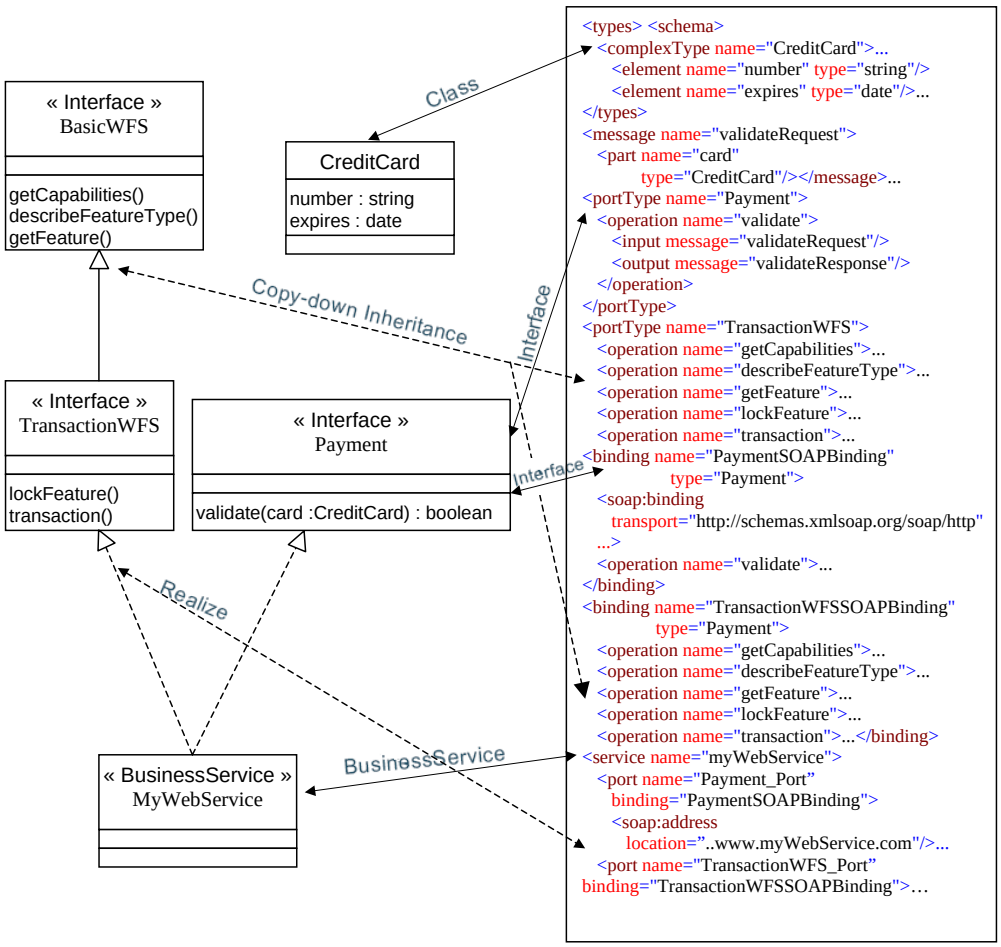


Figure 2.24 – Une conversion entre un modèle UML et un document WSDL [78]

D. Skogan, R. Gronmo et I. Solheim proposent les diagrammes d'activités d'UML pour réaliser la composition de services. Le service composé est ensuite exporté comme un document WorkSCo ou BPEL4WS [180]. La figure 2.25 présente la conversion d'UML en BPEL4WS. Selon cette figure, une activité stéréotypée `WebServiceCall` est associée aux éléments `Invoke` et à `PartnerLink` en BPEL4WS. L'objet d'entrée `plumeInfo` est transformé en `Variable` en BPEL4WS. L'objet de sortie `plumeLayer` est aussi transformé en `Variable`. Plusieurs valeurs marquées dans les activités (`service`, `portType` et `operation`) sont des références de définitions du document WSDL, et correspondent aux attributs de `Invoke` en BPEL4WS. Le flux de séquence des activités correspond à `Sequence` en BPEL4WS. Les transformations de données entre `plumeLayer` et `plume` sont faites en BPEL4WS par l'instruction `Assign`.

B. Bordbar et A. Staikopoulos ont aussi étudié les aspects dynamiques des diagrammes d'activités d'UML et des services Web, en particulier la correspondance entre les diagrammes d'activités d'UML et WSCI, et entre les diagrammes d'activités d'UML et BPEL4WS [26] [27].

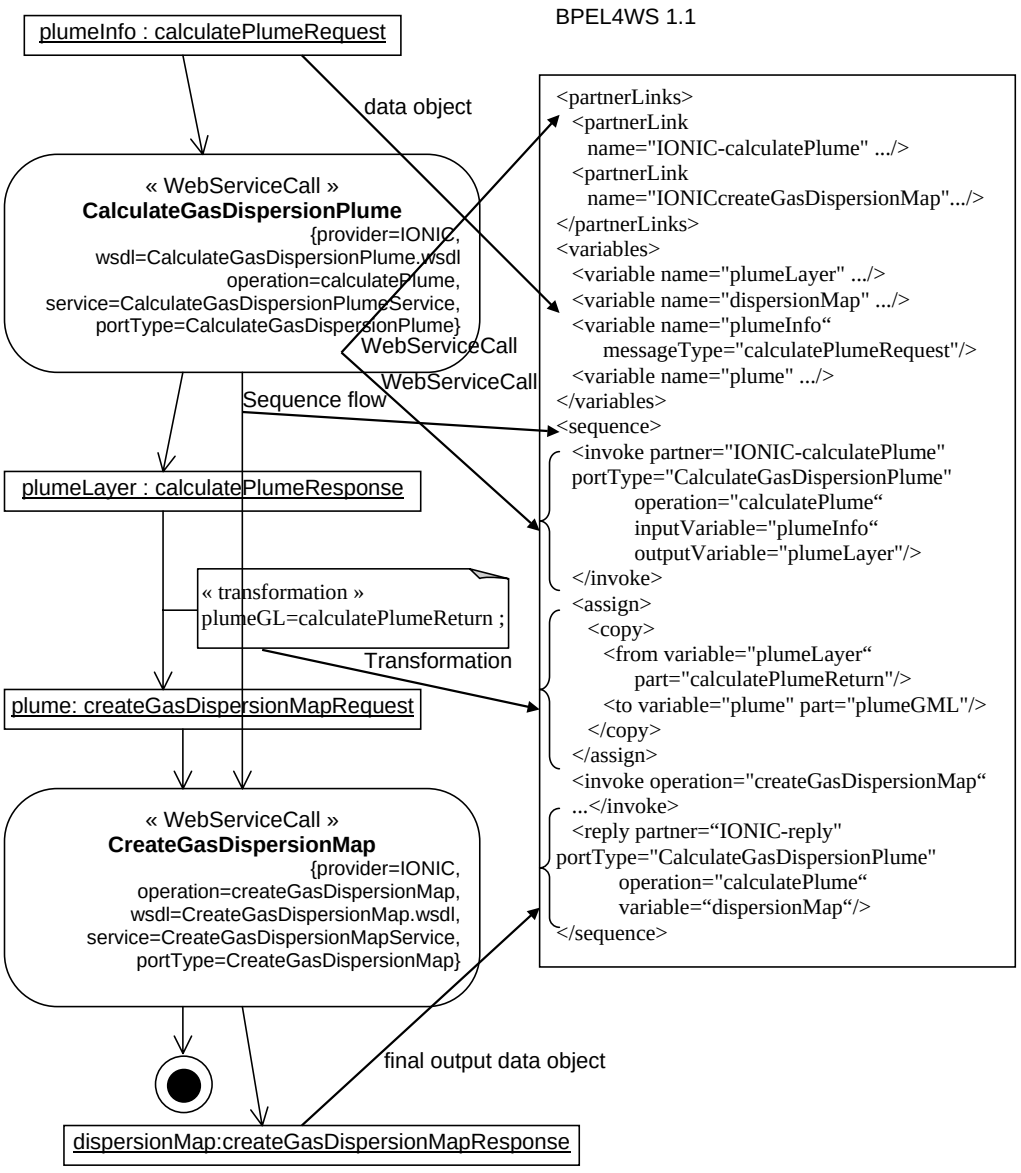


Figure 2.25 – Une conversion d'UML en BPEL4WS proposée en [180]

Dans [27], B. Bordbar et A. Staikopoulos présentent une autre étude de cas et sa réalisation avec les services Web. Ils proposent un métamodèle pour BPEL4WS (cf. figure 2.26), spécifient la correspondance entre le diagramme d'activité d'UML et BPEL4WS (cf. table 2.3) et définissent des règles de transformation en OCL (cf. exemple 2.5).

Dans le métamodèle de la figure 2.26, l'élément `Process` contient d'autres éléments tels que `Activity`, `Variable`, `Partner`, `PartnerLink` et `Role`. Parmi les différents types d'actions, nous pouvons mentionner des séquences (`Sequence`) , des boucles (`While`) et des appels (`Invoque`).

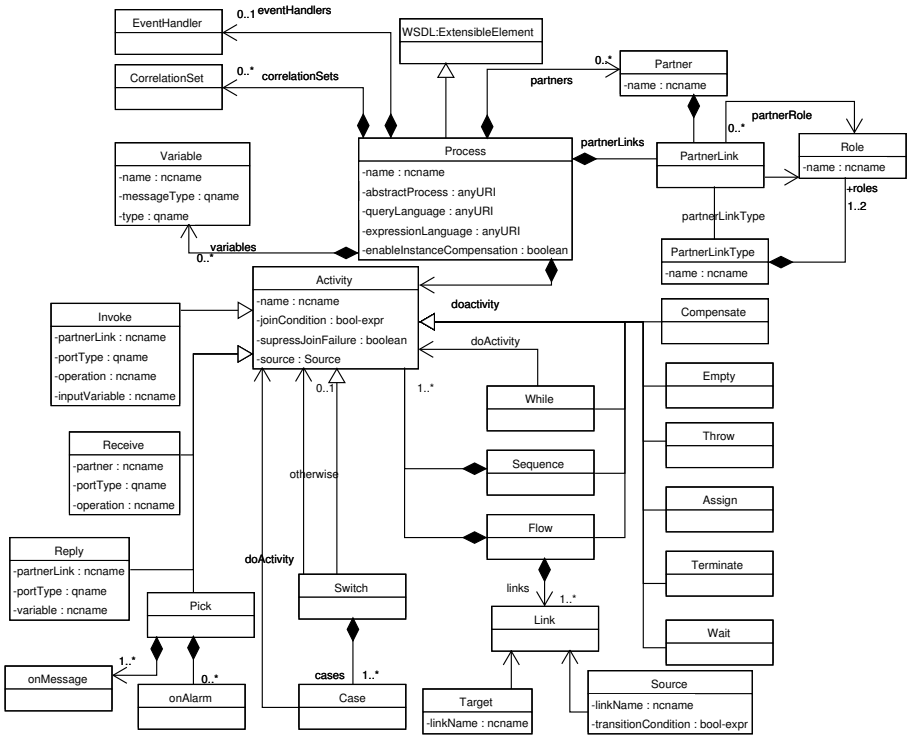


Figure 2.26 – Un fragment du métamodèle de BPEL4WS proposé en [27]

Le tableau 2.3 illustre une correspondance entre UML et BPEL4WS.

UML 2.0	**BPEL4WS 1.1**
Class comme BehavioredClassifier, Activity, StructuredActivityNode	<process>
Datastore, StructureNode Variable, ObjectNode, Class Attributes	<variable>
ControlFlow	<sequence>
AcceptCallAction	<receive>
ReadVariableAction, WriteVariableAction	<assign>
Variable, ObjectNode, OutputPin	<from>
Variable, ObjectNode, InputPin	<to>
CallOperationAction	<invoke>
ReplyAction	<reply>

Table 2.3 – Une correspondance entre UML et BPEL4WS [27]

L'exemple 2.5 contient des fragments de règles de transformation en OCL : UML vers BPEL4WS.

Exemple 2.5 – Fragments de règles de transformation en OCL : UML vers BPEL4WS [27]

```
params srcActivity: UML::Activity -- the UML source Model
source srcActNodes: OCL::Set(ActivityNode)
target trgProcess: BPEL4WS::Process
                        trgVariable: BPEL4WS::Variable
                        trgPartnerLink: BPEL4WS::PartnerLink
                        trgActivity: BPEL4WS::Activity
```

```
source srcActNodes = srcActivity.nodes -> asSet()-> union(
cond     srcActivity.group -> collectNested(ActivityNode))

--mapping pocess
srcActivity.name <- trgProcess.name
try UMLDatastore2BPVariable on srcActNodes -> collect(DataStore)
<~> trgVariable.type
try ActivityPartition2BPPartnerLink on srcActNodes ->
collect(ActivityPartition) <~> trgPartnerLink.type
try UMLProcessActivity2BPActivity on srcActNodes ->
collect(Action) <~> trgActivity.type


-- UMLProcessActivity2BPActivity (UML,BPEL4WS)
params srcActions: OCL::Set(UML::Action)
source srcActivity: UML::Activity
target trgSequence: BPEL4WS::Sequence
                    trgFlow: BPEL4WS::Flow
--mapping process
if srcActivity.oclIsTypeof(Sequence) then
try UMLSequence2BPSequence on srcAction <~> trgSequence.type
try UMLFlow2BPFlow on srcActions <~> trgFlow.type
endif


-- UMLSequence2BPSequence (UML, BPEL4WS)
***
```

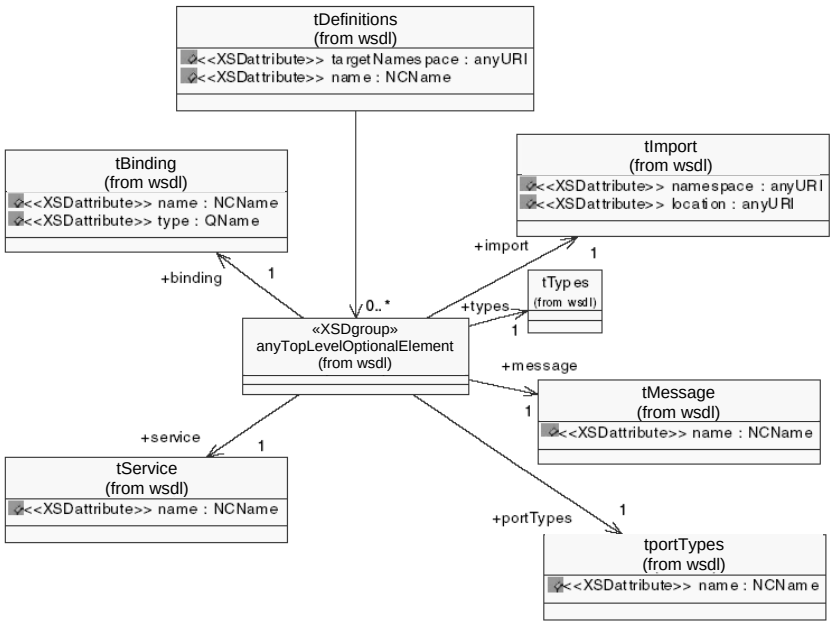


Figure 2.27 – Un métamodèle pour WSDL [99]

Kath et al. proposent une infrastructure pour mettre en œuvre l'approche MDA et ils présentent des expérimentations avec EDOC et les services Web [99]. L'infrastructure proposée est constituée de dif-

férents outils qui sont assemblés pour supporter l'approche MDA. Dans leur approche, ils présentent un métamodèle pour les services Web (WSDL), une correspondance entre EDOC et WSDL, un métamodèle pour BPEL4WS et une correspondance entre EDOC et BPEL4WS. La figure 2.27 présente le métamodèle de WSDL. La figure 2.28 présente la correspondance entre EDOC et WSDL. Selon la figure 2.28, `ProcessComponent` en EDOC-CCA est l'équivalent de `Service` en WSDL. Le `Protocol-Port` en EDOC-CCA est l'équivalent de `Port` en WSDL. Le `Protocol/Interface` en EDOC-CCA est l'équivalent de `Binding`, `BindingOperation` et `PortType`. L'`OperationPort` est l'équivalent d'`Operation`.

Dans [99], les auteurs spécifient des correspondances entre EDOC et BPEL4WS comme suit :

- Une Interface fournie par un `ProcessComponent` d'EDOC correspond à `PaternLink`.
- Les paramètres `in` et `out` (`FlowPorts`) des `Operation-Ports` d'une `Interface` correspondent aux variables en BPEL.
- Les ports de communication d'EDOC correspondent aux `activities` en BPEL.
- Pour chaque élément d'une chorographie EDOC un élément `invoke` (`reply` ou `receive`) est généré en BPEL.

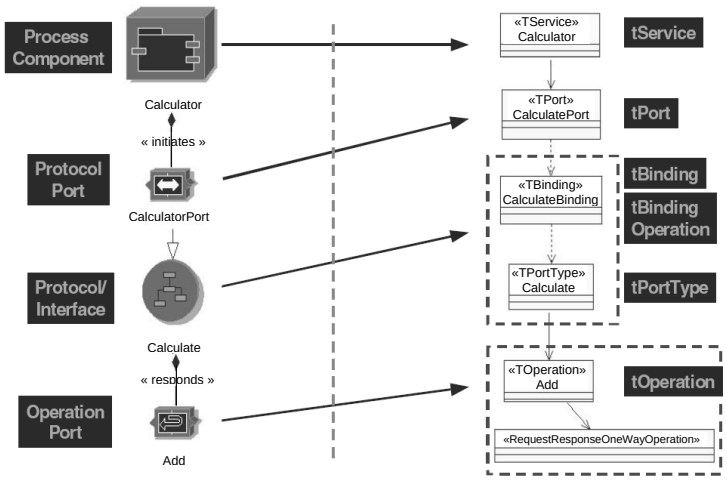


Figure 2.28 – Une correspondance entre EDOC et WSDL [99]

K. Baina et al. proposent aussi une démarche pour le développement de services Web dirigée par les modèles [15], mais en dehors de l'approche MDA. Ils fournissent un *framework* qui génère automatiquement des modèles de composition BPEL4WS à partir des spécifications de protocoles basées sur la description de service et sur le modèle de composition (une variation de *statecharts*) de Self-Serv [19]. Ainsi, ce modèle de composition met l'accent sur les états et les transitions. La figure 2.29 présente un exemple d'utilisation de ce modèle de composition basé sur la conversation.

La figure 2.30 illustre la méthodologie proposée par K. Baina et al. Selon cette figure, leur méthodologie s'articule autour de trois phases de transformation : d'abord, chaque transition est transformée dans un squelette de processus ; ensuite, chaque état est transformé dans un squelette d'état en ajoutant la transition sous-jacente à l'état ; finalement, une fois que les squelettes individuels des transitions et des états ont été réalisés, les squelettes des états sont liés ensemble dans un squelette de processus général selon la topologie du graphe de *statechart*.

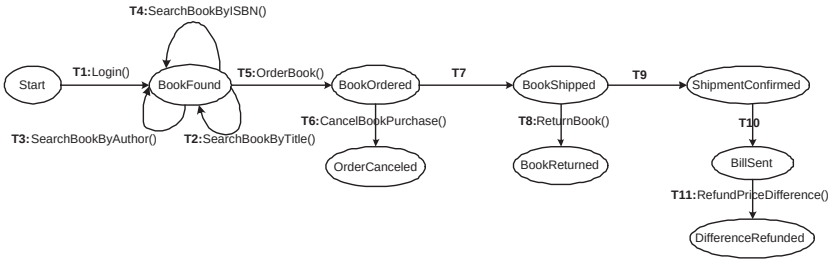


Figure 2.29 – Un exemple du modèle du protocole de conversation de service : *EBookShop* [15]

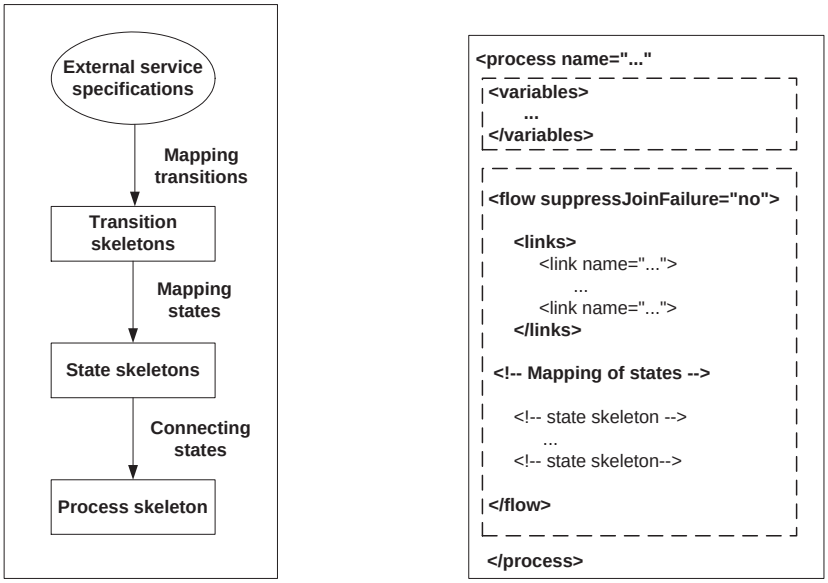


Figure 2.30 – Un processus de génération de *skeletons* [15]

Le tableau 2.4 présente les règles de génération du modèle de composition en BPEL4WS à partir de la spécification du protocole. Comme le montre ce tableau, les auteurs distinguent les opération en quatre catégorie : *internal*, *singleton*, *serial* et *parallel*. Pour chacune de ces opérations, ils définissent les éléments de BPEL4WS correspondant. Ainsi, si une opération d'une interface *op* associée à une transition t est une opération du type *singleton*, alors le squelette de BPEL4WS correspondant inclura une activité `<invoke ../>` pour appeler l'opération op_j du service S_i. Si la transition t est compensable (*compensable transition*), l'opération de compensation de l'opération *op*, dite op_c, peut être inférée automatiquement, mais seulement si $S_i.op_j$ est compensable en conformité avec le protocole de conversation du service S_i. Dans ce cas, le `<compensationHandler>` du squelette de BPEL4WS de t incluira une activité `<invoke ../>` pour appeler l'opération de compensation op_j. Dans le cas d'une opération *serial*, l'équivalent en BPEL4WS inclura une activité `<sequence>` (`<sequence> ...</sequence>`). Dans le cas d'une opération parallèle, l'équivalent en BPEL4WS inclura une activité `flow` (`<flow>...</flow>`).

Entité	**Méthode**	**Élément de BPEL**
Internal	invoke	<!- - empty implementation logic- ->
	compensation	<!- - empty compensation logic - ->
Singleton $S_i.op_j$	invoke	<invoke partner="Si" operation="opj"../>
	compensation	op_j is related to a `Definite` or `Effect-less` transition in S_i protocol <!- - empty compensation logic - ->
		op_j is related to a $Compensable$ transition in S_i protocol <invoke partner="Si" operation="opjc" ../>
Serial $OP_j; OP_k$	invoke	<sequence> <!- - generating operation implementation logic of OPj - -> <!- - generating operation implementation logic of OPk - -> </sequence>
	compensation	<sequence> <!- - generating OPk compensation logic - -> <!- - generating OPj compensation logic - -> </sequence>
Parallel $OP_j // OP_k$	invoke	<flow> <!- - generating operation implementation logic of OPj - -> <!- - generating operation implementation logic of OPk - -> </flow>
	compensation	<flow> <!- - generating OPj compensation logic - -> <!- - generating OPk compensation logic - -> </flow>

Table 2.4 – Les règles pour générer un modèle de composition en BPEL présentées par [15]

2.5 Synthèse

Plusieurs intergiciels existent pour supporter le développement d'applications orientées Internet. En particulier, les Services Web sont devenus l'intergiciel préféré pour le développement ou l'intégration de systèmes sur l'Internet. Cependant, un intergiciel n'est pas une solution complète pour le développement et la maintenance de nouveaux systèmes informatiques qui sont devenus plus complexes qu'auparavant. Dans ce contexte, MDA est une solution adaptée pour gérer la complexité de ces systèmes. Dans ces conditions, l'application de MDA à la plate-forme de services Web pourrait devenir la solution indispensable dans l'avenir pour le développement des applications orientées Internet.

Dans l'approche MDA, nous avons pointé l'importance des techniques de transformations et leurs places primordiales dans le processus de développement selon cette approche. Cependant, nous avons relevé que la distinction entre la spécification de correspondances et la définition de transformations n'était pas bien établie dans la plupart des travaux actuels. Cette distinction nous paraît importante, et d'un point de vue conceptuel est en conformité avec la séparation des aspects (différents niveaux d'abstraction). Les problèmes telles que la spécification de correspondances et la définition de transformations sont déjà connus dans d'autres domaines comme celui des bases de données.

Plusieurs langages de transformation dans le contexte de MDA sont disponibles. Cependant, une théorie bien établie pour MDA est en cours d'évolution. Parmi les plates-formes cibles, celle des services

Web est une des plus sollicitées pour expérimenter une approche MDA.

Dans cet état de l'art, nous nous sommes concentrés d'une part sur les techniques de correspondance et de transformation dans le cadre de MDA, et d'autre part sur l'application de l'approche MDA à la plate-forme Service Web. Dans le chapitre suivant, nous présentons notre approche MDA pour la plate-forme Service Web, en nous focalisant sur la partie modélisation. Nous introduirons brièvement notre démarche de distinction entre la spécification de correspondances et la définition de transformations. Le chapitre 4 présentera en détail cette démarche.

PARTIE II

Les Services Web dans le contexte de MDA

CHAPITRE 3

Approche MDA pour la plate-forme Services Web : modélisation

3.1 Introduction

Ce chapitre présente notre approche MDA pour la plate-forme Services Web et concerne la partie dédiée à la modélisation. Notre approche est illustrée par l'étude de cas d'une agence de voyages. Nous avons choisi cette étude de cas car elle est un exemple typique d'application orientée Internet qui est à la fois B2C (*Business-to-Consumer*) et B2B (*Business-to-Business*). Elle illustre donc bien la problématique des applications métiers qui mettent en œuvre des interactions aussi bien avec les clients qu'avec d'autres applications de partenaires.

Nous avons choisi les langages de modélisation UML (*Unified Modeling Language*) et EDOC (*Enterprise Distributed Object Computing*) pour modéliser l'agence de voyages. Ainsi, nous présentons un modèle indépendant de la plate-forme (PIM) créé avec UML et un autre avec EDOC.

La plate-forme des Services Web a été choisie pour créer un premier modèle dépendant de la plate-forme (PSM). De plus, nous avons retenu J2EE (plus précisément, Java et JWSDP - *Java Web Services Developer Pack*) pour créer un deuxième modèle dépendant de la plate-forme. Le premier PSM peut être considéré comme une plate-forme abstraite [6], ou autrement dit utilisant un langage d'implémentation indépendant de l'environnent (*implementation language environment independent*) [138]. Le deuxième PSM peut être considéré comme une plate-forme concrète [6] ou autrement dit utilisant un langage d'implémentation dépendant d'un environnent spécifique (*implementation language environment specific*) [138]. Nous avons choisi dotNET (plus précisément, C# et API *framework*) comme plate-forme alternative pour créer un troisième modèle dépendant de la plate-forme.

Un modèle de composition de services peut être développé en UML avec des diagrammes d'activités ou en EDOC avec le modèle de comportement (nommé *choreographies*). Pour la mise en œuvre d'un modèle de composition, nous avons opté pour BPEL4WS comme plate-forme cible du fait de sa adoption par le milieu académique et industriel.

Nous avons choisi d'utiliser plusieurs langages de modélisation pour créer les PIMs et plusieurs plates-formes cibles pour créer les PSMs afin d'identifier les problèmes liés à l'approche dirigée par les modèles et d'en proposer ensuite des solutions viables. Ainsi, nous procédons par l'expérimentation de l'application de MDA pour la plate-forme des Services Web pour en tirer des propositions qui représentent une contribution de cette thèse [35] [36] [37] [81] [112].

Dans ce chapitre, nous introduisons un exemple illustratif d'une agence de voyages à partir d'un cahier des charges minimum pour développer le système informatique de cette agence de voyages. Nous

présentons ensuite notre méthodologie utilisant MDA, la modélisation de l'agence de voyages, les plates-formes cibles utilisées et leur métamodèles. Dans les deux chapitres suivants, nous poursuivons notre démarche de spécification de correspondances entre métamodèles et de définition de transformations de modèles.

3.2 Étude de cas : l'agence de voyages

Une agence de voyages fournit typiquement les services pour : consultation, réservation, paiement et annulation de chambres d'hôtel, de billets d'avion, et de locations de voiture.

Afin de fournir ces services à ses clients, l'agence de voyages doit établir des liens avec d'autres entreprises : compagnies aériennes, compagnies de location de voitures et réseaux hôteliers. Une institution financière (une banque) est également nécessaire pour faciliter les transactions financières entre les clients et l'agence de voyages, ou entre l'agence de voyages et les autres partenaires. L'agence de voyages a besoin d'un système informatique capable de réutiliser les services fournis par ses partenaires et de les vendre à ses clients.

Le principe général du système peut se résumer ainsi :

- Les services de l'agence de voyages sont fournis aux clients finaux au travers d'un navigateur Web qui se charge de l'affichage de données et de l'interaction.
- La compagnie aérienne doit fournir des services permettant à une agence de voyages de consulter les vols, de réserver un billet, de le payer, et de l'annuler.
- La compagnie de location de voitures doit fournir des services permettant à une agence de voyages de consulter les voitures disponibles, d'en réserver une, de payer sa location, et d'annuler la réservation.
- De même, le réseau hôtelier doit fournir des services permettant à une agence de voyages de consulter les chambres disponibles d'un hôtel, d'en réserver une, de la payer, et d'annuler la réservation.
- La banque doit fournir un service permettant à une entreprise d'effectuer une transaction financière à partir de données de payement (identités du débiteur et du créancier).

L'agence de voyages peut émettre les confirmations de réservation, les factures de paiement de ses services, et les billets d'avion. Pour simplifier, nous considérons que pour toutes les confirmations de réservation, les factures et les billets d'avion sont affichés sur un navigateur Web et envoyés par email, puis imprimés par l'utilisateur. Ceci nous permet de nous affranchir d'une quatrième entreprise qui serait responsable de la livraison de la confirmation des réservations, des factures et des billets d'avion.

3.2.1 Étapes de développement du système informatique : l'agence de voyages

Les étapes de développement du système informatique de l'agence de voyages sont présentées dans le tableau 3.1. Pour réaliser ces différentes étapes, nous avons adopté une approche MDA. Par conséquent, la méthodologie proposée doit être basée sur la création de modèles métiers (PIM) qui seront transformés en modèles dépendants de plates-formes technologiques (PSM). La méthodologie sera plus détaillée dans la section 3.3. Étant donné que l'étude de cas sera utilisée pour expérimenter la démarche MDA, nous utilisons plusieurs langages de modélisation et plates-formes cibles représentative de Service Web. La modélisation de l'agence de voyages et des autres entreprises sera d'abord faite en UML (PIM), puis en EDOC (PIM).

Tâche	Description
Choisir une méthodologie	Le processus le plus adapté au type de l'application doit être formalisé (de manière abstraite).
Choisir une architecture en couches	L'architecture de l'application peut être 1-*tier*, 2-*tiers*, 3-*tiers* ou N-*tiers*.
Modéliser la logique métier des compagnies d'aviation, de location de voitures et hôtelières	La modélisation de chaque logique métier doit être précise et complète pour guider le développement.
Modéliser la logique métier de l'agence de voyages	La logique métier de l'agence de voyages doit réutiliser les services fournis par ses partenaires.
Choisir les plates-formes pour l'implémentation	Les plates-formes doivent être facilement adaptables à l'Internet et largement répandue sur le marché informatique.
Développer les systèmes informatiques	Le développement doit être en conformité avec les modèles.
Livraison	Le système de l'agence de voyages doit être livré avec sa documentation dans les délais négociés avec le client.

Table 3.1 – Les étapes de développement du système informatique de l'agence de voyages.

Nos plates-formes cibles sont les services Web (PSM : plate-forme abstraite), J2EE (PSM : plate-forme concrète), dotNET (PSM : plate-forme concrète) et BPEL4WS. Ce choix est justifié par le fait que, d'une part, les Services Web sont actuellement la plate-forme la plus adaptée pour développer les applications orientées Internet et, d'autre part, J2EE et dotNET sont les plates-formes les plus utilisées pour développer les systèmes logiciels basés sur les nouvelles technologies (comme les services Web).

L'expérimentation que nous proposons pour analyser la démarche MDA peut être organisée comme l'indique le tableau 3.2.

PIM\PSM	Service Web	J2EE	dotNET
UML	DC°\WSDL et DA•\BPEL4WS	DC\(Java + JWSDP)	DC\(C# + API framework)
EDOC	CCA⋆\WSDL	CCA\(Java + JWSDP)	-

°DC (Diagramme de Classe) de UML

•DA (Diagramme d'Activité) de UML

⋆CCA (*Component Collaboratiion Architecture*) de EDOC

Table 3.2 – L'expérimentation proposée pour analyser la démarche MDA

Les spécificités d'une agence de voyages tant d'un point de vue fonctionnel qu'architectural, nous conduisent à privilégier une architecture 3-tiers client léger (3-*tiers thin client*) pour l'agence de voyages et 3-*tiers* pour les autres entreprises. Nous simplifions la présentation du développement de l'application en ne considérant que le *middle tier*. Les couches de présentation et de gestion de ressources (base de données) ne seront pas abordées ici. Dans [89] et [129], les auteurs présentent une approche dirigée par les modèles pour créer des bases de données et des pages Web.

3.3 Méthodologie

Les systèmes informatiques de l'agence de voyages et de ses partenaires étant initialement inexistants, nous pouvons appliquer directement une démarche *top-down* sans être gênés par les contraintes d'une adaptation à un système du passé (*legacy systems*).

La démarche MDA que nous proposons peut être présentée par le diagramme d'états-transition de la figure 3.1.

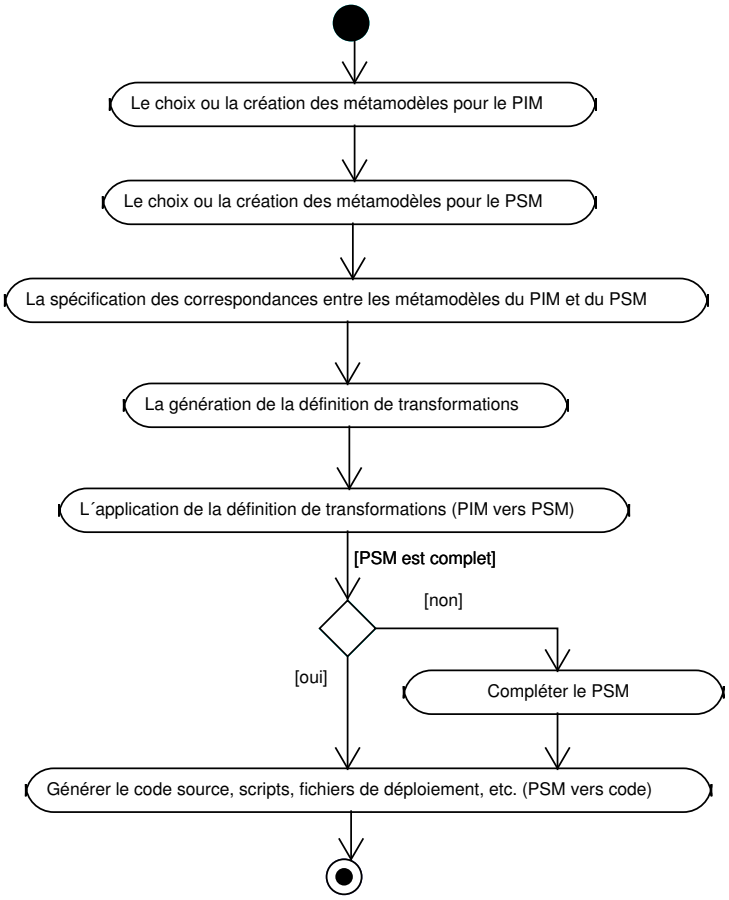


Figure 3.1 – Un diagramme d'activités minimal pour l'application de la démarche MDA

Le processus de la figure 3.1 définit les étapes suivantes :

1. Le choix ou la création des métamodèles pour construire le PIM.
2. Le choix ou la création des métamodèles pour construire le PSM.
3. La spécification des correspondances entre les métamodèles de PIM vers les métamodèles de PSM.
4. La génération de la définition de transformations à partir de la spécification de correspondances.
5. L'application de la définition de transformations pour transformer un PIM en un PSM.
6. La vérification du PSM : passage à l'étape 7 s'il est incomplet, à l'étape 8 s'il est complet.
7. L'intervention de l'utilisateur pour compléter le PSM.

8. La génération du code source, des scripts, des fichiers de déploiement.

L'étape 1 peut être réalisée par le choix d'UML ou d'EDOC comme PIM. D'autres approches peuvent opter pour la création de métamodèles spécifiques pour modéliser un métier spécifique, tel que DSL (*Domain Specific Language*) [51].

L'étape 2 correspond au choix d'un métamodèle pré-existant dans la littérature ou la création d'un nouveau métamodèle spécifique aux plates-formes cibles tels que les Services Web, J2EE et dotNET.

L'étape 3 est dédiée à la correspondance. Elle consiste à spécifier quels éléments du PIM (par exemple d'UML) sont équivalents ou similaires aux éléments du PSM (par exemple quels éléments d'UML sont équivalents ou similaires aux éléments de WSDL).

L'étape 4 utilise la correspondance définie dans l'étape 3 pour générer la définition de la transformation dans un langage de transformation. En fait, la spécification de correspondances peut être traitée comme un langage abstrait. Les langages abstraits peuvent spécifier les relations entre les métamodèles, mais ils ne sont pas exécutables. Par exemple, les langages abstraits proposés dans [26] et [102] sont basés sur OCL. Les langages concrets, comme ATL et YATL, permettent de définir l'exécution de transformations.

L'étape 5 est réalisée par l'application de définitions de transformations pour transformer un modèle d'entrée (PIM) en un modèle de sortie (PSM).

L'étape 6 vérifie si le modèle généré lors de l'étape précédente est complet.

L'étape 7 nécessite l'intervention du développeur pour compléter le PSM.

L'étape 8 consiste à générer le code source, les scripts et le fichier de déploiement à partir du PSM final créé lors de l'étape 5 ou 7. Dans ce cas, une transformation de type modèle vers code est nécessaire.

Les étapes 1 et 2 visent à permettre le choix ou la création de métamodèles pour construire un PIM ou un PSM. Il y a deux possibilités pour créer de nouveaux métamodèles :

- L'utilisation du concept de profils UML pour étendre le métamodèle d'UML à la prise en charge d'autres aspects non prévus antérieurement.
- La création de nouveaux métamodèles basés directement sur un langage de métamodélisation comme MOF [141] ou Ecore [30].

L'OMG incite à créer des profils et en fournit plusieurs qui peuvent être utilisés dans le contexte du MDA. Cependant, nous considérons que l'approche MDA sera plus à même de tenir ses promesses en utilisant la création et la manipulation de petits métamodèles spécifiques au domaine ou à la plate-forme. Ainsi, nous sommes plus en conformité avec la proposition de DSL [51]. En réalité, un profil UML peut être vu comme une extension de UML pour supporter un domaine spécifique. Cependant, les profils UML ne sont pas suffisants pour assurer la précision ou la validité d'un modèle.

Généralement, les développeurs créent un métamodèle et proposent ensuite un profil UML conforme à ce métamodèle, puisque la majorité des outils ne supportent que la création et l'édition de modèles UML. Nous pensons que MDA pourrait avoir un impact plus important des lors que les outils seront capables de créer et d'éditer des métamodèles et des modèles conformes à ces métamodèles. Au moment de la rédaction de cette thèse, un nombre limité d'outils fournit ce support. Nous citons EMF (*Eclipse Modeling Framework*) [30] qui supporte d'une part la création et l'édition de métamodèles conformes à Ecore, et d'autre part la création et l'édition de modèles conformes à ces métamodèles.

De plus, les outils existants ne sont pas encore capables de permettre l'édition, la visualisation et la manipulation directe de métamodèles et modèles d'une manière uniforme, intégrée et intuitive. Pour développer l'approche MDA, les développeurs utilisent souvent un ensemble d'outils entre lesquels métamodèles et modèles peuvent être échangés via XMI [99].

Selon notre proposition de processus minimum d'application de la démarche MDA, une architecture type pour la transformation de modèles peut être illustrée par la figure 3.2 [113]. Dans cette architecture, le modèle de correspondance (la spécification de correspondances - `mapping M`) est séparé du modèle de transformation (la définition de transformations - `transformation M`).

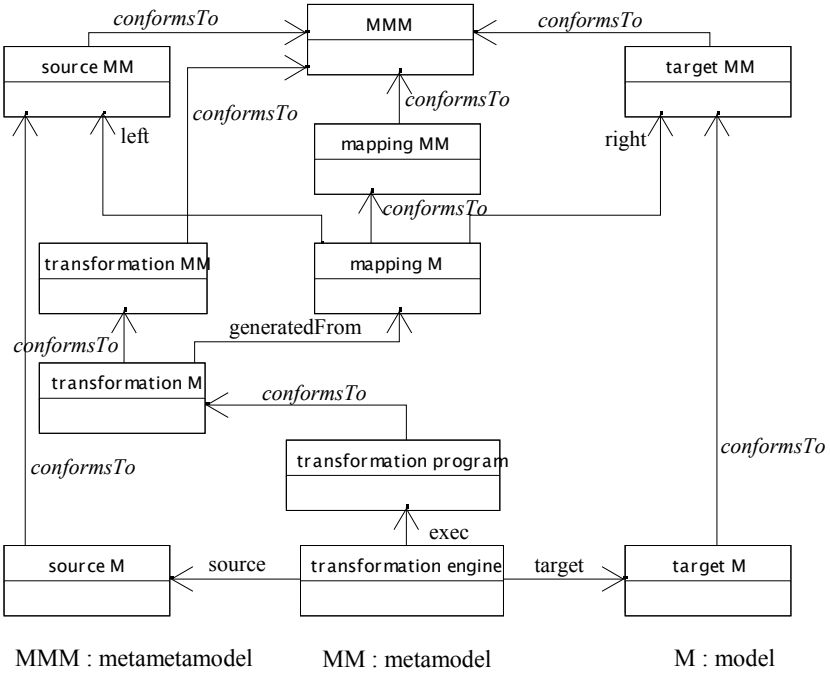


Figure 3.2 – Une proposition d'architecture type pour la transformation de modèles [114]

Cette proposition d'architecture type pour la transformation de modèles contient les entités suivantes :

- `MMM` (un métamétamodèle) : par exemple, MOF ou Ecore.
- `source MM` et `target MM` (un métamodèle source et cible) : par exemple, le métamodèle de UML ou de UEML.
- `source M` et `target M` (un modèle source et un cible): par exemple, un modèle d'une agence de voyages en UML qui doit être transformé dans un modèle Java.
- `mapping MM` (un métamodèle de correspondance) : le langage pour modéliser les correspondances entre les éléments d'un métamodèle source et les éléments d'un métamodèle cible.
- `mapping M` (un modèle de correspondance) : le modèle contenant les correspondances entre deux métamodèles.
- `transformation MM` (un métamodèle de transformation) : le formalisme qui permet la création précise de définitions de transformations d'un modèle source en un modèle cible.
- `transformation M` (un modèle de transformation) : décrit la transformation d'un métamodèle dans un autre.
- `transformation program` (un programme de transformation) : un programme exécutable pour réaliser la transformation.
- `transformation engine` (un moteur de transformation) : exécute un programme de transformation.

Le modèle de correspondance spécifie les correspondances entre un métamodèle source et un métamodèle cible. Un modèle de transformation est généré à partir d'un modèle de correspondance. Un programme de transformation est conforme à un modèle de transformation. La transformation est alors faite par un moteur de transformation qui exécute un programme de transformation. Le moteur de transformation prend comme entrée un modèle source, exécute le programme de transformation, et fournit un modèle cible comme résultat.

3.4 Modélisation en UML

Les besoins peuvent être exprimés sous forme de diagrammes de cas d'utilisation d'UML. Ce diagramme est un langage proche du langage naturel et permet la décomposition du système en conformité avec les fonctionnalités demandées. Chaque cas d'utilisation apparaît alors comme une classe de scénarios.

La figure 3.3 fournit le diagramme de cas d'utilisation pour l'agence de voyages et présente un client (`Customer`) qui planifie son voyage. Ce client accède au site Web de l'agence de voyages qui vend des billets d'avion, propose la location de voitures et la réservation d'une chambre d'hôtel (`TravelService`). Le client exprime dans un premier temps ses choix auprès du système de l'agence de voyages via son navigateur Web. L'agence de voyages reçoit les choix du client et les envoie à la compagnie aérienne (`AirLineServices`), à la compagnie de location de voitures (`RentingCarService`) et au réseau d'hôtels (`HotelService`). Elle reçoit ensuite les possibilités de vols, de location de voitures et de réservation de chambres de ses partenaires, qu'elle renvoie au client. Le client choisit le billet d'avion, la voiture et la chambre qui lui conviennent, et effectue la réservation. Enfin, il saisit les informations de paiement, l'agence de voyages les récupère et les renvoie à la banque (`BankService`) pour effectuer la transaction bancaire. Pour finir, l'agence de voyages paie ses partenaires pour leurs services en utilisant le service bancaire.

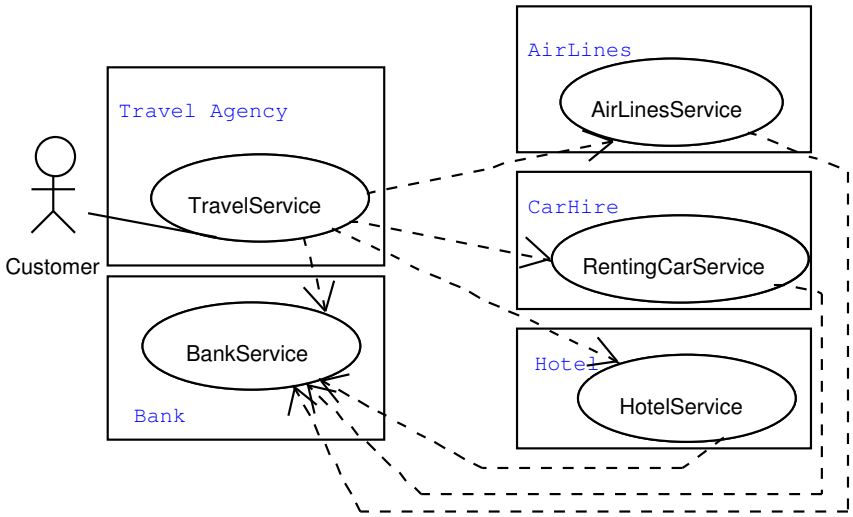


Figure 3.3 – Le cas d'utilisation de l'agence de voyages

La figure 3.4 donne un diagramme de classes pour le système de l'agence de voyages. Six classes réalisent les principales caractéristiques du système de notre exemple : `Customer`, `ServiceTravelAg`,

`ServiceBank`, `ServiceHotel`, `ServiceCarHire` et `ServiceAirLines`.

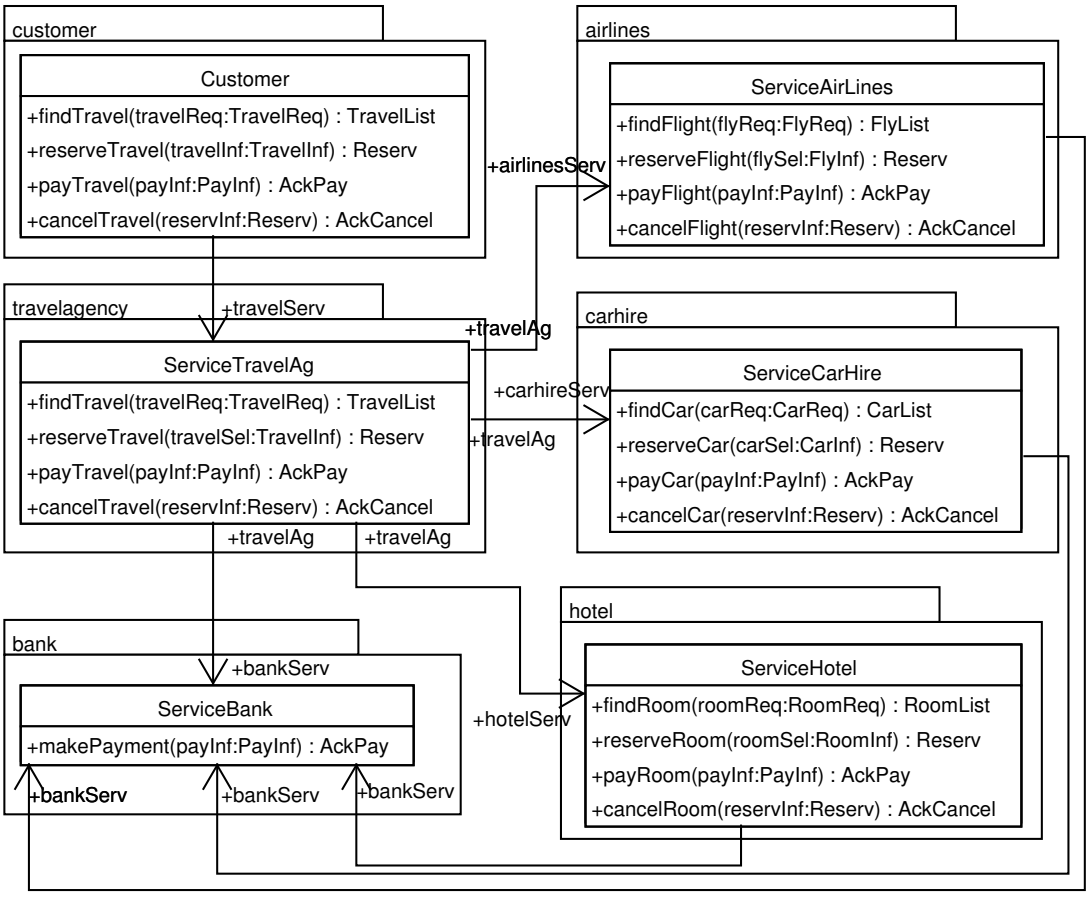


Figure 3.4 – Le diagramme de classe de l'agence de voyages

La figure 3.5 illustre un diagramme d'activité pour l'agence de voyages. Dans ce diagramme, l'agence de voyages fournit le service de recherche d'une chambre d'hôtel, d'une location de voiture et d'un billet d'avion. Pour réaliser ce service, l'agence de voyages fait la composition de trois services : `ServiceAirLines`, `ServiceCarHire` et `ServiceHotel`. Dans ce processus, l'agence de voyages reçoit une requête d'un client (illustré par l'action `ReceiveFindTravel`). Ensuite, le processus principal démarre trois processus secondaires en parallèle par le biais d'un `Fork`. Dans chaque processus démarré, une action d'affectation (`AssignInputFlight` ou `AssignInputCar` ou `AssignInputRoom`) fait la copie des données reçues dans l'action `ReceiveFindTravel` dans l'entrée d'une action réalisée par un service (`CallFlight` ou `CallCar` ou `CallRoom`). Par la suite, le processus principal (exécute `Join`) attend l'exécution des processus démarrés précédemment (par le `Fork`). Postérieurement, une action d'affectation copie le résultat de chaque service pour être envoyé au client. L'action `ReplyFindTravel` retourne le résultat de la recherche au client.

En réalité, chaque méthode de la classe `ServiceTravelAg` correspond à un diagramme d'activité. Et chaque diagramme d'activité est créé dans le contexte de la classe `ServiceTravelAg`.

Afin de modéliser le processus métier de l'agence de voyages comme une composition de services fournis par ses partenaires, nous utilisons des valeurs marquées d'UML pour lier le diagramme d'activité

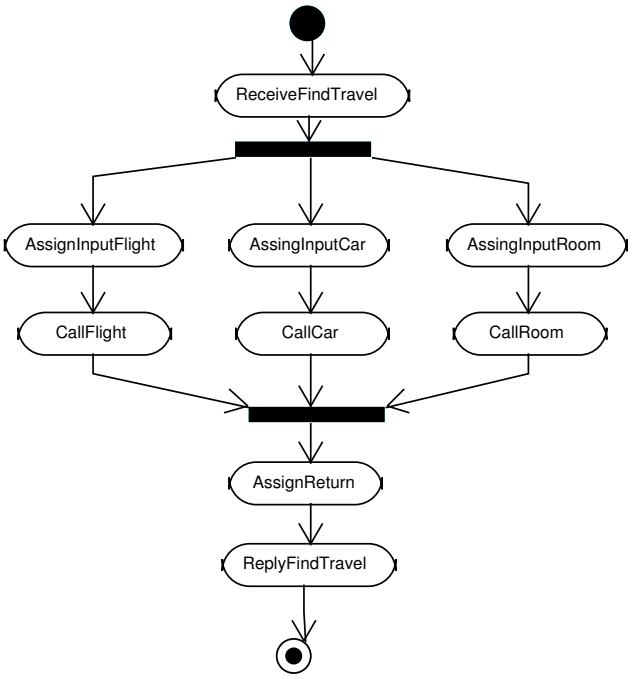


Figure 3.5 – Le diagramme d'activité de l'agence de voyages (fragment)

au diagramme de classe et permettre sa transformation vers BPEL4WS. À chaque action, des valeurs marquées sont utilisées pour faire la référence à une classe ou à une opération du diagramme de classe. Le tableau 3.3 présente dans la première colonne les actions, au milieu les marques, et à droite les valeurs.

L'utilisation de valeurs marquées est nécessaire parce qu'il y a une différence syntaxique et sémantique entre le métamodèle d'UML (version 1.4) et de BPEL4WS (version 1.1).

Dans le métamodèle de BPEL4WS, chaque action est réalisée par une opération d'un service, et chaque service est associé à un `partnerLink`. Celui-ci contient les informations sur le rôle et l'interface (`PortType`) du service qui réalise cette opération.

Dans le métamodèle d'UML (version 1.4), les actions (`ActionState`) ne sont pas liées à une opération (`Operation`) d'une classe. Dans ce cas, nous utilisons les valeurs marquées pour établir cette association (action - opération - classe). De plus, les actions peuvent correspondre à un appel (`invoque` en BPEL), une réception (`receive` en BPEL) ou une affectation (`assign` en BPEL).

D'autres travaux ont aussi remarqué cette limitation en UML et ont utilisé les profils pour la surmonter [180]. Cette limitation n'existe pas en UML 2.0 qui contient les éléments `AcceptCallAction`, `ReplyAction`, `CallOperationAction`, `ReadVariableAction`, `WriteVariableAction`. Ces éléments permettent de préciser l'élément `Action` comme cela a été démontré dans [27].

