MeTSI : Une Méthode de Transformation des Services Intentionnels

Yves-Roger Nehan

THESE DE DOCTORAT DE L'UNIVERSITE PARIS I – SORBONNE

Spécialité : **INFORMATIQUE**

Présenté par

Yves-Roger Nehan

Pour l'obtention du titre de :

DOCTEUR DE L'UNIVERSITE PARIS I – SORBONNE

Sujet de la thèse :

MeTSI : Une Méthode de Transformation des Services Intentionnels

Soutenance prévue le 28 Mai 2010 devant le jury composé de :

Mme Carine SOUVEYET	Directeur de thèse
M. Michel LEONARD	Rapporteur
M. André FLORY	Rapporteur
Mme Colette ROLLAND	Examinateur
M. Naveen PRAKASH	Examinateur
M. Camille SALINESI	Examinateur

REMERCIEMENTS

Je voudrais tout d'abord exprimer mes vifs remerciements à Colette Rolland, Professeur à l'Université de Paris 1 Panthéon - Sorbonne pour la confiance qu'elle m'a témoignée en m'accueillant dans son équipe et en acceptant la direction scientifique de mes travaux. Je lui suis reconnaissant de m'avoir fait bénéficier tout au long de ce travail de sa grande compétence, de sa rigueur intellectuelle, de son efficacité et de ses précieux conseils.

Je remercie vivement Carine Souveyet, Professeur à l'Université de Paris 1 Panthéon - Sorbonne pour avoir accepté d'encadrer cette thèse et pour avoir contribuer à son aboutissement en apportant ses compétences dans le domaine de l'ingénierie des services.

Je remercie sincèrement Michel Léonard, Professeur à l'Université de Genève, et André Flory, Professeur à l'Institut National des Sciences Appliquées de Lyon, qui me font la gentillesse d'accepter les rôles de rapporteurs.

Je remercie également Naveen Prakash, Professeur à l'Université de MRCE faridabad, Inde et Camille Salinesi, Maître de conférence à l'Université de Paris 1 Panthéon - Sorbonne pour avoir accepté de faire partie du jury de cette thèse.

Je tiens également à remercier Manuele Kirsch Pinheiro, Maître de Conférences à l'université Paris 1 Panthéon-Sorbonne, pour sa disponibilité, ses conseils et son soutien. Ses nombreux encouragements ainsi que les nombreuses discussions ont été importants pour l'aboutissement de ces travaux.

Je suis très reconnaissant à Rebecca Deneckère, Maîtres de Conférences à l'université Paris 1 Panthéon-Sorbonne, qui a eu le courage et la patience de relire des chapitres de mon mémoire.

Un grand merci particulier à mes amis et collègues de l'équipe du Centre de Recherche en Informatique – CRI de l'université Paris 1 Panthéon-Sorbonne.

Enfin, je remercie de tout mon cœur mes proches, amis et famille et plus particulièrement mon adorable épouse Didelle, pour leur confiance, leur soutien et leur aide inconditionnels ainsi que leur présence inestimable à mes cotés durant toute la durée de cette thèse.

TABLE DES MATIERES

CHAPITRE 1 : INTRODUCTION GENERALE

1. Contexte

La thèse s'inscrit dans le domaine de l'ingénierie des applications orientées services.

Le paradigme SOC

Le paradigme SOC (*Service-Oriented Computing*) [Papazoglou *et* Georgakopoulos 2003] émerge comme un nouveau paradigme prometteur dans le domaine des systèmes d'information. Il se concentre sur la notion de service comme élément fondamental pour le développement des applications logicielles. Les services sont des composants auto-descriptifs qui permettent la composition rapide et à faible-coût des applications distribuées. Les services sont offerts par les fournisseurs de services qui procurent aux clients les implémentations et la maintenance des services, ainsi que leurs descriptions.

L'approche SOC repose sur trois acteurs majeurs, les clients, les fournisseurs et les annuaires. Les fournisseurs publient leurs services dans des annuaires, tandis que les clients interrogent ces mêmes annuaires pour découvrir les services.

Les infrastructures de services Web s'appuient sur différent standards, dont WSDL (*Web Services Description Language*) [W3C 2001], SOAP (*Simple Object Access Protocol*) [W3C 2001] et UDDI (*Universal Description and Discovery Interface*) [UDDI 2006]. WSDL est un langage basé sur XML pour la description de ce qu'est un service Web et comment l'invoquer. SOAP est un protocole de communication pour l'échange de messages sur HTTP entre le client et le fournisseur. UDDI permet la définition des annuaires dans lesquels la description des services est publiée.

La vision SOC permet ainsi aux développeurs d'accroître dynamiquement leurs portefeuilles d'applications plus vite que dans les approches d'ingénierie traditionnelle en créant des solutions par composition. Celles-ci combinent des composants organisationnels, incluant des systèmes d'information d'entreprise et des anciens systèmes (« *legacy systems* »), et des

composants externes disponibles à travers les réseaux distants. La vision du SOC permet un assemblage facile des composants applicatifs à travers un réseau de services faiblement couplés.

L'indépendante aux plateformes qui caractérise la notion de service crée ainsi une opportunité pour la composition des nouveaux services plus complexes à partir des services existants, offerts par les différents fournisseurs de services [Yang *et al.* 2003]. Ces nouveaux services peuvent être construits dans le but de traiter des transactions complexes, comme la vérification de crédit, la commande de produit ou l'approvisionnement.

Les méthodologies d'ingénierie d'application orientée service

Le domaine des services a été et est l'objet de nombreuses recherches au point que de nombreux séminaires et conférences y sont consacrés [ICSOC 2008] [IEEE SCC 2009]. Les sujets traités sont variés et comportent entre autres, l'étude du concept de services, les architectures à base de services, la composition de services, les mécanismes de contrôle de l'exécution distribuée d'applications à base de services, et les méthodes de conception de ces applications. Les méthodologies de développement des applications orientées services se concentrent sur l'analyse, la conception et la production de services et tentent très difficilement de s'aligner sur les interactions métiers [Papazoglou 2006].

Plusieurs auteurs ont proposé des méthodologies orientées services basées sur l'approche SOC. Parmi ces méthodologies, nous pouvons citer celle de [Belhajjame *et al.* 2006]. Les auteurs de cette approche explorent les possibilités d'utilisation d'une source additionnelle d'information sous forme d'annotations sémantiques.

Un deuxième exemple de modélisation de services est proposé par [Ryu *et al.* 2008]. Ces auteurs mettent en place un framework qui aide la gestion de l'évolution du protocole métier. Ce framework fournit aux gestionnaires de services plusieurs caractéristiques, telles que l'analyse d'impact d'un changement de protocole, lequel détermine quelles instances en cours d'exécution peuvent être migrées vers une nouvelle version de protocole métier. Par définition, un protocole métier spécifie les séquences de messages qu'un service et ses clients échangent pour réaliser un but métier précis [Alanso *et al.* 2004], par exemple, réserver un billet d'avion. La motivation de cette approche vient du fait qu'il est souvent impossible d'abandonner toutes les conversations en cours et de demander aux clients (utilisateurs) de recommencer l'invocation de service depuis le début de la conversation. C'est donc dans le

but de poursuivre l'exécution des protocoles métiers, malgré leur évolution, que l'approche [Ryu *et al.* 2008] a été proposée.

Notre dernier exemple d'approche méthodologique est celle proposée par [Brogi *et al.* 2008]. En effet, ces auteurs présentent un nouvel algorithme pour la découverte orientée composition de services Web. Cet algorithme, appelé SAM (pour *Service Aggregation Matchmaking*), peut être utilisé pour faire correspondre les requêtes avec les annuaires des services faisant usage des ontologies OWL-S. L'idée est que les requêtes qui ne peuvent pas être satisfaites par un seul et même service mais peuvent l'être par plusieurs compositions de services. L'exemple est celui d'un client qui souhaite planifier ses vacances en réservant un billet d'avion, ainsi qu'un hébergement d'hôtel, en tenant compte des différents paramètres tels que le temps, le prix des saisons, les offres spéciales et bien d'autres.

Notre travail s'inscrit dans la lignée des travaux méthodologiques précédents et à pour objet de définir une méthode de développement d'applications à base de services.

2. Position adoptée dans cette thèse

Il n'y aucun doute sur le fait que SOC soit désormais un standard pour la conception, le développement et le déploiement d'applications logicielles flexibles. Le paradigme sous jacent permet le couplage faible de systèmes disparates et hétérogènes, fonctionnant sur des plateformes hétérogènes qui peuvent évoluer sans que leurs architectures de base soient remises en cause. Force est cependant de constater que SOC est dédiée aux développeurs d'applications : (a) les langages de description des services sont de bas niveau technique; le service étant vu en termes de messages, de formats, de types, de protocoles de transport et d'une adresse physique ; (b) la description d'un service et celle d'une composition de services sont de nature fonctionnelle, faisant référence à des fonctions de base telles qu'envoyer/recevoir un message de commande.

Pour que le paradigme SOC soit transposé au niveau du business de l'entreprise, il est nécessaire d'établir un pont entre les services de haut niveau au sens où le monde du business les comprend et les services techniques et logiciels de bas niveau tels que les développeurs d'applications informatiques les comprennent aujourd'hui. [Arsanjani 2004], [Zimmermann 2004].

Cette extension requiert que l'on s'interroge sur la notion de service au sens du monde du business, sur l'identification et la spécification de tels services et sur la correspondance avec

les services logiciels qui les réalisent. Il nous semble que c'est à ce prix que le paradigme SOC peut porter ses meilleurs fruits.

La position adoptée dans cette thèse est que de tels services de haut niveau, pour être en phase avec le monde du business, doivent être exprimés dans un *mode intentionnel* que nous qualifions de *service intentionnel*, c'est-à-dire par référence à des buts et stratégies que l'organisation choisit pour les atteindre. Nous pensons qu'une telle vue des services peut minimiser la discordance conceptuelle entre la définition des services logiciels et l'énoncé des exigences des utilisateurs en réduisant le décalage pénalisant entre des expressions fonctionnelles telles que celles qu'on formule dans les différent standards, dont WSDL, SOAP et UDDI et celles des besoins organisationnels tels que le monde du business les formule naturellement.
Nous avons retenu la définition de la notion de *service intentionnel* fondée sur le méta-modèle MIS de Kaabi [2007] comme point de départ de notre travail.
Nous avons pour objectif de proposer une méthode de transformation des services intentionnels en applications exécutables construites à base de services logiciels interactifs.
Un service intentionnel MIS se définit par rapport au but que les services logiciels correspondants permettront d'atteindre. Le méta-modèle MIS classe les services intentionnels en deux catégories : agrégat et atomique. Un service intentionnel de type agrégat est utilisé lorsque le but est affiné par un ou plusieurs sous-buts de services intentionnels. Par opposition, le service intentionnel atomique est associé à un but qui n'est pas affinable en sous-buts mais qui est directement opérationnalisable par l'exécution d'un ou plusieurs éléments opérationnels.

L'approche proposée dans cette thèse permet de définir une démarche de développement d'une application à base de services par transformations successives des services intentionnels en services logiciels interactifs.

La méthode appelée **MeTSI** applique les principes de l'ingénierie dirigée par les modèles. Elle adopte une approche transformationnelle, dans le sens où, à chaque étape, les descriptions de services sont des modèles instances de méta-modèles qui sont transformés et affinés. Elle permet d'aboutir à une solution logicielle exécutable sous la forme d'une application à base de services interactifs qui contient toutes les dimensions d'une application, à savoir, l'aspect interface d'interaction de l'utilisateur avec l'application, l'aspect métier réalisé par des services web et leur coopération dans une composition orchestrée de services.

Elle opère par transformations successives de façon à séparer les sujets d'intérêt pour apporter de la flexibilité et permettre l'adaptation, en particulier, à différentes plateformes d'implémentation. Elle réutilise des services web déjà réalisés lorsque c'est possible.

3. Problématique et approche de résolution

Nous étudions dans cette thèse les problèmes posés par la mise en place d'une approche de transformation de services intentionnels.

La thèse considère les trois problèmes suivants que nous développons dans les sections suivantes :

- La transformation de services intentionnels en services opérationnels,
- La transformation de services opérationnels en services logiciels interactifs dépendants d'une plateforme et,
- La génération de code spécifique à une plateforme.

3.1. La transformation de services intentionnels en services opérationnels

Cette problématique est centrée sur la définition de la notion du méta-modèle opérationnel de service - ***MOS*** et du processus de transformation des éléments MIS en MOS.

Le méta-modèle MOS permet de guider l'identification, au niveau opérationnel, de services qui correspondent aux services intentionnels de type MIS. Ceci est réalisé à l'aide de la représentation explicite des services opérationnels qui découlent des services intentionnels dans les termes d'un méta-modèle spécifique MOS (Modèle Opérationnel de Services) et un ensemble de règles méthodologiques pour identifier les services opérationnels à partir des services intentionnels.

Le méta-modèle MOS introduit la notion de *service opérationnel* qui est une unité assemblant des éléments opérationnels de natures différentes. Le service opérationnel remplit les actions nécessaires à l'accomplissement du but du service intentionnel tels que la gestion de la logique métier que le service implémente et l'administration de la coordination des différents agents fournisseurs de services. MOS se focalise sur l'opérationnalisation du service intentionnel atomique. Il est réalisé par l'assemblage de deux types d'éléments : le service d'interface utilisateur qui prend en compte les interactions des utilisateurs et le service métier qui prend en compte la composition de services web.

Il existe une relation bidirectionnelle entre le méta- modèle MIS et le méta-modèle MOS :

- Relation causale de MIS vers MOS : le service intentionnel atomique du méta-modèle MIS est opérationnalisé par un service logiciel du méta-modèle MOS.
- Relation causale de MOS vers MIS : le service opérationnel délivre un résultat considéré comme l'information nécessaire pour que le but du service atomique soit réalisé.

Cette relation bidirectionnelle entre les méta-modèles MIS et MOS permet de faire la transition entre la notion de service d'un niveau intentionnel et celle du niveau opérationnel. Ceci met en avant le fait que les deux notions de service coexistent et coopèrent à la fois au niveau intentionnel et au niveau opérationnel.

La deuxième caractéristique du méta-modèle MOS est son indépendance à une plateforme d'implémentation.

Pour capturer le comportement nécessaire à la réalisation du but du service intentionnel atomique MIS, nous utilisons une approche à base de scénarios inspirée de Crews L'Ecritoire. L'approche Crews L'Ecritoire est fondée sur le couple <but, scénario>. Cette approche guide la description du comportement selon deux activités : concrétiser un but par un scénario et analyser un scénario pour identifier des cas fonctionnels alternatifs ou des cas exceptionnels de non satisfaction du but.

Le choix de l'utilisation d'une approche méthodologique à base de scénarios est motivé par le fait que ceux-ci permettent de se focaliser sur la vision de l'utilisateur, sur ce qui doit se passer, comment et pourquoi. Les scénarios aident les utilisateurs et les développeurs à se poser de nouvelles questions, y trouver des réponses et ainsi explorer de nouvelles possibilités.

3.2. La transformation de services opérationnels en services logiciels interactifs dépendant d'une plateforme

Cette problématique est centrée sur de la définition du méta-modèle d'implémentation de service interactif – *MISI* qui représente la première étape d'une modélisation spécifique à une plateforme dans la méthode MeTSI.

Le méta-modèle MISI permet de guider l'identification, au niveau logiciel, de services qui correspondent aux services opérationnels de type MOS. Ceci est réalisé à l'aide de la représentation explicite des services logiciels interactifs qui découlent des services opérationnels dans les termes d'un modèle spécifique MISI (Modèle d'Implémentation de

Service Interactif) et un ensemble de règles méthodologiques prenant en compte des paramètres caractérisant les différentes plateformes d'implémentation.
Les paramètres caractérisant les plateformes de services interactifs sont modélisés dans un méta-modèle architectural d'infrastructure facetté utilisant des facettes déclinées en attributs. Les facettes d'architecture ont été mise en place pour aider l'agent du modèle opérationnel à faire un choix entre les différentes plateformes. Les facettes comportent des valeurs telles que *stratégie de distribution* ou *acteur d'implémentation*.
Le niveau de transformation MISI est motivé par le besoin de définir une solution architecturale qui est déterminée en fonction de paramètres qui caractérisent une classe d'architectures de même nature. L'effort fait par MISI est d'avoir identifié les paramètres qui caractérisent les différentes classes architecturales et de les avoir modélisées dans un méta-modèle. Le processus de transformation tient évidemment compte des valeurs des paramètres choisis par l'ingénieur d'application pour configurer la solution architecturale en fonction de ces valeurs.

3.3. La génération de code spécifique à une plateforme de service interactif

Cette problématique est centrée sur de la génération de code pour trois plateformes spécifiques. C'est la dernière étape de mise en œuvre de la méthodologie MeTSI. Ces plateformes d'implémentation sont l'OSGi [OSGi 2009], le WSRP [WSRP 2006] et le Mashup [Wikipedia, Mashup 2006].

La plateforme WSRP permet au client d'agréger ses services interactifs (interface utilisateur et coordination de services web) sur une page web unique à partir de différentes fournisseurs de services distants sans se soucier de leur implémentation.
La plateforme Mashup est une application composite qui en toute transparence, combine le contenu d'un ou de plusieurs services web assemblés sur une même page.
Enfin, la plateforme OSGi permet d'avoir les services web localement à partir de téléchargement chez des fournisseurs de services distants. L'exécution de tels services se fait localement chez le client alors que la provenance a été faite de manière distante.

Nous avons choisi ces trois plateformes car elles représentent un panel significatif de l'ensemble des solutions disponibles à ce jour.

Le processus de transformation s'appuie sur des règles spécifiques à chaque plateforme. Ces règles sont automatisées. On peut donc parler de génération de code. Au bout du compte, l'application est générée dans une forme exécutable. Elle comporte les aspects relatifs à

l'interaction avec l'usager, de composition des services et d'encapsulation des règles métiers dans les services.
Dans ce travail, nous avons concrètement utilisé la plateforme de service WSRP qui permet d'exécuter les services logiciels interactifs invoqués.

4. Résultats de la thèse

Ce mémoire de thèse présente cinq résultats de recherches :

- Un méta-modèle de représentation des services opérationnels : le méta-modèle MOS,
- Une démarche pour dériver les services opérationnels à partir des services intentionnels,
- Un méta-modèle d'implémentation des services interactifs dépendant de la plateforme, MISI.
- Une démarche pour transformer les services opérationnels en services logiciels interactifs dépendant d'une plateforme,
- La mise en place de la méthode de transformation des services intentionnels par la génération de code de services logiciels interactifs basés sur la plateforme WSRP.

5. Plan de la thèse

La suite de ce document est organisée en sept chapitres comme suit :

- **Chapitre 2 :** ce chapitre présente un état de l'art des approches orientées services. Cet état de l'art est organisé selon un cadre de référence qui permet de présenter différents aspects du concept de service et de positionner les approches les unes par rapport aux autres.
- **Chapitre 3 :** dans ce chapitre, nous décrivons et justifions la méthode MeTSI qui combine les services intentionnels à haut niveau d'abstraction et l'approche transformationnelle dirigée par les modèles. Dans une seconde partie, nous rappelons le méta-modèle de services intentionnels MIS. Enfin, la troisième partie de ce chapitre, décrit les plateformes d'implémentation retenue dans la thèse. Nous passons en revue toutes les plateformes qui satisfont totalement ou partiellement à des degrés différents notre approche de transformation des services intentionnels.
- **Chapitre 4 :** dans ce chapitre, nous présentons, dans une première partie, le modèle de scénario, ainsi que le méta-modèle des services opérationnels MOS. Ce méta-modèle

définit les éléments caractérisant les services opérationnels. Dans la seconde partie, nous détaillons la transformation MIS – MOS qui permet le passage des services caractérisés par les buts utilisateurs du modèle MIS en services opérationnels du modèle MOS. Enfin, dans une troisième partie, nous définissons un langage de requêtes pour interroger les services recherchés dans un annuaire de services web.

- **Chapitre 5 :** ce chapitre décrit le méta- Modèle d'Implémentation du Système Interactif (MISI) qui implémente le modèle opérationnel MOS à travers le choix d'une architecture d'implémentation. Les alternatives d'architectures d'implémentation possibles sont décrites dans la seconde partie de ce chapitre.
- **Chapitre 6 :** ce chapitre illustre notre approche par une étude de cas nommée e-Pension. Dans cette partie, nous avons opté pour la génération de code basé sur la plateforme de service WSRP qui permet d'invoquer et d'agréger le service interactif sur une page web unique.
- **Chapitre 7 :** ce chapitre conclut la thèse et propose des perspectives de recherche faisant suite à ce travail.

CHAPITRE 2 : ETAT DE L'ART

1. Introduction

Les approches orientées services ont pour objectif de construire des applications distribuées en réutilisant des briques logicielles hétérogènes appelées services. Un *service* peut être vu comme une entité indépendante capable de fournir une certaine fonctionnalité à travers une interface bien définie, laquelle permet son invocation de manière standard, sans que les clients soient obligés de connaître les détails de son implémentation [Issarny *et al.* 2007] [Sassen et al. 2005]. Nous incluons dans la définition d'approches orientées services les approches suivantes :

- Les approches SOC (*Service oriented Computing*) [Papazoglou *et* Georgakopoulos 2003] [Ryu 2008]
- les approches de services intentionnels [Kaabi 2007] [Penserini *et al.* 2007],
- les approches de services interactionnels [Chevrin 2006] et,
- les approches de services contextuels (context-aware) [Maamar *et al.* 2001] [Najar *et al.* 2009].

Les travaux sur les approches orientées services sont variées et traitent différents sujets. Le domaine des services a été et est l'objet de nombreuses recherches, au point que de nombreux séminaires et conférences sont y consacrés [ICSOC 2008] [IEEE SCC 2009]. Les sujets traités sont variés et comportent, entre autres, l'étude du concept de services, les architectures à base de services, la composition de services, les méthodes de conception d'applications à base de services et les mécanismes de contrôle de l'exécution de ces applications.

Ce chapitre a pour but de définir un cadre multidimensionnel utile à l'analyse des approches orientées services, à la compréhension et à la classification des problèmes dans le développement des méthodologies à base de services. Ce cadre de référence s'inspire de [Rolland 1998] et est composé de quatre vues différentes, représentant des points de vue complémentaires. Chacune de ces vues capture un aspect particulier des services, explorant

ainsi différentes caractéristiques des approches de services. Les relations entre les différentes vues montrent comment elles s'influencent les unes des autres.

Dans le but d'étudier, comprendre et classifier une vue particulière des approches orientée services dans sa diversité, il est associé à chaque vue un ensemble de facettes. Chaque facette permet une description en profondeur d'un aspect spécifique des services, les vues montrent la variété et la diversité de ces aspects.

La suite du chapitre est organisée de la façon suivante : la section 2 décrit le cadre de référence pour les approches de services; les sections 3 à 6 expliquent les différentes vues et établissent à la fin une liste de leurs facettes pour la comparaison et l'évaluation des approches orientées services. La section 7 présente et illustre sept approches orientées services les plus récentes, puis analyse chaque approches selon les différentes vues du framework. Un résumé de l'application du cadre de référence à ces approches est présenté à la section 8, tandis que la section 9 propose, en guise de conclusion, l'application du cadre de référence à l'approche MeTSI proposée dans cette thèse.

2. Cadre de référence pour les approches orientées services

Les quatre vues du cadre de référence [Rolland *et al.* 1998], ont prouvé leur efficacité dans l'amélioration de la compréhension des différentes disciplines de l'ingénierie, telles que l'ingénierie des systèmes d'informations [Jarke *et al.* 1992], l'ingénierie des besoins [Jarke et al. 1993], l'ingénierie du processus de développement des systèmes d'information [Rolland 1998], l'ingénierie des méthode [Rolland 1997] et l'ingénierie des méthodes situationnelles [Nehan *et* Deneckere 2007]. Nous croyons qu'un framework similaire peut être utilisé pour aider à comprendre les approches orientées services en tenant compte des techniques, des outils et des méthodes utilisés pour construire et représenter les services. Le but de ce cadre est donc de proposer un outil pour analyser l'état de l'art des approches orientées services.

Le framework que nous considérons ici s'articule autour de quatre vues (objet, but, méthode et développement), lesquelles correspondent aux quatre questions suivantes :

- Qu'est ce qu'une approche orientée service ?
- Comment est représentée une approche orientée service ?
- Comment sont implémentés les concepts des approches orientées services ?
- Pourquoi et comment les approches orientées services sont-elles utilisées ?

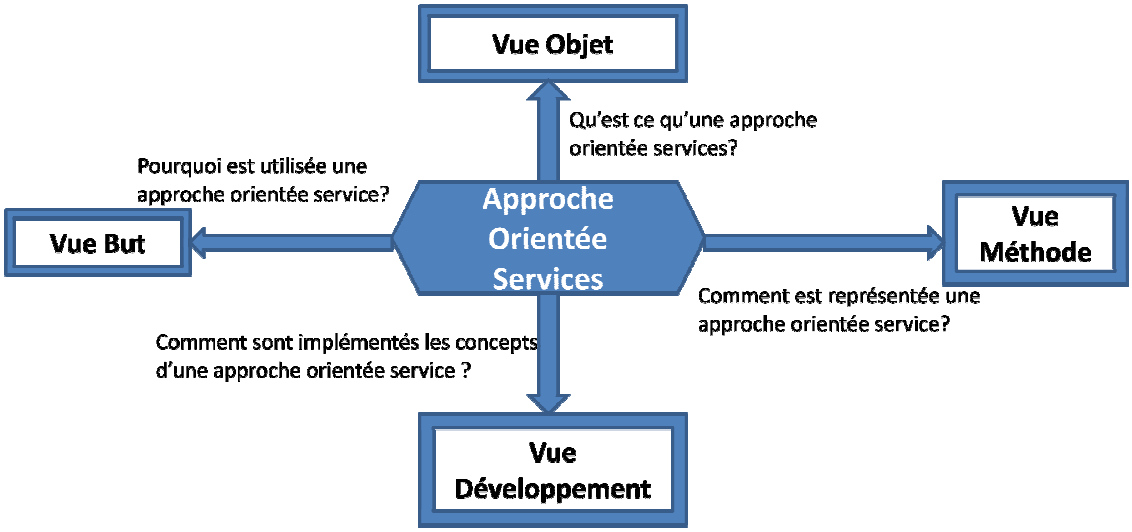


Figure 2.1. Les quatre vues du cadre de référence

Ce framework permet de discuter d'une manière ciblée les différents problèmes à travers les quatre vues : la définition d'une approche orientée service, sa représentation, la manière de développer cette représentation et les justifications pour l'utiliser. Ceci est fait dans les vues objet, but, méthode et implémentation. Chaque vue est décrite par un ensemble de facettes qui permet une étude plus détaillé des approches à base de services.

Chaque vue est caractérisée et mesurée à l'aide d'un ensemble de facettes. Toutes les facettes possèdent des valeurs définies dans un domaine qui peut être un type prédéfini (entier, booléen…), un type énuméré (Enum {x, y}) ou un type structuré (Ensemble ()).

Le domaine de chacun des attributs permettant de décrire les différentes vues est détaillé dans la suite de cette section.

3. La vue objet

On constate aujourd'hui que la littérature scientifique traitant de la notion de service est très hétérogène. Elle se caractérise par une absence d'unification et d'intégration des concepts rendant difficile une appréhension globale et synthétique de ce domaine. Ce phénomène est accentué par la diversité des visions proposées par les différentes communautés de recherche. En effet, des divergences de vues sur le rôle, le type et le contenu apparaissent clairement dans la littérature. Ces aspects sont capturés par les trois facettes suivantes : *définition, réflexivité, spécification de service.* Nous détaillons chacune des facettes dans ce qui suit.

3.1. La facette Définition

La facette *définition* se concentre sur la notion même de service, en considérant les différentes perspectives utilisées lors de la définition d'un service. [Baida 2004] suggère qu'au moins trois perspectives sur les approches orientées services doivent être comprises afin de fournir une terminologie partagée pour les services :

- Une perspective technologique (infrastructure technique) ;
- Une perspective métier (business) ;
- Une perspective applicative.

A ces perspectives initiales, d'autres peuvent être ajoutées. En se référant à [Chevrin 2006], nous pouvons ajouter une perspective interactionnel aux perspectives précédentes. Il s'agit de prendre en compte l'aspect interaction (IHM) qui expose le point de vue du client à travers l'interface utilisateur.

Nous introduisons également la perspective contextuelle (*context-aware*) qui utilise les informations contextuelles afin de fournir aux utilisateurs des informations et des solutions informatiques pertinentes, étant donné leur contexte d'utilisation [Dey *et al.* 2001] [Najar *et al.* 2009].

Finalement, nous introduisons la perspective intentionnelle [Kaabi 2007]. La perspective intentionnelle considère l'intention ou le but d'un acteur dans un contexte donné. La perspective intentionnelle se traduit par le fait qu'un service exhibe une intentionnalité formulée par le *but* qu'il permet à ses clients d'atteindre [Rolland *et* Kaabi 2005].

Les différentes perspectives déjà citées sont détaillés dans ce qui suit :

- **La perspective technologique :**

Dans cette catégorie, nous pouvons citer les approches SOC. L'approche SOC [Papazoglou *et* Georgakopoulos 2003] apparaît comme un paradigme informatique présentant la notion de services comme un élément fondamental pour le développement des applications logicielles. La définition des services est fortement liée aux domaines proches de l'ingénierie du Web et du développement des standards liés aux services Web.

Les services sont des composants auto-descriptifs qui devraient supporter une composition rapide et à faible coût des applications distribuées. Ils sont offerts par les fournisseurs de services qui garantissent leur maintenance et leurs implémentations, ainsi que les descriptions des services fournis. La description d'un service fournit une base pour la découverte et la

composition des services. Les services possèdent la possibilité de s'engager avec d'autres services dans le but d'effectuer des transactions plus complexes comme la vérification de crédit, la commande de produits ou l'approvisionnement.

La vision SOC permet ainsi aux développeurs d'accroître dynamiquement leurs portefeuilles d'applications plus vite en créant des solutions composées. Celles-ci combinent des composants organisationnels, incluant des systèmes d'information d'entreprise et des anciens systèmes (« *legacy systems* »), et des composants externes disponibles à travers les réseaux distants. La vision du SOC permet un assemblage facile des composants applicatifs à travers un réseau de services faiblement couplés. Ceci permettrait de créer des processus métiers dynamiques et des applications agiles, capables de jouer un rôle de lien entre les plateformes informatiques et le processus métier issu de l'organisation.

Les infrastructures de services s'appuient sur les standards WSDL (*Web Services Description Language*) [W3C 2001], SOAP (*Simple Object Access Protocol*) [W3C 2001] et UDDI (*Universal Description and Discovery Interface*) [UDDI 2000]. WSDL est un langage basé sur XML pour la description de ce qu'est un service et comment l'invoquer. SOAP est un standard pour échanger les messages sur http entre les applications. UDDI permet la définition des annuaires globaux ou l'information sur les services est publiée.

- **La perspective métier :**

La facette métier se concentre sur la spécification des exigences des utilisateurs qui sont satisfaites par des services. Au sein de cette perspective, les experts métiers n'ont pas à traiter les détails de la plateforme d'exécution.

Selon [Pallos 2001], un service est un groupement logique de composants requis pour satisfaire une demande métier particulière. Ainsi, de la même manière qu'il est intéressant de mutualiser un ensemble de fonctions techniques indispensables aux applications réparties sur le Web sous forme de services indépendants et standards, il peut être également avantageux de partager des services ayant trait à certaines grandes fonctions de l'entreprise, quel que soit son secteur d'activité. Ces fonctions sont regroupées dans un service métier. Elles visent finalement à reproduire dans le monde virtuel les transactions commerciales du monde réel (transactions, contrats, facturation, paiement, etc.).

Traditionnellement, l'ingénierie des systèmes d'information se concentre sur la modélisation conceptuelle. Celle-ci vise à abstraire la spécification du système requis à partir de l'analyse des informations nécessaires à la communauté des utilisateurs [Rolland 2007]. Bien que la

modélisation conceptuelle permette aux experts techniques de comprendre la sémantique du domaine, elle ne permet pas forcément de construire des systèmes acceptés par la communauté des utilisateurs.

De nombreuses études mettent l'accent sur les risques liés à une mauvaise expression des services dans la perspective métier ou à un manque dans les exigences. Par exemple, [Sawyer *et al.* 2005] exprime le fait que le succès d'un projet de développement est intimement lié à la qualité des exigences, en somme des besoins métiers.

- **La perspective applicative :**

Le point de vue applicatif permet d'établir un pont entre la perspective métier et la perspective technologique. Les approches orientée services dans cette perspective ne sont pas encore totalement mature, car il existe encore peu de méthodologies qui comblent entièrement les deux perspectives. Ces définitions oscillent entre la perspective métier et la perspective technologique. En théorie, la définition de cette perspective est de couvrir la totalité du cycle de développement d'une approche orientée services. Dans la pratique, les approches contribuant à cette perspective ne couvrent que partiellement le cycle de développement : soit l'analyse ou la conception, soit l'implémentation.

Dans cette perspective applicative, nous pouvons citer [Penserini *et al.* 2007] [Penserini *et al,* 2006] qui adoptent la méthodologie de développement orientée agent *Tropos* [Bresciani *et al.* 2004] pour la mise en œuvre d'un processus de développement orientée service. Cette méthodologie repose un processus de développement dirigé par les modèles qui guide le concepteur dans la construction, dans un premier moment, d'un modèle initial des utilisateurs et leurs intentions. Elle adopte, par la suite, une approche transformationnelle, dans le sens où, à chaque étape, les modèles vont être raffinés de manière itérative par l'ajout ou la suppression d'éléments dans les modèles. Le processus de développement soutenu par cette approche est composé alors de cinq phases, dont les quatre premières font partie du processus de conception *Tropos.* Ces cinq phases, dont l'exécution se fait de manière itérative, sont les suivantes : *analyse des besoins initiaux,* l'*analyse des besoins finaux, la conception architecturale, la conception détaillée et l'implémentation.* Chaque phase est caractérisée par des objectifs spécifiques.

Cependant, l'approche [Penserini *et al.* 2007] n'est applicable réellement que durant les phases d'ingénierie des besoins (analyse de besoins initiaux et analyse de besoins finaux). C'est donc au développeur qui revient la charge de créer les entités correspondantes à

l'implémentation au niveau des phases d'analyse et de conception pour pouvoir couvrir le processus de développement de tout le système. La perspective interactionnelle :

Un service de point de vue interactif est vu, d'après [Chevrin 2006] comme une collection de tâches utilisateur. Ces tâches sont regroupées de manière à permettre l'exécution de l'activité requise par un utilisateur. Ce regroupement doit également être cohésif du point de vue du métier. Les travaux contribuant à cette perspective se concentrent ainsi sur les interactions entre l'utilisateur et le système.

- **La perspective intentionnelle :**

La perspective intentionnelle permet d'atteindre l'intention ou le but d'un acteur dans un contexte donné. Selon la granularité de l'intention, la réalisation de celle-ci peut se faire par l'exécution d'un service applicatif et par la composition de services dirigée par les intentions.

L'avantage de ce modèle est qu'il offre une définition des services orientées but, laquelle reste compréhensible aux utilisateurs finaux. Ce modèle offre également un mode de composition et un mode de recherche de service eux-mêmes dirigé par les buts, ainsi qu'un mécanisme d'affinement permettant d'atteindre, à partir des buts complexes, des buts opérationalisables implémentés par des services applicatifs.

Dans cette perspective intentionnelle, nous pouvons citer également [Penserini *et al.* 2007], dont la méthodologie de développement Tropos [Bresciani *et al.* 2004] se base aussi sur la notion de but. Cette approche [Penserini *et al.* 2007] adopte un framework de conception pour modéliser et analyser les besoins initiaux dans l'ingénierie logicielle. Le framework se concentre sur la capture et l'analyse des utilisateurs et de leurs buts, afin de guider la conception du futur système. Cette conception est composée d'agents logiciels qui ont leurs propres buts et leurs capacités et qui sont destinés à supporter la réalisation des buts client.

- **La perspective contextuelle (context-aware) :**

La perspective contextuelle (*context-aware*) des services se réfère à la capacité des applications orientées services de fournir des services pertinents aux utilisateurs par la détection et l'exploration des contextes utilisateurs [Dey *et al.* 2001]. Les systèmes ″sensibles aux contextes″ sont des applications qui s'adaptent à plusieurs situations impliquant les utilisateurs, le réseau, les données et l'application lui-même [Najar *et al.* 2009]. Le contexte est défini comme tout élément d'information qui peut être utilisé pour caractériser la situation d'une entité (une personne, un objet, l'utilisateur…) considérée comme pertinente pour l'interaction entre l'utilisateur et l'application [Dey *et al.* 2001]. Le contexte des utilisateurs

consiste souvent en une collection de paramètres telle que la localisation des utilisateurs, les aspects environnementaux (température, intensité lumineuse, etc.), ou encore leurs activités [Chen *et al.* 2003]. Le contexte des utilisateurs peut changer dynamiquement. Par conséquent, un besoin de base pour un système sensible au contexte est sa capacité de réaction à un changement de contexte, sans intervention de l'utilisateur. Plusieurs travaux proposent des approches orientées services sensibles au contexte. Cependant la plupart des approches existantes [IST-SPICE 2010] [Suraci *et al.* 2007] [Toninelli et al. 2008] restent encore liés aux problèmes d'exécution des systèmes sensibles au contexte sur les plateformes mobiles.

A partir des différences perspectives discutées ci-dessus, nous pouvons classer les approches orientées services selon la facette définition comme suit :

Définition : ENUM : {Technologique, Métier, Applicatif, Interactionnel, Intentionnel, Contextuelle}

3.2. La facette Réflexivité

La composition des services est une caractéristique importante des approches orientées services. La facette réflexivité s'intéresse à cet aspect. La réflexivité peut être perçue comme le fait qu'un service composite soit réutilisable dans d'autres compositions. La composition de services est un processus par lequel un service est créé par l'arrangement d'autres services. Ce nouveau service est appelé service composite et est formé d'un ensemble d'autres services. Dans ce cas, les services qui ont été utilisés sont cachés et réutilisés par le service composite.

[Yang 2003] propose le concept de « *composant service* » comme un mécanisme qui permet de combiner les services existants afin de développer de nouvelles applications. De nature réflexive, un « composant service » peut être réutilisé dans d'autres compositions. Une approche qui ne permet pas, dans sa méthodologie de services, la réutilisation, à travers la composition, d'un service composite ne peut pas être considérée comme étant réflexive.

La facette Réflexivité est définie comme suit :

Réflexivité : Booléen

3.3. La facette Spécification du service

Selon la définition de [Arni-Bloch *et al.* 2009], on distingue deux types de spécification de services : (i) les services décrits comme une *boîte noire,* avec une partie interface unique disponible à d'autres services lui permettant d'être composé ; et (ii) les services décrits

comme une *boîte blanche,* lesquels incluent dans leur spécification les informations sur les règles, les processus et la structure du service, informations qui sont donc partagés avec d'autres services.

De manière similaire, [Quartel *et al.* 2004] distinguent deux perspectives dans la spécification d'un service : externe et interne. D'une part, la perspective externe tient à présenter le service comme une *boîte noire* : seules les entrées/sorties (les messages) sont gérées, sans que les spécifications internes soient fournies. Ceci permet un haut niveau de modularité et d'interopérabilité. L'avantage de ce modèle de message est qu'il permet de s'abstraire de l'architecture, du langage ou encore de la plateforme qui prendra en charge le service. Il suffit juste que le message respecte une structure donnée pour qu'il puisse être utilisé. Cela permet un faible couplage à travers un échange asynchrone de messages.

D'autre part, la perspective interne tient à présenter le service comme une *boîte blanche*. Le service dans ce cas n'est plus une entité monolithique, mais il est composé d'un groupe d'éléments qui interagissent afin de réaliser la fonction du service.

La facette Spécification illustre donc cet aspect des services, pouvant être définie comme suit :
Spécification du service : ENUM {Boite noire, Boite blanche}

4. La vue but

Les buts recherchés par le approches orientées services sont diversifiés. La vue but considère cette question à travers une facette unique, appelée Nature. Cette facette répond à la question : « Pourquoi et comment est utilisée une approche orientée services ? ». Pour cela, nous considérons l'utilisation des approches à base de services selon quatre perspectives : la modélisation du système d'information, la découverte de service, la composition ou l'intégration de services et le *développement (implémentation) de services*.

- **La modélisation du système d'information :**

De nombreux chercheurs s'intéressent aux différentes approches utilisées pour la *modélisation des systèmes d'information orientée services* [Papazoglou 2006] [Kaabi 2007] [Penserini *et al.* 2007] [Mirbel *et al.* 2009] [Arni-Bloch *et al.* 2009]. Ces approches méthodologiques permettent de passer graduellement des processus métiers aux services techniques à l'aide d'un ensemble de principes et de directives.

[Papazouglou 2006] propose des modèles et des techniques relatives aux fondements des services Web, afin de simplifier leur découverte et leur composition aussi bien dans les environnements transactionnels que dans les non-transactionnels.

[Penserini *et al.* 2007] et [Kaabi 2007] s'intéressent aux approches guidées par les buts pour la modélisation des systèmes d'information. Les buts des utilisateurs sont identifiés et affinés jusqu'à l'obtention des services capables de réaliser ces buts.