Dans notre approche, nous avons utilisés des marques telles que « activity » (acceptant les valeurs « receive » , « assign », « invoque » et « reply »), « class » (acceptant le nom d'une classe) et « operation » (acceptant le nom d'une opération).

Les informations ajoutées dans les actions par le biais des valeurs marquées seront utilisées plus tard pour réaliser la transformation du diagramme d'activité dans vers BPEL [9].

Action	**Marque**	**Valeur**
ReceiveFindTravel	activity	receive
	class	ServiceTravelAg
	operation	findTravel
AssignInputFlight	activity	assign
	class	ServiceTravelAg
	fromOperation	findTravel
	toClass	ServiceAirLines
	toOperation	findFlight
CallFlight	activity	invoke
	class	ServiceAirLines
	operation	findFlight
AssingInputCar	activity	assign
	class	ServiceTravelAg
	fromOperation	findTravel
	toClass	ServiceCarHire
	toOperation	findCar
CallCar	activity	invoke
	class	ServiceCarHire
	operation	findCar
AssingInputRoom	activity	assign
	class	ServiceTravelAg
	fromOperation	findTravel
	toClass	ServiceHotel
	toOperation	findRoom
CallRoom	activity	invoke
	class	ServiceHotel
	operation	findRoom
AssignReturn	activity	assign
	class	ServiceAirLines
	toClass	ServiceTravelAg
	fromOperation	findFlight
	toOperation	findTravel
ReplyFindFlight	activity	reply
	class	ServiceTravelAg
	operation	findTravel

Table 3.3 – Les valeurs étiquetées de chaque action du diagramme d'activité (figure 3.5)

3.5 Modélisation en EDOC

Avant de proposer les modèles de l'agence de voyages en EDOC, nous présentons dans le tableau 3.4 la notation EDOC-CCA[1] et sa relation avec le métamodèle d'EDOC [145].

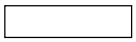
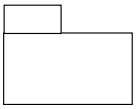
Symboles graphiques	Éléments du métamodèle	Description
	• FlowPort • PortConnector	• Port de communication reçevant des données. • Port utilisé par une connexion.
	• FlowPort • PortConnector	• Port de communication émettant des données. • Port utilisé par une connexion.
	• ProtocolPort • PortConnector	• Port de communication représentant une interaction complexe entre composants. Le composant contenant ce port est appelé par un autre composant via ce port. • Port utilisé par une connexion.
	• ProtocolPort • PortConnector	• Port de communication représentant une interaction complexe entre composants. Le composant contenant ce port initie la communication. • Port utilisé par une connexion.
	• ProcessComponent • CommunityProcess • ComponentUsage	• Composant réalisant un traitement spécifique (achat, vente, gestion de stock, etc). • Composition de haut niveau dans laquelle des composants collaborent pour réaliser un travail donné. L'imbrication de ce symbole représente à la fois un des concepts précédents et le concept ComponentUsage. • Décrit l'utilisation d'un composant.
	• Connection	• Une Connection représente une communication entre deux composants. Connecte deux ports de communication par l'intermédiaire de PortConnector.

Table 3.4 – La notation EDOC-CCA

Un composant EDOC-CCA possède une vue externe qui inclut la définition de ses propriétés et la déclaration de ses ports. Ces ports permettent de réaliser des interactions simples ou plus évoluées entre composants. Une communication complexe s'effectue par un port qui peut contenir des sous-ports, les interactions avec ce type de port étant décrites à l'aide d'un protocole. Ce dernier permet de préciser l'initiateur et le récepteur de la communication, ainsi que l'ordre des informations qui sont échangées entre ces interlocuteurs.

La figure 3.6 présente le *Community Process* de l'agence de voyages modélisé avec EDOC-CCA. Il s'agit d'un modèle simplifié qui présente uniquement les participants au processus métier. Le *Community*

[1] EDOC-CCA (*Enterprise Distributed Object Computing-Component Collaboratiion Architecture*)

Process décrit comment le client (`Customer`), l'agence de voyages (`TravelAg`), la compagnie aérienne (`AirLines`), la compagnie de location de voitures (`CarHire`), le réseau d'hôtel (`Hotel`) et la banque (`Bank`) collaborent pour effectuer une transaction commerciale.

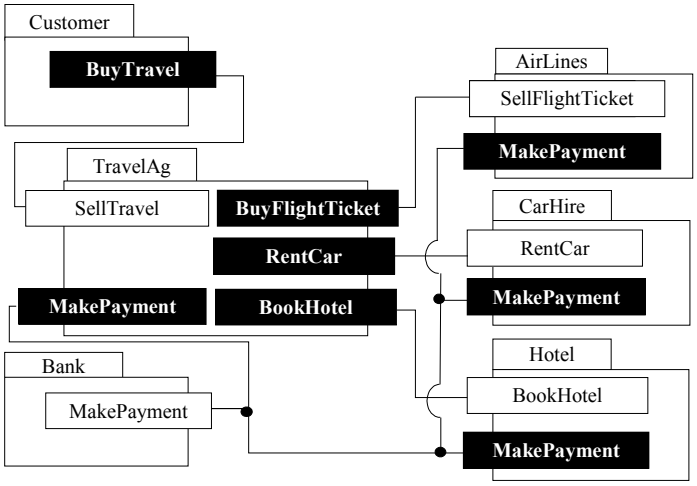


Figure 3.6 – Le *Community Process* de l'agence de voyages (Notation CCA)

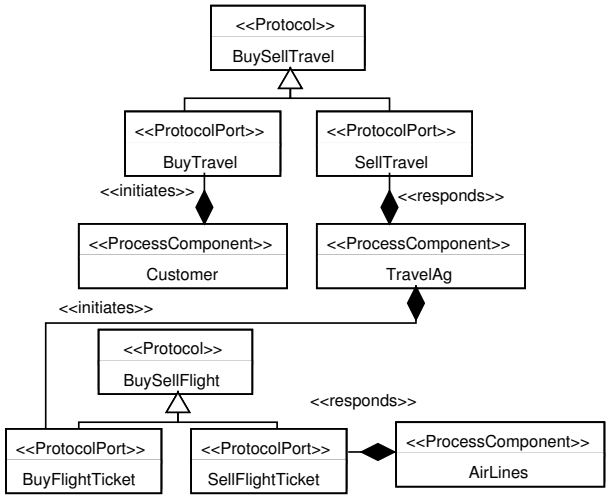


Figure 3.7 – Les contrats de composants

Le client (`Customer`) collabore directement avec l'agence de voyages (`TravelAg`) à travers les ports de protocole `BuyTravel` et `SellTravel`, en conformité avec le protocole `BuySellTravel`. Le `TravelAg` collabore avec `AirLines` à travers les ports de protocoles `BuyFlightTicket` et `SellFlightTicket`, en conformité avec le protocole `BuySellFlight`. Il collabore aussi avec `CarHire`, `Hotel` et `Bank` pour effectuer une transaction. La figure 3.7 illustre le contrat entre les composants en utilisant les profils UML pour EDOC (avec une notation UML) [145]. La figure 3.8 fournit l'interface pour le composant `TravelAg`.

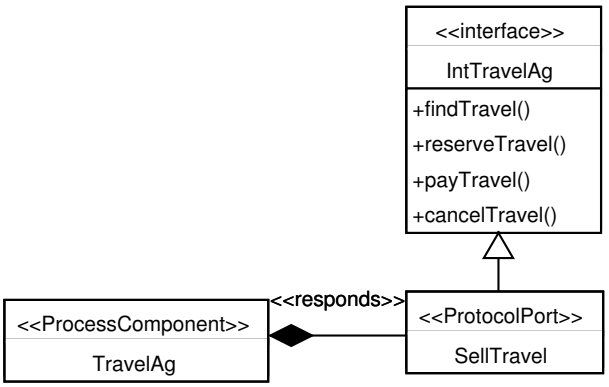


Figure 3.8 – L'interface pour le composant TravelAg

Les activités dans les *community process* de l'agence de voyages démarrent avec le client (`Customer`). Il initialise les interactions sur son port de protocole `BuyTravel`, en conformité avec le protocole `BuySellTravel` présenté par la figure 3.7. La figure 3.9 détaille la structure de ce protocole, et la figure 3.10 illustre la chorégraphie. La structure est composée d'autres protocoles, tels que `FindTravel`, `ReserveTravel`, `CancelTravel`, etc. L'agence de voyages (`TravelAg`) suit le protocole `BuySellTravel`, et initialise les protocoles `BuySellFlight`, `BuySellHotel` au moment opportun.

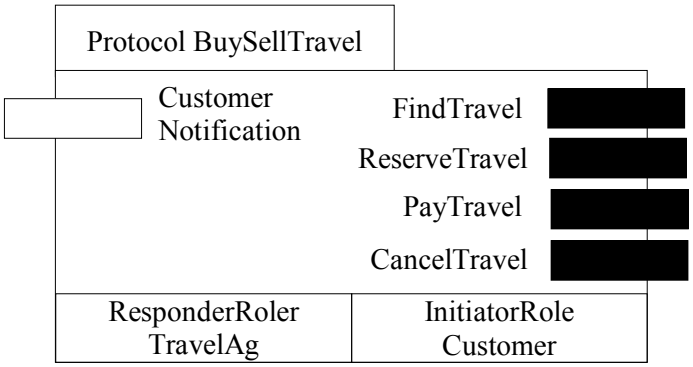


Figure 3.9 – La structure du protocole BuySellTravel

Les interactions dans le protocole `BuySellTravel` démarrent avec l'`InitiatorRole` qui initialise et réalise les interactions avec le `FindTravel` (`ProtocolPort`). Cet `InitiatorRole` prépare le voyage (composé des vols, des réservation de voitures, et des réservation de chambres d'hôtels) puis répond et réalise des interactions via le `Customer Notification` (`ProtocolPort`). Ensuite l'`InitiatorRole Customer` initialise et réalise les interactions avec le `ReserveTravel` (`ProtocolPort`), et ainsi de suite (conformément à la figure 3.10).

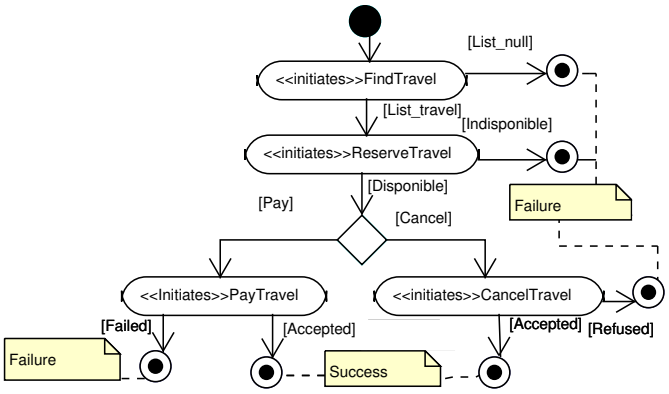


Figure 3.10 – La chorégraphie du protocole BuySellTravel

Le protocole `BuySellTravel` est composé d'autres protocoles, tel que le protocole `FindTravel` (cf. figure 3.11 et figure 3.12). Le protocole `FindTravel` est composé de `FindFlight`, `FindCar` et `FindHotel`. L'`InitiatorRole` invoque `FindFlight`, `FindCar` et `FindHotel`, et il reçoit les listes de possibilités. Si le `TestLists` vérifie que les listes ne sont pas vides, l'`Initiator` recevra un `ListTravel` ; sinon, il recevra un `ListNull`.

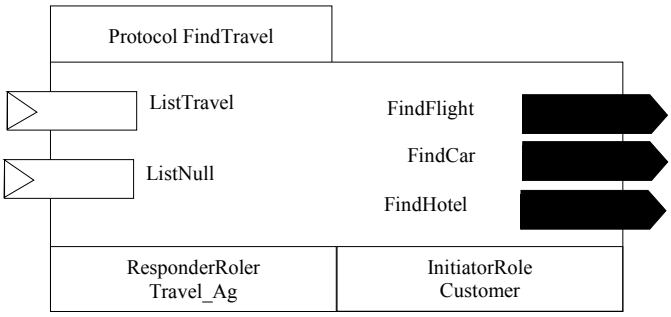


Figure 3.11 – La structure du protocole FindTravel

Enfin, le modèle de l'agence de voyages en EDOC est constitué des composants qui participent dans un processus. Ces composants interagissent par le biais de protocoles qui déterminent l'utilisation de ports. Chaque protocole est exécuté selon une chorégraphie qui détermine l'ordre d'utilisation de ports.

Le modèle de l'agence de voyages conforme à EDOC a été présenté en utilisant la notation CCA[2] et des modèles UML (avec des profils pour EDOC). Actuellement, presque tous les AGL[3] UML supportent la création et l'édition de modèles avec profils, tel que *Poseidon for UML* qui est basé sur la version 1.4 de MOF. Ainsi, la création du modèle de l'agence de voyages conforme au profil EDOC-CCA a été faite sans problème. Cependant, les AGL EDOC ne sont pas aussi nombreux que les AGL UML. De plus, les AGL EDOC ne sont généralement pas basés sur MOF. De ce fait, nous avons créé un artifice pour éditer des modèles EDOC-CCA conformes au métamodèle EDOC-CCA qui, à son tour, lui-même doit être conforme à MOF. Ainsi, nous avons proposé un métamodèle de EDOC-CCA inspiré de la spécification de EDOC [152] et conforme à MOF (annexe A, figure A.1). Une fois que le métamodèle EDOC-CCA

[2]La notation CCA est seulement une notation graphique pour représenter un modèle conforme au métamodèle EDOC-CCA.
[3]AGL (Atelier Génie Logiciel).

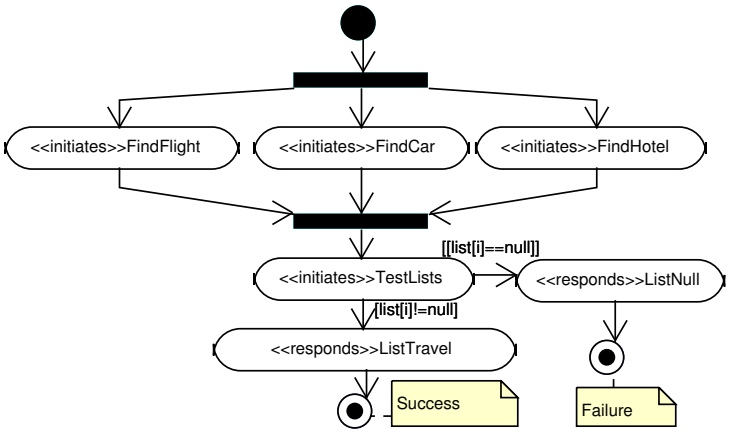


Figure 3.12 – La chorégraphie de protocole FindTravel

était disponible, nous avons utilisé une définition de transformations écrite en ATL (annexe A, exemple A.1) pour créer le modèle de l'agence de voyages (figure 3.6).

3.6 Les Services Web comme plate-forme cible

La plate-forme des services Web est constituée de différentes technologies. La figure 3.13 illustre ces technologies et leurs relations.

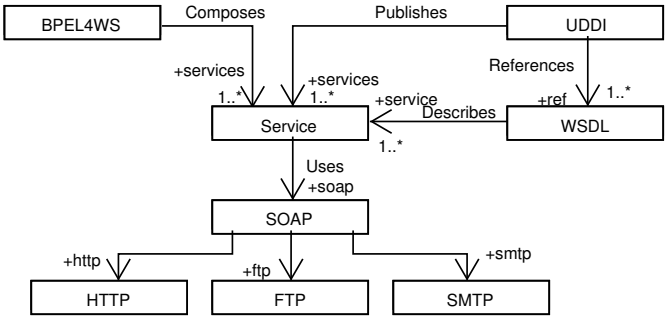


Figure 3.13 – Les principales technologies des Services Web

L'utilisation de services comme plate-forme cible dans le contexte de MDA doit être en conformité avec le processus d'application de MDA présenté précédemment (figure 3.1). Nous devons donc choisir ou créer des métamodèles[4] pour les plates-formes cibles. Parmi les technologies constituant les services Web, les plus importantes pour notre recherche sont WSDL, UDDI et BPEL4WS. A ce jour, il n'y a pas un métamodèle officiel de WSDL, UDDI et BPEL4WS. Nous présentons donc des métamodèles créés en conformité avec les schémas XML proposés dans les spécifications de WSDL [205], UDDI [133] et BPEL4WS [9].

[4]Selon la figure 1.21, un modèle doit être conforme à un métamodèle qui, à son tour, doit être conforme à un métamétamodèle. Ainsi, nous avons proposé des métamodèles conformes à MOF (utilisés dans les expérimentations présentées dans les chapitres 3 et 4) et aussi des métamodèles conformes à Ecore (utilisés dans les expérimentations présentées dans le chapitre 5).

La figure 3.14 présente un métamodèle proposé pour WSDL obtenu à partir de son schéma [205]. Dans l'annexe A (figure A.2), un métamodèle de WSDL plus complet est fourni.

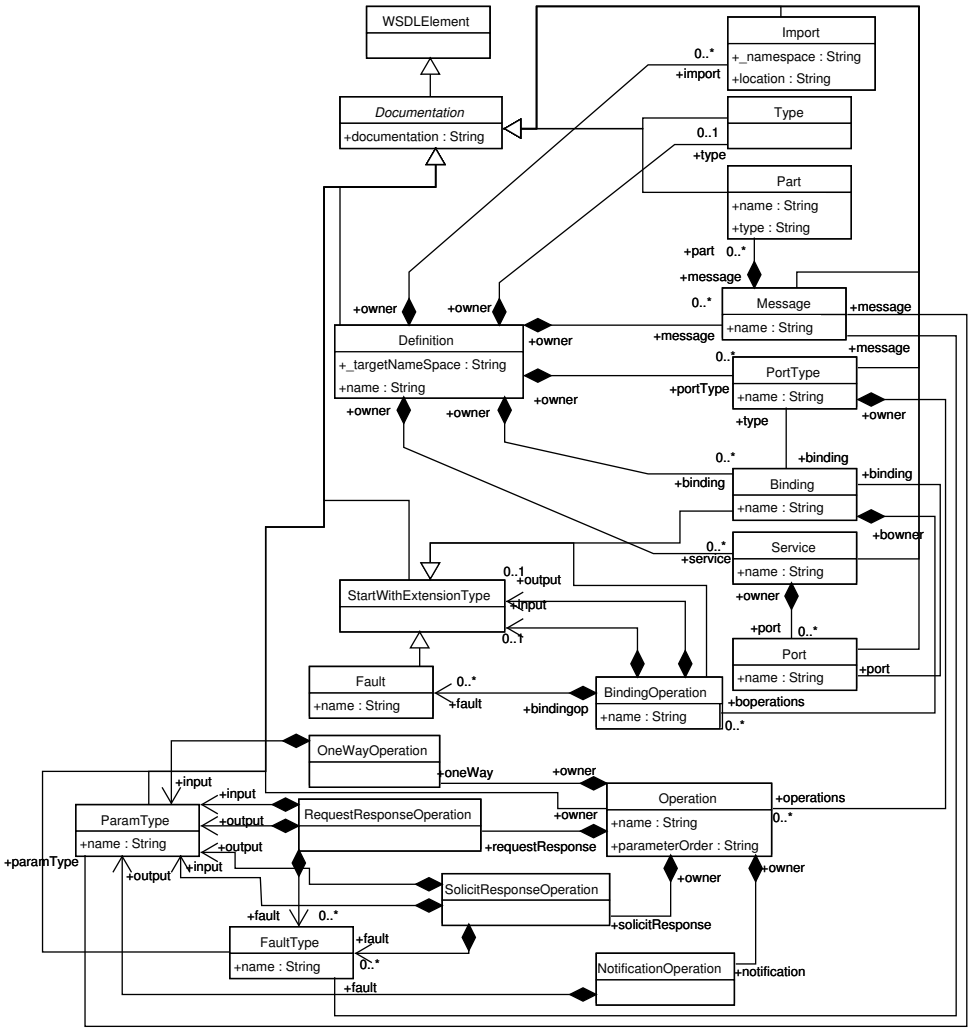


Figure 3.14 – Un métamodèle de WSDL (fragment)

Ce métamodèle de WSDL est composé des éléments suivants :

- `Definition` : élément principal de ce métamodèle contenant un ensemble de `Import`, `Type`, `Message`, `PortType`, `Binding` et `Service`.
- `Import` : permet l'association d'un espace de noms (`namespace`) à la localisation d'un document XML.
- `Type` : utilisé pour définir un type de données abstraites simples ou complexes en conformité avec un schéma XML.
- `Message` : décrit un format abstrait d'un message particulier que le service Web envoie ou reçoit. Il contient des parties (par exemple `Part`) décrivant chaque partie d'un message.
- `portType` [5] : définit l'interface d'un service. Il contient un ensemble d'operations qu'un service

[5] `PortType` est appelé `Interface` à partir de WSDL v1.2.

envoie et/ou reçoit. Ces opérations sont caractérisées par l'élément `Operation` qui décrit les types d'appels de manière abstraite. Les types d'appel sont caractérisés par les éléments `OneWayOperation`, `RequestResponseOperation`, `SolicitResponseOperation` et `NotificationOperation`. L'élément `ParamType` identifie quels sont les messages de `input`, `output` et `fault`.

- `Binding` : décrit une liaison concrète des composantes d'une interface avec le protocole de communication utilisé, c.-à-d. la manière dont un appel peut être achevé de façon concrète (par exemple, en utilisant SOAP avec HTTP). L'élément `BindingOperation` contient les `input`, `output`, et `fault` décrits selon la réalisation concrète de l'appel.
- `Service` : décrit un service, identifie son interface (`PortType`) et ses localisations (`endpoints`).

La figure 3.15 présente un métamodèle de UDDI. Ce métamodèle est basé sur celui proposé dans le « *UDDI Executive White Paper* » [132] avec quelques modifications.

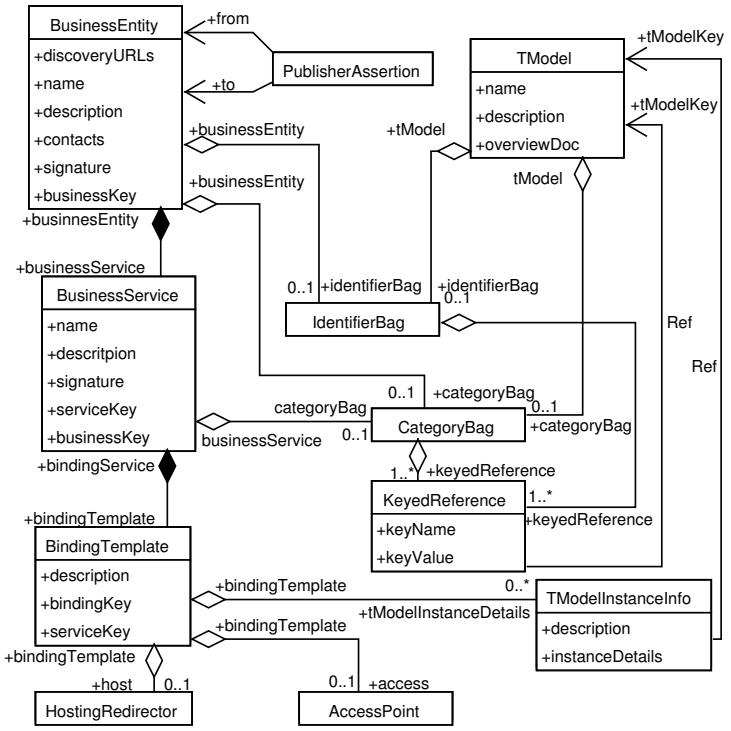


Figure 3.15 – Un métamodèle de UDDI

Ce métamodèle est composé des éléments suivants :

- `BusinessEntity` : élément principal qui supporte la publication et le stockage des informations rélatifs à un fournisseur de service et à son métier. Il contient le nom, une description, des taxonomies et des contacts rélatifs au service métier ou à une organisation qui fournit les services Web.
- `BusinessService` : description d'un ensemble de services Web conforme au type de métier ou à la catégorie de services. Il hiérarchise les services en catégories d'industrie, de produit et service.
- `BindingTemplate` : contient une information technique pour utiliser un service Web individuel. Il fournit le point d'accès ou un mécanisme de repère (*indirection mechanism*) pour le point d'accès.

- `TModelInstanceInfo` : contient les détails spécifiques d'une entité de `BindingTemplate` pour chaque `TModel` référencé.
- `TModel` : contient des métadonnées de services qui peuvent être une spécification, un protocole de transport ou un espace de noms.
- `PublisherAssertion` : décrit les relations entre `BusinessEntities`.

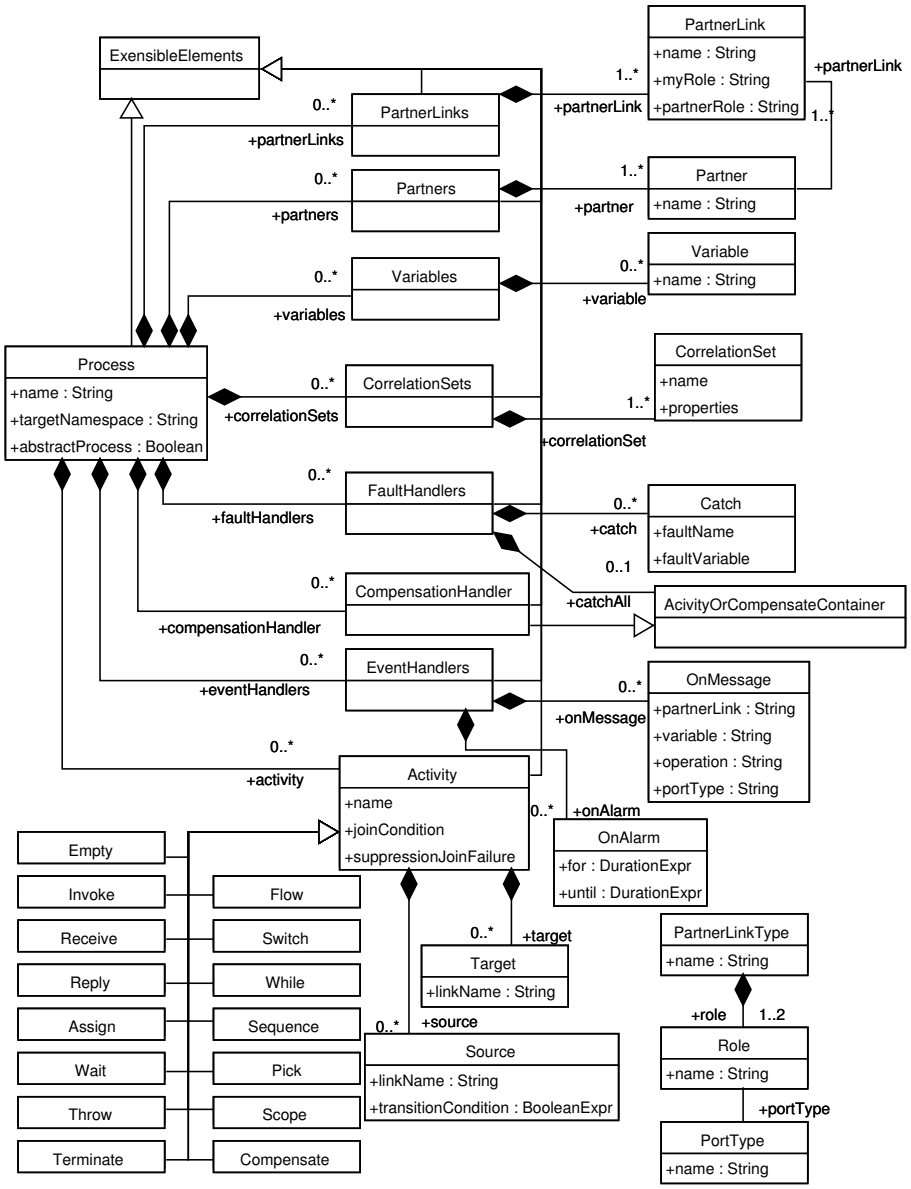


Figure 3.16 – Un métamodèle de BPEL4WS (fragment)

BPEL4WS est un langage qui permet la définition et l'exécution d'un processus dans lequel les activités peuvent être constituées de services Web. BPEL4WS a un rapport étroit avec WSDL car il fait référence à un `portType` de services utilisés dans le processus. La figure 3.16 présente le fragment d'un métamodèle de BPEL4WS obtenu à partir de son schéma [9]. Un métamodèle de BPEL4WS plus complet est présenté dans l'annexe A (figure A.3).

Ce métamodèle de BPEL4WS est constitué des éléments suivants :

- `Process` : contient un ensemble d'activités (`Activity`), `Partners`, `Variables`, `CorrelationSets`, `FaultHandlers`, `CompensationHandlers` et `EventHandlers`.
- `PartnerLinks` : définit les différentes parties qui agissent les unes sur les autres lors de l'exécution d'un processus. Il est caractérisé par un ou plusieurs `PartnerLink` qui spécifient les relations de conversation entre deux services par le biais de la déclaration de leurs rôles. Chaque rôle spécifie un `portType` de WSDL qu'un partenaire réalise.
- `Partner` : sous-ensemble de `PartnerLink` (références), il introduit une contrainte de la fonctionnalité qu'un partenaire doit fournir.
- `Variables` : définit les variables de données utilisées par le processus, et permet au processus de maintenir les données et les historiques basés sur l'échange de données. Une variable peut être définie en terme de type de message WSDL (`Message`), type simple (`type`) ou complexe (`element`) décrit en schéma XML.
- `CorrelationSets` : ensemble de propriétés utilisées pour prendre en charge les corrélations. Il spécifie les groupes d'opérations ayant une corrélation dans une instance de service.
- `FaultHandlers` : gestionnaire de défauts de l'exécution de services définissant les activités qui seront exécutées en réaction à la faute.
- `CompensationHandler` : *wrapper* pour les activités de compensation.
- `EventHandlers` : ce gestionnaire est appelé quand un évènement déterminé arrive et résulte dans l'invocation des activités.
- `Activity` : activité structurée en plusieurs parties comme le flux de contrôle (par exemple, `Switch` et `While`), de messages (données transférées entre les activités à travers `Assign`), de transactions (transactions de longue durée supportées dans un processus unique) et l'extensibilité (l'espace de nom en forme d'URI peut être utilisé dans plusieurs éléments de BPEL4WS).

3.7 La plate-forme J2EE

La plate-forme J2EE est constituée de différentes technologies, notamment le langage Java, JDBC (*Java DataBase Connectivity*), JWSDP (*Java Web Services Developer Pack*), EJB (*Enterprise JavaBeans*) et JSP (*Java Server Pages*). Parmi ces technologies, nous nous intéressons particulièrement au langage Java et JWSDP. Ce dernier est le *framework* du monde Java permettant la programmation et le déploiement de services Web. Ainsi, nous présentons un métamodèle de Java et un autre de JWSDP[6] (*template*, fichiers de déploiement et fichiers de configuration).

La figure 3.17 expose un possible métamodèle de Java. D'autres métamodèles de Java peuvent être trouvés dans la littérature (par exemple, celui proposé dans la spécification d'UML *profile for* EDOC [145]) ou sur Internet (par exemple, celui proposé par NetBeans [7]).

Ce métamodèle de Java est constitué des éléments suivants :

- `JElement` : élément racine de ce métamodèle.
- `JPackage` : contient `JClass`, `JInterface`, etc.
- `JClassifier` : contient `JMember` (`JField` et `JMethod`).
- `JClass` : spécialisation de `JClassifier`, implémente une `JInterface`.

[6] Dans nous expérimentations, nous utilisons JWSDP avec Tomcat version 5.0 (comme le serveur des applications Web).
[7] Disponible sur http://java.netbeans.org/models/java/java-model.html

- `JInterface` : autre spécialisation de `JClassifier`, contenant seulement les prototypes de méthodes (sans le corps) et attributs de type `final static` (constant).
- `JField` : contient seulement un `JPrimitiveType` ou `JClass` ou `JInterface`.
- `JMethod` : contient les opérations (i.e. comportement) d'une classe ou interface en Java. Il a un paramètre de retour et un ou plusieurs paramètres d'entrée.
- `JParameter` : spécifie les paramètres d'une méthode en Java.

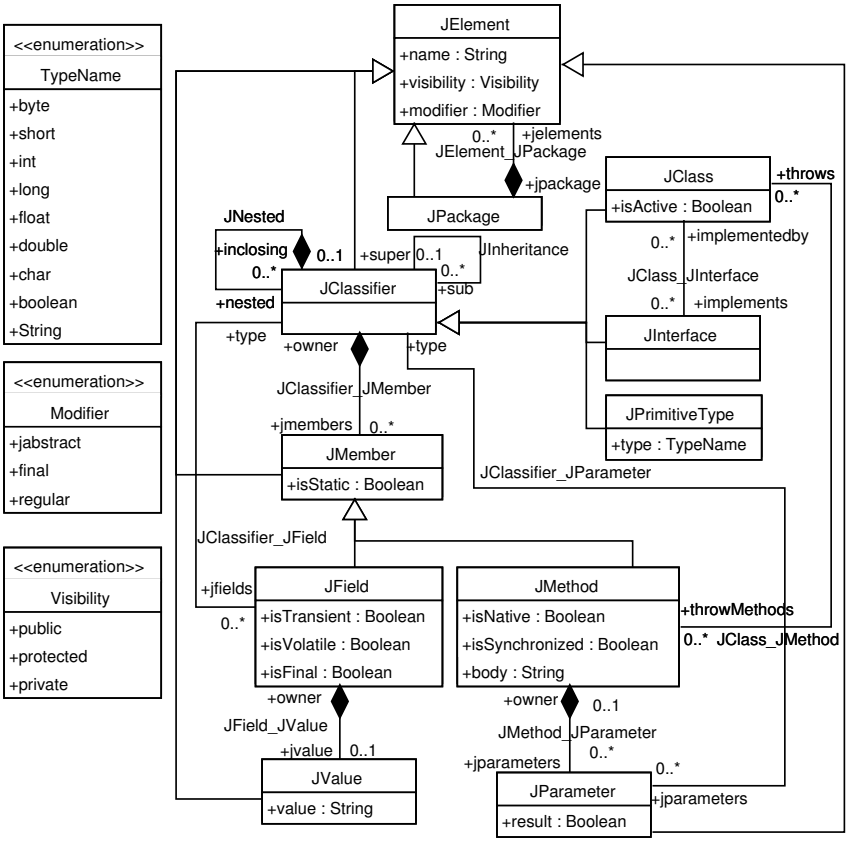
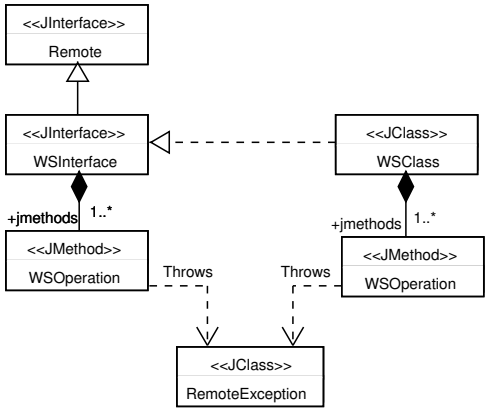


Figure 3.17 – Un métamodèle de Java (fragment)

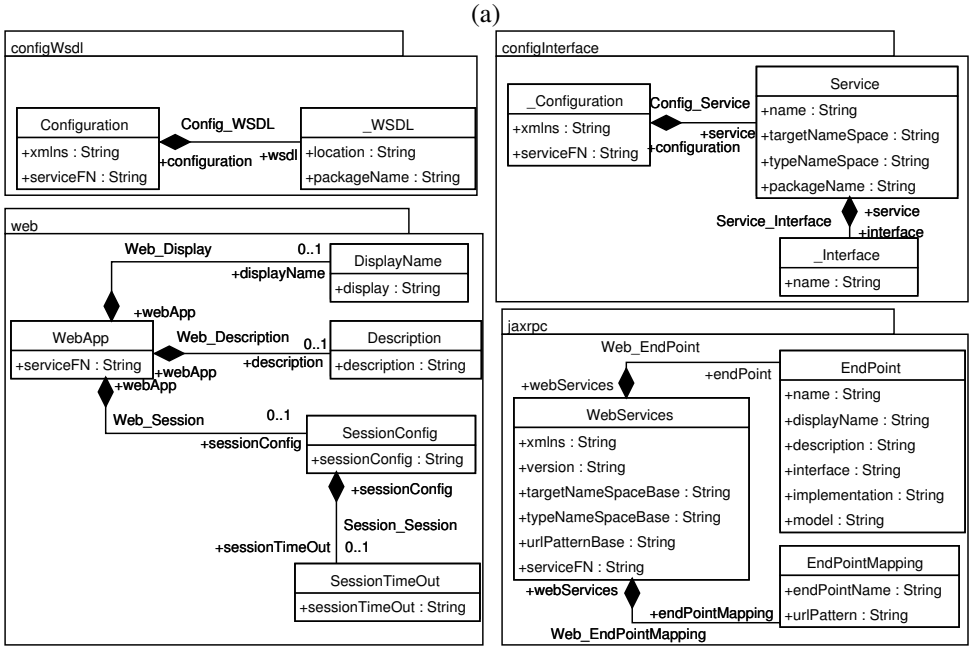
La figure 3.18 présente (a) un *template* (modèle) et (b) un métamodèle pour JWSDP. Ce *template* décrit la réalisation d'un service Web en JWSDP. En réalité, ce *template* spécifie l'utilisation de l'API de JWSDP qui se situe au niveau modèle dans l'architecture de quatre niveaux de métadonnées [141]. Un service Web est créé en JWSDP à travers une `WSClass` (conforme à `JClass`) qui implémente une `WSInterface` (conforme à `JInterface`). Cette interface doit hériter de `java.rmi.Remote` et leurs méthodes doivent lancer une `java.rmi.RemoteException` en cas d'exception. Le `WSClass` réalise le service et la `WSInterface` est utilisée par le client pour appeler ce service. Le métamodèle de JWSDP que nous proposons a été construit à partir de schémas XML et de DTD (*Document Type Definitions*) de la spécification de JWSDP version 1.3 [185]. Ce métamodèle de JWSDP est constitué de quatre *metapackages* : `configInterface`, `jaxrpc`, `web` et `configWsdl`. Ils contiennent des élément qui représentent les informations utilisées pour générer les fichiers de déploiement et de configuration de JWSDP : `config-interface.xml`, `jaxrpc-ri.xml`, `web.xml` et `config.xml` .

Les fichiers `config-interface.xml` et `jaxrpc-ri.xml` sont utilisés pour la compilation et la

génération de WSDL et de `Tie` (*stubs* du côté du serveur) d'un service Web. Le fichier `web.xml` est utilisé pour déployer un service Web dans le conteneur d'applications Tomcat. Le fichier `config.xml` est utilisé pour générer les *stubs* (pour le côté client) à partir d'un document WSDL qui décrit le service Web désiré.



(a)



(b)

Figure 3.18 – (a) un template et (b) un métamodèle pour JWSDP (fragments)

3.8 La plate-forme dotNET

La plate-forme dotNET est également composée de différentes technologies, tels que le langage C#, le *framework* pour les services Web et WebForms. Parmi ces technologies, nous sommes concernés par le langage C# et le support des services Web.

La figure 3.19 illustre un métamodèle de C# obtenu à partir de sa grammaire [124].

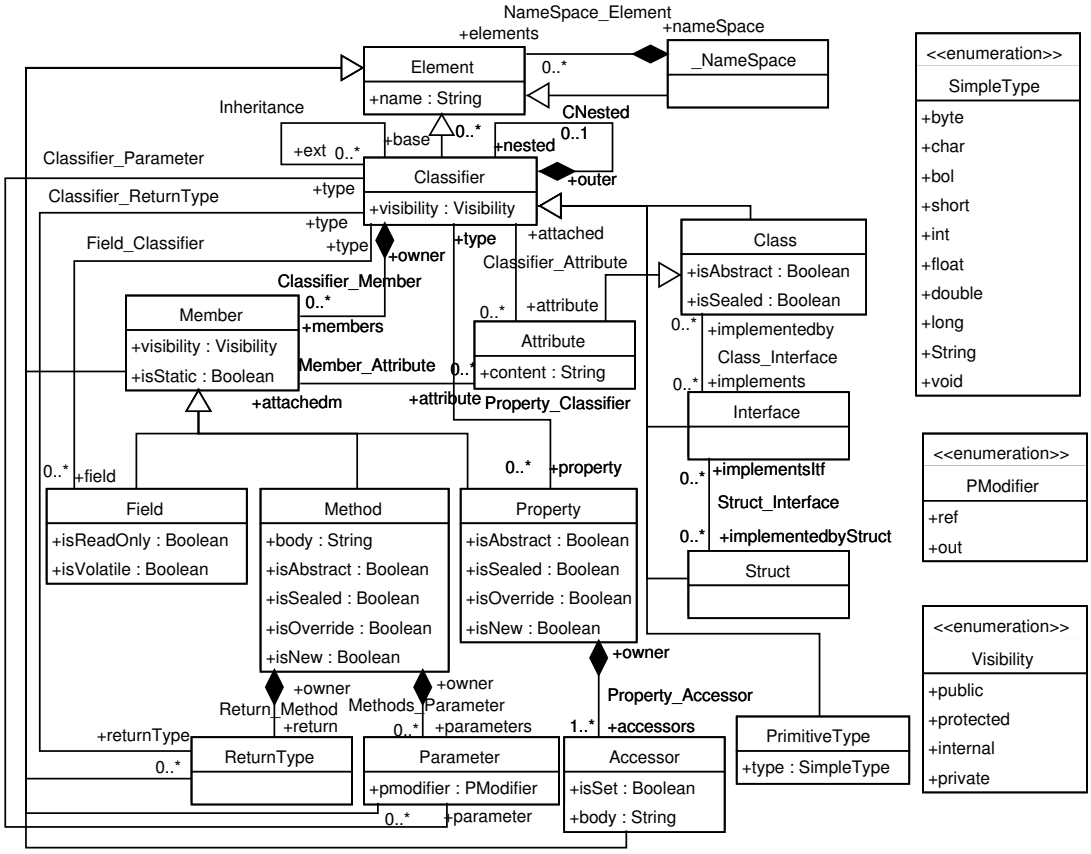


Figure 3.19 – Un métamodèle de C# (fragment)

Ce métamodèle de C# est constitué des éléments suivants :

- `Element` : élément racine.
- `Namespace` : conteneur pour d'autres éléments (`Class`, `Interface`, etc).
- `Classifier` : généralisation des membres, classes, interfaces et structures. Il contient un ensemble de membres qui peuvent faire partie de : `Field`, `Method`, et `Property`.
- `Class` : spécialisation de `Classifier`, réalise l'`Interface`.
- `Interface` : autre spécialisation de `Classifier`, contenant seulement les signatures de méthodes (sans le corps).
- `Struct` : similaire à `Class`, mais n'a pas d'héritage.
- `Attribute` : information déclarative pouvant être attachée aux entités de programmes (comme `Class` et `Methods`) et récupérée pendant l'exécution. Toutes les classes de type `Attribute` doivent dériver de la classe de base `System.Attribute` fournie par le *framework* de dotNET. Même si `Attribute` appartient à l'API de C# (modèle ou niveau M1), nous l'avons utilisé comme partie du métamodèle de C#, afin de le manipuler au niveau du métamodèle (niveau M2). La définition d'`Attribute` en C# est différente de celle en UML. D'une part, en UML, un attribut est une caractéristique d'un `Classifier` (par exemple, `Class`) qui décrit une gamme de valeurs dont le type est un `Classifier`. D'autre part, en C#, un attribut est utilisé pour fournir des informations additionnelles à un `Classifier` (par exemple, `Class` et `Methods`).

- `Member` : élément de base pour `Field`s et `Method`s.
- `Field` : composition d'un type pouvant être un `ValueType` ou un `ReferenceType`.
- `Method` : possède une signature d'opérations composée du type de retour, de l'identificateur et des paramètres.
- `Property` : présente des caractéristiques d'un `Field` et d'un `Method`. Il contient une méthode `get` et/ou `set`.
- `ReturnType` : type de retour d'une méthode.
- `Parameter` : arguments d'une opération.
- `Accessor` : il peut également être une méthode `get` ou `set` d'une propriété.

La création d'un service Web en dotNET avec C# est faite à travers l'utilisation des classes suivantes :

- `WebService` : classe de base pour toute autre classe qui réalise un service Web.
- `WebServiceAttribute` : attribut qui peut être associé à une classe qui réalise un service.
- `WebMethodeAttribute` : attribut associé aux membres d'une classe qui réalise un service.

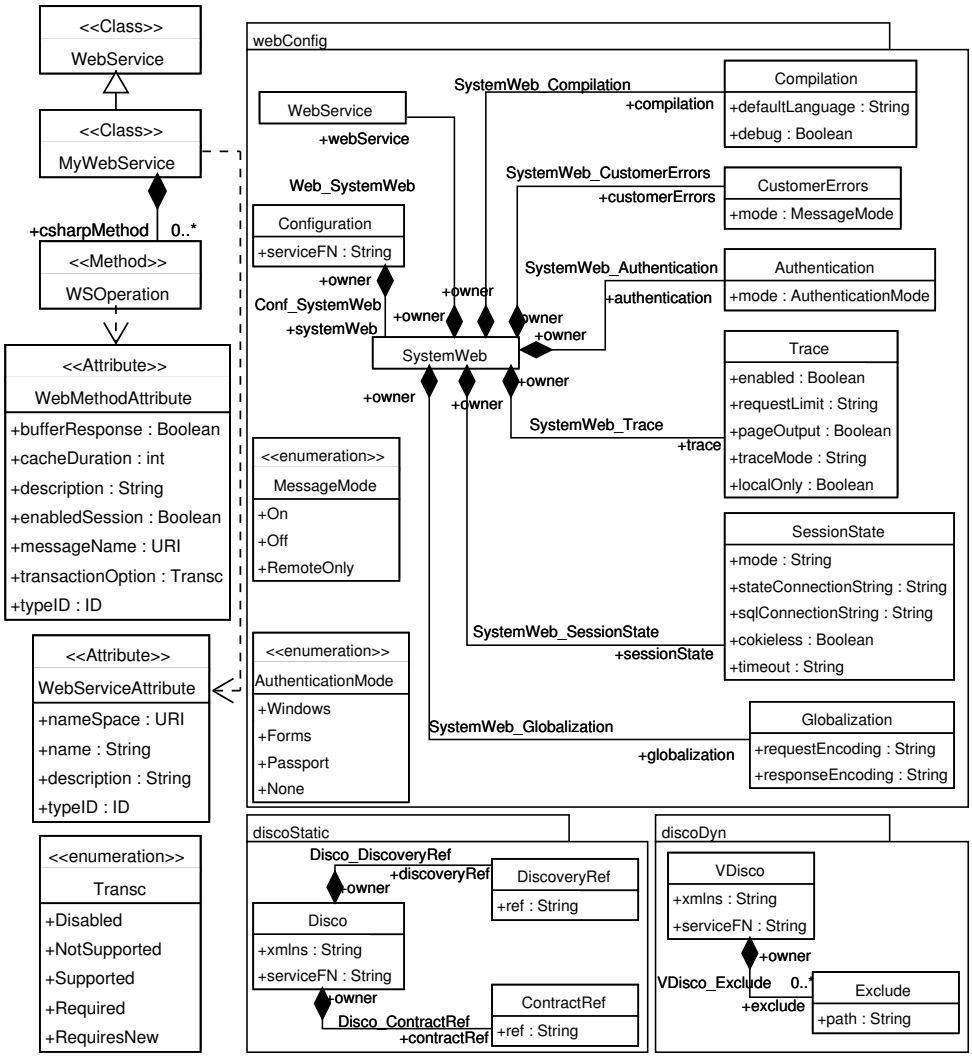


Figure 3.20 – (a) un *template* et (b) un métamodèle de dotNET (fragments)

La figure 3.20 présente un *template* et un métamodèle de dotNET. Il détaille les principaux éléments permettant de créer un service Web. Le template indique qu'un service Web est réalisé en utilisant une classe héritant de la classe `WebService`. Cette classe est attachée à un attribut appelé `WebService-Attribute`. Les méthodes invoquées à la façon d'un service sont liées à un attribut appelé `WebMethod-Attribute`. Le *template* de dotNET utilise l'API du *framework* dotNET pour créer et exécuter un service Web. Il appartient donc au niveau modèle (niveau M1). Le métamodèle de dotNET présente trois *metapackage* : `webConfig`, `discoStatic` et `discoDyn`. Ils sont nécessaires pour générer les fichiers de configuration et de déploiement d'un service Web sur l'IIS[8] (*Internet Information System*) [64]. `webConfig` contient des informations spécifiques qui permettent à IIS d'exécuter un service Web. `discoStatic` et `discoDyn` contiennent les informations pour publier un service Web et la localisation de la description du service, c'est-à-dire WSDL. Les deux derniers sont une alternative à UDDI.

3.9 Synthèse

Dans ce chapitre, nous avons exposé une méthodologie pour le développement d'applications selon l'approche MDA. Dans le cadre de cette méthodologie, nous avons proposé une architecture type pour la transformation de modèles dans laquelle nous avons explicitement distingué la spécification de correspondances et la définition de transformations.

Pour mettre en œuvre notre méthodologie, une étude de cas illustrative des applications Internet a été présentée puis modélisée. Pour cette modélisation, nous avons utilisé deux formalismes : UML et EDOC. Notre plate-forme cible abstraite est celle des services Web (WSDL et BPEL). Nous avons sélectionné deux plates-formes concrètes J2EE et dotNET qui sont les plus répandues aujourd'hui. Selon notre méthodologie, nous avons proposé un métamodèle (conforme à MOF) pour chaque plate-forme cible, généralement, obtenu à partir des schémas XML ou DTD.

Le chapitre suivant est réservé à la seconde phase importante de notre méthodologie dédiée à la spécification de correspondance et à la définition de transformation. Ensuite, nous appliquons les définitions de transformations à notre étude de cas pour générer le code source ou les fichiers de déploiements conformes à une de nos plates-formes cibles.

Cette expérimentation va nous permettre de tirer des conclusions sur notre approche, et aussi sur les langages de modélisation (UML et EDOC) et les plates-formes cibles (Services Web, J2EE et dotNET).

[8] Nous avons utilisés l'IIS (*Internet Information System*) fournit avec Windows Professionnel 2000.

CHAPITRE 4

Spécification de correspondances et définition de transformations

4.1 Introduction

Ce chapitre fait suite à la présentation de notre démarche permettant de développer les services Web dans le contexte de MDA. Précédemment, nous avons proposé un processus minimum d'application de la démarche MDA (figure 3.1, page 100) et une architecture type pour la transformation de modèles (figure 3.2, page 102) dans lesquels nous avons clairement distingué la spécification de correspondances et la définition de transformations. En effet, la spécification de correspondances vise à spécifier les correspondances entre les éléments de deux métamodèles, et la définition de transformations permet de décrire en détail et d'exécuter les étapes de la transformation d'un modèle en un autre modèle en respectant la spécification de correspondances. Cette approche va dans le sens de la vision de l'OMG pour développer les systèmes informatiques, la séparation des préoccupations (*concerns*). En fait, une spécification de correspondances peut être vue comme un PIM et une définition de transformations comme un PSM. Les deux décrivent la même chose mais à des niveaux différents.

Dans cette thèse, nous utilisons l'approche dirigée par les modèles pour générer les plates-formes cibles à partir d'un modèle métier, mais avec une forte granularité[1] : nous générons les squelettes du code source, ainsi que les fichiers de déploiement et de configuration. Une faible granularité [2] peut être obtenue avec l'approche *Action Semantics* d'UML [150] pour définir le corps des méthodes, mais ceci ne fait pas partie de nos objectifs. Le choix d'aborder le développement des systèmes informatiques avec une forte granularité est limitatif pour arriver à la génération complète des plates-formes cibles. Cependant, dans un premier temps, la compréhension et la proposition de solutions viables pour générer les artefacts logiciels avec une forte granularité est un challenge à surmonter afin de pouvoir ensuite aborder le sujet avec une faible granularité.

Dans ce chapitre, nous proposons une notation graphique avec l'utilisation conjointe d'OCL (*Object Constraint Language*) [148] pour spécifier des correspondances entre deux métamodèles. Nous utilisons cette notation graphique et OCL pour spécifier des correspondances entre un métamodèle source (dans notre étude de cas, UML ou EDOC) et un métamodèle cible (dans notre étude de cas, services Web,

[1] Forte granularité fait référence aux classes, interfaces et les méthodes d'un langage orienté objet.
[2] Faible granularité fait référence à aux instructions utilisées dans les corps de méthodes.

J2EE ou dotNET). La définition de transformations en ATL est ensuite créée manuellement[3] à partir de la spécification des correspondances. Par la suite, ces définitions de transformations sont appliquées à l'étude de cas. La figure 4.1 illustre les transformations qui seront présentées dans ce chapitre. Après avoir présenté notre approche, nous établissons des comparaisons entre les langages de modélisation UML et EDOC, et entre les plates-formes J2EE (Java et JWSDP) et dotNET (C# et le *framework* dotNET).

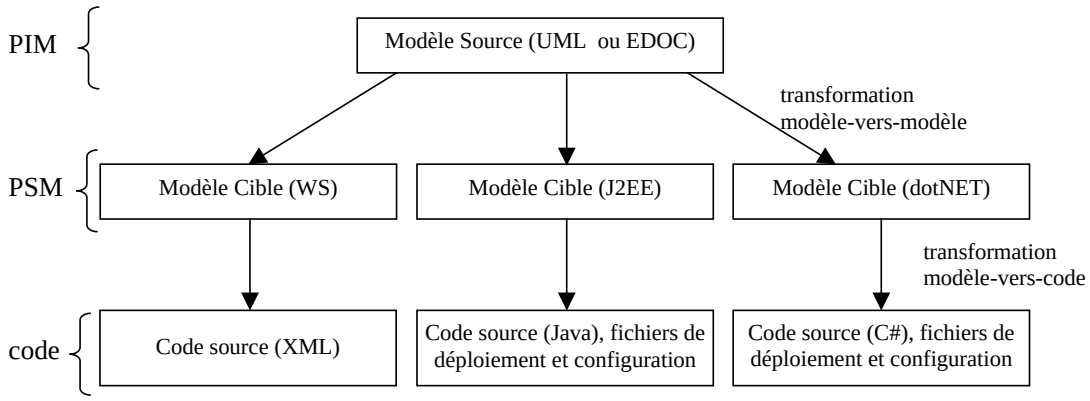


Figure 4.1 – Les transformations expérimentées : modèle-vers-modèle et modèle-vers-code

4.2 Notation graphique pour spécification de correspondances

Dans le cadre de MDA, l'utilisation d'une notation graphique pour spécifier des correspondances doit être intuitive, simple, et permettre la réutilisation de l'existant. Nous avons donc adopté la notation présentée sur la figure 4.2. Cette notation graphique réutilise la notation d'UML[4]. Le métamodèle source est localisé dans la partie gauche, le métamodèle cible dans la partie droite, et la correspondance (*mapping*) se situe entre les deux. Une ligne en pointillés et fléchée à gauche illustre une référence à un élément du métamodèle source. Une ligne en pointillés doublement fléchée à droite illustre une référence à un élément du métamodèle cible. Le cercle dénote un élément de correspondance identifié par un nom. Les lignes commencées par une extrémité en forme de losange représentent la notion de composition d'UML : un élément de correspondance peut être composé d'autres éléments de correspondance. Par conséquent, une correspondance est aussi un modèle qui est utilisé pour spécifier les inter-relations entre deux métamodèles.