[Arni-Bloch *et al.* 2009] examinent l'intégration d'un nouveau Service de Système d'Information (ISS) dans un ancien Système d'Information (SI). Les auteurs décrivent de manière conceptuelle le service offert par un système d'information comme un composant autonome, interopérable et cohérent, lequel peut être classé selon trois aspects : aspects techniques, aspects organisationnels et aspects informationnels. Le modèle de service informationnel proposé par [Arni-Bloch *et al.* 2009] utilise trois espaces distincts pour la définition d'un tel service : (i) un espace statique (ensemble des classes de service SI), qui représente la structure de l'information ; (ii) un espace dynamique (ensemble d'actions), qui capture les manipulations que l'information peut subir, l'information pouvant être les fonctionnalités du système, les activités, l'état et le comportement ; et (iii) un espace de règles (ensemble des règles de service SI), qui représente les contraintes que l'espace statique doit satisfaire.

- **La découverte de services :**

La découverte de service Web désigne le processus par lequel les utilisateurs d'un service recherchent manuellement ou semi-automatiquement le service correspondant à leurs besoins. Les services auront été préalablement modélisés et publiés dans un annuaire de services. La découverte de service se focalise ainsi sur la phase d'appariement sémantique entre l'offre et la demande de service. Il d'agit d'un processus crucial dans le domaine des approches orientées services.

La requête du client pour la satisfaction des ses besoins peut prendre deux formes, selon le choix du fournisseur : (i) soit le fournisseur met à la disposition du client un mécanisme d'interrogation ; (ii) soit le client exprime sa requête dans le langage de description de service et l'envoie directement au fournisseur, qui fait correspondre la demande à l'offre via un algorithme d'appariement. Dans le premier cas, la requête du client correspond à une interrogation directe de la part du client à l'annuaire utilisé par le fournisseur. Dans le second cas, la requête du client est exprimée de manière explicite. Le fournisseur compare toutes les

entrées de la requête, pour fournir comme résultat les entrées qui sont proches de cette requête.

Dans cette perspective, nous voulons souligner si une approche orientée services explicite, dans sa méthodologie, la découverte de services en prenant en compte les requêtes des clients et leurs structures, ou non. En somme, nous voulons savoir si l'approche s'intéresse l'étape de la découverte en analysant plus en profondeur les buts des utilisateurs pour la recherche des services ainsi que les algorithmes d'appariement.

- **La composition (ou l'intégration) de services :**

La composition des services est un processus à travers lequel un service est construit à partir d'autres services préexistants. Le nouveau service obtenu est appelé service composite. Un service composite est donc un ensemble organisé d'autres services. Dans ce cas, les services composants sont réutilisés par le service composite.

[Arni-Bloch *et al.* 2009] définit le concept d'intégration de services dans un système d'information (SI) comme le fait d'étendre le SI avec de nouvelles capacités, des nouvelles fonctionnalités : dans ce cas, le domaine du SI est alors étendu. Pour ces auteurs, l'intégration des nouveaux services est nécessaire lorsqu'une approche orientée services est développé alors que d'anciens services existent. Dans ce cas, le processus d'intégration demande à prendre en compte le chevauchement qu'un service peut avoir avec d'autres services existant.

- **L'implémentation (exécution) de services :**

Les approches orientées services sont basées sur la composition de services, qui est définie comme la capacité d'un service à être réutilisé par d'autres services.

L'exécution des services requiert d'orchestrer l'enchaînement des services composites selon un canevas prédéfini et de les exécuter à travers des "scripts d'orchestration". Ces scripts représentent soit des processus métier, soit des workflows inter/intra entreprise. L'orchestration décrit la manière selon laquelle les services peuvent interagir au moyen de messages, d'une logique métier et selon un ordre d'exécution des interactions.

La facette *Nature* de la vue But est définie comme suit :
Nature : ENSEMBLE (ENUM : {Modéliser, Découvrir, Composer (ou Intégrer), Exécuter})

5. La Vue Méthode

La vue Méthode est proposée dans le but d'analyser les différentes méthodes utilisées pour la conception et la mise en œuvre des différentes approches orientées servies. Pour cela, nous avons identifié cinq facettes : *Méthode de Conception*, *Méthode de Découverte de Services*, *Méthode d'Assemblage*, *Langage de spécification et Processus de la Démarche*. Chacune de ces facettes est détaillée dans les sections suivantes.

5.1. La facette Méthode de Conception

De manière générale, les méthodes de conception peuvent être classées en deux catégories : les approches *top-down* et les approches *bottom-up*. Les méthodes de conception utilisées par les approches orientées services ne sont pas une exception. Elles peuvent aussi être classées selon ces deux catégories.

Certaines méthodologies de développement des applications orientées services proposent un processus par étapes pour vérifier que ces applications correspondent bien aux besoins et aux attentes des utilisateurs. Ainsi, des méthodologies comme celles présentées par [Penserini 2006a] et [Traverso 2004] proposent des approches qui utilisent des modèles de buts pour comprendre les besoins des usagers et pour identifier les services qui permettent leur satisfaction. Ces méthodes de construction sont de nature *top-down*. Elles vont de la spécification au développement. Elles considèrent que le processus métier est développé à partir de rien (*from scratch*) sans réutiliser les spécifications des services déjà existantes, ni des processus métier spécifiés auparavant.

A l'inverse, il est possible d'adopter une démarche *bottom-up* permettant la réingénierie des applications existantes en services. Il s'agit alors d'une méthode de construction *bottom-up*. Elle permet d'isoler les fonctionnalités existantes dans un système '*legacy'* et de les présenter sous la forme d'un ou plusieurs services. [Penserini et al 2006b] considère, par exemple, qu'il est possible de dériver les modèles *i*,* représentant les buts et les acteurs de l'organisation, à partir d'un ensemble de plans et d'activités.

La facette Méthode de Conception est définie comme suit :
Méthode de Conception : ENUM {Top-down, Bottom-up}

5.2. La facette Méthode de Découverte de Services

Dans la facette *Méthode de Découverte de Services*, nous nous intéressons à toutes les techniques qui nous permettent de découvrir un service. Nous avons remarqué, dans la facette *Nature* de la vue *But* (cf. Section 4), qu'il existe deux techniques de découverte de service :

- Le fournisseur met à la disposition du client un mécanisme d'interrogation. La requête du client correspond alors à une interrogation de l'annuaire du fournisseur. Nous nommons cette technique : *méthode de découverte de base*. Dans cette technique, c'est le fournisseur qui met à la disposition du client un mécanisme d'interrogation. Il s'agit de la technique la plus simple. Elle n'est cependant pas précise.
- Le client exprime sa requête dans le langage de description de service et envoie cette requête à un fournisseur qui fait correspondre la demande à l'offre via un algorithme d'appariement. Nous nommons cette technique *méthode d'interrogation par un algorithme d'appariement*. L'algorithme d'appariement compare toutes les entrées de la requête, pour fournir comme résultat les entrées qui sont proches de la requête. La découverte de services par un processus d'appariement est à ce jour un problème crucial dans le domaine des approches orientées services.

Nous proposons à la section 7 différentes approches orientées services *qui traitent de la méthode de découverte* à travers d'algorithmes d'appariement. Parmi ceux-là, [Brogi *et al.* 2008] présentent un algorithme pour la découverte de services Web prenant en compte la composition de ces services. L'algorithme appelé SAM (*Service Aggregation Matchmaking*) est utilisé pour faire correspondre les requêtes avec les annuaires des services, en faisant usage des ontologies OWL-S. SAM dispose d'un appariement flexible et prend en compte la composition de services de manière plus importante.

D'autres méthodes complètent cette facette, comme celle de [Mirbel *et al.* 2009], laquelle offre aux utilisateurs finaux la possibilité de décrire leurs démarches de recherche d'un service Web afin d'opérationnaliser un processus métier. Dans cette approche, les démarches sont décrites *sous forme d'intentions et de stratégies*, desquelles sont dérivées les requêtes permettant de rechercher les services Web. Contrairement à [Brogi *et al.* 2008], [Mirbel *et al.* 2009] ne prennent pas en compte la composition de services. Cependant, l'algorithme proposé

par [Mirbel *et al.* 2009] se concentre sur les utilisateurs et leurs buts et stocke dans un référentiel les requêtes des utilisateurs pour les *réutiliser*.

La facette Méthode de Découverte de services est ainsi définie comme suit :
Méthode de découvertes : ENSEMBLE (ENUM : {Découverte de base, Algorithme d'appariement, dérivation à partir des besoins utilisateur, réutilisation de la requête}

5.3. La facette Méthode d'Assemblage

La facette méthode d'assemblage considère à la fois la méthode de composition et la méthode d'intégration des services. Les méthodes de composition de services peuvent être classées en composition *automatique* et composition *semi-automatique*.

La composition *automatique* de services requiert qu'elle soit générée « à la volée », sur la base d'une requête utilisateur. L'utilisateur spécifie les fonctionnalités requises d'un service et les services candidats sont choisis en fonction de la requête.

L'approche Self-Serv [Benatallah *et al.* 2002] supporte la composition automatique des services à travers la notion de conteneur de services. Un conteneur est un service qui représente, en fait, l'agrégation d'un ensemble de services substituables. Au moment de son invocation, le conteneur détermine quel service parmi ceux répertoriés sera exécuté. Le choix est effectué suivant les paramètres de la demande, les caractéristiques des services disponibles, l'historique des exécutions passées et le statut des exécutions en cours.

La composition *semi-automatique* est définie par le processus suivant : d'abord les services à composer sont générés automatiquement « à la volée » sur la base d'une requête utilisateur, puis l'utilisateur spécifie les fonctionnalités requises d'un service et les services candidats sont choisis en fonction de la requête. D'un autre coté, le processus de mise en correspondance peut générer une série d'alternatives de compositions de services. L'utilisateur doit alors choisir les alternatives qui correspondent au mieux à ses attentes.

Plusieurs travaux, dont [Schaffner *et al.* 2006] [Sirin *et al.* 2004] [Kim *et al.* 2004] et [Meyers *et al.* 2002], proposent des outils interactifs dédiés à la composition semi-automatique des services. Le comportement de ces outils se résume bien souvent à suggérer, à chaque étape, un ensemble de services identifiés automatiquement suivant la requête de l'utilisateur. L'utilisateur doit choisir, par conséquent, les services qui lui conviennent. L'outil permet aussi d'intégrer les services choisis dans une composition déjà existante.

La composition de services dite *fixe* requiert que les services constituants soient combinés d'une manière fixe (prédéfinie).

Enfin, [Arni-Bloch *et al.* 2009] définit le concept d'intégration de services dans un système d'information comme étant l'extension d'un système d'information (SI) par l'intégration des nouveaux services garantissant des nouvelles capacités, nouvelles fonctionnalités. Le domaine du SI est alors étendu. Pour [Ralyte *et al.* 2009], l'intégration de services est nécessaire lorsqu'une approche orientée services est développée alors que d'anciens services existent. Dans ce cas, le processus d'intégration demande à prendre en compte le chevauchement qu'un service peut avoir avec d'autres services existants dans le SI. Pour l'auteur, ce cas de figure n'est pas traité par la composition de services qui est utile surtout lorsqu'il n'y pas d'anciens services et que le système d'information est construit à partir des nouveaux services.

La facette Méthode d'Assemblage est définie comme suit :

Méthode d'assemblage : ENSEMBLE (ENUM : {composition automatique, composition semi-automatique, composition fixe, intégration}

5.4. La facette Langage de spécification

La facette langage de spécification considère les différents langages utilisés par les approches orientées services pour la spécification des services. Les langages de spécification permettent d'exprimer les spécifications liées à un système. Différents types de notation existent. Elles vont des "plus naturelles" aux plus formelles.

- **Spécification informelle.**

Les spécifications informelles sont des descriptions en langage naturel. Bien que pouvant être standardisées, ces notations sont bien souvent propres à une entreprise, voire même à un projet ou à une équipe de développeurs. L'inconvénient majeur vient de l'ambiguïté du langage. Cette ambigüité entraine des risques d'incohérence, de non complétude, de difficulté d'organisation, ou encore de redondance dans les informations représentées. Pour pallier l'ambiguïté, certains adoptent des présentations formatées, telles que les dictionnaires de données ou les glossaires, dans lesquels est défini le vocabulaire propre à un domaine d'application précis. Ceci reste cependant insuffisant et les risques cités précédemment perdurent.

- **Spécification semi-formelle.**

Les spécifications semi-formelles sont relativement simples mais structurées et permettent ainsi aux différents participants d'un projet de communiquer facilement et efficacement.

Les notations classiques de cette catégorie sont :

- les diagrammes d'états-transitions dont l'objectif est, comme pour les tables états-transitions, de représenter les différents états du système et les évènements qui déclenchent les changements d'états ;
- les diagrammes de flots de données qui intègrent les flots de données et les processus de transformation. Ils montrent comment chaque processus transforme les données qu'il reçoit en flots de sortie ;
- l'ensemble des notations utilisées dans UML (diagramme de cas d'utilisation, de séquence, de collaboration, d'états-transitions, etc.).

- **Spécification formelle.**

Les spécifications formelles reposent essentiellement sur des notations mathématiques. Ces notations permettent, lors de la spécification d'un système, d'éliminer les ambiguïtés des notations informelles ou semi-formelles, ainsi que les mauvaises interprétations qui en découlent.

Les langages formels possèdent trois composantes :

- une syntaxe qui précise la notation utilisée pour procéder à la spécification ;
- une sémantique qui donne une définition non ambiguë de cette syntaxe ;
- un ensemble de relations qui définissent les règles donnant les propriétés des objets mathématiques satisfaisant la spécification.

Certaines spécifications non formelles peuvent se voir adjoindre des langages formels pour l'expression des contraintes. On peut citer, par exemple, l'usage du langage OCL [BKPPT 2000] dans le cas d'UML.

La facette *langage de spécification* est donc définie comme suit :
Langage de spécification : ENUM {Informelle, Semi-formelle, Formelle}

5.5. La facette Processus de la Démarche

Un processus est « un ensemble d'activités inter-reliées et menées dans le but de définir un produit » [Franckson *et al.* 1991]. Il est exprimé dans les termes d'un modèle de processus. Un modèle de processus est une démarche méthodologique décrivant la dynamique de la méthode. Le produit est le résultat d'application de cette démarche.

D'après [Dowson 1988], les modèles de processus peuvent être classés en trois groupes : orientés activités, orientés produit et orientés décision. A ces trois catégories, nous pouvons ajouter les modèles contextuels. Ainsi, les processus utilisés au sein des approches orientées services peuvent être classés selon quatre modèles distincts :

Les modèles *orientés-activité.* Ils se focalisent sur les activités exécutées pour élaborer un produit et sur leur ordonnancement. Par exemple, [Papazoglou 2006a] propose un modèle de processus orienté-activités pour la modélisation conceptuelle et le développement des applications à base de services. Ce processus est structuré en un ensemble de directives et principes méthodologiques pour spécifier, construire et implémenter des applications supportant une composition de services.

Les modèles *orientés-produit.* Ils couplent l'état du produit à l'activité qui génère cet état. Ils visualisent ainsi le processus comme un diagramme de transition d'états. Les travaux de [Quartel *et al.* 2004] s'inscrivent dans ce groupe. Ces auteurs proposent un processus méthodologique structuré en étapes pour la modélisation des services. A chaque étape, l'état du service change.

Les modèles *orientés-décision*. Ils perçoivent les transformations successives du produit causées par le processus comme les conséquences de prises de décisions.

Les modèles *contextuelles*. C'est une nouvelle catégorie de modèles processus. Les modèles contextuels définissent les processus à travers la combinaison des situations observables à un ensemble d'intentions spécifiques [Rosemann *et al.* 2007] [Saidani *et al.* 2007]. Le travail à faire est décrit dans le processus comme étant dépendant à la fois de la situation et de l'intention. En d'autres termes, il dépend du contexte dans lequel on se trouve au moment de le réaliser.

La facette Processus de la Démarche s'appuie sur cette typologie établie des processus et est défini comme suit :

Processus de la démarche : ENUM {Activité, Produit, Décision, Contexte}

6. La vue Implémentation

La vue Implémentation traite du processus de construction des approches orientées services. Quatre facettes permettent de couvrir cette vue : (i) Degré de couverture des phases du cycle de développement ; (ii) Outil de conception ; (iii) Généricité du méta-modèle à plusieurs

plateformes ; et (iv) Passage spécification métier – spécification technique. Ces facettes sont décrites dans les sections ci-dessous.

6.1. Degré de couverture des phases du cycle de développement

Le processus de développement spécifie la manière dont le développement est organisé en phases, indépendamment du type de support technologique utilisé dans le développement [Derniame *et al.* 1999], et ce processus diffère selon les méthodologies proposées. Chacun des différents processus définit un certain cycle de développement. De manière générale, le cycle de développement comporte quatre phases qui partent des besoins jusqu'à l'implémentation : l'élicitation des besoins, l'analyse des besoins, la conception et l'implémentation. Au-delà de ces quatre phases, nous pouvons également citer les phases de vérification, de teste et de déploiement. Malgré leur importance pour la mise en œuvre des applications orientées services, nous ne nous intéressons pas à ces dernières phases, préférant concentrer notre analyse sur les quatre phases précédentes, pour lesquels les approches orientées services offrent des solutions plus riches.

Par exemple, [Penserini *et al.* 2007] adoptent la méthodologie de développement orientée agent Tropos [Bresciani *et al.* 2004] pour la mise en œuvre de son processus de développement composé d'agents logiciel. Tropos suit une approche incrémentale par raffinement itératif. Il s'agit d'une méthodologie générique pouvant être appliquée à des contextes différents. Elle couvre une grande partie du cycle de développement, et se base sur une notation formelle (*Formal Tropos*) qui lui permet de vérifier et valider ses modèles. Cependant, ces outils fournis par Tropos ne sont réellement applicables que durant les phases d'élicitation et d'analyse des besoins.

Pour chacune des quatre phases du cycle de développement que nous analysons ici, nous associons des degrés qui partent du signe « ++ », signifiant que la phase est très importante dans le processus de développement, au signe « -- », signifiant que la phase est beaucoup moins importante. De manière similaire, lorsque la mise en place d'une phase par le concepteur d'une approche n'est pas aussi importante (ou inversement moins importante), nous la qualifions par le signe «+ » ou « - », ce qui signifie que la phase est seulement importante (ou inversement peu importante). Il peut arriver qu'une approche orientée service n'ait pas une des phases implémentée, dans ce cas, cette phase n'est pas mentionnée.

La facette Degré de recouvrement des phases se définit comme suit :

Degré de recouvrement des phases : ENSEMBLE (ENUM : {Elicitation des besoins (--/-/+/++), Analyse des besoins (--/-/+/++), Conception (--/-/+/++), Implémentation (--/-/+/++)}

6.2. Outil de conception

La facette Outil de conception se concentre sur les outils permettant la production des spécifications selon des modèles et des diagrammes définis par les notations. Ces outils permettent d'avoir des spécifications propres, lisibles, facilement modifiables. Bien souvent, ils permettent aussi la vérification de leurs syntaxes, leurs cohérences et de leur complétude. Les outils peuvent être regroupés dans des environnements.

Au sein de l'approche [Pensirini *et al.* 2007], plusieurs outils aident à l'application de la méthodologie Tropos. Par exemple, T-Tool [Kazhamiakin *et al.* 2003] est un outil utilisable durant les phases préliminaires de collecte des besoins. Il permet l'analyse et la validation des spécifications en utilisant les notations formelles introduites par Formal Tropos. T-Tool applique pour cela la technique du *model checking*. T-Tool permet par ailleurs d'animer certains modèles afin d'être validés par les développeurs. Un autre exemple, l'outil GR-Tool [Sebastiani *et al.* 2004] est aussi fourni par la méthodologie Tropos. C'est un outil de conception graphique permettant de spécifier les diagrammes de buts et d'exécuter les algorithmes qui leur correspondent.

La facette Outil de conception se définit alors come suit :

Outils de conception : ENSEMBLE (outils)

6.3. Généricité du méta-modèle à plusieurs plateformes

Les méthodologies liées aux approches orientées services se limitent bien souvent à des aspects très spécifiques des systèmes en cours de développement. Dans leur majorité, elles ne couvrent pas la totalité du processus de développement. [Cossentino *et al.* 2007] explique que les différentes méthodologies ont différents avantages et ne peuvent pas être appliquées toutes sur différentes plateformes, car elles se concentrent sur des problèmes bien trop spécifiques. Par conséquent, on ne peut pas avoir une méthodologie unique pour une utilisation générale, avec un certain niveau de personnalisation. Selon [Cossentino *et al.* 2007], les concepteurs, pour résoudre leurs problèmes spécifiques dans un contexte d'une application aussi spécifique, préfèrent définir leur propre méta-modèle, particulièrement conçu pour leurs besoins. Ainsi, pendant la phase d'implémentation, la plateforme cible utilisée est souvent

spécifiée directement sur le méta-modèle décrit dans la phase de conception de chaque méthodologie.

Ainsi, la méthodologie PASSI [Cossentino *et al.* 2005] propose, par exemple, un méta-modèle dont les concepts permettent à l'outil PTK (PASSI Toolkit) [Chella *et al.* 2004] de générer de code uniquement pour la plateforme JADE, alors que la méthodologie Tropos [Pensirini *et al.* 2007] permet aussi la génération de code pour la plateforme JACK.

La facette Généricité du méta-modèle peut être définie comme suit :

Généricité du méta-modèle à plusieurs plateformes : Booléen

6.4. Passage spécification métier – spécification technique

Selon [Azaiez 2007], un large fossé sépare encore les phases d'analyse et de conception des phases d'implémentation. Le problème vient du fait que les plates-formes ne couvrent pas tous les concepts pris en compte au niveau des modèles produits par la méthodologie. Cela entraine une perte de sémantique, d'une part, entre les phases d'analyse des besoins et les phases de conception et, d'autre part, entre les phases d'implémentation. C'est donc au développeur que revient la charge de créer les entités correspondantes afin de palier aujourd'hui à l'absence de certains concepts dans le méta-modèle. Le degré de la perte de sémantique entre les niveaux n'est pas le même pour toutes les méthodologies. Certaines méthodologies perdent moins, c'est le cas de la méthodologie de [Pensirini *et al.* 2007], alors que d'autres en perte plus.

La facette Passage Métier – Technique est définie comme suit :

Passage Métier – Technique : ENUM {Inexistant, Perte de sémantique ++, Perte partielle +}

La figure 2.2 résume les vues et facettes du cadre de référence présenté ci-dessus :

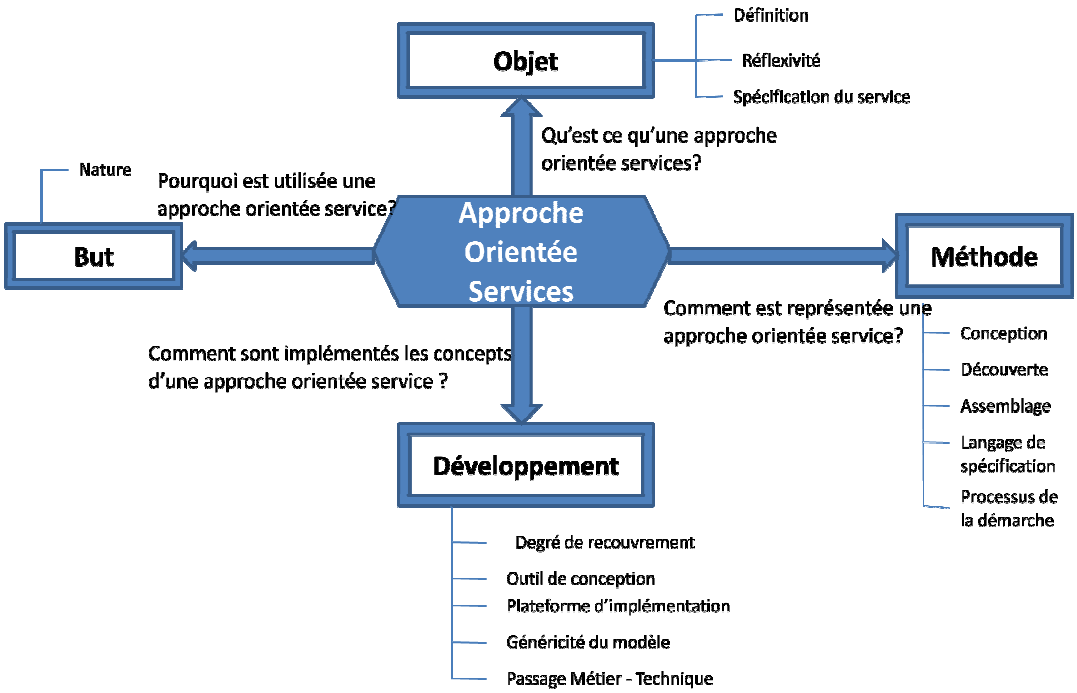


Figure 2.2. Les Vues et les Facettes du cadre de référence

7. Analyse de sept approches orientées services au moyen du cadre de référence

A partir du cadre de référence présenté ci-dessus, nous proposons un examen de sept approches orientées services. Nous les choisissons dans l' ensemble des approches les plus récentes avec l'intention d'offrir une étude complète à travers les différentes vues et leurs facettes. L'objectif est d'obtenir une ″vue d'ensemble″ du domaine de recherche associé aux approches orientées services et d'aider à la compréhension des réalisations. Cette partie décrit ainsi brièvement ces approches et les situe par rapport au cadre de référence décrit ci-dessus.

7.1. Annotation automatique de services web basés sur les définitions de workflow

[Belhajjame *et al.* 2006] explorent l'utilisation des annotations sémantiques pour la composition de services exprimée à travers les workflow. Ces auteurs considèrent les workflows comme un réseau des services connectés par une série d'opérations. Ils proposent alors l'usage de cette source additionnelle d'information, qu'ils nomment référentiel de workflows dirigés par des données, pour l'annotation des services. Les auteurs montrent

comment l'information peut être déduite à partir de la sémantique associée aux paramètres des opérations basé sur leurs connections avec d'autres paramètres d'opération dans le workflow.

Ces auteurs proposent alors un algorithme pour l'annotation automatique des services dans un workflow. Cet algorithme se base les workflows dirigé par les données, lesquels sont vu comme un ensemble d'opérations connectées à travers des liens de données. Le workflow dirigé par les données est ainsi modélisé comme un triple *swf=⟨nameWf, OP, DL⟩*, où *nameWf* est un identifiant unique pour le workflow, *OP* est un ensemble d'opérations à partir duquel le workflow est composé, et *DL* est un ensemble de liens de données connectant les opérations dans *OP*.

Une *opération*, à son tour, est définie comme un quadruple *op=⟨nameOp, loc, in, out⟩*, op ∈ OP, où *nameOP* est un identifiant unique pour l'opération, *loc* est l'URL du service Web qui l'implémente et *in* et *out* sont des ensembles représentant respectivement les paramètres d'entrée et de sortie de l'opération. Chaque *paramètre* spécifie le type de données d'une entrée ou d'une sortie et est un pair *⟨nameP, type⟩*, où *nameP* est l'identifiant du paramètre (unique dans une opération) et *type* est le type de paramètre de données. Pour les services Web, les paramètres sont fréquemment typés en utilisant le système typage proposé par les schémas XML, qui supportent les types simples (tel que xs : String et xs : int) et les types complexes structurés à partir des autres types.

L'algorithme proposé par [Belhajjame et al. 2006] analyse les *liens de données* existant entre les opérations afin de déduire des nouvelles informations, utilisées, par la suite, pour annoter les services impliqués. Un lien de données décrit un flux de données entre la sortie d'une opération et l'entrée d'une autre. Soit *IN* l'ensemble des paramètres d'entrée de toutes les opérations présentes dans le workflow *swf*, c'est-à-dire, $IN \equiv \{i \mid i \in in \wedge \langle -, -, in, - \rangle \in OP\}$, et soit *OUT* l'ensemble des paramètres de sortie présents dans *swf*, c'est-à-dire $OUT \equiv \{o \mid o \in out \wedge \langle -, -, -, out \rangle \in OP\}$, l'ensemble des liens de données connectant les opérations dans swf doit alors satisfaire : $DL \subseteq (OP \times OUT) \times (OP \times IN)$.

7.2. Aide à l'évolution dynamique des protocoles dans les architectures orientées service

[Ryu *et al.* 2008] mettent en place un framework qui aide la gestion de l'évolution du protocole métier. Un protocole métier, dans le cadre d'un service, spécifie les séquences de message qu'un service et ses clients échangent pour réaliser un but métier précis [Alonso et

al. 2004], par exemple, réserver un billet d'avion. Le framework proposé par [Ryu et al. 2008] fournit aux gestionnaires des services impliqués dans un changement de protocole métier une base de données comportant une analyse de l'impact de ce changement. Celle-ci détermine automatiquement quelles instances en cours d'exécution peuvent être migrées vers une nouvelle version du protocole métier.

La contribution de ces auteurs consiste à la mise en place d'une méthode pour classifier automatiquement les conversations basées sur l'impact de l'évolution du protocole sur elles. Dans cette contribution, deux propriétés sont étudiées. Ces propriétés sont appelés *compatibilité antérieure* et *postérieure* du protocole. Ces propriétés sont utilisées pour déterminer les conversations qui peuvent être migrées vers un nouveau protocole lorsque l'ancien protocole a été changé. La migration d'une conversation à un protocole P signifie que, dans le futur, une conversation aura à obéir aux règles et aux contraintes définies par le protocole P. En fonction de ces propriétés, [Ryu et al. 2008] définissent des opérateurs pour analyser l'impact d'un changement de protocole dans les conversations en cours, et pour les classifier dans des conversations « migrables » et « non migrables ».

De manière formelle, [Ryu *et al.* 2008] définissent un protocole métier comme un tuple $P = (S, S_0, F, M, R)$ défini comme suit : S est un ensemble fini d'état, $S_0 \in S$ est un état initial, F est un ensemble d'état final, M est un ensemble fini de messages (un ensemble de noms d'opérations), et $R \subseteq S^2 \times M$ est la relation de transition entre les états. Chaque transition (s, s', m) identifie un état source s, un état cible s' et un message m, qui est consommé durant cette transition.

Les auteurs représentent également les stratégies possibles qui peuvent être appliquées pour migrer une instance s'exécutant sous un ancien protocole. Trois stratégies ont été identifiées par [Ryu *et al.* 2008], le choix entre ces stratégies étant déterminé par les besoins des gestionnaires des services qui peuvent imposer sur le processus de migration une certaine stratégie plutôt qu'une autre :

- *Continue* : les instances actives sont autorisées à continuer à s'exécuter selon l'ancien protocole. Les gestionnaires de service appliquent cette stratégie aux instances en cours quand il est acceptable de les laisser compléter leur exécution selon l'ancien protocole ;

- *Migration vers le nouveau protocole* : les instances actives sont migrées vers la nouvelle version du protocole. Chaque fois qu'il y a une migration, l'état dans le nouveau protocole à partir duquel la conversation reprendra devra être défini ;
- *Migration vers le protocole ad hoc* : les gestionnaires de service peuvent définir des protocoles *ad hoc* pour les instances qui ne peuvent pas être migrées vers le nouveau protocole, mais qui ne peuvent pas continuer à exécuter selon l'ancien protocole. Les protocoles *ad hoc* sont ainsi définis pour gérer ces conversations *actives pour lesquelles les autres* stratégies ne sont pas applicables.

Enfin, [Ryu *et al.* 2008] fournissent des outils de gestion pour la modification des protocoles, pour l'analyse de l'impact des changements, laquelle se base sur les modèles des protocoles, pour l'analyse de migration basée sur le *datamining,* et pour assister les utilisateurs dans la détermination des stratégies de migration.

7.3. Un environnement pour les compensations avancées dans les transactions de service Web

Les auteurs [Schäfer *et al.* 2008] décrivent un environnement pour les opérations de compensations avancées en adoptant la reprise postérieure des transactions de services Web. La reprise postérieure change l'état et la structure d'une transaction après que l'échec d'un service soit survenu. L'objectif est ainsi d'éviter un retour en arrière et de permettre à la transaction de finir avec succès. Ces auteurs proposent l'introduction d'un nouveau composant appelé *service abstrait,* lequel joue un rôle de médiateur pour les compensations et permet ainsi de masquer la logique derrière les compensations introduites. De plus, il spécifie et gère les remplacements potentiels d'un service Web dans une transaction. Les compensations sont exécutées selon des règles prédéfinies et sont soumises aux contrats.

Une telle solution a des multiples avantages. D'abord, les stratégies de compensation peuvent être définies à la fois sur les fournisseurs de service et sur le côté client. Elles utilisent la connaissance local disponible (par exemple, un fournisseur de service connaît le mieux comment son service peut être remplacé en cas d'échec) et les préférences, ce qui contribue à augmenter la flexibilité et l'efficacité des stratégies. Par ailleurs, l'environnement proposé par [Schäfer et al. 2008] peut gérer à la fois les compensations déclenchées de manière externe et interne. Le client de service est informé sur les opérations de compensation complexe, ce qui permet de déclencher des compensations supplémentaires. Les compensations peuvent ainsi

consister à plusieurs opérations sur différents niveaux et l'efficacité est obtenue à travers des protocoles de communication bien définis.

La figure 2.3 montre un extrait de l'environnement proposé par [Schäfer et al. 2008] avec les composants principaux.

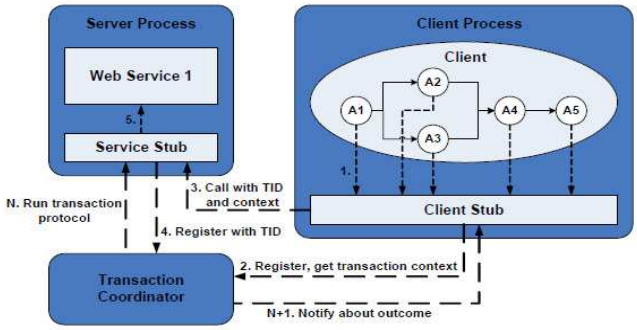


Figure 2.3. Environnement transactionnel pour les Web services

A la figure 2.3, le client exécute les activités A1 et A2 qui sont embarquées dans un contexte transactionnel. Le contexte transactionnel et la conversation sont maintenus par un coordinateur de transaction. Les *stubs* client et serveur sont responsables pour obtenir et pour enregistrer les activités et les appels des services dans le bon contexte. [Schäfer *et al.* 2008] étendent l'architecture et l'infrastructure basées sur les spécifications de [Arjuna Technologies *et al.* 2005], afin qu'ils puissent gérer de manière interne et externe les compensations déclenchées.

La figure 2.4 montre l'extension à la transaction de l'environnement service Web, nommé le service abstrait et les composants adaptateur. Cette extension ne change pas la manière dont le client, le coordinateur de transaction et les fournisseurs de services opèrent. Au lieu d'invoquer un service Web normal, un client invoque un service abstrait. Cependant, le service abstrait est considéré comme un composant de gestion pour les reprises postérieures dans le traitement de la compensation. Celle-ci enveloppe plusieurs services concrets qui offrent la même fonctionnalité et qui peuvent ainsi être remplacés par d'autres services. Le service abstrait est, par conséquent, un médiateur entre un client et le service concret qui exécute les opérations acquises.

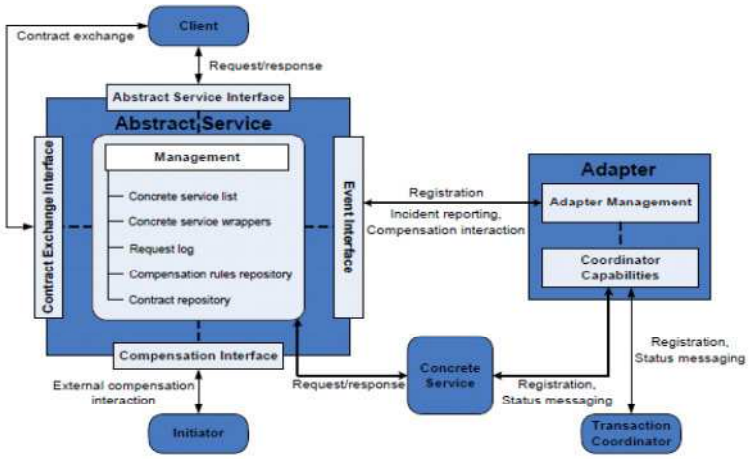


Figure 2.4. Le service abstrait et l'environnement de transaction adaptateur

7.4. Une approche pour la découverte orientée composition de services web

Les auteurs [Brogi *et al.* 2008] présentent un algorithme pour la découverte orientée composition de services Web. L'algorithme, appelé SAM (pour *Service Aggregation Matchmaking*), peut être utilisé pour faire correspondre les requêtes avec les annuaires des services faisant usage des ontologies OWL-S. SAM dispose d'un appariement flexible et prend en compte la composition de service. L'idée est que les requêtes qui ne peuvent pas être satisfaites par un seul et même service peuvent être satisfaites par plusieurs compositions de services. Par exemple, un client qui souhaite planifier ses vacances en réservant à la fois un billet d'avion et un hébergement en hôtel, tout en considérant les différents paramètres tels que le temps, le prix des saisons, les offres spéciales et bien d'autres.

Les différentes caractéristiques de l'algorithme proposé par [Brogi *et al.* 2008] peuvent être résumées ainsi :

Correspondance flexible : SAM exécute une correspondance de granularité fine au niveau des processus atomique ou des sous-services, plutôt qu'au niveau des services entiers. Plutôt que de retourner uniquement des correspondances complètes (lorsqu'un simple service peut entièrement satisfaire la requête du client), SAM retourne une liste de correspondance partielle, lorsqu'aucune correspondance complète n'est possible. Une correspondance partielle est une composition de sous-services capable de fournir seulement une partie des résultats demandés par le client. Par ailleurs, quand aucune correspondance n'est possible, SAM est aussi capable de proposer au client des entrées supplémentaires, en plus des correspondances partielles, qui suffiraient pour obtenir une correspondance complète.

Correspondance orientée composition. SAM permet de fournir une correspondance orientée composition basée sur les descriptions sémantiques des requêtes et des services, en prenant en compte le comportement des services. Lorsqu'un seul service peut satisfaire une requête client, SAM vérifie s'il y a des compositions de service qui peuvent satisfaire la requête, incluant plusieurs exécutions possible de services. Quand SAM trouve une correspondance, il retourne explicitement la séquence d'invocation des processus atomiques que le client doit exécuter dans le but d'obtenir le résultat désiré.

SAM est en mesure d'accéder à un annuaire local de service stockant les descriptions OWL-S. L'objectif est de déterminer si une requête peut être satisfaite par une composition de services annoncés dans un annuaire OWL-S.

7.5. Une approche méthodologique orientée intentions pour les implémentations d'application agent à base de services

[Penserini *et al.* 2007] [Penserini *et al.* 2006a] adoptent la méthodologie de développement orientée agent Tropos [Bresciani *et al.* 2004] pour la mise en œuvre d'un processus de développement composé d'agents logiciel.

La méthodologie Tropos a été originellement présentée dans [Bresciani *et al.* 2004] et [Castro *et al.* 2002]. Plusieurs outils aident à l'application de la méthodologie Tropos : T-Tool [Kazhamiakin *et al.* 2003] et GR-Tool [Sebastiani *et al.* 2004].

La méthodologie Tropos adopte un langage de modélisation basé sur le framework de modélisation i* [Yu 1995]. Ce framework offre les concepts de base, tels qu'*acteur* (*actor*), *but* (*goal*), *plan,* but non fonctionnels *(softgoal), ressource (resource)* et *capability.* De plus, le framework supporte plusieurs relations entre ces concepts, telles que les dépendances stratégiques (*strategic dependencies*) entre des réseaux d'acteurs capables de spécifier les différentes relations de dépendances entre les agents. Ces réseaux indiquent qu'un acteur dépend d'un autre afin d'atteindre un but, d'exécuter un plan, ou de fournir une ressource. Le premier acteur (l'acteur "source") s'appelle le *depender*, tandis que le second acteur est un dit *dependee.* L'objet de leur dépendance est le *dependum,* lequel peut être un but, une ressource ou une tâche.

Le processus de conception Tropos utilise un algorithme non-déterministe [Bresciani *et al.* 2004], lequel débute par l'identification des *acteurs* critiques et de leurs buts dans un domaine. Ensuite, s'ensuit une analyse des buts à partir de la perspective de chaque acteur. En

particulier, chaque but pour un acteur donné peut être *délégué* à un autre acteur existant dans le domaine ou à un nouvel acteur. Alternativement, un but peut être raffiné en plusieurs sous but via une décomposition AND/OR (ET/OU), mais il peut être aussi non décomposable. Ce processus génère un réseau de délégation parmi les acteurs (utilisateurs, autres acteurs externes ou acteurs systèmes). De plus, le processus de raffinement de but génère un modèle de but. Celui-ci consiste à une hiérarchie de buts, dans lequel les combinaisons de feuilles représentent des solutions alternatives au but racine. Le processus de conception est complet quand tous les buts ont été traités.

L'approche orientée agent proposé par [Pensirini *et al.* 2007] est un processus de développement dirigé par les modèles qui guide le concepteur dans la construction d'un modèle initial des utilisateurs et de leurs intentions. Elle adopte une approche transformationnelle, dans le sens où, à chaque étape, les modèles vont être raffinés de manière itérative par l'ajout ou la suppression d'éléments dans les modèles.

Le processus de développement de cette approche est un processus itératif composé de cinq phases dont les quatre premières font partie du processus de conception Tropos : *analyse des besoins initiaux,* l'*analyse des besoins finaux, la conception architecturale, la conception détaillée et l'implémentation.* Chaque phase est caractérisée par des objectifs spécifiques.