Cette notation graphique met en évidence les inter-relations entre deux métamodèles, mais elle est insuffisante pour permettre d'exécuter des requêtes sur les métamodèles. Les requêtes sont importantes puisqu'elles permettent, par une navigation dans les métamodèles, de trouver quels sont les éléments équivalents ou similaires entre deux métamodèles [21][168]. Actuellement, OCL est une solution largement répandue pour spécifier des requêtes [27] [82] [102], ce qui justifie son utilisation dans nos travaux pour permettre la création de requêtes et de vues. Ainsi, chaque élément de correspondance (`Mapping Element`) contient une requête en OCL pour naviguer dans le métamodèle source et retourner l'élément source correspondant à l'élément du métamodèle cible. De plus OCL peut être utilisé pour spécifier

[3] Le chapitre suivant de cette thèse sera dédié à l'automatisation de la génération de définition de transformations.
[4] D'autres notations proposées dans [82] et [219] sont également basées sur la notation d'UML.

des contraintes entre les correspondances, par exemple faire le filtrage des éléments source avec une caractéristique particulière.

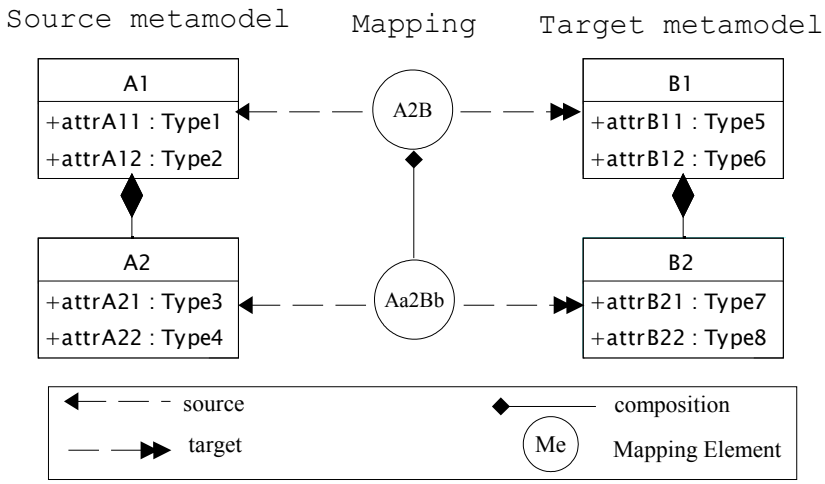


Figure 4.2 – Une proposition de notation graphique pour les correspondances

Les correspondances entre métamodèles ne sont pas toujours du type $1 : 1$ (un-vers-un), mais elles peuvent être $1 : n$, $n : 1$ et $m : n$ comme illustre la figure 4.3. Sur ce point, notre notation graphique peut être étendue pour prendre en charge la cardinalité des correspondances. La figure 4.3 illustre l'extension de la notation graphique, présentant les différentes catégories de correspondances selon la cardinalité des éléments de correspondance.

Une correspondance un-vers-un est caractérisée par un élément du métamodèle cible présentant une structure et une sémantique similaire à un élément du métamodèle source. Une correspondance un-vers-plusieurs est caractérisée par un ensemble non-unitaire et non-vide d'éléments d'un métamodèle cible contenant une structure et une sémantique similaire à un élément du métamodèle source. Une correspondance plusieurs-vers-un est caractérisée par un élément du métamodèle cible présentant une sémantique et une structure similaire à un ensemble non-unitaire et non-vide des éléments du métamodèle source. Ce dernier type de correspondance n'est pas directement réalisé avec des langages de transformation comme ATL, mais il peut être simulé en utilisant une règle de transformation qui prend un seul élément du métamodèle source et détermine les autres éléments en utilisant les relations entre eux. Une correspondance de type plusieurs-vers-plusieurs n'est pas présentée, puisqu'elle peut être simulée par les autres types.

4.2.1 De la spécification de correspondances à la génération de définition de transformations

L'exemple de la figure 4.4 illustre une application de notre démarche. La structure et la sémantique des deux métamodèles sont très similaires afin de faciliter leur présentation. Le métamodèle source (appelé `A`) est constitué des éléments `ElementModel`, `Package`, `Class` et `Attribute`. Le métamodèle cible (appelé `B`) est constitué des éléments `Element`, `NameSpace`, `Concept` et `Property` . Les éléments `Package`, `Class`, et `Attribute` sont les correspondants respectifs de `NameSpace`, `Concept` et `Property`. La correspondance comporte des éléments `P2NS`, `C2C` et `A2P`.

La composition de correspondances est une conséquence de la structure des métamodèles. Par exem-

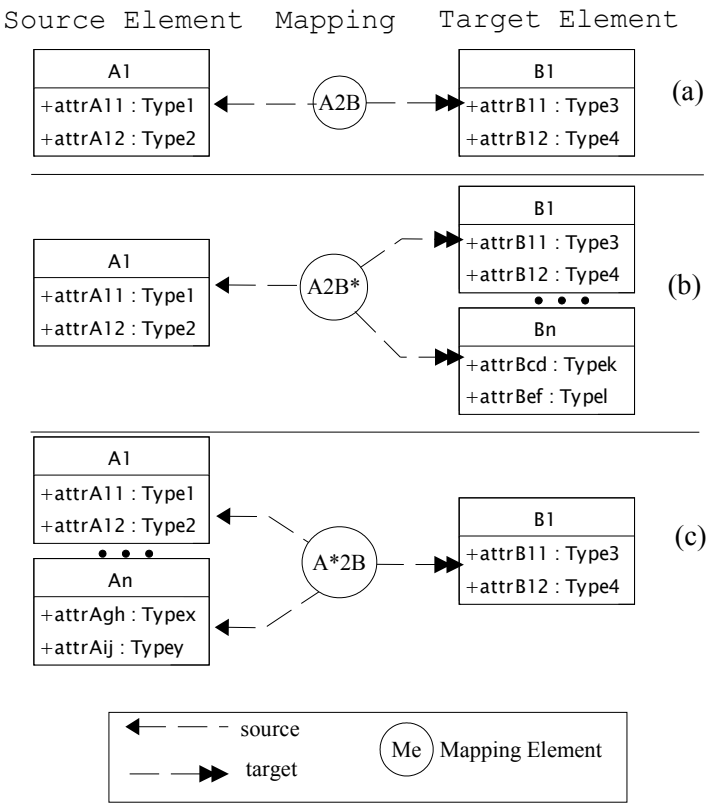


Figure 4.3 – Les types de correspondances : (a) un-vers-un, (b) un-vers-plusieurs, (c) plusieurs-vers-un

ple, un `Package` contient un élément, un élément peut être une `Class`, et une `Class` contient des éléments `Attributes`. Le métamodèle cible doit aussi avoir une structure et/ou une sémantique similaire au métamodèle source.

Indépendamment du langage de métamodélisation (par exemple MOF ou Ecore), les éléments d'un métamodèle pourront avoir des attributs et/ou des références. Les éléments avec des attributs et/ou des références vont aussi être impliqués dans une composition de correspondances : chaque correspondance pourra être constituée d'autres correspondances imbriquées pour spécifier les relations entre les attributs ou références qui appartiennent aux éléments sources et cibles. Ainsi, nous pourrons dire que la correspondance `P2NS` contient un élément de correspondance qui met en relation l'attribut `name` de `Package` (hérité de `ElementModel`) avec un autre élément `name` de `NameSpace` (hérité de `Element`). Remarquons que la correspondance doit aussi présumer que les deux attributs soient du même type, sinon une conversion plus complexe sera nécessaire. Le type de chaque attribut est spécifié dans le métamétamodèle, c.-à-d. le langage utilisé pour construire les métamodèles. Dans notre étude, nous considérons toujours que les métamodèles sont décrits avec un langage de métamodélisation unique (par exemple, MOF ou Ecore). L'utilisation de métamodèles différents va rendre la transformation finale plus complexe [35].

La définition de transformation peut être créée, par exemple, en utilisant ATL [32] ou YATL [161]. Dans notre étude, nous avons préféré utiliser ATL pour créer la définition de transformation car il présente des caractéristiques très intéressantes, comme le fait qu'il soit basé sur OCL. Ceci va nous permettre de passer plus facilement de la spécification de correspondances à la définition de transformation. En fait un modèle `a` conforme au métamodèle `A` peut être transformé en un autre modèle `b` conforme au métamodèle

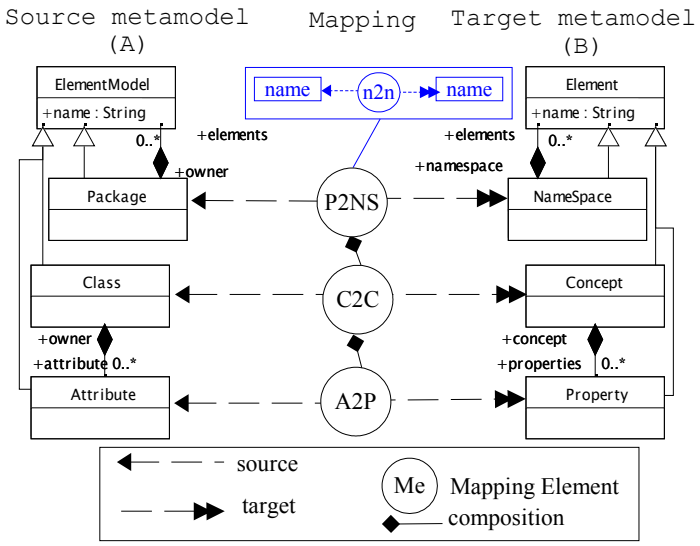


Figure 4.4 – Exemple de spécification de correspondances

B, si et seulement si le métamodèle A contient des similarités structurelles/sémantiques avec le métamodèle B. De plus, OCL devient un dénominateur commun entre la spécification de correspondances et la définition de transformation. La définition de transformation en ATL de notre exemple (figure 4.4) peut être générée manuellement comme suit (exemple 4.1) :

Exemple 4.1 – Définition de transformation en ATL

```
module  A2B;
create OUT :B from IN :A;

rule P2NS{
        from pck : A!Package
        to ns : B!NameSpace(
                name <- pck.name
        )
}

rule C2C{
        from c : A!Class
        to cp : B!Concept(
                name <- c.name,
                namespace <- owner
        )
}

rule A2P{
        from attr : A!Attribute
        to pt : B!Property(
                name <- attr.name,
                concept <- attr.owner
        )
}
```

Si la spécification de correspondances est simple, alors la requête en OCL se résume à l'utilisa-

tion de l'opérateur point '.' pour déterminer l'attribut ou la référence correspondant. Si la spécification de correspondances est plus complexe, une requête en OCL plus complexe est alors nécessaire, généralement avec l'utilisation d'opérations sur les collections (par exemple, `size()`, `includes()`) ou d'itérations sur les collections (par exemple, `select()`, `collect()`, `forAll()`) ou d'expressions conditionnelles (`if-then-else-endif`) ou de requêtes spéciales (`allInstances()`, `oclIsKindOf()`, `oclIsTypeOf()`) [148]. L'exemple 4.2 illustre une expression OCL qui contient des itérations sur collections (`select` et `collect`).

Exemple 4.2 – Exemple d'utilisation d'itérations sur les collections

```
implements <- c.clientDependency ->
        select(e|e.stereotype->
                exists(e|e.name='realize'))->collect(e|e.supplier)
```

4.3 UML vers Services Web

Les technologies de Services Web les plus importantes de notre étude sont WSDL et BPEL. D'une part, WSDL permet la modélisation des aspects statiques d'un service, et d'autre part BPEL permet la modélisation des aspects dynamiques d'un service à partir de la composition d'autres services.

Dans cette section, nous présentons d'abord la spécification des correspondances entre les diagrammes de classes d'UML et WSDL, puis entre les diagrammes d'activités d'UML et BPEL. Nous exposons ensuite la définition des transformations en ATL. Une fois que le modèle conforme à la plate-forme cible est réalisé, nous présentons l'utilisation d'ATL pour générer le code source (le document WSDL et le document BPEL).

4.3.1 Spécification de correspondances entre UML et WSDL

Une spécification de correspondances nécessite d'abord les deux métamodèles : source et cible. Dans notre exemple, le métamodèle source est UML et les métamodèles cibles sont WSDL et BPEL, présentés dans le chapitre précédent (section 3.6). Nous utilisons ces métamodèles pour spécifier les correspondances.

La figure 4.5 montre une spécification de correspondances entre UML et WSDL. L'élément de correspondance `P2D` met en relation l'élément `Package` d'UML avec l'élément `Definition` de WSDL. L'élément de correspondance `C2S` met en relation l'élément `Class` d'UML avec les éléments `Service`, `Port`, `Binding` et `PortType` de WSDL. L'élément de correspondance `O2O` met en relation l'élément `Operation` d'UML avec les éléments `Operation` et `Message` de WSDL. L'élément de correspondance `P2Part` met en relation l'élément `Parameter` d'UML avec l'élément `Part` de WSDL. L'élément de correspondance `Dt2T` met en relation l'élément `DataType` d'UML avec l'élément `Type` de WSDL.

Chacune de ces correspondances est constituée d'autres correspondances qui ne sont pas montrées par la figure 4.5, afin de simplifier leur présentation. Par exemple, l'élément de correspondance `P2D` est constitué des éléments de correspondances composite `n2n` et `n2t` illustrés par la figure 4.6. Selon cette figure, l'élément `name` de `Package` est mis en relation avec l'élément `name` et l'élément `targetNameSpace` de `Definition`. A priori, un seul élément de correspondance un-vers-plusieurs serait nécessaire pour mettre en relation l'attribut `name` de l'élément source avec l'attribut `name` et `target-`

`NameSpace` de l'élément cible. Cependant, ceci n'est pas nécessaire étant donné qu'un seul attribut[5] `name` est contenu dans un `Package`. Ainsi, nous pouvons considérer les correspondances un-vers-plusieurs comme plusieurs correspondances un-vers-un.

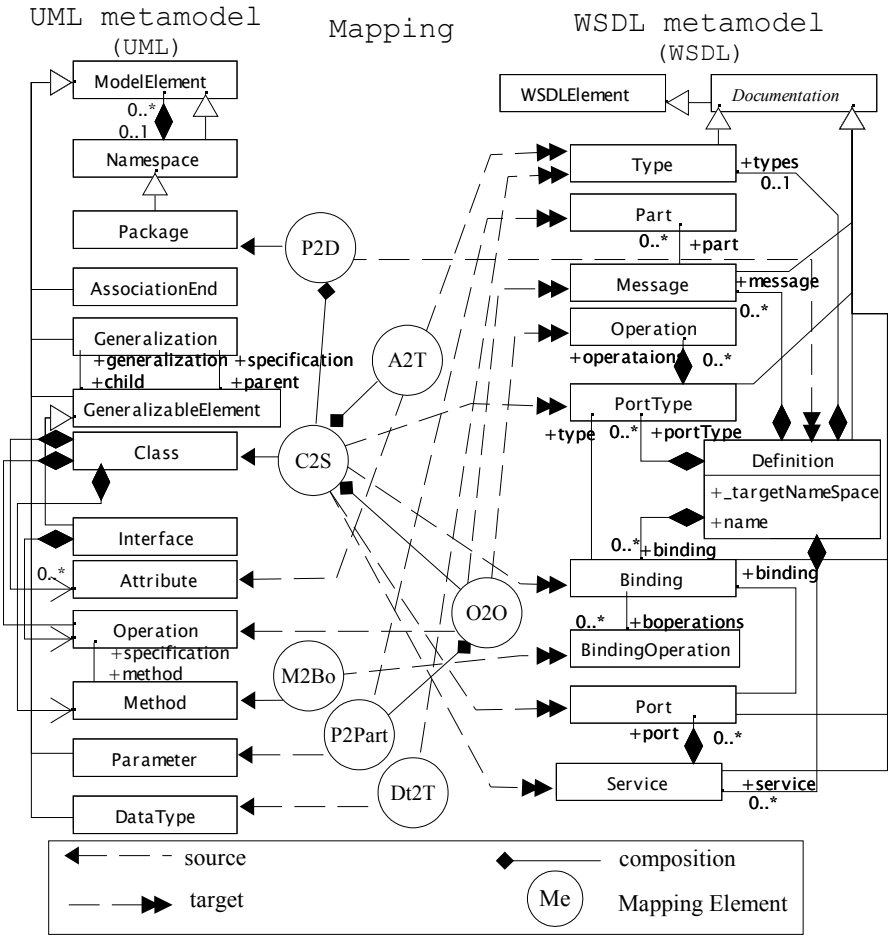


Figure 4.5 – Une spécification des correspondances entre UML et WSDL

La correspondance `n2t` est une correspondance complexe : une expression OCL est par conséquent nécessaire. Cette correspondance met en relation l'attribut `name` d'UML avec l'attribut `targetNameSpace` de WSDL par le biais de l'expression OCL `'urn://'+ @.name + '.wsdl'`. Ici nous utilisons le caractère '`@`' pour faire référence à l'élément source.

Pour réaliser la correspondance, il faut aussi faire l'équivalence entre les types de données permis dans chaque métamodèle. La correspondance `Dt2T` utilise l'équivalence entre les types de données de UML et de WSDL. Ainsi, le tableau 4.1 contient une équivalence entre les types de données d'UML et de WSDL.

[5]Les termes méta-attribute, méta-classe, méta-référence seraient plus corrects pour nommer les éléments d'un métamodèle, mais nous préférons simplifier le langage.

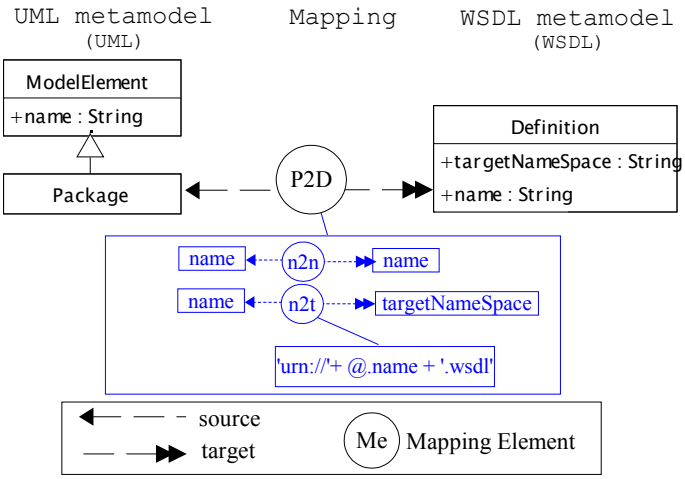


Figure 4.6 – Vue détaillée de l'élément de correspondance P2D

UML (DataType)	**WSDL (PrimitiveType)**
Float	xsd:float
Byte	xsd:byte
Double	xsd:double
Integer	xsd:int
Long	xsd:long
Short	xsd:short
String	xsd:string
Boolean	xsd:boolean

Table 4.1 – La correspondance entre les types d'UML et WSDL (Schéma)

4.3.2 Définition de transformations entre UML et WSDL

La définition de transformations en ATL pour réaliser la transformation d'un modèle UML en un modèle WSDL doit être basée sur la spécification de correspondances présentée précédemment. Partant de ce fait, l'exemple 4.3 montre un fragment du code source en ATL qui est créé manuellement à partir de la spécification de correspondances.

Exemple 4.3 – La définition de transformation d'UML en WSDL (fragment)

```
-- === file Transf_uml2wsdl.atl ===
--
module UML2WSDL;

create OUT : WSDL  from IN : UML;

--
-- Rules
--
-- UML-Package is mapped in the
-- Definition through the rule P2D:

```

```
rule P2D{
        from pck : UML!Package(pck.name <> 'java' and pck.name <> 'lang')
        to df : WSDL!Definition(
                name <- 'Service_' + pck.name,
                _targetNameSpace <- 'urn://'+ pck.name + '.wsdl',
                type <- t
        ),
        t : WSDL!Type(
        ***
        )
}

rule C2S{
        from c : UML!Class
        to s : WSDL!Service(
                name <- 'Service' + c.name,
                owner <- c.namespace,
                port <- pt
        ),
        pt : WSDL!Port(
                name <- c.name + 'Port',
                binding <- bd,
                documentation <- '\t\t<soap:address location="'+ '
                    REPLACE_WITH_ACTUAL_URL' +'"/>\n'
        ),
        bd : WSDL!Binding(
                name <- c.name + 'Binding',
                owner <- c.namespace,
                type <- pType,
                boperations <- c.feature ->select (e|e.oclIsTypeOf(UML!Method)),
                soapBinding <- soapB
        ),
        soapB : WSDL!SoapBinding(
                transport <- 'http://schemas.xmlsoap.org/soap/http',
                style <- 'rpc'
        ),
        pType: WSDL!PortType(
                name <- c.name,
                owner <- c.namespace,
                binding <- bd,
                operations  <- c.feature ->select (e|e.oclIsTypeOf(UML!Operation))
        )
}
***
```

Selon notre démarche, l'élément de correspondance `P2D` (figure 4.5) est utilisé pour générer la règle de transformation en ATL `P2D` (exemple 4.3), l'élément de correspondance `C2S` est utilisé pour générer la règle de transformation en ATL `C2S`, et ainsi de suite. L'élément de correspondance `C2S` est de type un-vers-plusieurs. Ceci est traduit en ATL par une règle qui prend un élément et en génère plusieurs.

Selon la figure 4.1 (page 122), une première transformation de modèle-vers-modèle était nécessaire pour faire passer les informations d'un modèle UML vers un modèle WSDL. Ensuite, une transformation de modèle WSDL vers le code WSDL, c.-à-d. document WSDL, est nécessaire pour finaliser le cycle de transformations et obtenir la plate-forme finale. L'exemple 4.4 présente la définition de transformation en ATL qui utilise une requête pour déclencher la transformation modèle-vers-code, et l'exemple 4.5 pré-

sente les *helpers* [32] pour naviguer et extraire les informations du modèle WSDL et écrire le document WSDL.

Exemple 4.4 – La définition de transformations de modèle WSDL en code WSDL

```
-- === File : WSDL2SourceCode_query.atl
--
query WSDL2SourceCode_query = WSDL!Definition.allInstances()->
        collect(x | x.toString().writeTo('C:/SourceCode/WSDL/' + x.name.replaceAll('
            .', '/') + '/' + x.name + '.wsdl'));

uses WSDL2SourceCode;
```

L'exemple 4.4 contient une requête utilisée pour déclencher la transformation modèle-vers-code. L'opération `allInstances()` retourne une collection constituée de toutes les instances de l'élément `WSDL!Definition` présentes dans le modèle WSDL. L'opération sur les collections, appelée `collect()`, nous permet ensuite de récupérer chaque élément de cette collection (représenté par `x`), et d'invoquer l'opération `toString()` sur chacun d'eux. Cette opération est un *helper* en ATL défini dans la bibliothèque `WSDL2SourceCode` (présentée par l'exemple 4.5). Après l'exécution de cet *helper*, la chaîne de caractères est écrite dans un fichier « `*.wsdl` » par le biais de l'opération `writeTo`. L'*helper* `toString()` défini dans le contexte de `WSDL!Definition` enchaîne aussi d'autres appels à d'autres *helpers* définis dans le contexte de `Service`, `Port`, `PortType` et ainsi de suite.

Exemple 4.5 – La bibliothèque WSDL2SourceCode en ATL (fragment)

```
-- === File : WSDL2SourceCode.atl
library WSDL2SourceCode;

helper context WSDL!Definition def: toString() : String =
        '<?xml version="1.0" encoding="UTF-8"?>'+
        '<definitions name="'+ self.name +'" targetNamespace="'+self.
            _targetNameSpace+'">\n' +
                self.printType() +
                self.printMessage()+
                self.printPortType()+
                self.printBinding()+
                self.printService()+
        '</definitions>\n';
***
helper context WSDL!Message def: toString():String =
        '<message name="'+self.name+'">\n' +
         self.part -> iterate(i; acc:String='' | acc+ i.toString())+
        '</message>\n';
***
helper context WSDL!PortType def: toString(): String =
        '<portType name="'+self.name+'">\n' +
                self.operations-> iterate(i; acc:String='' | acc+ i.toString())+
        '</portType>\n';
***
helper context WSDL!Port def: toString(): String =
        '\t<port name="'+self.name+'" binding="'+self.binding.name+'">\n'+
           self.documentation +
        '\t</port>\n';

helper context WSDL!Service def: toString(): String =
        '<service name="'+ self.name+'">\n'+
```

```
            self.port -> iterate(i; acc:String='' | acc+ i.toString()) +
32          '</service>\n';

34  helper context WSDL!Definition def: printService(): String =
          self.service -> iterate(i; acc:String='' | acc+ i.toString());
36  ***
```

4.3.3 Spécification de correspondances entre UML et BPEL

Un diagramme d'activité est un cas particulier de diagramme d'état dans lequel les états sont des actions ou des sous-actions, et les transitions sont déclenchées par l'accomplissement d'actions. Un diagramme d'activité est attaché à un classificateur tel qu'un cas d'utilisation, un paquetage (*package*) ou une classe.

BPEL est basé sur un modèle de composant à granularité fine constitué d'activités qui peuvent être de deux types : basique ou structuré. Les activités basiques représentent l'invocation d'une opération réalisée par un service Web, par exemple `invoke`, `receive`, `reply` et `assign`. Les activités structurées sont basées sur l'approche des hiérarchies d'activités dans laquelle un processus est spécifié par le raffinement progressif d'une activité dans un arbre d'activités. Les nœuds représentent les actions à exécuter, et les nœuds intermédiaires définissent une contrainte entre les activités secondaires [8].

Une spécification de correspondances entre le diagramme d'activité d'UML et BPEL n'est pas une tâche simple. Ils sont tous les deux utilisés pour représenter des processus, mais ils utilisent des approches différentes : celle du diagramme d'activité est basée sur les automates d'états-finis (dans la forme d'état-transition) alors que celle de BPEL est basée sur une hiérarchie d'activités. La spécification de correspondances et la définition de transformations entre UML et BPEL doivent considérer leurs caractéristiques respectives. D'une part, un diagramme d'activités n'est pas structuré et, d'autre part, BPEL est structuré. Ainsi, nous devons considérer qu'un diagramme d'activités peut être mis en correspondance avec un processus BPEL seulement si le diagramme d'activités est réalisé de manière structurée. Dans ces conditions, pour chaque `Fork` [6] il doit y avoir un `Join`[7]. De plus, chaque processus secondaire créé par un même `Fork` doit se synchroniser par le biais d'un même `Join` avant de rendre la main au processus principal. Pour simplifier la transformation, nous considérons que les diagrammes d'activités seront créés de manière structurée. La figure 4.7 illustre la différence entre un diagramme d'activité structuré et un diagramme d'activité non structuré.

Nous utilisons des valeurs marquées pour étendre UML afin de diminuer la distance structurelle et sémantique entre UML et BPEL et de permettre la spécification de correspondance entre une action d'un diagramme d'activité d'UML et une activité `Invoke`, `Assign`, `Receive` ou `Reply` de BPEL, tel qu'introduit dans la section 3.4 (page 106). Nous utilisons aussi des valeurs marquées pour lier une action à une opération et à une classe. Le tableau 4.2 présente les valeurs marquées utilisées.

La figure 4.8 montre une spécification de correspondances entre le métamodèle d'UML (diagramme d'activité) et celui de BPEL. L'élément de correspondance `Ag2P` met en relation l'élément `Activity-Graph` d'UML avec l'élément `Process` de BPEL. L'élément de correspondance `O2V` met en relation l'élément `Operation` d'UML avec l'élément `Variable` de BPEL. L'élément de correspondance `C2P` met en relation l'élément `Class` d'UML avec l'élément `PartnerLink` de BPEL. L'élément de correspondance `A2I` met en relation l'élément `ActionState` (avec une valeur marquée `activity=invoke`)

[6] `Fork` permet la création de plusieurs processus enfants qui sont exécutés en parallèle.

[7] `Join` permet la synchronisation entre plusieurs processus enfants avant de la continuation de l'exécution du processus principal.

d'UML avec l'élément `Invoke` de BPEL. L'élément de correspondance `A2Rp` met en relation l'élément `ActionState` (avec une valeur marquée `activity=reply`) d'UML avec l'élément `Reply` de BPEL. L'élément de correspondance `A2A` met en relation l'élément `ActionState` (avec une valeur marquée `activity=assign`) d'UML avec l'élément `Assign` de BPEL. L'élément de correspondance `P2F` met en relation l'élément `Pseudostate` (avec `kind=pk_fork`) d'UML avec l'élément `Flow` de BPEL. L'élément de correspondance `P2Sw` met en relation l'élément `Pseudostate` (avec `kind=pk_choice`) d'UML avec l'élément `Switch` de BPEL.

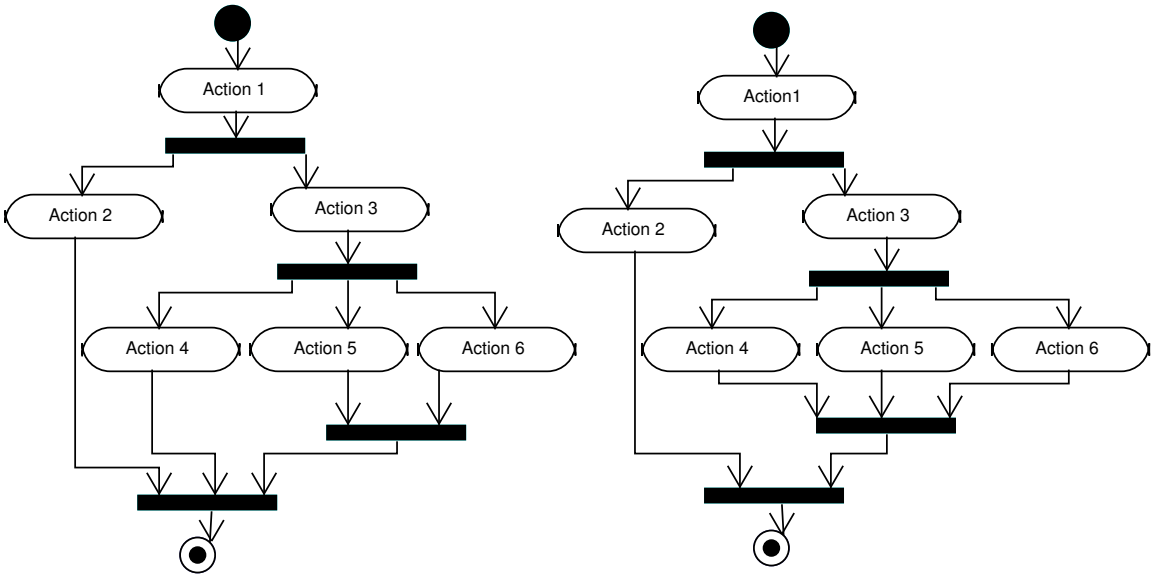


Figure 4.7 – (a) un diagramme d'activité non-structuré, (b) un diagramme d'activité structuré

Nom	**Valeur**
activity	invoke, receive, reply, assign
class	une classe liée à une action ou qui contient une opération participant à une affectation
operation	une opération liée à une action
fromOperation	une opération d'entrée qui participe à une affectation
toClass	une classe qui contient une opération participant à une affectation
toOperation	une opération qui participe à une affectation

Table 4.2 – Les valeurs marquées permettant d'étendre un diagramme d'activité.

Une des conséquences de la différence structurelle et sémantique entre le diagramme d'activité d'UML et BPEL est le besoin d'utiliser massivement les requêtes en OCL. Ces requêtes permettent la navigation dans le métamodèle d'UML afin d'obtenir l'information nécessaire correspondant au métamodèle de BPEL. Par exemple, dans la correspondance `A2R`, l'expression OCL suivante est nécessaire pour récupérer le `partnerLink` associé à l'activité `receive` à partir d'un `ActionState` :

Exemple 4.6 – L'expression OCL associée à la correspondance « A2R »

```
if @.taggedValue->size() > 1 then
  if (@.taggedValue->select(e|e.type.name='class')->size() > 0 ) then
     @.taggedValue->select(e|e.type.name='class')->first().getDataValue()
  else
     'null'
  endif
else
  'null'
endif
```

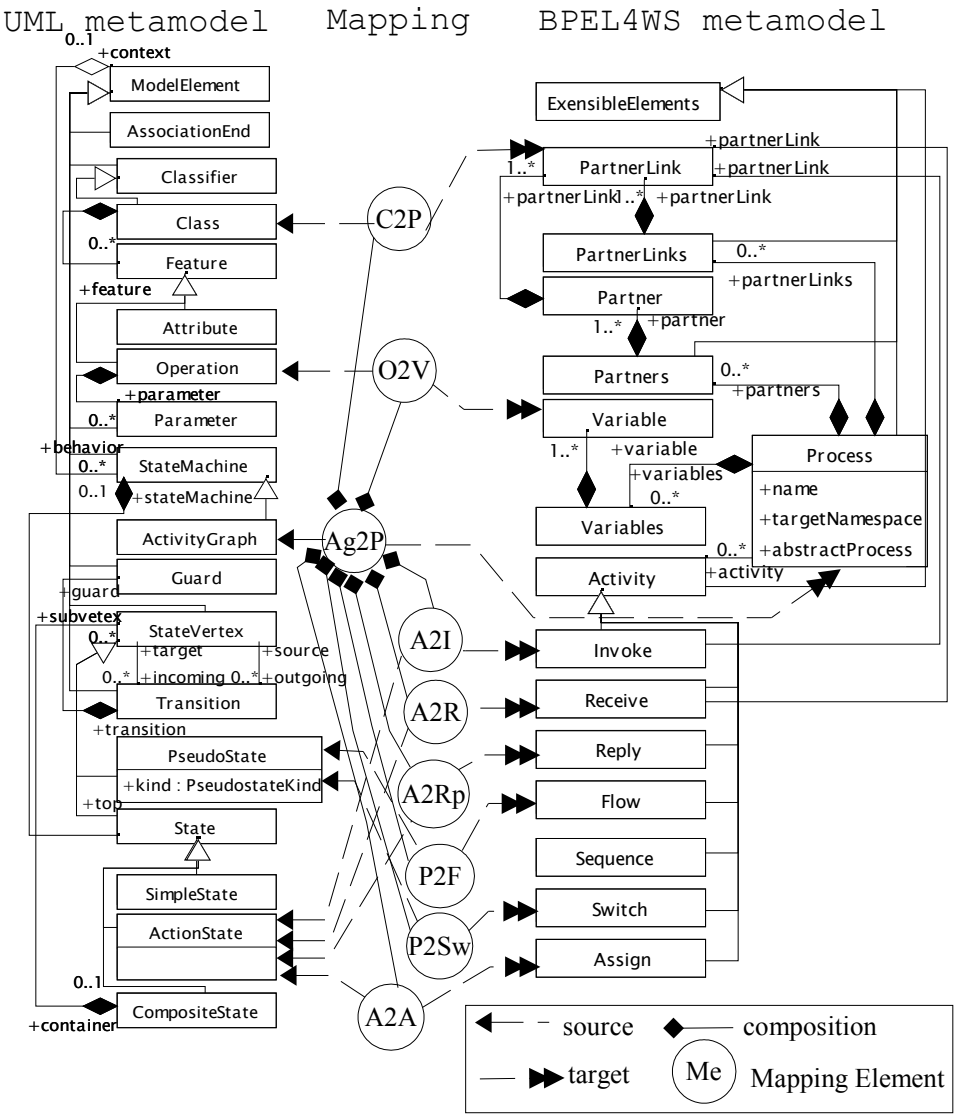


Figure 4.8 – Une spécification de correspondances entre UML et BPEL

4.3.4 Définition de transformations entre UML et BPEL

La définition de la transformation en ATL d'un diagramme d'activité UML en un modèle BPEL doit être conforme à la spécification de correspondances présentée précédemment. Partant de ce fait, l'exemple 4.7 montre un fragment du code source en ATL qui est généré manuellement à partir de la spécification de correspondances.

Exemple 4.7 – La définition de transformation d'UML en BPEL (fragment)

```
-- File: UML2BPEL.atl
module UML2BPEL;
create OUT : BPEL from IN : UML;
uses strings;

rule Ag2P{
        from ag : UML!ActivityGraph (ag."context".oclIsKindOf(UML!Class))
        to p : BPEL!Process(
                name <- ag.namespace.name,
                targetNamespace <- 'http://' + ag.namespace.name + '.org/',
                abstractProcess <- false,
                partnerLinks <- pt,
                variables <- var,
                activity <- seq
                ),
                pt : BPEL!PartnerLinks(
                        partnerLink <- Sequence{myPartnerLink} ->union( ag."context"
                            .getServiceClasses() )
                ),
                myPartnerLink : BPEL!PartnerLink(
                        name <- ag."context".name,
                        partnerLinkType <- pLT,
                        myRole <- 'ITF_'+ ag."context".name + 'Provider',
                        partnerRole <- ''
                ),
                pLT: BPEL!PartnerLinkType(
                        name <- 'ITF_'+ ag."context".name + 'Link',
                        role <- rL
                ),
                rL:BPEL!Role(
                        name <- 'ITF_'+ ag."context".name + 'Provider',
                        portType <- 'ITF_'+ ag."context".name
                ),
                var: BPEL!Variables(
                        variable <- ag."context".getAllOperations()->union( ag."
                            context".getAllMethods() )
                ),
                seq : BPEL!"Sequence"(
                        activity <- ag.top.subvertex->select(e|e.oclIsKindOf(UML!
                            Pseudostate))->select(e|e.kind=#pk_initial)->first().
                            outgoing->select(x|x.oclIsKindOf(UML!Transition))->first
                            ().getStateVertex(Sequence{})
                )
}

rule C2P{
        from class: UML!Class
        to ptLink: BPEL!PartnerLink(
```

```
44                  name <- class.name,
                    partnerLinkType <- pLT,
46                  partnerRole <- 'ITF_'+class.name + 'Provider'
            ),
48          pLT: BPEL!PartnerLinkType(
                    name <- 'ITF_'+class.name + 'Link',
50                  role <- rL
            ),
52          rL:BPEL!Role(
                    name <- 'ITF_'+class.name + 'Provider',
54                  portType <- 'ITF_'+class.name
            )
56  }
    ***
```

Selon notre démarche, l'élément de correspondance `Ag2P` (figure 4.8) est utilisé pour générer la règle de transformation `Ag2P` en ATL (exemple 4.7), l'élément de correspondance `C2P` est utilisé pour générer la règle de transformation `C2P` en ATL, et ainsi de suite.

Selon la figure 4.1 (page 122), une première transformation modèle-vers-modèle était nécessaire pour faire passer les informations d'un modèle UML vers un modèle BPEL. Une transformation de modèle BPEL vers le code BPEL (document BPEL) est ensuite nécessaire pour finaliser le cycle de transformation et obtenir la plate-forme finale. L'exemple 4.8 présente la définition de la transformation écrite en ATL et qui utilise une requête pour déclencher la transformation modèle-vers-code, et l'exemple 4.9 présente les *helpers* permettant de naviguer et extraire les informations dans le modèle BPEL, et écrire le document BPEL.

Exemple 4.8 – La définition de transformation de modèle BPEL en code BPEL

```
    -- File: BPEL2SC_query.atl
2   query BPEL2SC_query =
            BPEL!Process.allInstances()->
4                   select(e | e.oclIsTypeOf(BPEL!Process))->
                    collect(x | x.toString().writeTo('C:/SourceCode/BPEL/' + x.name +'.
                        bpel'));
6   uses BPEL2SC;
```

L'exemple 4.8 contient une requête utilisée pour déclencher la transformation modèle-vers-code. L'opération `allInstances()` retourne une collection constituée de toutes les instances de l'élément `BPEL!Process` présentes dans le modèle BPEL. Puis une opération sur les collections appelé `collect()` nous permet de récupérer chaque élément de cette collection (représenté par `x`), et d'appeler ensuite l'opération `toString()` sur chacun d'eux. Cette opération est un *helper* en ATL défini dans la bibliothèque `BPEL2SC` (présentée par l'exemple 4.9). Après l'exécution de cet *helper*, la chaîne de caractères est écrite dans un fichier « *.bpel » par le biais de l'opération `writeTo`. L'*helper* `toString()` défini dans le contexte de `BPEL!Process` enchaîne aussi d'autres appels à d'autres *helpers* qui sont définis dans le contexte de `PartnerLink`, `Variable`, `Receive` et ainsi de suite.

Exemple 4.9 – La bibliothèque BPEL2SC en ATL (fragment)

```
    -- File: BPEL2SC.atl
2   library BPEL2SC;

4   helper context BPEL!Process def: toString() : String =
            '<process name="' + self.name +'" targetNamespace="'+ self.targetNamespace +
                '" suppressJoinFailure="'+ self.supressJoinFailure.toString() +'">\n' +
```

```
6           self.partnerLinks-> iterate(x; acc: String='' | acc + x.toString() ) +
            self.variables-> iterate(x; acc: String='' | acc + x.toString() ) +
8           self.activity-> iterate(x; acc: String='' | acc + x.toString('') ) +
        '</process>';
10  ***
    helper context BPEL!PartnerLinks def: toString() : String =
12      '\t<partnerLinks>\n' +
         self.partnerLink-> iterate(x; acc: String='' | acc + x.toString('\t')   ) +
14      '\t</partnerLinks>\n';

16  helper context BPEL!PartnerLink def: toString(tab:String) : String =
        if self.partnerRole='' then
18      tab+'\t<partnerLink name="'+ self.name+'" partnerLinkType="'+self.
            partnerLinkType.name+'" myRole="'+self.myRole+'"/> \n'
        else
20      tab+'\t<partnerLink name="'+ self.name+'" partnerLinkType="'+self.
            partnerLinkType.name+'" partnerRole="'+self.partnerRole+'"/> \n'
        endif;
22
    helper context BPEL!Variables def: toString() : String =
24      '\t<variables>\n' +
         self.variable -> iterate(x; acc: String='' | acc + x.toString('\t')   ) +
26      '\t</variables>\n';
    ***
```

4.4 UML vers J2EE

Dans cette section, nous utilisons notre démarche MDA pour générer le code Java plus JWSDP à partir d'un diagramme de classes UML.

La génération de document WSDL à partir d'UML est plus simple lorsqu'il y a seulement des métamodèles. Cependant, la transformation devient plus complexe dès que des bibliothèques [8] doivent être utilisées. JWSDP (*Java Web Services Developer Pack*) est constitué d'une bibliothèque et de fichiers de configuration et de déploiement. La bibliothèque est construite avec du code Java, et le métamodèle sert à créer des modèles dont les informations sont nécessaires pour créer les fichiers de configuration et de déploiement. Étant donné que la bibliothèque est construite à partir de code Java, nous avons décidé de la considérer comme un modèle. Cependant, dans certains cas, nous avons remarqué qu'une partie des bibliothèques est mieux représentée au niveau des métamodèles. Par exemple, dans le métamodèle de C# (section 3.8, page 118), nous avons décidé d'inclure l'attribut (représenté par l'élément `Attribute`) dans le métamodèle, puisqu'il a une influence très forte dans le langage et dans le comportement d'exécution des programmes. De plus, l'attribut est aussi présent dans la bibliothèque et grammaire EBNF (*Extended Backus-Naur Form*) de C# [124]. Un autre cas intéressant est celui du métamodèle de Java proposé par netBeans (http://java.netbeans.org/models/java/java-model.html) contenant un élément `Exception`, qui est présent dans la bibliothèque de Java ainsi que dans sa grammaire EBNF. Ainsi, une partie de la bibliothèque peut se situer au niveau du métamodèle, du modèle ou des deux. Ceci peut paraître contradictoire ou superflu mais, au cours des nos expérimentations, nous avons remarqué qu'il s'agit d'une caractéristique des plates-formes qui doit être conservée dans l'architecture à quatre niveaux. En fait, une approche dirigée par les modèles ne doit pas changer les plates-formes mais en préserver les

[8] Bibliothèques et APIs (*Applications Programming Interface*) peuvent aussi être considérées comme des modèles.

caractéristiques.

En considérant l'existence des bibliothèques, nous proposons la création d'entrepôts de modèles de bibliothèques. Ainsi un modèle en Java peut réutiliser les modèles des bibliothèques pour détailler avec plus de précision la plate-forme cible. Ceci est nécessaire puisque les modèles de ce niveau sont dépendants de la plate-forme, laquelle est composée non seulement du langage mais aussi des bibliothèques. Étant donné que les outils que nous utilisons ne sont pas capables de prendre en compte de tels entrepôts, nous avons choisi d'indiquer les bibliothèques nécessaires à la spécification de correspondances et de créer des fragments de ces bibliothèques dans la définition de transformations.

Un autre aspect important d'une approche dirigée par les modèles est la forme sous laquelle les bibliothèques doivent être utilisées. Dans le chapitre précédent, nous avons proposé un *template* et un métamodèle pour JWSDP. Le *template* guide l'utilisation du modèle de la bibliothèque de JWSDP à respecter dans la spécification de correspondances et, par conséquent, dans la définition de transformations. L'ajout des entités (dans ce cas, les éléments d'un modèle de bibliothèque) dans la spécification de correspondances démontre qu'une correspondance entre métamodèles n'est pas seulement de type *sound* et *exact*, mais aussi *complete* [201]. Ainsi, nous pouvons être amenés à introduire de nouveaux éléments pour ajouter des informations nécessaires au modèle spécifique de la plate-forme, mais absentes dans le modèle indépendant de la plate-forme. Ceci est naturel car le modèle indépendant de la plate-forme contient seulement la logique métier qui est transmise à un modèle dépendant de la plate-forme, ce dernier devant être enrichi des caractéristiques de la plate-forme. Par exemple, une classe UML qui implémente un service Web doit correspondre à une classe et une interface en Java, comme cela a été présenté par le *template* de JWSDP (section 3.7, page 117). De plus, les méthodes en UML doivent correspondre à de méthodes en Java qui doivent retourner une `RemoteException` en cas d'erreur.

Cet exemple montre les différences qui existent entre la gestion de modèles en base de données [21] [201] et une approche MDA. Pour les bases de données, a priori, il suffit d'utiliser des métamodèles (ou schémas), leurs aspects statiques et leur manipulation, alors qu'en MDA il faut travailler avec les métamodèles, les aspects statiques, les aspects comportementaux, les modèles de bibliothèques et les *templates*. Nous nous sommes inspirés dans une partie de nos travaux de la théorie de gestion de modèles des bases de données et nous avons vérifié qu'elle peut être adaptée à l'approche MDA.

La création d'une spécification de correspondances entre UML et J2EE peut être effectuée en une seule étape ou en étapes progressives. Dans le premier cas, nous pouvons utiliser le *template* de JWSDP dès la transformation d'UML en Java. Dans le second cas, nous pouvons tout d'abord spécifier les correspondances entre UML et Java, puis entre Java et le *template* de JWSDP. Dans les deux cas, la spécification de correspondances entre le métamodèle d'UML et celui de JWSDP doit être faite, puis les fichiers de configuration et de déploiement peuvent alors être générés. Il faut rappeler que nous avons créé la plate-forme JWSDP composée de deux parties, un *template* et un métamodèle. Le *template* est conforme au métamodèle de Java, et le métamodèle de JWSDP contient la description des fichiers de déploiement. Nous montrons par la suite l'utilisation des étapes progressives pour la transformation d'UML en J2EE et, d'une seule étape dans le cas de la transformation d'UML en dotNET.

4.4.1 Spécification de correspondances

La figure 4.9 présente une spécification de correspondances entre le métamodèle d'UML et celui de Java. La correspondance `P2P` met en relation le `Package` d'UML avec le `JPackage` de Java, `C2Jc` met en relation le `Class` d'UML avec le `JClass` de Java, `A2F` met en relation l'élément `Attribute` d'UML avec l'élément `JField`, `OM2M` met en relation l'`Operation` et la `Method` d'UML avec la `JMe-`

thod de Java, et ainsi de suite. La correspondance OM2M est du type plusieurs-vers-un. Ceci n'est pas traduit directement dans un langage de transformation comme ATL, mais peut être traduit par une règle de transformation du type un-vers-un, et les informations d'un des éléments peuvent être obtenues grâce à la navigation, par le biais de l'association specification <-> method.

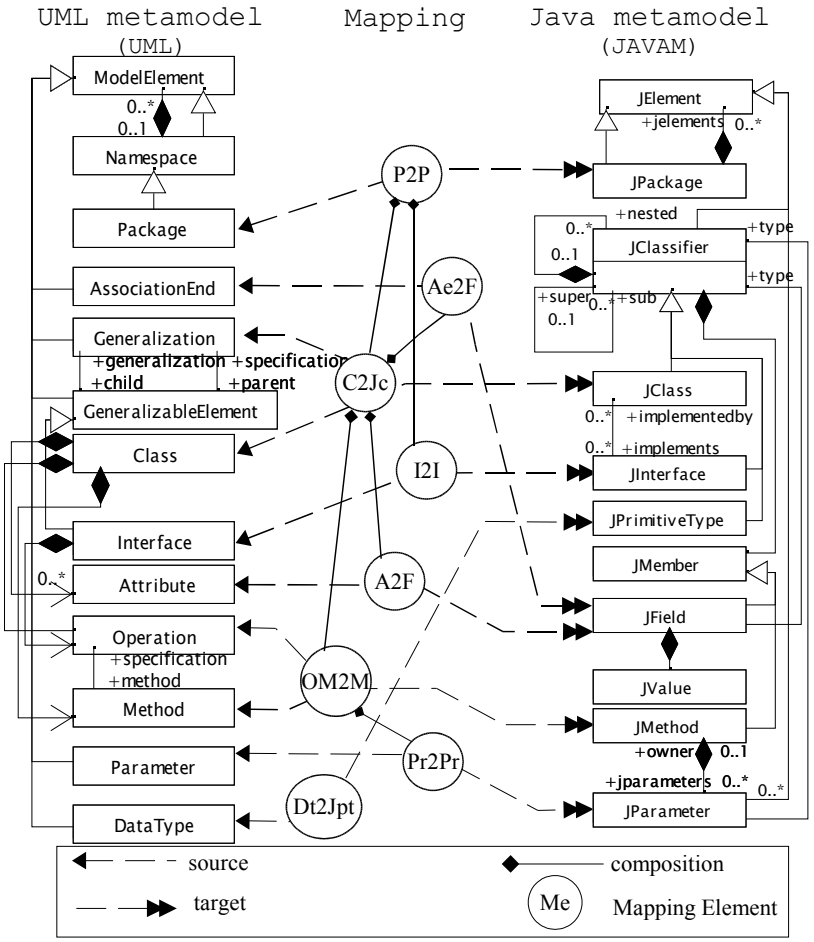


Figure 4.9 – Une spécification de correspondances entre UML et Java

Nous pouvons remarquer que la composition de correspondances suit aussi la structure des deux métamodèles. Ainsi, la correspondance P2P est composée de C2J et I2I. La correspondance C2J est composée d'OM2M et A2F, et ainsi de suite. Cette composition signifie qu'une correspondance appelle une autre correspondance pour déterminer l'un des attributs ou l'une des références. Cet appel peut être fait de deux façons : par exemple, C2J peut appeler A2F par la mise en relation de feature de type Attribute avec jmember de type JField, ou bien l'owner d'Attribute peut être mis en relation avec l'owner de JField. La première solution est plus complexe car il faut utiliser une requête OCL pour sélectionner le Feature de type Attribute d'UML qui correspond au JField de Java. La deuxième solution est plus simple puisqu'elle demande seulement la mise en correspondance des éléments owner d'Attribute d'UML et owner de JField de Java.

Pour réaliser la correspondance, il faut aussi faire l'équivalence entres les types de données permis dans chaque métamodèle. La correspondance Dt2Jpt utilise l'équivalence entre les types de données

de UML et de Java. Ainsi, le tableau 4.3 contient une équivalence entre les types de données d'UML et Java.

UML (DataType)	**Java (PrimitiveType)**
Float	float
Byte	byte
Double	double
Integer	int
Long	long
Short	short
String	String
Boolean	boolean

Table 4.3 – Une correspondance entre les types d'UML et Java

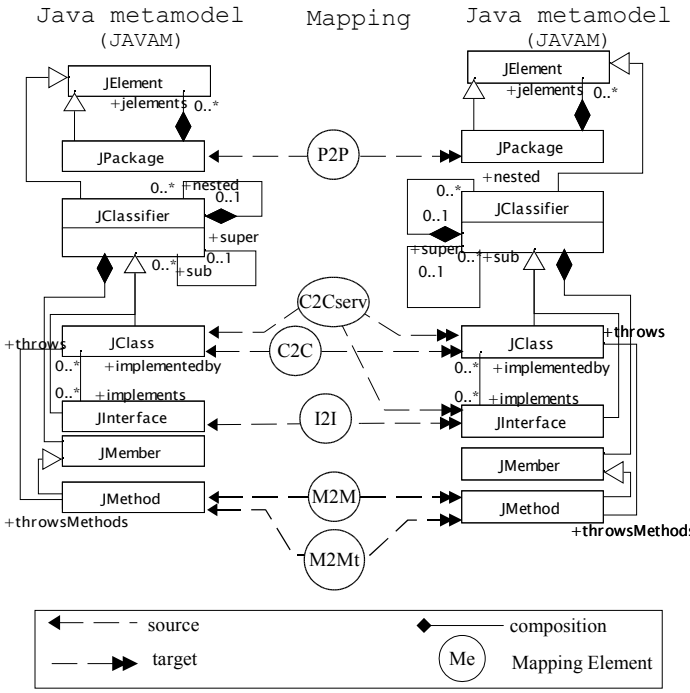


Figure 4.10 – Spécification de correspondances Java vers Java : raffinement

Comme nous l'avons énoncé précédemment, nous utilisons une démarche progressive pour créer la spécification de correspondances entre UML et JWSDP. Nous avons d'abord présenté la spécification de correspondances entre UML et Java. Nous devons ensuite raffiner un modèle Java en un autre modèle Java selon le *template* (figure 3.18a, page 117). La figure 4.10 présente la spécification de correspondances de Java en Java selon le *template*. Ce raffinement doit prendre en compte le fait que certains `JClass` réalisent des services alors que d'autres sont ordinaires. Par conséquent, nous devons dupliquer les correspondances pour qu'une `JClass` ordinaire soit mise en relation avec une autre `JClass` ordinaire, alors qu'une `JClass` réalisant un service doit être mise en relation avec une autre `JClass` et une

`JInterface`. De plus, les méthodes d'une `JClass` réalisant un service doivent retourner une exception de type `RemoteException`. Le filtrage entre une `JClass` ordinaire et une `JClass` réalisant un service est effectué par le biais d'une expression OCL[9]. Pour faire cette distinction, nous allons considérer qu'une `JClass` dont le nom commence par le mot « Service » [10] fournit un service. Par exemple, l'expression `c.name.startsWith('Service')` peut réaliser ce filtre. Ainsi, la correspondance `C2C` établit l'équivalence entre une `JClass` ordinaire du modèle source et une `JClass` du modèle cible. Similairement, `I2I` établit l'équivalence entre une `JInterface` du modèle source et une `JInterface` du modèle cible, alors que `C2CI` se charge de la mise en forme d'une `JClass` conforme au *template* de JWSDP. La correspondance `M2M` établit l'équivalence entre l'élément `JMethod` ordinaire de Java et l'élément `JMethod` de Java, alors que `M2Mt` se charge de la mise en forme d'une méthode réalisant une opération de service Web. L'expression `owner.name.startsWith('Service'))` peut être utilisée comme filtre pour `M2Mt` et son inverse comme filtre pour `M2M`.