7.6. Une approche pour l'évolution des Systèmes d'Information dirigée par les services

[Arni-Bloch *et al.* 2009] et [Arni-Bloch *et al.* 2008] examinent l'intégration d'un nouveau service, appelé Service de Système d'Information (ISS), dans un Système d'Information (SI) existant. Dans leur approche, les auteurs se focalisent sur les nouveaux services avec un haut niveau de modularité, chacun d'eux représentant une unité de travail avec une sémantique précise. Un ISS fournit un espace d'information et les capacités aux acteurs qui ont la responsabilité de les utiliser dans le but d'exécuter leurs activités quotidiennes. Lorsque la granularité d'un service ISS est élevée, le chevauchement (*overlap*) avec d'autres services devient difficile à éviter. Il est donc important de gérer ce chevauchement afin de garantir la qualité du processus et des données du SI, en consolidant les données, les capacités, les règles et les responsabilités et en spécifiant la stratégie de coopération.

[Arni-Bloch *et al.* 2009] décrivent le service offert par un système d'information comme un composant autonome, interopérable et cohérent, lequel peut être classé selon trois aspects :

aspects techniques, aspects organisationnels et aspects informationnels. L'intérêt d'une telle approche est de prendre en compte les changements et les évolutions qu'une organisation subit tout au long de son cycle de vie, qu'il s'agit de l'ajout d'une nouvelle application, ou de la restructuration organisationnelle (les fusions avec un groupe de partenaire), ou encore de l'innovation et de la réingénierie d'un processus métier. Dans leur approche, ces auteurs se basent notamment sur le concept de service informationnel qui représente la clé de voûte de leur proposition pour supporter l'intégration et l'ingénierie des services de systèmes d'information.

Formellement, au niveau informationnel, un service de système d'information (ISS) est un ensemble d'objets $S^{**}= \{o_1,....o_2\}$. Un ISS est alors exprimé, dans le modèle de service informationnel, par les quatre espaces : <*sSs, sDs, sRs, sOs*> où :

- *sDs* est l'espace statique d'nu ISS, c'est-à-dire un ensemble de classe de ISS, qui représente la structure de l'information ;
- *sDs* est un espace dynamique de l'ISS. Il est composé d'un ensemble d'actions, qui capturent les manipulations que l'information peut subir, c'est-à-dire les fonctionnalités du système, les activités, l'état et le comportement. On dit alors que sDs est composé d'un ensemble d'actions (sDs_{action}), ainsi que d'un ensemble d'effets (sDs_{effet}) que les actions ISS sont autorisées à produire ;
- *sRs* est l'espace de règle de l'ISS, c'est-à-dire l'ensemble des règles de l'ISS, qui représentent les contraintes que l'espace statique doit satisfaire ;
- *sOs {organizationalRole}* représente l'ensemble des rôles de l'organisation qui ont des droits et des responsabilités sur le service S.

Un système d'information est la construction d'une collection de services : ε représente l'ensemble de tous les services de IS ($\varepsilon= \{S_1,....., S_2\}$), tandis que ε** représente l'union de tous les objets de tous les services de l'IS ($\varepsilon^{**} = S^{**}_1 \cup .... \cup S^{**}_n$).

7.7. Une approche sémantique pour la recherche de services web guidée par les intentions : approche SATIS

L'approche SATIS proposée par [Mirbel *et al.* 2009] offre aux utilisateurs finaux la possibilité de décrire leurs démarches de recherche d'un service Web afin d'opérationnaliser un processus métier. Dans cette approche, les démarches sont décrites sous forme d'intentions

et de stratégies, desquelles sont dérivées les requêtes permettant de rechercher les services Web.

Au delà d'une approche pour rechercher les services Web, les auteurs proposent d'annoter les services Web, avec les besoins intentionnels des utilisateurs finaux. Egalement, ils proposent des moyens spécifiques de recherche d'information dédiés à l'opérationnalisation de processus de traitement d'images médicales à l'aide des services Web. Pour cela, les patrons de descriptions de services Web et les besoins intentionnels des utilisateurs finaux leur correspondant sont rassemblés dans les fragments autonomes et réutilisables permettant la réutilisation de la connaissance sur l'opérationnalisation d'un processus de traitement d'image d'une démarche à une autre. Ces fragments sont implémentés dans des règles. Les auteurs [Mirbel *et al.* 2009] s'appuient sur un moteur de chaînage arrière pour raisonner sur ces règles et ainsi trouver des descriptions de services Web pertinentes pour opérationnaliser un processus de traitement d'image donnée.

Dans l'approche SATIS, trois acteurs principaux sont déterminés : le concepteur de services Web, l'expert en modélisation de processus et l'expert du domaine métier. Le concepteur de services Web a pour mission d'écrire des patrons de descriptions de services Web et de leur associer des spécifications de besoins intentionnels de haut niveau dans le but de promouvoir les services Web dont il a la charge. L'expert en modélisation de processus a pour mission d'enrichir une mémoire sémantique de la communauté construite avec les fragments de démarches dédiés à l'implémentation de besoins intentionnels de haut niveau. Enfin, l'expert du domaine métier (ou utilisateur final) recherche des descriptions de services Web qui lui permettent d'opérationnaliser un processus métier qui l'intéresse, une chaîne de traitement d'image dans le cas présenté par [Mirbel et al. 2009]. Pour cela, il peut chercher parmi les besoins intentionnels déjà explicités dans la mémoire sémantique de la communauté ou il peut décider de créer par lui-même une nouvelle spécification. Cette étape de création consiste à spécifier des besoins intentionnels de haut niveau à l'aide des buts et des stratégies, et de les raffiner en buts et stratégies intermédiaires jusqu'à spécifier des buts suffisamment précis pour qu'ils soient associés à des patrons de spécification de services Web. Ensuite, l'étape de mise en œuvre de la démarche de recherche de services Web ainsi spécifiée consiste à opérationnaliser le processus métier (la chaîne de traitement d'image en l'occurrence), dont les besoins ont été élicités durant l'étape de création, à l'aide de services Web dont les annotations sont disponibles dans la mémoire collective de la communauté au moment de la recherche.

Durant l'étape de création, le modèle de carte [Rolland 2007] est utilisé pour la capture des besoins intentionnels des utilisateurs finaux. Trois ontologies sont utilisées à la fois pour annoter les besoins intentionnels de haut niveau et pour spécifier les patrons de descriptions de services Web qui leur sont associés [Mirbel *et al.* 2009] : l'ontologie pour les processus intentionnels, l'ontologie du domaine métier et l'ontologie OWL-S. Les annotations RDF, qui représentent les besoins intentionnels, et les requêtes en SPARQL, qui représentent les patrons de descriptions de services Web, sont ensuite rassemblées dans des règles. Celles-ci sont considérées comme des fragments de démarches de recherche de services Web réutilisables et forment la mémoire sémantique de la communauté.

L'étape de mise en œuvre est supportée par un moteur de chaînage arrière qui exploite les règles et les annotations de services Web. L'approche s'appuie sur le moteur sémantique CORESE [Corby *et al.* 2008] à la fois pour le mécanisme de chaînage arrière sur la base des règles de SATIS, et pour l'appariement des requêtes spécifiées dans les règles avec la base de connaissance des annotations OWL-S de services Web (base d'annotation des services de la communauté). Ainsi, un membre de la communauté qui cherche des services Web pour opérationnaliser une chaîne de traitement d'images particulière pourra tirer profit de toutes les règles et de toutes les annotations de services Web présentes dans la mémoire sémantique de la communauté au moment de sa recherche.

8. Résumé de l'évaluation

Les approches discutées précédemment ont été analysées par rapport au cadre de référence que nous avons présenté dans ce chapitre. Cette analyse est illustrée par le Tableau 2.1 ci-dessous.

Vues	Objet			But	Méthode					Implémentation			
Facettes	**Définition**	**Réflexivité**	**Spécification du service**	**Nature**	**Conception**	**Découverte de services**	**Assemblage de services**	**Langage de spécification**	**Processus de la démarche**	**Degré de couverture des phases du cycle de développement**	**Outil de conception**	**Généricité du méta-modèle à plusieurs plateformes**	**Passage Spécifications Métiers aux Spécifications techniques**
[Belhajjame *et al.* 2006]	Technologique	Faux	Boite Noire	Composer, Découvrir, Exécuter	Bottom-up	Découverte de base	Composition automatique	formelle	Produit	Implémentation (++)	Inexistant	Faux	Inexistant (pas de métier)
[Ryu *et al.* 2008]	Technologique	Vrai	Boite Noire	Composer, Exécuter	Bottom-up	Découverte de base	Composition semi-automatique	formelle	Produit	Implémentation (++)	Inexistant	Faux	Inexistant (pas de métier)
[Schäfer *et al.* 2008]	Technologique	Vrai	Boite Noire	Composer, Exécuter	Bottom-up	Découverte de base	Composition semi-automatique	formelle	Produit	Implémentation (++)	Inexistant	Faux	Inexistant (pas de métier)
[Brogi *et al.* 2008]	Technologique	Faux	Boite Noire	Composer, Découvrir, Exécuter	Bottom-up	Algorithme d'appariement	Composition automatique	formelle	Produit	Implémentation (++)	Inexistant	Faux	Inexistant (pas de métier)
[Penserini *et al.* 2009]	Intentionnel	Faux	Boite blanche	Modéliser, Composer, Exécuter	Top-down	Découverte de base	Composition semi-automatique	Semi-formelle	Activité	Elicitation des besoins (++) Analyse des besoins (++) Conception (++) Implémentation(-)	T-Tool GR-Tool (Complé-mentaire)	Vrai	Perte partielle de sémantique
[Arni-Bloch *et al.* 2009]	Métier	Vrai	Boite blanche	Modéliser, Intégrer	Bottom-up	Découverte de base	Intégration	Semi-formelle	Produit	Analyse des besoins (++) Conception (++)	Inexistant	Faux	Inexistant (pas de technique)
[Mirbel *et al.* 2009]	Intentionnel	Faux	Boite blanche	Modéliser, Découvrir, Exécuter	Top-down	Algorithme d'appariement, dérivé à partir des besoins, Requête réutilisable	Inexistant	Semi-formelle	Activité	Elicitation des besoins (++) Analyse des besoins (++) Conception (++) Implémentation (- -)	notation (Carte)	Faux	Perte de sémantique

Tableau 2.1 : Résumé de l'évaluation des sept approches.

La première vue que nous pouvons apercevoir dans le Tableau 2.1 est la vue objet. Celle-ci contient trois aspects qui concernent la définition de l'approche, la réflectivité et la spécification du service. Nous notons que quatre approches [Belhajjame *et al.* 2006], [Ryu *et al.* 2008], [Schäfer *et al.* 2008] et [Brogi *et al.* 2008] sont définies avec une perspective technologique. Ces approches sont dépourvues d'une conception métier. À l'inverse, deux autres approches [Penserini *et al.* 2007] [Mirbel *et al.* 2009] sont guidées par les buts, ayant donc une conception métier. Enfin, l'approche de [Arni-Bloch *et al.* 2009] propose uniquement une perspective métier.

Nous constatons également qu'il existe un lien étroit entre la spécification des services et la définition des approches. Les approches technologiques décrivent généralement les services comme une boîte noire, alors que les approches conceptuelles (intentionnelle ou métier) décrivent les services comme une boîte blanche.

La vue but contient une facette unique qui est la nature de l'approche orientée service. Le tableau 2.1 montre que toutes les approches technologiques ne modélisent pas leurs concepts de services. Seules les approches intentionnelles ou métier, telles que [Penserini *et al.* 2007], [Arni-Bloch *et al.* 2009] et [Mirbel *et al.* 2009], proposent la modélisation de leurs services. Il est à noter que toutes les approches implémentent leurs services, à l'exception de l'approche de [Arni-Bloch *et al.* 2009], qui se limite seulement à la conception de son approche de services.

Aussi, la plupart des approches orientées services composent leurs services. Enfin, nous remarquons que [Arni-Bloch *et al.* 2009] n'utilise pas le concept de composition de services, mais plutôt celui d'intégration de services.

La vue méthode comprend cinq facettes. Nous notons dans la facette méthode de conception que plusieurs approches technologiques ont une structure *bottom-up*. Ceci parce qu'ils réutilisent des services traditionnels '*legacy'* et font de la réingénierie d'applications de services existantes. Peu d'approches orientée services [Penserini *et al.* 2007] [Mirbel et al. 2009] ont une structure *top-down*. Ces approches considèrent que le processus métier est développé à partir de zéro, sans la re-spécification des services existants ou des parties des processus métier existants.

Seules quelques approches analysées proposent leurs propres méthodes de découverte de services. La plupart d'entre elles ont une méthode de découverte de base. Inversement,

[Mirbel *et al.* 2009] mettent en place un algorithme d'appariement pour la découverte orientée buts, ainsi que la possibilité de réutiliser les requêtes de recherche de l'utilisateur.

Toutes les approches technologiques sont essentiellement axées sur l'implémentation des services et ignorent les phases de conception. Inversement, les approches intentionnelles et métier ont quasiment les quatre phases du cycle de développement. Toutefois, leurs phases d'implémentation sont considérées comme moins importantes, ou simplement n'existent pas. Il est également intéressant de noter que toutes les approches analysées proposent des méta-modèles spécifiques à une seule plateforme. Seule l'approche [Penserini *et al.* 2007] utilise un méta-modèle qui peut être appliqué sur deux plates-formes (Jade et JADEX). Un méta-modèle générique applicable sur des multiples plateformes pourrait éviter la reconstruction du méta-modèle à chaque fois qu'un nouveau problème spécifique à une plateforme est découvert.

9. Caractérisation de l' approche MeTSI

Cette thèse présente l'approche MeTSI. Afin de l'introduire nous utilisons le cadre de référence qui permet de mettre ses caractéristiques en lumière. Le Tableau 2.2 résume le positionnement de l'approche MeTSI dans le cadre de référence présenté dans ce chapitre.

Vue	Facette	Approche MeTSI
Objet	**Définition**	Intentionnel
	Réflexivité	Vrai
	Spécification du service	Boite blanche
But	**Nature**	Modéliser, composer, exécuter
Méthode	**Conception**	Top-down
	Découverte de services	Découverte de base
	Assemblage de services	automatique
	Langage de spécification	Semi-formel
	Processus de la démarche	Contextuel
Implémentation	**Degré de couverture des phases du cycle de développement**	Elicitation des besoins (++), Analyse des besoins (++), Conception (++), Implémentation (++)
	Outil de conception	Notation (carte)
	Généricité du méta-modèle à plusieurs plateformes	Vrai
	Passage Spécifications Métiers aux Spécifications techniques	Pas de perte de sémantique

Tableau 2.2 : Résumé de l'application du cadre de référence à l'approche MeTSI.

L'approche MeTSI, que nous proposons dans cette thèse, essaie de combler certaines lacunes identifiées dans les approches précédentes.

L'approche MeTSI se place clairement dans la perspective intentionnelle. Nous mettons en place également un algorithme de découverte de services basé sur un algorithme d'appariement pour la recherche et la découverte de services dans un annuaire de services intentionnels.

Par ailleurs, l'approche MeTSI offre une flexibilité supplémentaire par rapport aux approches précédemment citées, et notamment [Penserini *et al.* 2007] qui tient au processus transformationnel en étapes assurant la transformation progressive des buts utilisateur en application à base de services logiciels exécutables.

.

Enfin, contrairement aux approches existantes (basées sur des méta-modèles spécifiques à une seule et parfois deux plateformes) MeTSI introduit un niveau qui permet de configurer la solution en fonction des paramètres de la classe de plateformes choisie. A partir de cette configuration, MeTSI propose des règles de génération de l'application finale pour trois plateformes, WSRP (Web Service remote Portlet), l'OSGi et les Mashups.

CHAPITRE 3: LA METHODE DE TRANSFORMATION DE SERVICES INTENTIONNELS - MeTSI

1. Introduction

Ce chapitre donne un aperçu de l'approche MeTSI proposée dans cette thèse. Nous rappelons d'abord notre point de départ, la vue intentionnelle des services, puis nous introduisons les phases de la démarche MeTSI qui suit les principes de l'ingénierie dirigée par les modèles. Ceci est l'objet de la section 2. Nous rappelons dans une troisième section les fondements de l'ingénierie dirigée par les modèles qui vise à fournir un cadre conceptuel, technologique et méthodologique dans lequel les modèles sont au centre des activités d'ingénierie tout au long du cycle de développement d'un logiciel. En section 4, nous décrivons le modèle intentionnel de services (MIS) qui est le modèle de départ de MeTSI. Enfin, en section 5, nous présentons les plateformes d'exécution des services que nous avons retenues comme cibles dans cette thèse.

2. L'approche MeTSI

2.1. Survol de l'approche

***Le point de départ : les services intensionnels*.**

L'approche orientée services intentionnels qui est le point de départ de ce travail, est une approche à base de services dirigée par les intentions de l'utilisateur. Elle modélise explicitement l'aspect intentionnel de l'application à concevoir. Il a été reconnu récemment que le point de vue intentionnel d'un système d'information facilite la prise de décision par la suite (le choix des services). En effet, les décisions sont prises dans le cadre d'un raisonnement reposant sur « ce que l'utilisateur veut (les besoins/intentions ou buts) » et « ce que le système peut faire (les capacités du système) ». A partir de ce qu'il souhaite,

l'utilisateur peut identifier les services qui satisfont ses besoins. Ainsi, les modèles orientés intentions facilitent la sélection des services adéquats.

MeTSI propose de positionner la description du service au niveau intentionnel avec l'objectif de mettre en avant le but que ce service permet de réaliser et non la fonctionnalité qu'il permet d'atteindre. Le concept de *service intentionnel* est au cœur de notre approche.

L'approche orientée services intentionnels se place à un niveau plus abstrait que l'architecture SOA traditionnelle. Cette vision intentionnelle des services permet de rapprocher la description du service logiciel des exigences des utilisateurs finaux. L'approche orientée services intentionnels fait intervenir trois acteurs principaux [Kaabi 2007]: le client, l'annuaire et le fournisseur, dont les définitions sont centrées ″intention″. Le fournisseur expose des services intentionnels en les publiant dans l'annuaire de services. Un service intentionnel est défini comme un service logiciel permettant à un utilisateur de réaliser un but. Le fournisseur a la charge d'implémenter le service intentionnel par des services logiciels mais ceux-ci sont cachés à l'usager qui ne raisonne que par les services intentionnels. C'est donc le fournisseur qui met en place à la fois le front et le back office du service. L'annuaire de services permet d'effectuer une recherche dirigée par les buts de l'utilisateur (côté client à la figure 3.1). En effet, le descripteur de services est centré sur la définition du but que le service permet de satisfaire. L'objectif de l'annuaire est de mettre à la disposition des utilisateurs un référentiel de services intentionnels. La recherche des services se fait au travers d'un langage de requête qui permet de découvrir un service adapté à la demande par un processus de coïncidence d'intentions.

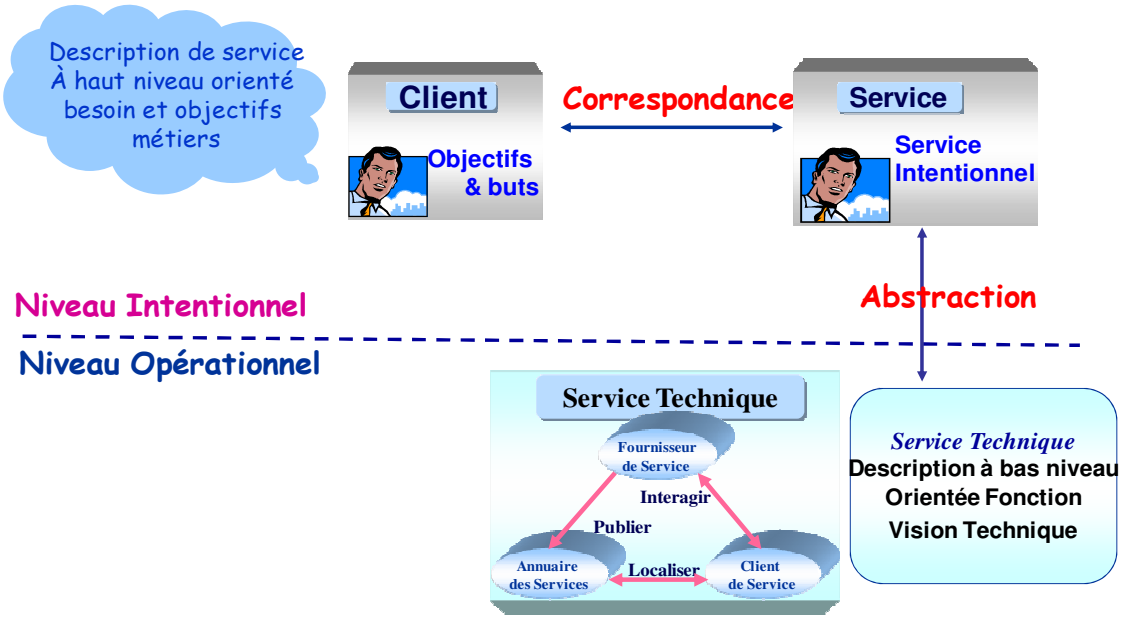


Figure 3.1. Approche orientée services intentionnels

La démarche transformationnelle MeTSI

La démarche adoptée dans cette thèse est une démarche inspirée de l'ingénierie dirigée par les modèles (IDM). En respectant les principes d'IDM, MeTSI est basée sur la construction et la transformation des modèles. MeTSI repose sur un processus de transformation constitué de 5 phases comme le montre la figure 3.2. Les descriptions de services sont des modèles qui instancient des méta-modèles. Les méta-modèles utilisés dans ce processus sont les suivants: le Méta-Modèle Intentionnel de Service (MIS), le Méta-Modèle de Scénario (phase intermédiaire) (MS), le Méta-Modèle Opérationnel de Service (MOS) et le Méta-Modèle d'Implémentation de Service Interactif (Phase intermédiaire) (MISI). En ce sens la démarche MeTSI est une démarche de type IDM (Ingénierie Dirigée par les modèles).

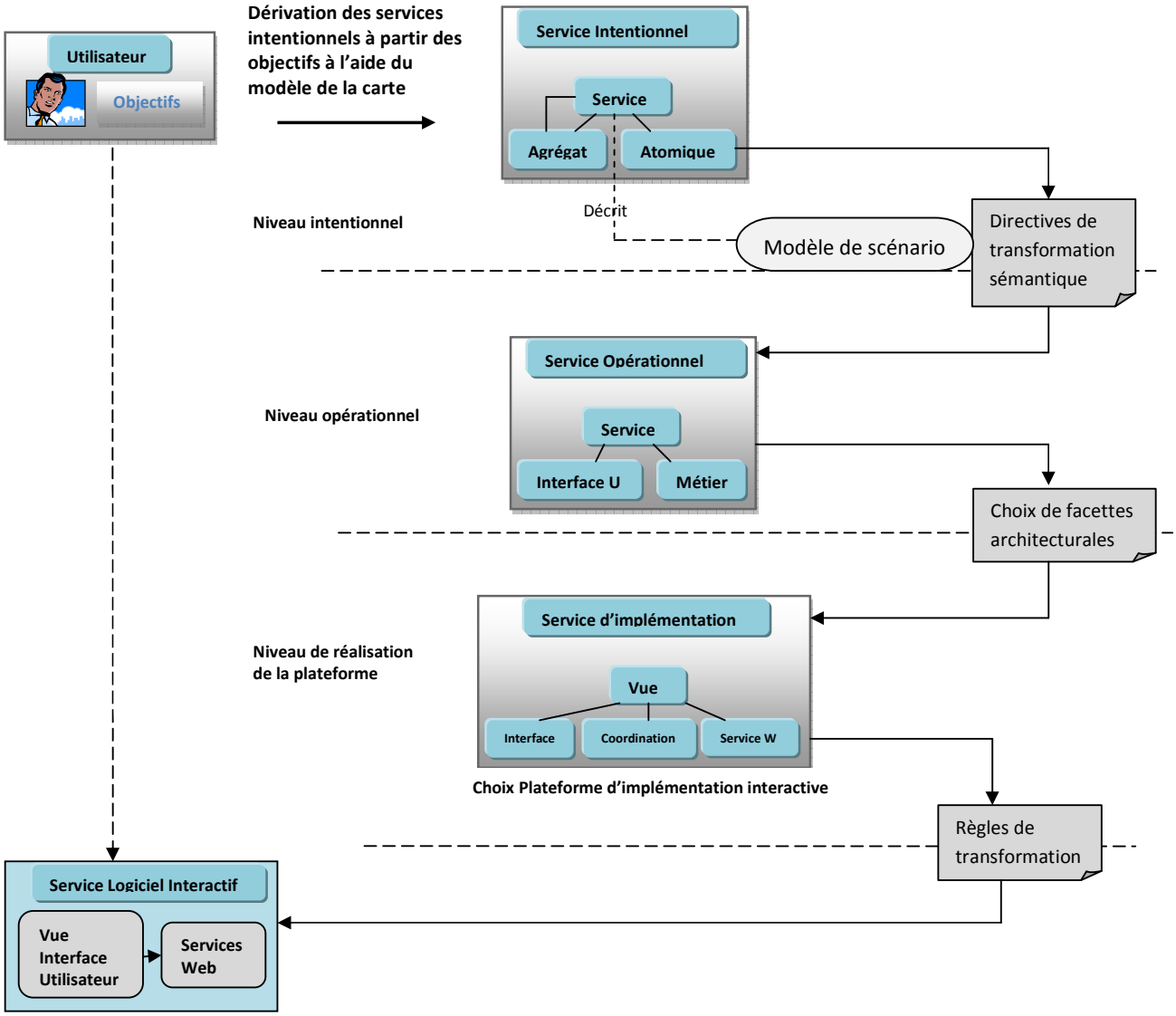


Figure 3.2. La démarche MeTSI

La phase de définition des services intentionnels *(Méta-Modèle Intentionnel de Service).* Le méta-modèle MIS est permet de représenter, à un niveau intentionnel, les services de l'application à développer et d'aider un client à exprimer ses buts et objectifs que des services logiciels permettront de réaliser. Cette phase aboutit à spécifier l'intention (qui est l'interface d'un service intentionnel), les ressources, les règles de transitions, les conditions (pré-/post-), la situation initiale et la situation finale. Le chapitre 4 rappelle en détail le méta-modèle MIS.

Les services intentionnels peuvent être (i) publiés par les fournisseurs dans l'annuaire des services et (ii) localisés par le client dans le cas où ils répondraient à leurs attentes par coïncidence d'intentions.

Le méta-modèle MIS introduit la variabilité dans la manière de réaliser l'intention d'un service. Les variantes correspondent aux différentes manières d'atteindre l'intention. Etant donné qu'un service intentionnel peut être composé d'autres services intentionnels, ayant chacun sa propre intention et par conséquent des variantes associées, il en résulte que le service intentionnel est défini comme un réseau de variantes attachées à des points de variation.
La composition dans le méta-modèle MIS s'adapte à la perspective intentionnelle du modèle et débouche sur une modélisation d'intentions qui s'inspire des graphes ET/OU des arbres de buts.

Les services intentionnels sont de deux types: agrégat et atomique. Un service intentionnel est agrégat lorsque l'intention est affinée par une ou plusieurs sous-intentions de services intentionnels. Par opposition, il est atomique lorsque l'intention n'est pas affinable mais est directement opérationnalisable et exécutable par un élément logiciel.

La phase d'analyse des scénarios *(Méta--Modèle de Scénario).* Cette phase enrichit la définition du service intentionnel en guidant le concepteur dans l'obtention d'une description du comportement interactif de l'usager et de l'application qui permet d'atteindre l'intention du service intentionnel. Ce guidage intervient dans deux activités: concrétiser une intention par un scénario et analyser un scénario pour identifier des cas fonctionnels alternatifs ou des cas exceptionnels de non satisfaction de l'intention. Cette phase aboutit à une description de l'application à développer dans son environnement intentionnel et la modélise en tant qu'ensemble d'interactions entre l'utilisateur et le back office métier. Cette phase est décrite en détail au chapitre 4.

La phase d'opérationnalisation de services intentionnels *(Méta-Modèle Opérationnel de Service).* Cette phase permet l'opérationnalisation d'un service intentionnel en ce qui est appelé un *service opérationnel interactif ou service opérationnel* en abrégé. La description d'un service opérationnel est un modèle qui instancie le méta-modèle MOS (méat-Modèle Opérationnel de Service). Cette phase transforme un modèle MIS (un service intentionnel) en un modèle MOS (un service opérationnel). Elle s'appuie sur un ensemble de règles de transformation.

Le service opérationnel interactif est un bloc insécable composé de deux composants, tous les deux modélisés comme des services: le service d'interface utilisateur, le service métier. Le service métier est défini en termes de trois sous-services de granularité fine interconnectés dans un bloc de services de grosse granularité par des flots de données en entrée et sortie et

par des ressources. Les interconnections sont des dépendances entre les différents sous-services. D'un point de vue architectural, le service opérationnel fournit deux sous-systèmes: le sous-système frontal (« front end ») qui représente les interactions avec l'utilisateur final à l'aide des interfaces, et le sous-système dorsal (« back end ») qui représente la composition de services web. Les services web sont obtenus par réutilisation, c'est-à-dire par la recherche dans un annuaire de services. Le service opérationnel est un service ayant aussi les trois aspects suivants: aspects techniques, aspects organisationnels et aspects informationnels. C'est aussi à cette phase que les requêtes XQuery sont faites pour l'invocation des services web. Cette phase est d'écrite en détail au chapitre 4.

La phase d'implémentation de services interactifs *(MISI).* La définition du service opérationnel obtenue à la fin de la phase précédente est indépendante de la plateforme. A contrario, cette phase vise à adapter le service opérationnel en prenant en compte les paramètres architecturaux. A cette fin, MeTSI a défini des classes d'architectures et des facettes architecturales qui permettent de les caractériser. Comme le montre la figure 3.3 il y a 2 grandes classes, celle des architectures de services interactifs et celle des architectures de composition interactive de services. Chacune est précisée par d'autres facettes. L'utilisateur s'appuie sur le méta-modèle architectural (classes d'architectures et facettes) pour faire son choix des différents paramètres architecturaux. Sur la base de ces choix, cette phase de MeTSI transforme le service opérationnel en un *service logiciel interactif* conforme au Modèle MISI. Il y a donc à nouveau, transformation de modèles : transformation d'un modèle MOS (un service opérationnel) en un modèle MISI (un service logiciel interactif). Le méta-modèle MISI décrit un service par trois sous-composants correspondant à trois vues qui sont la vue interface utilisateur, la vue coordination et la vue service Web. La vue interface utilisateur tient compte des interactions de l'application avec l'utilisateur. La vue coordination prend en charge l'orchestration des services web invoqués. L'emplacement des différentes vues est fonction de l'architecture choisie: soit chez le fournisseur de service, soit chez le consommateur de service. La description du service logiciel interactif ainsi obtenue est transformée dans la phase suivante pour s'adapter à l'une de trois plateformes: WSRP [WSRP 2006] (Web Service remote Portlet), l'OSGi [OSGi 2009] et les Mashups [Wikipedia, Mashup 2006].

La phase d'implémentation du service logiciel interactif. Cette phase aboutit à une solution finale sur l'une des plateformes suivantes: WSRP (Portlet Distant),OSGi, Mashup. Ces plateformes ont la particularité d'offrir des services composites qui combinent l'interface

utilisateur et la composition des services web. MeTSI génère un ensemble de règles qui permet d'implémenter un service conforme à MISI pour une des plateformes cibles.

Comme le décrit la figure 3.3, la démarche proposée permet de concevoir l'implémentation d'un service logiciel interactif à partir de la description du modèle intentionnel de service. Le développement dirigé par les modèles que nous adoptons est structuré en quatre niveaux. Le code généré est fonction de l'architecture et de la plateforme choisie. Les transformations de modèles peuvent être soit sémantique comme les transformations T1, T2 et T3 ou automatique comme les transformations T4, T5 et T6 qui génèrent des règles de transformations.

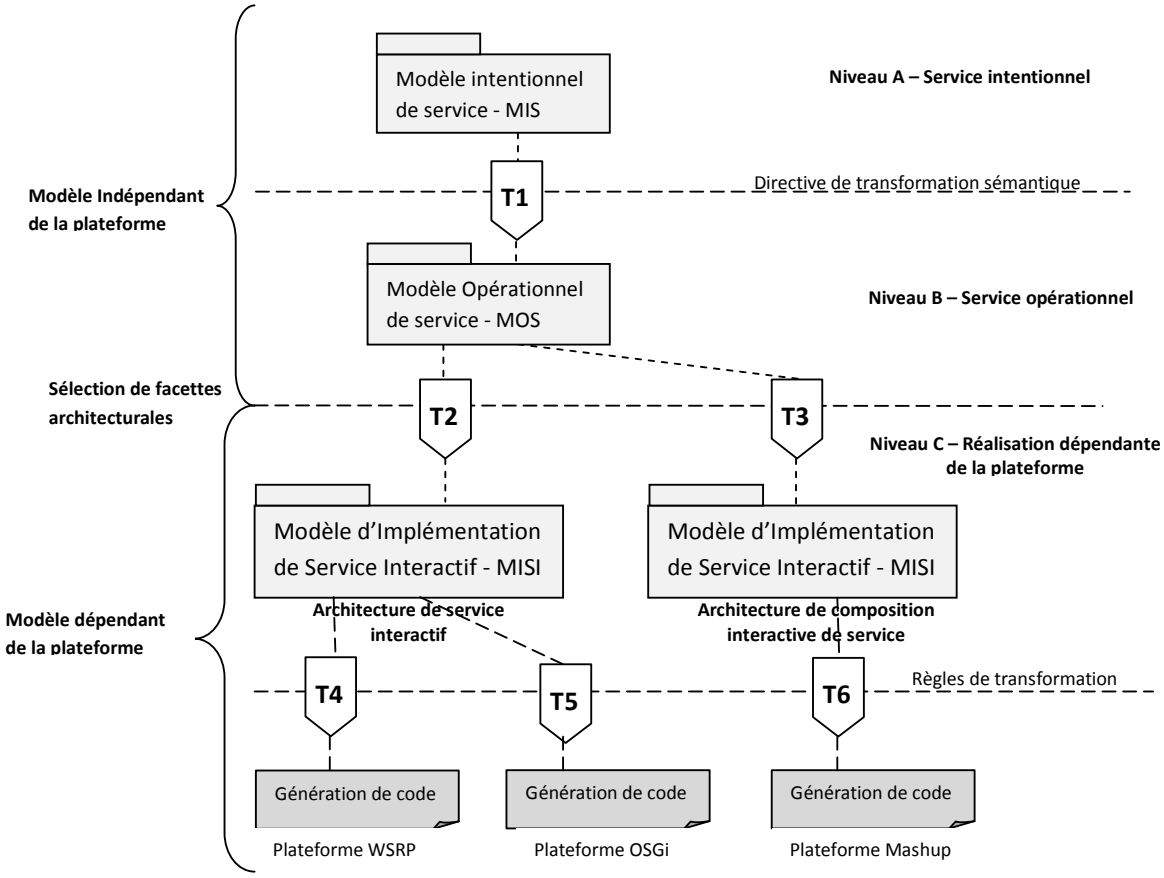


Figure 3.3. Phases de MeTSI

3. Fondements de l'ingénierie dirigée par les modèles

L'approche IDM vise à fournir un cadre conceptuel, technologique et méthodologique dans lequel les modèles sont au centre des activités de développement. Dans cette section, nous

rappelons brièvement les fondements de cette approche avant d'introduire les principes de son application. Nous considérons l'étude:

- des niveaux d'abstraction choisis,
- des transformations de modèles,
- des processus de développement à mettre en œuvre.

L'IDM spécifie l'intégralité du cycle de vie du logiciel comme un processus de production, de raffinement itératif et d'intégration de modèles. En ce sens, l'IDM fait évoluer l'usage des modèles. En effet, une traçabilité entre les éléments des différents modèles est gardée et ceci quel que soit leur niveau d'abstraction. L'utilisation des modèles permet de capitaliser les connaissances qu'elles soient des connaissances du domaine métier ou du domaine technique. Par ailleurs, la traçabilité (entre les différentes vues du système) ainsi que la transformation de modèles (entre les différentes phases de développement) permet de capitaliser le savoir-faire.

Les travaux menés sur l'IDM font suite à la définition de l'approche Model Driven Architecture (MDA) par l'OMG (Object Management Group) [OMG 2003]. MDA a recours aux différents standards de l'OMG pour décrire les démarches basées sur l'ingénierie des modèles. L'approche IDM entend ne pas se limiter au jeu de standards spécifiques à un organisme particulier, et désire proposer une vision unificatrice autour des notions de modèle et de méta-modèle.

Dans ce contexte, MDA est considéré comme un exemple particulier d'ingénierie dirigée par les modèles. Néanmoins, la plupart des travaux sur l'IDM font référence à MDA. Cet état de fait provient du fait que l'OMG catalyse, centralise, et synthétise bon nombre de travaux sur l'IDM. Pour cette raison, nous détaillerons dans ce qui suit plusieurs travaux proposés dans le cadre de MDA.

3.1. Les quatre niveaux de l'OMG

Afin de mieux comprendre l'approche IDM, il convient de préciser plus en détail la nature des modèles utilisés tout au long du cycle de vie, les divers usages de ces modèles, ainsi que les possibles intentions du concepteur au moment de leur construction. Ce qui est connu sous le sigle MDA (model Driven Architecture) a proposé une architecture à quatre niveaux qui structure les différents modèles pouvant être produits lors de l'application de l'approche IDM. Cette architecture fait maintenant l'objet d'un consensus [Seidewitz 2003], [Bézivin *et al.* 2005]. On retrouve ainsi des références à cette architecture dans de nombreux travaux désirant se situer par rapport à l'IDM [Ionita *et al.* 2005]. Cette architecture comporte quatre niveaux d'abstraction (cf. figure 3.2) que nous allons détailler dans ce qui suit.

Le niveau M0. Le niveau M0, représente les objets du monde du réel. Il représente, par exemple, un compte bancaire avec son numéro et son solde actuel (cf. figure 3.2).

Le niveau M1. C'est au niveau M1 que les modèles sont édités. Ces modèles sont conformes aux méta-modèles définis au niveau M2. Ainsi, MDA considère que si l'on veut décrire des informations appartenant au niveau M0, il faut d'abord construire un modèle appartenant au niveau M1. De ce fait, un modèle UML (comme le diagramme de classes ou le diagramme d'état/transition) est considéré comme appartenant au niveau M1. Il représenterait des objets manipulés dans le monde réel (décrits au niveau M0).

Le niveau M2. Le niveau M2, est le lieu de définition des méta-modèles. Un méta-modèle peut être considéré comme un langage spécialisé pour un aspect du système. Il peut aussi décrire les aspects spécifiques aux différents domaines, chaque aspect étant pris en compte dans un méta-modèle spécifique. Les méta-modèles contenus au niveau M2 sont tous des instances du niveau M3 (notons qu'au niveau M3, il ne peut y avoir qu'un seul méta-méta-modèle). Dans le cadre de MDA, c'est le méta-modèle d'UML [OMG 2004] qui est le plus utilisé, celui-ci définit la structure interne des modèles UML.

Le niveau M3. Le niveau supérieur, M3 correspond au méta-méta-modèle. Il définit les notions de base permettant l'expression des méta-modèles (niveau M2), et des modèles (M1). Pour éviter la multiplication des niveaux d'abstraction, le niveau M3 est réflexif, c'est-à-dire qu'il se définit par lui-même. Le plus souvent, c'est le méta-méta-modèle MOF (Meta Object Facility) qui est utilisé. Celui-ci est standardisé par l'OMG [OMG 2006]. Cependant d'autres méta-méta-modèles ont été proposés tels qu'eCore défini dans le cadre de l' "Eclipse Modeling Framework" (EMF) [Budinsky *et al.* 2003] et d'OWL [Dinh *et al.* 2006]. Des plates-formes de modélisation génériques implémentant le MOF permettent la production de méta-modèles spécifiques à des domaines et de produire ensuite des modeleurs spécifiques à ces domaines. Par exemple, la plateforme de méta-modélisation GME [Ledeczi *et al.* 2001] est compatible avec eCore d'EMF [Bézivin *et al.* 2005]. Afin d'avoir une idée plus précise de l'architecture à quatre niveaux de l'OMG, nous illustrons à la figure 3.2, les modèles pouvant appartenir à chaque couche.