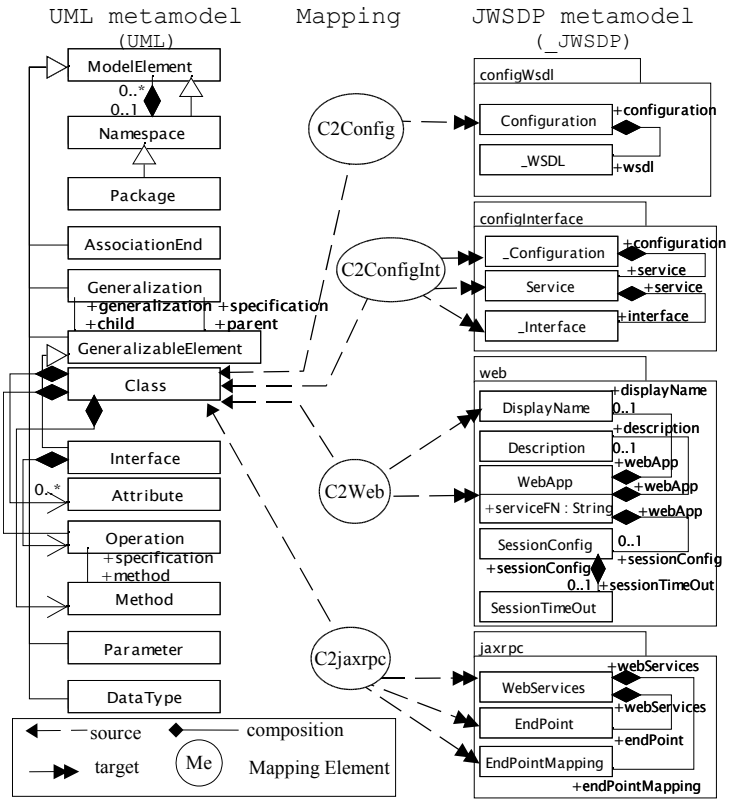


Figure 4.11 – La spécification de correspondances entre UML et JWSDP

La spécification de correspondance entre le métamodèle d'UML et celui de JWSDP (cf. figure 4.11) contient quatre éléments de correspondance : `C2Config`, `C2Web`, `C2ConfigInt` et `C2jaxrpc`.

[9]Nous utilisons les expressions OCL en ajoutant d'autres opérations telles que `startsWith(par :String)`, `endsWith(par :String)` pour pouvoir manipuler des chaînes de caractères.

[10]Une autre alternative serait l'utilisation d'un profil UML, mais nous voulons éviter l'utilisation de ce mécanisme d'extension.

4.4.2 Définition de transformations

La définition de transformations en ATL permettant de réaliser la transformation d'un modèle UML en un modèle Java doit être créée à partir de la spécification de correspondances présentée précédemment. Partant de ce fait, l'exemple 4.10 montre un fragment du code source en ATL qui est généré manuellement à partir de la spécification de correspondances.

Exemple 4.10 – La définition de transformation d'UML en Java (fragment)

```
-- === File : Transf_uml2java.atl
module UML2JAVAM;
create OUT : JAVAM from IN : UML;

rule P2P{
        from pck : UML!Package
        to jp : JAVAM!JPackage(
                name <-pck.name,
                jelements <- pck.ownedElement
        )
}

rule C2Jc{
        from c : UML!Class (c.namespace <> 'service')
        to jc : JAVAM!JClass(
        name <- c.name,
        visibility <- if c.visibility = #vk_public then
                                                #public
                                        else if c.visibility = #vk_private then
                                                #private
                                        else
                                                #protected
                                        endif   endif,
        modifier <- if c.isAbstract then
                                                #jabstract
                                        else if c.isLeaf then
                                                #final
                                        else
                                                #regular
                                        endif   endif,
        isActive <- c.isActive,
        jpackage <- c.namespace,
        super <- if c.generalization -> select(e|e.parent <> c)->size() > 0
                        then
                                c.generalization -> select(e|e.parent <> c)->
                                                        first().parent
                        else
                                JAVAM!JClass
                        endif,
        implements <- c.clientDependency -> select(e|e.stereotype->
                                exists(e|e.name='realize'))->collect(e|e.supplier)
        )
}
***
```

Selon notre démarche, l'élément de correspondance `P2P` (figure 4.9) est utilisé pour générer la règle de transformation `P2P` (exemple 4.10) en ATL, l'élément de correspondance `C2Jc` est utilisé pour générer

la règle de transformation `C2Jc` en ATL, et ainsi de suite. L'élément de correspondance `C2Jc` est de type plusieurs-vers-un, ce qui est traduit en ATL par une règle de la catégorie un-vers-un. Dans ce cas, l'information d'héritage peut être transmise à `super` de `JClass` par le biais d'une navigation utilisant l'association `Generalization <-> GeneralizableElement`.

Une fois que la définition de transformations de diagramme de classes d'UML vers Java est faite, il faut créer une deuxième transformation pour mettre un modèle Java en conformité avec le *template* JWSDP (figure 3.18a). Pour cette transformation, la définition de transformations doit être générée selon la spécification de correspondances illustrée par la figure 4.10. En fait cette étape est une transformation d'un modèle Java en un autre modèle Java dans lequel une `JClass` et une `JInterface` doivent être créées pour chaque `JClass` qui réalise un service Web. De plus, les méthodes qui réalisent les opérations d'un service doivent retourner une exception de type `RemoteException`. L'exemple 4.11 présente la définition de transformation qui fait ce raffinement.

Exemple 4.11 – La définition de transformation de Java en Java : raffinement (fragment)

```
-- File: Java2JWSDP_Template.atl
module Java2Java;
create OUT : JAVAMR  from IN : JAVAM;
uses strings;
rule P2P{
        from pck : JAVAM!JPackage(JAVAMR!JPackage.allInstances()->select(e|e.name='
            java.rmi')->size() =0)
        to jp : JAVAMR!JPackage(
                name <- pck.name
        ),
        pckRMI : JAVAMR!JPackage(
                name <- 'java.rmi'
        ),
        itf: JAVAMR!JInterface(
                name <- 'Remote',
                jpackage <- pckRMI
        ),
        cls : JAVAMR!JClass(
                name <- 'RemoteException',
                jpackage <- pckRMI
        )
}
rule P2Psimple{
        from pck : JAVAM!JPackage (JAVAMR!JPackage.allInstances()->select(e|e.name='
            java.rmi')->size() <> 0)
        to jp : JAVAMR!JPackage(
                name <- pck.name
        )
}
***
rule C2Cserv{
        from c : JAVAM!JClass   (c.name.startsWith('Service'))
        to jc : JAVAMR!JClass(
        name <- c.name,
        visibility <- c.visibility,
        modifier <- c.modifier,
        isActive <- c.isActive,
        jpackage <- c.jpackage,
        super <- c.super,
        implements <- c.implements->asSequence()->including(ITF)
```

```
        ),
        ITF : JAVAMR!JInterface(
        name <- 'ITF_'+c.name,
        visibility <- c.visibility,
        modifier <- c.modifier,
        super <- JAVAMR!JInterface.allInstances()->select(e|e.name='Remote')->first
            (),
        jpackage <- c.jpackage
        )
}
***
rule M2Mt{
        from m : JAVAM!JMethod (m.owner.name.startsWith('Service'))
        to jm : JAVAMR!JMethod(
                name <- m.name,
                owner <- m.owner,
                visibility <- m.visibility,
                modifier <- m.modifier,
                jparameters <- m.jparameters,
                body <- m.body,
                throws <- JAVAMR!JClass.allInstances()->select(e|e.name='
                    RemoteException')->first()
        ),
        jmitf : JAVAMR!JMethod(
                name <- m.name,
                owner <- JAVAMR!JClassifier.allInstances()->select(x|x.name='ITF_'+m
                    .owner.name)->first(),
                visibility <- m.visibility,
                modifier <- m.modifier,
                body <- m.body,
                throws <- JAVAMR!JClass.allInstances()->select(e|e.name='
                    RemoteException')->first()

        )
}
***
```

L'étape suivante permettant d'obtenir la plate-forme finale est la création d'une définition de la transformation de modèles Java vers du code Java. L'exemple 4.12 présente la définition de la transformation en ATL utilisant une requête pour déclencher la transformation modèle-vers-code, et l'exemple 4.13 présente les *helpers* permettant de naviguer et extraire les informations du modèle de Java, et écrire le code source en Java.

Exemple 4.12 – La définition de transformation de Java model en Java code

```
-- File: JWSDP_Template2SC_query.atl
query JAVAMR2SC_query = JAVAMR!JClassifier.allInstances()->
        select(e | e.oclIsTypeOf(JAVAMR!JClass) or e.oclIsTypeOf(JAVAMR!JInterface))
            ->
        collect(x | x.toString().writeTo('C:/SourceCode/J2EE/' + x.jpackage.name.
            replaceAll('.', '/') + '/' + x.name + '.java'));
uses JAVAMR2SC;
```

L'exemple 4.12 contient une requête utilisée pour déclencher la transformation modèle-vers-code. L'opération `allInstances()` retourne une collection constituée de toutes les instances de l'élément `JAVAMR!JClass` ou `JAVAMR!JInterface` présentes dans le modèle Java. Ensuite, une opération sur

les collections appelé `collect()` nous permet de récupérer chaque élément de cette collection (représenté par `x`) et d'appeler l'opération `toString()` sur chacun d'eux. Cette opération est un *helper* en ATL défini dans la bibliothèque `Java2SourceCode` (exemple 4.13). Après l'exécution de cet *helper*, la chaîne de caractères est écrite dans un fichier « *.java » par le biais de l'opération `writeTo()`. L'*helper* `toString()` défini dans le contexte de `JAVAMR!JClass` et de `JAVAMR!JInterface` enchaîne aussi d'autres appels à d'autres *helpers* définis dans le contexte de `JField`, `JMethod`, et ainsi de suite.

Exemple 4.13 – La bibliothèque JAVAMR2SC en ATL (fragment)

```
-- File: JWSDP_Template2SC.atl
library JAVAMR2SC;
***
helper context JAVAMR!JClass def: toString() : String =
        self.jpackage.toString() +
        self.visibility() +
        self.modifierAbstract() +
        'class ' +
        self.name +
        self.getSuperType()+
        self.getImplements() +
        ' {\n' +
        self.jmembers->select(e|e.oclIsTypeOf(JAVAMR!JMethod))->
                iterate(i; acc : String = '' |
                acc + i.toString()) +
        self.jmembers->select(e|e.oclIsTypeOf(JAVAMR!JField))->
                iterate(i; acc : String = '' |
                acc + i.toString()) +
        '\n}\n\n';
***
```

Il faut encore proposer une définition des transformations entre UML et le métamodèle de JWSDP (figure 3.18b, page 117) afin de générer complètement la plate-forme JWSDP. L'exemple 4.14 présente les règles de transformation pour générer un modèle basé sur JWSDP. Ces règles de transformation récupèrent les informations nécessaires à partir d'une `UML!Class` dont le nom commence par « Service » .

Les règles `C2Config`, `C2Web`, `C2jaxrpc` et `C2ConfigInt` récupèrent les informations nécessaires pour générer les fichiers de configuration et de déploiement respectifs « config-wsdl.xml », « web.xml », « jaxrpc-ri.xml » et « config-interface.xml ».

Exemple 4.14 – La définition de transformation entre UML et JWSDP (fragment)

```
-- == File : UML2JWSDP_Deploy.atl
module UML2_JWSDP_Deploy;
create OUT : _JWSDP  from IN : UML;
uses strings;
rule C2Config{
        from c : UML!Class(c.name.startsWith('Service'))
        to conf : _JWSDP!Configuration(
                serviceFN <- c.namespace.name,
                xmlns <- 'http://java.sun.com/xml/ns/jax-rpc/ri/config',
                wsdl <- _wsdl
        ),
        _wsdl: _JWSDP!_WSDL(
                location <-'http://Host:Port/'+ c.namespace.name+ '-jaxrpc/'+ c.
                    namespace.name +'?WSDL',
                packageName <- c.namespace.name)
```

```
}
rule C2Web{
        from c : UML!Class(c.name.startsWith('Service'))
        to web : _JWSDP!WebApp(
                serviceFN <- c.namespace.name,
                displayName <- displayN,
                description <- desc,
                sessionConfig <- sessconf
        ),
        displayN: _JWSDP!DisplayName(
                display <- 'This service: '+c.name
        ),
        desc: _JWSDP!Description(
                description <- 'Put here your description'
        ),
        sessconf: _JWSDP!SessionConfig(
                sessionConfig <- '',
                sessionTimeOut <- session
        ),
        ***
}
rule C2ConfigInt{
        from c : UML!Class(c.name.startsWith('Service'))
        to conf : _JWSDP!_Configuration(
                serviceFN <- c.namespace.name,
                xmlns <- 'http://java.sun.com/xml/ns/jax-rpc/ri/config',
                service <- serv
                ),
***
}
rule C2jaxrpc{
        from c : UML!Class(c.name.startsWith('Service'))
        to ws: _JWSDP!WebServices(
                serviceFN <- c.namespace.name,
                xmlns <- 'http://java.sun.com/xml/ns/jax-rpc/ri/dd',
                version <- '1.0',
                targetNameSpaceBase <- 'urn://'+c.namespace.name+'.com/wsdl',
                typeNameSpaceBase <- 'urn://'+c.namespace.name+'.com/types',
                urlPatternBase <- '/'+c.namespace.name,
                endPoint <- endP,
                endPointMapping <- endPM
        ),
***
}
```

Un modèle de JWSDP doit encore être transformé pour obtenir le code source (transformation modèle-vers-code). Le programme « _JWSDP_Deploy2SC_query.atl » (cf. annexe B) contient la requête en ATL pour récupérer les informations d'un modèle JWSDP et les enregistrer dans les documents JWSDP (fichiers de configuration et fichiers de déploiements). De plus, ce programme utilise la bibliothèque « _JWSDP_Deploy2SC » (cf. annexe B) qui contient les *helpers* utilisés dans la transformation.

4.5 UML vers dotNET

Dans cette section, nous utilisons notre démarche MDA pour générer le code source, les fichiers de déploiement et la configuration pour la plate-forme dotNET (le langage C# et le *framework* dotNET). Nous avons modélisé cette plate-forme en deux parties : un *template* et un métamodèle (figure 3.20, page 119). Comme nous l'avons dit précédemment, nous générons la plate-forme cible dotNET en une seule étape.

4.5.1 Spécification de correspondances

La figure 4.12 présente une spécification de correspondances entre le diagramme de classes d'UML et le langage C#. La correspondance `P2N` met en relation le `Package` d'UML avec le `Namespace` de C#, `C2C` met en relation la `Class` d'UML avec la `Class` de C#, et ainsi de suite. La correspondance `M2M` est de la catégorie plusieurs-vers-un. Elle doit par conséquent être adaptée à une correspondance un-vers-un, et l'information contenue dans l'élément `Operation` d'UML peut être récupérée par le biais de l'association `specification <->method`.

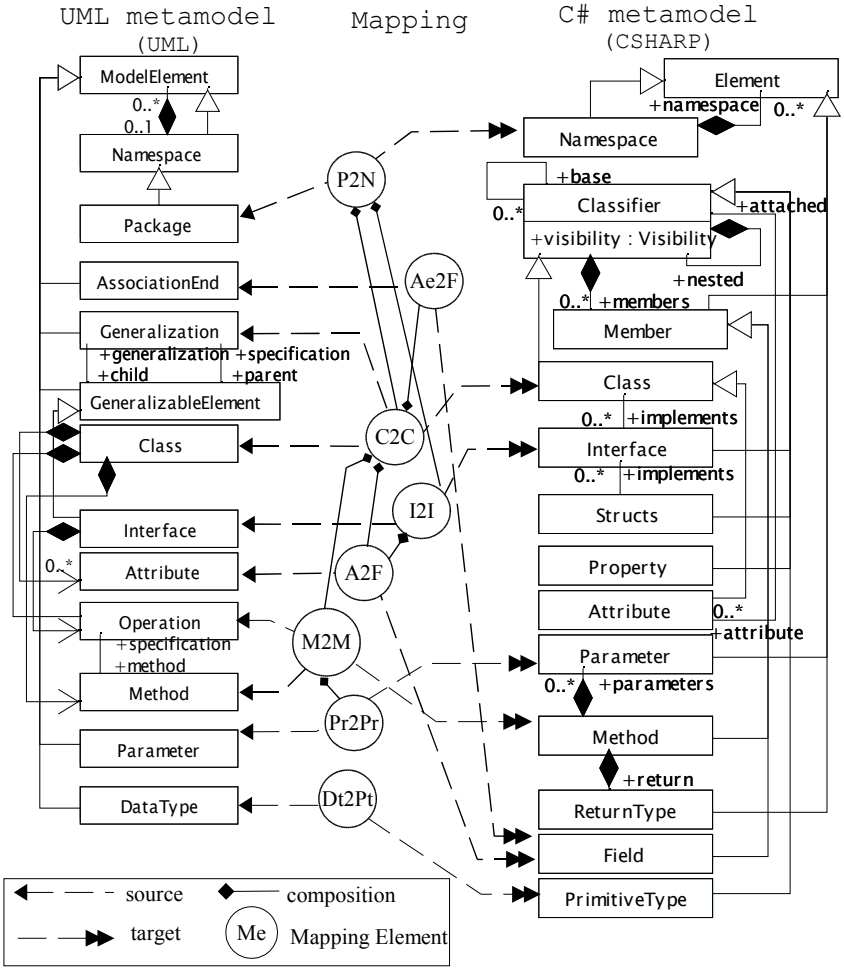


Figure 4.12 – Une spécification de correspondances entre UML et C#

Les correspondances `P2P`, `C2C`, `M2M`, ..., `Dt2Pt` sont constituées d'autres correspondances qui mettent en relation les attributs[11] et les références de chaque élément d'UML et de C#. Elles ne sont pas détaillées ici pour simplifier la présentation.

Comme nous l'avons dit précédemment, une classe en UML réalisant un service doit être identifiée par un nom commençant par « Service ». Dans la spécification de correspondances, nous devons faire le filtrage de ces classes et utiliser le *template* de dotNET pour créer une classe équivalente en C# (figure 3.20, page 119). Compte tenu de l'utilisation du *template*, une `Class` en C# doit hériter de la classe `WebService`, être attachée à l'attribut `WebServiceAttribute`, et ses méthodes doivent être attachées à l'attribut `WebMethodAttribute`. Dans ces conditions, la correspondance `C2C` doit être divisée en une correspondance ordinaire `C2Csimple` (c.-à-d. que le nom de la classe UML ne commence pas par « Service »), et une autre correspondance `C2Cs` qui réalise la mise en forme selon le *template*. Pour la même raison, la correspondance `M2M` doit être décomposée en une correspondance `M2Msimple` dans le cas de méthodes ordinaires, et en une autre correspondance `M2Mservice` dans le cas d'une méthode qui réalise un service. Cette dernière méthode doit être attachée à l'attribut `WebMethodAttribute`.

Pour réaliser la correspondance, il faut aussi faire l'équivalence entres les types de données permis dans chaque métamodèle. La correspondance `Dt2Pt` utilise l'équivalence entre les types de données de UML et de C#. Ainsi, le tableau 4.4 contient une équivalence entre les types de données d'UML et C#.

UML (DataType)	**C# (PrimitiveType)**
Float	float
Byte	byte
Double	double
Integer	int
Long	long
Short	short
String	String
Boolean	bool

Table 4.4 – La correspondance entre les types d'UML et de C#

4.5.2 Définition de transformations

La définition de transformations en ATL qui permet de transformer un modèle UML en un modèle C# doit être en conformité avec la spécification de correspondances présentée précédemment. Partant de ce fait, l'exemple 4.15 fournit un fragment du code source en ATL généré manuellement à partir de la spécification de correspondances. Selon notre démarche, l'élément de correspondance `P2N` (figure 4.12) est utilisé pour générer la règle de transformation `P2N` (exemple 4.15), l'élément de correspondance `C2Csimple` est utilisé pour générer la règle de transformation `C2Csimple`, et ainsi de suite. Comme énoncé précédemment, chaque classe réalisant un service Web doit hériter de la classe `WebService` et être liée à l'attribut `WebServiceAttribute`. Ceci est respecté par la règle de transformation `C2Cs`.

Exemple 4.15 – La définition de la transformation d'UML en C# (fragment)

```
-- == File :  UML2Csharp.atl
2 module UML2CSHARP;
```

[11] Plus précisément un méta-attribut et une métaréférence

```
create OUT : CSHARP from IN : UML;
uses strings;
rule P2N{
        from pck : UML!Package
        to ns : CSHARP!_NameSpace(
                name <-pck.name,
                elements <- pck.ownedElement
        )
}
rule C2Csimple{
        from c : UML!Class (not c.name.startsWith('Service'))
        to cs : CSHARP!Class(
        name <- c.name,
        visibility <- if c.visibility = #vk_public then
                                        #public
                                else if c.visibility = #vk_private then
                                        #private
                                else
                                        #protected
                                endif   endif,
        isAbstract <- c.isAbstract,
        isSealed <- false,
        nameSpace <- c.namespace,
        base <- if c.generalization -> select(e|e.parent <> c)->size() > 0
                        then
                                c.generalization -> select(e|e.parent <> c)-> first
                                    ().parent--c.generalization -> select(e|e.parent
                                     <> c)->first().parent
                        else
                                CSHARP!Class
                        endif,
        implements <- c.clientDependency -> select(e|e.stereotype->exists(e|e.name='
            realize'))->collect(e|e.supplier)
        )
}
rule C2Cs{
        from c : UML!Class (c.name.startsWith('Service'))
        to cs : CSHARP!Class(
        name <- c.name,
        visibility <- ***,
        isAbstract <- c.isAbstract,
        isSealed <- false,
        nameSpace <- c.namespace,
        base <- CSHARP!Class.allInstances()->select(e|e.name='WebService' and e.
            nameSpace.name =c.namespace.name)->first(),
        attribute <- CSHARP!Attribute.allInstances()->select(e|e.name='
            WebServiceAttribute' and e.nameSpace.name =c.namespace.name)->asSequence
            (),
        implements <- c.clientDependency -> select(e|e.stereotype->exists(e|e.name='
            realize'))->collect(e|e.supplier)
        )
} ***
```

L'annexe C contient les programmes « Csharp2SC_query.atl » et « Csharp2SC.atl » qui effectuent la transformation d'un modèle C# en code source C#.

Un modèle dotNET (contenant les informations pour le déploiement et la configuration de services Web) est généré par le programme « UML2dotNET_Deploy.atl » fourni dans l'annexe D. Les fichiers de déploiement et configuration sont générés à partir d'un modèle dotNET en utilisant les programmes « dotNET_Deploy2SC_query.atl » et « dotNET_Deploy2SC.atl » fournis dans l'annexe D.

4.6 EDOC vers Services Web

EDOC est un ensemble de spécifications supportant la création de modèles pour les systèmes EDOC. Parmi ces spécifications, nous nous intéressons plus particulièrement à la spécification d'ECA [152] et à la spécification de profils UML pour ECA [154]. Dans la spécification d'ECA, le métamodèle de CCA (*Component Collaboration Architecture*) est celui qui nous intéresse le plus parmi les cinq métamodèles présentés. Il détaille la façon de modéliser la structure et le comportement des composants qui constituent un système. Parmi ceux présentés dans la spécification de profils UML pour ECA, nous nous intéressons plus particulièrement au profil de CCA permettant la création de modèles UML agrémentés de profils (stéréotypes et valeurs marquées) conforme au métamodèle de CCA. La spécification de profils UML pour ECA est bien entendu une extension d'UML dédiée aux systèmes EDOC. Cependant, elle n'est pas capable de représenter toutes les caractéristiques du métamodèle de CCA. Les profils UML pour ECA ont l'avantage de permettre l'utilisation des outils UML qui supportent les profils pour créer et éditer des modèles UML étendus en conformité avec la spécification d'EDOC.

Étant donné que la spécification EDOC est constituée de métamodèles et de profils, la création d'un modèle métier conforme à EDOC peut être réalisée de deux manières : en utilisant directement le métamodèle CCA ou en utilisant les profils UML pour CCA. Puisque ces profils sont créés conformément au métamodèle de CCA, la spécification de correspondances entre profils UML pour CCA et WSDL, ainsi que celle entre le métamodèle CCA et WSDL doivent avoir des similarités. Dans le cas de profils pour CCA, nous devons mettre en relation des éléments UML étendus par des profils pour CCA avec ceux de WSDL. Dans le cas du métamodèle de CCA, nous devons mettre en relation les éléments CCA avec ceux de WSDL. L'utilisation de profils pour CCA et le métamodèle de CCA vont nous permettre d'analyser l'impact de l'utilisation de profils UML et de métamodèles spécifiques dans la création de spécifications de correspondances et dans la définition de transformations.

Dans un premier temps, nous présentons la spécification de correspondances entre les profils UML pour CCA et WSDL. Nous exposons ensuite la définition de transformations en ATL entre des profils UML pour CCA et WSDL. Dans un deuxième temps, nous présentons la spécification de correspondances entre le métamodèle de CCA et WSDL, puis nous exposons la définition de transformations en ATL entre CCA et WSDL.

4.6.1 Spécification de correspondances : profils UML pour CCA et métamodèle de WSDL

La figure 4.13 montre une spécification de correspondances entre le profil UML pour CCA et le métamodèle de WSDL. Les stéréotypes `<DataElement>` et `<PortOwner>` ne sont pas définis dans la spécification de profils UML pour ECA. Nous les avons ajoutés à titre informatif. De plus, le profil `<OperationPort>` peut être remplacé par une `Operation` UML, et le `<CompositeData>` par un `Parameter` UML.

L'élément de correspondance `P2D` met en relation l'élément `Package` d'UML avec l'élément `Definition` de WSDL. L'élément de correspondance `PC2S` met en relation l'élément `Class` d'UML (avec le profil `<ProcessComponent>`) avec l'élément `Service` de WSDL. L'élément de correspondance

`P2PB` met en relation un élément `Class` d'UML (avec le profil `<ProtocolPort>`) avec les éléments `PortType`, `Binding` et `Port` de WSDL. L'élément de correspondance `O2O` met en relation l'élément `Class` d'UML (avec le profil `<OperationPort>`) avec les éléments `Operation` et `BindingOperation` de WSDL. L'élément de correspondance `F2M` met en relation l'élément `Class` d'UML (avec le profil `<FlowPort>`) avec l'élément `Message` de WSDL, et ainsi de suite.

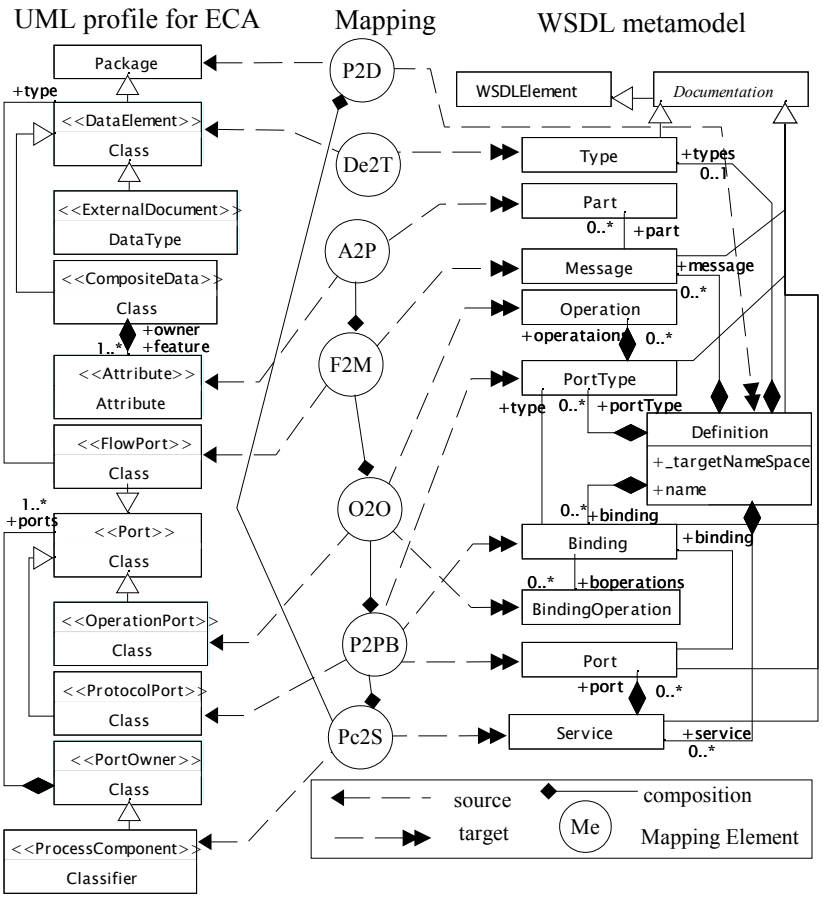


Figure 4.13 – Une spécification de correspondances entre le profil UML pour CCA et le métamodèle de WSDL

Chaque élément de correspondance contient une expression OCL permettant d'effectuer la conversion entre les éléments des profils UML pour CCA et WSDL, et un filtre pour faire la distinction entre les éléments d'UML agrémentés de profils. Par exemple, l'élément de correspondance `O2O` contient le filtre et l'expression OCL suivants :

- Filtre :

```
self.stereotype -> exists(e|e.name=name)
```

- Expression OCL de conversion :

```
UML!AssociationEnd.allInstances()->select(e|e.participant=self)->
   collect(x|x.association)->select(a |
```

```
      a.hasStereotype('initiates')) ->first().connection->
       select(y|y.participant <> self)->
       select(z | z.participant.hasStereotype('FlowPort') )->
       collect(e|e.participant)->first();
```

4.6.2 Définition de transformations : profils UML pour CCA et métamodèle de WSDL

La définition de transformations en ATL permettant la transformation d'un modèle construit avec des profils UML pour CCA en un modèle WSDL doit être créée à partir de la spécification de correspondances présentée précédemment. Partant de ce fait, l'exemple 4.16 montre un fragment du code source en ATL généré manuellement à partir de la spécification de correspondances. Selon notre démarche, l'élément de correspondance `Pc2S` (figure 4.13) est utilisé pour générer la règle de transformation `Pc2S` (exemple 4.16), l'élément de correspondance `P2PB` est utilisé pour générer la règle de transformation `P2PB`, l'élément de correspondance `O2O` est utilisé pour générer la règle de transformation `O2O`, et ainsi de suite. Dans la règle de transformation `O2O`, nous mettons l'accent sur le filtre et l'expression OCL de conversion. Dans le cas du filtre, nous l'avons transformé en un *helper* `hasStereotype` avec les mêmes instructions que celles présentées dans la spécification de correspondances. Dans le cas de la conversion, nous l'avons transformée en un autre *helper* appelé `getFlowInitiates`.

Exemple 4.16 – La définition de transformation de profil UML pour CCA en WSDL (fragment)

```
-- File: EDOCprof2WSDL.atl
module EDOCprof2WSDL;
create OUT : WSDL  from IN : UML;
-- helpers
helper context UML!ModelElement def: hasStereotype(name: String): Boolean =
        self.stereotype -> exists(e|e.name=name);
helper context UML!Class def: getEDOCProtocolPort(): Sequence(UML!Class) =
        UML!AssociationEnd.allInstances()->select(e|e.participant=self)->
                collect(x|x.association)->first().connection->
                        select(y|y.participant <> self)->
                                select(z | z.participant.hasStereotype('ProtocolPort
                                    ') and z.participant.inheritInterface())->
                                        collect(e|e.participant);
helper context UML!Class def: getBindingOperations(): Sequence(UML!Method) =
    if self.generalization -> select(e|e.parent <> self and e.parent.hasStereotype('
        Interface'))->size() > 0
                        then
                                self.generalization -> select(e|e.parent <> self and
                                     e.parent.hasStereotype('Interface'))-> first().
                                    parent.feature ->select (e|e.oclIsTypeOf(UML!
                                    Method))
                        else
                                Sequence{}
                        endif;
***
helper context UML!Class def: getFlowInitiates() :  UML!Class =
        UML!AssociationEnd.allInstances()->select(e|e.participant=self)->
                collect(x|x.association)->select(a | a.hasStereotype('initiates'))
                    ->first().connection->
                        select(y|y.participant <> self)->
                                select(z | z.participant.hasStereotype('FlowPort') )
                                    ->
```

```
                                        collect(e|e.participant)->first();
rule Pc2S{
        from c : UML!Class (c.hasStereotype('ProcessComponent') and c.
            hasAssocResponds())
        to s : WSDL!Service(
                name <-  c.name,
                owner <- c.namespace,
                port <- c.getEDOCProtocolPort()
        )
}
rule P2PB{
        from c:UML!Class (c.hasStereotype('ProtocolPort') and c.hasAssocResponds())
        to port:WSDL!Port(
                name <- c.name + 'Port',
                binding <- bd,
                documentation <- '\t\t<soap:address location="'+ '
                    REPLACE_WITH_ACTUAL_URL' +'"/>\n'
        ),
        pType: WSDL!PortType(
        ***
        ),
        bd: WSDL!Binding(
                ***
        ),
        soapB : WSDL!SoapBinding(
                ***
        )
}
***
rule O2O {
        from c : UML!Class (c.hasStereotype('OperationPort'))
        to wsdlOp : WSDL!Operation(
        ***
        )
}
***
```

Une fois que le modèle de WSDL est obtenu, les définitions de transformations de l'exemple 4.4 (page 130) et de l'exemple 4.5 (page 130) peuvent être utilisées pour générer le document WSDL.

4.6.3 Spécification de correspondances : métamodèle de CCA et de WSDL

La figure 4.14 montre une spécification de correspondances entre le métamodèle de CCA et le métamodèle de WSDL.

L'élément de correspondance `P2D` (cf. figure 4.14) met en relation l'élément `Package` de CCA avec l'élément `Definition` de WSDL. L'élément de correspondance `Pc2S` met en relation l'élément `ProcessComponent` avec l'élément `Service` de WSDL. L'élément de correspondance `P2PB` met en relation l'élément `ProtocolPort` avec les éléments `PortType`, `Binding` et `Port` de WSDL. L'élément de correspondance `O2O` met en relation l'élément `OperationPort` avec les éléments `Operation` et `BindingOperation` de WSDL. L'élément de correspondance `F2M` met en relation l'élément `FlowPort` avec l'élément `Message` de WSDL, et ainsi de suite.

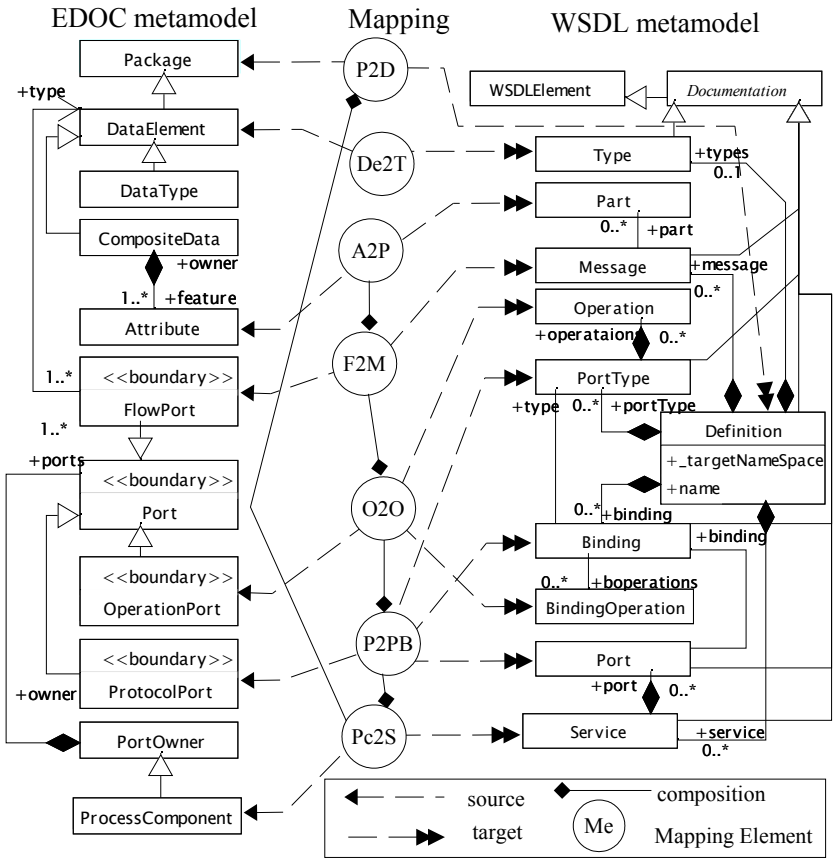


Figure 4.14 – Une spécification de correspondances entre le métamodèle d'EDOC (CCA) et celui de WSDL

4.6.4 Définition de transformations : métamodèle de CCA et de WSDL

La définition de transformations en ATL permettant la transformation d'un modèle CCA en un modèle WSDL doit être basée sur la spécification de correspondances présentée précédemment. Partant de ce fait, l'exemple 4.17 montre un fragment du code source en ATL généré manuellement à partir de la spécification de correspondances. Selon notre démarche, l'élément de correspondance `Pc2S` (figure 4.14) est utilisé pour générer la règle de transformation `Pc2S` (exemple 4.17), l'élément de correspondance `P2PB` est utilisé pour générer la règle de transformation `P2PB`, et ainsi de suite.

Exemple 4.17 – La définition de transformations EDOC (CCA) en WSDL (fragment)

```
-- File: EDOC2WSDL.atl
module EDOC2WSDL;
create OUT : WSDL  from IN : EDOC;
rule Pc2S{
        from pc : EDOC!ProcessComponent
        to sv: WSDL!Service(
                name <-  pc.name,
                owner <- pc.namespace,
                port <-  pc.ports->select(e|e.oclIsKindOf(EDOC!ProtocolPort)) )
}
```

```
rule P2PB{
        from pt:EDOC!ProtocolPort
        to port:WSDL!Port(
                name <- pt.name + 'Port',
                binding <- bd,
                documentation <- '\t\t<soap:address location="'+ '
                    REPLACE_WITH_ACTUAL_URL' +'"/>\n'
        ),
        bd : WSDL!Binding(
                name <- pt.name + 'Binding',
                owner <- port.owner.owner,
                type <- pType,
                soapBinding <- soapB
        ),
        soapB : WSDL!SoapBinding(
                transport <- 'http://schemas.xmlsoap.org/soap/http',
                style <- 'rpc'
        ),
        pType: WSDL!PortType(
                name <- pt.name,
                owner <- port.owner.owner,
                binding <- bd,
                operations  <- pt.protocol.ports->select(e|e.oclIsKindOf(EDOC!
                    OperationPort)) )
}
***
```

Une fois que le modèle de WSDL est obtenu, les définitions de transformations en ATL de l'exemple 4.4 (page 130) et de l'exemple 4.5 (page 130) peuvent être utilisées pour générer le document WSDL.

4.7 EDOC vers J2EE

Dans cette section, nous présentons la spécification de correspondances entre le métamodèle CCA et J2EE. Nous montrons ensuite la définition de transformation en ATL. Dans cette thèse, nous travaillons aussi avec les profils UML pour ECA afin de créer un modèle indépendant de la plate-forme et de le transformer en un modèle dépendant de la plate-forme J2EE. Cependant, nous limitons notre présentation à l'utilisation des métamodèles de CCA et de J2EE.

4.7.1 Spécification de correspondances

La figure 4.15 montre une spécification de correspondances entre le métamodèle de CCA et celui de Java. L'élément de correspondance `P2Jp` (cf. figure 4.15) met en relation l'élément `Package` de CCA avec l'élément `JPackage` de Java. L'élément de correspondance `Pc2Jc` met en relation l'élément `ProcessComponent` avec l'élément `JClass` de Java. L'élément de correspondance `Op2Jm` met en relation l'élément `OperationPort` avec l'élément `JMethod` de Java. L'élément de correspondance `A2Jpar` met en relation l'élément `Attribute` de CCA avec les éléments `JParameter` de Java, et ainsi de suite.

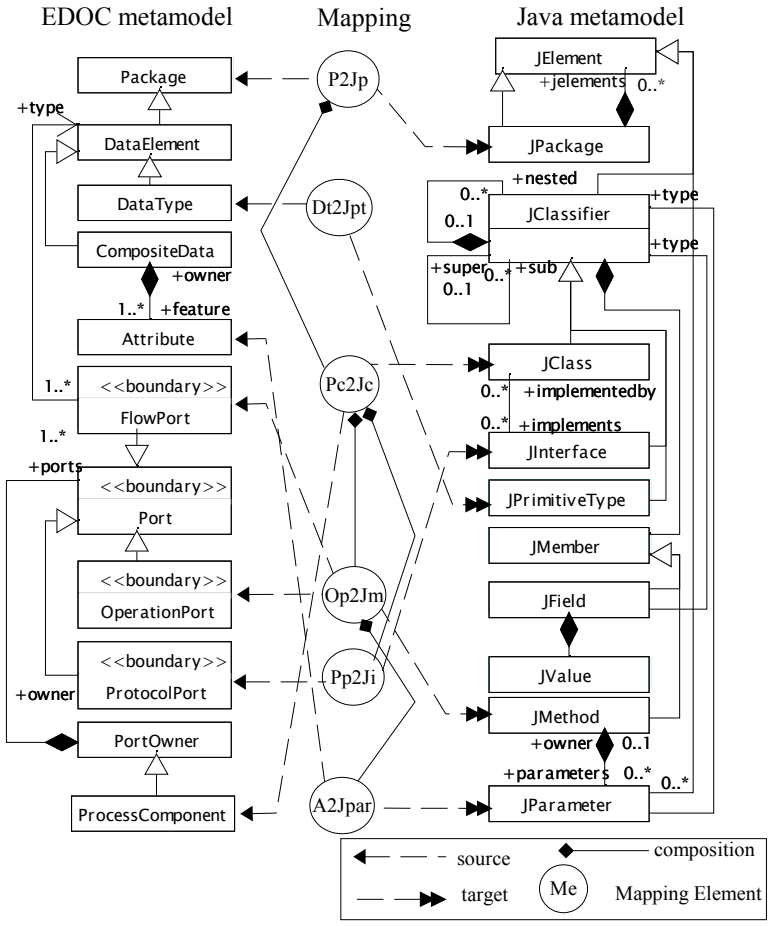


Figure 4.15 – Une spécification de correspondances entre le métamodèle d'EDOC (CCA) et Java

4.7.2 Définition de transformations

La définition de transformations en ATL permettant la transformation d'un modèle CCA en un modèle Java doit être basée sur la spécification de correspondances présentée précédemment. Partant de ce fait, l'exemple 4.18 montre un fragment du code source en ATL généré manuellement à partir de la spécification de correspondances. Selon notre démarche, l'élément de correspondance `P2Jp` (figure 4.15) est utilisé pour générer la règle de transformation `P2Jp` (exemple 4.18), l'élément de correspondance `Pc2Jc` est utilisé pour générer la règle de transformation `Pc2Jc`, et ainsi de suite.

Exemple 4.18 – La définition de transformation de CCA en Java (fragment)

```
-- File: EDOC2Java.atl
module EDOC2JAVAM;
create OUT : JAVAM  from IN : EDOC;

rule P2Jp{
        from pck: EDOC!Package( not pck.oclIsKindOf(EDOC!ProcessComponent)
        and
                not pck.oclIsKindOf(EDOC!CommunityProcess)
        and
```

```
10                  not pck.oclIsKindOf(EDOC!Protocol))
        to jp:JAVAM!JPackage(
12                  name <-pck.name,
                    jelements <- pck.ownedElements
14      )
}
16  rule Pc2Jc{
        from pc: EDOC!ProcessComponent
18      to jc: JAVAM!JClass(
                    name <- pc.name,
20      visibility <- #public,
        modifier <- #regular,
22      isActive <- false,
        jpackage <- pc.namespace,
24      super <- JAVAM!JClass,
        implements <- JAVAM!JInterface
26      )
}
28  rule Op2Jm{
        from op: EDOC!OperationPort
30      to jm:JAVAM!JMethod(
                    name <- op.name,
32                  owner <- op.owner.protocolPort.owner,
                    visibility <- #public,
34                  modifier <- #regular,
                    jparameters <- op.ports-> select(e | e.oclIsKindOf(EDOC!FlowPort))
36                  -> select(z|z.direction=#initiates)->
                            collect(x|x.type)->
38                          collect(y|y.feature) ->
                            union(Sequence{jpReturn}),
40                  body <- 'Type your code here!'
        ),
42      jpReturn : JAVAM!JParameter(
                    name <- 'return',
44                  result <- true,
                    type <- op.ports-> select(e | e.oclIsKindOf(EDOC!FlowPort)) ->
46                          select(z|z.direction=#responds)->
                                    first().type.feature ->asSequence()->
48                                          first().type,
                    owner <- jm
50      )
}
52  ***
```

Une fois que le modèle de Java est obtenu, la définition de transformations en ATL « Java2SourceCode_query.atl » (exemple 4.12, page 143) et « Java2SourceCode.atl » (exemple 4.13, page 144) peuvent être utilisées pour générer le code source Java.

4.8 Application à l'étude de cas : agence de voyages

Dans la section 3.2 (page 98), nous avons présenté une étude de cas illustrant notre démarche MDA. Dans la section 3.3 (page 100), nous avons exposé une méthodologie dans laquelle nous avons insisté sur la différence entre la spécification de correspondances et la définition de transformations. Nous avons

ensuite présenté la modélisation de l'étude de cas en UML (page 103) et EDOC (page 107), et nous avons également proposé un métamodèle pour chaque plate-forme cible.

Dans ce chapitre, nous avons parlé des étapes de transformations utilisées dans nos expérimentations (figure 4.1, page 122), puis nous avons détaillé en quoi consiste la spécification de correspondances et la définition de transformations entre un PIM (UML ou EDOC) et un PSM (Services Web, J2EE et dotNET). Ainsi, nous avons présenté des transformations modèle-vers-modèle, par exemple UML vers WSDL (page 128), et des transformations modèle-vers-code, par exemple modèle WSDL vers document WSDL (page 130). Par conséquent, nous présentons dans cette section les résultats des expérimentations obtenues avec notre démarche MDA.

Le résultat des transformations modèle-vers-modèle (cf. figure 4.1, page 122) consiste en l'obtention d'un modèle de Services Web, de J2EE et de dotNET. Pour simplifier la présentation des résultats, nous montrons dans l'annexe E un fragment du modèle de la plate-forme services Web (PSM - WSDL et BPEL) dans le format XMI obtenu à partir de la transformation du modèle UML de l'agence de voyages.

Le résultat des transformations modèle-vers-code (cf. figure 4.1, page 122) consiste en l'obtention du code source de Services Web (les documents WSDL et BPEL), de J2EE (le code source Java conforme à un modèle Java et au `template` de JWSDP, et les fichiers de configuration et déploiement), et de dotNET (le code source C# conforme à un modèle C# et au `template` de dotNET, et les fichiers de configuration et déploiement). Comme les expérimentations sont nombreuses, nous ne présentons que des fragments de code représentatifs. Nous omettons donc les résultats obtenus avec les profils UML pour CCA puisqu'ils sont très similaires à ceux obtenus avec le métamodèle de CCA.

Pour nos expérimentations, nous avons utilisé plusieurs outils présentés dans le tableau 4.5.

Outil	**Description de l'utilisation**
Poseidon for UML 1.6 *(Gentleware)*	Utilisé pour créer et éditer les métamodèles. Étant donné que cet outil ne permet que la création de modèles UML, nous avons créé et édité les métamodèles de services Web, J2EE, dotNET et EDOC comme modèles UML, puis nous les avons transformés en un méta-modèle en conformité avec MOF par le biais de l'outil uml2mof.
uml2mof *(netBeans)*	Transforme un modèle UML en un métamodèle équivalent en conformité avec MOF
J2EE *(Sun Microsystems)*	Parmi les technologies de J2EE, nous avons utilisé J2SDK version 1.4.2_04 et JWSDP version 1.3 (avec Tomcat version 1.5).
dotNET *(Microsoft)*	Parmi les technologies de dotNET, nous avons utilisé le framework dotNET 1.0.3705 (avec IIS)
ATL *(ATLAS group)*	Nous avons initialement utilisé la version 1.0 de ATL (basée sur MDR), puis la version 1.2 (basée sur Eclipse, MDR et EMF).
Oracle BPEL Process Manager	L'exécution du code BPEL a été fait par l'Oracle BPEL Process Manager version 2.0.

Table 4.5 – Les outils employés dans nos expérimentations.

Dans cette thèse, nous avons référencé plusieurs spécifications et différentes versions de la même spécification. C'est pourquoi, nous présentons dans le tableau 4.6 la version de chaque spécification utilisée dans nos expérimentations.

Spécification	Version	Date	Réference
UML	1.4	septembre 2001	[139]
EDOC - ECA	1.0	février 2004	[152]
EDOC - profil UML pour ECA	1.0	février 2004	[154]
MOF	1.4	avril 2002	[141]
XMI	1.2	janvier 2002	[146]
WSDL	1.1	mars 2001	[205]
BPEL	1.1	mai 2003	[9]

Table 4.6 – Les spécifications employées dans nos expérimentations.

4.8.1 UML vers Services Web

L'exemple 4.19 illustre le document WSDL de l'agence de voyages obtenu par la transformation du diagramme de classes d'UML (figure 3.4, page 104).

Exemple 4.19 – Le document WSDL de l'agence de voyages (à partir de UML)

```
<?xml version="1.0" encoding="UTF-8"?><definitions name="Service_travelagency"
    targetNamespace="urn://travelagency.wsdl">
<types>
        ***
</types>
<message name="findTravel">
        <part name="travelReq" type="ns:TravelReq"/>
</message>
<message name="findTravelResponse">
        <part name="return" type="ns:TravelList"/>
</message>
***
<portType name="ServiceTravelAg">
        <operation name=">findTravel" parameterOrder="">
                <input message="findTravel"/>
                <output message="findTravelResponse"/>
        </operation>
        ***
</portType>
<binding name="ServiceTravelAgBinding" type="ServiceTravelAg">
        <operation name="findTravel">
                <input>
                        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                            encoding/" use="encoded" namespace="urn://travelagency/
                            wsdl"/>
                </input>
                <output>
                        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                            encoding/" use="encoded" namespace="urn://travelagency/
                            wsdl"/>
                </output>
                <soap:operation soapAction=""/>
        </operation>
        ***
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
</binding>
```

```
<service name="ServiceTravelAg">
        <port name="ServiceTravelAgPort" binding="ServiceTravelAgBinding">
                <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
        </port>
</service>
</definitions>
```

L'exemple 4.20 montre le document BPEL de l'agence de voyages (comme composition d'autres services) obtenu par la transformation du diagramme d'activité d'UML (figure 3.5, page 105).

Exemple 4.20 – Le document BPEL de l'agence de voyages

```
<process name="ServiceTravelAg" targetNamespace="http://ServiceTravelAg.org/"
    suppressJoinFailure="false">
        <partnerLinks>
                <partnerLink name="ServiceTravelAg" partnerLinkType="
                    ITF_ServiceTravelAgLink" myRole="ITF_ServiceTravelAgProvider"/>
                <partnerLink name="ServiceCarHire" partnerLinkType="
                    ITF_ServiceCarHireLink" partnerRole="ITF_ServiceCarHireProvider"
                    />
                <partnerLink name="ServiceHotel" partnerLinkType="
                    ITF_ServiceHotelLink" partnerRole="ITF_ServiceHotelProvider"/>
                <partnerLink name="ServiceAirLines" partnerLinkType="
                    ITF_ServiceAirLinesLink" partnerRole="
                    ITF_ServiceAirLinesProvider"/>
        </partnerLinks>
        <variables>
                <variable name="inputServiceTravelAgfindTravel" messageType="
                    ITF_ServiceTravelAg_findTravel"/>
                <variable name="inputServiceCarHirefindCar" messageType="
                    ITF_ServiceCarHire_findCar"/>
                <variable name="inputServiceHotelfindRoom" messageType="
                    ITF_ServiceHotel_findRoom"/>
                ***
        </variables>
        <sequence>
                <receive name="ReceiveFindTravel" partnerLink="ServiceTravelAg"
                    portType="ITF_ServiceTravelAg" operation="findTravel" variable="
                    inputServiceTravelAgfindTravel" createInstance="true"/>
                <flow name="ForkA">
                                <sequence>
                                        <assign name="AssignInputFlight">
                                        ***
                                        </assign>
                                        <invoke name="CallFlight" partnerLink="
                                            ServiceAirLines" portType="
                                            ITF_ServiceAirLines" operation="
                                            findFlight" inputVariable="
                                            inputServiceAirLinesfindFlight"
                                            outputVariable="
                                            outputServiceAirLinesfindFlightResponse"
                                            />
                                </sequence>
                                <sequence>
                                        <assign name="AssingInputCar">
                                        ***
                                        </assign>
```

```
                                        <invoke name="CallCar" partnerLink="
                                            ServiceCarHire" portType="
                                            ITF_ServiceCarHire" operation="findCar"
                                            inputVariable="
                                            inputServiceCarHirefindCar"
                                            outputVariable="
                                            outputServiceCarHirefindCarResponse"/>
28                                  </sequence>
                                    <sequence>
30                                          <assign name="AssingInputRoom">
                                            ***
32                                          </assign>
                                            <invoke name="CallRoom" partnerLink="
                                                ServiceHotel" portType="ITF_ServiceHotel
                                                " operation="findRoom" inputVariable="
                                                inputServiceHotelfindRoom"
                                                outputVariable="
                                                outputServiceHotelfindRoomResponse"/>
34                                  </sequence>
                    </flow>
36                  <assign name="AssignReturn"> *** </assign>
                    <reply name="ReplyFindFlight" partnerLink="ServiceTravelAg" portType
                        ="ITF_ServiceTravelAg" operation="findTravel" variable="
                        outputServiceTravelAgfindTravelResponse"/>
38                  <terminate name="EndProcess"/>
            </sequence>
40  </process>
```

4.8.2 UML vers J2EE

L'exemple 4.21 illustre le code Java (conforme au *template* de la figure 3.18a) de l'agence de voyages obtenu par la transformation du diagramme de classes d'UML (figure 3.4, page 104).