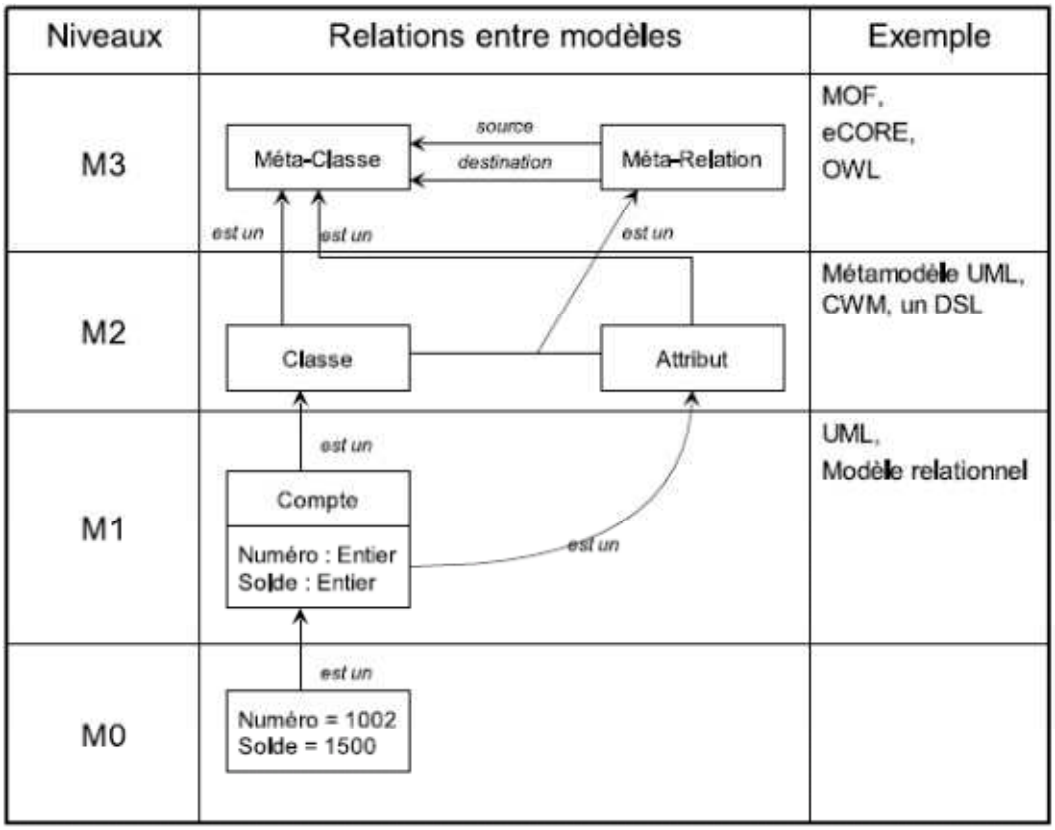


Figure 3.2. Les quatre niveau du MDA

3.2. La transformation de modèles

Les transformations de modèles sont au cœur de l'approche de l'ingénierie dirigée par les modèles. Elles représentent l'un des grands défis à relever d'un point de vue technique pour envisager une large diffusion de l'ingénierie dirigée par les modèles. La transformation de modèles intéresse aussi bien les chercheurs que les industriels. En effet, leur mise en œuvre augmentera considérablement la productivité et la qualité des logiciels produits.

Cependant, à l'heure actuelle, il n'existe pas encore de consensus sur la définition et la mise en œuvre d'une transformation. Une transformation est une opération qui sert à produire un modèle (appelé modèle cible) à partir d'un autre modèle (appelé modèle source). Plusieurs sortes de modèles cibles et sources peuvent exister. Selon la nature des modèles, plusieurs types de transformation sont répertoriés [Mens *et al.* 2005]:

- Transformation du *modèle vers le code*: dans l'ingénierie dirigée par les modèles, le code source est considéré comme un modèle ayant un niveau d'abstraction bas (modèle concret). Une telle transformation consiste à générer du code à partir d'une spécification.

- Transformation du *code vers le modèle*: dans une transformation le code source peut aussi jouer le rôle de modèle source. Il s'agit alors de réingénierie des systèmes.
- Transformation *endogènes* et *exogènes:* la transformation endogène implique des modèles sources et cibles conformes au même méta-modèle; alors qu'une transformation exogène implique des modèles issus de méta-modèles différents. Dans ce dernier cas, on parle de *translation*.

La génération de code, la réingénierie ainsi que les migrations (cf. figure 3.4) sont des exemples typiques de translation (ou transformation exogène). Les transformations endogènes peuvent quant à elles servir à l'optimisation (pour améliorer la qualité du modèle), la refactorisation (pour améliorer la lisibilité du modèle), la simplification ou normalisation (pour limiter la complexité syntaxique relative à un modèle, il s'agit par exemple de rajouter du sucre syntaxique).

- Transformation verticale et transformation horizontale: une transformation horizontale est une transformation où le modèle cible et le modèle source appartiennent au même niveau d'abstraction (un exemple de cette transformation est la refactorisation). La transformation verticale s'applique sur des modèles appartenant à des niveaux d'abstraction différents (exemple génération de code).

MDA définit le langage QVT (Query/View/Transformation) [OMG 2005] comme standard pour la spécification des règles de transformation. Ce langage est formé de trois parties:

- la définition de requêtes qui permet une navigation dans les modèles,
- la spécification de vues qui permet de focaliser sur une partie des modèles,
- la spécification de règles de transformations de modèles dont l'application permet de générer un modèle source d'un modèle cible.

Différents langages de transformation implémentent QVT (exemples Smart-QVT [Smart 2006], Medini QVT [Höbler*et al.* 2006], OptimalJ[Jonkers*et al.* 2007]) ou s'en rapprochent (principalement ATL [Jouault et al. 2006]). Ces langages se comparent selon différents critères, définis par Czarnecki*et* Helsen[Czarnecki*et*Helsen 2003]:

- Les règles de transformation: avec ce critère, nous pouvons regarder la structure de construction des règles de transformation. Les règles peuvent être paramétrées, décrites d'une manière déclarative, impérative ou hybride (à la fois déclarative et impérative).
- La portée d'application des règles: les règles de transformation peuvent avoir comme portée la totalité du modèle ou une partie du modèle.

- Les relations entre modèle source et modèle cible: ce critère compare la nature de relations qui peut exister entre le modèle cible et le modèle source. Celles-ci peuvent être de deux types; le modèle cible est un nouveau modèle différent, créé à partir du modèle source ou le modèle cible est le même que le modèle source sur lequel des modifications ont été faites.
- La stratégie d'application des règles: l'application des règles peut être déterministe (c'est-à-dire que l'exécution des règles est faite de manière préétablie ou standard comme par exemple le parcours classique d'un arbre); ou bien indéterministe (c'est-à-dire pouvant être effectuées dans un mode interactif voire concurrent).
- L'ordonnancement des règles: ce critère vérifie si l'ordre d'application des règles peut être explicité. Dans le cas contraire c'est l'outil interprétant les règles qui détermine l'ordre.
- L'organisation des règles: ce critère vérifie s'il est possible de rassembler et d'organiser les règles dans des modules. Il vérifie également si elles peuvent être réutilisées par d'autres règles en utilisant le mécanisme d'héritage ou de composition.
- La traçabilité: durant la transformation de modèles, les liens de traçabilité entre modèles peuvent être conservés.
- La direction: les règles de transformation peuvent être appliquées de manière unidirectionnelle ou bidirectionnelle (dans ce cas la traçabilité entre les modèles doit être conservée).

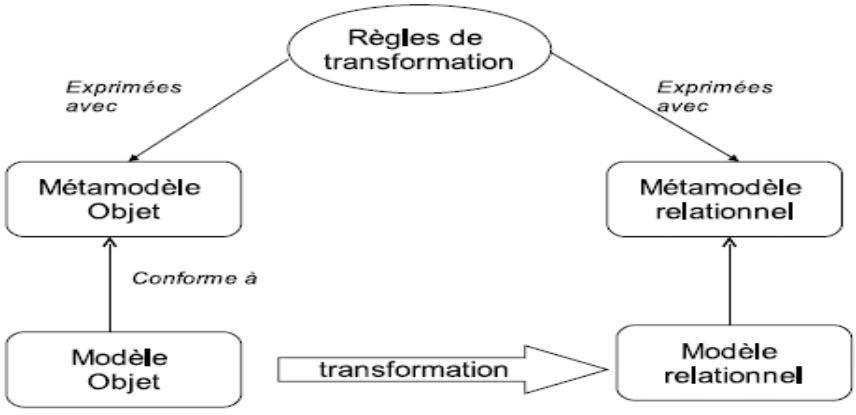


Figure. 3.4. Migration de modèle objet à un modèle relationnel

3.3. Processus de développement orienté par les modèles

Le modèle de cycle de développement ne fait pas encore l'unanimité dans l'ingénierie dirigée par les modèles. Cependant, nous avons pu distinguer deux sortes de modèles de cycle de vie.

Le premier a été proposé dans le cadre de l'approche MDA. Le second est considéré comme plus généraliste et est basé sur une adaptation du cycle de vie en Y. Chacun de ces modèles est présenté dans les paragraphes suivants.

i. Le cycle de développement de MDA

Le cycle de développement de MDA est comparable à un cycle de développement classique tel que le processus en cascade. On y trouve en effet les mêmes phases de développement à savoir l'expression des besoins, l'analyse, la conception, l'implémentation, la validation et le déploiement.

La différence réside cependant dans la nature des artefacts produits à chacune des phases. La figure 3.5 décrit ce processus de développement.

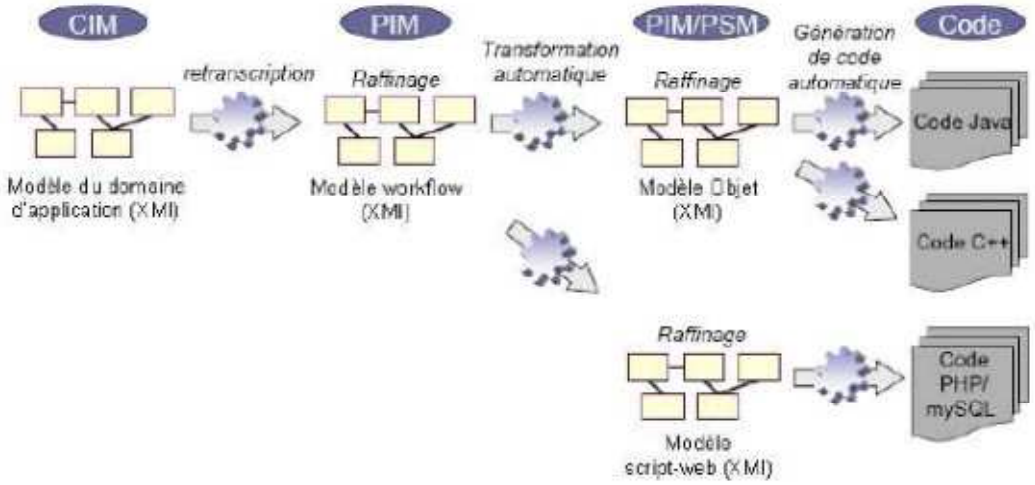


Figure 3.5. Le cycle de vie de MDA

Le premier modèle produit appelé Computer Independent Model (CIM) correspond à la description en langage naturel du domaine d'application. Dans ce modèle aucune considération informatique n'apparaît. Par exemple, dans le domaine Bancaire, il pourrait s'agir d'un texte indiquant que certaines compensations peuvent être traitées le soir. Un modèle CIM sert de base à la définition du modèle PIM (Platform Independent Model) où le méta-modèle sous-jacent est cette fois-ci de type informatique mais non rattaché à une technologie particulière (ex: un modèle objet). Le passage d'un modèle CIM à un modèle PIM est normalement manuel et nécessite plusieurs discussions entre experts du domaine et informaticiens bien que certains travaux préconisent une projection automatique [Zhang et al. 2005]. Le modèle PIM est, dans une troisième étape raffiné afin d'indiquer des aspects propres au paradigme de développement qui va être utilisé (ex: l'objet, le procédural, etc.). La

quatrième étape consiste à produire, en appliquant une transformation automatique, un modèle PSM (Platform Specific Model) c'est-à-dire un modèle spécifique à une plateforme. Un modèle EJB (modèle UML utilisant le profil EJB) est un exemple de modèle de type PSM. Après raffinement, ce type de modèle sert à générer automatiquement une partie du code final qui va être par la suite lui-même raffiné puis testé (manuellement) par le développeur. Ce processus de développement a cependant fait l'objet de critiques. Bien qu'il semble simple de premier abord, plusieurs travaux récents portant sur les compositions de modèles ont montré la difficulté de relier entre eux les différents modèles représentant différents aspects d'une application [Estublier *et al.* 2005]. D'autres difficultés ont été relevées quant à l'utilisation d'UML et du MOF. Ceux-ci sont considérés comme très généralistes et difficilement manipulables par des non spécialistes. Microsoft par exemple a fait le choix de privilégier des langages de domaines (Domain Specific Languages ou DSL) de petite taille, facilement manipulables et transformables [Jack et al. 2004].

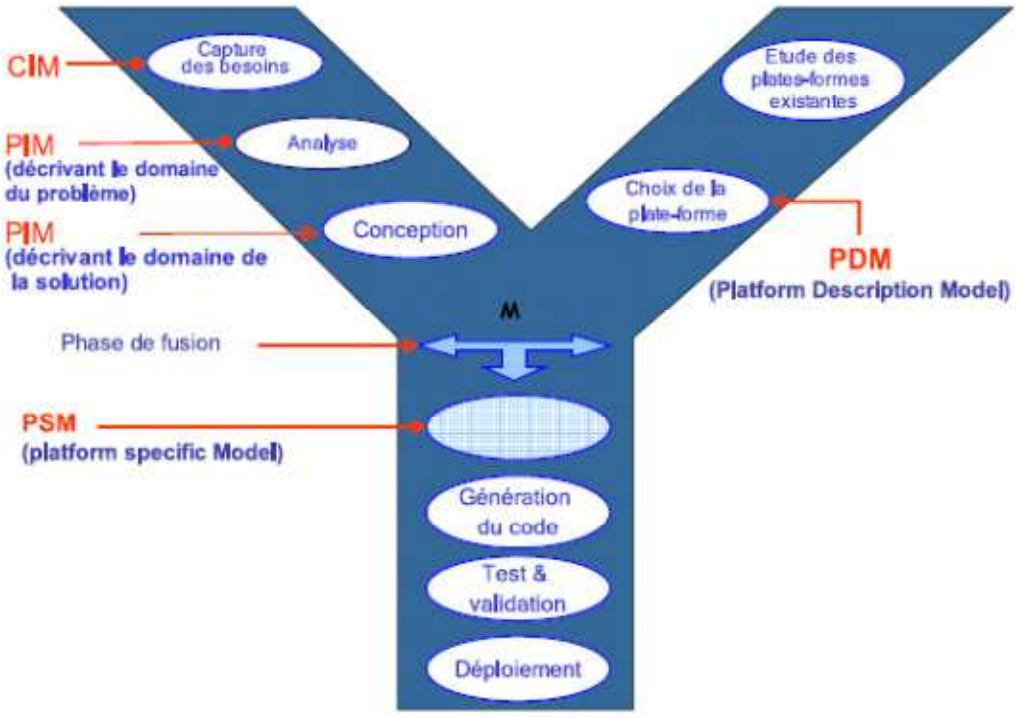


Figure 3.6. Le cycle de vie en Y

ii. Processus de développement en Y

Plusieurs travaux concernant l'IDM font référence au cycle de développement en Y [Boulet *et al.* 2002], [Kadima 2005]. Initialement, ce processus de développement avait pour objectif

une maîtrise des coûts et délais grâce à une parallélisation des tâches d'analyse et de conception [Larvet 1994] mais certains travaux l'ont adapté dans le cadre de l'IDM [Belangour *et al.* 2006]. Dans ce contexte, il s'agit de mettre en parallèle l'élaboration du PIM et la description de la plateforme. Une jonction entre les deux branches va constituer le PSM. La figure 3.6 illustre ce processus.

Selon ce processus, les phases d'analyse/conception sont menées en parallèle avec l'étude des différentes plates-formes existantes. Durant l'analyse et la conception, les modèles CIM et PIM sont produits comme le préconise le processus fourni par MDA. Notons qu'un premier PIM est élaboré décrivant le domaine du problème. Celui-ci est ensuite raffiné pour produire un PIM décrivant le domaine de la solution. Ce raffinement doit cependant conserver l'indépendance des modèles par rapport aux détails d'implémentation. Il s'agirait de spécifier l'échange de flux de données entre les différentes entités du système. Parallèlement, les aspects techniques de chaque plateforme sont étudiés et le choix de la plateforme d'implémentation est effectué. Les caractéristiques techniques de celle-ci sont modélisées dans le PDM (Platform Description Model). Celui-ci va décrire par exemple, les types supportés par la plateforme (exemple: une chaîne de caractère en JAVA correspond à un objet qui instancie la classe String). Une mise en correspondance est ensuite effectuée entre les entités décrites dans le PIM et celles qui leur correspond dans le PDM.

C'est à la suite de la mise en correspondance PDM/PIM que le PSM est généré. Celui-ci est donc le résultat de la fusion du PDM et du PSM. Une fois le PSM effectué, le code est généré. Ensuite il est raffiné, testé et validé par le développeur avant que le déploiement n'ait lieu.

4. Définition du modèle de description intentionnelle de services

Cette section présente le méta-modèle des services intentionnels dénommé MIS (Modèle Intentionnel de Service) décrit dans [Kaabi 2007]. Cette partie, introduit le modèle MIS sans toutefois en spécifier les détails qui ont présentés dans [Kaabi 2007].

4.1. Meta- Modèle Intentionnel de Services

Comme précédemment défini dans [Kaabi 2007], MIS présente les trois caractéristiques suivantes:

- **Le besoin d'une meilleure homogénéité dans l'expression intentionnelle des besoins des clients et celles des services.**

Chaque service intentionnel s'adapte à une situation particulière afin de réaliser une intention particulière. Les services intentionnels peuvent être (i) publiés par les fournisseurs dans l'annuaire des services et (ii) localisés par le client dans le cas où ils répondraient à leurs attentes par coïncidence d'intentions.

- **Le besoin de variabilité dans le monde des services.**

Le besoin de variabilité dans le monde des services émerge à la suite d'un changement du comportement des utilisateurs, ne voulant plus s'adapter aux capacités des logiciels et qui préfèrent que les logiciels se configurent en fonction de leurs besoins.

Le méta-modèle MIS introduit la variabilité dans la manière de réaliser l'intention d'un service. Les variantes correspondent aux différentes manières d'atteindre l'intention. Etant donné qu'un service intentionnel peut être composé d'autres services intentionnels, ayant chacun leur propre intention et par conséquent des variantes associées, il en résulte que le service intentionnel est défini comme un réseau de variantes attachées à des points de variation.

- **La réutilisation des services.**

La réutilisation des services est mise en œuvre par des mécanismes de réflexivité: un service composite étant un service à part entière, réutilisable dans d'autres compositions de services. La composition dans le méta-modèle MIS s'adapte à la perspective intentionnelle du modèle et débouche sur une composition dirigée par les intentions. Notre proposition prend la forme d'une composition de services qui s'inspire des graphes ET/OU des arbres de buts. La composition de services dirigée par les intentions introduit une composition à plusieurs niveaux: l'intention du service de plus haut niveau, qui peut être de nature stratégique, est décomposé en sous intentions/services qui eux-mêmes, peuvent nécessiter une nouvelle décomposition jusqu'à atteindre des intentions/services opérationalisables. Il y a donc une composition récursive des services.

On observe aussi que la perspective intentionnelle choisie dans cette approche se traduit par le fait qu'un service exhibe une intentionnalité formulée par l'*intention* qu'il permet à ses clients d'atteindre. Une intention reflète un but, un objectif que l'on cherche à atteindre sans pour autant dire comment l'atteindre. Une intention est associée à un résultat souhaité matérialiser par un ensemble d'états d'objets. Par exemple l'intention *Payer une réservation par carte de crédit* correspond à un résultat escompté: le paiement effectif de la réservation, et plus précisément se décline par l'état de l'objet réservation: l'état réservation. Statut= 'payée'.

Chercher un service intentionnel revient à établir la coïncidence entre l'intention poursuivie (celle qu'un client d'un annuaire de services cherche à atteindre) et l'intention affichée par le service (celle dont le service garantit la satisfaction).

L'intentionnalité sous entend que le client, qui agit de manière intentionnelle, fasse son possible pour atteindre le résultat préfiguré par l'intention. On peut voir le service comme un acteur agissant de manière intentionnelle c'est-à-dire ayant les moyens de permettre au client d'atteindre l'intention visée. Les sections suivantes montrent que ce sont les services atomiques qui sont porteurs des actions exécutables nécessaires à la réalisation de l'intention et que ce sont les services agrégats qui orchestrent la composition des actions de base.

La Figure 3.7 présente le méta-modèle MIS en utilisant la notation du diagramme de classes UML.

Par le jeu des trois couleurs utilisées (gris clair, blanc, gris foncé), la figure 3.7 montre que la description d'un service intentionnel comporte trois parties correspondant à son *interface*, l'expression de son *comportement* et celle de sa *composition.*

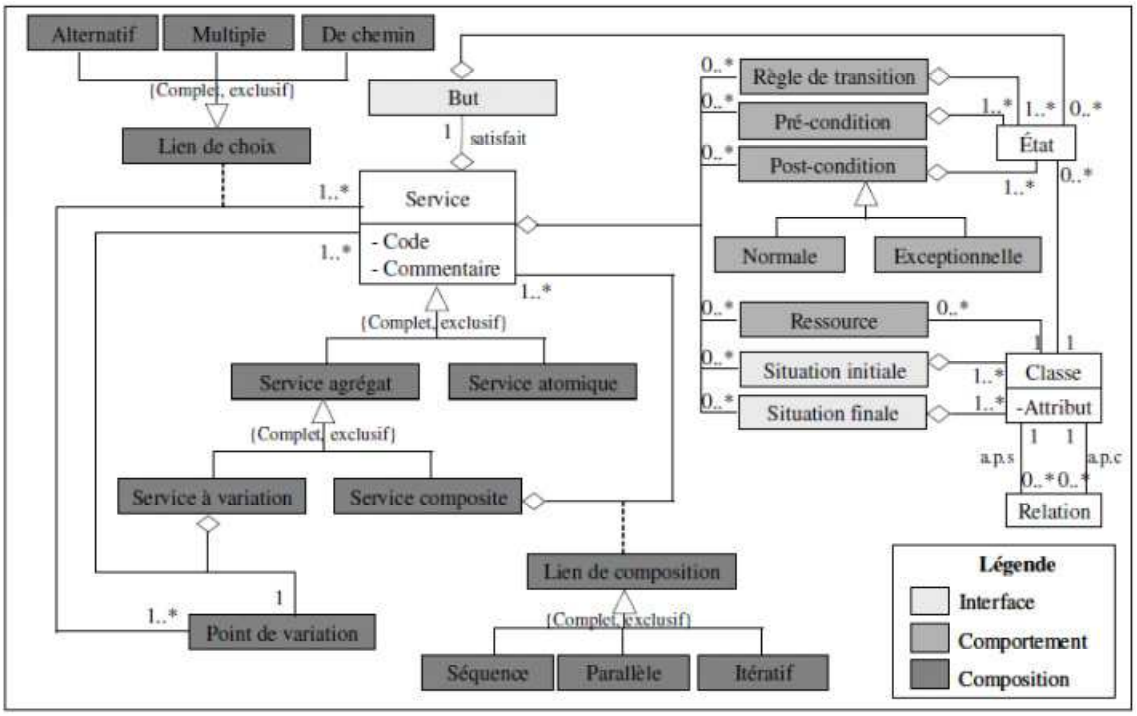


Figure 3.7. Le Modèle Intentionnel de Services (MIS) [Kaabi 2007]

L'interface d'un service est la partie coloriée en gris clair sur la figure 3.7. L'interface, comme dans toute description de service, est la partie visible depuis l'extérieur, celle qui permet de comprendre ce que sait faire le service sans entrer dans les détails du corps du service. Dans le cas du modèle MIS, la partie clé de l'interface est l'***intention*** qui remplace les méthodes ou opérations de l'interface d'un service logiciel. Elle est complétée par la

description des paramètres d'entrée et de sortie que l'on désigne, ***situation initiale*** et ***situation finale***, respectivement. Selon l'acception générale du monde des services, l'interface considère le service comme une boîte noire qui dans notre cas, assure la réalisation de l'intention en utilisant un ensemble de paramètres d'entrée (l'instance de la structure de classes décrite par la situation initiale) afin de produire un ensemble de paramètres en sortie qui correspondent à la situation finale (l'instance de la structure de classes décrite par la situation finale).

L'interface du service intentionnel permet d'affiner une recherche de services dirigée par les intentions. Pour retrouver le ou les services qui coïncident avec les exigences d'un client exprimées par une ou plusieurs intentions, le processus de correspondance opérera non seulement sur l'expression de l'intention elle-même mais aussi sur la situation initiale et la situation finale.

La deuxième partie descriptive du service est celle relative à son ***comportement***. Cette partie est coloriée en gris foncé au niveau de la Figure 3.7. Dans l'optique intentionnelle de MIS, un service se comporte de façon à satisfaire l'intention qui lui est associée. Sachant qu'une intention est caractérisée par un ensemble d'états d'objets qui traduisent le résultat obtenu par la satisfaction de l'intention, il est usuel de définir le processus de réalisation d'une intention par deux ensemble d'états: celui de la situation initiale (au départ du processus de réalisation de l'intention) et celui de la situation finale (lorsque le processus de réalisation de l'intention est achevé). Nous proposons donc dans MIS, de qualifier le comportement d'un service par (i) deux conditions d'états désignées ***pré-condition*** et ***post-condition*** et (ii) une règle de transition. Les conditions sont respectivement des conditions des états des objets de la situation initiale et la situation finale. La ***Règle de transition*** permet de préciser les contraintes régissant la réalisation d'opérations entraînant le changement d'états d'objets. Ainsi, un service peut être vu comme la transition d'une pré-condition vers une post-condition résultant de la réalisation du but du service par l'exécution de règles de transition associées à un service.

La *pré-condition* précise les états des objets constituant les conditions devant être satisfaites pour que le but du service puisse être satisfait. La *post-condition* précise les états des objets obtenus suite à la réalisation de l'intention du service.

La troisième partie descriptive du service est celle de sa *composition*. Comme le montre la Figure 3.7, il y a différents types de compositions de services. Au premier niveau de spécialisation les services sont soit ***atomiques*** soit ***agrégats***. Un ***service atomique*** n'est pas

décomposable en d'autres services intentionnels alors qu'un ***service agrégat*** l'est. Cette classification prend sa source dans la nature des buts associés aux services.

Un **service atomique** est associé à une intention que l'on qualifie d'**opérationnalisable**', c'est-à-dire pour laquelle on est capable de définir une séquence d'actions qui permet de la réaliser.

Dans notre cas, ce sont les actions qui composent le service atomique qui permettent d'assurer l'opérationnalisation de l'intention du service atomique. Cela permet de dire qu'un **service atomique est *exécutable***. Par exemple le service permettant de *Payer une réservation par carte de crédit* est un service atomique car l'intention qui lui associée est opérationnalisable dans le sens où l'on sait définir un processus de paiement d'une réservation par carte de crédit.

Les intentions de haut niveau tels que les objectifs stratégiques ne sont pas d'emblée 'opérationnalisables'. Elles doivent être décomposés en sous intentions qui elles-mêmes peuvent nécessiter une décomposition etc. jusqu'à atteindre des intentions opérationnalisables. Les services agrégats sont associés à des intentions de haut niveau et la composition d'un service agrégat en services suit la décomposition de l'intention père en sous intentions. Les services agrégats ne sont pas directement exécutables mais leur composition en services qui eux-mêmes, peuvent récursivement être décomposés en services est le moyen par lequel le service agrégat permet, indirectement l'exécution d'actions logicielles. Par exemple l'intention *Optimiser le remplissage des hôtels* n'est pas une intention opérationnalisable; elle requiert la satisfaction d'autres intentions telles que *Gérer une liste d'attente des demandes non satisfaites*, *Suggérer des modifications de demandes, Garantir la mise à jour instantanée des ressources* etc. Chacune de ces intentions est associée à un service composant du service *Optimiser le remplissage des hôtels* qui lui-même peut être un service agrégat. On comprend donc que la composition de MIS est elle aussi dirigée par les intentions.

L'affinement OU est utile dans le contexte de MIS car il permet d'introduire plusieurs décompositions alternatives d'intentions. Le méta-modèle MIS comporte la notion de *service à variation* pour satisfaire ce besoin. Un service à variation introduit de la flexibilité dans la manière de satisfaire une intention et donc de la flexibilité dans la composition de services et permet d'adapter la manière de rendre un service aux circonstances particulières desa demande.

Toutes les notions d'interface, de comportement et de composition sont développées plus en détail dans les travaux de [Kaabi 2007]. Dans la méthode de services intentionnels

MeTSI, le méta-modèle MIS constitue le point de départ de l'approche d'ingénierie dirigée par les modèles.

La section suivante, décrit le point d'arrivée de la méthode qui constitue les plateformes d'implémentation des services intentionnels. Les plateformes sont le résultat pour la mise en œuvre d'une approche orientée intentions, elles décrivent le service logiciel interactif final qui sera produit à la fin du processus de développement de l'approche.

5. Les plateformes d'implémentation retenues dans la thèse

Comme indiqué précédemment le produit final de MeTSI est un service logiciel interactif qui se compose de trois vues insécables et inter-reliées: vue interface utilisateur, vue coordination et vue métier. Les différents méta-modèles composant le processus de développement de services intentionnels seront détaillés dans les prochains chapitres. Cette section introduit les différentes plateformes d'implémentation des services interactifs qui permettent de mettre en place l'approche MeTSI. Ces plateformes d'implémentation, au nombre de trois, Mashup, WSRP (Web Service Remote Portlet) et OSGi, ont des spécifications différentes selon l'intention de l'utilisateur et prennent en compte les trois caractéristiques du services du service logiciel interactif à savoir la vue interface utilisateur, la vue coordination et la vue métier. Cependant, elles diffèrent dans certains aspects qui sont détaillés ci-dessous.

5.2. La plateforme OSGi

5.2.1 Présentation

L'Alliance OSGi™ a été fondée en mars 1999. Elle propose des standards pour les services internet à destination des maisons, voitures, téléphones mobiles, ordinateurs de bureau, PMI-PME et d'autres environnements. Ses propositions sont concrétisées sous la forme de spécifications ouvertes pour la livraison de services sur des réseaux locaux, ces services peuvent être gérés et délivrés à distance via l'utilisation de tout type de réseau.

La spécification de la plateforme OSGi [OSGi 2009] propose une architecture commune et ouverte pour les fournisseurs de services, les développeurs, les éditeurs de logiciels, les opérateurs de passerelles (par exemple, les décodeurs satellites) et les fabricants de telles passerelles, afin que tous puissent développer, déployer et gérer des services de manière cohérente.

L'environnement d'exécution (c'est-à-dire le *framework*) est le noyau de la spécification de la plateforme OSGi. Cet environnement d'exécution est destiné à un usage général. Il a pour caractéristiques d'être basé sur la technologie Java, ainsi que d'être géré et sécurisé. Il supporte aussi le déploiement (local et à distance) de *bundles*.

Les *bundles* peuvent avoir deux types de dépendances, des dépendances au niveau package (au sens package Java) et des dépendances de services via la spécification d'un ou plusieurs services requis (par le biais d'une ou plusieurs interfaces Java). La plateforme OSGi offre des mécanismes qui permettent de connaitre l'état de résolution (satisfaction) de ces dépendances. De plus, la spécification OSGi présente les *bundles* comme des applications extensibles.

La plateforme OSGi offre, au développeur de *bundle*, les ressources et les points d'entrée suffisants pour tirer parti, d'une part, de l'indépendance qui est donnée vis-à-vis de la plateforme Java (JVM) et, d'autre part, de la capacité de chargement dynamique de code (c'est-à-dire de classes) de la plateforme OSGi. Ces deux propriétés permettent de développer des services pour des passerelles relevant du domaine de l'embarqué/réactif. De telles passerelles sont massivement répandues, par exemple, les décodeurs satellites dans les maisons, ou les ordinateurs de bords dans des voitures, typiquement, le 4x4 X5 de BMW embarque une plateforme OSGi.

Les fonctionnalités de la plateforme OSGi sont réparties selon plusieurs couches:

- la couche sécurité (*Security Layer*),
- la couche *bundle* (*Module Layer*),
- la couche concernant le cycle de vie des *bundles* (*Life Cycle Layer*)
- et enfin, la couche service (*Service Layer*).

Une vue globale de la plateforme à service OSGi sur une passerelle générique est présentée dans la figure ci-dessous.

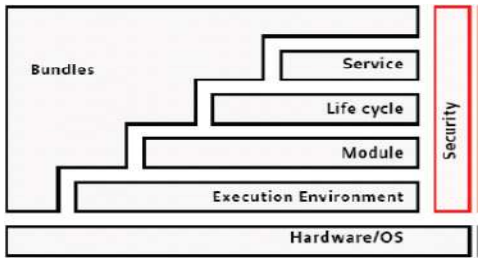


Figure 3.8. Les différentes couches de la plateforme OSGi [OSGi07].

5.2.2 La plateforme OSGi

La couche sécurité est une couche qui est transverse aux différentes couches de la plateforme OSGi et qui est optionnelle. Elle est basée sur l'architecture *Java 2 Security*. Elle offre une infrastructure pour déployer et gérer des bundles OSGi qui doivent s'exécuter dans des environnements sécurisés à grain-fin. Les possibilités concernant la sécurité vont jusqu'au niveau applicatif.

La couche *bundle* définit son propre modèle de modularisation au-dessus de la technologie Java. Cette couche possède des règles strictes concernant le partage de packages Java entre *bundles* et la mise à disposition de packages Java internes à un bundle.

La couche *cycle de vie* définit le cycle de vie des bundles. Elle offre aussi une API permettant de gérer effectivement le cycle de vie de chaque bundle. Cette API définit un modèle d'exécution pour les bundles. Ce modèle d'exécution définit le déploiement des bundles. Les activités de déploiement qui sont couvertes sont l'installation, l'activation, la désactivation, la mise à jour statique et la désinstallation. Ces activités correspondent respectivement aux commandes: *install*, *start*, *stop*, *update* et *uninstall*.

Enfin, la couche *service* d'OSGi fournit les mécanismes liés à l'approche à service. Elle définit un modèle de programmation dynamique pour les développeurs de bundle. Ce modèle simplifie le développement et le déploiement de bundles orientés service en découplant le contrat des services de leur implémentation. Dans OSGi, le contrat d'un service est un contrat de niveau 1 (concrètement, c'est une interface Java), l'implémentation d'un service est, quant à elle, enfouie dans le bundle. Ce modèle permet aux développeurs de bundles de définir des fournisseurs de services et des demandeurs de services. Les demandeurs définissent implicitement des dépendances sur des contrats de services. Lors de son activation, chaque bundle cherche à résoudre ces dépendances de contrats de services, cette résolution se fait dans le cadre fixé par le patron SOA. Une fois qu'une dépendance est résolue, le demandeur et le fournisseur du contrat de service entrent en interaction. Il est intéressant de noter qu'un demandeur de service qui s'est lié avec un fournisseur de service pourra tout à fait en changer tout au long de son exécution et cela selon sa propre politique, par exemple, lors de l'apparition d'un nouveau fournisseur de service ou encore si le fournisseur actuellement choisit vient à ne plus être disponible. Enfin, il faut noter que lorsqu'un bundle requérant un service vient à être activé et qu'il cherche un fournisseur pour ce service (afin de satisfaire son requis de service), il ne sait pas par avance avec quel fournisseur de service (c'est-à-dire bundle, ou plus précisément, implémentation de service) il

va interagir. En effet, les dépendances de service d'un bundle sont définies dès son développement, mais le réel ensemble de bundles interagissant n'apparait que lors de l'exécution effective de chaque bundle, en fonction des bundles déployés dans la même plateforme à service et de la stratégie de sélection qu'il leur applique.

L'environnement d'exécution OSGi autorise un bundle à choisir les implémentations de services correspondant à ses requis de services et cela aussi bien au cours de son activation, qu'à l'exécution (c'est à- dire une fois que le bundle est activé). Ce choix s'effectue en suivant le patron SOA, c'est-à-dire en utilisant le registre de service de l'environnement d'exécution OSGi. Ainsi, les bundles activés peuvent enregistrer leurs propres fournisseurs de services, recevoir des notifications à propos de tous les fournisseurs de services présents dans le registre de services. Ils peuvent aussi chercher des fournisseurs de services dans le registre de service, récupérer ceux qui leur conviennent et relâcher ceux qui ne leur conviennent plus. Cet aspect de l'environnement d'exécution OSGi permet aux bundles déployés de s'adapter au contexte dans lequel ils se trouvent. De plus, cette faculté d'adaptation ne nécessite pas la réactivation (c'est-à-dire la désactivation puis l'activation), ni de l'environnement d'exécution, ni du bundle embarquant les demandeurs de services. Par opposition, la mise à jour statique d'un bundle nécessite, quant à elle, la désactivation du bundle, sa désinstallation, l'installation et optionnellement l'activation du "nouveau" bundle (ces quatre activités peuvent être enchainées dans l'ordre présenté ici, mais aussi selon d'autres ordres).

Comme nous l'avons vu, OSGi introduit le concept de *bundle*. Un bundle est une entité logicielle qui contient des classes, mais aussi des ressources (par exemple, des fichiers images, des sons), ainsi qu'un ensemble de métadonnées sous la forme d'un fichier nommé manifest.mf. De même, un bundle peut être le fournisseur de zéro, un ou plusieurs services et le demandeur de zéro, un ou plusieurs services. Il est enfin important de noter que le bundle est l'unité de déploiement en OSGi.

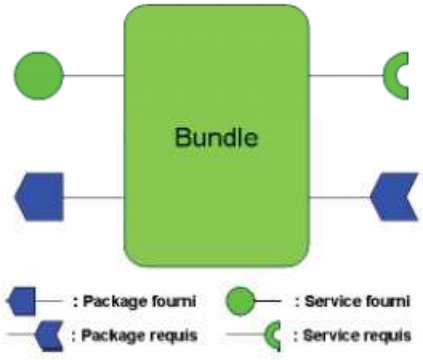


Figure 3.9. Schéma d'un *bundle* OSGi avec ses packages et services.

La figure ci-après, présente le méta-modèle des bundles OSGi du point de vue des entités logicielles et non du point de vue descriptif (c'est-à-dire du point de vue des contrats de services, par exemple).

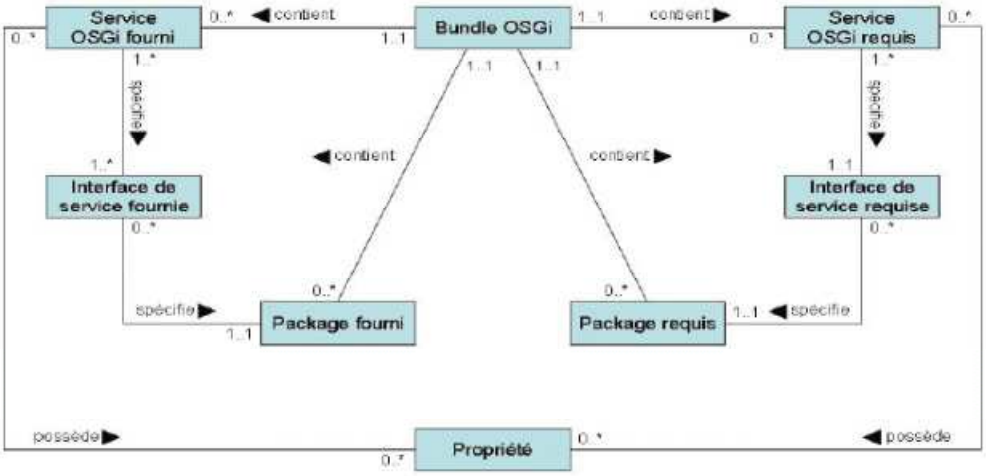


Figure 3.10. Méta-modèle des *Bundles* OSGi (du point de vue entité logicielle).

Le registre de service OSGi peut contenir plusieurs "Services OSGi fournis" implémentant la même "Interface de service fournie". Ainsi, plusieurs bundles OSGi peuvent offrir la même interface par le biais de services qui leur sont propres.

5.3. La plateforme WSRP

5.3.1. Définition de Portail et de Portlet

La plateforme WSRP (Web Service for Remote Portlets)[WSRP 2006] est une spécification définie par OASIS (Organization for the Advancement of Structured Information Standards). Elle permet à un portail, placé du côté du client, d'interroger des portlets par le biais d'un service web. Le portail permet de centraliser sur une unique page appelée portail différents modules appelés portlets.

D'un point de vue technique, le portail est une interface graphique qui offre une porte d'entrée unique sur un large panel de portlets distants par le biais de services web distants. Il fournit un conteneur pour l'agrégation des portlets provenant de diverses applications de services distants; le portail fourni aux portlets un environnement d'exécution. Il les compose sur une page et gère leurs cycles de vie.

L'utilisateur ne connaît pas les différentes applications de services qu'il demande et il ne se soucie pas de comment le contenu ou la fonctionnalité de ses services sont fournis. Le

portail utilise une interface unique qui exécute les portlets à travers une composition. L'utilisateur voit le portail comme un espace de travail personnalisable donnant accès à tous les services web et dont l'agrégation nécessite une seule connexion et une authentification centralisée (Single Sign-On: SSO) ce qui lui permet d'accéder à tous les services auxquels il a autorisation d'accès, en s'étant identifié une seule fois sur le réseau. L'objectif du SSO est ainsi de propager l'information d'authentification aux différents services.

Un portail est aussi vu comme une application Web agissant en tant que passerelle entre les utilisateurs et une gamme de services métiers différents. Les portlets peuvent tous être considérés comme une interface utilisateur à une application et sont définis comme des composants d'interface utilisateur qui sont gérés par un container (le portail). Ils traitent les requêtes et génèrent du contenu dynamique. Par exemple, quand un utilisateur appuie sur un bouton ou lors d'une autre sorte d'événement de l'interface utilisateur, le portlet traite la demande et génère le contenu adéquat qui est ensuite affiché à l'utilisateur. Chacun de ces portlets génère des fragments, qui les agrègent dans le portail pour créer une page complète qui est présentée à l'utilisateur.

Avec le WSRP, l'utilisateur agrège ses services web (portlets) à partir de différents fournisseurs de services distants sans se soucier de l'implémentation et de la fourniture du portlet qui délivre les services web [Link *et al.* 2006].

5.3.2. La plateforme WSRP

OASIS (Organization for the Advancement of Structured Information Standards) a adopté un protocole de service Web d'agrégation de contenu et des applications web interactives à partir de services web distants. Avec WSRP il est possible d'intégrer des portlets à partir de différents fournisseurs distants sans se soucier de l'implémentation et de la fourniture du portlet. Il y a une interface de service web bien définie où le portlet peut être invoqué. Ainsi, par exemple, si une entreprise a besoin d'implémenter un nouveau processus métier avec interaction utilisateur, elle peut d'abord chercher un service d'interface utilisateur adéquat dans l'annuaire de service.

La plateforme WSRP introduit les acteurs suivants:

- **Le producteur**: Les producteurs sont modélisés comme des conteneurs de portlets. Les producteurs sont des services web avec un ensemble commun d'opérations tel que: la description autonome, l'enregistrement et la gestion des portlets. Les producteurs peuvent gérer optionnellement l'enregistrement des consommateurs et

exigent qu'ils se pré-enregistrent avant d'interagir avec les portlets. Un enregistrement établi une relation entre les consommateurs et les producteurs. Le producteur WSRP est un vrai service web.

- **Le consommateur**: C'est lui qui est chargé de rassembler et d'agréger les fragments fournis par les portlets pour les présenter à l'utilisateur final. En général, c'est un portail. Ce dernier est un client web qui invoque le service web WSRP du producteur proposé et fournit un environnement pour les utilisateurs afin d'interagir avec les portlets offerts par une ou plusieurs producteurs WSRP. Les consommateurs sont similaires en nature aux routeurs qui travaillent au nom de l'utilisateur final. Le consommateur acheminera les demandes utilisateurs au producteur approprié.
- **Le portlet WSRP**: une portlet WSRP est un composant d'interface utilisateur« pluggable » qui vit à l'intérieur d'un producteur WSRP et est accessible de manière distante à travers une interface définie par ce producteur. Un WSRP portlet n'est pas accessible directement, mais plutôt doit être accédé à travers son producteur parent. Les portlets sont hébergées par le Producer des Web Services.

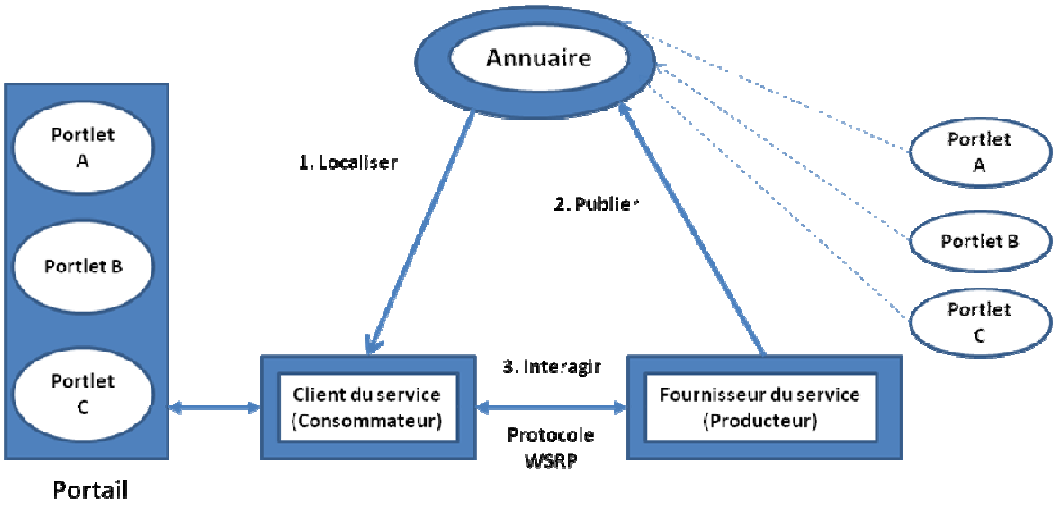


Figure 3.11. Architecture WSRP (paradigme localiser-publier-intégrer).

L'exemple de la figure 3.11 illustre le paradigme localisé – publié – intégré de l'architecture SOA (Service Oriented Architecture) dans l'architecture WSRP. On remarque que le fournisseur de services, le consommateur de services et l'annuaire de service sont appliqués pour les portlets. La figure 3.11 montre un modèle de portail traditionnel où le portail a un conteneur de portlet qui héberge un nombre quelconque de portlets discrètes. Chacun de ces portlets génère des fragments sur le portail pour créer une page complète qui est présentée à l'utilisateur. La spécification Java Portlet (JSR-168) [JSR168], approuvé en

Octobre de 2003, définit une API standard pour les plates-formes J2EE portail. L'objectif de la JSR-168 est de fournir un ensemble de normes afin que les portlets conformes puissent être déployé sur n'importe quel portail qui supporte la spécification.

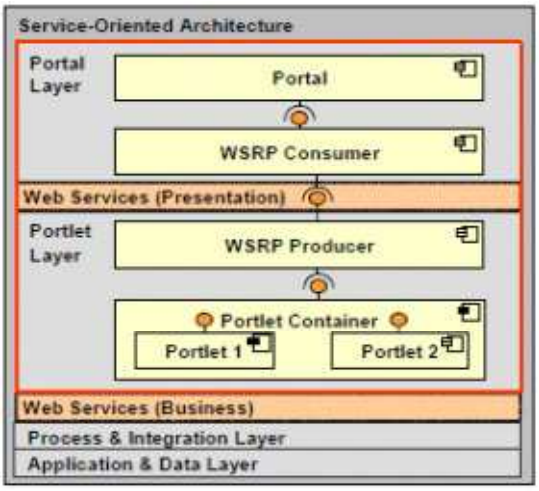


Figure 3.12. Architecture WSRP [Link *et al.*2006]

Le scénario d'un WSRP se déroule de la manière suivante:

(i) Le consommateur découvre un producteur, (ii) la relation entre le consommateur et le producteur est établie, (iii) le consommateur apprend toutes les capacités du producteur, (iv) l'utilisateur final établi une relation avec le consommateur, (v) le consommateur agrège les portlets pour les utilisateurs, (vi) l'utilisateur final envoie une page de requête au consommateur, (vii) le consommateur sollicite des informations du producteur, (viii) le producteur fournit au consommateur les services métiers et le service d'interface, (ix) l'utilisateur final voit les services web agrégés sur une page.

WSRP utilise WSDL pour décrire les interfaces, requiert au minimum les liaisons SOAP pour fournir l'ensemble des services autorisées, XML Schéma pour définir la structure des messages passées et XML pour porter les messages entre les services et les clients.

5.4. La plateforme Mashup

Un mashup [Wikipedia, Mashup 2006] est une application composite qui en toute transparence combine le contenu d'un ou de plusieurs services assemblés sur une même page Web dans le navigateur. Le contenu utilisé est généralement fourni par une interface publique de services mise à disposition par un fournisseur tiers. Les mashups incorporent de manière simple la conception dirigée utilisateur et le développement de nouvelles applications composites.

Selon Maness [2006], Les applications *mashup* sont des applications hybrides dans lesquelles au moins deux services ou technologies sont combinés dans le but de créer un nouveau service. Ce type d'application inclut les mêmes problématiques que les compositions de services Web: le partage, la sélection, la réutilisation, et l'intégration des services [Benslimane *et al.* 2008]. Les applications *Mobile Web 2.0* – un autre type d'application Web 2.0 – regroupent les services Web 2.0 dont l'accès et l'utilisation sont réalisés à travers des dispositifs mobiles [Wahlster *et al.* 2006]. L'utilisation nomade de systèmes peut entraîner une prise en compte, lors de la conception de ces applications, du *contexte* afin de proposer aux utilisateurs des informations et services adaptés. Le contexte (par exemple, la localisation de l'utilisateur, les dispositifs qu'il a à sa disposition) permet de mettre en œuvre des applications adaptées au contexte et ainsi valorise ce type de logiciel.

La plateforme mashup a pour but de réutiliser des services et sous-entend rapidité et facilité. Le mashup impulse le développement Web en permettant à toute personne de combiner les données existantes provenant de services différents comme eBay, Amazon, Google, Windows Live et Yahoo dans des moyens novateurs et donc supporte une capacité de structuration pour l'intégration des contenus et applications Web. Le mashup utilise des technologies XML (Extensible Markup Language), AJAX (Asynchronous JavaScript And XML) pour composer les différents contenus de services.

Les *mashups* reposent sur des API (Application Programming Interface), utilisant la technologie des web services, qui permettent de créer de nouveaux services "composites" en exploitant et en agençant de manière originale des programmes, des services et des contenus créés par d'autres.

Plusieurs exemples, tels Google (pour sa base de données ou ses cartes),eBay (pour créer sur des sites personnels des catalogues d'objets en vente) ou FedEx (qui permet à un commerçant de présenter sur son propre web l'état d'avancement d'une livraison qu'il a sous-traitée au logisticien),Youtube, Flickr et Amazon, offrent gratuitement des API. Sur le plan architectural, l'application comprend trois acteurs principaux qui sont les fournisseurs de services/API, l'annuaire de services qui est l'hébergeur et le consommateur qui est le navigateur web. Ces trois acteurs sont évidemment compatibles avec la notion SOA. L'architecture est représentée à la figure 3.13. Le fournisseur de service/API présente souvent son contenu par le biais de protocole web tels que REST (REpresentational State Transfer), web services et RSS/Atometles met à la disposition des clients. Quant au consommateur de service, c'est l'interface utilisateur ″personnalisée″ où est organisée l'application composite qui correspond au besoin de l'utilisateur.

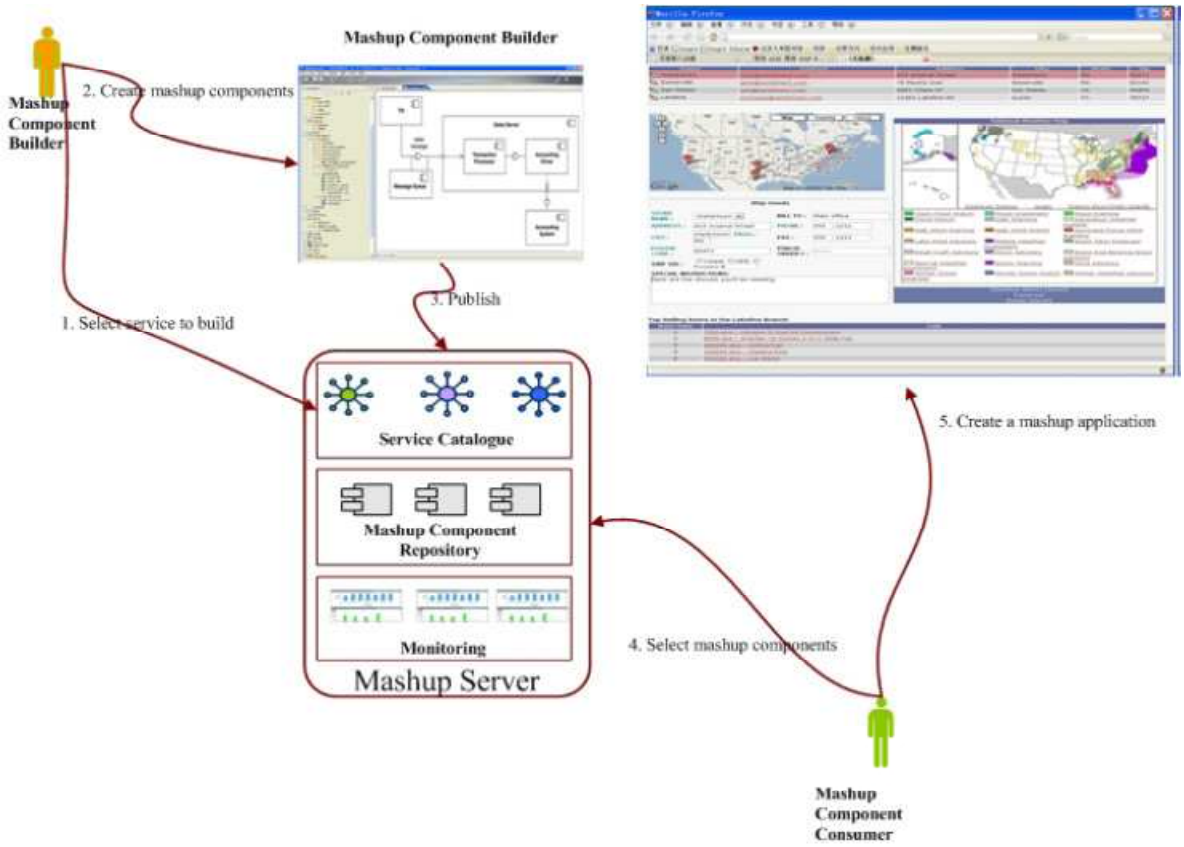


Figure 3.13. L'architecture Mashup

5.5. Comparaison des plateformes

MeTSI conduit à des services logiciels interactifs implémentés sur trois plateformes interactives de services qui sont WSRP, OSGi et Mashup. Ces trois plateformes réutilisent les services mais de manière différente et au travers des stratégies de distribution qui sont propres à chacune d'elles.

	WSRP	**OSGi**	**Mashup**
Stratégie de Distribution	Invocation à distance	Téléchargement	Local
Acteur d'implémentation du service interactif	Fournisseur de services	Fournisseur de service	Consommateur de service
Développement de la composition de services	Fort (traditionnel)	Fort (traditionnel)	Facile
Utilisateur de la composition de services	Expert (centré développeur)	Expert (centré développeur)	Non expert (Centré consommateur)
Possibilité d'agrégationdes services.	Oui (à travers un portail)	Non	Non
Lieu d'agrégation des services	Conteneur de portlet	Machine virtuelle	Conteneur de widget

Tableau 3.1. Tableau comparatif des plateformes interactives d'implémentation de la méthodologie orientée services intentionnels

Les stratégies de distribution peuvent être soit une invocation à distance, soit un téléchargement de services à partir de fournisseurs distants exécutés de manière local, soit une conception et une exécution des services faites de manière locale chez le client. La stratégie *distante*, comme c'est le cas de la plateforme WSRP, est celle des services web. Qu'ils soient services métiers ou services d'interface utilisateur, ils sont publiés par des fournisseurs distants et localisés par les clients puis exécutés. La stratégie *par téléchargement*, comme c'est le cas pour la plateforme OSGi, permet d'avoir les services web localement à partir de téléchargement chez des fournisseurs de services distant. L'exécution de tels services se fait localement chez le client alors que la provenance a été faite de manière distante. La dernière stratégie est celle de la plateforme Mashup donc la fourniture et l'exécution du service interactif est faite de manière locale. A partir du point précédent, nous constatons que les différentes plateformes ne sont pas tous conçu par les mêmes acteurs. Tandis que dans les plateformes WSRP et l'OSGi les services sont conçuschez différents fournisseurs de services et localisés par les clients pour être soit exécutés à distance ou téléchargés localement pour être exécutés, la plateforme Mashup, elle, conçoit son service interactif sur le navigateur du client de manière locale pour exécuter les services métiers invoqués de manière distante. La réutilisation de services par la composition au sein de ces trois plateformes interactives demeure l'un des éléments important de la méthode MeTSI. Cependant, le développement de cette composition peut varier d'une plateforme à une autre. Ainsi, mashup a pour but de réutiliser des services de manière rapide et facile. Comparés aux technologies de composition traditionnelles ou ″centrée développeur″ que sont l'OSGi et la WSRP, les mashups fournissent une solution flexible et facile à utiliser pour la composition de services. Il permet aux utilisateurs de composer librement leurs services.Ainsi, on pourrait dire que la composition des deux plateformes interactives que sont le WSRP et l'OSGi sont centrées sur le développeur alors que celle de Mashup est centrée sur l'utilisateur. Enfin, le portail du WSRP permet d'agréger plusieurs portlets (fragments d'interface graphique) et permettre à l'utilisateur de composer librement des services interactifs.

6. Conclusion

L'approche MeTSI décrite dans ce chapitre nous a permis de proposer un cadre général de développement flexible (prenant en compte les concepts de services orientés buts) et cohérent (en conservant la sémantique des concepts orientées buts tout au long du cycle de développement) se basant sur une combinaison de l'approche orientée services intentionnels et de l'approche dirigée par les modèles.

Dans la première section, nous avons décrit notre approche de services intentionnels qui permet la représentation des services à un niveau intentionnel. Il offre une meilleure homogénéité dans l'expression des besoins des clients et les services logiciels. Il définit chaque service intentionnel en tant que brique de construction d'applications visible au travers de son interface qui apporte les connaissances situationnelle et intentionnelle. Chaque service intentionnel s'applique dans une situation particulière pour réaliser une intention particulière.

Dans la seconde partie de ce chapitre, nous avons décrit le processus de transformation d'un service intentionnel. MeTSI applique les principes de l'ingénierie dirigée par les modèles.

Elle opère par transformations successives de façon à séparer les sujets d'intérêt pour apporter de la flexibilité et permettre l'adaptation, en particulier, à différentes plateformes d'implémentation. Elle réutilise des services web déjà réalisés lorsque c'est possible. Dans la dernière partie de ce chapitre, nous avons étudié les différentes plateformes d'implémentation de services interactifs. Nous avons choisi ces trois plateformes car elles représentent un panel représentatif de l'ensemble des solutions disponibles à ce jour.

CHAPITRE 4 : DES BUTS UTILISATEURS AU SERVICE OPERATIONNEL : DEFINITION D'UNE APPROCHE DE DEVELOPPEMENT SEMANTIQUE DIRIGEE PAR LES MODELES

1. Introduction

Nous introduisons dans cette thèse un processus de transformation sémantique basé sur deux méta-modèles : le méta-modèle intentionnel de service - MIS et le méta-modèle opérationnel de service - MOS. Dans ce processus, le méta-modèle MOS représente le système attendu au niveau opérationnel. Les modèles sont représentés en utilisant les langages de modélisation dont la syntaxe est spécifiée au moyen d'un méta-modèle. Ce chapitre introduit donc le processus proposé pour lier la modélisation orientée service intentionnelle et la modélisation MOS au moyen d'un processus de développement dirigé par les modèles (MDD) basé sur les directives de transformation sémantiques.

2. Une approche de développement dirigée par les modèles pour la génération du modèle MOS à partir du modèle MIS

L'approche de développement dirigée par les modèles permet de passer d'un modèle à un autre modèle, ou d'un modèle à du code, par un processus de transformation. Ce processus est formulé au moyen d'un processus générique basé sur des standards ou des technologies, dans le but de faciliter son implémentation. Tout au long de ce chapitre, nous montrons comment dériver de manière sémantique, et à partir d'un processus DDM (Développement Dirigée par les Modèles), la modélisation MOS. Le processus de transformation est élaboré en prenant en compte les directives de transformation sémantique qui sont, dans notre cas, basées sur les scénarios. Ceux-ci fournissent un moyen de description des perspectives utilisateurs, ainsi que

de leur façon d'utiliser les services. La définition du méta-modèle de scénario, ainsi que celles des méta-modèles d'entrée et de sortie se font avec la spécification EMOF (Essentiel MOF) défini selon le standard MOF [EMOF 2004]. Pour la spécification des méta-modèles invoqués, nous utilsons l'outil Eclipse UML2Tool [UML2Tool], lequel permet de fournir la génération automatique des méta-modèles EMF à partir des méta-modèles UML2 définis. EMF est le Framework Modeling Eclipse basé sur la spécification EMOF.

La Figure 4.1 présente le schéma de liaison entre le modèle intentionnel et le modèle opérationnel. Dans ce schéma, les méta-modèles MIS, MOS et celui basé sur les directives de transformation utilisent le langage générique EMOF. La description de ces méta-modèles dans un langage générique de haut niveau de spécification permet d'avoir des instances spécifiques conformes à ces langages.

Dans les sous-sections suivantes, nous définissons le processus de transformation de modèles, ainsi que les méta-modèles décrits dans les différents langages.

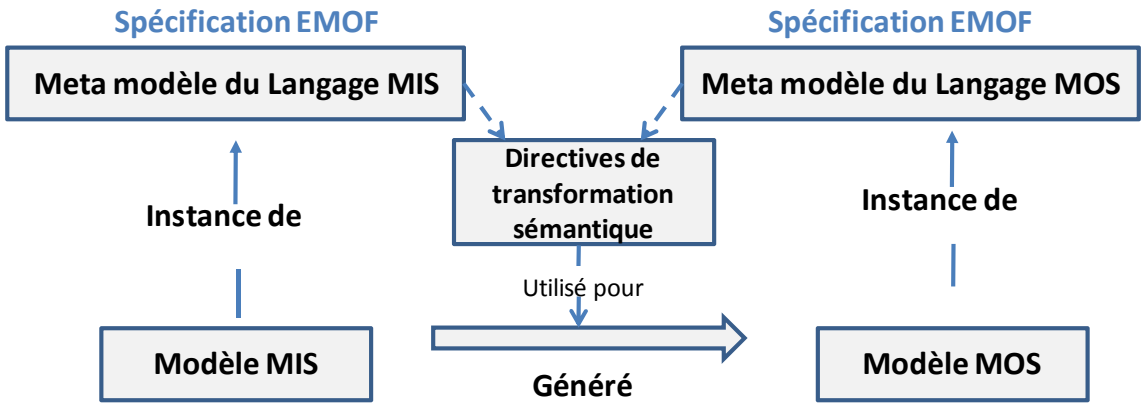


Figure 4.1. Schéma de liaison entre le modèle intentionnel et le modèle opérationnel.

Comme cela a été décrit au chapitre 3, les services intentionnels du modèle MIS offrent le moyen de satisfaire les buts de haut niveau tels que les buts stratégiques. Ceci est réalisé en décomposant le but stratégique en sous buts, qui eux-mêmes peuvent nécessiter une décomposition etc., jusqu'à atteindre des buts opérationnalisables.

La réalisation du but stratégique d'un client se fait en le décomposant, ou en l'affinant, en sous-buts, mais aussi en distribuant ces sous-buts sur différents clients. De cette manière, la réalisation d'un but est considérée comme un processus intentionnel faisant coopérer plusieurs clients.

2.1. Une Vue générale du méta-modèle du langage MOS.

Le méta-modèle du langage MOS est le méta-modèle de sortie une fois le processus de transformation appliqué. Ce méta-modèle est constitué d'un service opérationnel issu d'un service atomique du méta-modèle du langage MIS. Le méta-modèle MIS a été présenté dans le chapitre 3 précédent.

Le méta-modèle MIS classe les services intentionnels en deux catégories : agrégat et atomique. Un service intentionnel de type agrégat est utilisé lorsque le but est affiné par un ou plusieurs sous-buts de services intentionnels. Par opposition, le service intentionnel atomique est associé à un but qui n'est pas affinable en sous-buts mais qui est directement opérationnalisable par l'exécution d'un ou plusieurs services opérationnels. La spécificité du service opérationnel est donc qu'il est ainsi dirigé par les buts.

D'un point de vue architectural, le service opérationnel met en œuvre deux types d'éléments logiciels : les éléments d'avant plan - ou service opérationnel centré utilisateur - et les éléments d'arrière plan - ou service opérationnel centrée métier. Les éléments d'avant plan représentent les services gérant les interactions avec l'utilisateur à l'aide des modèles basés sur les interfaces graphiques, alors que les éléments d'arrière plan sont les services gérant le métier et invoquées à partir du service utilisateur.
Dans le cadre de l'ingénierie des architectures orientées services, le service métier peut être réalisé par la réutilisation, la coordination et la recherche de plusieurs services web. Par conséquent, le modèle MOS introduit le concept de *service web ou service* pour représenter l'unité logicielle qui est fournie par une organisation et le concept de *service métier* pour représenter l'application métier construite par composition de services web recherchés et invoqués par le *service d'interaction utilisateur*.

De ce fait, le service opérationnel du méta-modèle MOS est réalisé par l'assemblage de deux types d'éléments : le service d'interface utilisateur et le service métier. Le service métier est constitué de la coordination, de la recherche et de l'invocation de services web.

Le modèle MOS est une représentation des services opérationnels suivant une forme canonique. Les services opérationnels remplissent les actions nécessaires à l'accomplissement du but du service intentionnel telles que la gestion de la logique métier que le service implémente sur la couche opérationnelle, l'administration de la coordination des différents agents fournisseurs de services, la recherche dynamique et l'invocation de service web. Le

modèle MOS est donc considéré comme une opérationnalisation du service intentionnel atomique.

Les services opérationnels du modèle MOS sont décrits indépendamment de la plateforme d'implémentation choisie, elle est de type PIM dans la spécification MDE (Model-Driven Enginieering).

La figure ci-dessous présente le méta-modèle du langage MOS.

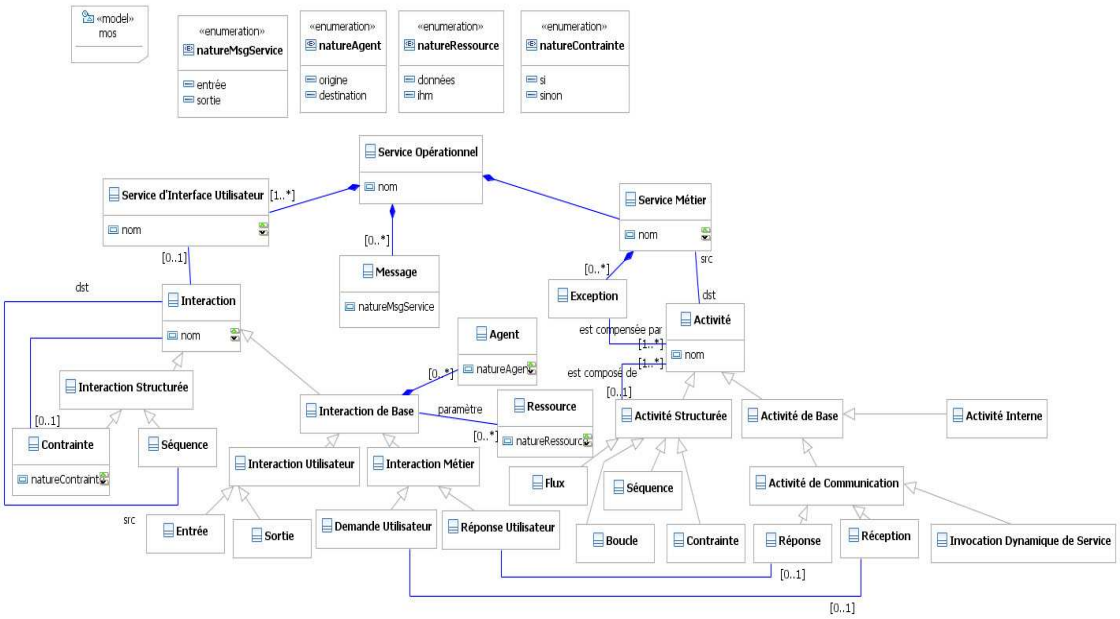


Figure 4.2. Le méta-modèle opérationnel du langage MOS

Le service opérationnel est le moyen d'opérationnaliser le service intentionnel atomique.

L'exécution du service opérationnel permet d'interagir avec le service d'interface utilisateur, qui délègue au service métier la réalisation d'une transaction métier complexe et distribuée, qui, à son tour, délègue aux services web la réalisation de transactions métier. A la fin de cette chaîne de délégation, la réalisation du but se traduit par le fait que le service d'interface délivre le résultat du service opérationnel.

Le service opérationnel est caractérisé par les éléments suivants :

- un nom permettant de l'identifier de manière unique,
- un message en entrée : une structure de données passée en entrée pour adapter le comportement du service opérationnel au contexte d'utilisation,

- un message en sortie : une structure de données représentant le résultat produit et permettant de constater la délivrance du service à l'utilisateur.

Par exemple, le service intentionnel atomique S $_{Effectuer\ Réservation}$ est concrétisé par le service Opérationnel (SO) de nom SO $_{Effectuer\ Réservation}$ avec le *nom du demandeur* comme message d'entrée et la *réservation de vol & chambres payée* comme message de sortie.

2.1.1. Le méta-modèle du Service d'Interface Utilisateur du service opérationnel MOS

Le méta-modèle du service d'interface utilisateur représente la spécification nécessaire au développement de l'application d'avant plan. Ceci peut être fait à l'aide de techniques de développement d'interface web.

Dans le cadre du service opérationnel, nous ne nous intéressons pas à la structure graphique de l'interface web mais plutôt aux interactions que l'application d'avant plan doit mettre en œuvre pour obtenir le résultat attendu.

Un service d'interaction utilisateur est défini par un nom et un ensemble ordonné d'interactions que le service d'interface utilisateur doit mettre en œuvre pour gérer les interactions avec l'application d'arrière plan.

La Figure 4.3 est un extrait de la Figure 4.2 montrant les éléments du modèle du service d'interface utilisateur.

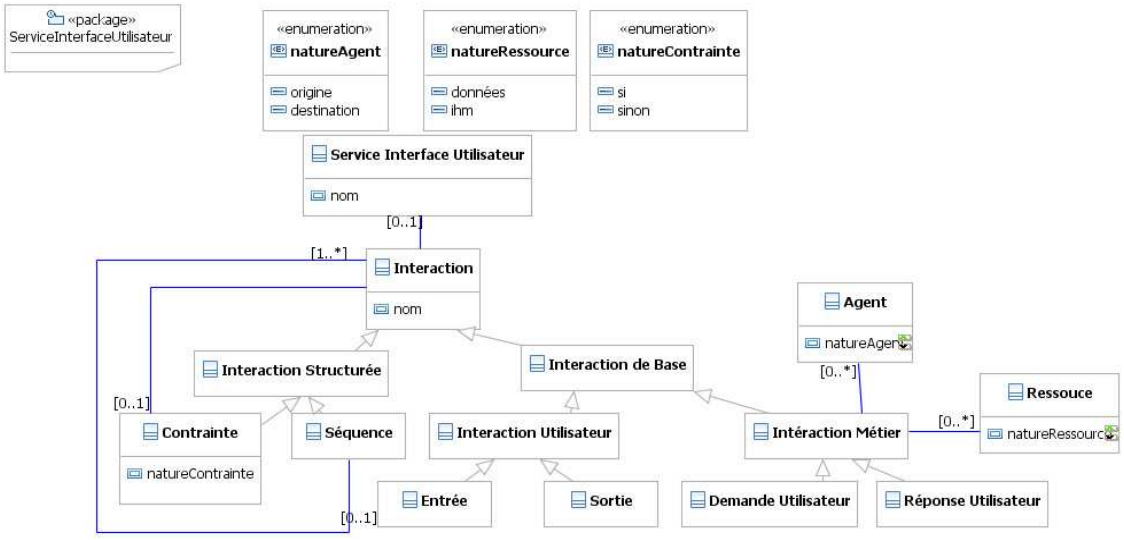


Figure 4.3. Le méta-modèle du Service d'Interface Utilisateur

Le service d'interface utilisateur a comme objectif la gestion du dialogue avec l'utilisateur pour que le service logiciel l'aide à atteindre son but. Le dialogue est pris en charge dans le service d'interface utilisateur à l'aide du concept d'interaction.

Le service d'interface utilisateur propose deux types d'interactions : *l'interaction de base* et *l'interaction structurée*.

L'interaction de base se traduit par le fait qu'un *agent* communique à un autre *agent* des données que l'on appelle des *ressources.* Elles peuvent être de deux types : les ressources en termes de données ou les ressources en terme d'interfaces utilisateur (permettant à l'utilisateur de saisir sa requête, proposant une interaction structurée permettant de modéliser l'enchaînement des interactions de base nécessaires). Une interaction de base est spécialisée en deux sous types : l'*interaction utilisateur* et l'*interaction métier.*

Une *interaction utilisateur* décrit les données échangées entre l'agent utilisateur et le service d'interface utilisateur. Par contre, une *interaction métier* décrit les communications mises en place entre le service d'interface et le service de coordination. Par exemple, l'interaction *FormuleDemandeRéservationVol&Chambres* représente le fait que 'le client formule une demande de réservation de vol et de chambres d'hôtel à partir de l'agence de voyage'.

L'*interaction utilisateur* décrit les communications de l'utilisateur vers l'interface mais aussi celles allant de l'interface vers l'utilisateur. Le premier type constitue *les entrées* et permettent de spécifier les données que l'utilisateur doit fournir en entrée. Alors que le deuxième type constitue *les sorties* et modélise les informations que l'interface fournit à l'utilisateur.

L'interaction *FormuleDemandeRéservationVol&Chambres* est un exemple d'*interaction utilisateur* de type entrée. Par contre, l'interaction correspondant au fait que *L'agence de voyage confirme la réservation au client* représente une *interaction utilisateur* de type sortie.

Une *interaction métier* spécifie le principe de délégation mis en place entre l'application d'avant plan et l'application d'arrière plan. Le service d'interface délègue la partie métier du service opérationnel au *Service Métier*. Une délégation se matérialise en termes de communication par deux interactions métier : la première est une *demande utilisateur* et la seconde est une *réponse utilisateur.*

La *demande utilisateur* représente l'interaction métier initiée par l'interface pour invoquer un service métier nécessitant une réponse. L'interaction *EnvoieDemandeRéservationVol&Hotel* correspond au fait que l'interface envoie une demande de réservation de vol et de chambres au

coordonnateur. C'est un exemple d'interaction métier entre le service d'interface et le service de coordination de l'agence de voyage.

La *réponse utilisateur* est une interaction métier qui représente le résultat du service métier invoqué à partir d'une demande utilisateur.

Une interaction structurée est spécialisée en : *contrainte* et *séquence*.

Une *contrainte* représente un branchement conditionnel et adapte l'ensemble des interactions qui la suivent selon l'évaluation d'une condition. La *contrainte* décrit les interactions alternatives (Si-Alors–Sinon) qui permettent de décrire plusieurs comportements possibles dans le même service. Une *contrainte* est décrite par une condition et une interaction représentant le cas où la condition est vraie (branche si) et l'interaction à exécuter lorsque la condition n'est pas satisfaite (branche sinon).

Une s*équence* permet de décrire l'enchaînement séquentiel d'un ensemble d'interactions. Les enchaînements séquentiels et les enchaînements conditionnels caractérisent le comportement global du service d'interface utilisateur.

Nous décrirons dans la section dédiée au *service métier*, les relations qui existent entre les interactions métier du service d'interface et le service métier.

Le modèle de service d'interface utilisateur permet donc de décrire le contenu de l'application web comme un ensemble ordonné d'interactions utilisateur et d'interactions métier :

- Les *interactions utilisateur* décrivent le contenu informationnel de l'interface graphique web de l'application d'avant plan. Une conception graphique des pages web peut ensuite en être dérivée.
- Les *interactions métier* permettent de décrire les délégations métier mises en œuvre à partir du service d'interface. La description des délégations métier permet de dériver la conception des composants applicatifs web invoqués à partir des pages web.

2.1.2. Le méta-modèle du Service Métier du service opérationnel MOS

Le Service Métier permet, d'une part, de coordonner l'ensemble des services web à travers un ensemble d'activités ordonnées et, d'autre part, de rendre le Service d'Interface Utilisateur indépendant des services web mis en œuvre. Ceci permet de développer des applications d'avant plan indépendamment des applications métier.

Un Service Métier repose sur la définition des règles de travail. Cette définition est généralement basée sur un modèle spécifique qui décrit l'enchaînement d'activités à accomplir pour la réalisation d'un objectif et son exécution assure la coordination des agents. La formalisation des règles oblige les partenaires à suivre strictement les conventions établies.
Le Service Métier permet l'exécution d'un processus déclenché par l'arrivée d'une action de communication. Il évalue les données en entrée et appelle le service web concerné. Ce dernier renvoie son résultat au Service Métier.
Le rôle d'intermédiaire dans la communication attribue au service métier la fonction de synchronisation et de coordination qui sont nécessaires entre les services web.
Le Service Métier permet également de tracer la séquence de messages pouvant impliquer plusieurs autres services. Il spécifie l'ordre d'exécution des messages échangés entre les services et le traitement des fautes et exceptions spécifiant le comportement dans le cas d'erreurs ou d'exceptions.
Enfin, le Service Métier réutilise les services Web en permettant la recherche de services au travers de requêtes décrites en XQUERY.
La Figure 4.4 est un extrait du méta-modèle MOS et montre les différents éléments composant le Service Métier.

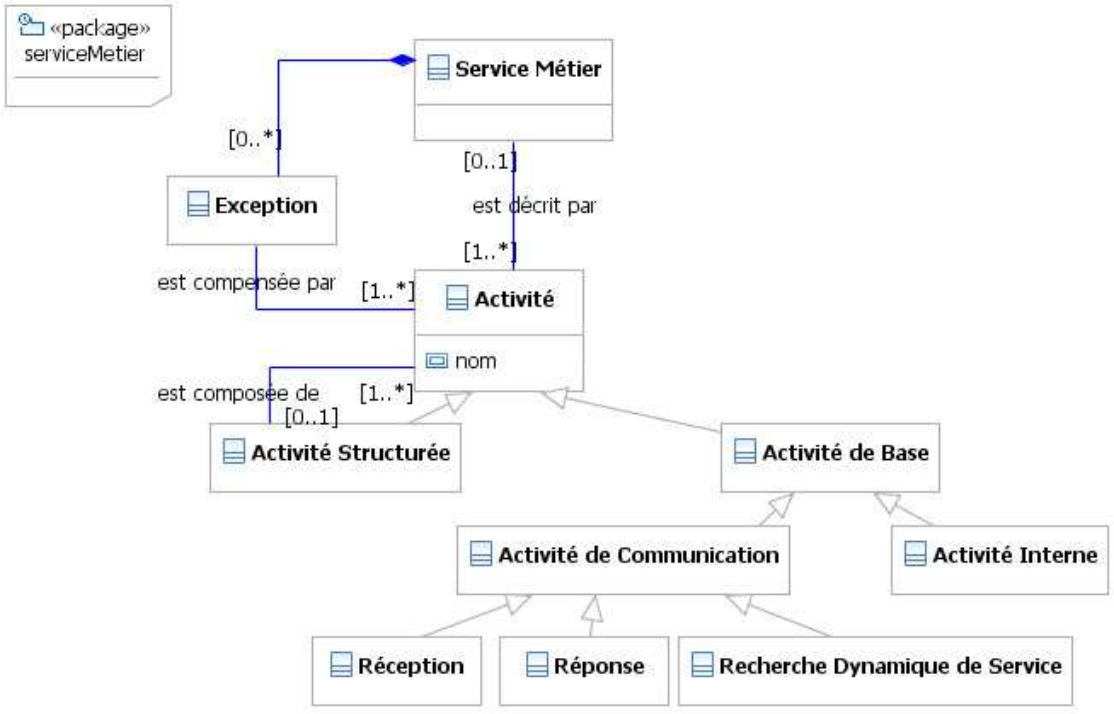


Figure 4.4. Le méta-modèle du Service Métier

La structure d'un Service Métier est relativement complexe par rapport au service web. [Alanso 2004].
Nous présentons les différents concepts formant le service de coordination dans les sections suivantes.

a. La notion d'activité

Un service métier est défini comme un ensemble d'activités échangées entre plusieurs agents. Les activités peuvent être de deux types : de *base* ou *structurée*. Les *activités de base* sont des opérations réalisées par un service web. Les *activités structurées* sont organisées hiérarchiquement.
Dans le modèle du Service Métier, les activités structurées sont les *activités complexes* alors que les activités de base sont les activités atomiques. La Figure 4.4 montre que ces deux notions sont généralisées par la notion d'activité. Cette généralisation est exclusive.
D'un autre côté, une activité fait partie d'une description de Service Métier ou d'une activité structurée. Par conséquent, les activités ne sont pas partagées, ni entre plusieurs Services Métier, ni même entre plusieurs activités structurées, au sein d'un même Service Métier (d'où la cardinalité 0..1 de *Activité* vers *Service Métier*).
L'association *est composée de* entre *Activité structurée* et *Activité* à la Figure 4.4 indique qu'une *activité structurée* est composée d'au moins une activité (cardinalité 1..*). De manière récursive, les activités peuvent à leur tour être composées d'activités structurées.
L'association *est décrit par* entre *Service Métier* et *Activité*, montre qu'un Service Métier est en général décrit par un ensemble d'activités (la cardinalité 1..*).
Les deux sous sections suivantes détaillent la notion d'activité de base et la notions d'activité structurée.

b. La notion d'activité de base

Une activité de base peut avoir différents types tels que demande de service, fourniture de service, demande d'information, fourniture d'information, communication d'une ressource physique, etc. Les activités de base se divisent en deux catégories : les *activités de communication* et les *activités internes.*
Une *activité de communication* s'établit entre le service de coordination et les services métier ou le service d'interface utilisateur. Elle est spécialisée en : *Recherche Dynamique de service web, Réception* et *Réponse*.

Une *Recherche Dynamique de service* correspond à la recherche d'un service par une requête écrite en XQUERY et qui recherche les services web dans un annuaire de service UDDI. Une fois le service web trouvé, il est invoqué. Une *réception* correspond à l'attente d'un message provenant d'un service d'interface utilisateur et une *réponse* est utilisée pour répondre au service d'interface utilisateur. Une activité interne correspond à toutes les activités qui sont exécutées de manière interne par le Service Métier. Ses activités peuvent correspondre à l'enregistrement de la requête de l'utilisateur. Chaque activité de base peut manipuler une ou plusieurs *ressources*.

c. La notion d'activité structurée

Les activités structurées permettent de définir l'ordonnancement des activités de base. Les activités structurées sont classées en quatre types : s*équence, flux, boucle* et *contrainte*. Ces quatre types sont modélisés à la Figure 4.4 et sont détaillés dans les sections suivantes.

- **La séquence**

Une séquence est composée de deux activités, la deuxième activité se déroulant après la première. L'activité structurée *« Réception d'une demande de réservation de vol et de chambres d'hôtel »* puis la transformation du message *« demande de réservation de vol et d'hôtel »* en message *«demande de vol »* donne un exemple de deux activités en séquence. Après l'activité de réception d'une « demande de réservation de vol et de chambres d'hôtel », la deuxième activité permet de dériver le message « demande de réservation de vol » à partir du message « demande de réservation de vol et de chambres d'hôtel ».