Exemple 4.21 – Le code Java de l'agence de voyages (à partir d'UML)

```
    // CLASS
2   package travelagency;

4   public class ServiceTravelAg implements travelagency.ITF_ServiceTravelAg {
    public Reserv reserveTravel(TravelInf travelSel) throws java.rmi.RemoteException {
6                           //Type your code here
    }
8
    public TravelList findTravel(TravelReq travelReq) throws java.rmi.RemoteException {
10                          //Type your code here
    }
12
    public AckCancel cancelTravel(Reserv reservInf) throws java.rmi.RemoteException {
14                          //Type your code here
    }
16
    public AckPay payTravel(PayInf payInf) throws java.rmi.RemoteException {
18                          //Type your code here
    }
20
```

```
public ServiceCarHire servCarHire;
public ServiceAirLines servAirlLines;
public ServiceHotel servHotel;
public ServiceBank servBank;

}
// INTERFACE
package travelagency;

public interface ITF_ServiceTravelAg extends java.rmi.Remote {
public Reserv reserveTravel(TravelInf travelSel) throws java.rmi.RemoteException;
public TravelList findTravel(TravelReq travelReq) throws java.rmi.RemoteException;
public AckCancel cancelTravel(Reserv reservInf) throws java.rmi.RemoteException;
public AckPay payTravel(PayInf payInf) throws java.rmi.RemoteException;

}
```

L'exemple 4.22 illustre le document XML correspondant au fichier de configuration « config-wsdl-.xml » de JWSDP de l'agence de voyages obtenu par la transformation du diagramme de classes d'UML (figure 3.4, page 104).

Exemple 4.22 – Le document XML du fichier « config-wsdl.xml » pour l'agence de voyages

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
        xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
        <wsdl location="http://Host:Port/travelagency-jaxrpc/travelagency?WSDL"
                packageName="travelagency"/>
</configuration>
```

4.8.3 UML vers dotNET

L'exemple 4.23 illustre le code source en C# (ainsi que l'utilisation du *template* de la figure 3.20a) de l'agence de voyages obtenu par la transformation du diagramme de classes d'UML (figure 3.4, page 104).

Exemple 4.23 – Le code C# de l'agence de voyages

```
***

namespace travelagency
{
        [WebService]
        public  class ServiceTravelAg : WebService{

        [WebMethod]
        public Reserv reserveTravel(TravelInf travelSel) {
                        //Type your code here
        }

        [WebMethod]
        public TravelList findTravel(TravelReq travelReq) {
                        //Type your code here
        }
```

```
18          [WebMethod]
            public AckPay payTravel(PayInf payInf) {
20                          //Type your code here
            }
22
            [WebMethod]
24          public AckCancel cancelTravel(Reserv reservInf) {
                            //Type your code here
26          }

28          public ServiceHotel servHotel;
            public ServiceAirLines servAirlLines;
30          public ServiceCarHire servCarHire;
            public ServiceBank servBank;
32
            }
34  }
```

L'exemple 4.24 illustre le document XML correspondant au fichier de configuration du service Web (dotNET) de l'agence de voyages obtenu par la transformation du diagramme de classes d'UML (figure 3.4).

Exemple 4.24 – Le document XML du fichier « Web.config » pour l'agence de voyages

```
<?xml version="1.0" encoding="utf-8">
2   <configuration>
            <system.web>
4                   <webservice/>
                    <compilation defaultLanguage="c#" debug="true" />
6                   <customerErrors mode="RemoteOnly" />
                    <authentication mode="Windows" />
8                   <trace enabled="false" requestLimit="10" pageOutput="false"
                        traceMode="SortByTime" localOnly="true"/>
                    <sessionState mode="InProc" stateConnectionString="tcpip=host:port"
                        sqlConnectionString="data source=host;user id=sa:password="
                        cookieless="false" timeout="20"/>
10                  <globalization requestEncoding="utf-8" responseEncoding="utf-8"/>
            </system.web>
12  </configuration>
```

4.8.4 EDOC (métamodèle CCA) vers services Web

L'exemple 4.25 illustre le WSDL (document XML) de l'agence de voyages obtenu par la transformation du modèle EDOC-CCA (figure 3.6, page 108).

Exemple 4.25 – Le document WSDL de l'agence de voyages (à partir d'EDOC)

```
<?xml version="1.0" encoding="UTF-8"?>
2   <definitions name="Service_travelagency" targetNamespace="urn://travelagency.wsdl">
    <types>
4           ***
    </types>
6   <message name="findTravelCall">
            <part name="travelReq" type="ns:TravelReq"/>
8   </message>
```

```
<message name="findTravelResponse">
        <part name="travelList" type="ns:TravelList"/>
</message>
***
<portType name="SellTravel">
        <operation name="findTravel" parameterOrder="">
                <input message="findTravelCall"/>
                <output message="findTravelResponse"/>
        </operation>
        ***
</portType>
<binding name="SellTravelBinding" type="SellTravel">
        <operation name="findTravel">
                <input>
                        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                            encoding/" use="encoded" namespace="urn://travelagency/
                            wsdl"/>
                </input>
                <output>
                        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                            encoding/" use="encoded" namespace="urn://travelagency/
                            wsdl"/>
                </output>
                <soap:operation soapAction=""/>
        </operation>
        ***
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
</binding>
<service name="TravelAg">
        <port name="SellTravelPort" binding="SellTravelBinding">
                <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
        </port>
</service>
</definitions>
```

4.8.5 EDOC (métamodèle CCA) vers J2EE

L'exemple 4.26 illustre le code Java de l'agence de voyages obtenu par la transformation du modèle EDOC-CCA (figure 3.6, page 108).

Exemple 4.26 – Le code Java de l'agence de voyages (à partir d'EDOC)

```
//Class
package travelagency;

public class TravelAg implements travelagency.SellTravel {
public TravelList findTravel(TravelReq _TravelReq) throws java.rmi.RemoteException {
        Type your code here!
}

public Reserv reserveTravel(TravelInf _TravelInf) throws java.rmi.RemoteException {
        Type your code here!
}

public AckCancel cancelTravel(Reserv _Reserv) throws java.rmi.RemoteException {
```

```
14                          //Type your code here
    }
16
    public AckPay payTravel(PayInf _PayInf) throws java.rmi.RemoteException {
18                          //Type your code here
    }
20  }

22  //Interface
    package travelagency;
24
    public interface SellTravel extends java.rmi.Remote {
26  public TravelList findTravel(TravelReq _TravelReq) throws java.rmi.RemoteException;
    public Reserv reserveTravel(TravelInf _TravelInf) throws java.rmi.RemoteException;
28  public AckCancel cancelTravel(Reserv _Reserv) throws java.rmi.RemoteException;
    public AckPay payTravel(PayInf _PayInf) throws java.rmi.RemoteException;
30
    }
```

4.9 Comparaisons

Les comparaisons que nous présentons sont qualitatives. Nous avons pris comme base l'effort nécessaire pour créer une spécification de correspondances et une définition de transformations. Toutefois, une comparaison qualitative peut être réalisée aisément en évaluant la complexité des expressions OCL (par exemple, nombre de lignes du code) utilisées dans la spécification de correspondances et, par conséquent, dans la définition de transformation. Une comparaison quantitative peut également avoir lieu par l'utilisation des opérateurs fournis par la gestion de modèles [21], mais ceci sort du contexte de cette thèse.

4.9.1 PIM : UML versus EDOC

Dans cette section, nous comparons le métamodèle d'UML et celui d'EDOC. L'utilisation d'UML ou d'EDOC pour créer un PIM a un impact significatif dans une approche MDA. UML est un langage de modélisation d'utilisation générale, alors que EDOC est un langage de modélisation spécifique pour modéliser les systèmes EDOC. En fait, EDOC est un DSL (*Domain Specific Language*) pour les systèmes EDOC vus comme un ensemble de composants.

Pour analyser UML et EDOC, nous considérons d'abord les métamodèles d'UML et CCA, puis les profils UML pour CCA et le métamodèle CCA.

La définition de la transformation en ATL de UML vers WSDL et d'EDOC-CCA vers WSDL ont produit des résultats similaires. Cependant, nous avons remarqué que le métamodèle de CCA est plus proche de celui de WSDL que le métamodèle d'UML, ce qui peut être vérifié par les règles de transformation en ATL : les règles pour passer de CCA vers WSDL sont moins complexes et plus courtes que celles pour passer d'UML vers WSDL.

La différence la plus significative a été remarquée entre les profils d'UML pour CCA et le métamodèle de CCA. Nous avons remarqué que la spécification de correspondances entre les profils d'UML pour CCA et WSDL est similaire à celle entre le métamodèle de CCA et le métamodèle de WSDL, à l'exception des expressions OCL. Ainsi, la différence majeure a été vérifiée lors de la création des expressions

OCL pour définir la conversion entre les éléments de métamodèle : dans le cas des éléments agrémentés de profils UML pour CCA, il y a eu une utilisation massive des expressions OCL pour passer d'UML avec profils pour CCA vers WSDL, alors que, dans le cas de l'utilisation du métamodèle de CCA, les expressions OCL ont été moins utilisées pour passer de CCA vers WSDL. De plus, la spécification de correspondances entre des profils UML et WSDL a été insuffisante pour générer la définition de transformation en ATL qui a été complétée par des *helpers*. Ces *helpers* n'ont pas uniquement été utilisés pour éviter la répétition de code ATL, mais aussi pour pouvoir naviguer dans le modèle UML avec profils pour récupérer les informations nécessaires à la création d'un modèle WSDL. Enfin, les *helpers* ont rendu la définition de transformations entre les profils d'UML pour CCA et le métamodèle de WSDL plus impératives, quand, par exemple, un *helper* appelle un autre `helper` ou quand un *helper* utilise la récursivité.

L'utilisation de profils UML a ainsi rendu la spécification de correspondances difficile et, par conséquent, a exigé l'intervention du programmeur pour compléter la définition de transformations. Cette intervention a souvent abouti à la création d'un code impératif avec des appels d'*helpers* et l'utilisation de la récursivité. Par contre, l'usage directe du métamodèle de CCA et de WSDL a rendu la spécification de correspondances plus simple. Par conséquent, la définition de transformations a été générée presque complètement à partir de la spécification de correspondances car elle était davantage déclarative.

Dans [147], les auteurs ont classé quatre différentes approches de transformation de modèles dans MDA. Notre démarche étant plus proche de la transformation de métamodèles, il est naturel que la transformation et les métamodèles y aient une place très importante. L'utilisation de profils UML doit donc être évitée puisqu'ils rendent la création de la spécification de correspondances et la définition de transformations plus difficiles. Ainsi nous encourageons l'utilisation de métamodèles spécifiques au domaine comme EDOC. Cependant, les profils UML sont indispensables dans d'autres techniques, comme la technique de marquage. Le choix des éléments (profils ou métamodèles) utilisés dans une approche MDA est ainsi fortement dépendant de la technique utilisée.

De plus, les profils UML peuvent être utilisés pour réduire la distance sémantique entre le métamodèle d'UML et un autre métamodèle tel que celui proposé pour BPEL (section 4.3.3, page 131).

4.9.2 PSM : J2EE versus dotNET

Dans nos expérimentations, nous avons uniquement utilisé les outils de base liés aux plates-formes J2EE (Java et JWSDP) et dotNET (C# et *framework* pour services Web). Par exemple, nous avons utilisé les compilateurs de Java et de C#, et les outils pour déployer un service Web J2EE ou dotNET. Nous avons évité l'utilisation d'environnements de développement pour J2EE (comme Eclipse) ou pour dotNET (comme Visual Studio dotNET). Nous comparons les plates-formes et non les outils de développement. L'utilisation de J2EE ou de dotNET comme plate-forme cible n'a pas eu d'impact dans notre approche MDA, même si le métamodèle utilisé n'était que des fragments, mais assez représentatifs des plates-formes expérimentées. Le métamodèle de J2EE et celui de dotNET présentent finalement plusieurs similarités, les langages Java et C# étant très semblables. De plus, les fichiers de configuration et déploiement de JWSDP et dotNET ont été générés à partir du PIM puis complétés.

4.10 Synthèse

Dans ce chapitre, nous avons complété notre démarche MDA pour développer des applications orientées Internet. Après avoir présenté un formalisme graphique minimal et l'utilisation d'OCL pour spécifier

la correspondance entre métamodèles, nous avons démontré comment utiliser la spécification de correspondances pour générer la définition de transformations.

Nous avons appliqué notre démarche pour spécifier les correspondances et définir les transformations entre les métamodèles de PIM (UML ou EDOC) et ceux de PSM (Service Web, J2EE et dotNET). Une fois les définitions de transformations en ATL disponibles, nous les avons utilisées pour passer d'un modèle indépendant de plate-forme vers un modèle dépendant de la plate-forme. Nous avons ensuite appliqué une transformation modèle-vers-code pour obtenir la plate-forme finale.

Comme nous l'avons illustré, nous avons généré du code final de WSDL, mais nous avons été incapables de générer la totalité des autres plates-formes. D'une part, nous avons choisi de modéliser l'étude de cas avec une forte granularité. Il est ainsi logique que nous ayons seulement obtenu les squelettes de J2EE et de dotNET. D'autre part, nous avons suggéré l'utilisation d'*Action Semantics* pour permettre la modélisation avec une faible granularité. Dans le cas des diagrammes d'activités d'UML et BPEL, nous avons remarqué que la distance sémantique entre les deux métamodèles a été réduite par l'utilisation de profils UML (valeurs marquées). Cependant, nous avons aussi remarqué que l'utilisation de profils rend la création de spécifications de correspondances et de définitions de transformations difficiles.

À partir de cette étude, nous pouvons émettre les remarques suivantes :

- Les niveaux d'abstraction sont nécessaires pour développer les systèmes complexes ainsi que la séparation explicite entre la spécification de correspondances et la définition de transformation.
- La distance sémantique entre métamodèles est décisive pour le processus de transformation.
- L'utilisation massive des profils UML rend la création de la spécification de correspondance et, par conséquent, la définition de transformation difficile.
- Les bibliothèques et les APIs des différentes plates-formes doivent être intégrées dans l'architecture à quatre niveaux soit comme une partie d'un métamodèle de plate-forme ou d'un modèle de bibliothèque.
- Dans le cas de C#, la programmation orientée attributs (*Attribute-Oriented Programming*) doit être prise en compte afin de bien modéliser la plate-forme dotNET.
- Un métamodèle pivot entre UML et BPEL4WS pourrait être utile pour faciliter la création de correspondances et transformations entre leurs métamodèles. Pour notre part, nous avons utilisé les valeurs marquées pour remplir la distance sémantique entre UML et BPEL.

Dans le chapitre suivant, nous présentons notre outil qui permet la création et l'édition d'une spécification de correspondances (autrement dit, modèles de correspondances) puis la génération d'une définition de transformation (autrement dit, modèles de transformations).

PARTIE III

Prototypage : développement d'un *plug-in* sous « Eclipse »

CHAPITRE 5

Un outil pour la spécification de correspondances

5.1 Introduction

Dans les chapitres précédents, nous avons présenté notre démarche MDA et l'avons appliquée à la plate-forme des Services Web. Une des caractéristiques de notre démarche est la séparation entre spécification de correspondances et définition de transformations. Ainsi, nous avons d'abord présenté une représentation graphique et l'utilisation d'OCL permettant de spécifier les correspondances, puis nous avons créé manuellement la définition de transformation. Cependant, la création de correspondances entre éléments de métamodèles et la génération de définitions de transformation sont des tâches difficiles, voire ardues et sources d'erreurs si elles sont réalisées manuellement [115]. Ainsi, la semi-automatisation (ou automatisation) du processus de création de spécification de correspondances et la génération de définitions de transformations est un axe important dans le cadre de MDA. Certaines propositions de prise en charge de la spécification de correspondances se sont inspirées de techniques mettant en œuvre des heuristiques [172] (pour identifier des similarités structurelles entre modèles) et de techniques d'apprentissage (pour l'apprentissage de correspondances) [120]. D'autres propositions basées sur la théorie des graphes [5] ont aussi étudié les problèmes de correspondances et de transformations.

Dans ce chapitre, nous exposons notre proposition permettant de spécifier des correspondances entre métamodèles. Notre approche de la spécification de correspondances est basée sur un métamodèle et mise en œuvre par notre outil implémenté comme un *plug-in* sous « Eclipse ». Un modèle de correspondance conforme à ce métamodèle spécifie les relations entre les éléments de deux métamodèles. Ce métamodèle de correspondance est conforme à Ecore, langage de métamodèlisation défini dans EMF (*Eclipse Modeling Framework*). Le *plug-in* supporte la spécification de correspondances, qui est l'étape préliminaire à la création de définitions de transformations. Nous appliquons notre outil aux différentes expérimentations présentées dans les chapitres précédents.

5.2 Fondements pour la spécification de correspondances

Les techniques de spécification de correspondances ne sont pas nouvelles en informatique. Cela fait longtemps que dans le domaine des bases de données on étudie les techniques de correspondances et de transformations entre modèles (c.-à-d. schémas). L'exemple le plus connu est le passage entre le modèle conceptuel entité association et le modèle relationnel. Plus généralement, le problème de l'intégration

des bases de données illustre bien l'importance des techniques de correspondances et de transformations [159]. Cependant, la création de correspondances entre éléments de métamodèles et la génération de définitions de transformations a pris une nouvelle dimension avec l'approche MDA, car le modèle est au centre du développement des systèmes.

Une spécification de correspondances peut être formalisée comme suit [114] :

Soit $M_1(s)/M_a$, $M_2(s)/M_b$, et $C_{M_a \to M_b}/M_c$, où M_1 est un modèle du système s créé en utilisant le métamodèle M_a, M_2 est un modèle du même système s créé en utilisant le métamodèle M_b, et $C_{M_a \to M_b}$ est la correspondance entre M_a et M_b créée en utilisant le métamodèle M_c, alors la transformation peut être définie comme la fonction suivante : $Transf(M_1(s)/M_a, C_{M_a \to M_b}/M_c) \to M_2(s)/M_b$. En général, M_a, M_b et M_c sont conformes au même métamodèle, ce qui simplifie la spécification de correspondances. Dans cette section, nous souhaitons considérer la fonction de correspondance $C_{M_a \to M_b}/M_c$. Dans un premier temps, nous pouvons définir $C_{M_a \to M_b}/M_c$ comme suit : $C_{M_a \to M_b} \supseteq \{M_a \cap M_b\}$, où l'opérateur $\cap$, renvoie les éléments de M_a et M_b ayant une structure et une sémantique équivalentes.

Nous pouvons aussi représenter une correspondance comme un ensemble d'éléments de correspondances, soient :

$M_a = \{a_1, a_2, a_3, ..., a_m\}$ et
$M_b = \{b_1, b_2, b_3, ..., b_n\}$, alors
$C_{M_a \to M_b} = \{c_1, c_2, c_3, ..., c_p\}$
où
$c_i = \{a_k, b_j\}$,
$i = \{i \in N \mid 1 \leq i \leq p\}$,
$k = \{k \in N \mid 1 \leq k \leq m\}$ et
$j = \{j \in N \mid 1 \leq j \leq n\}$.

En réalité, les modèles (dans le sens général : modèles, métamodèles et métamétamodèles) peuvent être représentés comme des ensembles d'éléments. De plus ces ensembles sont complexes et hétérogènes car il s'agit de classes, attributs, relations, énumérations et types de données. Ainsi, la création de correspondances n'est donc pas facile.

Pour simplifier cette tâche, nous divisons les éléments d'un métamodèle en deux catégories : éléments de bases et relations. Les éléments de bases peuvent être des classes, des attributs, des énumérations et des types de données. Les relations établissent une liaison entre classes. $C_{M_a \to M_b}$ doit donc satisfaire les exigences suivantes pour être une correspondance complète [113] :

1. **Préservation des éléments de bases** : chaque élément de base de M_a doit vérifier une des exigences suivantes :
   - Il correspond à un élément de base égal ($=$) à M_b.
   - Il correspond à un ensemble d'éléments de base de M_b qui sont similaires ($\cong$)[1].
   - Il fait partie d'un ensemble d'éléments de base de M_a qui est similaire à un élément de base de M_b.
2. **Préservation des relations** : chaque relation de M_a doit vérifier une des conditions suivantes :
   - Elle a une relation correspondante en M_b.
   - Elle a un ensemble de relations correspondantes en M_b qui sont similaires ($\cong$).
   - Elle peut être implicitement préservé en M_b (par le biais des attributs ou classes fusionnées).

[1] Dans [21], le terme similaire est défini comme suit : « *we mean that they are related but we do not express exactly how* ».

Si M_b demande un élément de base ou une relation pouvant être déduite à partir de deux ou plusieurs éléments ou relations de M_a, respectivement, alors la préservation des éléments de base et la préservation des relations sont satisfaites.

Si une correspondance ne peut pas satisfaire aux deux exigences, alors elle n'est pas complète, et la définition de transformation ne le sera pas non plus. Par conséquent, le modèle cible généré à partir d'un modèle source en utilisant la définition de transformation ne contiendra pas toutes les informations du modèle source.

Le processus d'identification et de caractérisation des correspondances entre métamodèles est désigné par *schema matching*[172] dans le contexte des bases de données. En réalité, les correspondances décrivent comment deux métamodèles sont mis en relation. Le *schema matching* génère ainsi des correspondances. Selon l'algèbre de gestion des modèles [21], la correspondance est produite à partir de l'opérateur `match`, qui prend deux métamodèles en entrées et renvoit la correspondance entre eux.

L'identification des correspondances entre métamodèles est généralement basée sur la structure des métamodèles. Cette structure dépend des relations entre les éléments du métamodèle.

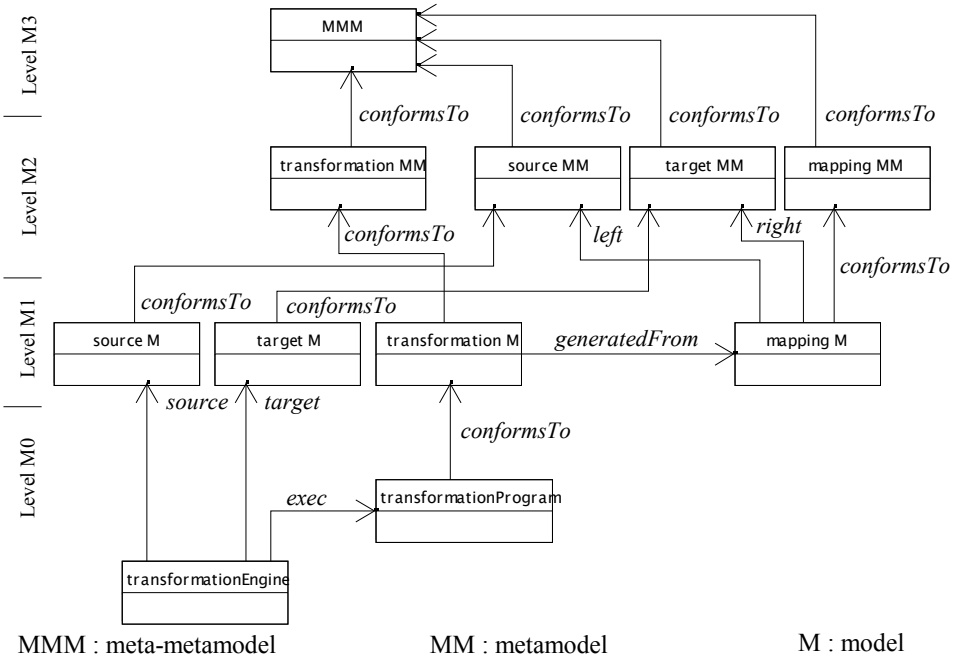


Figure 5.1 – L'architecture type de la transformation de modèles (quatre niveaux)

Dans la figure 5.1, nous reprenons selon notre approche l'architecture type pour la transformation de modèles et la modifions pour en faire ressortir chaque participant classé en conformité avec l'architecture à quatre niveaux. Ainsi, dans le niveau `M3`, il y a un métamétamodèle (`MMM`). Dans le niveau `M2`, il y a les métamodèles de transformation (`transformation MM`), le métamodèle de correspondance (`mapping MM`), et les métamodèles source et cible. Dans le niveau `M1`, il y a le modèle source, le modèle cible, le modèle de transformation (`transformation M`) et le modèle de correspondance (`mapping M`). Dans le niveau `M0`, il y a le programme de transformation (`transformationProgram`). Nous mettons l'accent sur le modèle de correspondance et le modèle de transformation. Ainsi, le modèle de correspondance est conforme au métamodèle de correspondance, et met en relation un métamodèle source (`left`) et cible (`right`), et le modèle de transformation est généré à partir d'un modèle de correspondances.

Pour mettre en œuvre cette architecture type, nous proposons un métamodèle de correspondance.

5.3 Métamodèle de correspondance

Un métamodèle de correspondance doit permettre la spécification des inter-relations (correspondances) entre les éléments de deux métamodèles sans les modifier. De plus, il doit permettre de :

- Identifier puis relier les éléments équivalents de deux métamodèles (source et cible). Ceci constitue la spécification de correspondances.
- Gérer l'historique des différentes alternatives de correspondances.
- Spécifier des correspondances bidirectionnelles (*Bi-directional mapping*). Elles sont souhaitables, mais souvent complexes à réaliser [101].
- Naviguer entre les éléments des deux métamodèles en correspondance.
- Rester le plus possible indépendant du langage de transformation.

La figure 5.2 présente notre proposition d'un métamodèle pour la spécification de correspondances [81] [113] [114].

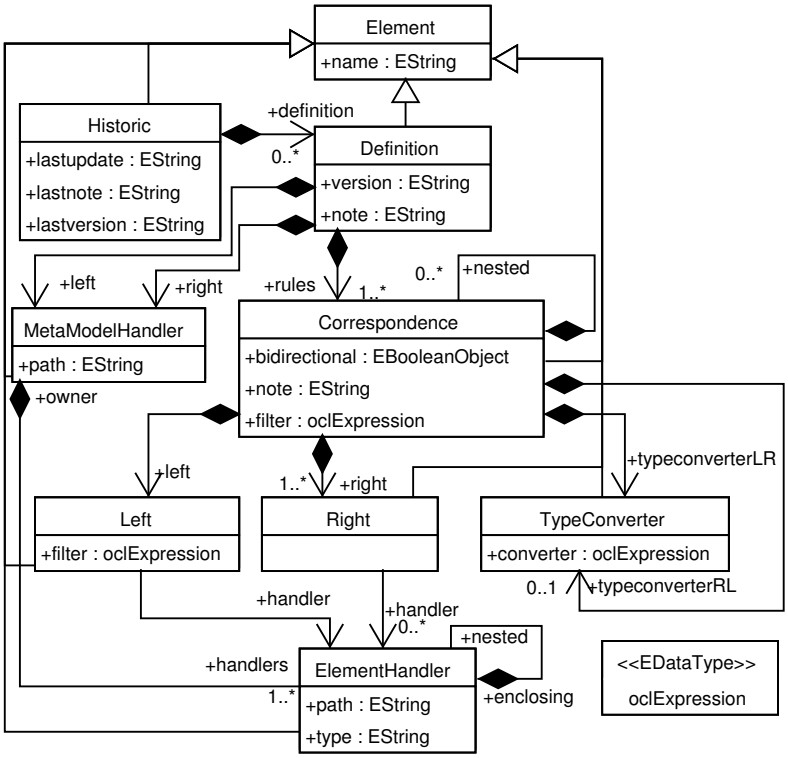


Figure 5.2 – Un métamodèle de spécification de correspondances

Dans ce métamodèle, nous considérons que la correspondance pourrait être unidirectionnelle ou bidirectionnelle. Dans le premier cas, un métamodèle source est mis en correspondance avec un métamodèle cible, dans le second la correspondance est spécifiée dans les deux sens. Ainsi, de manière générale, les deux métamodèles sont désignés par le métamodèle gauche et le métamodèle droit.

Ce métamodèle présente les éléments suivants :

- `Element` est une généralisation des autres éléments.
- `Historic` permet de préciser les différents choix effectués lors de la spécification de correspondances. Il comprend la date de la dernière mise à jour, une note, le numéro de la dernière version et un ensemble de définitions représentant les différentes alternatives de correspondances.
- `Definition` est l'élément principal contenant toutes les définitions de correspondances entre les deux métamodèles gauche et droit (c.-à-d. chaque correspondance possède un élément gauche et un ou plusieurs éléments droits).
- `Correspondence` permet de spécifier la correspondance proprement dite entre deux ou plusieurs éléments des métamodèles gauche et droit. La correspondance possède un filtre défini par une expression OCL. La propriété bidirectionnelle est précisée par un booléen. Deux convertisseurs de types, désignés respectivement par `typeconverterRL` et `typeconverterLR`, contiennent une expression OCL nécessaire pour une conversion complexe dans laquelle une navigation est requise. L'élément `typeconverterRL` permet la conversion d'un élément du métamodèle droit en un élément du métamodèle gauche. L'élément `typeconverterLR` permet la conversion d'un élément du métamodèle gauche en un élément du métamodèle droit. En général, il est uniquement nécessaire de préciser le `typeconverterLR`.
- `Left` identifie un élément du métamodèle gauche dans une correspondance.
- `Right` identifie un ou plusieurs éléments du métamodèle droit dans une correspondance.
- `MetaModelHandler` permet de naviguer dans un métamodèle. Il possède les informations nécessaires pour accéder et explorer un métamodèle participant dans une spécification de correspondances.
- `ElementHandler` permet l'accès aux éléments en correspondance sans les modifier.
- `TypeConverter` permet une conversion de type entre un élément de gauche et un élément de droite. Si un élément du métamodèle gauche et un élément du métamodèle droit sont égaux alors la correspondance est simple et direct. En revanche, si les deux éléments sont équivalents ou similaires alors il est nécessaire de spécifier la conversion entre les deux éléments en utilisant `TypeConverter`. Dans ce cas, une expression OCL permettra de spécifier cette conversion et sera utilisée par la suite pour transformer l'élément de gauche en élément de droite.

5.4 Un *plug-in* pour la modélisation de correspondances

Un outil conçu pour supporter la spécification de correspondances entre métamodèles doit fournir les caractéristiques suivantes :

- importation de métamodèles existants à partir d'un fichier XMI.
- visualisation graphique du modèle de correspondance et des métamodèles.
- édition de modèles de correspondance.
- vérification de conformité entre le modèle de correspondance et son métamodèle.
- représentation simplifiée d'un modèle de correspondance telle que la représentation textuelle.
- navigation entre les métamodèles mis en correspondance.
- *schema matching* semi-automatique.
- génération d'une définition de transformation à partir d'une spécification de correspondances (c.-à-d. modèle de correspondance).
- exportation d'un modèle de correspondance en utilisant un fichier XMI.

Actuellement, notre outil supporte toutes ces caractéristiques, à l'exception de *schema matching* semi-automatique dont l'intégration constitue la prochaine étape d'évolution.

5.4.1 Métamodèle de correspondance et notation graphique

La description d'une spécification de correspondances conforme avec notre métamodèle de correspondance doit avoir une notation graphique. Dans la section 4.2 (page 122), nous avons proposé une notation graphique pour la spécification de correspondances. Étant donné que les éléments de la notation graphique et les éléments du métamodèle de correspondance sont identiques, nous pouvons facilement lier cette notation graphique à ce métamodèle de correspondance.

Puisque ce métamodèle sera utilisé dans un plug-in d'« Eclipse », nous pouvons étendre cette notation graphique pour être conforme aux *widgets* fournis par l'environnement de développement « Eclipse ». Parmi les *widgets*, nous utilisons des arbres hiérarchisés pour faciliter la présentation d'un modèle de correspondance et des deux métamodèles participant à la correspondance. Nous utilisons aussi des formulaires avec des tables pour remplir les informations de chaque élément de correspondance. La figure 5.3 présente le métamodèle de correspondance et l'icône[2] représentative d'un élément. Selon cette figure, plusieurs éléments du métamodèle de correspondance ont une représentation graphique :

- `Historic` est représenté par un nœud d'un arbre et par un formulaire avec une table.
- `MappingDefinition` est représenté par un nœud d'un arbre et par un formulaire avec une table.
- `Correspondance` est représenté par un nœud d'un arbre et par un formulaire avec une table.
- `Left` est représenté par un nœud d'un arbre et par un formulaire avec une table.
- `Right` est représenté par un nœud d'un arbre et par un formulaire avec une table.

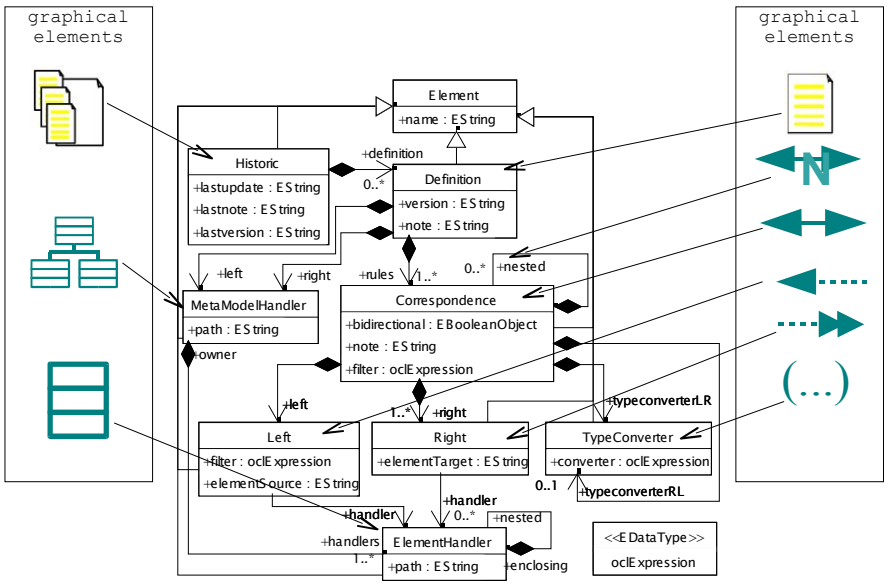


Figure 5.3 – Le métamodèle de spécification de correspondances et sa représentation graphique

Par rapport à la notation graphique présentée précédemment (section 4.2, page 122), nous avons remplacé le cercle (symbole d'un élément de correspondance) par une ligne fléchée en chaque extrémité.

[2]Une interface graphique plus élaborée peut être obtenue avec l'utilisation de GEF (*Graphical Editing Framework*) [61].

5.4.2 MMT (*Mapping Modeling Tool*)

La figure 5.4 montre notre *plug-in* en « Eclipse » [113] [114] nommé MMT (*Mapping Modeling Tool*). L'outil présente le premier métamodèle à gauche, le modèle de correspondance au centre et le second métamodèle à droite. Dans cette figure, les fragments des métamodèles d'UML et de Java sont mis en correspondance. L'éditeur de propriétés du modèle de correspondance est présenté au bas de la figure. Un développeur peut utiliser cet éditeur pour ajuster les propriétés d'un modèle de correspondance.

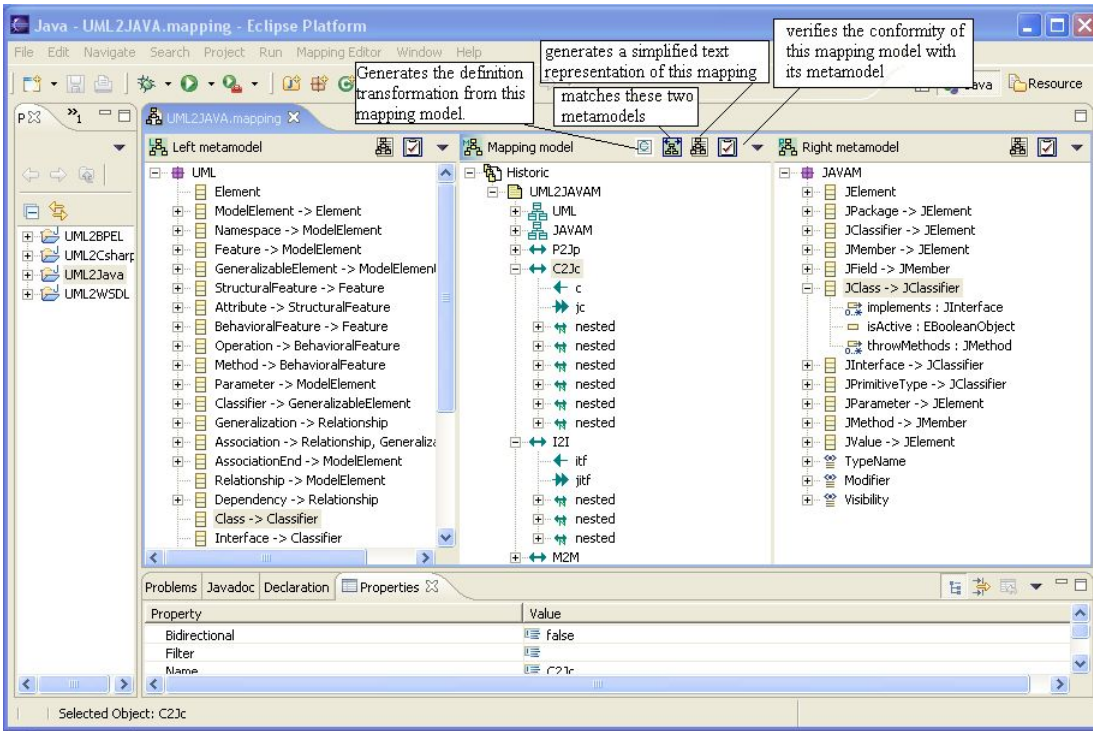


Figure 5.4 – La spécification de correspondances avec MMT (*Mapping Modeling Tool*)[114]

Ce *plug-in* est basé sur EMF (*Eclipse Modeling Framework*), un *framework* de modélisation qui facilite la génération de code permettant de créer des outils et des applications dirigés par les modèles [30]. En fait EMF et MDA sont des approches similaires du développement de systèmes logiciels, mais chacune utilise des technologies différentes. MDA a été initialement développée par l'OMG en utilisant MOF et UML, alors qu'EMF est basé sur le métamodèle Ecore et stimule la création des métamodèles spécifiques (DSL - *Domain Specific Language*) [51]. Dans le contexte d'« Eclipse », EMF est un *plug-in* supplémentaire pouvant être utilisé tel qu'il est ou étendu par d'autres *plug-ins*. *Plug-in* est un mécanisme permettant d'étendre l'architecture de base d'« Eclipse » avec des extensions qui partagent le même environnement [70].

Avant de spécifier les correspondances avec notre outil, nous avons besoin de créer des métamodèles conformes à Ecore. Quelques outils permettent l'édition de métamodèles conformes à Ecore tel que : Omondo [155] ou bien l'éditeur Ecore fourni par EMF [30].

Les étapes suivantes permettent d'illustrer l'utilisation de notre outil :

1. Nous créons un projet sous « Eclipse » et nous importons les métamodèles (UML et Java) dans ce projet. Nous utilisons un guide (*wizard*) pour créer le modèle de correspondances et nous choisis-

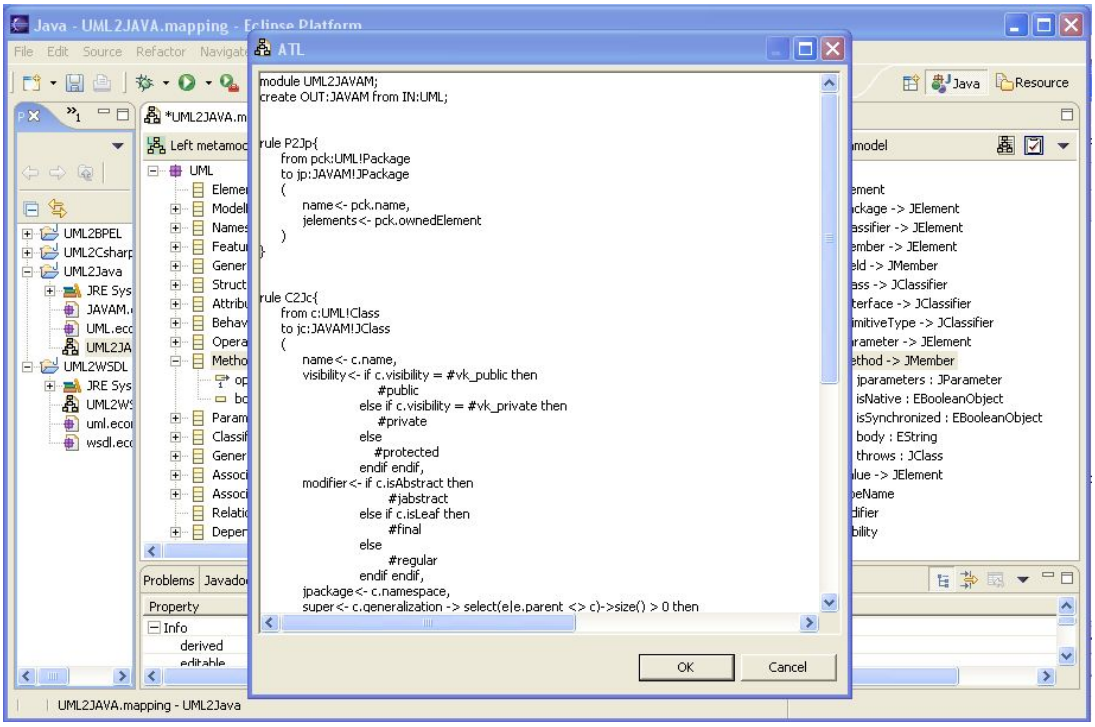


Figure 5.5 – La définition de transformation en ATL : de UML vers Java [114]

sons le nom du modèle de correspondances, le codage du fichier de correspondances (par exemple, Unicode et UTF- 8) et les fichiers des métamodèles au format XMI.

2. Les métamodèles à gauche (UML) et à droite (Java) sont chargés à partir de fichiers XMI, et le modèle de correspondances est créé. Il contient les éléments suivants : `Historic`, `Definition`, `left`, `right` et `MetamodelHandlers`. Pour chaque élément `MetamodelHandler`, des éléments `ElementHandlers` représentant des références aux éléments des métamodèles en correspondance sont créés.
3. Nous éditons le modèle de correspondance. Nous commençons d'abord par créer les correspondances entre les éléments des deux métamodèles. Ensuite, pour chaque correspondance, des correspondances imbriquées sont créées. Enfin, pour chaque correspondance imbriquée, on désigne un élément de gauche et un ou plusieurs éléments de droite. De plus, chaque élément de gauche ou droite a un `ElementHandler`. Si nécessaire, l'élément `TypeConverter` est utilisé pour expliciter une conversion entre les éléments en correspondance par le biais d'une expression OCL.
4. Le modèle de correspondances peut être validé conformément à son métamodèle et peut être utilisé pour générer une définition de transformations, en utilisant ATL par exemple.

La figure 5.5 illustre la définition de transformation en ATL d'UML vers Java, obtenue à partir du modèle de correspondance défini précédemment. Ce fragment de code ATL présente la définition d'un module `UML2JAVAM`, la règle `P2Jp` et la règle `C2Jc`. Dans le corps de la règle `P2Jp` l'élément source `Package` d'UML et l'élément cible `JPackage` de JAVAM apparaissent, ainsi que les bindings entre l'attribut `name <- pck.name` et la référence `jelements <-pck.ownedElement`. Dans le corps de la règle `C2Jc` l'élément source `Class` d'UML et l'élément cible `JClass` de JAVAM apparaissent, ainsi que les *bindings* entre les attributs, par exemple `name <- class.name`, et les références.

L'annexe F présente ces étapes de manière plus détaillée.

5.4.3 Modélisation et implémentation du prototype

L'outil MMT a été conçu en respectant le cahier de charges présenté précédemment (section 5.4). Selon ce cahier de charge, nous proposons initialement le diagramme de cas d'utilisation de la figure 5.6.

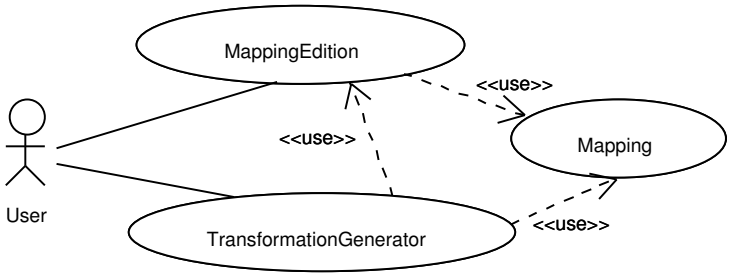


Figure 5.6 – Le diagramme de cas d'utilisation: MMT (fragment)

Ce diagramme contient trois cas d'utilisation :

- `MappingEdition` : représente l'interface home-machine.
- `Mapping` : représente le métamodèle de correspondance et les fonctionnalités fournies pour permettre l'édition de correspondances.
- `TransformationGenerator` : représente la fonctionnalité de génération d'une définition de transformations à partir d'une spécification de correspondances.

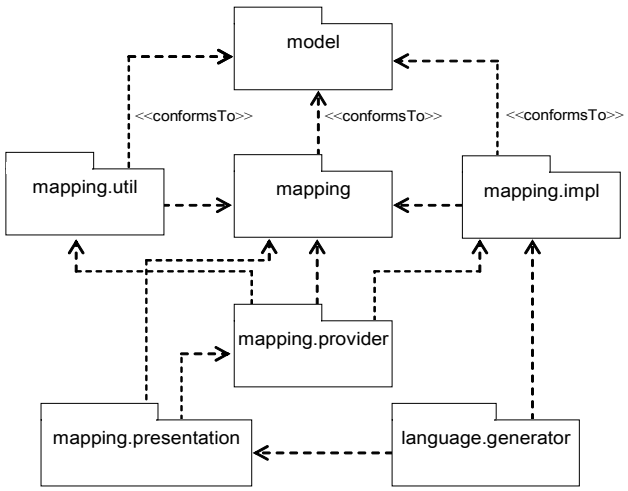


Figure 5.7 – Organisation du modèle MMT (fragment)

À partir du diagramme de cas d'utilisation, nous avons organisé la conception de notre outil MMT en sept paquetages, illustré par la figure 5.7. Ainsi, nous avons les cas d'utilisation et leur paquetages associés :

- Le cas d'utilisation `Mapping` a donné origine à cinq paquetages : `model`, `mapping.util`, `mapping`, `mapping.impl` et `mapping.provider`.

- Le cas d'utilisation `MappingEdition` a donné origine au paquetage `mapping.presentation`.
- Le cas d'utilisation `TransformationGenerator` a donné origine au paquetage `language.generator`.

La figure 5.8 détaille les paquetages « model » et « mapping ».

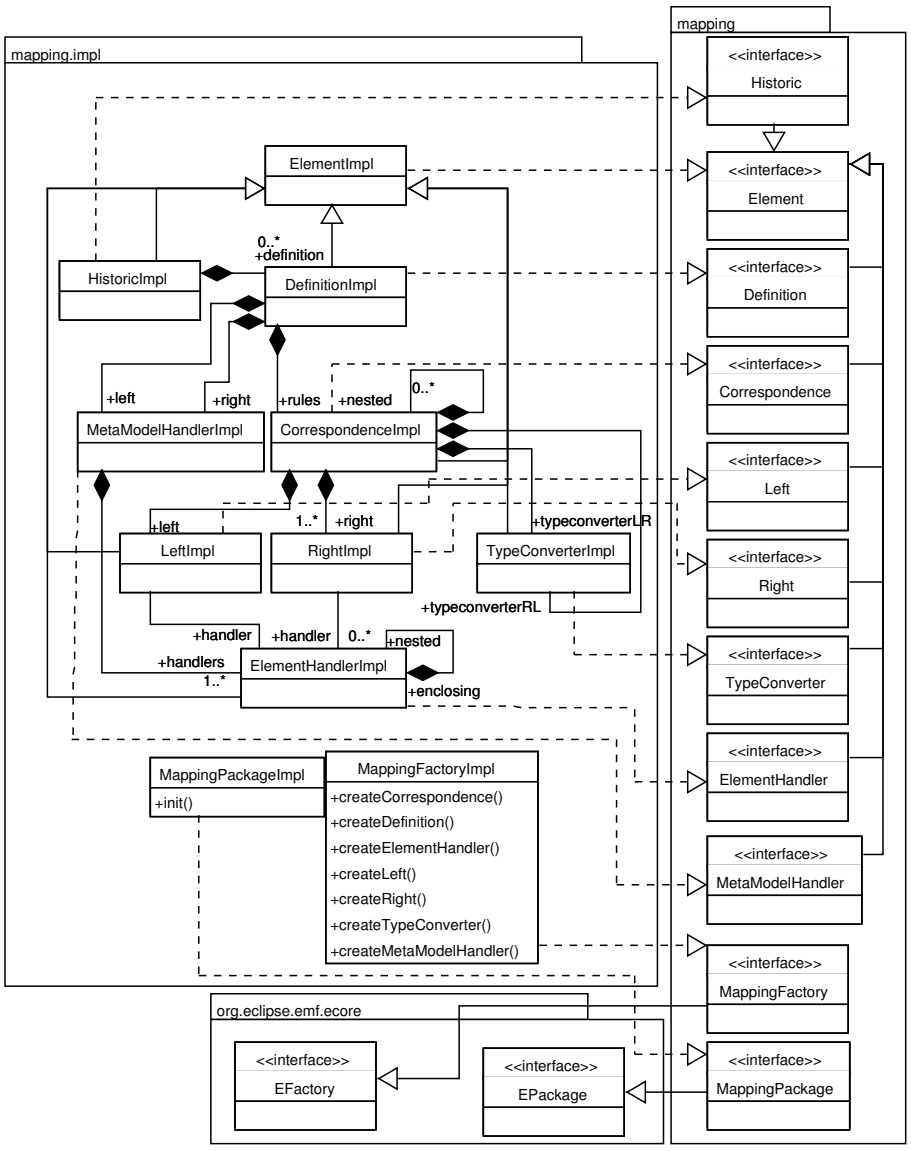


Figure 5.8 – Les paquetages « mapping.impl » et « mapping » (fragment)

Le paquetage `model` contient le metamodèle de spécification de correspondances (section 5.3, figure 5.2). Ce métamodèle conforme à Ecore est utilisé pour générer le fichier « mapping.genmodel » nécessaire pour que EMF puisse créer une version initiale des paquetages `mapping.util`, `mapping`, `mapping.impl` et `mapping.provider`. Ces paquetages sont modifiés ensuite pour répondre à nos besoins.

L'édition de modèles conformes au métamodèle de spécification de correspondances est possible par

le biais d'interfaces et de classes qui implémentent ce métamodèle.

Le paquetage `mapping` contient les interfaces, par exemple, `Historic`, `Element`, `Definition` et `Correspondences`, nécessaires pour la manipulation du métamodèle de correspondances. Les interfaces `MappingFactory` et `MappingPackage` permettent d'accéder à l'implémentation du paquetage du métamodèle de correspondances et aussi au *factory* qui sont nécessaires pour la manipulation des éléments (comme classes en Java) de ce métamodèle.

Le paquetage `mapping.impl` contient les classes `ElementImpl`, `DefinitionImpl`, `CorrespondenceImpl` qui font l'implémentation concrète du métamodèle de correspondances. La classe `MappingPackageImpl` permet la manipulation du paquetage du métamodèle de correspondances, et `MappingFactoryImpl` permet la création des éléments de ce métamodèle. Par exemple, `MappingFactoryImpl` a les méthodes `createCorrespondance()` et `createLeft()`.

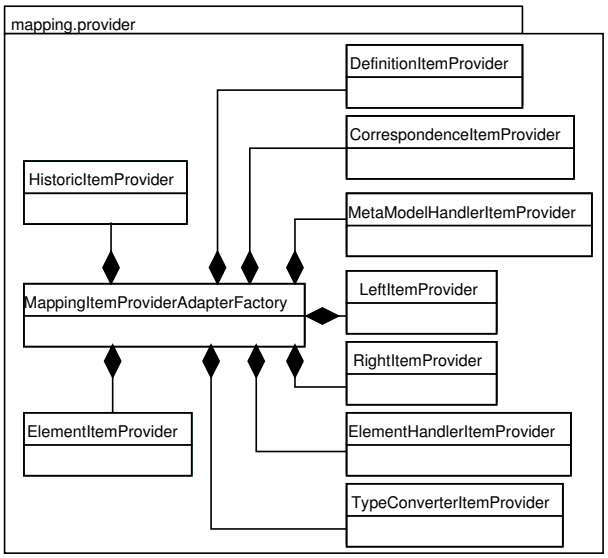


Figure 5.9 – Le paquetage « mapping.provider » (fragment)

Le paquetage `mapping.provider` (cf. figure 5.9) contient les classes nécessaires pour l'édition d'un modèle de correspondances conforme au métamodèle de spécification de correspondances. La classe `MappingItemProviderAdapterFactory` permet la manipulation du métamodèle de correspondances. Cette classe contient d'autres classes telles que `HistoricItemProvider`, `DefinitionItemProvider` et `CorrespondenceItemProvider`. Ces classes fournissent des méthodes nécessaires pour la manipulation d'« icones », de textes et d'autres propriétés liées à l'édition d'un modèle de correspondance.

Le paquetage `mapping.presentation` (cf. figure 5.10) fournit les classes nécessaires pour l'interface homme-machine, le contrôle des événements, l'affichage de modèles et de métamodèles sous la forme d'arbres, et les interfaces requises pour appeler les implémentations des générateurs d'une définition de transformation.

La classe `MappingModelWizard` implémente le guide (*wizard*) qui dirige la création d'un projet de correspondances (cf. annexe F, figures F.1, F.2 et F.3).

La classe `MappingActionBarContributor` permet l'édition et la gestion d'un modèle de correspondance, par exemple, la création et suppression d'un élément (cf. annexe F, figure F.5).

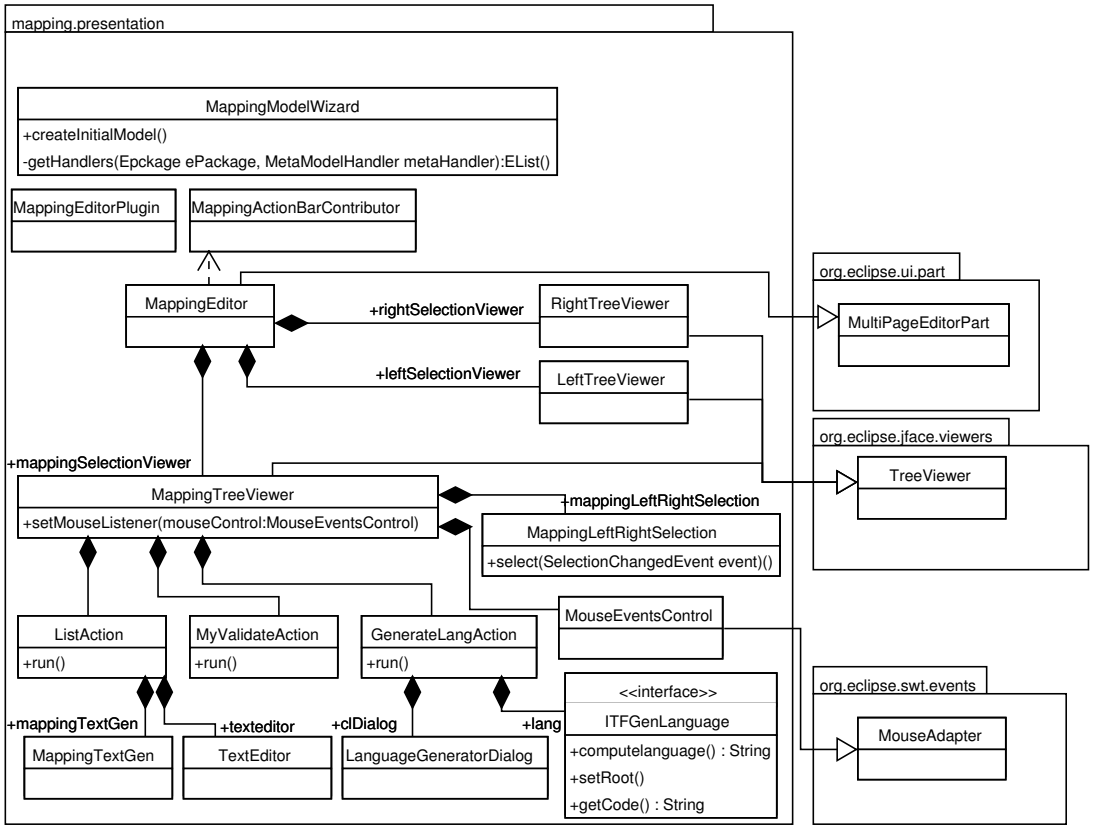


Figure 5.10 – Le paquetage « mapping.presentation » (fragment)

La classe `MappingEditor` contient des attributs et des méthodes nécessaires pour la création de l'interface homme-machine. Par exemple, elle contient les classes `MappingTreeViewer`, `LeftTreeViewer` et `RightTreeViewer`. La première classe permet la visualisation d'un modèle de correspondances sous forme d'arbre. Les deux autres classes permettent la visualisation de métamodèles (gauche et droite).

Les classes `ListAction`, `MyValidateAction` et `GenerateLangAction` réalisent respectivement les actions de présentation d'un modèle de correspondance dans un format textuel, validation d'un modèle de correspondance, et la création d'une définition de transformation.

Les classes `MappingLeftRightSelection` et `MouseEventsControl` contrôlent les événements pendant l'exécution pour rendre possible l'interaction entre un modèle de correspondances et les métamodèles (gauche et droite).

L'interface `ITFGenLanguage` doit être implémentée par un générateur de définition de transformation, par exemple celui d'ATL tel que présenté par la figure 5.11.

Les générateurs de définition de transformation sont créés indépendamment de notre métamodèle et *plug-in*. L'unique exigence à respecter est l'implémentation de l'interface `ITFGenLanguage` (cf. figure 5.11). Ainsi les méthodes `computelanguage()`, `setRoot(object :Object)` et `getCode()` doivent être implémentées par la classe qui réalise un générateur de définition de transformation.

Un générateur devient reconnu par notre *plug-in* par l'insertion des informations du nom de la classe et du titre du générateur dans le fichier « plugin.properties » tel qu'illustré comme suit :

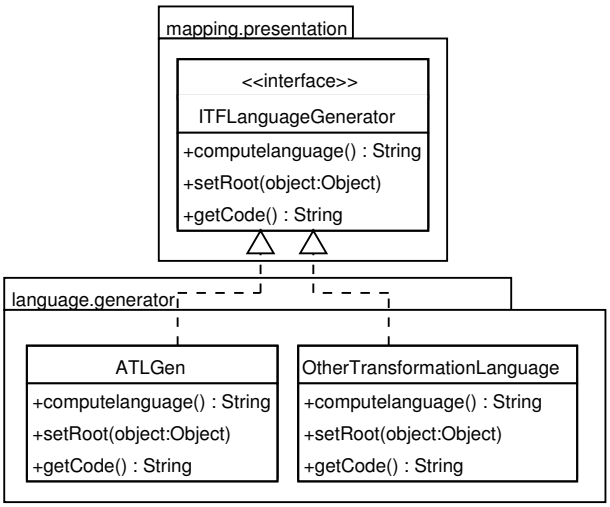


Figure 5.11 – Le paquetage « language.generator » (fragment)

```
_UI_Languages=ATL YATL OTHER
_UI_Languages_classes=language.generator.ATLGen
```

5.5 Application aux Services Web

Conformément à la figure 5.12, la correspondance `C2S` associe l'élément `Class` d'UML aux cinq éléments `PortType`, `Binding`, `SoapBinding`, `Port` et `Service`.

Notre outil permet de générer les règles de transformations à partir du modèle de correspondance. Actuellement, nous avons implémenté un générateur pour ATL. Une partie du code en ATL de la transformation entre les métamodèles d'UML et WSDL est présentée dans la figure 5.13. Dans cette figure, un fragment du code ATL est présenté comme suit :

```
module uml2wsdl; rule P2D{...} rule C2S{...}
```

La Figure 5.14 illustre la correspondance entre le métamodèle d'UML et celui de BPEL4WS. Dans cette figure, l'élément de correspondance `Ag2P` associe l'élément `ActivityGraph` d'UML à l'élément `Process` de BPEL4WS.

Un fragment du code ATL généré à partir de la correspondance entre les fragments des métamodèles d'UML et BPEL4WS est présenté sur la figure 5.15. Dans cette figure, un fragment du code ATL est présenté comme suit :

```
rule Ag2P{...} rule P2S{...}
```

Dans notre approche, le code ATL est généré automatiquement à partir du modèle de correspondance. Notre outil libère le développeur des tâches fastidieuses de génération de codes, l'aidant à spécifier uniquement les correspondances entre éléments de deux métamodèles.

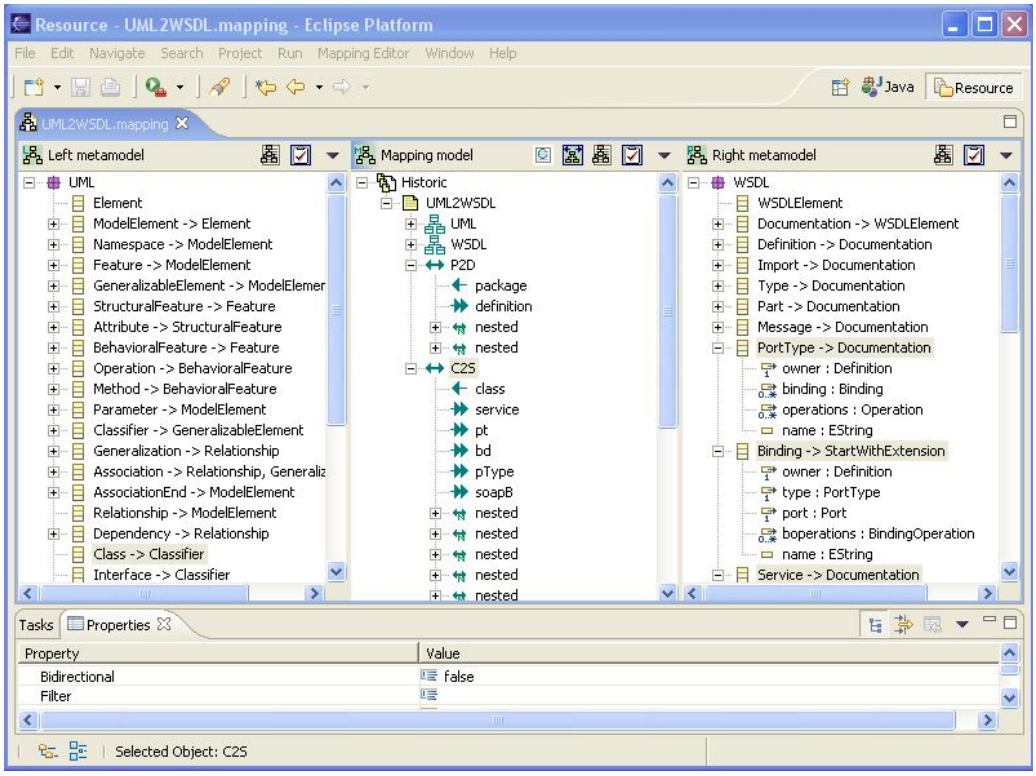


Figure 5.12 – La spécification de correspondances: d'UML à WSDL [113]

5.6 Apports de notre outil

L'outil MMT repose sur les apports suivants :

- La séparation de préoccupations :
  - La spécification de correspondances est un PIM.
  - La définition de transformations est un PSM.
- « L'indépendance » par rapport à un langage de transformation. À partir d'un modèle de correspondances, une définition de transformations peut être générée en plusieurs langages de transformations.
- Un atout pour l'évolution et la préservation d'un modèle de correspondances (c.-à-d., la logique métier d'établir les relations entre métamodèles) qui peut évoluer si les métamodèles changent.
- Une approche plus intuitive et plus simple que le codage direct d'une définition de transformations.

La séparation explicite entre spécification de correspondances et définition de transformations est en conformité avec l'approche dirigée par les modèles qui a comme objectif principale la séparation des préoccupations.

5.7 D'autres travaux

Nous pouvons citer « Clio » [10] [167], « Tess » (*Type Evolution Software System*) [106] et RVM (*Rimu Visual Mapper*) [79] comme exemples représentatifs d'outils qui supportent la création et l'édition

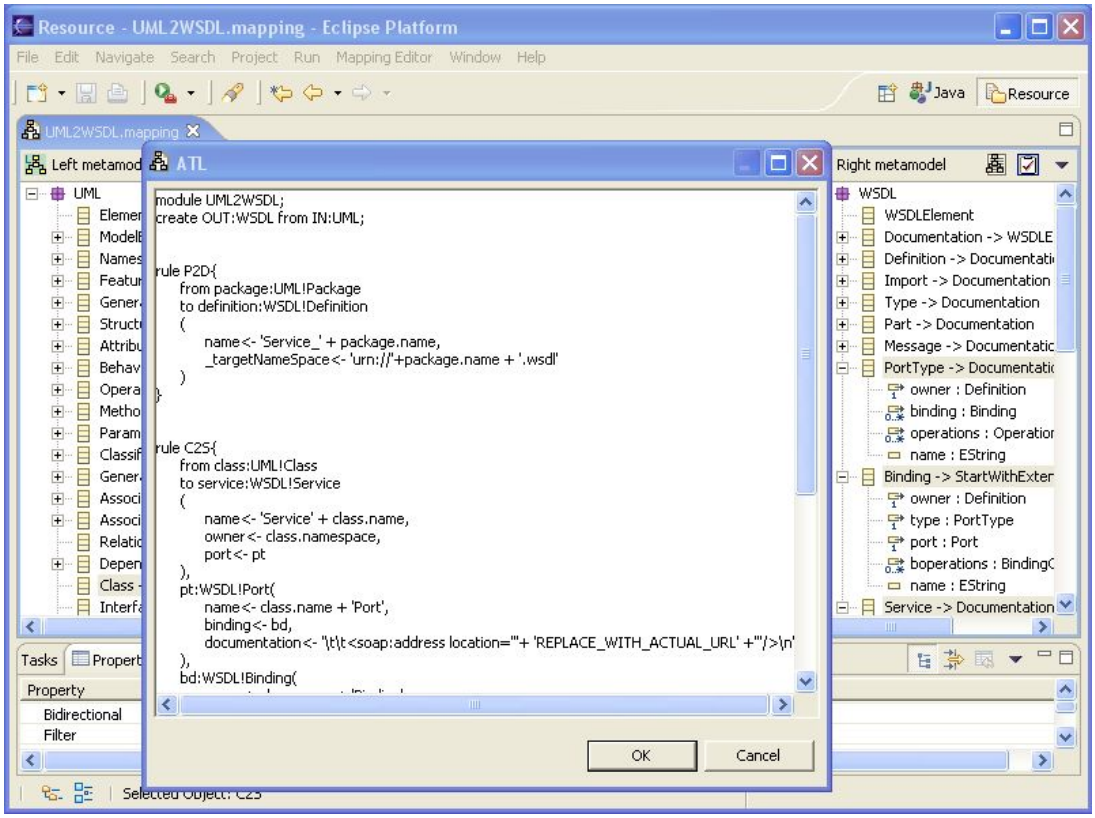


Figure 5.13 – Transformation d'UML vers WSDL : code ATL généré [113]

de correspondances.

« Clio » est un outil développé par IBM Almaden Research Center (http://www.almaden.ibm.com/software/km/clio/index.shtml). Il s'agit d'un système pour la gestion de données dont le but est de transformer et d'intégrer des données hétérogènes. Il supporte la génération et la gestion des schémas, les correspondances entre schémas et les requêtes entre schémas. Il est composé de *schema engine*, *correspondence engine* et *mapping engine*.

Une session typique avec « Clio » commence par le chargement d'un ou plusieurs schémas dans le système. Ces schémas sont lus à partir de différentes sources telle qu'une base de donnée objet-relationnel, *legacy systems*, ou d'un fichier XML et son schéma. Dans le cas où les schémas sont incomplets, le *schema engine* augmente son contenu par des contraintes additionnelles selon les besoins.

Étant donné une paire de schémas, le *correspondence engine* génère un ensemble de correspondances entre leurs éléments. Les correspondances peuvent, ensuite, être manipulées en utilisant le *Schema View* (qui fournit une représentation des schémas) ou le *Data View* (qui fournit une « interface WYSIWYG[3] » montrant les données des attributs utilisés dans les correspondances).

Mapping Engine supporte la création, l'évolution et la maintenance de *mappings* [4] entre les paires

[3]WYSIWYG est l'acronyme de la locution anglaise *What You See Is What You Get* signifiant littéralement « ce que vous voyez est ce que vous obtenez » ou plus élégamment « tel écran, tel écrit » (source Wikipédia - http://fr.wikipedia.org/wiki/WYSIWYG).

[4]*mappings* est un ensemble de requêtes d'un schéma source à un schéma cible qui réalise la transformation d'un schéma source dans un cible.

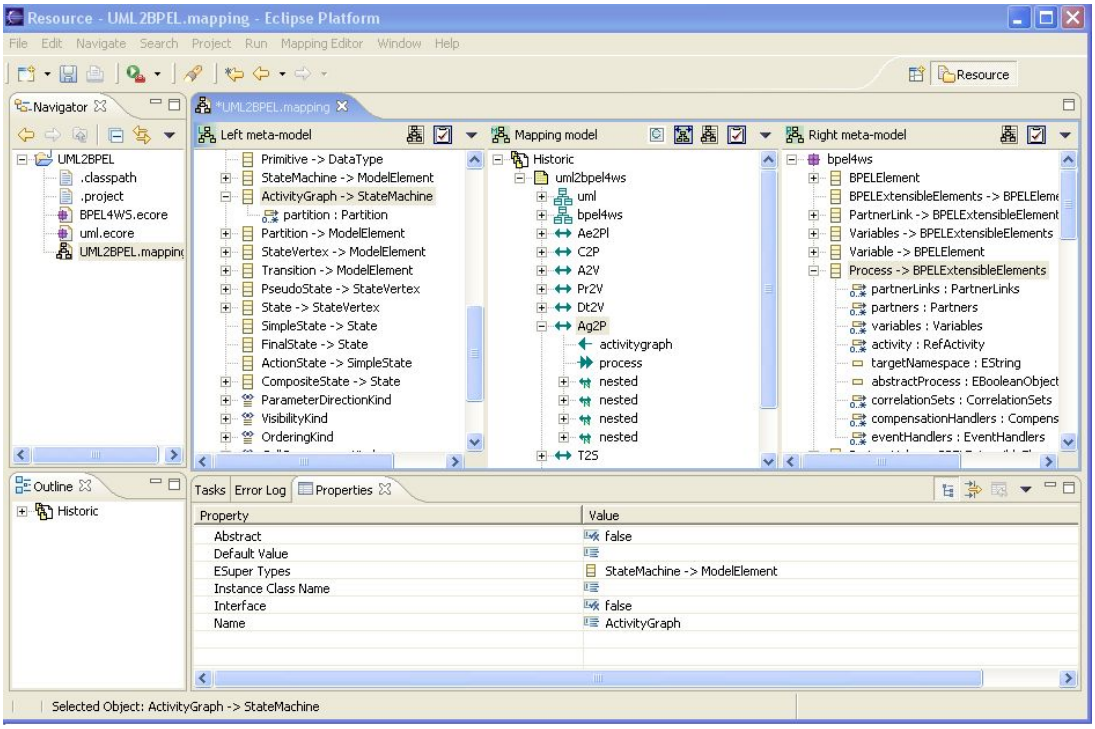


Figure 5.14 – La spécification de correspondances : d'UML à BPEL4WS [113]

de schémas.

« Tess » est un outil développé dans l'Université de Massachusetts (Amherst). Il supporte l'évolution de schémas. Dans ce cas, un schéma est un ensemble de types. « Tess » prend en entrée un vieux et un nouveau schéma, et produit un programme pour transformer les données du vieux schéma dans le nouveau schéma. Pour créer un programme de transformation de schémas, il crée automatiquement un *mapping* entre les deux schémas, en utilisant un algorithme de *schema matching*.

RVM fournit le support pour la création et l'édition de spécification de correspondances (*mappings*) entre schémas. Étant donné qu'une spécification de correspondances est complète, cet outil génère un programme traducteur qui est ensuite compilé dans un *tree-structured byte code*. Ce code est interprété par le *mapping engine* de RVM.

Le domaine de bases de données est sans doute un des premiers à aborder le problème de spécification de correspondances et de définition de transformations entre modèles (schémas). Notre outil, MMT, fournit plusieurs caractéristiques présentes dans « Clio », « Tess » et RVM, telle que la création et l'édition des spécifications de correspondances. De plus, MMT génère une définition de transformation à partir d'une spécification de correspondances. La génération semi-automatique des spécifications de correspondances (c.-à-d. *schema matching*) est sans doute une fonctionnalité indispensable pour MMT qui pourra bénéficier des approches utilisées par « Clio », « Tess » et autres [172].

5.8 Perspectives

Parmi les améliorations possibles de notre prototype, nous pouvons citer :

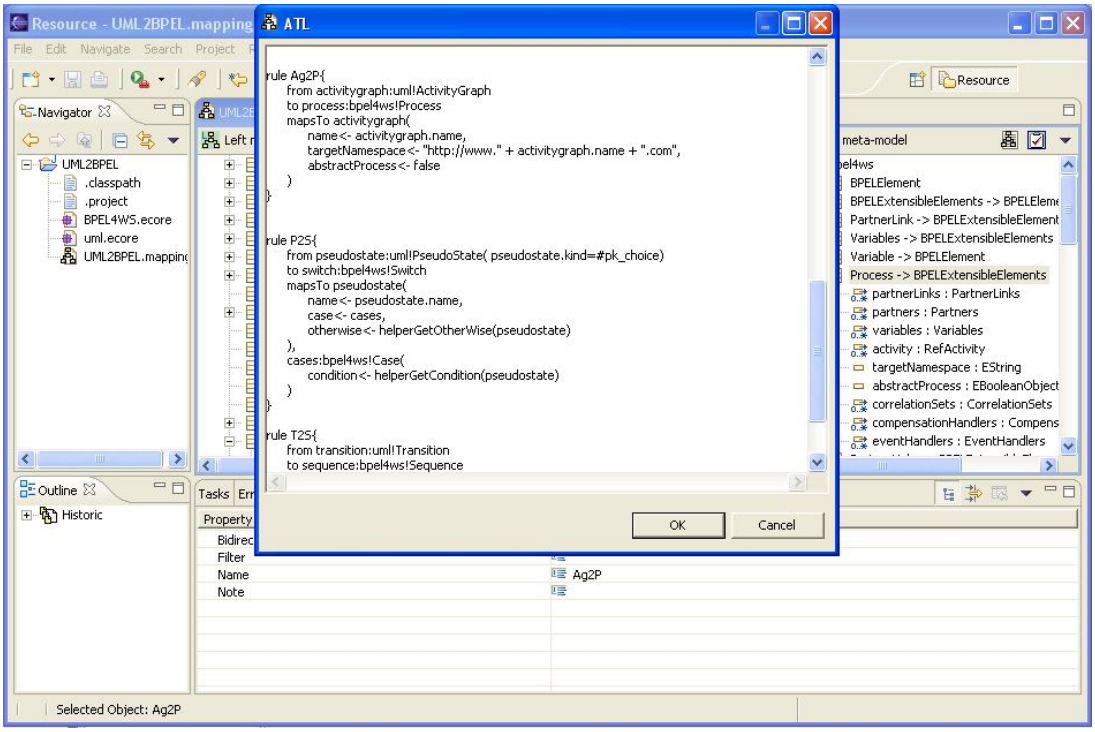


Figure 5.15 – Transformation d'UML vers BPEL4WS : code ATL généré

- L'intégration d'autres générateurs de définition de transformation, comme [161] pour YATL ou le futur langage de transformation standardisé de l'OMG [142].
- La conception et le développement d'un autre plug-In pour réaliser le *schema matching* de manière semi-automatique afin de créer des correspondances.
- La création d'une deuxième interface graphique plus intuitive, en proposant la visualisation complète de métamodèles et modèles de correspondances, de façon similaire à notre notation graphique présentée dans le chapitre 4 (section 4.2, page 123).

Parmi les utilisations de notre prototype, nous pouvons citer :

- La création de spécifications de correspondances entre UML et IDL-CORBA, EDOC et IDL-CORBA, UML et JXTA [97], et leur utilisation pour générer la définition de transformation en ATL.
- L'application à d'autres domaines comme les bases de données pour supporter l'évolution des schémas [201].

5.9 Synthèse

Dans ce chapitre, nous avons proposé un métamodèle et un plug-in en « Eclipse » pour supporter la création de spécification de correspondances entre métamodèles. La modélisation de notre outil a illustré en détail leur conception, structure et principales classes et interfaces.

Pour valider cet outil, nous l'avons expérimenté, par exemple, sur les plates-formes de Services Web (WSDL et BPEL) en partant d'UML comme PIM.

La génération d'une définition de transformations à partir d'une spécification de correspondances représente une des plus importantes fonctionnalités de notre outil. Parmi les améliorations de notre outil, nous mettons l'accent sur l'intégration du *schema matching* semi-automatique.

Enfin, ce prototype permet de valider plusieurs hypothèses (par exemple, la génération d'une définition de transformations à partir d'une spécification de correspondances) discutées dans les chapitres précédents.

Conclusion

Dans notre travail de thèse, nous avons étudié et appliqué l'approche MDA pour des plates-formes de services Web.

Dans notre démarche, nous avons utilisé une étude de cas, modélisée en UML et EDOC, puis nous avons choisi les plates-formes cibles : services Web, J2EE et dotNET. Ensuite, nous avons proposé des spécifications de correspondances et des définitions de transformations. Cette expérimentation nous a permis d'identifier d'autres problèmes non prévus initialement pour lesquels nous avons proposé des solutions.

Étant donné que la création manuelle de spécifications de correspondances et de définitions de transformations est une source d'erreurs et un processus fastidieux, nous avons proposé un métamodèle de correspondance et un outil pour prendre en compte, d'une part, la création et l'édition de correspondances et, d'autre part, la génération automatique de définitions de transformations.

Problématique

La mise en œuvre de l'approche MDA pour développer des applications orientées Internet sur des plates-formes de services Web, nous a amenés à étudier les problèmes suivants :

- La plate-forme des Services Web dans le contexte de MDA.
- La transformation de modèles métier en modèles de la plate-forme de services Web.
- L'impact d'un choix de langages de modélisation et de plates-formes cibles dans la modélisation d'un métier, ainsi que dans l'approche de transformation de modèles.

Au cours de notre étude, nous avons perçu l'importance d'étudier d'autres problèmes qui devraient être surmontés pour réaliser notre approche :

- L'utilisation de bibliothèques dans une approche dirigée par les modèles.
- Les différentes possibilités de créer un métamodèle pour une plate-forme, et les conséquences de placer une caractéristique déterminée de cette plate-forme dans les niveaux métamodèle ou modèle.
- La différence sémantique entre métamodèles et leur manipulation afin d'achever l'approche MDA.
- L'automatisation de la génération de la définition de transformations à partir de la spécification de correspondances.

Contributions

Cette thèse fournit les contributions suivantes :

- L'étude de la plate-forme des Services Web dans le contexte de l'approche MDA. Dans cette étude, nous avons utilisé l'approche MDA et les services Web pour développer l'étude de cas de l'agence de voyages. Nous avons commencé par la modélisation (utilisant UML et EDOC), puis nous avons utilisé des définitions de transformations en ATL pour générer une partie de la plate-forme finale (services Web, J2EE et dotNET).

- Une méthodologie pour mettre en œuvre l'approche MDA. Dans cette méthodologie, la transformation de modèles est divisée en deux étapes : la spécification de correspondances et la définition de transformations. Dans la spécification de correspondances, nous nous sommes concentrés sur le problème d'identification des éléments correspondants. Dans la définition de transformation, nous avons décrit les étapes opérationnelles d'une transformation. Dans cette perspective, la spécification de correspondance peut être vue comme un PIM et la définition de transformation comme un PSM. Ceci est en accord avec l'approche MDA qui vise la séparation des préoccupations (concernes).
- Une proposition de métamodèles pour la plate-forme de services Web, J2EE et dotNET. Ces métamodèles ont été créés à partir de schémas, DTD ou grammaires EBNF.
- Des spécifications de correspondances entre les métamodèles UML et EDOC (comme PIM) et les métamodèles des Services Web, J2EE et dotNET (comme PSM). La génération des définitions de transformations de modèles en ATL est conforme à ces spécifications.
- Une étude de l'impact d'un langage de modélisation (UML versus EDOC) et d'une plate-forme cible (J2EE versus dotNET) dans une approche dirigée par les modèles, nous a permis de montrer qu'un langage spécifique au domaine peut fournir plus de bénéfices qu'un langage généraliste, et qu'une plate-forme a peu d'impact dans une approche MDA.
- Une conception et une implémentation d'un outil comme un *plug-in* en « Eclipse » permettant d'une part, de spécifier la correspondance entre métamodèles et, d'autre part, de générer la définition de transformations à partir de cette spécification de correspondances.
- Une énumération de difficultés et une proposition de solutions pour passer d'une abstraction à une implémentation (transformation PIM vers PSM). Par exemple, l'utilisation de bibliothèques dans le niveau correspondant au modèle. Ceci démontre que l'approche MDA ne reste pas que dans le monde abstrait, mais s'approche également du monde réel. D'autre part, la différence sémantique entre UML et BPEL a été réduite grâce à l'utilisation de valeurs marquées d'UML.

Critique de notre approche : apports et limitations

Tout d'abord, nous présentons un résumé des points forts et faibles fournis par des lecteurs (*reviewers*) des différentes conférences dans lesquelles nous avons soumis nous travaux. Enfin, nous faisons le point sur les limitations de notre approche.

Les points forts de notre approche :

- Les expérimentations réalisées avec UML, EDOC, les Services Web et J2EE démontrent comment une approche MDA peut être réalisée dans la pratique.
- L'utilisation de différents formalismes de modélisation tel que UML (formalisme généraliste) et EDOC (formalisme spécialisé aux systèmes distribués) est une bonne démarche pour valider l'approche MDA. De plus, ceci a permis d'évaluer les implications du choix d'un formalisme déterminé (langage de modélisation) sur une approche dirigée par les modèles .
- L'identification des difficultés dans l'approche MDA due à la distance sémantique entre métamodèles est un obstacle qui nécessite d'être surmonté. Cette distance sémantique peut rendre difficile la création de spécifications de correspondances. Par conséquent, la génération de définitions de transformations aboutit à un code plus impératif et récursif au détriment d'un code déclaratif recherché.
- La représentation graphique de correspondances permet une compréhension plus simple qu'une

notation textuelle sur les relations entre métamodèles et sur les passages d'un modèle vers d'autres.
- Le travail présente un métamodèle de correspondances dans le contexte de l'approche MDA. Ensuite, ce métamodèle est implémenté comme un *plug-in* en Eclipse et sa validation a été démontrée par des exemples.
- La création de définition de transformation à partir d'une spécification de correspondances est automatisée et validée par un *plug-in* en Eclipse.
- Une séparation claire est établie et maintenue entre la spécification de correspondances (entre les concepts appartenant à deux métamodèles) et la transformation de ces concepts (dans le niveau de modèles).
- Le métamodèle pour la spécification de correspondances est très intéressant car il permet au lecteur de comprendre comment les correspondances et les transformations sont présentées dans ce travail.

Les points faibles de notre approche :

- Le travail ne fournit pas une comparaison directe (quantitative et qualitative) entre le métamodèle de correspondances proposé et d'autres métamodèles de correspondances. Une comparaison (quantitative dans le sens de *benchmark*) avec d'autres outils n'est pas non plus proposée.
- Il n'existe pas de quantification de la distance sémantique entre métamodèles.
- Dans le cas d'UML, les profils peuvent être utilisés pour diminuer la distance sémantique entre UML et un autre métamodèle. Cependant, ce travail ne fournit pas de recommandations pour obtenir une solution plus généraliste qui puisse être appliquée à d'autres métamodèles sans un mécanisme d'extension.
- L'opérateur *match* peut retourner les correspondances entre les éléments de métamodèles. Ce point est discuté dans ce travail, mais il n'est pas encore implémenté dans l'outil. Une correspondance peut dépendre des décisions humaines qui sont basées sur des contraintes de conception. En outre, les correspondances entre les éléments de différents métamodèles ne sont pas orthogonales parce qu'il existe des dépendances entre elles. Ce point devra faire l'objet de travaux futurs.
- Les versions de métamodèles utilisées sont des simplifications, par exemple, le fragment du métamodèle d'UML, d'EDOC et des plates-formes cibles. Dans ce contexte, la réalisation industrielle de l'approche présentée peut être incomplète.
- Les exemples utilisés pour valider l'approche ne contiennent qu'une partie des aspects liés aux plates-formes. D'autres aspects tels que la sécurité et la disponibilité, ne sont pas discutés dans ce travail.

Les limitations de notre approche selon nos critiques :

- Dans nos expérimentations, nous avons choisi de modéliser la logique métier de notre exemple illustratif avec une forte granularité. De plus, les spécifications de correspondances ne prenaient pas en compte les corps des méthodes. Ainsi, la plate-forme finale n'a pas pu être complètement générée à partir de modèles.
- Le métamodèle de correspondance a été validé en utilisant UML, EDOC, les Services Web, J2EE et dotNET. Cependant, nous ne pouvons pas encore assurer qu'il est suffisant pour être appliqué comme tel avec d'autres formalismes de modélisation et d'autres plates-formes. Ainsi, une extension de notre métamodèle de correspondance pourrait être nécessaire.
- La spécification de correspondance entre UML (diagramme d'activité) et BPEL a montré l'impact de la distance sémantique entre métamodèles dans l'approche dirigée par les modèles. Nous avons utilisé des profils pour diminuer cette distance, mais cette solution a introduit d'autres problèmes tels que la complexité de la définition de transformation. D'autres solutions plus généra-

listes doivent être proposées afin de diminuer la distance sémantique.

Nous sommes conscients que notre approche a des points forts et ceci nous a encouragé à poursuivre son développement. Cependant, nous reconnaissons que les points faibles mentionnés par les lecteurs (*reviewers*) sont pertinents et doivent être pris en compte pour assurer la poursuite de notre approche et leur valorisation scientifique.

La comparaison de notre métamodèle de correspondance et de notre outil MMT avec d'autres métamodèles et d'autres outils n'a pas été possible, parce que les informations disponibles sur les autres métamodèles et sur les autres outils sont insuffisantes.

Les remarques de lecteurs nous ont donné des pistes de travaux futurs présentés dans la section suivante.

Perspectives

L'étude d'autres langages de modélisation tel que UEML, et d'autres plates-formes cibles telles que JXTA dans le contexte de MDA contribueront à renforcer notre approche.

L'utilisation d'*action semantics* pourrait permettre une faible granularité de la modélisation des systèmes informatiques. Ainsi, une spécification de correspondances pourra prendre en charge les corps des méthodes en UML et les mettre en relation avec leur équivalent ou similaire dans une plate-forme cible, tel que le langage Java. Par conséquent, la définition de transformation sera plus complète. L'utilisation d'*action semantics* pourra rendre possible la génération de presque 100% de la plate-forme cible.

Étant donné que la transformation de modèles est au cœur de l'approche MDA et que la transformation manipule[5] généralement des métamodèles, la distance sémantique entre métamodèles doit être plus étudiée dans le cadre de l'approche MDA.

Une des promesses de MDA est l'unification des différents espaces technologiques et leur harmonisation. Cependant, avant que l'harmonisation devienne une réalité courante, la distance sémantique entre les différents espaces technologiques doit également être gérée.

L'intégration des algorithmes de *schema matching* dans notre outil MMT (*Mapping Modeling Tool*) doit être envisagée au vu des remarques pertinentes et continues des lecteurs (*reviewers*).

L'utilisation de GEF (*Graphical Editing Framework*) [61] pourra être envisagé pour le perfectionnement de l'interface graphique de MMT.

L'introduction d'un mécanisme d'extension dans notre métamodèle de correspondance, tel que les valeurs marquées, doit permettre l'adaptation à un contexte donné.

Une étude plus détaillée entre les différents types de relations entre les éléments de correspondances, par exemple, l'association, la composition, l'agrégation et la dépendance devra être enfin menée de manière plus détaillée.

[5] Nous parlons de l'approche de transformation de métamodèles.

Publications

Cette section regroupe, par ordre chronologique inverse, les publications que nous avons produites pendant cette thèse.

Slimane Hammoudi, Denivaldo Lopes et Jean Bézivin, *Approche MDA pour le développement des e-applications sur la plate-forme des services Web*, Revue d'Ingénierie des Systèmes d'Information, *Special issue on: Web services, theory and applications*, Hermes science publications, 2005.

Denivaldo Lopes, Slimane Hammoudi, Jean Bézivin and Frédéric Jouault, *Generating Transformation Definition from Mapping Specification: Application to Web Service Platform*, The 17th Conference on Advanced Information Systems Engineering (CAiSE'05), LNCS 3520, 309-325, Porto, Portugal, June 13-17, 2005.

Denivaldo Lopes, Slimane Hammoudi, Jean Bézivin and Frédéric Jouault, *Mapping Specification in MDA: From Theory to Practice*, First International Conference INTEROP-ESA 2005 Interoperability of Enterprise Software and Applications, Geneva, Switzerland, February 21 - 25, 2005.

Jean Bézivin, Slimane Hammoudi, Denivaldo Lopes and Frédéric Jouault, *B2B Applications, BPEL4WS, Web Services and dotNET in the context of MDA*, International Conference on Enterprise Integration and Modelling Technology (ICEIMT2004), Canada, Toronto, October 9-11, 2004.

Jean Bézivin, Slimane Hammoudi, Denivaldo Lopes and Frédéric Jouault, *Applying MDA Approach for Web Service Platform*, The 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC2004), Monterey, USA, September 20-24, 2004.

Jean Bézivin, Slimane Hammoudi, Denivaldo Lopes and Frédéric Jouault, *Applying MDA Approach to B2B Applications: A Road Map*, Workshop on Model Driven Development (WMDD2004) - at ECOOP 2004, Oslo, Norway, June 15, 2004.

Jean Bézivin, Slimane Hammoudi, Denivaldo Lopes and Frédéric Jouault, An Experiment in Mapping Web Services to Implementation Platforms, Research Report, RR-IRIN2004-01, France, March 2004.

Denivaldo Lopes and Slimane Hammoudi, *Web Services in the Context of MDA*, The 2003 International Conference on Web Services (ICWS2003), Las Vegas, USA, June 23-26, 2003.

Bibliographie

[1] *AndroMDA*, december 2004. Disponible sur htpp://www.andromda.org/,

[2] Serge ABITEBOUL. Distributed Information Management with XML and Web Services. *7th International Conference Fundamental Approaches to Software Engineering (FASE 2004), Lecture Notes in Computer Science*, 2984:1–11, 2004.

[3] Serge ABITEBOUL, Omar BENJELLOUN, Ioana MANOLESCU, Tova MILO et Roger WEBER. Active XML: Peer-to-Peer Data and Web Services Integration. *VLDB 2002 - 28th International Conference on very Large Data Bases*, pages 1087–1090, August 2002.

[4] Serge ABITEBOUL, Omar BENJELLOUN, Ioana MANOLESCU, Tova MILO et Roger WEBER. *Active XML: a data-centric perspective on Web Services, Book chapter, In Web Dynamics*. Springer, March 2004.

[5] Aditya AGRAWAL, Tihamer LEVENDOVSZKY, Jon SPRINKLE, Feng SHI et Gabor KARSAI. Generative Programming via Graph Transformation in the Model-Driven Architecture. *OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture*, November 2002. Disponible sur http://www.softmetaware.com/oopsla2002/karsaig.pdf,

[6] João Paulo ALMEIDA, Remco DIJKMAN, Marten van SINDEREN et Luís Ferreira PIRES. On the Notion of Abstract Platform in MDA Development. *Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf (EDOC 2004)*, pages 253–263, September 2004.

[7] Gustavo ALONSO. Myths around Web Services. *Bulletin of the Technical Committee on Data Engineering - IEEE Computer Society*, 25(4):3–9, December 2002.

[8] Gustavo ALONSO, Fabio CASATI, Harumi KUNO et Vijay MACHIRAJU. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 1rst édition, 2004.

[9] Tony ANDREWS, Francisco CURBERA, Hitesh DHOLAKIA, Yaron GOLAND, Johannes KLEIN, Frank LEYMANN, Kevin LIU, Dieter ROLLER, Doug SMITH, Satish THATTE, Ivana TRICKOVIC et Sanjiva WEERAWARANA. *Business Process Execution Language for Web Services (BPEL4WS) version 1.1*, May 2003.

[10] Periklis ANDRITSOS, Ronald FAGIN, Ariel FUXMAN, Laura M. HAAS, Mauricio A. HERNANDEZ, Ching-Tien HO, Anastasios KEMENTSIETSIDIS, Renée J. MILLER, Felix NAUMANN, Lucian POPA, Yannis VELEGRAKIS, Charlotte VILAREM et Ling-Ling YAN. Schema Management. *In IEEE Data Engineering Bulletin*, 25(3):32–38, September 2002.

[11] Luis ANIDO, Manuel CAEIRO, Judith RODRIGUEZ et Juan SANTOS. Applying MDA Concepts to Develop a Domain CORBA Facility for E-learning. *5th International Unified Modeling Language (UML 2002)*, 2460:321–335, September-October 2002.

[12] Biju APPUKUTTAN, Tony CLARK, Sreedhar REDDY, Laurence TRATT et R. VENKATESH. A Model Driven Approach to Model Transformations. *Workshop on Model Driven Architecture: Foundations and Applications (MDAFA2003)*, June 2003. Disponible sur http://trese.cs.utwente.nl/mdafa2003/,

[13] Biju APPUKUTTAN, Tony CLARK, Sreedhar REDDY, Laurence TRATT et R. VENKATESH. A Pattern based Model-Driven Approach to Building Model Transformations. *Metamodelling for MDA 2003*, November 2003.

[14] ATL. *ATL Development Tools*. INRIA, LINA et Université de Nantes, december 2004. Disponible sur http://www.sciences.univ-nantes.fr/lina/atl/,

[15] Karim BAINA, Boualem BENATALLAH, Fabio CASATI et Farouk TOUMANI. Model-Driven Web Service Development. *Procs of 16th International Conference on Advanced Information Systems Engineering - CAiSE 2004, Lecture Notes in Computer Science*, 3084:290–306, June 2004.

[16] Siddharth BAJAJ, Giovanni DELLA-LIBERA, et ET AL. *Web Services Federation Language (WS-Federation) version 1.0*, July 2003. Disponible sur http://www-106.ibm.com/developerworks/webservices/library/ws-fed/,

[17] Luciano BARESI, Reiko HECKEL, Sebastian THUNE et Daniel VARRO. Modeling and Validation of Service-Oriented Architectures: Application vs. Style. *ESEC / SIGSOFT FSE 2003*, pages 68–77, 2003.

[18] Mariano BELAUNDE, Jean BÉZIVIN et Pham Thanh HA. Implementing EDOC Business Components on Top of a CCM Platform. *The 7th International IEEE Enterprise Distributed Object Computing Conference (EDOC'03)*, pages 208–221, September 2003.

[19] Boualem BENATALLAH, , Marlon DUMAS et Quan Z. SHENG. The SELF-SERV Environment for Web Services Composition. *IEEE Internet Computing*, pages 40–48, January 2003.

[20] Philip A. BERNSTEIN. Transaction Processing Monitors. *Comm. ACM*, 33(11):75–86, 1990.

[21] Philip A. BERNSTEIN. Applying Model Management to Classical Meta Data Problems. *Proceedings of the 2003 CIDR*, pages 209–220, January 2003.

[22] Jorn BETTIN, Ghica van EMDE BOAS et E.D. WILLINK. Generative Model Transformer: An Open Source MDA Tool Initiative. *Companion of the 18th annual ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2003)*, pages 294–295, October 2003.

[23] Andrew D. BIRELL et Bruce Jay NELSON. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, pages 39–59, February 1984.

[24] Grady BOOCH, Alan BROWN, Sridhar IYENGAR, James RUMBAUGH et Bran SELIC. An mda manifest. *MDA Journal*, May 2004.

[25] Behzad BORDBAR et Athanasios STAIKOPOULOS. Automated Generation of Metamodels for Web Service Languages. *Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations (EWMDA)*, September 2004.

[26] Behzad BORDBAR et Athanasios STAIKOPOULOS. Modeling and Transforming the Behavioural Aspects of Web Services. *WiSME@UML 2004*, October 2004.

[27] Behzad BORDBAR et Athanasios STAIKOPOULOS. On Behavioural Model Transformation in Web Services. *Proceeding of the 23rd International Conference on Conceptual Modeling (ER2004) and eCOMO 2004*, 3289:667–678, November 2004.

[28] Peter BRAUN et Frank MARSCHALL. *BOTL The Bidirectional Object Oriented Transformation Language, TUM-INFO-05-I0307-0/1.-FI.* Institut fur Informatik, Technische Universitat Munchen, May 2003.

[29] Mario BRAVETTI, Roberto LUCCHI, Gianluigi ZAVATTARO et Roberto GORRIERI. Web Services for E-commerce: Guaranteeing Security Access and Quality of Service. *Proceedings of the 2004 ACM symposium on Applied computing, ACM Press*, pages 800–806, 2004.

[30] Frank BUDINSKY, David STEINBERG, Ed MERKS, Raymond ELLERSICK et Timothy J. GROSE. *Eclipse Modeling Framework: A Developer's Guide*. Addison-Wesley Pub Co, 1st édition, August 2003.

[31] Geoof BULLEN, Ash NANGIA, Doug STEIN, Art HARKIN et ET AL. *Open Management Interface (OMI) Specification version 1.0*. OASIS. Disponible sur http://xml.coverpages.org/OMISpecification-10rev1-OASIS.pdf,

[32] Jean BÉZIVIN, Grégoire DUPÉ, Frédéric JOUAULT, Gilles PITETTE et Jamal Eddine ROUGUI. First Experiments with the ATL Model Transformation Language: Transforming XSLT into XQuery. *2nd OOPSLA Workhop on Generative Techniques in the context of Model Driven Architecture*, October 2003.

[33] Jean BÉZIVIN et Olivier GERBÉ. Towards a Precise Definition of the OMG/MDA Framework. *In Proceedings of the 16th International Conference on Automated Software Engineering*, (November):273–280, 2001.

[34] Jean BÉZIVIN, Slimane HAMMOUDI, Denivaldo LOPES et Frédéric JOUAULT. An Experiment in Mapping Web Services to Implementation Platforms, RR-LINA-2004-01. Rapport technique 01, University of Nantes, march 2004.

[35] Jean BÉZIVIN, Slimane HAMMOUDI, Denivaldo LOPES et Frédéric JOUAULT. Applying MDA Approach for Web Service Platform. *8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pages 58–70, September 2004.

[36] Jean BÉZIVIN, Slimane HAMMOUDI, Denivaldo LOPES et Frédéric JOUAULT. Applying MDA Approach to B2B Applications: A Road Map. *Workshop on Model Driven Development (WMDD 2004) in conjunction with ECOOP 2004*, June 2004.

[37] Jean BÉZIVIN, Slimane HAMMOUDI, Denivaldo LOPES et Frédéric JOUAULT. B2B Applications, BPEL4WS, Web Services and dotNET in the context of MDA. *International Conference on Enterprise Integration and Modeling Technology - ICEIMT 2004*, October 2004.

[38] Jean BÉZIVIN, Frédéric JOUAULT et Patrick VALDURIEZ. First Experiments with a ModelWeaver. *OOPSLA 2004 - Workshop on Best Practices for Model Driven Software Development*, 2004.

[39] Jean BÉZIVIN, Frédéric JOUAULT et Patrick VALDURIEZ. On the Need for Megamodels. *OOPSLA 2004 - Workshop on Best Practices for Model Driven Software Development*, 2004.

[40] Jean BÉZIVIN et Nicolas PLOQUIN. Tooling the MDA Framework: a new Software Maintenance and Evolution Scheme Proposal. *Journal of Object-Oriented Programming (JOOP)*, December 2001. Disponible sur http://www.adtmag.com/joop/article.asp?id=5736'&mon=12&yr=2001,

[41] Luis Ferreira CABRERA, George COPELAND et ET AL. *Web Services Atomic Transaction (WS-Atomic Transaction)*, November 2004. Disponible sur ftp://www6.software.ibm.com/software/developer/library/WS-Ato% -micTransaction.pdf,

[42] Luis Ferreira CABRERA, George COPELAND et ET AL. *Web Services Coordination (WS-Coordination)*, November 2004. Disponible sur ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf,

[43] Guy CAPLAT et Jean Louis SOURROUILLE. Model Mapping in MDA. *Workshop in Software Model Engineering (WISME2002)*, 2002.

[44] Guy CAPLAT et Jean Louis SOURROUILLE. Considerations about Model Mapping. *Workshop in Software Model Engineering*, October 2003.

[45] CARL-OLAV. MDA as Framework for Supporting Enterprise Standards. *Practitioner's Reports - ECOOP2004*, June 2004.

[46] Mark CARMAN, Luciano SERAFINI et Paolo TRAVERSO. Web Service Composition as Planning. *Proceedings of ICAPS'03 Workshop on Planning for Web Services*, June 2003.

[47] Paloma CÁCERES, Esperanza MARCOS et Belén VELA. A MDA-Based Approach for Web Information System Development. *Workshop in Software Model Engineering*, October 2003.

[48] Girish B. CHAFLE, Sunil CHANDRA, Vijay MANN et Mangala Gowri NANDA. Decentralized Orchestration of Composite Web Services. *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 134–143, 2004.

[49] CODAGEN. *Codagen Architect*. Codagen, december 2004. Disponible sur http://www.codagen.com/products/architect/,

[50] COMPUWARE CORPORATION. *OptimalJ*. Compuware Corporation, december 2004. Disponible sur http://wwww.compuware.com/products/optimalj/,

[51] Steve COOK. Domain-Specific Modeling and Model Driven Architecture. *MDA Journal*, pages 1–10, January 2004.

[52] Patrícia Dockhorn COSTA, Luís Ferreira PIRES, Marten van SINDEREN et José Gonçalves Pereira FILHO. Towards a Services Platform for Mobile Context-Aware Applications. *Proc. of the First International Workshop on Ubiquitous Computing (IWUC 2004)*, 2004.

[53] Lisa CRISPIN et Tip HOUSE. *Testing Extreme Programming*. Addison-Wesley, 1st édition, October 2002.

[54] Krzysztof CZARNECKI et Simon HELSEN. Classification of Model Transformation Approaches. *proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA*, October 2003.

[55] A. DAN, D. DAVIS, R. KEARNEY, R. KING, A. KELLER, D. KUEBLER, H. LUDWIG, M. POLAN, M. SPREITZER et A. YOUSSEF. Web Services on demand: WSLA-driven Automated Management. *IBM Systems Journal, Special Issue on Utility Computing*, 43(1):136–158, March 2004.

[56] Umeshwar DAYAL, Meichun HSU et Rivka LADIN. Business Process Coordination: State of the Art, Trends, and Open Issues. *VLDB 2001*, pages 3–13, 2001.

[57] DCE. Distributed Computing Environment (DCE), November 2004. Disponible sur http://www.opengroup.org/tech/dce/,

[58] A. DOAN, J. MADHAVAN, P. DOMINGOS et A. HAVELY. Learning to Map between Ontologies on the Semantic Web. *In Proc. of the 11 th International World Wide Web Conference (WWW2002)*, pages 662–673, 2002.

[59] DSTC, IBM et CBOP. *MOF Query / Views / Transformations - Second Revised Submission, ad/2004-01-06*, January 2004.

[60] ECLIPSE TOOLS PROJECT. *Eclipse Modeling Framework (EMF) version 2.0*, June 2004. Disponible sur http://www.eclipse.org/emf,

[61] ECLIPSE TOOLS PROJECT. *Graphical Editing Framework (GEF) version 2.1.3*, June 2004. Disponible sur http://www.eclipse.org/gef,

[62] E.D.WILLINK. UMLX - A Graphical Transformation Language for MDA. *Workshop on Model Driven Architecture Foundations and Applications*, pages 13–24, June 2003.

[63] Jean Marie FAVRE. Towards a Basic Theory to Model Driven Engineering. *UML 2004 - Workshop in Software Model Engineering (WISME 2004)*, 2004.

[64] Alex FERRATA et Matthew MACDONALD. *Programming .NET Web Services*. O'Reilly & Associates, 1st édition, September 2002.

[65] Franck FLEUREY, Jim STEEL et Benoit BAUDRY. Validation in Model-Driven Engineering: Testing Model Transformations. *In Proceedings of the SIVOES - Model Design and Validation Workshop*, November 2004.

[66] Daniela FLORESCU, Andreas GRUNHAGEN et Donald KOSSMANN. XL: A Platform for Web Services. *Proceedings of the 2003 CIDR Conference*, January 2003.

[67] Frédéric FONDEMENT et Raul SILAGHI. Defining Model Driven Engineering Processes. *WiSME@UML 2004*, October 2004.

[68] David FRANKEL et John PARODI. Using Model-Driven Architecture^{TM} to Develop Web Services. Rapport technique, IONA Technologies PLC, April 2002.

[69] David S. FRANKEL. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley and OMG Press, 2003.

[70] Erich GAMMA et Kent BECK. *Contributing to Eclipse: Principles, Patterns, and Plug-Ins*. Addison-Wesley Pub Co, 1st édition, October 2003.

[71] Tracy GARDNER, Catherine GRIFFIN, Jana KOEHLER et Rainer HAUSER. A Review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard. July 2003.

[72] Anastasius GAVRAS, Mariano BELAUNDE, Luís Ferreira PIRES et João Paulo A. ALMEIDA. Towards an MDA-based development methodology. *First European Workshop on Software Architecture (EWSA 2004)*, May 2004.

[73] Anna GERBER, Michael LAWLEY, Kerry RAYMOND, Jim STEEL et Andrew WOOD. Transformation: The Missing Link of MDA. *First International Conference on Graph Transformation (ICGT2002)*, October 2002.

[74] Anirudda GOKHALE, Balachandran NATARJAN, Douglas C. SCHMIDT, Andrey NECHYPURENKO, Nanbor WANG, Jeff GRAY, Sandeep NEEMA, Ted BAPTY et Jeff PARSONS. CoSMIC: An MDA Generative Tool for Distributed Real-time and Embdedded Component Middleware and Applications. *Proceedings of the OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture*, November 2002.

[75] Aniruddha GOKHALE, Bharat KUMAR et Arnaud SAHUGUET. Reinventing the Wheel? CORBA vs. Web Services. *The Eleventh International World Wide Web Conference (WWW2002)*, 2002.

[76] H T GORANSON. Semantic distance and enterprise integration. *International Conference on Enterprise Integration and Modelling Technology (ICEIMT2004)*, October 2004.

[77] Jeff GRAY, Jing ZHANG, Yuehua LIN, Hui WU, Suman ROYCHOUDHURY, Rajesh SUDARSAN, Aniruddha GOKHALE, Sandeep NEEMA, Feng SHI et Ted BAPTY. Model-Driven Program Transformation of a Large Avionics Framework. *Generative Programming and Component Engineering (GPCE 2004)*, pages 361–378, October 2004.

[78] Roy GRONMO, David SKOGAN, Ida SOLHEIM et Jon OLDEVIK. Model-Driven Web Services Development. *2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04)*, pages 42–45, March 2004.

[79] J.C. GRUNDY, J.G. HOSKING, R.W. AMOR, W.B. MUGRIDGE et Y. LI. Domain-specific visual languages for specifying and generating data mapping systems. *Journal of Visual Languages and Computing*, 15(3-4):243–263, June-August 2004.

[80] Rachid HAMADI et Boualem BENATALLAH. A Petri Net-based Model for Web Service Composition. *Fourteenth Australasian Database Conference (ADC2003)*, 17, 2003.

[81] Slimane HAMMOUDI, Denivaldo LOPES et Jean BÉZIVIN. Approche MDA pour le développement des e-applications sur la plate-forme des services Web. *Revue d'Ingénierie des Systèmes d'Information, Special issue on: Web services, theory and applications*, 2005.

[82] Jan Hendrick HAUSMANN et Stuart KENT. Visualizing Model Mappings in UML. *Proceedings of the 2003 ACM symposium on Software Visualization (SOFTVIS)*, pages 169–178, June 2003.

[83] Sumi HELAL, Stanley SU, Jie MENG, Raja KRITHIVASAN et Arun JAGATHEESAN. The Internet Enterprise. *Second IEEE/IPSJ Symposium on Applications and the Internet -SAINT'02*, pages 54–63, January 2002.

[84] W. HÜMMER, W. LEHNER et H. WEDEKIND. Contracting in the Days of eBusiness. *ACM Sigmod Record: Data management issues in electronic commerce*, 31(1):31–36, March 2002.

[85] K. HOGG, P. CHILCOTT, M. NOLAN et B. SRINIVASAN. An Evaluation of Web Services in the Design of a B2B Application. *Proceedings of the 27th conference on Australasian computer science*, pages 331–340, 2004.

[86] David HOLLINGSWORTH. *Workflow Management Coalition: The Workflow Reference Model, Document Number TC00-1003, Document Status - Issue 1.1*. WfMC, January 1995.

[87] Marty HUMPHREY. Web services as the Foundation for Learning Complex Software System Development. *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 457–461, 2004. ACM Press,

[88] IBM. *Rational Rose XDE Developer*. IBM, december 2004. Disponible sur http://www-306.ibm.com/software/awdtools/developer/rosexde/,

[89] INRIA. *ATL Transformation Example: Class to Relational*, January 2005.

[90] INTERACTIVE OBJECTS. *ArcStyler*. Interactive Objects, december 2004. Disponible sur http://www.arcstyler.com/,

[91] INTERACTIVE OBJECTS SOFTWARE GMBH AND PROJECT TECHNOLOGY, INC.. *2nd Revised Submission to MOF Query / View / Transformation RFP*, January 2004. Disponible sur http://www.omg.org/docs/ad/04-01-14.pdf,

[92] M. C. JAEGER, G. ROJEC-GOLDMANN et G. MÜHL. QoS Aggregation for Web Service Composition Using Workflow Patterns. *Eight IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pages 20–24, September 2004.

[93] Arun JAGATHEESAN et Sumi HELAL. Sangam: Universal Interop Protocols for E-service Brokering Communities using Private UDDI Nodes. *IEEE Symposium on Computers and Communications - ISCC'2003*, 2003.

[94] JAVA COMMUNITY PROCESS. *Java^{TM} Metadata Interface(JMI) Specification, Version 1.0*, June 2002.

[95] Lamia Labed JILANI, Jules DESHARNAIS et Ali MILI. Defining and applying measures of distance between specifications. *IEEE Transactions on Software Engineering*, 27(8):673–703, August 2001.

[96] Matjaz B. JURIC, Bostjan KEZMAH, Marjan HERICKO, Ivan ROZMAN et Ivan VEZOCNIK. Java RMI, RMI Tunneling and Web Services Comparison and Performance Analysis. *SIGPLAN Not., ACM Press*, 39(5), 2004.

[97] JXTA.ORG. JXTA, Avril 2005. Disponible sur http://www.jxta.org,

[98] Audris KALNINS, Janis BARZDINS et Edgars CELMS. Basics of Model Transformation Language MOLA. *Workshop on Model Driven Development (WMDD 2004) at ECOOP 2004*, June 2004.

[99] O. KATH, A. BLAZARENAS, M. BORN, K.-P. ECKERT, M. FUNABASHI et C. HIRAI. Towards Executable Models: Transforming EDOC Behavior Models to CORBA and BPEL. *Eight IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pages 267–274, September 2004.

[100] Raman KAZHAMIAKI, Marco PISTORE et Marco ROVERI. A Framework for Integrating Business Processes and Business Requirements. *The 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC2004)*, pages 9–20, September 2004.

[101] Stuart KENT et Robert SMITH. The Bidirectional Mapping Problem. *Electronic Notes in Theoretical Computer Sciences*, 82(7), 2003.

[102] Anneke KLEPPE, Jos WARMER et Wim BAST. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 1st édition, August 2003.

[103] Jana KOEHLER, Rainer HAUSER, Shubir KAPOOR, Fred Y. WU et Santhosh KUMARAN. A Model-Driven Transformation Method. *The 7th International IEEE Enterprise Distributed Object Computing Conference (EDOC'03)*, pages 186–197, September 2003.

[104] Ivan KURTEV, Jean BÉZIVIN et Mehmet AKSIT. Technological Spaces: An Initial Appraisal. *CoopIS, DOA'2002 Federated Conferences, Industrial track*, 2002.

[105] Neal LEAVITT. Are Web Services Finally Ready to Deliver? *Computer: Innovative Technology for Computer Professionals*, 37(11):14–18, November 2004.

[106] Barbara Staudt LERNER. A model for compound type changes encountered in schema evolution. *ACM Transactions on Database Systems (TODS)*, 25(1):83–127, March 2000.

[107] Sébastien LETÉLIÉ. Les Services Web asynchrones. *Développeur Référence*, pages 6–10, décembre 2002.

[108] Kees LEUNE, Mike PAPAZOGLOU et Willem-Jan van den HEUVEL. Specification and Querying of Security Constraints in the EFSOC Framework. *ACM Proceedings of the 2nd International Conference on Service Oriented Computing - ICSOC'04*, pages 125–133, 2004.

[109] Kees LEUNE, Willem-Jan van den HEUVEL et Mike PAPAZOGLOU. Exploring a Multi-Faceted Framework for SOC: How to develop secure web-service interactions? *Proceedings of the 14th International Workshop on Research Issues on Data Engineering (RIDE'04)*, March 2004.

[110] Frank LEYMANN. *Web Services Flow Language (WSFL 1.0)*, May 2001.

[111] Benchaphon LIMTHANMAPHON et Yanchun ZHANG. Web Service Composition Transaction Management. *ACM Proceedings of the fifteenth Conference on Australian Database*, pages 171–179, 2004.

[112] Denivaldo LOPES et Slimane HAMMOUDI. Web Services in the Context of MDA. *The 2003 International Conference on Web Services (ICWS'03)*, pages 424–427, June 2003.

[113] Denivaldo LOPES, Slimane HAMMOUDI, Jean BÉZIVIN et Frédéric JOUAULT. Generating Transformation Definition from Mapping Specification: Application to Web Service Platform. *The 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, (LNCS 3520):309–325, June 2005.

[114] Denivaldo LOPES, Slimane HAMMOUDI, Jean BÉZIVIN et Frédéric JOUAULT. Mapping Specification in MDA: From Theory to Practice. *First International Conference INTEROP-ESA'2005 Interoperability of Enterprise Software and Applications*, February 2005.

[115] Jayant MADHAVAN, Philip A. BERNSTEIN, Pedro DOMINGOS et Alon Y. HALEVY. Representing and Reasoning about Mappings between Domain Models. *Eighteenth National Conference on Artificial intelligence (AAAI'02)*, pages 80–86, 2002.

[116] Daniel J. MANDELL et Sheila A. MCILRAITH. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. *Proceedings of the Second International Semantic Web Conference (ISWC2003)*, 2003.

[117] Frank MARSCHALL et Peter BRAUN. Model Transformations for the MDA with BOTL. *Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications*, June 2003.

[118] David MARTIN, Mark BURSTEIN, Jerry HOBBS, Ora LASSILA, Drew MCDERMOTT, Sheila MCILRAITH, Srini NARAYANAN, Massimo PAOLUCCI, Bijan PARSIA, Terry PAYNE, Evren SIRIN, Naveen SRINIVASAN et Katia SYCARA. *OWL-S: Semantic Markup for Web Services*. W3C, November 2004. Disponible sur http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/,

[119] Joanne MARTIN, Ali ARSANJANI, Peri TARR et Brent HAILPERN. Web Services: Promises and Compromises. *Queue*, 1(1):48–58, 2003.

[120] Georg Groh MARTIN S. LACHER. Facilitating the Exchange of Explicit Knowledge through Ontology Mappings. *14th International FLAIRS conference*, pages 21–23, May 2001.

[121] Brahim MEDJAHED, Boualem BENATALLAH, Athman BOUGUETTAYA, Anne H. H. NGU et Ahmed K. ELMAGARMID. Business-to-business Interactions: Issues and Enabling Technologies. *The VLDB Journal The International Journal on Very Large Data Bases*, 12(1):59–85, 2003.

[122] Stephen J. MELLOR, Kendall SCOTT, Axel UHL et Dirk WEISE. *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley, 1st édition, March 2004.

[123] MIA. *Model-in-Action*. Mia-Software, december 2004. Disponible sur http://www.mia-software.com,

[124] MICROSOFT. *C# - Language Specification version 1.2*. Microsoft, 2003.

[125] MICROSOFT. COM: Component Object Model Technologies, novembre 2004. Disponible sur http://www.microsoft.com/com/,

[126] MIDDLEWARE COMPANY. Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach. Rapport technique, The Middleware Company, June 2003. Disponible sur www.middleware-company.com/casestudy,

[127] Nirmal K. MUKHI, Ravi KONURU et Francisco CURBERA. Cooperative middleware specialization for service oriented architectures. *Proceedings of the 13th international World Wide Web*, pages 206–215, 2004.

[128] Pierre-Alain MULLER et Nathalie GAERTNER. *Modélisation Objet avec UML*. Eyrolles, 514, 2004.

[129] Pierre-Alain MULLER, Philippe STUDER, Frederic FONDEMENT et Jean BÉZIVIN. Platform Independent Web Application Modeling and Development with Netsilon. *Journal on Software and Systems Modelling (SoSym) - Springer*, 2005.

[130] Srini NARAYANAN et Sheila A. MCIIRAITH. Simulation, Verification and Automated Composition of Web Services. *The Eleventh International World Wide Web Conference (WWW2002) ACM Press*, pages 77–88, May 2002.

[131] NETBEANS.ORG. MetaData Repository (MDR), 2003. Disponible sur http://mdr.netbeans.org,

[132] OASIS. *UDDI Executive White Paper*, November 2001.

[133] OASIS. *UDDI Version 2.03 Data Structure Reference, UDDI Committee Specification*, July 2002.

[134] OASIS. *OASIS Web Services Business Process Execution Language TC*, November 2004. Disponible sur http://www.oasis-open.org/committees/wsbpel/charter.php,

[135] OASIS. *Security Assertions Markup Language version 2.0*. OASIS, December 2004. Disponible sur http://www.oasis-open.org/committess/tc_home.php?wg_abbrev=securi% -ty,

[136] OASIS. *Web Service Security: SOAP Message Security 1.0 (WS-Security 2004)*. OASIS, March 2004. Disponible sur http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf,

[137] OMG. *Trading Object Service Specification version 1.0, formal/00-06-27*, May 2000.

[138] OMG. *Model Driven Architecture (MDA)- document number ormsc/2001-07-01*, July 2001.

[139] OMG. *OMG Unified Modeling Language Specification, Version 1.4*, September 2001.

[140] OMG. *CORBA Components, V3.0, formal/02-06-65*, June 2002.

[141] OMG. *Meta Object Facility (MOF) specification - version 1.4, formal/01-11-02*, April 2002.

[142] OMG. *Request for Proposal: MOF 2.0 Query/Views/Transformations RFP*, October 2002. Disponible sur http://www.omg.org/docs/ad/02-04-10.pdf,

[143] OMG. *The Common Object Request Broker Architecture: Core Specification, Version 3.0.6 formal/02-12-02*, December 2002.

[144] OMG. *UML Profile and Interchange Models for Enterprise Application Integration (EAI) Specification, ptc/02-02-02*, September 20 2002.

[145] OMG. *UML Profile for Enterprise Distributed Object Computing Specification*, February 2002. OMG Adopted Specification ptc/02-02-05,

[146] OMG. *XML Metadata Interchange (XMI) Specification, Version 1.2*, January 2002.

[147] OMG. *MDA Guide Version 1.0.1, Document Number: omg/2003-06-01*. OMG, June 2003.

[148] OMG. *Object Constraint Language (OCL) Specification, ad/03-10-14*, 2003.

[149] OMG. *UML v 1.5, formal/2003-03-01*, 2003.

[150] OMG. *Unified Modeling Langage: Superstructure version 2.0 Final Adopted Specification, OMG ptc/03-08-02*. OMG, August 2003. ,

[151] OMG. *XML Metadata Interchange (XMI) Specification, Version 2.0, formal/03-05-02*, May 2003.

[152] OMG. *Enterprise Collaboration Architecture (ECA) Specification, OMG formal/04-02-01*, February 2004.

[153] OMG. *Naming Service Specification version 1.3, formal/04-10-03*, October 2004.

[154] OMG. *UML Profile for Enterprise Collaboration Architecture Specification, OMG formal/04-02-05*, February 2004.

[155] OMONDO. *Omondo Eclipse UML*, October 2004. Disponible sur http://www.omondo.com,

[156] Bart ORRIENS, Jian YANG et Mike. P. PAPAZOGLOU. Model Driven Service Composition. *Proceedings of the First International Conference on Service Oriented Computing*, December 2003.

[157] Michael P. PAPAZOGLOU. *Web Services and Business Transactions*. Kluwer Academic Publishers, World Wide Web: Internet and Web Information Systems, 2003.

[158] Mike P. PAPAZOGLOU. Service-Oriented Computing: Concepts, Characteristics and Directions. *IEEE Proceedings of 4th International Conference on Web Information Systems Engineering (WISE 2003)*, pages 3–12, December 2003.

[159] Christine PARENT et Stefano SPACCAPIETRA. Intégration de bases de données: Panorama des problèmes et des approches. *Ingénierie des sytèmes d'information*, 4(3), 1996.

[160] Abhijit PATIL, Swapna OUNDHAKAR, Amit SHETH et Kunal VERMA. METEOR-S Web Service Annotation Framework. *Proceedings of the 13th international conference on World Wide Web ACM Press*, pages 553–562, May 2004.

[161] Octavian PATRASCOIU. Mapping EDOC to Web Services using YATL. *8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pages 286–297, September 2004.

[162] Octavian PATRASCOIU. Model Transformations in YATL. Studies and Experiments - Reference Manual Version 1.0. Rapport technique 3-04, March 2004.

[163] Octavian PATRASCOIU. YATL: Yet Another Transformation Language. *In Proceedings of the 1st European MDA Workshop MDA-IA*, pages 83–90, January 2004.

[164] Mikael PELTIER. *Techniques de Transformation de Modèles Basées sur la méta-modélisation*. Thèse de Doctorat, UFR Sciences et Techniques, Université de Nantes, 2003.

[165] Giacomo PICCINELLI, Wolfgang EMMERICH, Scott Lane WILLIAMS et Mary STEARNS. A Model-Driven Architecture for Electronic Service Management Systems. *Proc. of the 1st Int. Conference on Service-Oriented Computing*, 2910:241–255, 2003.

[166] T. PILIOURA, A. TSALGATIDOU et S. HADJIEFTHIMIADES. Scenarios of using Web Services in M-Commerce. *SIGecom Exchanges*, 3(4), 2003.

[167] L. POPA, Y. VELEGRAKIS, R.J. MILLER, M. HERNANDEZ et R. FAGIN. Mapping Generation and Data Translation of Heterogeneous Web Data. *In International Workshop on Data Integration over the Web (DIWeb)*, May 2002.

[168] Rachel A. POTTINGER et Philip A. BERNSTEIN. Merging Models Based on Given Correspondences. *Proceedings of the 29th VLDB Conference*, pages 826–873, 2003.

[169] Eclipse PROJECT. *Java Emitter Templates (JET)*, december 2004. Disponible sur http://www.eclipse.com/emf,

[170] J. Mark PULLEN, Ryan BRUNTON, Don BRUTZMAN, David DRAKE, Michael HIEB, Katherine MORSE et Andreas TOLK. Using Web Services to Integrate Heterogeneous Simulations in a Grid Environment. *International Conference on Computational Science 2004*, June 2004.

[171] QVT-MERGE GROUP. *Revised submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10)*, April 2004. Disponible sur http://www.omg.org/docs/ad/04-04-01.pdf,

[172] Erhard RAHM et Philip A. BERNSTEIN. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001.

[173] Anand RANGANATHAN et Scott MCFADDIN. Using Workflows to Coordinate Web Services in Pervasive Computing Environments. *In 2nd International Conference on Web Services (ICWS 2004)*, July 2004.

[174] Sergi SALICKI. Interview Thales. *IT-expert*, (51):1–6, septembre/octobre 2004.

[175] Richard E. SCHANTZ et Douglas C. SCHMIDT. Research Advances in Middleware for Distributed Systems: State of the Art. *Kluwer Academic Publishers Communication Systems: The State of the Art*, 2003.

[176] Tony Chao SHAN. Building a Service-Oriented eBanking Platform. *2004 IEEE International Conference on (SCC'04)*, pages 237–244, September 2004.

[177] Oliver SIMS. Enterprise MDA or How Enterprise Systems Will Be Built. *MDA Journal*, September 2004. Disponible sur http://www.bptrends.com/search.cfm?keyword=MDA+journal&go-go=1&go.x=68&go.y=4,

[178] SINTEF. *UML Model Transformation Tool (UMT)*, december 2004. Disponible sur http://umt-qvt.sourceforge.net,

[179] James SKENE et Wolfgang EMMERICH. Model Driven Performance Analysis of Enterprise Information Systems. *ETAPS 2003 - Electronic Notes in Theoretical Computer Science*, 82(6), 2003.

[180] David SKOGAN, Roy GRONMO et Ida SOLHEIM. Web Service Composition in UML. *Eight IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pages 47–57, September 2004.

[181] Monika SOLANKI, Antonio CAU et Hussein ZEDAN. Augmenting Semantic Web Service Description With Compositional Specification. *Proceedings of the 13th international conference on World Wide Web ACM Press*, pages 544–552, 2004.

[182] SourceFORGE.net. *Jamda*, december 2004. Disponible sur http://jamda.sourceforge.net/,

[183] Biplav SRIVASTAVA et Jana KOEHLER. Web Service Composition - Current Solutions and Open Problems. *13th International Conference on Automated Planing&Scheduling (ICAPS03)*, June 2003.

[184] SUN MICROSYSTEMS. Java Remote Method Invocation (Java RMI), novembre 2004. Disponible sur http://java.sun.com/products/jdk/rmi,

[185] SUN MICROSYSTEMS. Java Web Services Developer Pack, March 2004. Disponible sur http://java.sun.com/webservices,

[186] Satish THATTE. *XLANG - Web Services for Business Process Design*, 2001.

[187] Zhong TIAN, Leo Y. LIU, Jing LI, Jen-Yao CHUNG et Vibby GUTTEMUKKALA. Business-to-Business e-Commerce with Open Buying on the Internet, May 1999. Disponible sur http://www.research.ibm.com/iac/papers/obi-paper/,

[188] Andreas TOLK. Avoiding another Green Elephant - A Proposal for the Next Generation HLA based on the Model Driven Architecture. *2002 Fall Simulation Interoperability Workshop*, September 2002.

[189] Andreas TOLK et James A. MUGUIRA. M&S within the Model Driven Architecture. *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, December 2004.

[190] Shalom TSUR, Serge ABITEBOUL, Rakesh AGRAWAL, Umeshwar DAYAL, Johannes KLEIN et Gerhard WEIKUM. Are Web services the next revolution in e-commerce? *27 th International Conference on Very Large Data Bases, Panel*, pages 614–617, September 2001.

[191] S. TUECK, I. FOSTER, J. FREY, S. GRAHAM, C. KESSELMAN, T. MAQUIRE, T. SANDHOLM, D. SNELLING et P. VANDERBILT. Open Grid Services Infrastructure (OGSI) Version 1.0. June 2003.

[192] UDDI.ORG. *Universal, Description, Discovery and Integration (UDDI) Version 3.0.1*, July 2003. Disponible sur http://www.uddi.org,

[193] Unified Enterprise Modeling Language (UEML), 2003. Disponible sur http://www.ueml.org,

[194] Carnegie Mellon UNIVERSITY. Message-Oriented Middleware, CMU/SEI-97-HB-001. Rapport technique, Software Engineering Institute - Carnegie Mellon University, 1997.

[195] Carnegie Mellon UNIVERSITY. Middleware, CMU/SEI-97-HB-001. Rapport technique, Software Engineering Institute - Carnegie Mellon University, 1997.

[196] Carnegie Mellon UNIVERSITY. Remote Procedure Call, CMU/SEI-97-HB-001. Rapport technique, Software Engineering Institute - Carnegie Mellon University, 1997.

[197] Wil M.P. van der AALST, Marlon DUMAS et Arthur H.M. ter HOFSTEDE. Web Service Composition Languages: Old Wine in New Bottles? *29th EuroMicro Conference*, September 2003.

[198] Ghica van EMDE BOAS. From the Workfloor: Developing Workflow for the Generative Model Transformer. *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*, 2003.

[199] Robert A. van ENGELEN. Pushing the SOAP Envelope With Web Services for Scientific Computing. *International Conference on Web Services (ICWS 2003)*, pages 346–352, July 2003.

[200] Daniel VARRO et Andreas PATARICZA. Generic and Meta-Transformations for Model Transformation Engineering. *Proc. UML 2004: International Conference on the Unied Modeling Language*, October 2004.

[201] Yannis VELEGRAKIS, Renée J. MILLER et Lucian POPA. Mapping Adaptation under Evolving Schemas. *Proceedings of the 29th VLDB Conference*, pages 584–595, September 2003.

[202] Tom VERDICKT, Bart DHOEDT et Frank GIELEN. Incorporating SPE into MDA: Including Middleware Performance Details into System Models. *ACM Proceedings of the fourth international workshop on Software and performance*, pages 120–124, 2004.

[203] W3C. *Web Services Architecture Requirements*. W3C. Disponible sur http://www.w3c.org/2002/-ws/arch/2/wd-wsawg-reqs-04232002.html,

[204] W3C. *SOAP Version 1.2 Part 0: Primer, W3C Working Draft*, December 2001.

[205] W3C. *Web Services Description Language (WSDL) 1.1*, March 2001. Disponible sur http://www.w3c.org/tr/wsdl,

[206] W3C. *Web Services Description Language (WSDL) 2.0 Part1: Core Language, W3C Working Draft*, November 2003.

[207] W3C. *XSL Transformations (XSLT), Version 2.0, W3C Working Draft 12 November*, 2003.

[208] W3C. *Extensible Markup Language (XML) 1.0, W3C Recommendation 04 February*, third edition édition, February 2004.

[209] W3C. *Web Services Architecture (WSA)*, February 2004. Disponible sur http://www.w3c.org/TR/2004/NOTE-ws-arch-20040211/,

[210] W3C. *Web Services Choreography Description Language version 1.0*, April 2004. Disponible sur http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/,

[211] W3C. Web Services Glossary, 2004. Disponible sur http://www.w3.org/TR/ws-gloss,

[212] W3C. *WML Schema Part0: Primer Second Edition*. W3C, 2 édition, October 2004. Disponible sur http://www.w3c.org/TR/xmlschema-0/,

[213] W3C. *XML Path Language (XPath), version 2.0, W3C Working Draft*, July 2004.

[214] Don WELLS. *Extreme Programming: A Gentle Introduction*. Extreme Programming, November 2004. Disponible sur http://www.extremeprogramming.org,

[215] WFMC. Glossaire de la version 2.0, 1999. Disponible sur http://www.wfmc.org/standards/docs/Glossary_French.PDF,

[216] WFMC. *Workflow Process Definition Interface -XML Process Definition Language (XPDL), Document Number WFMC-TC-1025, Document Status -1.0 Final Draft*, October 2002.

[217] Andrew WILLIAMS, Anand PADMANABHAN et M. Brian BLAKE. Local Consensus Ontologies for B2B-Oriented Service Composition. *AAMAS'03*, 2003.

[218] Andrew WILLIAMS, Anand PADMANABHAN et M. Brian BLAKE. Local Consensus Ontologies for B2B-oriented Service Composition. *Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems*, pages 647–654, 2003.

[219] Edward D. WILLINK. *A Concrete UML-based Graphical Transformation Syntax: The UML to RDBMS Example in UMLX*, 2003. Disponible sur http://dev.eclipse.org/viewcvs/index-tech.cgi/gmt-home/doc/umlx/M4M03.pdf?rev=1.2,

[220] Dave WINER. *XML-RPC Specification*. XML-RPC.com, June 1999. Disponible sur http://www.xmlrpc.com/spec,

[221] WS-INTEROPERABILITY. *Basic Profile Version 1.0*. WS-Interoperability, March 2004. Disponible sur http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html,

[222] Jia ZHANG et Liang-Jie ZHANG. WS-Trustworthy: A Framework for Web Services Centered Trustworthy Computing, Services Computing. *2004 IEEE International Conference on (SCC'04)*, pages 186–193, 2004.

Liste des tableaux

Partie I — Contexte de l'étude

Partie II — Les Services Web dans le contexte de MDA

Partie III — Prototypage : développement d'un *plug-in* sous « Eclipse »

Table des figures

Partie I — Contexte de l'étude

Partie II — Les Services Web dans le contexte de MDA

Partie III — Prototypage : développement d'un *plug-in* sous « Eclipse »

Crédit figure

Je remercie G. CAPLAT et J. L. SOURROUILLE (INSA, Bat. Blaise Pascal, France) pour avoir permis l'utilisation des figures 2.2, 2.3, 2.4, 2.5 avec le © **2005 G. Caplat and J. L. Sourrouille**.
Je remercie ACM (ASSOCIATION FOR COMPUTING MACHINERY) pour avoir permis l'utilisation des figures 2.6, 2.7, 2.8 avec le © **2005 ACM**.
Je remercie SPRINGER-VERLAG pour avoir permis l'utilisation des figures 2.26 avec le © **2005 Springer-Verlag**.
Je remercie IEEE COMPUTER SOCIETY pour avoir permis l'utilisation des figures 2.15, 2.16, 2.21, 2.22, 2.23, 2.24, 2.25, 2.27, 2.28 avec le © **2005 IEEE**.
Je remercie J. BÉZIVIN et F. JOUAULT (Atlas Group, LINA, Université de Nantes) pour avoir permis l'utilisation des figures 2.13, 2.14 avec le © **2005 J. Bézivin et F. Jouault**.
Je remercie L. POPA, M. HERNANDEZ et R. FAGIN (IBM Almaden Research Center), Y. VELEGRAKIS et R.J. MILLER (University of Toronto) pour avoir permis l'utilisation de la figure 2.11 avec le © **2002 L. Popa, Y. Velegrakis, R.J. Miller, M. Hernandez, R. Fagin**.

Liste des exemples

Table des matières

Annexes

ANNEXE A

Les métamodèles de EDOC, WSDL et BPEL

La figure A.1 présente le fragment du métamodèle de EDOC-CCA qui a été utilisé dans nos expérimentations. D'abord, nous avons utilisé l'outil *Poseidon for UML 1.6* de *Gentleware*[1] pour créer et éditer un modèle de EDOC-CCA en tant que diagramme de classes UML. Ensuite, nous avons exporté ce modèle dans le format XMI. Finalement, nous avons utilisé l'outil *uml2mof* de *netBeans* [2] pour transformer ce modèle de EDOC-CCA conforme à UML dans un métamodèle de EDOC-CCA conforme à MOF. Le modèle de l'agence de voyages (figure 3.6) conforme à ce métamodèle EDOC-CCA a été créé en utilisant une définition de transformations écrite en ATL. Cette définition de transformations est illustrée dans l'exemple A.1.

Exemple A.1 – La définition de transformation en ATL pour créer le modèle de l'agence de voyages conforme au métamodèle de EDOC-CCA (fragment)