Une séquence doit être composée de deux activités au maximum. En conséquence, le séquencement de plusieurs activités de base est exprimé par une composition récursive de plusieurs séquences d'activités. Prenons l'exemple suivant:

« Réception d'une demande de réservation de vol et de chambres d'hôtel » puis la transformation du message *« demande de réservation de vol et d'hôtel »* par le message *« demande de vol ».* **Invocation dynamique de services web** *« demande de disponibilités pour un vol de la compagnie aérienne.*

Cette séquence est composée de manière récursive de l'activité : *« Réception d'une demande de réservation de vol et de chambres d'hôtel »* qui est en séquence avec l'activité de flux qui est composée de deux activités de base : L'affectation transformant le message *« demande de réservation de vol et d'hôtel »* par le message *« demande de vol »* ***et*** **Invocation dynamique de services web** *« demande de disponibilités pour un vol »* de la compagnie aérienne.

Le même raisonnement s'applique aux quatre types d'activités structurées. Ainsi, une séquence peut être composée de contraintes, de flux, de boucles ou de séquences d'activités.

- **Le flux**

Contrairement à une séquence, il n'y a aucun ordre spécifique dans un flux de deux activités. Un flux de deux activités ne commence et ne se termine pas nécessairement au même moment. Du point de vue utilisateur, les activités d'un flux se déroulent simultanément.

Prenons l'exemple suivant: **Invocation dynamique de services web** *« demande de disponibilités pour un vol »* de la compagnie aérienne et appel du service *« demande de disponibilités de chambres »* de l'hôtel. Ces deux interactions n'ont pas d'ordre spécifique et ne commencent ni ne se terminent au même moment. Cependant, du point de vue de l'utilisateur, elles sont concurrentes et simultanées. Un flux est composé de plusieurs activités pouvant être des activités structurées.

- **La contrainte**

Une contrainte décrit les conditions nécessaires au déroulement de l'ensemble des activités qui suivent la contrainte. Une contrainte permet de décrire plusieurs comportements possibles du même service de coordination. Prenons l'exemple du service de coordination : « *Obtenir une réservation de vol et de chambres d'hôte ».* Ce service comprend l'expression : *'si la période de validité est valide'*. Ce service de coordination inclut donc une contrainte correspondant à la condition : *'la période de validité est valide'*. Le reste du Service Métier décrit le comportement correspondant au cas où la période serait valide et celle où elle ne le serait pas.

- **La boucle**

La boucle permet d'exprimer une répétition d'activités structurées. Lorsque l'agence de voyage a un partenariat avec plusieurs compagnies aériennes, **Recherche Dynamique et appel du service** de disponibilités se répète pour chacune des compagnies aériennes de la liste de l'agence de voyage. L'expression suivante : *Pour chaque compagnie aérienne de la liste, appel du service «demande de disponibilités pour un vol »* de la compagnie aérienne est un exemple de boucle.

Comme les autres types d'activités, une boucle peut être composée d'une combinaison d'activités. Dans l'exemple ci-dessus, la boucle est composée de trois activités de base: *(a)* **Invocation dynamique de services web** *« demande de disponibilités pour un vol » de la compagnie aérienne, (b)* ***réponse du service*** *« demande de disponibilités pour un vol » et (c)* ***envoi du message*** *« Résultat des disponibilités et des prix ».*

3. Le processus générique pour lier la modélisation des services intentionnels et la modélisation des services opérationnels

La première étape dans la transformation des modèles est d'identifier les concepts des approches de modélisation en entrée - dans notre cas l'approche de modélisation par services intentionnels (MIS). La modélisation MIS est pertinente pour la génération des concepts des approches de modélisation de services opérationnels. L'identification des concepts pertinents est exécutée au travers des méta-modèles des langages de modélisation invoqués. Ces modélisations sont conformes à EMOF [EMOF 2004]. Ainsi, l'ensemble des directives de transformation nécessaires pour obtenir les concepts de MOS correspondant doivent être définies à partir des concepts de MIS.

Nous avons défini au cours du chapitre 3 l'approche de modélisation MIS décrite dans le langage EMOF, puis nous avons défini dans la section 2 la modélisation Produit MIS dans la spécification EMOF. Nous définissons, dans la suite de ce chapitre, les directives de transformation qui permettent de passer des intentions aux interactions et des services web, c'est à dire des services intentionnels, aux services opérationnels.

3.1. Description du scénario

Les directives de transformation sont définies dans une approche MDD qui sépare la logique intentionnelle de la logique opérationnelle, ce qui permet une génération sémantique à partir de la représentation intentionnelle des systèmes opérationnels.

L'objectif de cette étape est de découvrir les éléments constituant le service opérationnel de MOS à partir des scénarios (scénario de base et scénarios alternatifs de succès ou d'échecs) qui décrivent les buts et intentions de l'utilisateur. La construction du modèle MOS par analyse des scénarios est réalisée en utilisant un ensemble de directives. L'instance du modèle MOS générée est conforme au méta-modèle MOS présenté à la section précédente (Présentation des concepts du modèle MOS). Ce méta-modèle est décrit dans la spécification EMOF et toutes les instances générées doivent être conformes à ce méta-modèle.
Les directives de transformation utilisées pour générer la modélisation MOS sont structurées en trois étapes prenant en compte les scénarios alternatifs de succès et d'échec.

3.2. Le modèle de scénario

3.2.1. La notion de scénario

La notion de scénario a été introduite dans les approches dirigées par les besoins pour fournir un moyen de description des perspectives utilisateurs et de leur façon d'utiliser l'application. Tout cela doit être décrit d'une manière compréhensible par tous les participants, pour faciliter la communication entre les différents types d'utilisateurs et les concepteurs de l'application.

Par analogie à [Tawbi 2001], nous notons que le couplage de la direction service – scénario permet de :

1. Concrétiser le but d'un acteur par la description du comportement d'usage du SI. Cette approche de concrétisation des buts par des scénarios permet d'éviter d'identifier des buts non réalistes.
2. Découvrir les scénarios alternatifs de succès ou d'échec d'un but aboutissant à une description complète du comportement à implémenter dans les services logiciels.
3. Associer la dimension *Pourquoi* du but aux dimensions *Qui*, *Quoi* et *Comment* des scénarios.

La rédaction des scénarios permet aux utilisateurs d'exprimer leurs connaissances sur l'utilisation du système. Le modèle de scénario définit son contenu conceptuel et présente les scénarios comme des narrations textuelles. Le langage utilisé pour exprimer les scénarios est par conséquent un sous-ensemble du langage naturel qui recouvre ces concepts modélisés par une structure linguistique spécifique. La structure linguistique de scénario a été définie par [Ben Achour 1999] et étendue dans [Tawbi 2001].

Ainsi, nous avons choisi d'utiliser des scénarios écrits sous forme de textes en langage naturel. Ce choix est motivé par le fait que les utilisateurs sont plus familiers des scénarios écrits de cette manière [Rolland *et al.* 1998]. Tout scénario encapsule de manière non formelle une partie des connaissances que le système a du système.

3.2.2. Le méta modèle de scénario

La Figure 4.4 présente le méta modèle de scénario en utilisant les notations du diagramme de classes UML. Les concepts sont détaillés dans les sections suivantes. Le méta-modèle ci-dessous est défini avec la notation EMOF.

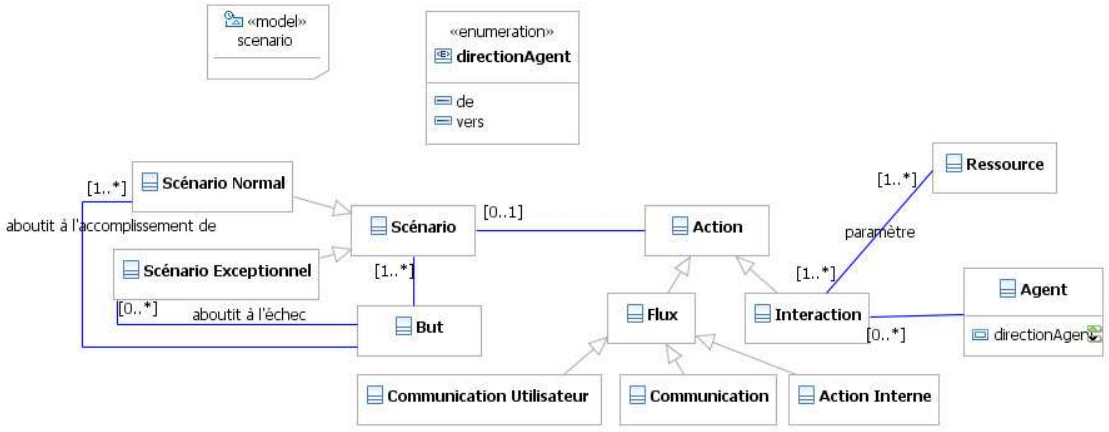


Figure 4.4. Le méta-modèle de Scénario

Le méta-modèle de scénario présenté à la Figure 4.4 inclus les méta-classes essentielles pour la définition de scénario, action, but, ressource, etc. Il met l'accent sur deux notions : la notion d'interaction et la notion de flux. Les flux sont des actions complexes alors que les interactions sont des actions atomiques. Cette généralisation est exclusive.

Le lien *est composé de* entre la classe *Flux* et la classe *Action* indique qu'un flux est composé d'au moins une action (cardinalité 1..*). De la même manière, les actions peuvent être à leur tour composées de flux.

Le lien *est décrit par* entre la classe *Scénario* et la classe *Action*, montre qu'un scénario est décrit par une seule action (la cardinalité 1). Les actions décrivant un scénario sont souvent composées de plusieurs flux. Cependant, il n'est pas exclu qu'un scénario soit décrit par une seule interaction. D'un autre côté, une action fait partie d'une description de scénario ou d'un flux d'interactions. Par conséquent, les actions ne sont pas partagées entre plusieurs scénarios ni même entre plusieurs flux d'interactions au sein d'un même scénario (d'où la cardinalité 0..1 de *Action* vers *Scénario*).

Les deux sous sections suivantes détaillent les deux notions d'interaction et de flux.

3.2.2.1. Les interactions

Les interactions décrivent le comportement des agents dans un scénario. Elles peuvent avoir différents types tel que demande de service, fourniture de service, demande d'information, fourniture d'information, communication d'une ressource physique, etc.

Par exemple, « *Le client saisit la date de réservation souhaitée* » et « *Le résultat trouvé est affiché au client* », sont deux interactions représentant des échanges d'information entre agents.

Il est à noter que les phrases en langage naturel (LN) sont souvent ambiguës et incomplètes par rapport au modèle de scénario. Dans le cadre de l'approche L'Ecritoire [Tawbi01], l'interpréteur de LN et l'outil de vérification permettent de compléter et de lever les ambiguïtés de la description en langage naturel afin de respecter le modèle de scénario.
Les agents d'une interaction sont modélisés par l'entité *Agent*. Dans les deux interactions de l'exemple ci-dessus, deux agents sont identifiés : *client* et *application*.
Chaque interaction possède une direction. Les deux liens *de* et *à* entre les deux entités *Interaction* et *Agent* indiquent qu'une interaction est dirigée d'un agent à un autre. Par exemple, l'interaction « *le client saisit la date de réservation souhaité* », est dirigée de l'agent client à l'agent application.

A noter qu'une interaction peut comprendre un seul agent, c'est le cas dans l'interaction : *« la période de réservation est vérifiée »,* où l'application est l'agent *de* et *à* de la même interaction.
La Figure 4.1 montre qu'un agent dans un scénario est source ou destination d'au moins une interaction. Un agent qui ne participe à aucune interaction n'a aucune raison de figurer dans le modèle de scénario.
Une interaction met en jeu une ou plusieurs *ressources*. Le paramètre dans une interaction est la ressource échangée entre les agents du scénario. Ceci est modélisé par la relation *paramètre* entre les deux entités *Interaction* et *Ressource*. Les ressources sur lesquelles les actions sont appliquées sont généralement de nature informationnelle mais peuvent également représenter des objets physiques lorsque le système à développer est un système automatique. Par exemple dans l'interaction « *le client saisit la date de réservation souhaitée* », la ressource est *date de réservation.*
Comme le montre la Figure 4.1, trois types d'interactions sont identifiés : *actions internes*, *communications* et *communications utilisateur.*
Les *actions internes* représentent une activité locale à un agent. Dans ce cas précis l'agent *de* et l'agent *à* sont identiques. Par exemple dans l'action interne *« la période de réservation est vérifiée »,* l'action de vérification est réalisée par le *système de réservation* et la ressource manipulée est la *période de réservation.*

Toutes les interactions qui ne sont pas des actions internes représentent des activités de communication entre deux agents distincts. Les activités de communication faisant participer un agent humain sont considérées comme des *communications utilisateur*. Les deux premiers exemples d'interaction sont des exemples de communication utilisateur car elles sont associées à l'agent *Client*.

Les interactions qui ne sont pas des communications utilisateur sont considérées dans le modèle comme des *communications* car elles sont associées à deux agents de type système.

Par exemple dans l'action de communication *« le système de réservation envoie une demande de réservation à l'hôtel »* le *système de réservation* et l'*hôtel* représente respectivement l'agent source et cible ; la *demande de réservation* représente la ressource échangée.

3.2.2.2. La notion de flux

Les flux permettent de définir l'ordonnancement entre les interactions dans un scénario. Les flux sont classés en quatre types : s*équence, concurrence, répétition* et *condition*. Ces quatre types sont modélisés à la Figure 4. et sont détaillés dans les sections suivantes.

- **La séquence**

Une séquence est composée de deux actions, la deuxième action se déroule suite à la première. La proposition « *Le client saisit la date de réservation souhaitée et le résultat trouvé est affiché au client »* est un exemple de deux interactions en séquence. Etant donné qu'une action peut être composée de plusieurs actions, une séquence qui est une spécialisation d'actions peut être composée de plusieurs flux qui, à leur tour, peuvent appartenir à n'importe quel type de flux. Par contre, une séquence est toujours composée de deux actions au maximum. Par conséquent, une composition récursive de plusieurs séquences est utilisée pour exprimer le séquencement entre plusieurs interactions. Prenons l'exemple suivant: « *Le client confirme les données bancaires. L'agence de voyage confirme la réservation de vol à la compagnie aérienne et elle envoie les données bancaires à la banque ».* Cette séquence est composée de manière récursive de l'interaction : « *Le client confirme les données bancaires »,* qui se trouve en séquence avec le flux de concurrence composé de deux interactions : « *L'agence de voyage confirme la réservation de vol à la compagnie aérienne et elle envoie les données bancaires à la banque ».*

Le même raisonnement s'applique aux quatre types de flux. Ainsi, une séquence peut être composée soit de contraintes, soit de concurrences, soit de répétitions ou de séquences d'actions.

- **La concurrence**

Contrairement à une séquence, il n'y a aucun ordre spécifique entre deux actions concurrentes. Deux actions en concurrence ne commencent pas et ne se terminent pas nécessairement au même moment. Il est vrai que, du point de vue utilisateur, elles se déroulent au même instant mais ceci n'est pas nécessairement le cas du point de vue temporel. Prenons l'exemple suivant: *« L'agence de voyage envoie une demande de réservation de vol à la compagnie et de chambres à l'hôtel »*. Cette phrase se reformule dans le modèle de scénario par deux propositions concurrentes où « *l'agence de voyage envoie la demande de vol à la compagnie aérienne* et *l'agence de voyage envoie la demande de chambres à l'hôtel ».* Ces deux interactions n'ont pas d'ordre spécifique, et du point de vue temporel, elles ne commencent ni ne se terminent au même moment. Cependant, du point de vue utilisateur, elles se déroulent en même temps et deviennent donc, selon la définition d'un scénario, concurrentes.

Une concurrence est composée de deux actions. Comme dans une séquence, on peut exprimer la concurrence entre deux flux d'actions complexes, tels que deux séquences d'interactions en concurrence.

Contrairement à la séquence et à la concurrence, la *contrainte* et la *répétition* sont basées sur une *condition de flux*. Nous détaillons ces deux notions dans les deux sous-sections suivantes.

- **La condition de flux**

La condition de flux exprime les conditions nécessaires pour décrire un enchaînement spécifique d'actions dans un scénario. La condition de flux est associée à la contrainte et à la répétition. Ainsi cette notion est modélisée par l'entité *condition de flux* qui est liée aux deux entités *contrainte* et *répétition.*

La Figure 4.4 montre qu'une répétition (respectivement une contrainte) *est basée sur* une condition de flux qui peut impliquer un à plusieurs objets du scénario. Prenons l'exemple : *« Si la période de réservation souhaitée est valide ».* Cette condition de flux est attachée à une contrainte. Elle implique l'objet *« La période de réservation souhaitée ».*

- **La contrainte**

Une contrainte décrit les conditions nécessaires au déroulement de l'ensemble des actions qui la suivent. Le flux de contrainte est différent du flux alternatif (if-then–else) permettant de décrire plusieurs comportements possibles dans le même scénario et qui ne fait pas partie de notre modèle. Rappelons que notre définition d'un scénario met l'accent sur le fait que celui-ci décrit 'un comportement possible'. Par conséquent, les contraintes dans un scénario limitent la description à celle d'un comportement unique.

Prenons l'exemple du scénario '*Effectuer une demande de réservation via l'application en vérifiant les disponibilités*'. Ce scénario commence comme suit : *« le client effectue une demande de réservation, la période de réservation est vérifiée. Si la période est valide. L'application affiche une page au client proposant les disponibilités. Le client choisit …… ».* Ce scénario inclut une contrainte correspondant à la condition de flux *« Si la période de réservation est valide ».* Le reste du scénario décrit un comportement unique correspondant au cas où la période de réservation est valide.

- **La répétition**

La répétition permet d'exprimer les flux d'actions qui se répètent plusieurs fois dans un scénario. La répétition doit définir le nombre exact de fois où ses actions peuvent se répéter.
Prenons l'exemple *« Au plus trois fois et jusqu'à ce que le numéro de la carte soit valide, l'application affiche un message confirmant le paiement de la réservation ».* Les interactions de ce flux peuvent se répéter au maximum trois fois.
Comme les autres types de flux, une répétition peut être composée d'une combinaison d'actions. Cela est exprimé dans l'exemple ci-dessus où le flux de répétition est composé de trois interactions correspondant à *« l'application affiche un message confirmant le paiement de la réservation ».*

3.2.3. Les scénarios normaux et les scénarios exceptionnels

Nous distinguons deux types de scénarios, les normaux et les exceptionnels (cf. Figure 4.4).
Un scénario normal est celui qui permet d'atteindre le but, alors qu'un scénario exceptionnel se termine par la non satisfaction du but. Par conséquent, on peut dire que les scénarios normaux sont les cas de succès et les scénarios exceptionnels représentent les cas d'échecs.
Prenons par exemple le but « *obtenir une réservation de chambres d'hôtel »,* on peut imaginer trois cas possibles : « *dans le cas normal »,* « *dans le cas de non disponibilité », « dans le cas d'une erreur de période de validité ».* Les trois scénarios décrivent donc comment réaliser ce but de trois manières différentes. Le premier scénario est de type normal puisqu'il permet au client d'obtenir une réservation. Alors que les deuxième et troisième cas sont exceptionnels puisqu'ils ne permettent pas de lui délivrer une réservation. En effet, « *dans le cas de non disponibilité »* correspond au cas où l'hôtel n'aurait plus de chambres libres pour la période demandée et le cas *« dans le cas d'une erreur de période validité »* est détecté lorsque le client saisit une période où la date de fin est antérieure à la date de début ou lorsque la période demandée n'est pas encore ouverte à la réservation par le site.

Comme nous l'avons vu précédemment, un scénario ne décrit qu'un seul cas fonctionnel. Par conséquent, l'opérationnalisation d'un but se fait par un ensemble de scénarios représentant les différents cas possibles de succès (satisfaction du but) et les différents cas possibles d'échecs (non satisfaction du but) qu'il faut prendre en compte dans le logiciel qui supportera le client dans la réalisation de son but.

3.3. Le processus de construction des scénarios

Cette étape a pour objectif de construire les scénarios permettant de décrire le comportement d'un service atomique dans la satisfaction du but associé. Ceci est réalisé en reprenant l'approche Crews-L'Ecritoire [Tawbi 2001] utilisée comme un guidage méthodologique dans la construction des scénarios en langage naturel. Le comportement du service atomique est décrit par un ensemble de scénarios de base et exceptionnels.

L'approche Crews-L'écritoire est fondée sur le couple <but, scénario> pour décrire le comportement du service atomique. Cette approche guide la description du comportement selon deux activités : concrétiser un but par un scénario et analyser un scénario pour identifier des cas fonctionnels alternatifs ou des cas exceptionnels de non satisfaction du but.
La construction d'un scénario dans cette approche consiste à (i) l'écrire, (ii) le conceptualiser, (iii) le compléter au moyen de la stratégie de complétude et (iv) découvrir les scénarios alternatifs.
L'écriture d'un scénario consiste à choisir l'une de ces quatre directives :

- les directives de style et de contenu,
- les directives de contenu,
- les directives de style et,
- sans directives.

3.3.1. Ecrire un scénario avec les directives de style et de contenu

Crews-L'Ecritoire propose des directives qui aident à l'écriture du scénario textuel.
Le scénario est écrit sous la forme d'un flux d'actions atomiques. Pour guider cette écriture, deux types de directives sont proposés, les directives de style et les directives de contenu. Le premier définit le style textuel selon lequel les actions du scénario doivent être formulées, tandis que le deuxième se focalise sur le contenu du scénario en indiquant le type de connaissances que le scénario doit capturer.

3.3.2. Ecrire un scénario avec les directives de style

Les directives de style (DS) présentées à la Figure 4.5 sont conçues pour le méta-modèle de scénario présenté à la Figure 4.4

DS1: Chaque action atomique doit être déclarée par une clause composée d'un verbe et de plusieurs paramètres. Ainsi, une action atomique doit être exprimée selon l'un des modèles de clause suivant :
<Agent> <Action> <Ressources>.
ex. : « Le système de réservation vérifie la validité des dates de réservation »
<Agent1> <Action> <Ressources> à partir de <Agent2>
ex : « Le client choisit une date de réservation à partir du système ».
<Agent1> <Action> <Paramètre> à <Agent2>
ex : « le système de réservation affiche un message d'erreur au client »
DS2: Le modèle du scénario ne supporte pas les circonstances. Ainsi, les actions atomiques ne doivent pas inclure des termes faisant référence au temps, à la durée, au lieu, à la manière, etc.
DS3: Pour ne pas omettre l'un des agents, utiliser la voie active.
DS4: Eviter de faire des références au modèle de scénario.
DS5: Les conditions de flux doivent être déclarées explicitement au moment où elles influencent le chemin d'actions. Par conséquent, une condition de flux peut être exprimée selon l'un des deux modèles suivants :
Si <condition> alors <Flux d'actions>
Répéter <Flux d'actions> jusqu'à <condition>
DS6: Eviter l'utilisation des références anaphoriques comme 'il', 'elle', 'lui' 'eux' ou 'leur', etc. Utiliser plutôt des termes explicites.

Figure 4.5. Les directives de style

La ***DS1*** aide à écrire les actions selon la structure d'une action atomique définie par le modèle du scénario.

La ***DS2*** rappelle que le modèle de scénario ne supporte pas la notion de temps, de durée, de lieu et demande ainsi d'éviter d'inclure ce genre de termes dans le scénario.

La ***DS3*** propose de ne pas utiliser la voie passive afin de ne pas omettre des agents dans les actions atomiques décrites.

La ***DS4*** traite avec les conditions du flux en indiquant où, quand et comment il doit insérer de telles conditions.

La ***DS5*** indique le style exigé pour formuler les conditions de flux, les flux de contrainte et les flux de répétition.

Enfin la ***DS6*** propose d'éviter les références anaphoriques.

3.3.3. Ecrire un scénario avec les directives de contenu

Les directives de contenu (DC) sont présentées à la Figure 4.6.

DC1: Le scénario doit décrire un seul chemin d'actions.
DC2: Les scénarios alternatifs doivent être décrits séparément.
DC3: Il faut décrire l'enchaînement d'actions dans le cas où tout se passerait comme attendu. Par conséquent, éviter les actions décrivant des événements non attendus, les actions impossibles, non pertinentes à l'égard du domaine d'application ou celles n'illustrant pas la réalisation du but.
DC4: L'enchaînement d'actions doit comprendre un ordonnancement séquentiel d'actions.
DC5: Si le scénario contient des itérations, celles-ci doivent être restreintes par des conditions où le nombre d'itérations doit être bien précis. Pour cela, utiliser le modèle proposé par la directive de style

Figure 4.6. Les directives de contenu

Les *DC1* et *DC2* proposent d'écrire un scénario comportant un seul chemin d'actions et de ne pas écrire de scénario comportant des séquences de type 'if-then-else', puisque ce type de flux n'est pas inclus dans le modèle de scénario. Ainsi, par exemple, un flux du type *« si le numéro de la carte bancaire est valide, la banque envoie une notification d'acceptation du paiement sinon la banque envoie une notification de rejet »* n'est pas permis.

La *DC3* suggère de toujours commencer par écrire un scénario illustrant le *'cas normal'*, c'est à dire lorsque tout se passe comme prévu. Les scénarios écrits sont ensuite analysés afin d'explorer les déviations possibles. Celles-ci sont décrites par des scénarios séparés, comme nous allons le montrer plus tard dans ce chapitre.

La *DC4* procède à l'écriture par ordre séquentiel d'actions afin de faciliter la lisibilité du scénario.

La *DC5* traite avec le flux de concurrence dans un scénario en définissant le nombre exact d'itérations.

3.4. Exemple d'application des scénarios relatifs à un service

3.4.1. Expression textuelle des scénarios normaux

Considérons le service intentionnel atomique $S_{\textit{Effectuer une réservation}}$ ayant comme but *Effectuer une réservation.*

On se base sur les directives de style présentées à la Figure 4.5 et celles de contenu présentées à la Figure 4.6 pour écrire les flux d'actions suivant :

Le client formule une demande de réservation de vol et de chambres d'hôtel. L'agence de voyage enregistre la demande de réservation de vol et de chambres d'hôtel du client. Elle envoie une demande de réservation de vol à la compagnie et de chambres à l'hôtel. S'il existe des disponibilités de vol alors la compagnie aérienne confirme les vols et les prix. S'il existe des disponibilités de chambres alors l'hôtel confirme les chambres et le montant total à payer à l'agence de voyage. L'agence de voyage notifie les propositions de vol, de chambres et le montant à payer au client. Si le client accepte la proposition alors l'agence de voyage affiche le formulaire de saisie pour compléter la réservation et effectuer le paiement en ligne au client. Le client confirme en envoyant ses informations personnelles et ses données bancaires à l'agence de voyage. L'agence de voyage confirme la réservation de vol à la compagnie aérienne et confirme celle des chambres à l'hôtel. L'agence de voyage envoie les données bancaires à la banque. Si les données bancaires sont valides alors la banque effectue la transaction bancaire. La banque notifie la transaction à l'agence de voyage. Celle-ci confirme le paiement de la réservation.

Figure 4.7. Expression textuelle des scénarios relatif au service S *Effectuer une réservation*

L'activité qui suit l'écriture des scénarios est celle de leur conceptualisation, qui consiste à vérifier et à compléter la description des scénarios déjà écrits en proposant des mécanismes permettant de détecter les violations des directives de style et de contenu et de les corriger. [Tawbi 2001] a proposé un ensemble de mécanismes qui permettent de conceptualiser les scénarios. Ceci consiste à (i) vérifier la terminologie du scénario par rapport au glossaire du projet, (ii) clarifier et compléter le scénario à l'aide des patrons sémantiques et (iii) conceptualiser le scénario.

La vérification de la terminologie du scénario par rapport au glossaire du projet est une fonctionnalité qui permet de détecter les synonymes entre les termes du scénario et les termes du glossaire du projet.

La clarification et la complétude des scénarios consistent à se référer aux patrons sémantiques garantissant ainsi toute mauvaise interprétation des actions du scénario.

Enfin, la conceptualisation est une technique qui consiste à transformer le format textuel d'un scénario en un format semi-structuré où chaque action atomique et chaque condition de flux a un numéro et se situe sur une ligne séparée. Les flux sont aussi séparés à l'aide de tabulations.

La stratégie de complétude est l'activité qui suit la conceptualisation des scénarios. Elle permet de détecter l'absence d'actions atomiques dans un scénario conceptualisé et de les ajouter ensuite.

Ceci est fait en proposant deux alternatives :

- Compléter le scénario en raisonnant sur les couples demande/provision d'information et,
- Compléter le scénario en raisonnant sur les actions de vérification / condition de validité de ressources.

Des outils ont été proposés dans [Tawbi 2001] pour automatiser les activités de conceptualisation et de complétude de scénarios. Ainsi, la génération d'un scénario complet est faite d'une manière compatible avec des normes existantes.

La Figure 4.8 illustre le scénario $S_{Effectuer\ une\ réservation}$ de la Figure 4.7 après avoir été conceptualisé et complété. Des activités de conceptualisation et de complétude, telles que la suppression des références anaphoriques ou l'ajout des agents source et cible, ont été appliquées.

1. Le client formule une demande de réservation de vol et de chambres d'hôtel à partir de l'agence de voyage.
2. L'agence de voyage enregistre la demande réservation de vol et de chambres d'hôtel du client.
3. L'agence de voyage envoie une demande de réservation de vol à la compagnie aérienne
4. La compagnie aérienne vérifie les disponibilités de vol
5. **S'il existe des disponibilités de vol alors**
    6. La compagnie aérienne confirme les vols et les prix à l'agence de voyage.

3bis. L'agence de voyage envoie une demande de réservation de chambres à l'hôtel.
4bis. L'hôtel vérifie les disponibilités de chambres
5bis. **S'il existe de disponibilités de chambres alors**
    6bis. L'hôtel confirme les chambres et le montant total à payer à l'agence de voyage.

7. **S'il existe des propositions de vol et d'hôtel**
    8. L'agence de voyage notifie les propositions de vol, de chambres et le montant à payer au client.
    9. Le client sélectionne une des propositions.
    10**. Si le client accepte une proposition alors**
        11. L'agence de voyage affiche le formulaire de saisie pour compléter la réservation et effectuer le paiement en ligne au client.
        12. Le client confirme en envoyant ses informations personnelles et ses données bancaires à l'agence de voyage.
        13. L'agence de voyage confirme la réservation de vol à la compagnie aérienne.
        13bis. L'agence de voyage confirme la réservation de chambres à l'hôtel.
        14. L'agence de voyage envoie les données bancaires à la banque.
        15. La banque vérifie la validité des données bancaires
        16. **Si les données bancaires sont valides alors**
            17. La banque effectue la transaction bancaire.
            18. La banque notifie la transaction à l'agence de voyage.
                19. L'agence de voyage confirme la réservation au client.

Figure 4.8. Expression textuelle des scénarios relatif au service S *Effectuer une réservation*

3.4.2. Recherche de scénarios exceptionnels

La découverte de nouveaux scénarios alternatifs à partir d'un scénario conceptualisé permet de trouver les nouveaux scénarios qui représentent des manières alternatives de réalisation du but du service atomique. Les scénarios ainsi découverts sont liés par un lien 'OU' au but du service atomique.

La découverte des scénarios alternatifs est structurée en trois activités à savoir :

1- *Identifier les conditions de flux dans le scénario* : ceci consiste à extraire les conditions imbriquées du scénario initial.

A partir du scénario $S_{Effectuer\ une\ réservation}$, les conditions extraites sont les suivantes :

C1 : *s'il existe des disponibilités de vol*

C2 : *s'il existe des disponibilités de chambres*

C3 *: s'il existe des disponibilités de vol et de chambres (C1 et C2)*

C4 : *si le client accepte une proposition*

C5 : *si les données bancaires sont valides*

Le scénario $S_{Effectuer\ une\ réservation}$ représente l'imbrication de conditions suivante : $(C1 \wedge C2) \wedge C4 \wedge C5$.

2- *Calculer toutes les imbrications alternatives possibles* : les scénarios générés à partir des négations de conditions identifient des manières alternatives de satisfaction/non satisfaction du but initial du service atomique.

Les cas alternatifs au scénario $S_{Effectuer\ une\ réservation}$ se déduisent de la négation des conditions précédentes :

$\neg$ C1 *: pas de disponibilité de vol*

$\neg$ C2 *: pas de disponibilité de chambres*

$\neg$ C4 *: le client n'accepte aucune proposition*

$\neg$ C5 *: les données bancaires ne sont pas valides*

Le scénario $S_{Effectuer\ une\ réservation}$ doit être complété par le développement d'un scénario alternatif pour chaque cas identifié.

3- *Générer un scénario alternatif correspondant à cette imbrication* : pour chaque imbrication possible, de nouveaux scénarios sont identifiés. Ensuite, il faut déterminer s'il s'agit d'un cas normal (c'est-à-dire de succès) ou exceptionnel (c'est-à-dire d'échec).

L'élicitation des scénarios alternatifs à partir du scénario $S_{Effectuer\ une\ réservation}$ permet de décrire quatre nouveaux scénarios synthétisés par le tableau suivant (cf. Tableau 4.1) :

Tableau 4.1. Liste des scénarios alternatifs

Scénarios	Type	Manière	Résumé
1	**Echec**	Pas de disponibilité de vol	L'agence de voyage suggère au client de reformuler une nouvelle demande avec une période ou une destination différente
2	**Echec**	Pas de disponibilité de chambres	L'agence de voyage suggère au client de reformuler une nouvelle demande avec une période ou une destination différente
4	**Echec**	Aucune proposition sélectionnée	L'agence de voyage suggère au client de reformuler une nouvelle demande avec une période ou une destination différente
5	**Echec**	Données bancaires invalides	L'agence de voyage annule la réservation pour défaut de paiement et en informe le client.

Le scénario suivant (cf. Figure 4.9) est le scénario associé au but *Effectuer une réservation* dans le cas où il n'y aurait pas de disponibilités de vol. Les interactions notées en style italique proviennent du scénario de la Figure 4.8. Les interactions notées en gras sont spécifiques au scénario associé au but *Effectuer une réservation* dans le cas où il n'y aurait pas de disponibilités de vol.

1. Le client formule une demande de réservation de vol et de chambres d'hôtel à partir de l'agence de voyage.

2. L'agence de voyage enregistre la demande réservation de vol et de chambres d'hôtel du client.

3. L'agence de voyage envoie une demande de réservation de vol à la compagnie aérienne

4. La compagnie aérienne vérifie les disponibilités de vol

5. S'il n'existe pas de disponibilité de vol alors

6. la compagnie aérienne envoie le message de non disponibilité à l'agence de voyage

5bis. S'il existe de disponibilités de chambres alors

6bis. L'hôtel confirme les chambres et le montant total à payer à l'agence de voyage.

7. S'il n'existe pas (de disponibilité de vol ou de chambres) alors

8. L'agence de voyage envoie un message de reformulation d'une nouvelle demande au client.

Figure 4.9. Le scénario 2 du tableau 4.1

3.5. Les directives de transformation

Les directives de transformation aboutissent à l'opérationnalisation des services intentionnels atomiques.

Le méta-modèle MOS est une représentation des services opérationnels suivant une forme canonique. Les services opérationnels remplissent les actions nécessaires à l'accomplissement du but du service intentionnel tels que la gestion de la logique métier que le service implémente et l'administration de la coordination des différents agents fournisseurs de services. Ainsi, le méta-modèle MOS est considéré comme une opérationnalisation du service

intentionnel atomique. Les services opérationnels de ce modèle sont décrits indépendamment de la plateforme d'implémentation choisie.

L'objectif de cette étape est de découvrir les éléments constituant le service opérationnel MOS à partir des scénarios obtenus à la Figure 4.8 (scénario de base et scénarios alternatifs de succès ou d'échecs) lors de l'étape précédente (Ecrire le scénario de base et découvrir les exceptions).

La construction du modèle MOS par analyse des scénarios est réalisée en procédant par un ensemble de sous étapes. La structure du modèle généré est conforme à la structure présentée à la section 2.1 (Présentation des concepts du méta-modèle MOS).

Les directives de transformation pour la génération du méta-modèle MOS à partir des besoins utilisateur sont construites sous la forme de deux étapes :

1- Directives de transformation par analyse du scénario de base,
2- Directives de transformation par analyse des scénarios alternatifs d'échec.

3.5.1. Directives de transformation par analyse du scénario de base

La génération du modèle MOS, à partir du scénario de base, est réalisée par application des trois sous étapes suivantes :

- Directives de transformation des agents,
- Directives de transformation pour la découverte du service interface utilisateur,
- Directives de transformation pour la découverte du service métier.

3.5.1.1. Directives de transformation des agents du modèle de buts MIS vers le modèle opérationnel de service du modèle MOS.

Un agent est décrit comme une entité autonome communicante ayant certains rôles. Dans un service opérationnel, un agent est source ou destination d'au moins une interaction. Cette notion est expliquée en détail à la section 3.1.3.1 (cf. la notion d'interaction).

Un agent peut être de nature physique ou logique. Il représente l'abstraction d'un rôle joué par un acteur (utilisateur final du système, dispositif matériel ou autre système) qui interagit avec le système étudié. Il peut jouer un ou plusieurs rôles et un même rôle peut être tenu par plusieurs agents. Le rôle est une caractérisation du comportement d'un agent dans un contexte précis et peut être spécialisé. Le tableau 4.2 présente les directives de transformation d'agent issu des buts vers le service opérationnel.

Tableau 4.2. Directives pour la transformation d'agent du modèle MIS au modèle MOS

Concepts de buts issus de MIS	Information additionnel	Traduction dans le modèle MOS
Agent	Entité utilisateur, Initie la demande et reçoit les informations/services.	demandeur
	Entité système, gère les interactions entre le Demandeur et le Fournisseur Principal.	Interface
	Entité système, gère l'ensemble des services invoqués et contrôle les échanges.	Fournisseur Principal
	propriétaire de l'information/services demandés	Fournisseur secondaire

L'agent Demandeur dépend de l'Interface et du Fournisseur Principal pour la demande de service ainsi que pour la demande d'un ensemble de données partagées. L'agent Fournisseur Principal de son coté dépend du Fournisseur Secondaire pour fournir le service demandé par le Demandeur. Il dépend aussi du Demandeur pour la décision d'acceptation ou de refus du service demandé. Enfin, l'agent Fournisseur Secondaire dépend du Fournisseur Principal pour la réalisation, l'acceptation ou le refus du service fourni ainsi que pour la demande d'un ensemble de données partagées.

L'identification des rôles des agents va servir à instancier le Services d'Interface Utilisateur et le Service Métier du modèle MOS.

- **Spécialisation des agents des scénarios de la Figure 4.8**

L'application de l'étape 3.5.1.1 (Spécialiser les agents du service atomique) du processus d'identification des services opérationnels au scénario de la Figure 4.8 aboutit à la transformation illustrée à travers le tableau 4.3.

Le client dépend de l'agence de voyage pour la formulation de ses choix en termes de demande de réservation de vols et de chambres d'hôtel. Il dépend aussi de l'agence de voyage pour effectuer le paiement de la réservation. L'agence de voyage dépend (i) de la compagnie aérienne et de l'hôtel pour recevoir les possibilités de vols et de réservation de chambres (ii) du client pour le choix du vol et la chambre qui lui conviennent et pour la récupération des informations de paiement et (iii) de la banque pour effectuer la transaction bancaire.

Tableau 4.3. Directives pour la transformation d'agent des buts vers le service opérationnel de l'exemple de la Figure 4.8

Concepts de buts issus de MIS	Information	Traduction dans le modèle MOS
Client	Entité utilisateur, Formulation demande de réservation de vol et de chambre d'hôtel, et paiement de la réservation.	demandeur
Agence de voyage	Entité système, Permettre au client de faire le choix du vol et de la chambre, Récupérer les informations de paiement,	Interface
Agence de voyage	Entité système, Recevoir les possibilités de vol et de réservation de chambre, Effectuer la transaction.	Fournisseur Principal
Hôtel, Compagnie Aérienne et Banque	Propriétaire de services de réservation vol, chambre d'hôtel et paiement.	Fournisseur secondaire

3.5.1.2. Directives de transformation pour la découverte du service d'interface utilisateur.

Le service d'interface utilisateur permet d'interagir avec l'utilisateur et le Fournisseur Principal afin de délivrer le service attendu.