```
-- File: CreateTravelAgPC.atl
-- created by Denivaldo LOPES (dlopes@eseo.fr)
-- This program create a EDOC model (Process Component: TravelAg).
-- EDOC metamodel (based on MOF) : EDOC (file: EDOCMetaDI.xml)
-- UML metamodel (based on MOF)  : UML (file: UMLDI-20030818.xmi)
-- This EDOC metamodel was created using Poseidon 1.6 and 'uml2mof'
-- This transformation is used to create a EDOC model. Afterwards, this model
-- is used to generate a WSDL and Java model.
module UML2EDOC;
create OUT:EDOC  from IN:UML;

rule C2PCTravelAg{
        from pck: UML!Package(pck.name <> 'java' and pck.name <> 'lang' and not pck.name.
            startsWith('model') )
        to pckEdoc: EDOC!Package(
                name <- 'travelagency'
        ),
        pc: EDOC!ProcessComponent(
                name <- 'TravelAg',
                ports <- sellTravel,
                namespace <- pckEdoc
        ),
        sellTravel:  EDOC!ProtocolPort(
                name <- 'SellTravel',
                protocol <- buySellTravel
        ),
        buySellTravel: EDOC!Protocol(
                name <- 'BuySellTravel',
                ports <- Sequence{findTravel, reservTravel},
```

[1] http://www.gentleware.com
[2] http://www.netbeans.com

```
                namespace <- pckEdoc
        ),
        findTravel:EDOC!OperationPort(
                name <- 'findTravel',
                ports <- Sequence{findTravelCall,findTravelResponse}
        ),
        findTravelCall:EDOC!FlowPort(
                name <- 'findTravelCall',
                type <- travelReq,
                direction <- #initiates
        ),
        findTravelResponse:EDOC!FlowPort(
                name <- 'findTravelResponse',
                type <- travelList,
                direction <- #responds
        ),
        travelReq: EDOC!CompositeData(
                name <- 'input',
                namespace <- pckEdoc,
                feature <- Sequence{attrReqInput}
        ),
        travelList:EDOC!CompositeData(
                name <- 'TravelList',
                namespace <- pckEdoc
        ),
        attrReqInput: EDOC!Attribute(
                byValue <- true,
                type <- typeReqInput
        ),
        typeReqInput: EDOC!CompositeData(
                name <- 'TravelReq',
                namespace <- pckEdoc
        ),
        reservTravel :EDOC!OperationPort(
                name <- 'reserveTravel',
                ports <- Sequence{reservTravelCall, reservTravelResponse}
        ),
        reservTravelCall:EDOC!FlowPort(
                name <- 'reserveTravelCall',
                type <- travelInf,
                direction <- #initiates
        ),
        reservTravelResponse:EDOC!FlowPort(
                name <- 'reserveTravelResponse',
                type <- reserv,
                direction <- #responds
        ),
        travelInf: EDOC!CompositeData(
                name <- 'input',
                namespace <- pckEdoc,
                feature <- Sequence{attrTravelInf}
        ),
        reserv:EDOC!CompositeData(
                name <- 'Reserv',
                namespace <- pckEdoc
        ),
        attrTravelInf: EDOC!Attribute(
                byValue <- true,
                type <- typeTravelInf
        ),
        typeTravelInf: EDOC!CompositeData(
                name <- 'TravelInf',
                namespace <- pckEdoc
        )
}
```

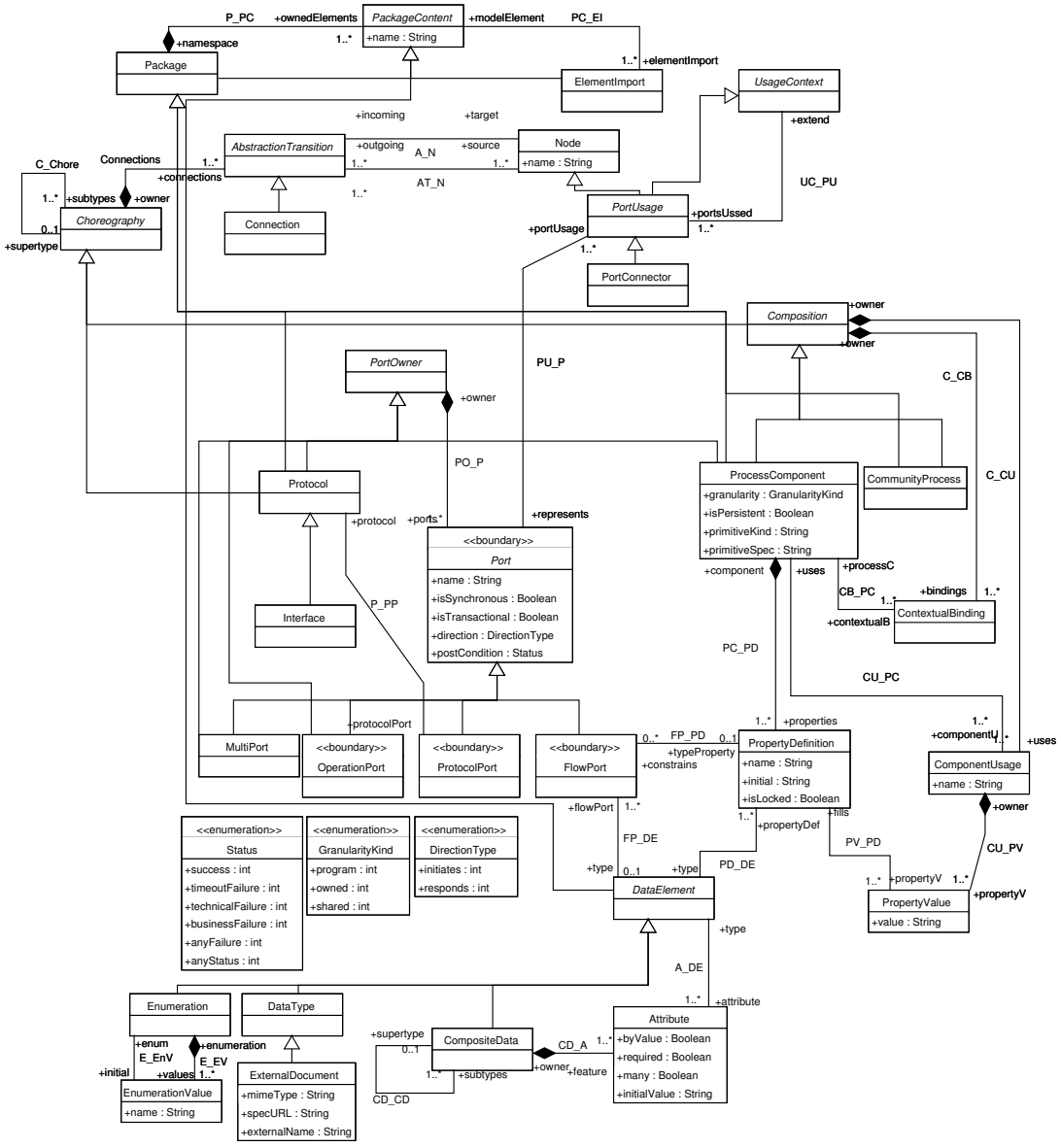


Figure A.1 – Le métamodèle de EDOC-CCA [152] (fragment)

La figure A.2 présente un métamodèle de WSDL qui a été construit à partir de son schéma [205]. D'abord, nous avons utilisé l'outil *Poseidon for UML 1.6* de *Gentleware* pour créer et éditer un modèle de WSDL en tant que diagramme de classes UML. Ensuite, nous avons exporté ce modèle dans le format XMI. Finalement, nous avons utilisé l'outil *uml2mof* de *netBeans* pour transformer ce modèle de WSDL conforme à UML dans un métamodèle de WSDL conforme à MOF. Nous avons utilisé ce même processus et ces outils pour créer d'autres métamodèles, tel que celui de BPEL (figure A.3).

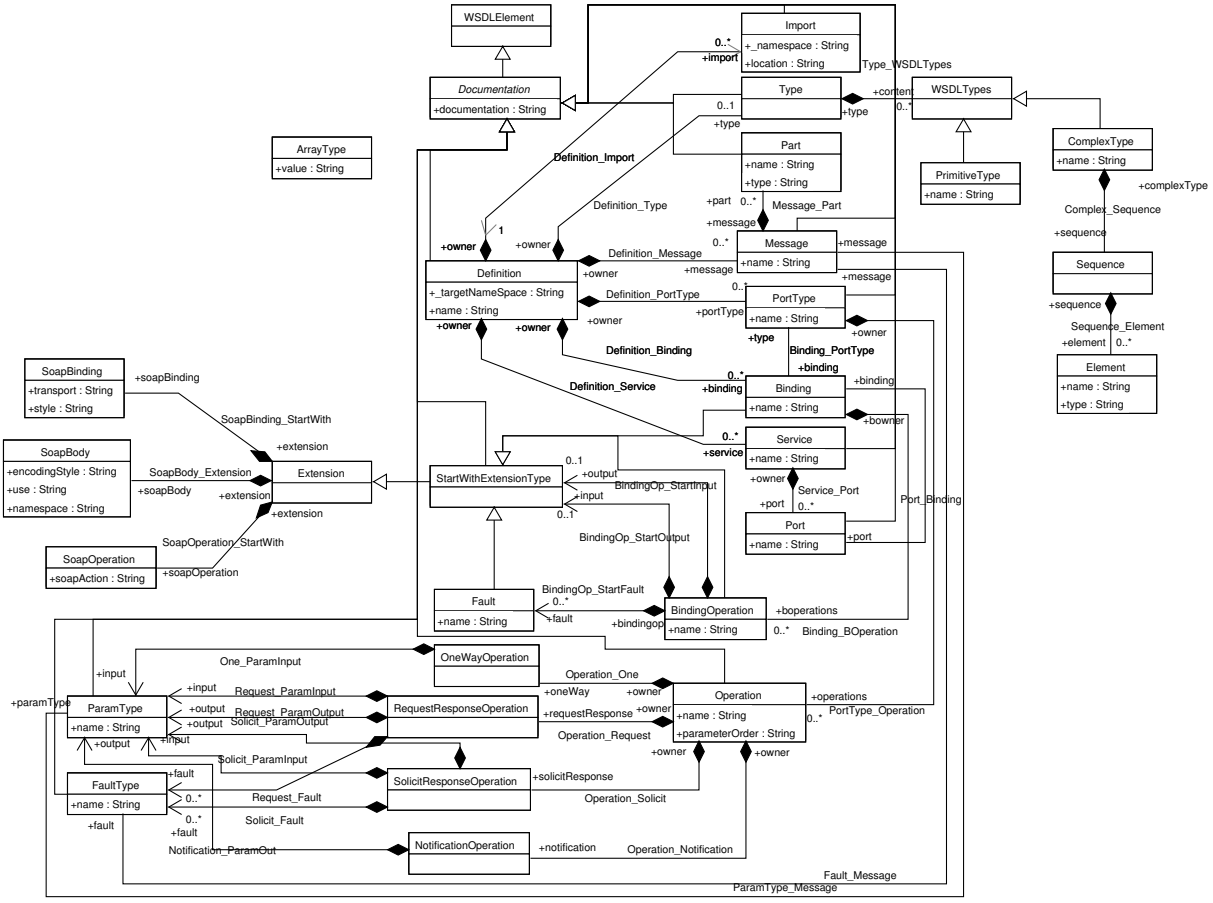


Figure A.2 – Un métamodèle de WSDL

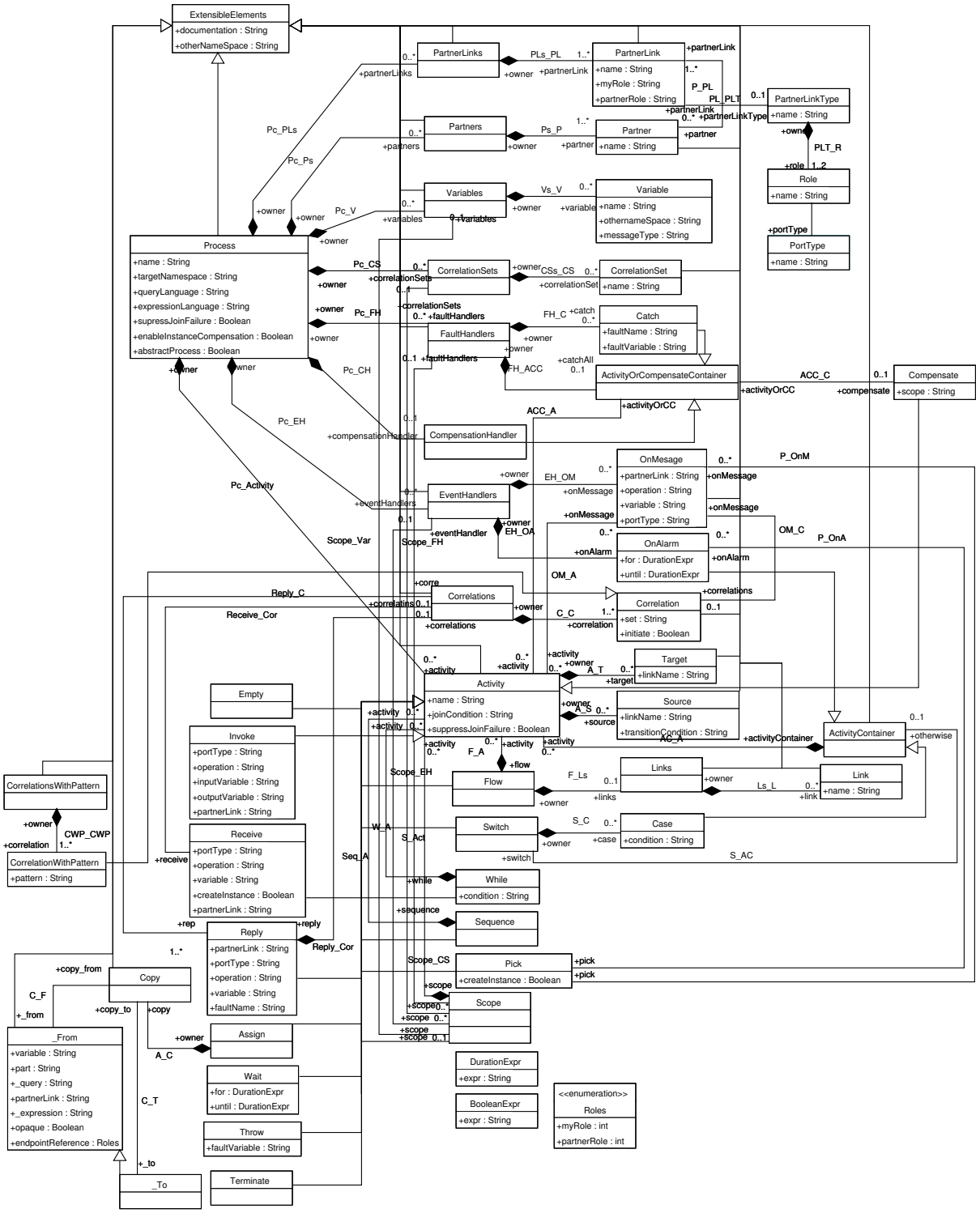


Figure A.3 – Un métamodèle de BPEL4WS

ANNEXE B

La transformation d'un modèle JWSDP vers documents JWSDP

Exemple B.1 – Le programme « _JWSDP_Deploy2SC_query.atl »

```
-- _JWSDP_Deploy2SC_query.atl
-- created by Denivaldo LOPES (dlopes@eseo.fr)
--
-- This program in ATL transform a JWSDP model in a
-- file containing its JWSDP config files.
-- depends on _JWSDP_Deploy2SC
--
-- The file containing the source code is written in
-- the directory 'C:/SourceCode/J2EE'
--

query _JWSDP_Deploy2SourceCode_query =
        _JWSDP!Configuration.allInstances()->
        select(e | e.oclIsTypeOf(_JWSDP!Configuration))->
        collect(x | x.toString().writeTo('C:/SourceCode/J2EE/' + x.serviceFN + '/' +'config-
            wsdl.xml'))->size() +
        _JWSDP!WebApp.allInstances()->
        select(e | e.oclIsTypeOf(_JWSDP!WebApp))->
        collect(x | x.toString().writeTo('C:/SourceCode/J2EE/' + x.serviceFN + '/' +'web.xml')
            )->size() +
        _JWSDP!_Configuration.allInstances()->
        select(e | e.oclIsTypeOf(_JWSDP!_Configuration))->
        collect(x | x.toString().writeTo('C:/SourceCode/J2EE/' + x.serviceFN + '/' +'config-
            interface.xml'))->size()+
        _JWSDP!WebServices.allInstances()->
        select(e | e.oclIsTypeOf(_JWSDP!WebServices))->
        collect(x | x.toString().writeTo('C:/SourceCode/J2EE/' + x.serviceFN + '/' +'jaxrpc-ri
            .xml'))->size();

uses _JWSDP_Deploy2SC;
```

Exemple B.2 – Le programme « _JWSDP_Deploy2SC.atl »

```
-- File: _JWSDP_Deploy2SC.atl
-- created by Denivaldo LOPES (dlopes@eseo.fr)
--
-- This is a library in ATL used to transform a JWSDP model in
-- a JWSDP source code (config and deploy files).

library _JWSDP_Deploy2SC;

```

```
-- config-wsdl.xml

helper context _JWSDP!_WSDL def: toString() : String =
        '\t<wsdl location="'+ self.location+'"\n'+
        '\t\tpackageName="'+ self.packageName +'"/>\n';

helper context _JWSDP!Configuration def: toString() : String =
        '<?xml version="1.0" encoding="UTF-8"?>\n' +
        '<configuration\n' +
        '\txmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">\n' +
         self.wsdl.toString()+
         '</configuration>\n';

-- web.xml

helper context _JWSDP!DisplayName def: toString(): String =
        '\t<display-name>'+self.display +'</display-name>\n';

helper context _JWSDP!Description def: toString(): String =
        '\t<description>' +self.description+ '</description>\n';

helper context _JWSDP!SessionConfig def: toString(): String =
        '\t<session-config>\n' +
    '\t\t<session-timeout>'+self.sessionTimeOut.sessionTimeOut+'</session-timeout>\n' +
        '\t</session-config>\n';

helper context _JWSDP!WebApp def: toString(): String =
        '<?xml version="1.0" encoding="UTF-8"?>\n' +
        '<!DOCTYPE web-app \n' +
        '\tPUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"\n' +
        '\t"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">\n\n' +
        '<web-app>\n' +
        self.displayName.toString()+
        self.description.toString()+
        self.sessionConfig.toString()+
        '</web-app>';

-- config-interface.xml
helper context _JWSDP!_Interface def: toString() :String =
        '\t\t<interface name="'+self.name+'"/>\n';

helper context _JWSDP!Service def: toString(): String =
        '\t<service\n'+
        '\t name="'+self.name+'"\n'+
        '\t targetNamespace="'+self.targetNameSpace+'"\n'+
        '\t typeNamespace="'+self.typeNameSpace+'"\n'+
        '\t packageName="'+self.packageName+'">\n'+
        self.interface.toString()+
        '\t</service>\n';


helper context _JWSDP!_Configuration def: toString() : String =
        '<?xml version="1.0" encoding="UTF-8"?>\n' +
        '<configuration xmlns="'+self.xmlns+'">\n'+
        self.service.toString()+
        '</configuration>\n';

-- jaxrpc-ri
helper context _JWSDP!EndPoint def: toString(): String =
        '\t<endpoint \n'+
        '\t name="'+self.name+'" \n'+
    '\t displayName="'+self.displayName+'" \n'+
    '\t description="'+self.description+'" \n' +
    '\t interface="'+self.interface+'" \n'+
    '\t implementation="'+self.implementation+'" \n'+
```

```
74      '\t model="'+self.model+'"/>\n';

76  helper context _JWSDP!EndPointMapping def: toString() : String =
            '\t <endpointMapping \n'+
78       '\t  endpointName="'+self.endPointName+'" \n'+
         '\t  urlPattern="'+self.urlPattern+'"/>\n';
80
    helper context _JWSDP!WebServices def: toString(): String =
82          '<?xml version="1.0" encoding="UTF-8"?>\n'+
            '<webServices \n'+
84          '  xmlns="'+self.xmlns+'" \n'+
            ' version="'+self.version+'" \n'+
86      ' targetNamespaceBase="'+self.targetNameSpaceBase+'" \n'+
            ' typeNamespaceBase="'+self.typeNameSpaceBase+'" \n'+
88          ' urlPatternBase="/'+self.urlPatternBase+'">\n' +
            self.endPoint.toString() +
90          self.endPointMapping.toString() +
            '</webServices>';
```

ANNEXE C

La transformation d'un modèle C# vers code source C#

Exemple C.1 – Le programme « Csharp2SC_query.atl »

```
-- File: Csharp2SC_query.atl
-- created by Denivaldo LOPES (dlopes@eseo.fr)
--
-- This program in ATL transform a Csharp model in a
-- file containing its Csharp source code.
-- depends on Csharp2SC.atl
--
-- The file containing the source code is written in
-- the directory 'C:/SourceCode/dotNET/'
--

query Csharp2SC_query = CSHARP!Classifier.allInstances()->
        select(e | e.oclIsTypeOf(CSHARP!Class) or e.oclIsTypeOf(CSHARP!Interface))->
        collect(x | x.toString().writeTo('C:/SourceCode/dotNET/' + x.nameSpace.name.replaceAll
            ('.', '/') + '/' + x.name + '.cs'));


uses Csharp2SC;
```

Exemple C.2 – Le programme « Csharp2SC.atl »

```
-- File: Csharp2SC.atl
-- created by Denivaldo LOPES (dlopes@eseo.fr)
--
-- This is a library in ATL used to transform a Csharp model in
-- a Csharp source code.

library Csharp2SC;


-- begin print Class

helper context CSHARP!Class def: visibility(): String =
        if self.visibility=#public then
                'public '
        else if self.visibility=#private then
                'private '
        else
                'protected'
        endif endif;

helper context CSHARP!Class def: modifierAbstract(): String =
```

```
22          if self.isAbstract then
                'abstract '
24          else
                ''
26          endif;

28  helper context CSHARP!Class def: modifierSealed(): String =
            if self.isSealed then
30              'sealed '
            else
32              ''
            endif;
34
    helper context CSHARP!Class def: getBase(): String =
36          if self.base.oclIsTypeOf(CSHARP!Class) then
                ' : ' + self.base.name
38          else
                ''
40          endif;

42  helper context CSHARP!Class def: getImplements(): String =
            self.implements ->
44              select(e|e.oclIsTypeOf(CSHARP!Interface))->
               iterate(i; acc: String = ''|
46              acc +
                if acc='' then
48                  ' : '
                else
50                  ', '
                endif
52              + i.name);

54  helper context CSHARP!Class def: printAttributes(): String =
            self.attribute -> iterate(i; acc: String = ''|
56              acc +
                if acc='' then
58                  ''
                else
60                  '\n'
                endif
62              + '['+i.name+']\n');

64  helper context CSHARP!Class def: toString(): String =
            'namespace '+ self.nameSpace.name + '\n'+
66          '{\n' +
            '\t' + self.printAttributes()+
68          '\t' + self.visibility() +
             self.modifierAbstract()+
70           self.modifierSealed() +
            ' class '+
72          self.name +
            self.getBase()+
74          self.getImplements()+
            '{\n\n'+
76          self.members -> select(e|e.oclIsTypeOf(CSHARP!Method))->
                iterate(i; acc : String = '' |
78              acc + i.toString()) +
            self.members->select(e|e.oclIsTypeOf(CSHARP!Field))->
80              iterate(i; acc : String = '' |
                acc + i.toString()) +
82          '\n\t}\n'+
            '}\n'; -- end namespace
84
    -- end printClass
86
```

```
-- begin print Method (Class)
helper context CSHARP!Method def: visibility(): String =
        if self.visibility=#public then
                'public '
        else if self.visibility=#private then
                'private '
        else
                'protected'
        endif endif;

helper context CSHARP!Method def: modifierStatic(): String =
        if self.isStatic then
                'static '
        else
                ''
        endif;

helper context CSHARP!Method def: modifierSealed(): String =
        if self.isSealed then
                'sealed '
        else
                ''
        endif;

helper context CSHARP!Method def: modifierAbstract(): String =
        if self.isAbstract then
                'abstract '
        else
                ''
        endif;

helper context CSHARP!Method def: modifierOverride(): String =
        if self.isOverride then
                'override '
        else
                ''
        endif;

helper context CSHARP!Method def : getType (): String =
        self.return.type.name;

helper context CSHARP!Method def: getBody() : String =
        if self.body='' then
        '\t\t//Type your code here'
        else
                self.body
        endif;

helper context CSHARP!Method def : getParameters (): String =
        self.parameters->iterate(i; acc:String=''|
        acc     +
        if acc= '' then
                ''
        else
                ','
        endif +
        i.toString());

helper context CSHARP!Parameter def: toString() : String =
        self.type.name +
        ' '+
        self.name;

helper context CSHARP!Method  def: printAttributes(): String =
        self.attribute -> iterate(i; acc: String = ''|
```

```
152                 acc +
                    if acc='' then
154                         ''
                    else
156                         '\n'
                    endif
158                 + '['+i.name+']\n');

160 helper context CSHARP!Method def: toString(): String =
        '\t' + self.printAttributes() +
162     '\t' + self.visibility() +
        self.modifierStatic() +
164     self.modifierSealed() +
        self.modifierAbstract() +
166     self.getType() +
        ' '+
168     self.name +
        '(' +
170     self.getParameters() +
        ') {\n\t'+
172     self.getBody() +
        '\n\t}\n\n';
174
    -- end print Method (Class)
176
    -- begin print Field (Class)
178 helper context CSHARP!Field def: visibility(): String =
        if self.visibility=#public then
180             'public '
        else if self.visibility=#private then
182             'private '
        else
184             'protected'
        endif endif;
186
    helper context CSHARP!Field def: modifierStatic(): String =
188     if self.isStatic then
            'static '
190     else
            ''
192     endif;

194 helper context CSHARP!Field def: modifierSealed(): String =
        if self.isSealed then
196             'sealed '
        else
198             ''
        endif;
200
    helper context CSHARP!Field def: modifierReadOnly(): String =
202     if self.isReadOnly then
            'readonly '
204     else
            ''
206     endif;

208


210 helper context CSHARP!Field def: toString(): String =
        '\t'+ self.visibility() +
212     self.modifierStatic() +
        self.modifierReadOnly() +
214     self.type.name+
        ' '+
216     self.name +
```

```
        ';\n';
218 -- end print Field (Class)

220 -- print Method (Interface)
    helper context CSHARP!Method def: visibilityITF(): String =
222         if self.visibility=#public then
                'public '
224         else if self.visibility=#private then
                'private '
226         else
                'protected'
228         endif endif;

230 helper context CSHARP!Method def: modifierSealedITF(): String =
            if self.isSealed then
232             'sealed '
            else
234             ''
            endif;
236
    helper context CSHARP!Method def : getTypeITF(): String =
238         self.return.type.name;

240 helper context CSHARP!Method def : getParametersITF(): String =
            self.parameters->iterate(i; acc:String=''|
242         acc     +
            if acc= '' then
244             ''
            else
246             ','
            endif +
248         i.toString());

250 helper context CSHARP!Parameter def: toStringITF() : String =
            self.type.name +
252         ' '+
            self.name;
254


256 helper context CSHARP!Method def: toStringITF(): String =
            '\t' + self.visibilityITF() +
258         self.modifierSealedITF() +
            self.getTypeITF() +
260         ' '+
            self.name +
262         '(' +
            self.getParametersITF() +
264         ');\n';
    -- end print Method (Interface)
266
    -- begin print Property (Interface)
268 helper context CSHARP!Property def: toStringITF(): String =
            '\t' +  self.type.name +
270         ' '+
            self.name +
272         '{ get; set;}\n';

274 -- end print Property (Interface)

276
    -- begin print Interface
278 helper context CSHARP!Interface def: visibility(): String =
            if self.visibility=#public then
280             'public '
            else if self.visibility=#private then
```