Nous proposons à la Figure 63, les étapes méthodologiques qui permettent d'identifier le service d'interface utilisateur, à partir des scénarios identifiés et de le modéliser selon les termes de MOS (cf. section 2.3). Le service d'interface utilisateur est conforme à la structure présentée à la section 2.3.2 :

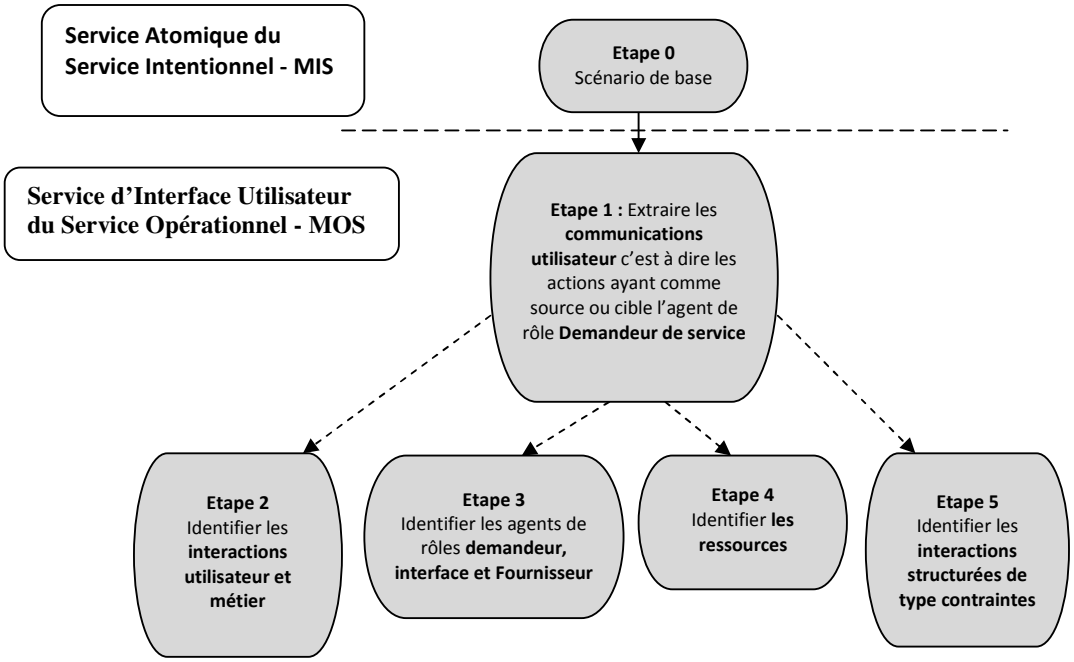


Figure 4.10. Le processus de découverte du Service d'Interface Utilisateur à partir du but du Service Atomique

Tableau 4.4. Directives pour la transformation des communications utilisateur du modèle MIS vers le service opérationnel du modèle MOS

Concepts de buts issus de MIS	Information	Traduction dans le modèle MOS
Service Atomique	Extraire les communications utilisateur du scénario en gardant l'ordre.	Interaction de base
	Identifier les agents de rôle demandeur, interface et fournisseur principal.	Agent
	Communication utilisateur entre le demandeur et Interface	Interaction utilisateur « entrée »
	Communication utilisateur entre l'Interface et le demandeur	Interaction utilisateur « sortie »
	Communication entre l'Interface et le fournisseur principal	Interaction métier « demande utilisateur »
	Communication entre le fournisseur principal et l'Interface	Interaction métier « réponse utilisateur »
	Associer à chaque communication utilisateur les ressources rattachées.	Ressource
	Extraire les cas de contrainte	Interaction structurée « contrainte »

Etape 1 :

L'étape 1 consiste à extraire les communications utilisateur du scénario de base en gardant l'ordre dans lequel elles sont mentionnées. Chaque communication utilisateur ayant comme source un agent dont le rôle est Demandeur correspond à une interaction en entrée. Une communication ayant comme cible un agent dont le rôle est Demandeur correspond à une interaction en sortie.

L'application de l'étape 1 au scénario $S_{\textit{Effectuer une réservation}}$ de la figure 4.8, distingue six interactions numérotées 1, 8, 9, 11, 12 et 19.

Etape 2 :

L'étape 2 permet d'identifier, à partir des communications utilisateur, les trois rôles : Demandeur, Interface et Fournisseur Principal.

En appliquant l'étape 2 aux communications utilisateur du scénario $S_{\textit{Effectuer une réservation}}$, on identifie deux agents à savoir le *client* qui joue le rôle de Demandeur et *l'agence de voyage* qui joue à la fois le rôle d'Interface et le rôle de Fournisseur.

Etape 3 :

L'étape 3 vise à identifier, à partir des communications utilisateur, les interactions utilisateur de type « entrée » et « sortie » ainsi que les interactions métier de type « demande utilisateur » et « réponse utilisateur ».

Tout d'abord, un scénario cohérent doit alterner les communications utilisateur de type « entrée » et les communications de type « sortie ». En effet, le système réagit à une entrée utilisateur par la production d'une sortie qui va permettre à l'utilisateur de réagir à son tour.

Une communication utilisateur de type « entrée » est opérationnalisée dans le service d'interface utilisateur de MOS par :

- une interaction utilisateur de type « entrée » entre le demandeur et l'Interface,
- une interaction métier de type « demande utilisateur » entre l'Interface et le Fournisseur Principal et,
- une interaction métier de type « réponse utilisateur » entre le Fournisseur Principal et l'Interface.

Une communication utilisateur de type « sortie » est opérationnalisée dans le service par une interaction utilisateur de type « sortie ».

Etape 4

L'étape 4 aide à compléter chaque interaction de base par l'identification des ressources échangées. Ceci est réalisé en associant, à chaque interaction, les ressources rattachées aux communications utilisateur. Plus précisément :

- Les ressources d'une communication utilisateur de type « entrée » sont les ressources de l'interaction utilisateur de type « entrée » et les ressources de l'interaction métier de type « demande utilisateur ».
- Les ressources d'une communication utilisateur de type « sortie » sont les ressources de l'interaction utilisateur de type « sortie » et les ressources de l'interaction métier de type réponse utilisateur qui la précède. Ceci est appliqué uniquement dans le cas où l'interaction métier qui précède a comme objectif de produire ce qui est fourni à l'utilisateur par l'interaction utilisateur de sortie.

Etape 5

L'étape 5 a comme objectif d'identifier les interactions structurées de type contrainte dans la séquence des interactions de base précédemment identifiée. Cette étape sert à compléter la modélisation du service d'interface utilisateur en associant les contraintes aux éléments déjà identifiés selon les termes du modèle MOS (cf. Figure 4.3).

Cette règle, appliquée à l'exemple de la Figure 4.8, commence par l'extraction des cas de contrainte à partir du scénario S *Effectuer une réservation*.

5. S'il existe des disponibilités de vol (C1)

5bis. S'il existe de disponibilités de chambres (C2)

7. S'il existe des propositions de vol et d'hôtel (C3)

10. Si le client accepte une proposition (C4)

16. Si les données bancaires sont valides (C5)

Chacune des contraintes est, potentiellement, une interaction qu'il faut positionner dans les interactions de base du service d'interface utilisateur, compte tenu de leur impact.

Les directives de transformations appliquées à l'exemple du scénario S *Effectuer une réservation* de la figure 4.8 donnent les résultats décris dans le tableau suivant. Les colonnes en bleu ne sont pas des communications pour aboutir au service d'interface utilisateur comme mentionné dans les directives de transformation du MIS au service d'interface utilisateur du modèle MOS. Les colonnes en gris sont les contraintes décrites dans les directives de transformations du MIS au service d'interface utilisateur du modèle MOS (l'étape 5).

Tableau 4.5. Les directives de transformation appliquées à l'exemple du S Effectuer une réservation

Actions du MIS S Effectuer une réservation	**Transformation pour la découverte du Service d'Interface Utilisateur du modèle MOS**					
	Interaction de base	**Type**	**Sens**	**Agent Source**	**Agent cible**	**Ressource**
Le client formule une demande de réservation de vol et de chambres d'hôtel à partir de l'agence de voyage	FormuleDemande RéservationVol& Chambres	Utilisateur	Entrée	Client	Agence - Interface	demandeReservation Vol&Chbre
L'agence de voyage enregistre la demande de réservation de vol et de chambre d'hôtel du client		Activité interne				
L'agence de voyage envoie une demande de réservation de vol à la compagnie aérienne	EnvoieDemande Réservation Vol&Hotel	métier	Requête	Agence - Interface	Agence – Fournisseur Principal	demandeReservation Vol&Chbre
La compagnie aérienne vérifie les disponibilités de vol et de chambre.		Activité interne				
S'il existe des disponibilités de vol ET des disponibilités de chambre ET des propositions de chambres et de vol		Contrainte C1 Λ C2 Λ C3				
L'agence de voyage reçoit les propositions de vol et chambre et le montant à payer, de la compagnie aérienne et de l'hôtel.	ReçoitListe Proposition	métier	réponse	Agence – Fournisseur Principal	Agence - Interface	propositionVol propositionCbre
L'agence de voyage notifie les propositions de vol, de chambres et le montant à payer au client	Notifie Propositions	Utilisateur	Sortie	Agence - Interface	client	propositionVol propositionCbre
Le client sélectionne l'une des propositions pour l'agence de voyage.	Sélection Proposition	Utilisateur	Entrée	Client	Agence - Interface	propositionVol Selectionnees propositionCbre Selectionnees
Si le client accepte une proposition		Contrainte C4				
L'agence de voyage envoie les propositions sélectionnées à la compagnie aérienne et à l'hôtel.	EnvoieProposition Selectionnée	Métier	requête	Agence - Interface	Agence – Fournisseur Principal	propositionVol Selectionnees propositionCbre Selectionnees

L'agence de voyage reçoit un accusé de réception de la part de la compagnie aérienne et de l'hôtel.	RéçoitAccusé Reception	métier	réponse	Agence – Fournisseur Principal	Agence - Interface	
L'agence de voyage affiche le formulaire de saisie pour compléter la réservation et effectuer le paiement en ligne au client.	AfficheInfo Paiement	Utilisateur	Sortie	Agence - Interface	client	formulaireInfo Perso&Paiement
Le client confirme en envoyant ses informations personnelles et ses données bancaires à l'agence de voyage	ConfirmePaiement EnLigne	Utilisateur	entrée	client	Agence - Interface	infoPerso& Paiement
L'agence de voyage envoie les informations personnelles et les données bancaires du client à la banque et confirme le vol et la chambre à la compagnie aérienne et à l'hôtel.	EnvoiePaiement EnLigne	métier	requête	Agence - Interface	Agence – Fournisseur Principal	infoPerso& Paiement
La banque notifie la transaction à l'agence de voyage	ReçoitConfirmation RéservationPayée Vol&Chambres	métier	réponse	Agence – Fournisseur Principal	Agence - Interface	reservationVol& Chbre
Si les données bancaires sont valides	Contrainte C5					
L'agence de voyage confirme la réservation au client	ConfirmeRéservation Vol&Chambres	Utilisateur	Sortie	Agence - Interface	client	reservationVol& Chbre

La dernière étape est la simplification des conditions formulées dans les contraintes. Par exemple, la première contrainte nécessite une simplification puisque *C1 and C2* sont incluses dans C3. Par conséquent, seule la condition C3 subsiste.

La représentation graphique indentée des interactions de base, et structurée du service d'interface utilisateur est donnée comme suit :

1. FormuleDemandeRéservationVol&Chambres
2. EnvoieDemandeRéservationVol&Chambres
3. ReçoitListeProposition
4. **Si (il existe au moins une proposition de vol et au moins une proposition de chambre) alors**
    5. SélectionProposition
    6. **Si (le client a accepté une proposition) alors**
        7. EnvoiePropositionSélectionnée
        8. ReçoitAccuséReception
        9. AfficheInfoPaiement
        10. ConfirmePaiementEnLigne
        11. EnvoiePaiementEnLigne
        12. ReçoitConfirmationRéservationPayéeVol&Chambres
        13. **Si (les données bancaires sont valides) alors**
            14. ConfirmeRéservationVol&Chambres

Cette étape permet d'avoir une première version du service d'interface utilisateur. L'analyse des autres scénarios alternatifs permettra de compléter cette version en développant les cas alternatifs qui traitent les 3 cas alternatifs suivants :

- Il n'y a pas de proposition de vol et de chambre,
- Le client n'a pas sélectionné une proposition,
- Les données bancaires ne sont pas valides.

3.5.1.3. Directives de transformation pour la découverte du Service Métier du modèle MOS.

La modélisation du service Métier selon les termes du modèle MOS, s'opère en cinq étapes :

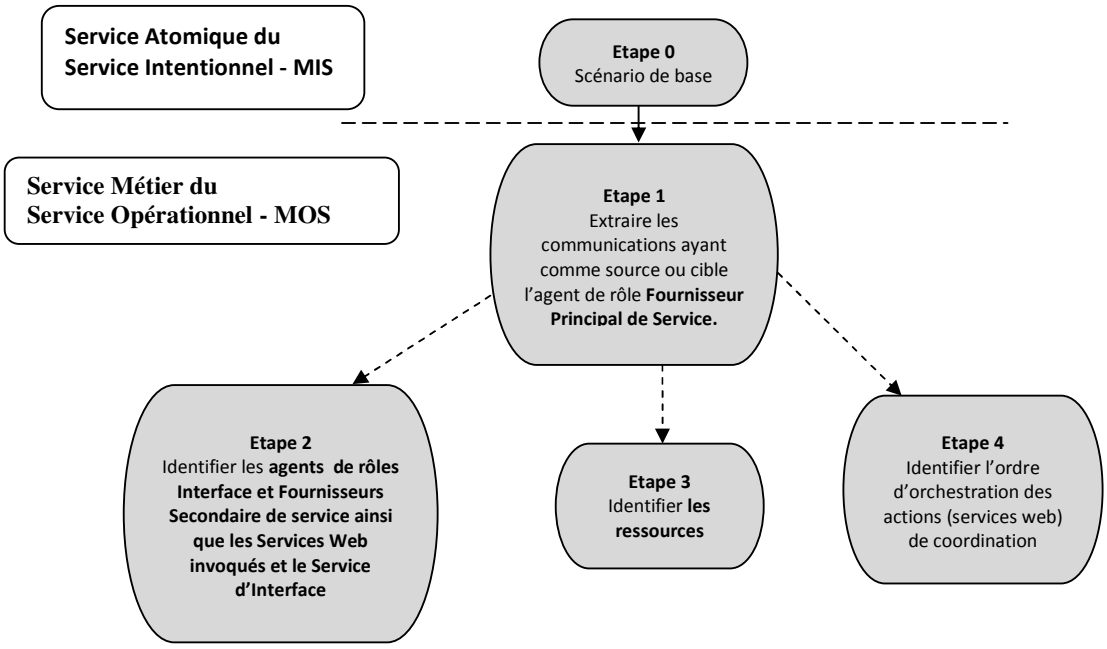


Figure 4.11. Le processus de découverte du Service Métier à partir du but du Service Atomique

Tableau 4.6. Directives pour la transformation des communications utilisateur du modèle MIS vers le service opérationnel du modèle MOS

Concepts de buts issus de MIS	Information	Traduction dans le modèle MOS
Service Atomique	Extraire les communications métier ayant comme source ou cible le fournisseur Principal en conservant l'ordre.	Activité de base
	Identifier les types de communications métier	Type : {Réception, Appel, Invocation, Réponse}
	Identifier les agents de rôle interface et fournisseur secondaire.	Agent
	Identifier les services web à invoquer	Services web à invoqués dynamiquement
	Identifier les services d'interface utilisateur invoquant le Service Métier	Service d'Interface Utilisateur
	Associer à chaque communication les ressources participant à l'orchestration	Ressource
	Identifier l'ordre d'orchestration des actions de coordination	Processus d'orchestration des services web à invoquer dynamiquement

Etape 1 :

L'étape 1 consiste à extraire, à partir du scénario, les actions de communication qui ont comme source ou cible un agent dont le rôle est « Fournisseur Principal », on les appelle les communications métier. Ces derniers correspondent aux actions échangées à partir de et vers le Fournisseur Principal pour réaliser l'objectif de l'orchestration. Par conséquent, les communications métier correspondent aux activités de communication du service Métier.

Une fois les communications identifiées, leur type est spécifié. Les communications métiers peuvent être : un appel pour invoquer dynamiquement une opération dans un service web, une réception pour attendre un message d'un Service Métier, une réponse pour répondre au Service.

Les communications métier qui se couplent en $C_{\text{Invocation dynamique}}$ et $C_{\text{Réponse}}$ sont traitées, dans le Service Métier, comme une activité de type appel prenant en compte la communication correspondant à l'appel et la communication correspondant à la réponse.

Les communications utilisateur en entrée sont prises en compte, dans le Service Métier, par les activités de type réception alors que les communications utilisateur en sortie sont considérées comme des activités de type réponse.

Etape 2:

L'étape 2 propose de déterminer (i) les agents participant au Service Métier, (ii) les services web à invoquer dynamiquement et (iii) les services d'interface utilisateur invoquant le Service Métier.

Dans un scénario, un agent est en même temps le consommateur d'un service que le Service Métier produit et le producteur d'un service que le Service Métier consomme. Dans ce cas, l'ensemble des agents dans un scénario forme les agents du Service Métier.

Le Service Métier a un rapport étroit avec les services web en faisant référence à leurs interfaces respectives lors de la coordination. Par conséquent, les services web sont identifiés à partir d'une requête d'invocation dynamique dans un annuaire de service web.

Les services d'interface utilisateur ont un rapport avec le Service Métier à travers les interactions métier de type Demande utilisateur. Ces interactions correspondent aux activités de base de type Réception dans le Service Métier. Par conséquent, les services d'interface utilisateur sont identifiés à partir des scénarios en appliquant les étapes proposées à la section 3.5.1.2.

L'étape 2 appliquée à la Communication métier du scénario $S_{Effectuer\ une\ réservation}$ aide à déterminer les éléments de composition à savoir :

- Les participants dans la communication métier tels que la compagnie aérienne, l'hôtel et la banque.
- Les services web qui devront être fournis par les participants sont *ServiceCompAérienne*, *ServiceHôtel*, *ServiceBanque*.
- Le service d'interface utilisateur à savoir Interface $_{Effectuer\ Réservation}$.

Dans le cadre de cette règle, il faut relier chaque *activité d'appel* à *une opération d'un service métier,* chaque activité de type *réception* à *l'interaction métier de type demande utilisateur* du service d'interface et chaque activité de type *réponse* à *l'interaction métier de type réponse utilisateur* du service d'interface qui réceptionne le message envoyé par le Fournisseur Principal.

Etape 3:

Cette étape aide à construire le modèle de données relatif à un service de coordination. Nous proposons d'extraire les ressources relatives aux communications métier qui ont comme cible un agent dont le rôle est Fournisseur Principal. Ces ressources constituent les messages en entrée nécessaires à l'orchestration. Les messages en sortie ne sont, dans ce cas, qu'une synthèse des messages en entrée.

Etape 4:

L'étape 4 définit le modèle d'orchestration du Service Métier en considérant l'ordre d'exécution des actions de communication métier. Les communications métier peuvent être structurées en séquence, concurrence, répétition ou contrainte.

Dans un Service Métier, les Communication Métier structurées en séquence correspondent aux activités structurées de type séquence. Les Communications Métier en concurrence correspondent, dans un service de coordination, à un flux d'activités pour l'acheminement parallèle. La répétition correspond à la structure de boucle dans un Service Métier. Enfin, la contrainte correspond à la structure de condition dans un Service Métier.

En appliquant l'étape 4 aux Communications métier du scénario S *Effectuer une réservation* on note qu'elles sont structurées en séquence. Par conséquent, les activités générées sont structurées de la même manière.

Tableau 4.7. Les directives de transformation de communication métier appliquées à l'exemple du S Effectuer une réservation

Actions du MIS S Effectuer une réservation	**Transformation pour la découverte du service Métier du modèle MOS**			
Communications Métier	**Interaction de base**	**Type communication métier**	**Type Service**	**Ressource**
Le client formule une demande de réservation de vol et de chambres d'hôtel à partir de l'agence de voyage	EnvoieDemande Reservation Vol&Chambres	Réception	Interface *Demande Réservation*	DemanderRéservation Vol&Chambres
L'agence de voyage enregistre la demande réservation de vol et de chambres d'hôtel du client	Activité interne			
L'agence de voyage envoie une demande de réservation de vol à la compagnie aérienne	Demander ReservationVol	Invocation dynamique de service web	ServiceCompAérienne	Liste_Vols Liste_Montant_total
La compagnie aérienne confirme les vols et les prix à l'agence de voyage.				
L'agence de voyage envoie une demande de réservation de chambres à l'hôtel.	Demander ReservationChbre	Invocation dynamique de service web	ServiceHotel	Liste_Chambres Liste_Montant_total
L'hôtel confirme les chambres et le montant total à payer à l'agence de voyage.				
L'agence de voyage notifie les propositions de vol, de chambres et le montant à payer au client	ReçoitListeProposition	réponse	Interface *Notifier Proposition*	
Le client sélectionne une des propositions.	EnvoieProposition Sélectionnée	Réception	Interface *Notifier Proposition*	
L'agence de voyage affiche le formulaire de saisie pour compléter la réservation et	ReçoitAccuséRception	Réponse	Interface *Suite* et paiement en ligne	

effectuer le paiement en ligne au client.				
Le client confirme en envoyant ses données personnelles et ses données bancaires à l'agence de voyage.	EnvoiePaiement EnLigne	Réception	Interface *Suite* et paiement en ligne	DonneesBancaire
L'agence de voyage confirme la réservation de chambres à l'hôtel.	ConfirmerReservation Chambres	Invocation dynamique de service web	ServiceCompAerienne	
L'agence de voyage confirme la réservation de vol à la compagnie aérienne	ConfirmerReservation Vol	Invocation dynamique de service web	ServiceHotel	
L'agence de voyage envoie les données bancaires à la banque.	EnvoyerDonnées Bancaires	Invocation dynamique de service web	ServiceBanque	reservationVol& Chbre
La banque notifie la transaction à l'agence de voyage.				
L'agence de voyage confirme la réservation au client.	ReçoitConfirmation RéservationPayée Vol&Chambres	réponse	Interface *Confirmation* paiement	

3.5.1.4. Directives de transformation pour la découverte des opérations et des ressources permettant l'invocation dynamique du service web

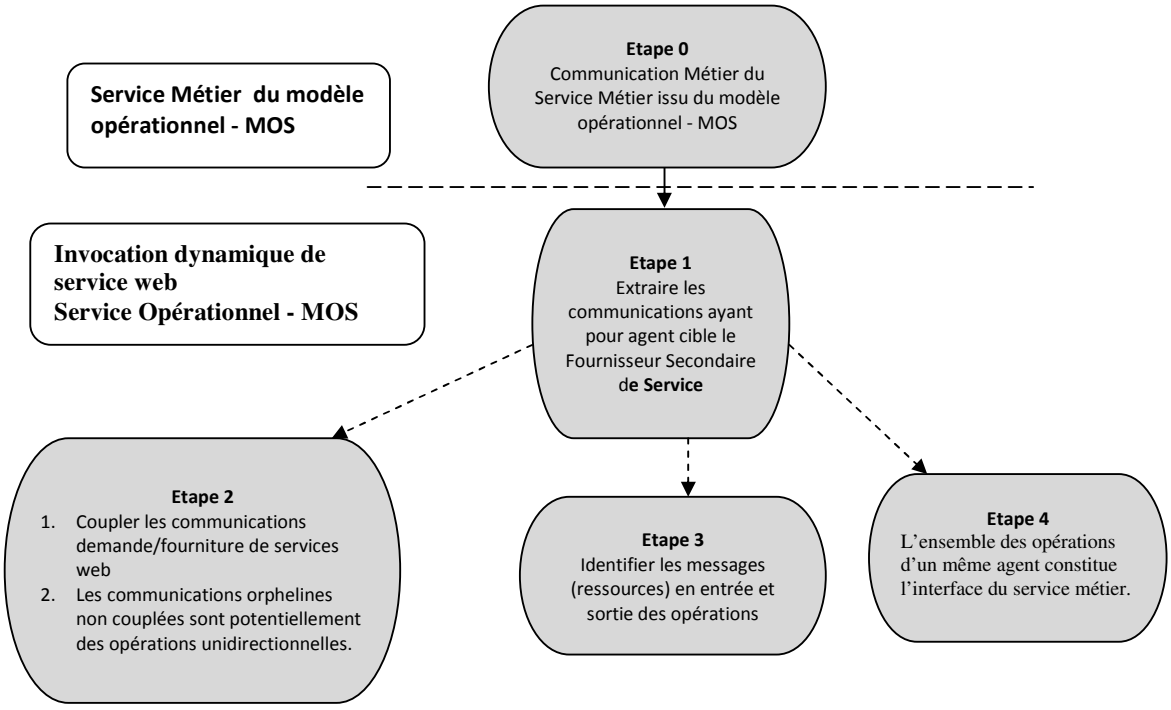


Figure 4.12. Le processus de découverte des opérations et des ressources permettant l'invocation dynamique du service web

Etape 1 :

L'étape 1 consiste à extraire les actions de communication représentant une invocation dynamique de services web. On appelle ces communications $C_{\text{Invocation dynamique}}$.

En appliquant l'étape 1 au scénario $S_{Effectuer\ une\ réservation}$, on obtient les cinq communications suivantes :

3. L'agence de voyage envoie une demande de réservation de vol à la compagnie aérienne
3bis. L'agence de voyage envoie une demande de réservation de chambres à l'hôtel.
13. L'agence de voyage confirme la réservation de vol à la compagnie aérienne.
13bis. L'agence de voyage confirme la réservation de chambres à l'hôtel.
14. L'agence de voyage envoie les données bancaires à la banque.

Etape 2 :

L'étape 2.1 a pour but d'identifier les opérations de type demande/réponse d'un service web. Ceci consiste à coupler les communications qui ont pour objet la demande/provision d'informations ou la production/consommation de ressources. Le couplage de communications est réalisé en procédant de la manière suivante :

- Identifier les ressources et,
- Construire les couples production/consommation en (i) identifiant les actions de production, (ii) identifiant les actions de consommation et (iii) couplant les activités de production avec les actions de consommation correspondantes.

En appliquant le principe de demande/provision d'information (respectivement de service), on cherche les paires de communications où une communication fournit le service ou l'information en réponse ($C_{Réponse}$).

Le couplage des actions de communications $C_{Invocation\ dynamique}$ et $C_{Réponse}$ se base aussi sur les couples de communications correspondantes à une production/consommation de ressources de type information ou physique dans un scénario. En appliquant le principe de production/consommation pour chaque ressource, on cherche les paires de communications où une communication consomme la ressource qui est produite par l'autre communication de la paire.

Afin d'identifier les opérations de type demande/réponse à partir des couples de communication, nous procédons comme suit :

Chaque communication $C_{Invocation\ dynamique}$ qui a pour objet une demande d'information est exprimée à l'aide du modèle de clauses suivant comme détaillé dans [Tawbi 2001]:

<Agent1><Action><Paramètres> à partir de <Agent2>

Afin d'identifier l'opération supportant cette communication, nous proposons d'associer une opération à chaque $C_{Invocation\ dynamique}$.

Par conséquent, l'opération suit le modèle de clauses suivant :

<Action><Ressources>

L'étape 2 appliquée au scénario S $_{Effectuer\ une\ réservation}$ permet d'obtenir les couples suivants :

a. *3. L'agence de voyage envoie une demande de réservation de vol à la compagnie aérienne : 6. La compagnie aérienne confirme les vols et les prix à l'agence de voyage.*

b. *3bis. L'agence de voyage envoie une demande de réservation de chambres à l'hôtel : 6bis. L'hôtel confirme les chambres et le montant total à payer à l'agence de voyage.*

c. *14. L'agence de voyage envoie les données bancaires à la banque : 18. La banque notifie la transaction à l'agence de voyage.*

L'étape 3 aide à identifier, à partir des communications orphelines, les opérations de type unidirectionnel.

L'étape 2.2 appliquée aux communications du scénario S $_{Effectuer\ une\ réservation}$, identifie les C $_{Invocation\ dynamique}$ qui sont des communications orphelines puisqu'il n'existe aucune action complémentaire de type C $_{Réponse}$:

- **a.** *L'agence de voyage confirme la réservation de vol à la compagnie aérienne.*
- ***b.*** *L'agence de voyage confirme la réservation de chambres à l'hôtel*

Etape 3 :

L'étape 3 détermine les messages en entrée et en sortie afin d'avoir une spécification complète de chacune des opérations de la manière suivante.

- Les ressources attachées à une action de communication C $_{Invocation\ dynamique}$ définissent les ressources en entrée de l'opération identifiée.
- Les ressources attachées à une communication C $_{Invocation\ dynamique}$ définissent les ressources en sortie.

Etape 4 :

L'étape 4 permet de décrire le service métier en fonction des opérations qu'il fournit. Ceci est exprimé à l'aide du concept d'interface où le service est décrit d'une manière abstraite suivant les fonctionnalités qu'il présente. Les interfaces servent à élucider les services métier candidats et réutilisables. Il s'agit ici d'identifier les parties des systèmes d'information de chaque agent Fournisseur Secondaire pouvant être encapsulées dans des services web et réutilisées dans l'application à base de services en cours de développement. Dans notre cas, ceci consiste à grouper les opérations (identifiées au cours de l'étape 2) d'un même agent fournisseur secondaire.

Tableau 4.7. Les directives de transformation pour la découverte des opérations et ressources des services web appliquées à l'exemple du S $_{Effectuer\ une\ réservation}$

Communication métier	Transformation pour la découverte des opérations et ressources des services web				
Communication représentant une invocation dynamique de service web.	**Opérations identifiée (unidirectionnel, bidirectionnel)**	**Message (ressource) en entrée**	**Message (ressource) en sortie**	**Liste des Fournisseurs Secondaires**	**Les Interfaces des services web**
L'agence de voyage envoie une demande de réservation de vol à la compagnie aérienne.	Demander Réservation Vol () (bidirectionnel)	Réservation Vol	Liste_vols Liste_prix	Compagnie Aérienne	ServiceComp Aérienne
L'agence de voyage envoie une demande de réservation de chambres à l'hôtel.	Demander Réservation Chambres () (bidirectionnel)	Réservation Chambre	Liste_Chambres ListeMontant Total	Hôtel	ServiceHôtel
L'agence de voyage confirme la réservation de vol à la compagnie aérienne.	Confirmer Réservation Vol () (Unidirectionnel)	Réservation Vol		Compagnie Aérienne	ServiceComp Aérienne
L'agence de voyage confirme la réservation de chambres à l'hôtel	Confirmer Réservation chambres () (Unidirectionnel)	Réservation Chambre		Hôtel	ServiceHôtel
L'agence de voyage envoie les données bancaires à la banque.	Envoyer Données bancaires () (bidirectionnel)	Données Bancaires	Reservation Vol&Chbre	Banque	Service Banque

3.5.2. Directives de transformation pour la découverte des scénarios alternatifs d'échec

Cette section permet de construire le modèle MOS par analyse des scénarios alternatifs d'échec. Un scénario alternatif d'échec représente un scénario qui se termine par la non satisfaction du but.

La construction du modèle MOS, à partir du scénario alternatif d'échec, ne considère que les actions précédemment présentées à la Figure 4.8 et générées à partir de la négation de la contrainte, à savoir :

7. S'il n'existe pas (de disponibilité de vol ou de chambres) alors

8. L'agence de voyage envoie un message de reformulation d'une nouvelle demande au client.

Ceci consiste à réappliquer le processus proposé à l'étape précédente et à l'étendre avec des règles supplémentaires.

L'application de la sous étape précédente précise les agents suivants et leurs rôles respectifs :

- Le client : correspond au rôle de *Demandeur* du service
- L'agence de voyage : l'agent est découpé en deux rôles systèmes, celui de l'*Interface* et celui de *Coordonnateur*.

Nous modélisons le service d'interface utilisateur. Le résultat obtenu est représenté dans le tableau 4.8.

Tableau 4.8. Les directives de transformation appliquées à l'exemple du S Effectuer une réservation

Actions du MIS S Effectuer une réservation	Transformation pour la découverte du Service d'Interface Utilisateur du modèle MOS					
	Interaction de base	**Type**	**Sens**	**Agent Source**	**Agent cible**	**Ressource**
L'agence de voyage envoie un message de reformulation d'une nouvelle demande au client.	AfficheMessage Reformulation	Utilisateur	Sortie	Agence - Interface	Client	Message Reformulation

Nous étendons notre processus avec la règle de l'étape 6 (*Directives pour la transformation des communications utilisateur du modèle MIS vers le service opérationnel du modèle MOS*). Le but est d'alimenter l'interaction structurée de type contrainte. En effet, la ligne 16 alimente la négation de la contrainte de la ligne 4 (**il existe au moins une proposition de vol et au moins une proposition de chambre**).

1. FormuleDemandeRéservationVol&Chambres

2. EnvoieDemandeRéservationVol&Chambres

3. ReçoitListeProposition ou MeesageReformulation

4. Si (il existe au moins une proposition de vol et au moins une proposition de chambre) alors

5. SélectionProposition

6. Si (le client a accepté une proposition) alors

7. EnvoiePropositionSélectionnée

8. ReçoitAccuséReception

9. AfficheInfoPaiement

10. ConfirmePaiementEnLigne

11. EnvoiePaiementEnLigne

12. ReçoitConfirmationRéservationPayéeVol&Chambres

13. Si (les données bancaires sont valides) alors

14. ConfirmeRéservationVol&Chambres

15. Sinon

16. AfficheMessageReformulation

17. Fin Si

L'application de la sous étape 3.5.1.4, complète la description de la modélisation du service web ServiceCompAérienne correspondant à son opération *DemanderRéservationVol ()* en générant des messages. En conséquence, un message de sortie de *non disponibilité* est obtenu à partir de la communication suivante :

6. la compagnie aérienne envoie le message de non disponibilité à l'agence de voyage.

L'application de la sous étape 3.1.5.3, complète la modélisation du Service Métier par la définition des activités de compensation.

Nous considérons le scénario alternatif d'échec comme étant une exception au niveau du Service Métier. L'exécution de l'exception produit une activité de type structurée qui représente les activités de compensation à prévoir. Nous présentons le résultat obtenu dans le tableau 4.9.

Tableau 4.9. Les directives de transformation de communication métier appliquées à l'exemple du S Effectuer une réservation

Actions du MIS S Effectuer une réservation	**Transformation pour la découverte du service Métier du modèle MOS**			
Communications Métier	**Interaction de base**	**Type Communications Métier**	**Type Service**	**Ressource**
L'agence de voyage envoie un message de reformulation d'une nouvelle demande au client	ReçoitProposition Reformulation	réponse	ServiceInterface	Proposition Reformulation

Nous étendons notre processus avec la règle de l'étape 5. Le but est d'alimenter le Service Métier par les activités de compensation déjà identifiées. L'application de l'étape 5 permet d'alimenter le Service Métier comme étant une activité de compensation produite suite à l'exécution du message d'exception *non disponibilité*.

4. Structure de l'Invocation Dynamique de Service Web au sein de la couche opérationnelle de service

L'invocation dynamique du service web est le mécanisme qui permet de rechercher dynamiquement les services web voulus par un agent au travers d'un annuaire de services de type UDDI [UDDI 2006]. La conception de l'invocation dynamique au sein du méta modèle MOS illustre le fait que notre approche de service intentionnel traite de la réingénierie de services en invoquant dynamiquement les services web déjà hébergés par les différents fournisseurs de services. Dans la section précédente, les directives de transformation nous ont permis d'obtenir les services web à travers le processus de transformation du service intentionnel atomique en service opérationnel. Ce processus de transformation a permis de transformer les communications métier du service atomique du modèle MIS en service web du modèle MOS tout en identifiant les opérations, les messages en entrée et les messages en sortie, de même que les interfaces et les fournisseurs secondaires des services qui devraient permettre la recherche et l'invocation du service. Ces informations utiles permettent de réaliser la requête d'invocation dynamique de service web dans un annuaire UDDI.

UDDI est un registre conçu pour héberger des informations sur les fournisseurs secondaires et leurs services de façon structurée. Au travers d'UDDI, il est possible de publier et de découvrir des informations sur un fournisseur secondaire et ses services Web. Ces données peuvent être classifiées à l'aide de taxinomies standards. Ces informations peuvent donc être recherchées par catégories. Enfin et surtout, UDDI contient des informations sur les interfaces techniques des services d'un fournisseur secondaire. Grâce à un jeu XML basé sur SOAP, il est possible d'interagir avec UDDI au moment de la conception et de l'exécution pour découvrir des données, de sorte que ces services peuvent être invoqués et utilisés. De cette façon, UDDI sert d'infrastructure sur les services Web.

L'UDDI est conçu comme un *registre* et non comme un *espace de stockage*. La distinction est subtile mais essentielle. Un registre dirige les utilisateurs vers les ressources, alors qu'un espace de stockage contient véritablement les informations. UDDI utilise des GUID (*Globally Unique IDentifiers*) pour réaliser des recherches et déterminer l'emplacement des ressources. Les requêtes UDDI aboutissent en fin de compte à une interface (un fichier .WSDL, .XSD, .DTD, etc.).

WSDL est un langage qui permet de décrire les services web, et en particulier, les interfaces des services web. Ces descriptions sont des documents XML. WSDL [W3C 2001] décrit un service web en deux étapes fondamentales : une abstraite et une concrète. Dans chaque étape, la description utilise un certain nombre de constructions pour favoriser la réutilisation de la description et pour séparer les préoccupations de conception indépendantes.

Au niveau abstrait, WSDL décrit un service Web en termes des *messages* qu'il envoie et reçoit; les *messages* sont décrits de façon indépendante d'un format spécifique de fil en utilisant un système de *types*, typiquement un schéma XML. La partie abstraite est composée de définitions de *"port type"* qui sont analogues aux interfaces dans les IDLs du "middleware" traditionnel. Chaque *"port type"* est une collection logique d'opérations. Une *"operation"* (i.e. opération) associe un modèle d'échange de message à un ou plusieurs messages. Un *message* est une unité de communication avec un service web. Il représente les données échangées dans une unique transmission logique.

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

Au niveau concret, un *"binding"* indique des détails de format de transport pour une ou plusieurs interfaces. Un *"endpoint"* (i.e. point final) associe une adresse de réseau à un

"*binding*" (i.e. attache). Finalement, un service groupe un ensemble d'*endpoints* qui implémente une interface commune.

Il n'est pas obligatoire de fournir des informations sur le port dans un fichier WSDL. Un WSDL peut contenir exclusivement des informations abstraites sur l'interface et ne fournir aucune donnée d'implémentation concrète. Un tel fichier WSDL est considéré valide.

XQuery est un standard récemment normalisé par le W3C. C'est un langage de requêtes extrêmement puissant et parfaitement adapté à l'interrogation de données XML.

4.1. Le langage de requête XQUERY

Le langage de requête XML, ou XQuery [XQuery 2003], est un langage intelligent et stable qui est optimisé pour l'interrogation de tous les types de données XML. Avec Xquery, nous pouvons exécuter des requêtes sur des variables et colonnes du type de données XML à l'aide des méthodes associées de ce dernier. Comme de nombreuses normes XML, le W3C (World Wide Web Consortium) supervise le développement de XQuery. XQuery est une évolution d'un langage de requête appelé Quilt, lui-même basé sur divers autres langages de requête tels que le Xpath (XML Path Language) version 1.0, XQL et SQL. Il contient également XPath 2.0 en tant que sous-ensemble. XQuery est le langage de requêtes W3C XML. La conception du langage XQuery est basé sur un ensemble d'expressions qui peuvent être composées ensemble arbitrairement. Ces expressions incluent la navigation dans les documents XML en utilisant XPath [XQuery 2004], la construction de nouvelles valeurs XML, des opérations sur les types de schema XML et des appels de fonctions. La plus significative dans le contexte des web services est la possibilité pour les utilisateurs XQuery de définir leurs propres fonctions.

4.2. Invocation dynamique de service web

Pour que les services web soient hébergés dans l'annuaire de service UDDI, ils doivent être décrits dans le langage WSDL. La contribution de cette directive est de permettre une liaison entre les services décrits en WSDL et le langage de requête XQuery. Cette approche permet de rechercher l'ensemble des services web équivalents à l'intention de l'utilisateur et à les invoquer pour que l'utilisateur puisse faire son choix de service web. L'exemple ci-dessous illustre le cas des services de réservation de vol de la compagnie aérienne découvert dans la directive précédente de la section 3.5.1.4 permettant de découvrir les services web. Le service web de la réservation de vol de la compagnie aérienne est décrit dans le langage WSDL pour être hébergé par un annuaire de service UDDI. Puis l'invocation de ces services de

réservation se fait au travers du langage de requête XQuery décrit dans la Figure 4.14. La fonction de la requête XQuery décrite dans cet exemple donne toutes les réservations de vol, c'est à dire la date et le montant total du vol pour une ville de départ et une ville d'arrivée donnée ainsi que la date de départ et le montant total. L'ensemble des vols sera transmit à un fournisseur principal du MOS qui se chargera de réunir les autres informations de la réservation de chambre et de les remettre à l'interface utilisateur puis au client final qui pourra faire son choix. Dans la requête de l'exemple de la Figure 4.14, la fonction a un nom (reg : DemanderReservationVol), prend des paramètres en entrée et retourne un résultat. Notons que le nom de la fonction, les paramètres en entrée et les résultats en sortie sont respectivement les opérations identifiées, les messages en entrée et les messages en sorties identifiés lors de la transformation des communications métiers en services web. Chaque paramètre est identifié par un nom de variable et un type. XQuery peut renvoyer des types de schémas XML existants utilisant des ″types séquences″, par exemple, ″element″ (*compagnieAerienne*) est un type de séquence qui renvoie à la déclaration de *″element″ compagnieAerienne* défini globalement. Le corps de la fonction est composé d'une expression XQuery qui calcule le résultat à partir des paramètres en entrée de la fonction. Dans notre exemple, la requête utilise la syntaxe XPath pour effectuer une recherche à l'intérieur du registre (supposée être ici un document XML), extrait les informations appropriées de la réservation de vol aux compagnies aériennes, puis construit un *element compagnieAerienne* qui contient la date et le montant total du vol ainsi que la ville de départ et la ville d'arrivée. Notons que la construction de *element* dans XQuery a la même syntaxe qu'en XML, et utilise des accolades pour revenir à la syntaxe de l'expression XQuery.

```
<definitions
    targetNamespace=″http://exemple.org/ReservationVol″
    xmlns : tns=″http://exemple.org/ReservationVol ″>
    <types>
     <xs: schema tragetNamespace =″http://exemple.org/ ReservationVol ″>
      <xs : element name=″compagnieAerienne″
       <xs: complextype>
         <xs : sequence>
          <xs : element name=″dateVol″ type=″xs:date″/>
          <xs : element name=″villeDepart″ type=″xs:string″/>
          <xs : element name=″villeArrivee″ type=″xs:string″/>
          <xs : element name=″montantTotal″ type=″xs:double″/>
```

```
          </xs : sequence>
        </xs: complextype>
      </xs : element>
    </xs: schema>
  </types>

<message name=″DemanderReservationVol″>
  <part name=″madateVol″ type=″xs:date″/>
  <part name=″mavilleDepart″ type=″xs:string″/>
  <part name=″mavilleArrivee″ type=″xs:string″/>
  <part name=″monmontantTotal″ type=″xs:double″/>
</message>
<message name = ″DemanderReservationVolResponse″>
  <part name=″listeVol″ element=″tns:compagnieAerienne″/>
</message>
<portype name=″ReservationVol″>
  <operation name=″DemanderReservationVol″>
  <input message=″tns: DemanderReservationVol ″/>
  <output message=″tns: DemanderReservationVolResponse ″/>
  </operation>
</portype>
<definitions>
```

Figure 4.13. Le WSDL pour le service ReservationVol

```
import module namespace reg=″http://resgister/reg″ ;
import schema namespace ″http://exemple.org/ ReservationVol.xsd ″ ;
declare function reg : DemanderReservationVol (
    $madateVol as xs : date,
    $mavilleDepart as xs : string,
    $mavilleArrivee as xs : string,
    $monmontantTotal as xs : double,
 ) as element (compagnieAerienne) {
 for $madateVol in $reg:dateVol,
       $mavilleDepart  in $reg: villeDepart,
       $mavilleArrivée  in $reg: villeArrivee,
       $mmontantTotal  in $reg: montantTotal
  where $madateVol
  order by $madateVol
 return
   < compagnieAerienne>
```

```
      <dateVol> {$madateVol } </dateVol>
      <villeDepart> {$mavilleDepart} </villeDepart>
      <villeArrivée> {$mavilleArrivée} </villeArrivée>
      <montantTotal> {$mmontantTotal} </montantTotal>
    </compagnieAerienne>
} ;
```

Figure 4.14. Requête XQuery sur un annuaire de service web décrit en WSDL

5. Conclusion

Dans ce chapitre, nous avons proposé une approche pour la construction d'applications à base de services logiciels. Cette approche repose sur l'introduction d'un modèle opérationnel de services MOS et d'une démarche méthodologique conduisant à sa construction.