```
282                 'private '
        else
284                 'protected'
        endif endif;
286
    helper context CSHARP!Interface def: getBase() : String =
288         if self.base.oclIsTypeOf(CSHARP!Interface) then
                    ' : ' + self.base.name
290         else
                    ''
292         endif;

294 helper context CSHARP!Interface def: toString() : String =
        'namespace '+ self.nameSpace.name + '\n'+
296         '{\n' + -- begin namespace
        '\t' + self.visibility() +
298         ' interface '+
        self.name +
300         self.getBase()+
        '{\n'+
302         self.members -> select(e|e.oclIsTypeOf(CSHARP!Method))->
                    iterate(i; acc : String = '' |
304                 acc + i.toStringITF()) +
        self.members -> select(e|e.oclIsTypeOf(CSHARP!Property))->
306                 iterate(i; acc : String = '' |
                    acc + i.toStringITF()) +
308         '\n\t}\n' +
        '}\n'; -- end namespace
310
    -- end print Interface
```

ANNEXE D

La définition de transformation de UML en dotNET

Exemple D.1 – La définition de transformation de UML en dotNET

```
-- File: UML2dotNET_Deploy.atl
--
--
module UML2dotNET;

create OUT: dotNET from IN:UML;

rule C2Conf{
        from c : UML!Class(c.name.startsWith('Service'))
        to conf : dotNET!Configuration(
                serviceFN <- c.namespace.name,
                systemWeb <- systemW
        ),
        systemW: dotNET!SystemWeb(
         webService <- webServ,
         compilation <- comp,
         customerErrors <- customer,
         authentication <- authen,
         trace <- trac,
         sessionState <- session,
         globalization <- glob
        ),
        webServ: dotNET!WebService(
        ),
        comp : dotNET!Compilation(
                defaultLanguage <- 'c#',
                debug <- true
        ),
        customer: dotNET!CustomerErrors(
                mode <- #RemoteOnly
        ),
        authen:dotNET!Authentication(
                mode <- #Windows
        ),
        trac: dotNET!Trace(
                enabled <- false,
                requestLimit <- '10',
                pageOutput <- false,
                traceMode <- 'SortByTime',
                localOnly <- true
        ),
        session: dotNET!SessionState(
                mode <- 'InProc',
```

```
44                  stateConnectionString <- 'tcpip=host:port',
                    sqlConnectionString <- 'data source=host;user id=sa:password=',
46                  cokieless <- false,
                    timeout<- '20'
48          ),
            glob: dotNET!Globalization(
50                  requestEncoding <- 'utf-8',
                    responseEncoding <- 'utf-8'
52          )
    }
54

56  rule C2Disco{
            from c : UML!Class(c.name.startsWith('Service'))
58          to disc : dotNET!Disco(
                    serviceFN <- c.namespace.name +'.'+c.name,
60                  xmlns <- 'disco="http://schemas.xmlsoap.org/disco/";;scl="http://schemas.
                        xmlsoap.org/scl/"',
                    discoveryRef <- discRef,
62                  contractRef <- contRef
            ),
64          discRef : dotNET!DiscoveryRef(
                    ref <- 'http://host:port/'+c.name+'.asmx?WSDL'
66          ),
            contRef: dotNET!ContractRef(
68                  ref <- 'http://host2:port2/hisWebServiceDirectory.disco'
            )
70  }

72
    rule C2VDisco{
74          from c: UML!Class(c.name.startsWith('Service'))
            to vdisc: dotNET!VDisco(
76                  serviceFN <- c.namespace.name +'.'+c.name,
                    xmlns <- 'urn:schemas-dynamicdiscovery:disco.2000-03-17',
78                  exclude <- Sequence{exc1,exc2,exc3,exc4,exc5,exc6}
            ),
80          exc1 : dotNET!Exclude(
                    path <- '_vti_cnf'
82          ),
            exc2 : dotNET!Exclude(
84                  path <- '_vti_pvt'
            ),
86          exc3 : dotNET!Exclude(
                    path <- '_vti_log'
88          ),
            exc4 : dotNET!Exclude(
90                  path <- '_vti_script'
            ),
92          exc5 : dotNET!Exclude(
                    path <- '_vti_txt'
94          ),
            exc6 : dotNET!Exclude(
96                  path <- 'Web References'
            )
98  }
```

Exemple D.2 – Le programme « dotNET_Deploy2SC_query.atl »

```
    -- File: dotNET_Deploy2SC_query.atl
2   -- created by Denivaldo LOPES (dlopes@eseo.fr)
    --
4   -- This program in ATL transform a dotNET model in a
    -- file containing its dotNET config files.
6   -- depends on dotNET_Deploy2SC
```

```
--
-- The file containing the source code is written in
-- the directory 'C:/SourceCode/dotNET/'
--
query dotNET_Deploy2SC_query =
        dotNET!Configuration.allInstances()->
        select(e | e.oclIsTypeOf(dotNET!Configuration))->
        collect(x | x.toString().writeTo('C:/SourceCode/dotNET/' + x.serviceFN + '/' +'Web.
            config'))->size() +
        dotNET!Disco.allInstances()->
        select(e | e.oclIsTypeOf(dotNET!Disco))->
        collect(x | x.toString().writeTo('C:/SourceCode/dotNET/' + x.serviceFN.replaceAll('.',
            '/') + '.disco'))->size() +
        dotNET!VDisco.allInstances()->
        select(e | e.oclIsTypeOf(dotNET!VDisco))->
        collect(x | x.toString().writeTo('C:/SourceCode/dotNET/' + x.serviceFN.replaceAll('.',
            '/') + '.vsdisco'))->size();

uses dotNET_Deploy2SC;
```

Exemple D.3 – Le programme « dotNET_Deploy2SC.atl »

```
-- File: dotNET_Deploy2SC.atl
-- created by Denivaldo LOPES (dlopes@eseo.fr)
--
-- This is a library in ATL used to transform a dotNET model in
-- a dotNET source code (config and deploy files).

library dotNET_Deploy2SC;

-- Web.config

helper context dotNET!WebService def: toString() : String =
        '\t\t<webservice/>\n';

helper context dotNET!Compilation def: toString() : String =
        '\t\t<compilation defaultLanguage="'+self.defaultLanguage +'" debug="'+ self.debug.
            toString() +'" />\n';


helper context dotNET!CustomerErrors def: toString() : String =
        '\t\t<customerErrors mode="'+
        if self.mode = #On then
                'On'
        else if self.mode = #Off then
                'Off'
        else 'RemoteOnly'
        endif endif +
        '" />\n';

helper context dotNET!Authentication def: toString() : String =
        '\t\t<authentication mode="'+
        if self.mode = #Windows then
                'Windows'
        else if self.mode = #Forms then
                'Forms'
        else if self.mode = #Passport then
                'Passport'
        else
                'None'
        endif endif endif +
        '" />\n';

helper context dotNET!Trace def: toString() : String =
```

```
42          '\t\t<trace enabled="'+ self.enabled.toString() +
            '" requestLimit="'+ self.requestLimit +
44          '" pageOutput="'+ self.pageOutput.toString()+
            '" traceMode="'+self.traceMode +
46          '" localOnly="'+self.localOnly.toString() +
            '"/>\n';
48
    helper context dotNET!SessionState def: toString() : String =
50          '\t\t<sessionState mode="'+self.mode +
            '" stateConnectionString="'+ self.stateConnectionString+
52          '" sqlConnectionString="'+ self.sqlConnectionString+
            '" cookieless="'+ self.cokieless.toString()+
54          '" timeout="'+ self.timeout+
            '"/>\n';
56
    helper context dotNET!Globalization def: toString() : String =
58          '\t\t<globalization requestEncoding="'+self.requestEncoding+
            '" responseEncoding="'+self.responseEncoding+'"/>\n';
60
    helper context dotNET!SystemWeb def: toString() : String =
62          '\t<system.web>\n'+
                    self.webService.toString() +
64                  self.compilation.toString()+
                    self.customerErrors.toString()+
66                  self.authentication.toString()+
                    self.trace.toString()+
68                  self.sessionState.toString()+
                    self.globalization.toString()+
70          '\t</system.web>\n';

72  helper context dotNET!Configuration def: toString() : String =
            '<?xml version="1.0" encoding="utf-8">\n'+
74          '<configuration>\n' +
                    self.systemWeb.toString() +
76          '</configuration>\n';

78  -- disco (static)
    helper context dotNET!DiscoveryRef def: toString() : String =
80          '\t<discoveryRef ref="'+self.ref+'" />\n';

82  helper context dotNET!ContractRef def: toString() : String =
            '\t<contractRef ref="'+self.ref+'" />\n';
84
    helper context dotNET!Disco def: toString() : String =
86          '<?xml version="1.0" encoding="utf-8">\n'+
            '<discovery xmlns:'+self.xmlns+' >\n' +
88                  self.discoveryRef.toString()+
                    self.contractRef.toString()+
90          '</discovery>\n';


92
    -- disco (dynamic)
94  helper context dotNET!Exclude def: toString() : String =
            '\t<exclude path="'+self.path+'" />\n';
96
    helper context dotNET!VDisco def: toString() : String =
98          '<?xml version="1.0" encoding="utf-8">\n'+
            '<dynamicDiscovery xmlns="'+ self.xmlns+'">\n' +
100         self.exclude -> select(e|e.oclIsTypeOf(dotNET!Exclude))->
                    iterate(i; acc : String = '' |
102                 acc + i.toString()) +
            '</dynamicDiscovery>\n';
```

ANNEXE E

Un modèle de Services Web (PSM)

Exemple E.1 – Le modèle WSDL (PSM) de l'agence de voyage (fragment)

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<XMI xmi.version = '1.2' timestamp = 'Fri Apr 09 19:48:59 CEST 2004'>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
      <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
  </XMI.header>
  <XMI.content>
    <WSDL.Definition xmi.id = 'a1' _targetNameSpace = 'urn://model 1.wsdl' name = '
        Service_model 1'>
      <WSDL.Definition.type>
                ***
      </WSDL.Definition.type>
    </WSDL.Definition>
    <WSDL.Definition xmi.id = 'a3' _targetNameSpace = 'urn://travelagency.wsdl'
      name = 'Service_travelagency'>
      <WSDL.Definition.type>
        <WSDL.Type xmi.id = 'a4'/>
      </WSDL.Definition.type>
      <WSDL.Definition.portType>
        <WSDL.PortType xmi.id = 'a5' name = 'ServiceTravelAg'>
          <WSDL.PortType.binding>
            <WSDL.Binding xmi.idref = 'a6'/>
          </WSDL.PortType.binding>
          <WSDL.PortType.operations>
            <WSDL.Operation xmi.id = 'a7' name = 'findTravel'>
              <WSDL.Operation.requestResponse>
                <WSDL.RequestResponseOperation xmi.id = 'a8'>
                  <WSDL.RequestResponseOperation.input>
                    <WSDL.ParamType xmi.id = 'a9'>
                      <WSDL.ParamType.message>
                        <WSDL.Message xmi.idref = 'a10'/>
                      </WSDL.ParamType.message>
                    </WSDL.ParamType>
                  </WSDL.RequestResponseOperation.input>
                  <WSDL.RequestResponseOperation.output>
                    <WSDL.ParamType xmi.id = 'a11'>
                      <WSDL.ParamType.message>
                        <WSDL.Message xmi.idref = 'a12'/>
                      </WSDL.ParamType.message>
                    </WSDL.ParamType>
                  </WSDL.RequestResponseOperation.output>
                </WSDL.RequestResponseOperation>
              </WSDL.Operation.requestResponse>
            </WSDL.Operation>
```

```
46              <WSDL.Operation xmi.id = 'a13' name = 'reserveTravel'>
                  <WSDL.Operation.requestResponse>
48                  <WSDL.RequestResponseOperation xmi.id = 'a14'>
                      <WSDL.RequestResponseOperation.input>
50                      <WSDL.ParamType xmi.id = 'a15'>
                          <WSDL.ParamType.message>
52                          <WSDL.Message xmi.idref = 'a16'/>
                          </WSDL.ParamType.message>
54                      </WSDL.ParamType>
                      </WSDL.RequestResponseOperation.input>
56                    <WSDL.RequestResponseOperation.output>
                        <WSDL.ParamType xmi.id = 'a17'>
58                        <WSDL.ParamType.message>
                            <WSDL.Message xmi.idref = 'a18'/>
60                        </WSDL.ParamType.message>
                        </WSDL.ParamType>
62                    </WSDL.RequestResponseOperation.output>
                    </WSDL.RequestResponseOperation>
64                </WSDL.Operation.requestResponse>
              </WSDL.Operation>
66           ***
            </WSDL.PortType.operations>
68        </WSDL.PortType>
        </WSDL.Definition.portType>
70      <WSDL.Definition.message>
          <WSDL.Message xmi.id = 'a10' name = 'findTravel'>
72              ***
          </WSDL.Message>
74        <WSDL.Message xmi.id = 'a12' name = 'findTravelResponse'>
              ***
76        </WSDL.Message>
       ***
78      </WSDL.Definition.message>
      <WSDL.Definition.binding>
80      <WSDL.Binding xmi.id = 'a6' name = 'ServiceTravelAgBinding'>
          <WSDL.Extension.soapBinding>
82          <WSDL.SoapBinding xmi.id = 'a39' transport = 'http://schemas.xmlsoap.org/soap/http
                '
              style = 'rpc'/>
84        </WSDL.Extension.soapBinding>
          <WSDL.Binding.port>
86          <WSDL.Port xmi.idref = 'a40'/>
          </WSDL.Binding.port>
88        <WSDL.Binding.boperations>
            <WSDL.BindingOperation xmi.id = 'a41' name = 'findTravel'>
90           ***
          </WSDL.Binding.boperations>
92        <WSDL.Binding.type>
            <WSDL.PortType xmi.idref = 'a5'/>
94        </WSDL.Binding.type>
        </WSDL.Binding>
96    </WSDL.Definition.binding>
      <WSDL.Definition.service>
98      <WSDL.Service xmi.id = 'a69' name = 'ServiceTravelAg'>
          <WSDL.Service.port>
100         <WSDL.Port xmi.id = 'a40' documentation = '&#9;&#9;&lt;soap:address location=&quot
                ;REPLACE_WITH_ACTUAL_URL&quot;/&gt;&#10;'
              name = 'ServiceTravelAgPort'>
102           <WSDL.Port.binding>
                <WSDL.Binding xmi.idref = 'a6'/>
104           </WSDL.Port.binding>
            </WSDL.Port>
106       </WSDL.Service.port>
        </WSDL.Service>
108   </WSDL.Definition.service>
```

```
    </WSDL.Definition>
110  ***
  </XMI.content>
112 </XMI>
```

Exemple E.2 – Le modèle BPEL (PSM) de l'agence de voyage (fragment)

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
2 <XMI xmi.version = '1.2' timestamp = 'Fri Apr 09 19:43:39 CEST 2004'>
  <XMI.header>
4   <XMI.documentation>
      <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
6     <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
8 </XMI.header>
  <XMI.content>
10   <BPEL.Process xmi.id = 'a1' name = 'ServiceTravelAg' targetNamespace = 'http://
         ServiceTravelAg.org/'
       supressJoinFailure = 'false' enableInstanceCompensation = 'false' abstractProcess = '
           false'>
12     <BPEL.Process.partnerLinks>
         <BPEL.PartnerLinks xmi.id = 'a2'>
14         <BPEL.PartnerLinks.partnerLink>
             <BPEL.PartnerLink xmi.id = 'a3' name = 'ServiceTravelAg' myRole = '
                 ITF_ServiceTravelAgProvider'
16             partnerRole = ''>
               <BPEL.PartnerLink.partnerLinkType>
18               <BPEL.PartnerLinkType xmi.idref = 'a4'/>
               </BPEL.PartnerLink.partnerLinkType>
20           </BPEL.PartnerLink>
                 ***
22         </BPEL.PartnerLinks.partnerLink>
         </BPEL.PartnerLinks>
24     </BPEL.Process.partnerLinks>
       <BPEL.Process.variables>
26       <BPEL.Variables xmi.id = 'a11'>
           <BPEL.Variables.variable>
28           <BPEL.Variable xmi.id = 'a12' name = 'inputServiceTravelAgfindTravel' messageType
                 = 'ITF_ServiceTravelAg_findTravel'/>
             ***
30         </BPEL.Variables.variable>
         </BPEL.Variables>
32     </BPEL.Process.variables>
       <BPEL.Process.activity>
34       <BPEL.Sequence xmi.id = 'a20' suppressJoinFailure = 'false'>
           <BPEL.Sequence.activity>
36           <BPEL.Receive xmi.id = 'a21' name = 'ReceiveFindTravel' suppressJoinFailure = '
                 false'
               portType = 'ITF_ServiceTravelAg' operation = 'findTravel' variable = '
                   inputServiceTravelAgfindTravel'
38             createInstance = 'true' partnerLink = 'ServiceTravelAg'/>
             <BPEL.Flow xmi.id = 'a22' name = 'ForkA' suppressJoinFailure = 'false'>
40             <BPEL.Flow.activity>
                 <BPEL.Sequence xmi.id = 'a23' suppressJoinFailure = 'false'>
42                 <BPEL.Sequence.activity>
                     <BPEL.Assign xmi.id = 'a24' name = 'AssignInputFlight' suppressJoinFailure
                         = 'false'>
44                     <BPEL.Assign.copy>
                         <BPEL.Copy xmi.id = 'a25'>
46                         <BPEL.Copy._to>
                             <BPEL._To xmi.idref = 'a26'/>
48                         </BPEL.Copy._to>
                           <BPEL.Copy._from>
50                           <BPEL._From xmi.idref = 'a27'/>
                           </BPEL.Copy._from>
```

```
52                        </BPEL.Copy>
                        </BPEL.Assign.copy>
54                    </BPEL.Assign>
                      <BPEL.Invoke xmi.id = 'a28' name = 'CallFlight' suppressJoinFailure = '
                          false'
56                      portType = 'ITF_ServiceAirLines' operation = 'findFlight' inputVariable
                            = 'inputServiceAirLinesfindFlight'
                        outputVariable = 'outputServiceAirLinesfindFlightResponse' partnerLink =
                            'ServiceAirLines'/>
58                  </BPEL.Sequence.activity>
                  </BPEL.Sequence>
60                ***
                  </BPEL.Sequence>
62              </BPEL.Flow.activity>
              </BPEL.Flow>
64            <BPEL.Assign xmi.id = 'a41' name = 'AssignReturn' suppressJoinFailure = 'false'>
                <BPEL.Assign.copy>
66                <BPEL.Copy xmi.id = 'a42'>
                    <BPEL.Copy._to>
68                    <BPEL._To xmi.idref = 'a43'/>
                    </BPEL.Copy._to>
70                  <BPEL.Copy._from>
                      <BPEL._From xmi.idref = 'a44'/>
72                  </BPEL.Copy._from>
                  </BPEL.Copy>
74              </BPEL.Assign.copy>
              </BPEL.Assign>
76            <BPEL.Reply xmi.id = 'a45' name = 'ReplyFindFlight' suppressJoinFailure = 'false'
                partnerLink = 'ServiceTravelAg' portType = 'ITF_ServiceTravelAg' operation = '
                    findTravel'
78              variable = 'outputServiceTravelAgfindTravelResponse'/>
              <BPEL.Terminate xmi.id = 'a46' name = 'EndProcess' suppressJoinFailure = 'false'/>
80          </BPEL.Sequence.activity>
          </BPEL.Sequence>
82      </BPEL.Process.activity>
    </BPEL.Process>
84  ***
  </XMI.content>
86 </XMI>
```

ANNEXE F

Utilisation de MMT : exemple illustratif

1. Nous créons un projet sous « Eclipse » et nous importons les métamodèles (UML et Java) dans ce projet. Nous utilisons un guide (*wizard*) pour créer le modèle de correspondance (figure F.1) et nous choisissons le nom du modèle de correspondance (figure F.2), le codage du fichier de correspondances (par exemple, Unicode et UTF- 8) et les fichiers des métamodèles au format XMI (figure F.3).
2. Les métamodèles à gauche (UML) et à droite (Java) sont chargés à partir de fichiers XMI, et le modèle de correspondances est créé (figure F.4). Il contient les éléments suivants : `Historic`, `Definition`, `left`, `right` et `MetamodelHandlers`. Pour chaque élément `MetamodelHandler`, des éléments `ElementHandlers` représentant des références aux éléments des métamodèles en correspondance sont créés.
3. Nous éditons le modèle de correspondance (figure F.5). Nous commençons d'abord par créer les correspondances entre les éléments des deux métamodèles. Ensuite, pour chaque correspondance, des correspondances imbriquées sont créées. Enfin, pour chaque correspondance imbriquée, on désigne un élément de gauche et un ou plusieurs éléments de droite. De plus, chaque élément de gauche ou droite a un `ElementHandler`. Si nécessaire, l'élément `TypeConverter` est utilisé pour expliciter une conversion entre les éléments en correspondance par le biais d'une expression OCL.
4. Le modèle de correspondance peut être validé conformément à son métamodèle et peut être utilisé pour générer une définition de transformation, en utilisant ATL par exemple (figure F.6 et F.7).

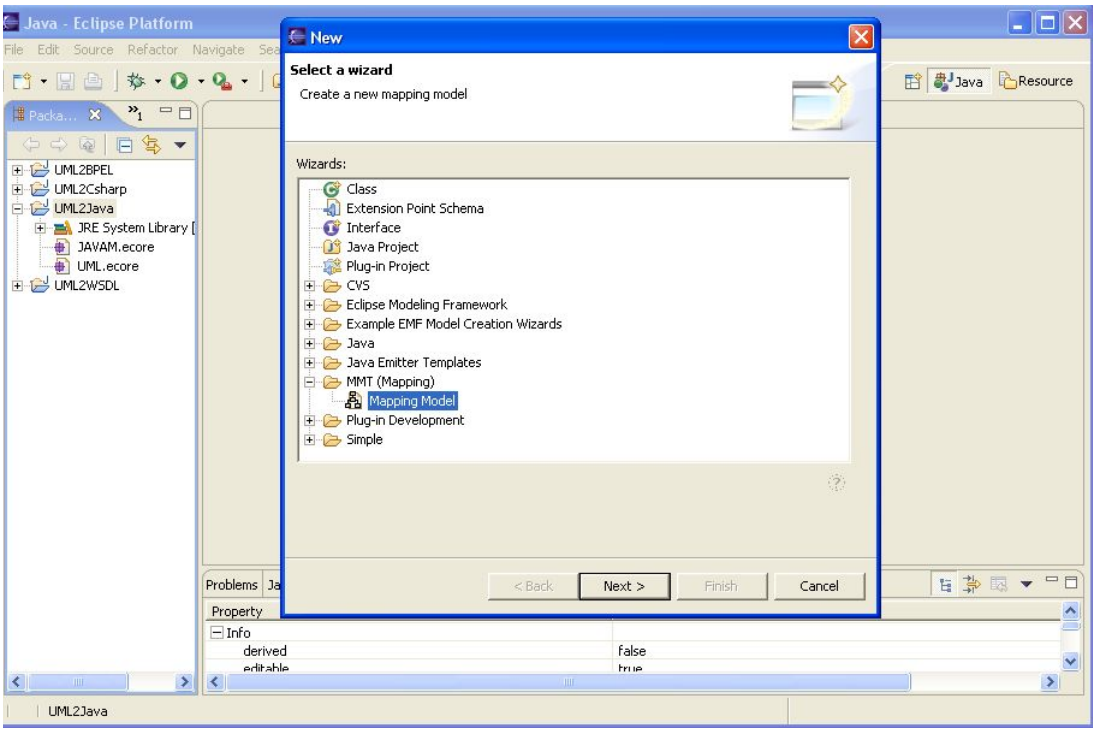


Figure F.1 – La création d'un modèle de correspondances: étape 1/3

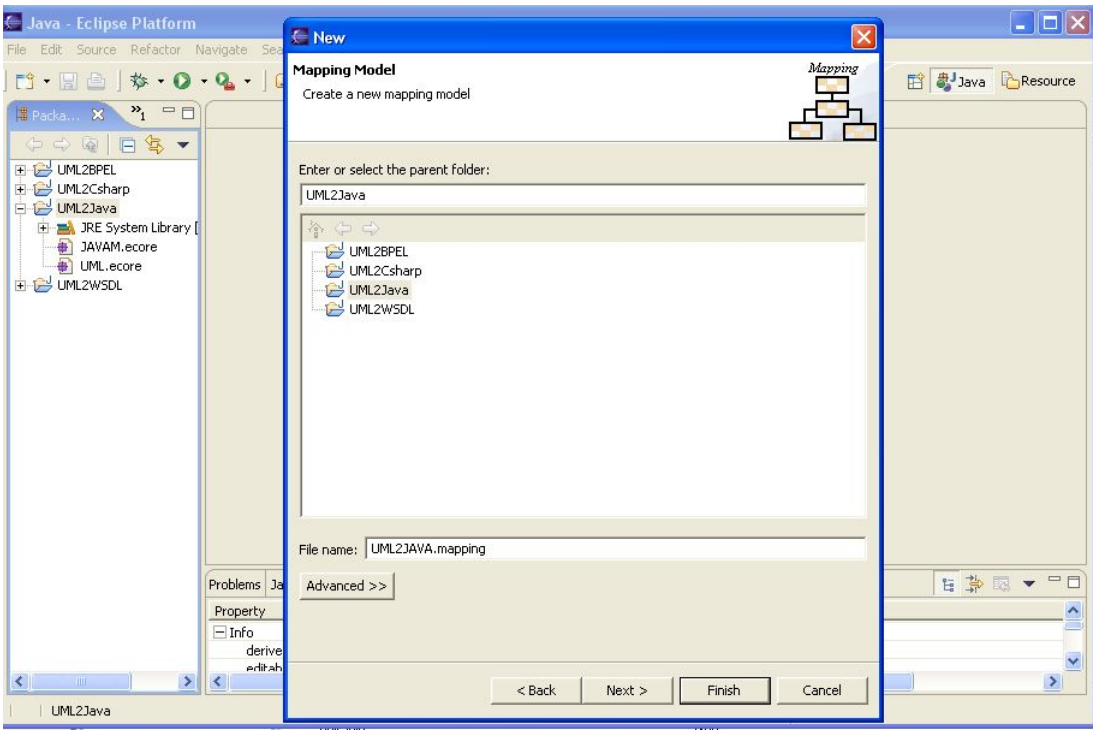


Figure F.2 – La création d'un modèle de correspondances: étape 2/3

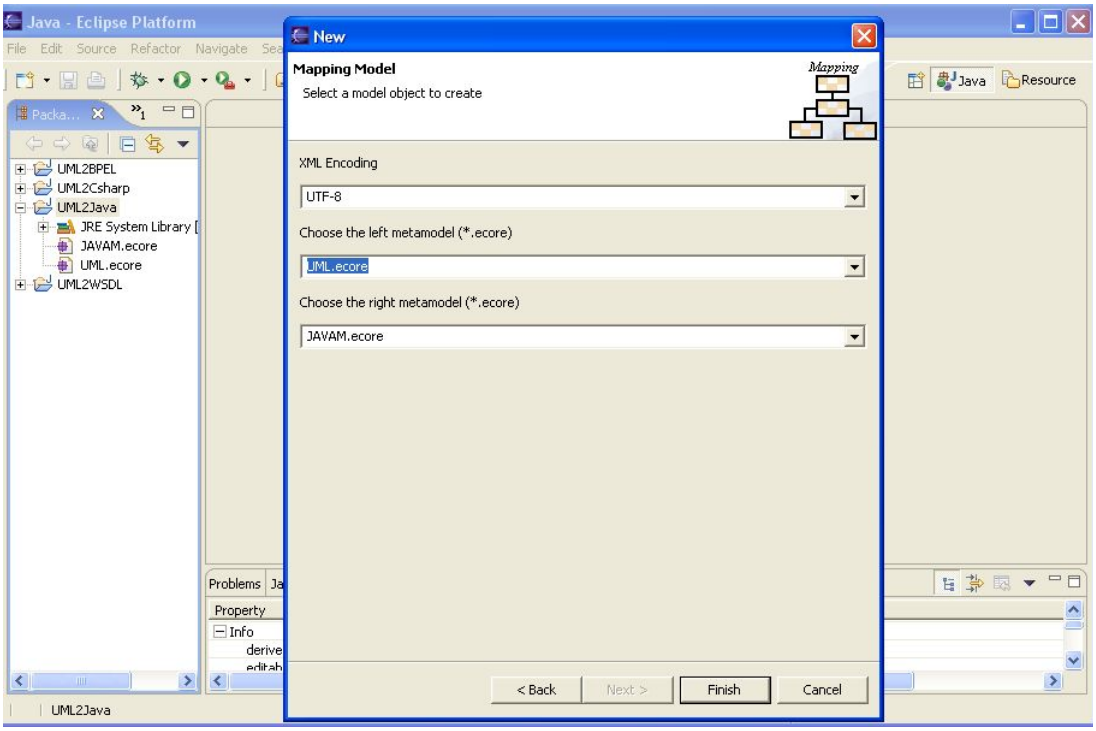


Figure F.3 – La création d'un modèle de correspondances: étape 3/3

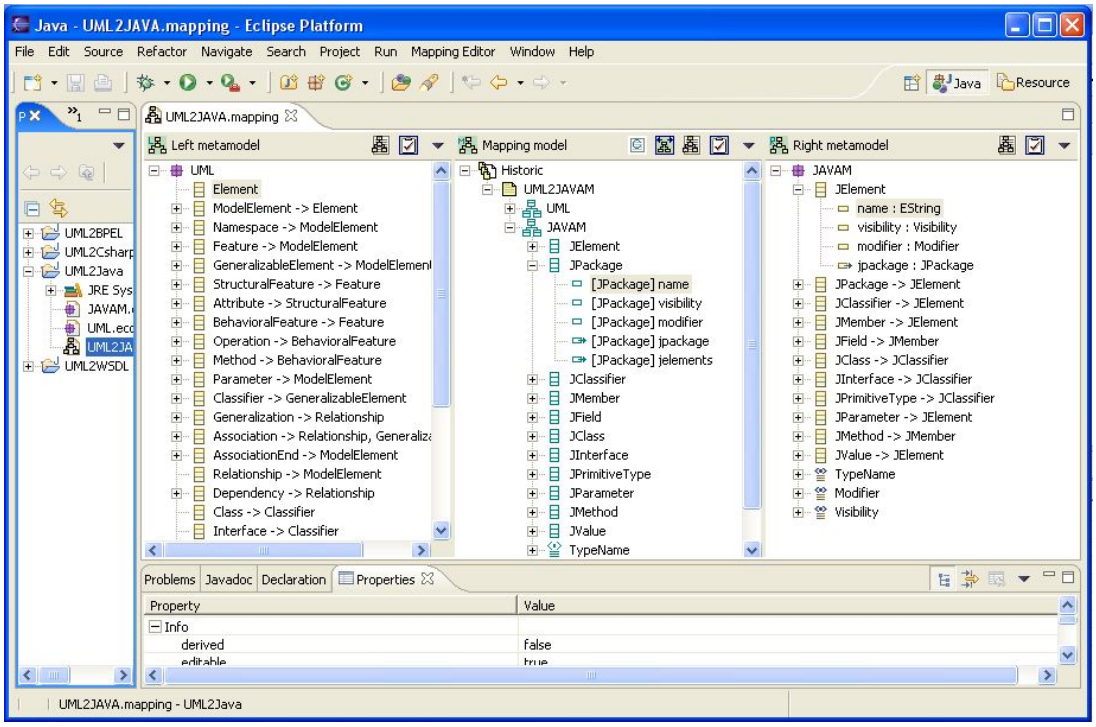


Figure F.4 – Les métamodèles à gauche (UML), à droite (Java) et au centre la correspondance

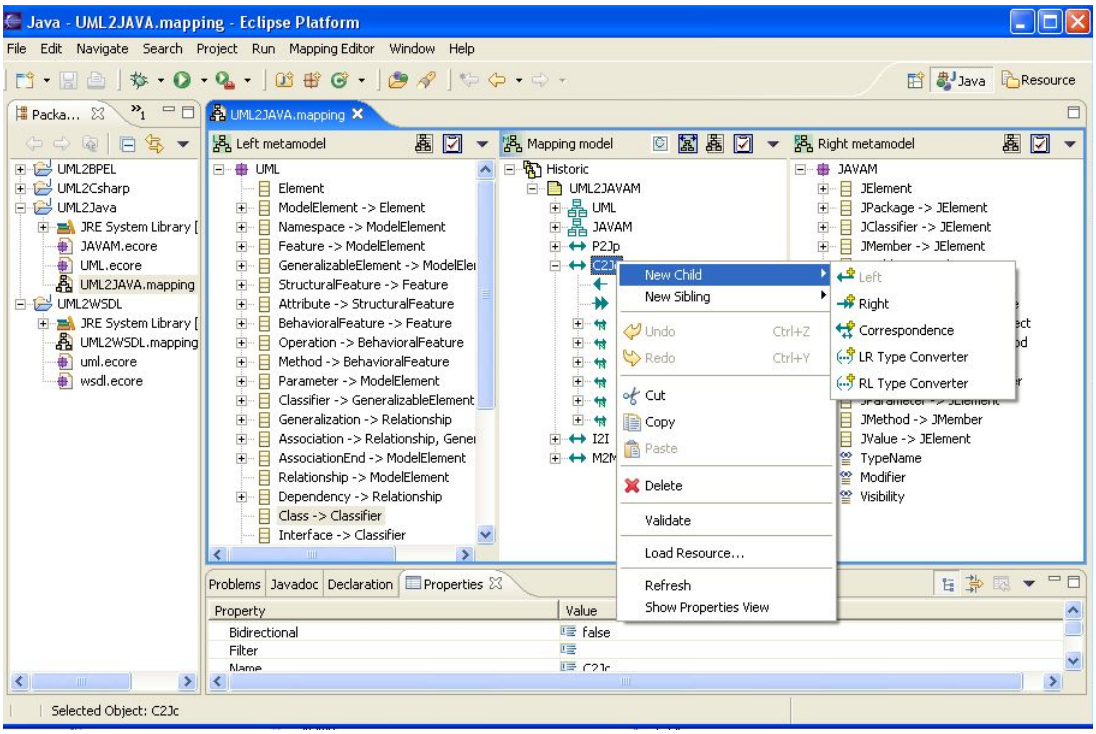


Figure F.5 – L'éditon d'un modèle de correspondance

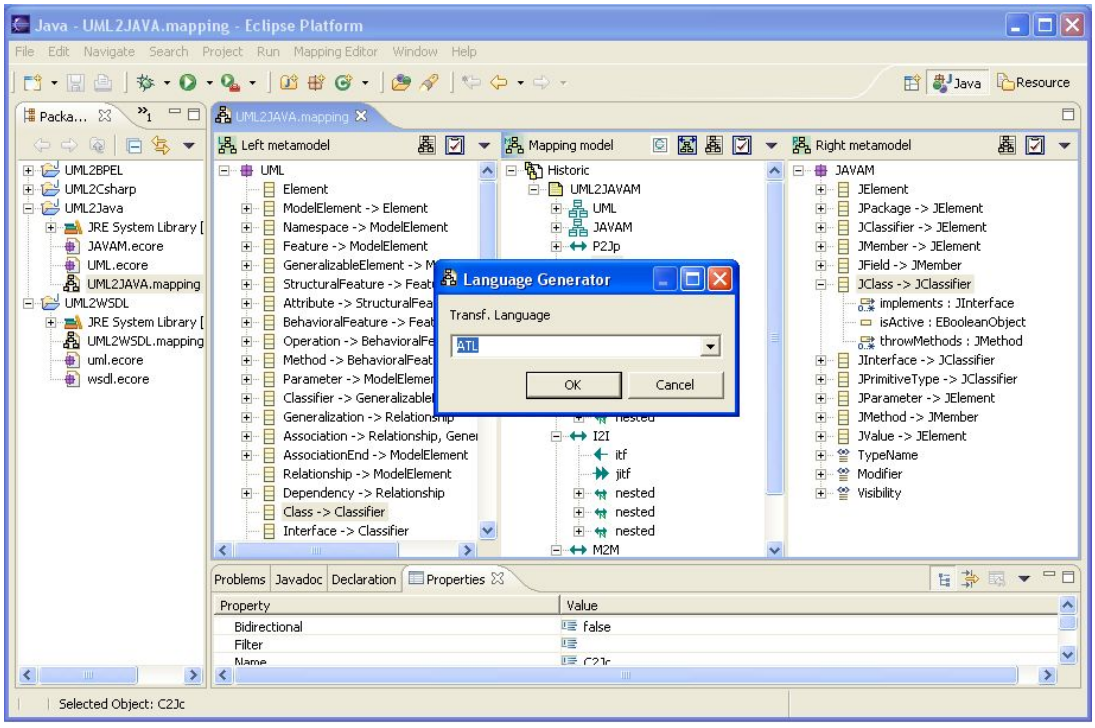


Figure F.6 – Le choix d'une langage de transformation

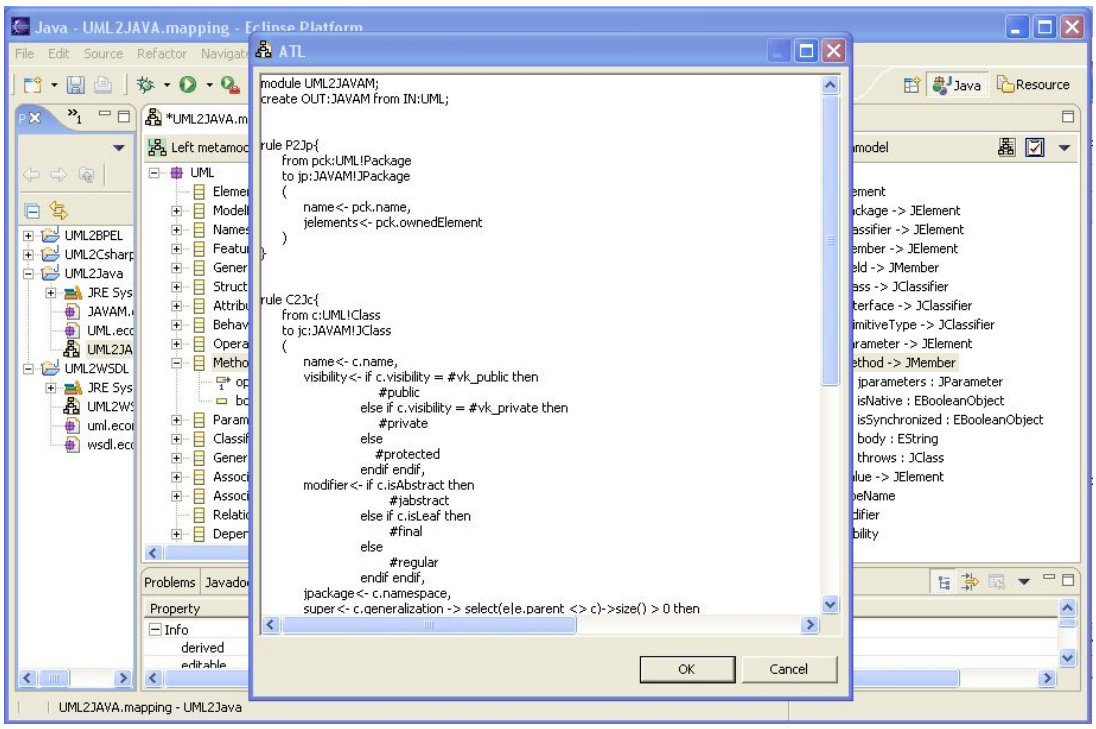


Figure F.7 – La définition de transformation en ATL : de UML vers Java

ANNEXE G

Acronymes

ASP.NET	*Active Server Pages dotNET*
ATL	*Atlas Transformation Language*
ATL	*ATLAS Transformation Language*
B2B	*Business to Business*
B2C	*Business-to-Customer*
BOTL	*Bidirectional Object oriented Transformation Language*
BPEL4WS	*Business Process Execution Language for Web Services*
BPEL	*Business Process Execution Language* (= BPEL4WS)
CCA	*Component Collaboration Architecture*
CCM	*CORBA Component Model*
CDR	*Common Data Representation*
CORBA	*Common Object Request Broker Architecture*
CWM	*Common Warehouse Metamodel*
DCE	*Distributed Computing Environment*
DCOM	*Distributed Component Object Model*
DSL	*Domain Specific Langage*
DTD	*Document Type Definitions*
EAI	*Enterprise Application Integration*
EDI	*Electronic Data Interchange*
EIS	*Enterprise Information System*
EJB	*Enterprise JavaBeans*
EMF	*Eclipse Modeling Framework*
FCM	*Flow Composition Model*
GIOP	*General Inter-ORB Protocol*
HTML	*HyperText Markup Language*
IDC	*International Data Corporation*
IDL	*Interface Definition Language*
IIOP	*Internet Inter-ORB Protocol*
IT	*Information Technology*
J2EE	*Java Enterprise Edition*
Java RMI	*Java Remote Methode Invocation*
JDBC	*Java DataBase Connectivity*

JMI	*Java Metadata Interface*
JRMP	*Java Remote Method Protocol*
JSP	*Java Server Pages*
JWSDP	*Java Web Service Developer Pack*
KMF	*Kent Modelling Framework*
LHS	*Left Hand Side*
MDA	*Model Driven Architecture*
MDE	*Model Driven Engineering*
MDR	*MetaData Repository*
MIA	*Model-in-Action*
MMT	*Mapping Modeling Tool*
MOF	*Meta-Object Facility*
MOLA	*MOdel transformation LAnguage*
OASIS	*Organization for the Advancement of Structured Information Standards*
OCL	*Object Constraint Language*
ODBC	*Open DataBase Connectivity*
OMG	*Object Management Group*
OMT	*Object Modeling Technique*
ORB	*Object Request Broker*
PC	*Personal Computer*
PHP	*Pesonal Home Page*
PIM	*Platform-Independent Model*
PSM	*Platform-Specific Model*
QVT	*Query/Views/Transformations*
RFP	*Request For Proposal*
RHS	*Right Hand Side*
RPC	*Remote Procedure Call*
RVM	*Rimu Visual Mapper*
SAML	*Security Assertions Markup Language*
SDTS	*Syntax-Driven Translation Scheme*
SGML	*Standard Generalized Markup Language*
SMTP	*Simple Mail Transport Protocol*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SOC	*Service Oriented Computing*
TPL	*Template Pattern Transformation Language*
UDDI	*Universal, Description, Discovery and Integration*
UML	*Unified Modeling Language*
URI	*Uniform Resource Identification*
URL	*Uniform Resource Locator*
W3C	*World Wide Web Consortium*

WAPI	*Workflow Application Programming Interface*
WfMS	*Workflow Management Systems*
WSA	*Web Service Architecture*
WS-CDL	*Web Service Choreography Description Language*
WSDL	*Web Services Description Language*
WS-I	*Web Services Interoperability Organization*
WYSIWYG	*What You See Is What You Get*
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*
XP	*eXtreme Programming*
XPath	*XML Path Language*
XPDL	*XML Process Definition Language*
XSD	*XML Schema*
XSLT	*Extensible Stylesheet Language Transformations*
YATL	*Yet Another Transformation Language*

ANNEXE H

Glossaire

Abstraction est « une description de quelque chose qui omet certains détails non pertinents pour l'objectif de l'abstraction » [138].

ATL (*Atlas Transformation Language*) est un langage pour la réalisation de transformations de modèles dans le contexte de MDA [32].

B2B (*Business-to-Business*) est l'échange des services, de l'information et/ou des produits d'un *business* à un autre *business* (Cf. Webopedia http://www.webopedia.com).

B2B e-commerce (*Business-to-Business Electronique-Commerce*) est le terme utilisé pour désigner un *business* qui communique avec autre *business* généralement dans une relation commerciale via le Web (Cf. TechEncyclopedia http://www.techweb.com/encyclopedia).

B2C (*Business-to-Consumer*) est l'échange des services, de l'information et/ou des produits d'un *business* à un consommateur (Cf. Webopedia http://www.webopedia.com).

BPEL4WS (*Business Process Execution Language for Web Services*) est un langage de composition de services [9].

Composition des Services Web est un processus par lequel un service Web est créé selon un arrangement d'autres services Web.

CORBA (*Common Object Request Broker Architecture*) est une spécification et une architecture pour la création et la gestion d'applications orientées objet distribuées sur un réseau. La spécification de CORBA est indépendante d'une implémentation, du langage de programmation et du système d'exploitation [143].

Correspondance (*mapping*) terme utilisé dans cette thèse pour désigner les inter-relations existantes entre les éléments d'un modèle (ou métamodèle) et ceux d'un autre modèle (ou métamodèle).

Définition de transformations contient des règles précises pour transformer un modèle en un autre modèle, en utilisant un langage de transformation exécutable.

DCOM (*Distributed Component Object Model*) est un intergiciel développé par Microsoft qui permet de développer de composants distribués pour la plate-forme Windows.

Distance sémantique « *The notion of* ***semantic distance*** *was developed to cover the notion of « how close is close enough ». (...) The concept of semantic distance is likely to play a major role in some way in the future of virtual enterprise integration and incidentally the semantic web (and other applications). But it is too early to guess in exactly what form, as there are all sorts of market, other economic and political forces at work* » [76].

dotNET (ou .NET) « *Microsoft's framework for Web Services and component software introduced in 2000. (...) dotNET supports all the Web-based features and functions, including XML and the Web services protocols such as SOAP and UDDI. (...) dotNET introduced a new programming language environment that compiles all source code into an intermediate language. dotNET languages are compiled into the Microsoft Intermediate Language (MSIL), which is executed by the Common Language Runtime (CLR) software in the Windows computer. The MSIL is similar to Java's bytecode, except that whereas Java is one language, dotNET supports multiple programming languages such as Microsoft's C# and VB.NET. A subset of the CLR has been standardized by ECMA so that third parties can port non-Microsoft programming languages and create runtime environments for operating systems other than Windows.* » (Cf. TechWeb http://www.techweb.com/encyclopedia)

DSL (*Domain Specific Langage*) DSLs sont *languages that instead of being focused on a particular technological problem such as programming, data interchange or configuration, are designed so that they can more directly represent the problem domain which is being addressed* [51].

Ecore est un langage de métamodélisation qui fait partie d'EMF (*Eclipse Modeling Framework*).

EDOC (*Enterprise Distributed Object Computing*) est une spécification pour le développement de composants basés sur les systèmes EDOC à travers un *framework* de modélisation [145].

Intergiciel (*middleware*) est un logiciel de connexion consistant en un groupe de services qui permettent l'exécution de plusieurs applications sur une ou plusieurs machines connectées en réseau. Cette technologie fournit une couche d'abstraction qui permet l'interopérabilité entre différents langages de programmation, systèmes d'exploitation et architectures d'ordinateurs [195].

J2EE (*Java 2 Platform, Enterprise Edition*) « est un *framework* pour le langage de programmation Java de Sun plus particulièrement destiné aux applications d'entreprise. Dans ce but, il contient un ensemble d'extension au *framework* standard afin de faciliter la création d'applications réparties. Voici une liste des API contenues dans J2EE : Servlets, JSP, JSF, EJB, JNDI, JDBC, JMS, JAXP, JAXM, JAX-RPC, JAXB » (Cf. Wikipédia http://fr.wikipedia.org/wiki). JWSDP utilise également les API : JAXP, JAXM, JAX-RPC, JAXB et JAXR.

Java RMI (*Java Remote Methode Invocation*) est un intergiciel spécifique à la plate-forme Java [184].

Langage de modélisation est une spécification formelle bien définie qui contient les éléments de base pour construire des modèles.

MDA (*Model Driven Architecture*) est l'approche dirigée par les modèles proposée par l'OMG. MDA est une démarche de développement basée sur les modèles et un ensemble de standards de l'OMG. Cette démarche permet de séparer les spécifications fonctionnelles d'un système (PIM) des spécifications de son implémentation (PSM).

MMT (*Mapping Modeling Tool*) est un outil qui permet d'une part, la création de spécification de correspondances entre deux métamodèles et, d'autre part, la génération de définition de transformations [113][114].

Modèle est « une simplification de quelque chose qui nous permet de voir, manipuler, et raisonner sur le sujet étudié, et qui nous aide à en comprendre la complexité inhérente » [122].

MOF (*Meta-Object Facility*) est un langage abstrait et un *framework* pour la spécification, la construction, et la gestion de métamodèles neutres de technologie.

OASIS *(Organization for the Advancement of Structured Information Standards)* est un consortium international qui conduit le développement, la convergence, et l'adoption des normes d'e-business. OASIS a été fondé en 1993, il a plus de 4.000 participants représentant plus de 600 organismes et différents membres dans 100 pays. Ce consortium est responsable par la normalisation de UDDI, SAML, BPEL, etc (http://www.oasis-open.org).

OMG (*Object Management Group*) est une organisation internationale créé en 1989 avec l'objectif de promouvoir la théorie et la pratique de la technologie orientée objet dans le développement de logiciels (http://www.omg.org).

PIM (*Platform-Independent Model*) est un modèle indépendant d'une plate-forme technologique telles que CORBA, Services Web, J2EE et dotNET.

Processus MDA processus spécifique pour la mise en ouvre d'un projet MDA.

PSM (*Platform-Specific Model*) est un modèle dépendant d'une plate-forme technologique telles que CORBA, Services Web, J2EE et dotNET.

Service est une ressource abstraite qui représente des possibilités d'accomplir des tâches qui assurent une fonctionnalité cohérente du point de vue des entités fournisseur et demandeur [211].

Services Web « est une manière standardisée d'intégration des applications basées sur le Web en utilisant les standards ouverts XML, SOAP, WSDL, UDDI et les protocoles de transport de l'Internet. XML est utilisé pour représenter les données, SOAP pour transporter les données, WSDL pour décrire les services disponibles, et UDDI pour lister les fournisseurs de services et les services disponibles » (Cf. Webopedia http://www.webopedia.com).

SOA (*Service Oriented Architecture*) est un ensemble de composants qui peuvent être appelés, et dont les descriptions d'interfaces peuvent être éditées et découvertes [211].

SOAP (*Simple Object Access Protocol*) fournit une définition des informations représentées en XML qui peuvent être utilisées pour échanger des informations structurées et typées entre les participants dans un environnement distribué et décentralisé [204].

Spécification de correspondances présente la logique utilisée pour établir les relations entre deux métamodèles. En effet, la spécification de correspondances peut être considérée comme un PIM, et la définition de transformations comme un PSM.

Transformation de modèles est « le processus de conversion d'un modèle en un autre modèle du même système » [147].

UDDI (*Universal, Description, Discovery and Integration*) fournit la définition d'un ensemble de services qui permettent la description et la découverte (1) des entreprises, des organismes, et d'autres fournisseurs de Services Web, (2) des Services Web qu'ils rendent disponibles, et (3) des interfaces techniques qui peuvent être utilisées pour accéder à ces services [192].

UML (*Unified Modeling Language*) est le langage unifié de modélisation. UML est basé sur les fondements de l'orientation objets : la notion de classe, l'objet, les attributs, les méthodes et les relations entre les classes ou les objets [149].

W3C (*World Wide Web Consortium*) est un consortium international où les organismes membres, le personnel à temps plein, et le public travaillent ensemble pour développer des normes de la Web. Ce consortium est responsable par la normalisation de XML, HTTP, WSDL, SOAP, Semantic Web, etc (http://www.w3.org).

WSDL (*Web Services Description Language*) fournit un modèle et un format XML pour décrire des Services Web [205].

XMI (XML *Metadata Interchange*) permet l'échange de modèles sérialisés en XML.

XML (*eXtensible Markup Language*) est un métalangage de représentation de données. Il définit un ensemble de règles de formatage pour composer des données valides. XML est une mise en forme de texte simple et très flexible dérivée de SGML (*Standard Generalized Markup Language*) (ISO 8879) [208].

XP (*eXtreme Programming*) est une approche délibérée et disciplinée pour le développement de logiciels. Cette approche est basée sur des règles simples et pratiques [53].

Index

Étude et applications de l'approche MDA pour des plates-formes de Services Web

Denivaldo Cicero Pavão LOPES

Résumé

Les services Web sont des technologies émergentes et prometteuses pour le développement, le déploiement et l'intégration d'applications Internet. Ces technologies, basées sur XML, fournissent une infrastructure pour décrire (WSDL), découvrir (UDDI), invoquer (SOAP) et composer (BPEL4WS) des services. Un des avantages majeurs des services Web par rapport à ses prédécesseurs tels que CORBA, DCOM et XML-RPC est l'apport de l'interopérabilité sur Internet. Cependant, les services Web ne sont pas capables de résoudre tous les problèmes. Actuellement, les systèmes d'information sur Internet sont créés en utilisant différentes technologies (par exemple, intergiciels et langages de programmation). Ils évoluent constamment et peuvent être intégrés avec des technologies anciennes. Ces facteurs rendent le développement et la maintenance de ces systèmes plus complexes qu'avant. Dans ce contexte, les services Web sont un intergiciel de plus et d'autres verront le jour. Un axe de recherche prometteur pour le développement d'applications Internet sur les plates-formes services Web consiste à séparer les aspects indépendants et dépendants de la plate-forme par une description séparée de leurs modèles. Cette tendance est mise en avant par l'approche de l'architecture dirigée par les modèles (MDA - *Model Driven Architecture*), définie par l'OMG (*Object Management Group*). Cependant, avant que ceci devienne une réalité courante, plusieurs questions nécessitent des réponses tels que la transformation de modèles, le choix entre les langages de modélisation et les méthodologies. Dans cette thèse, une démarche conceptuelle et expérimentale de l'approche MDA est utilisée sur les plates-formes de services Web afin d'identifier les problématiques autour de MDA et de proposer des solutions viables. Ainsi, nous utilisons UML (*Unified Modeling Language*) et EDOC (*Enterprise Distributed Object Computing*) comme formalisme pour décrire les aspects indépendants de la plate-forme (modèle métier), et nous proposons des métamodèles pour décrire les aspects dépendants de la plate-forme des services Web, J2EE et dotNET (modèle de plate-forme). Ensuite, nous présentons notre démarche pour passer d'un modèle métier vers un modèle de plate-forme. Les apports de cette thèse sont : la séparation explicite entre spécification de correspondances et définition de transformations ; l'étude de la plate-forme des Services Web dans le contexte de MDA ; la proposition des spécifications de correspondances entre les métamodèles de UML, de EDOC, des services Web, de J2EE et de dotNET ; les définitions de transformations en ATL (*Atlas Transformation Language*) générées à partir de ces spécifications ; un outil nommé MMT (*Mapping Modeling Tool*) pour supporter d'une part, la création de spécifications de correspondances et, d'autre part, la génération de définitions de transformations.

Mots-clés : L'architecture dirigée par les modèles, services Web, EDOC, UML, spécification de correspondances, définition de transformations et outils.

Classification ACM

Catégories et descripteurs de sujets : D.2.12 [**Software**]: Software Engineering—*Data mapping*

Termes généraux : Design, Experimentation, Theory