Le modèle MOS décris, à un niveau opérationnel, les services qui sont un moyen d'opérationnalisation du service intentionnel atomique. Dans cette optique, un service logiciel constitue une entité opérationnelle ayant une responsabilité spécifique et décrite indépendamment de toute plateforme.

La démarche, que nous avons définie dans ce chapitre, permet de (i) guider l'utilisateur dans l'écriture des scénarios où chaque scénario représente une manière possible de réalisation du service atomique, (ii) générer de façon systématique l'opérationnalisation des services intentionnels atomiques à l'aide du modèle MOS et (iii) invoquer dynamiquement les services web.

De cette façon, nous avons pu répondre au problème de la mise en correspondance entre les besoins des clients exprimés, à un niveau opérationnel, à l'aide des services du modèle MIS et les services logiciels proposés par les développeurs, à un niveau opérationnel, à l'aide des services du modèle MOS.

Certains avantages de l'approche proposée sont les suivants :

- Utilisation multiple du même modèle MOS pour générer des modèles sur des plateformes différentes (service web ou autre).
- Indépendance de la logique métier aux technologies d'implémentation.
- Evolution simultanée du niveau intentionnel et opérationnel des services.
- Invocation dynamique des services web au travers de requêtes Xquery.

CHAPITRE 5 : DU SERVICE OPERATIONNEL AU SERVICE LOGICIEL INTERACTIF

1. Introduction

Ce chapitre présente les phases 4 et 5 de la démarche MeTSI. La phase 4 transforme un service opérationnel basé sur le méta- modèle MOS en un service logiciel interactif basé sur le modèle d'implémentation de service interactif (MISI). Le processus de transformation de la phase 4 s'appuie sur un méta-modèle architectural d'infrastructure facetté qui comporte plusieurs facettes liées à plusieurs attributs. Les facettes d'architecture ont été mises en place pour aider l'ingénieur du modèle opérationnel à faire un choix sur la plateforme d'implémentation et l'architecture logicielle du service interactif. Les facettes comportent des valeurs telles que locale ou invocation à distance. A la fin de la transformation, le modèle MISI obtenu est semblable pour chacune des différentes plateformes, OSGi [OSGi 2009], WSRP [WSRP 2006] et de type Mashup [Wikipedia, Mashup 2006]. Le modèle MISI représente le système attendu au niveau technique de modélisation des plateformes interactives. Rappelons que le modèle MOS comporte deux blocs insécables qui sont l'interface utilisateur (qui prend en compte les interactions des utilisateurs à travers des composants de type vue et de type contrôleur), la coordination (qui traite de l'orchestration des web services à travers un processus d'orchestration) et les services web. La phase 6 produit le résultat final par génération du code dans le cas de la plateforme WSRP.

Ce chapitre comporte trois parties : la première partie est dédiée à l'explication des méta-modèles utilisés (facettes architecturales et MISI) ; la seconde décrit le processus de transformation de MOS à MISI en utilisant les facettes architecturales pour choisir la plateforme et l'architecture du service interactif à implémenter. Et le troisième montre le processus de génération d'une implémentation sur la plateforme WSRP.

2. La liaison entre le modèle opérationnel de service et le modèle d'implémentation de système Interactif

Le Modèle d'Implémentation de Système Interactif (MISI) est un modèle spécifique appliqué à une des plateformes étudiée au chapitre 2. A ce niveau de modélisation spécifique, le modèle obtenu reflète les besoins de l'utilisateur d'une manière liée à l'infrastructure de services.

2.1. Le méta-modèle de facettes architecturales.

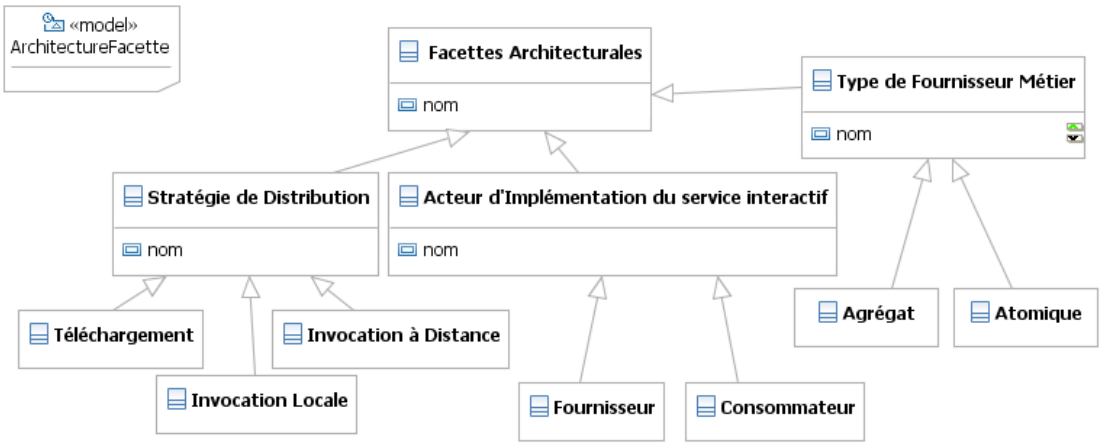


Figure 5.1. : Le méta-modèle de facettes architecturales

Le méta-modèle présenté à la figure 5.1 est le méta-modèle de facettes architecturales qui permet de décrire un ensemble d'attributs et de facettes liés aux besoins fonctionnels nécessaires à la prise de décision pour la construction d'un modèle spécifique de services interactifs relatif à un type de plateforme.

Le modèle de facettes architecturales rassemble les éléments suivants : la stratégie de distribution, le type de fournisseur métier et l'acteur d'implémentation des services. Ces facettes regroupent un certain nombre de valeurs qui décrivent le comportement de chaque organisation de manière unique. La figure 5.2 résume les différentes facettes et leurs valeurs

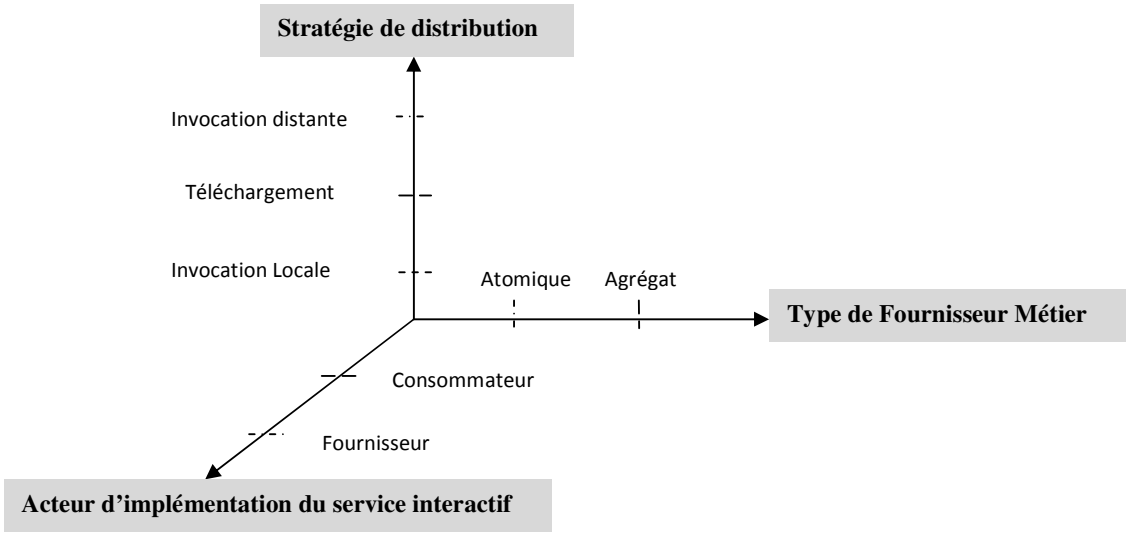


Figure 5.2. : Attributs des facettes architecturales

Selon l'axe de distribution des attributs de la figure 5.2, nous définissons trois grands domaines de besoin pour implémenter des services interactifs. Ces besoins permettent à l'utilisateur final d'avoir un panel de choix possibles d'architectures liées aux différentes plateformes. Ces besoins fonctionnels sont classés en fonction de trois facettes ayant des attributs différents. Ces facettes sont classées en fonction des stratégies de distribution, de l'acteur d'implémentation de services et du type de fournisseur métier. Ces trois éléments sont définis dans les sections ci-après. L'axe de distribution de la figure 5.2 donne une vue globale d'une architecture cliente lorsque trois attributs de chacun des axes des trois facettes sont sélectionnés par le client.

a. La stratégie de distribution

La stratégie de distribution indique quelle est la distribution de services que le consommateur de service aimerait voir appliquer aux services interactifs. La distribution des services est fonction de plusieurs facteurs dont la connectivité (débit de connexion), la taille de l'environnement (assistants personnels ou des passerelles domestiques, environnements limités ou pas en mémoire), puissance de traitement, temps de chargement, temps de réponse, temps d'exécution.

La stratégie de distribution prend la valeur de l'attribut *invocation à distance* lorsque les services interactifs restent chez le fournisseur pour être exécutés à distance. Dans ce cas de figure, le consommateur de service invoque les services interactifs directement chez le fournisseur. Les avantages d'une telle solution est que l'environnement client et la

connectivité ne sont pas limités pour recevoir les services composites. La stratégie de distribution prend la valeur de l'attribut *téléchargement* sur une plateforme centralisée lorsque le consommateur de service est limité par la taille de l'environnement, c'est le cas lorsque l'utilisateur possède un assistant personnel ou des passerelles domestiques. D'autres domaines d'application incluent les automobiles, l'automatisme industriel et les téléphones portables. Enfin, la troisième et dernière stratégie est la distribution *locale.* Dans cette solution, la partie « interface utilisateur » du service interactif est développée ou installée dans l'environnement du consommateur de service. Par conséquent, son exécution se fait dans l'environnement local du consommateur de service.

b. L'acteur d'implémentation du service interactif

La seconde facette intitulée *acteur d'implémentation du service*, détermine l'acteur qui implémente les services interactifs.

Lorsque le service interactif est implémenté chez l'acteur *consommateur* alors l'interactivité du service est réduite à une seule interface utilisateur dont les composants sont construits selon les intentions de l'utilisateur. Dans ce cas, les services web métier sont agrégés sur cette seule interface utilisateur et chaque application d'agrégation communique avec chacun des services web via son interface utilisateur unique définie selon le client. Lorsque le service interactif est implémenté chez le *fournisseur* de services alors il est possible d'implémenter plusieurs interfaces utilisateur qui pourront être invoquées en même tant que les services web. Le fournisseur de service pourrait fournir en plus des services web composites, des services d'interface utilisateur que l'utilisateur pourrait composer comme il le souhaite. Ce dernier aura le choix du service d'interface qui pourrait être combiné avec son service web composite pour chaque scénario d'interaction utilisateur. L'utilisateur pourra avoir accès à un registre de services interactifs disponibles et ces services sont définis de telle façon qu'ils peuvent être utilisés dans différents contextes. Les services peuvent être orchestrés dans le modèle d'interaction approprié.

c. Le type de fournisseur métier

Le type de fournisseur métier permet de savoir qui distribue et implémente la partie « orchestration » du service métier. En effet, si le fournisseur métier est de type *agrégat*, cela implique que l'orchestration des services web métier définit au sein du service interactif est elle-même considérée comme un web service métier digne d'intérêt. Dans ce cas,

l'orchestration peut être implémentée comme une orchestration BPEL avec un descripteur WSDL permettant de le publier et de l'exécuter comme un web service. Dans ce cas, l'architecture d'implémentation du service interactif est réalisée en 3 couches : la partie Interaction Utilisateur, le web service d'orchestration et les web services métiers atomiques.

Par contre, si le fournisseur métier est de type *atomique*, l'orchestration de services web métier n'est pas considérée comme un web service et peut être implémentée au même niveau que la partie « Interface utilisateur ». Dans ce cas, l'architecture du service interactif est plutôt répartie sur deux couches au lieu de trois.

La section 3 décrit l'usage de ces facettes architecturales pour choisir l'architecture du service interactif et la plateforme choisie pour son implémentation. La section 2.2 introduit le modèle MISI que nous avons défini pour la réalisation des services interactifs avec la plateforme WSRP.

2.2. Construction du modèle d'implémentation du service interactif

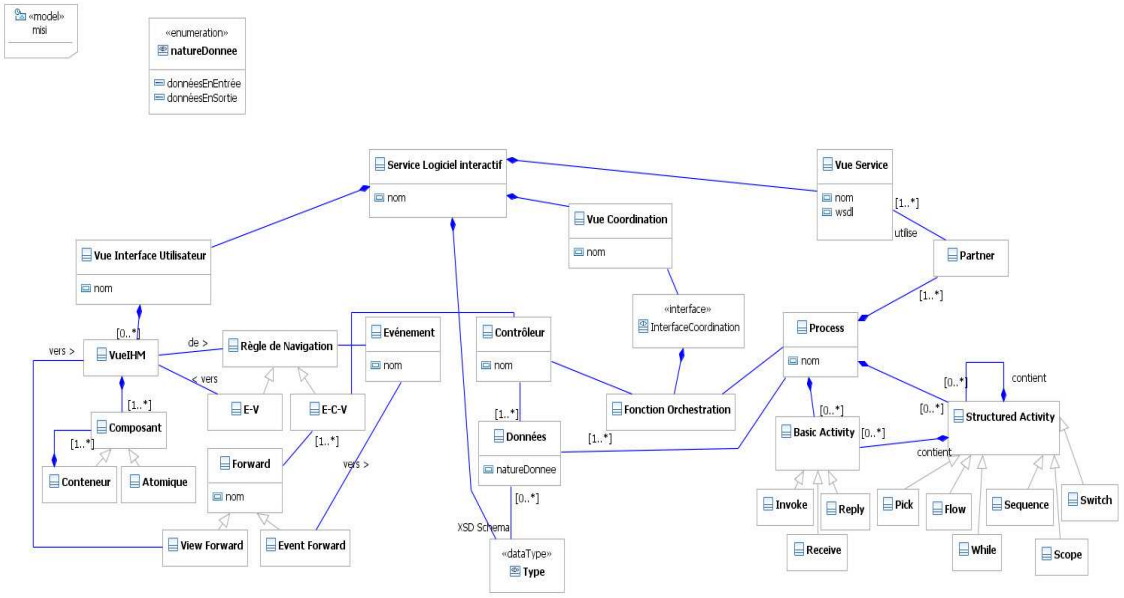


Figure 5.3. : Le méta-modèle du modèle MISI

La figure 5.3 présente le modèle d'implémentation de service Interactif ou méta-modèle MISI. Ce méta-modèle est dépendant d'une plateforme de service interactif. Comme mentionné au chapitre 3, l'approche MeTSI peut être implémentée à travers trois plateformes spécifiques de service interactif qui sont le WSRP (Web Service Remote Portlet), l'OSGi ou le Mashup. Le modèle d'une de ces plateformes est obtenu par transformation à partir du

méta-modèle MOS. Le processus de transformation est un processus basé sur le méta-modèle des facettes architecturales et prend en compte les différents besoins exprimés par l'utilisateur pour atteindre son but. Les différents besoins sont alors exprimés de manière technique et architecturale à travers le modèle de facettes architecturales. De ce modèle de facettes découle différentes architectures d'implémentations possibles. Le processus de transformation du passage du modèle MOS au modèle MISI est un processus de transformation spécifique d'une architecture technique qui permet le passage d'une modèle MOS de type PIM (Plateforme Independent Model) à un modèle MISI de type PSM (Plateform Specific Model).

Dans la section qui suit nous détaillons les différents éléments du méta-modèle MISI. Le modèle d'implémentation du Système Interactif est un méta-modèle composé de trois parties qui sont le méta-modèle de la vue Interface Utilisateur, le méta-modèle de la vue Coordination et le méta-modèle de la vue service web. Le méta-modèle de la vue Interface Utilisateur représente les applications gérant les interactions avec l'utilisateur à l'aide des interfaces graphiques. Le méta-modèle de la vue coordination représente quant à elle les applications métier qui réalisent la réutilisation et l'orchestration de plusieurs services web et qui est invoquée par la vue Interaction Utilisateur. Cette dernière est alors appelée le modèle d'avant plan alors que les vue coordination et service web sont appelées les modèles d'arrière plan. La vue Interface Utilisateur est développée à l'aide des techniques de développement d'interfaces graphiques. Les techniques d'intégration d'applications classiques proposent de développer des transactions métier distribuées sur plusieurs applications métier au sein d'une même organisation. Ces techniques d'intégration sont utilisées pour développer les services métier du modèle MISI.

Le regroupement de ces éléments logiciels permet de décrire *l'unité applicative technique* à développer pour supporter l'utilisateur dans la réalisation de son but. L'unité applicative technique est introduite, dans ce chapitre, par le concept de *service logiciel interactif* conforme au méta- modèle MISI.

Les plates-formes de coordination des services web proposent la modélisation et l'exécution de transactions métier distribuées dans plusieurs organisations. Chaque organisation fournit, par le biais de services web, des transactions métier distribuées. Les transactions métier inter-organisations sont modélisées par le concept de la vue coordination.

Comme nous pouvons le remarquer, le service logiciel interactif assemble des éléments logiciels réalisés séparément avec des techniques de développement différentes. Cette vision de l'unité applicative technique en trois couches s'applique aussi bien pour implémenter un

service logiciel intra organisation qu'un service inter organisation. En effet, les plates-formes de coordination de services sont utilisées d'un point de vue architectural pour deux raisons : pour implémenter l'orchestration de plusieurs services mais aussi pour rendre indépendante l'application d'avant plan de l'application d'arrière plan.

La vue coordination, dans ce cas, sert uniquement à rendre l'application cliente indépendante de l'application fournisseur afin de prendre en compte d'éventuelles évolutions où le service web réutilisé peut être différent pour un autre fournisseur.

Le méta-modèle MISI présenté dans cette section correspond au méta-modèle conforme à la plateforme WSRP.

Notons également que le méta-modèle de la figure 5.4 peut être vu selon l'architecture 3-tiers Modèle-Vue-Contrôleur (MVC) où le *modèle* est comparé à la fonction qui fait appel à la vue service web et qui se nomme *Fonction Orchestration*, la *vue* est comparée à la vue Interface Utilisateur et le *contrôleur* est comparé au *contrôleur* de la vue Interface Utilisateur.

Pourquoi se positionner par rapport à l'architecture MVC? Tout d'abord, dans cette architecture, il est possible d'avoir plusieurs vues "graphiques" qui dépendent d'un seul **modèle**. Comme le **modèle** renvoie les données sans appliquer aucune mise en forme, les mêmes composants peuvent être réutilisés et appelés pour n'importe quelle interface. Le modèle, représente les données et les règles métiers. Comme ce dernier est autonome et séparé du contrôleur et de la vue, il est beaucoup plus facile de modifier la couche de données ou les règles métier. Si vous changez de fournisseur de service métiers, il suffit de revoir la partie modèle.

L'architecture MVC accroît la productivité et facilité d'utilisation. Elle permet une réutilisation des services métiers, un choix de technologie et non pas « de tout ou rien », une flexibilité et un contrôle de l'interface utilisateur. Avec le modèle MVC, nous disposons d'un couplage très faible entre la vue *Interface Utilisateur* et la vue *Coordination*.

Nous pouvons ainsi élaborer des interfaces bien définies et des composants autonomes. Le **contrôleur** interprète les requêtes de l'utilisateur et appelle le **modèle** et la **vue** nécessaires pour répondre à la requête. Pour résumer, une requête utilisateur est interprétée par le **contrôleur**, qui détermine quelles portions du **modèle** et de la **vue** doivent être appelées. Le modèle gère les interactions avec les services métiers et applique les règles métier, puis renvoie les données. Enfin, le contrôleur sélectionne une vue et lui passe les données.

Les différentes parties du méta-modèle sont expliquées dans les sections suivantes :

2.2.1. Le méta-modèle de la vue Interface Utilisateur

La vue *Interface Utilisateur* est une application qui gère les interactions avec l'utilisateur à l'aide des interfaces graphiques. Elle est celle qui a la responsabilité de générer ce qui est affiché à l'écran, c'est-à-dire le positionnement des composants graphiques ainsi que les détails de leur aspect comme la couleur.

La vue *Interface Utilisateur* correspond à l'interface avec laquelle l'utilisateur interagit. Sa première tâche est de présenter les résultats renvoyés par le modèle de la vue coordination. Sa seconde tâche est de recevoir toutes les actions de l'utilisateur (clic de souris, sélection d'une entrée, boutons, …). Ces différents événements sont envoyés au contrôleur. La vue *Interface Utilisateur* n'effectue aucun traitement, elle se contente d'afficher les résultats des traitements effectués par le modèle. Plusieurs vues *Interface Utilisateur*, partielles ou non, peuvent afficher des informations d'un même modèle. La vue peut aussi offrir la possibilité à l'utilisateur de changer de vue.

Différents concepts sont définis dans la figure 5.4. Nous donnons une définition de chacun d'eux. Notons que notre méta-modèle ci-dessous est beaucoup spécifique à la plateforme WSRP qui sont conforme aux besoins de l'utilisateur final exprimés au travers les facettes architecturaux.

Le méta-modèle de la vue *Interface Utilisateur* est décrit à la figure 5.4.

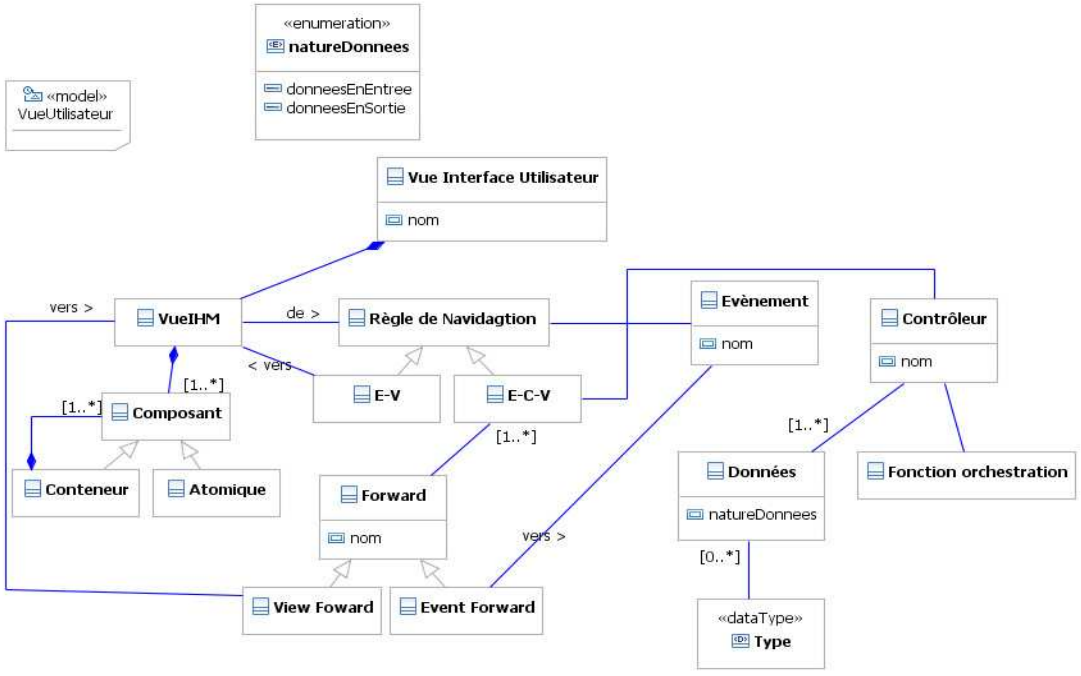


Figure 5.4. : Le méta-modèle de la vue Interface Utilisateur

Le méta-modèle de la vue interface utilisateur se fait dans la spécification EMOF (Essentiel MOF) qui est défini selon le standard MOF [EMOF 2004]. Pour la spécification des méta-modèles invoqués, nous proposons d'utiliser l'outil Eclipse UML2Tool [UML2Tool], lequel permet de fournir la génération automatique des méta-modèles EMF à partir des méta-modèles UML2 définis. EMF est le « Framework Modeling Eclipse » qui est basé sur la spécification EMOF. Le méta-modèle de la vue Interface Utilisateur de la Vue Interface défini un certain nombre d'éléments qui sont défini ci-dessous :

VueIHM représente un écran web utile pour acquérir les interactions utilisateurs de type entrée et sortie. Cet écran correspond à l'interface avec laquelle l'utilisateur interagit. Sa première tâche est de présenter les résultats renvoyés par le modèle. Sa seconde tâche est de recevoir toutes les actions de l'utilisateur (clic de souris, sélection d'une entrée, boutons, …). Ces différents événements sont envoyés au contrôleur. L'objet ***VueIHM*** n'effectue aucun traitement, elle se contente d'afficher les résultats des traitements effectués par le modèle. Plusieurs écrans peuvent afficher des informations d'un même modèle.

Composant : Un objet composant est un objet qui permet de définir le décor d'un objet de type *vueIHM* qui est visible à l'utilisateur et lui permet d'interagir. Ce sont des éléments configurables et réutilisables qui composent les interfaces utilisateur de l'objet application.

On distingue 2 types de composants : les conteneurs et les atomiques.

Les conteneurs : Un conteneur est un objet de type **composant** qui a comme fonction de regrouper d'autres composants graphiques **atomiques** ou **conteneurs**, en d'autre terme c'est une hiérarchie de composition d'objets graphiques. L'application propose des conteneurs tels que les formulaires.

Les atomiques : Ce sont les divers objets graphiques usuels et non décomposables d'une interface graphique qui sont visibles à l'utilisateur. Nous en citons quelques uns :

- ✓ *Command* **:** objet graphique qui permet de réaliser une action qui lève un événement.
- ✓ *Graphic* : objet graphique qui représente une image.
- ✓ *Input* : objet graphique qui permet de saisir des données
- ✓ *Output* : objet graphique qui permet d'afficher des données
- ✓ *SelectItem* : objet graphique qui représente un élément sélectionné parmi un ensemble d'éléments.
- ✓ *SelectItems* : objet graphique qui représente un ensemble d'éléments
- ✓ *SelectBoolean* : objet graphique qui permet de sélectionner parmi deux états.
- ✓ *SelectMany* : objet graphique qui permet de sélectionner plusieurs éléments parmi un ensemble

✓ *SelectOne* : objet graphique qui permet de sélectionner un seul élément parmi un ensemble.

Règle de Navigation : Les *règles de navigation* permettent de décrire comment les écrans décrits par des objets de type vueIHM s'enchainent entre eux en fonction des traitements. Une règle de navigation expliquent quels sont les éléments logiciels à exécuter pour répondre à un événement (requête web) déclenché à partir d'un écran (lien « de »).

L'application logicielle *ServiceLogicielInteractif* comprend un ensemble de *vueIHM* dans lequel l'utilisateur navigue en fonction des cas applicatifs et de ses interactions. Chaque requête http généré à partir des écrans de l'application est appelé un *événement*. Chaque *événement* fait l'objet d'une règle de navigation permettant de décrire les éléments logiciels à exécuter pour traiter cette requête et répondre à l'utilisateur.
Les règles de navigation sont de plusieurs types en fonction de la complexité des éléments logiciels à assembler pour répondre à l'événement. Une règle de type E-V (Evénement – Vue) permet de stipuler que lorsque l'événement survient, la réponse consiste à exécuter l'objet de type *vueIHM* référencé par le lien « vers ». La règle de navigation consiste alors à passer d'un écran à l'autre sans effectuer de traitement applicatif.
Une règle de navigation de type E-C-V qui signifie Evénement-Contrôleur-Vue. La règle E-C-V spécifie que la réponse à l'événement est d'abord constituer par l'exécution du traitement applicatif à travers un élément logiciel de type *contrôleur* (lien entre règle et contrôleur) et ensuite l'écran a affiché est déterminer en fonction du cas applicatif obtenu par l'exécution du contrôleur (objet de type « forward »). L'élément de type contrôleur matérialise le traitement applicatif qu'il faut exécuter pour répondre à l'événement. Le traitement exécuté par le contrôleur consiste à invoquer une *fonction d'orchestration* qui appartient à la partie « Vue Coordination » du *service_logiciel_interactif*. Le contrôleur récupère de la *VueIHM* les données fournies par l'utilisateur et les passe à la *fonction d'orchestration*. Le résultat de la fonction d'orchestration est transmis au contrôleur. Celui-ci interprète le résultat afin de préparer leur affichage. En effet, ces données seront passées à l'élément de type *VueIHM* pour être affichées à l'utilisateur.
En résumé, la règle de navigation E-C-V représente un traitement applicatif qui donne lieu à un aiguillage sur plusieurs chemins de navigation (« forward ») alors que la règle E-V correspond simplement à un chemin de navigation.

Il existe deux types d'élément de type « forward » : celui qui représente comme prochain élément logiciel directement une vueIHM (« ViewForward ») et celui qui nécessite un traitement applicatif avant d'afficher la vueIHM à l'utilisateur (« EventForward »).

L'objet *contrôleur* permet de faire le pont entre la « vue Interface Utilisateur » et la « vue Coordination ». Il analyse la requête du client et se contente d'appeler le modèle adéquat et de renvoyer la vue correspondante à la demande. Une application logicielle *ServiceLogicielInteractif* utilise généralement plusieurs objets *contrôleurs* dans une application. Le traitement du *contrôleur* consiste à récupérer les données de la requête de l'utilisateur (***entrées***). Ces données servent de ressources pour le processus d'orchestration (fonction d'orchestration) de la « vue coordination ». Ensuite, le contrôleur récupère le résultat de la fonction d'orchestration, ces données représentent souvent les ***sorties*** du contrôleur car elles vont être passées à la *vueIHM* pour être affichée.
Les données de type entrées ou sorties sont décrites par un type de donnée exprimé par un schéma de données XML (XSD).

La partie « Vue interface utilisateur » permet de spécifier les différents éléments logiciels qui rentrent dans l'implémentation de la partie interactive du service logiciel. La section suivante décrit la partie orchestration.

2.2.2. Le méta-modèle des vues Coordination et Service

La *vue coordination* comprend plusieurs activités et structures de contrôles. Les activités sont des tâches de processus telles que les invocations de services ou le gestionnaire de données tandis que les structures de contrôle décrivent les ordres d'exécution des activités pour réaliser un certains but. Le méta-modèle de la vue coordination se fait dans la spécification EMOF (Essentiel MOF) qui est défini selon le standard MOF [EMOF 2004].

Afin d'exploiter la dynamique du web et de prendre en compte les profils utilisateurs, il apparaît nécessaire de permettre une composition adaptable des services. La *vue coordination* prend en charge la synchronisation des services web. Elle reçoit les appels de services web de la part de la vue *Interface Utilisateur* et enclenche les actions à effectuer.
Il ya plusieurs approches pour modéliser la coordination des processus tel que state-charts, block structures [ISM 2003], activity diagrams [OMG 2004], Petri-nets [Van der Aalst et al. 2000], et bien d'autres. En dépit de la diversité dans le la modélisation du flux de contrôle, il

est bien accepté que les langages de modélisation existantes partage cinq patterns de base : la séquence, le parallèle, la synchronisation, le choix exclusif et la fusion simple [Van der Aalst et al. 2003] [Wohed et al. 2002]. Ainsi, nous adoptons ces patterns comme des blocks de construction de notre méta-modèle.

Les structures de contrôle de BPEL [ISM 2003] tel que les séquences, les flux et les itérations sont plus ou moins équivalent aux patterns mentionnés ci-dessus. Au lieu de réinventer un nouveau méta-modèle d'orchestration, nous construisons notre méta-modèle sur les structures de contrôle de BPEL de base et définissons leurs sémantiques plus exactement.

La *vue coordination* du méta-modèle MISI est basée sur les structures du modèle BPEL. BPEL permet de composer des services web. La composition de services web spécifie quels services ont besoin d'être invoqués, dans quel ordre et comment gérer les conditions d'exception. BPEL se base sur l'orchestration. L'orchestration décrit l'interaction des services au niveau de messages, incluant le logique métier et l'ordre d'exécution des interactions. Les services web n'ont pas de connaissance (et n'ont pas besoin de l'avoir) d´être mêlées dans une composition et d'être partie d'un processus métier. Seulement le coordinateur de l'orchestration a besoin de cette connaissance.

Un orchestrateur prend le contrôle de tous les services web impliqués et coordonne l'exécution des différentes opérations des services web qui participent dans le processus.

BPEL est un langage pour les processus métier basé sur XML conçu pour permettre charger/partager les données distribuées, même à travers des organismes multiples, en employant une combinaison de services web. Il décrit l'interaction des processus métiers basés sur les services Web, à la fois au sein des utilisateurs et entre elles. Les utilisateurs du langage BPEL pourront ainsi définir leurs processus métiers et en garantir l'interopérabilité non seulement à l'échelle de l'entreprise, mais également avec leurs partenaires commerciaux, au sein d'un environnement de services Web. BPEL rend possible l'interopérabilité entre des activités commerciales basées sur des technologies différentes.

Le processus BPEL spécifie l'ordre exact de l'invocation des services web participants dans la composition. L'invocation peut se faire soit en parallèle soit séquentiellement. Nous pouvons conditionner le comportement, par exemple, lorsque l'invocation d'un service web peut être dépendante du résultat d'une invocation antérieur.

La construction des boucles, la déclaration de variables, la copie et l'assignement des valeurs, etc. est aussi possible. De plus, il est possible combiner tous ces constructions et de définir des processus métier complexes d'une manière algorithmique [Juric 2005].

Comme nous venons d'indiquer, un processus métier correspond à une séquence d'opérations ou plus exactement à un flux d'activités. Ces activités peuvent faire intervenir un à plusieurs services Web. Le langage BPEL permet deux types d'activités : les activités de base et les activités structurées.

Le méta-modèle de la vue coordination est décrit à la figure 5.5:

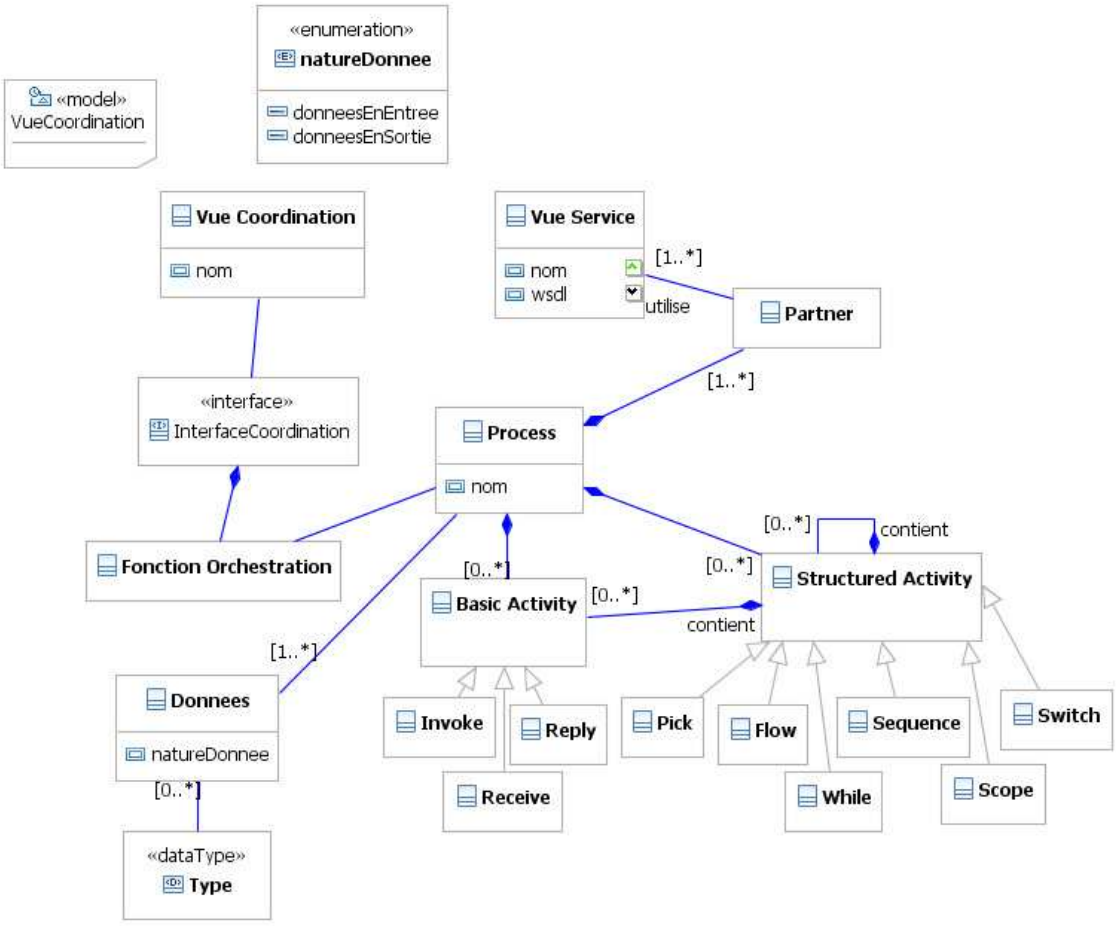


Figure 5.5. : Le méta-modèle de la vue coordination et service

Les activités de base (basic activity) permettent:

- ✓ d'invoquer une opération d'un service Web tiers (activité *invoke*),
- ✓ de présenter la composition comme un nouveau service Web avec l'activité *receive* pour d´écrire la réception d'une requête et l'activité *reply* pour générer une réponse.

Les activités structurées (structured activity) utilisent les activités de base pour décrire :

- ✓ des séquences ordonnées (*sequence*) et des exécutions en parallèle (*flow*),
- ✓ des branchements (*switch*, *if)* et des boucles (*while*),
- ✓ des chemins alternatifs (*pick*).

Il permet aussi de déclarer des variables, avec *donnée*, et de définir des services web, avec *partner*. Nous définissons un partner comme une relation entre un service web et le processus pendant l'invocation d'un service Web ou, aussi, comme une relation créée entre le client qui invoque un processus BPEL et le processus lui même. Il faut, obligatoirement, avoir au moins un client partner.
Le méta-modèle intègre également un mécanisme de gestion des exceptions (*throw*, *catch*), ainsi qu'un mécanisme de compensation (*scope*) qui permet d'annuler une transaction dans son intégralité lorsque celle-ci échoue. Il constitue une couche supérieure au langage de description WSDL. Il utilise, en effet, WSDL pour définir les opérations de services Web élémentaires à appeler et pour présenter le processus métier comme un nouveau service Web.

La *vue service* permet de recenser les web services métier utilisés par le service logiciel interactif. Ces web services sont invoqués par les activités de type « **invoke** » contenus dans la description des processus d'orchestration de la vue coordination ». De plus, le concept de « partner » permet de représenter dans le processus d'orchestration le fournisseur qui héberge le service web invoqué.

Cette deuxième partie de chapitre est consacrée au processus de transformation d'un modèle MOS en un modèle MISI. Le modèle de facettes architecturales est utilisé comme étape préliminaire pour choisir l'architecture et la plateforme d'implémentation du service interactif.

2.2.3. Le processus de passage du modèle MOS au modèle MISI

Le passage du modèle MOS au modèle MISI nécessite une étape intermédiaire permettant de choisir l'architecture et la plateforme d'implémentation du service logiciel.

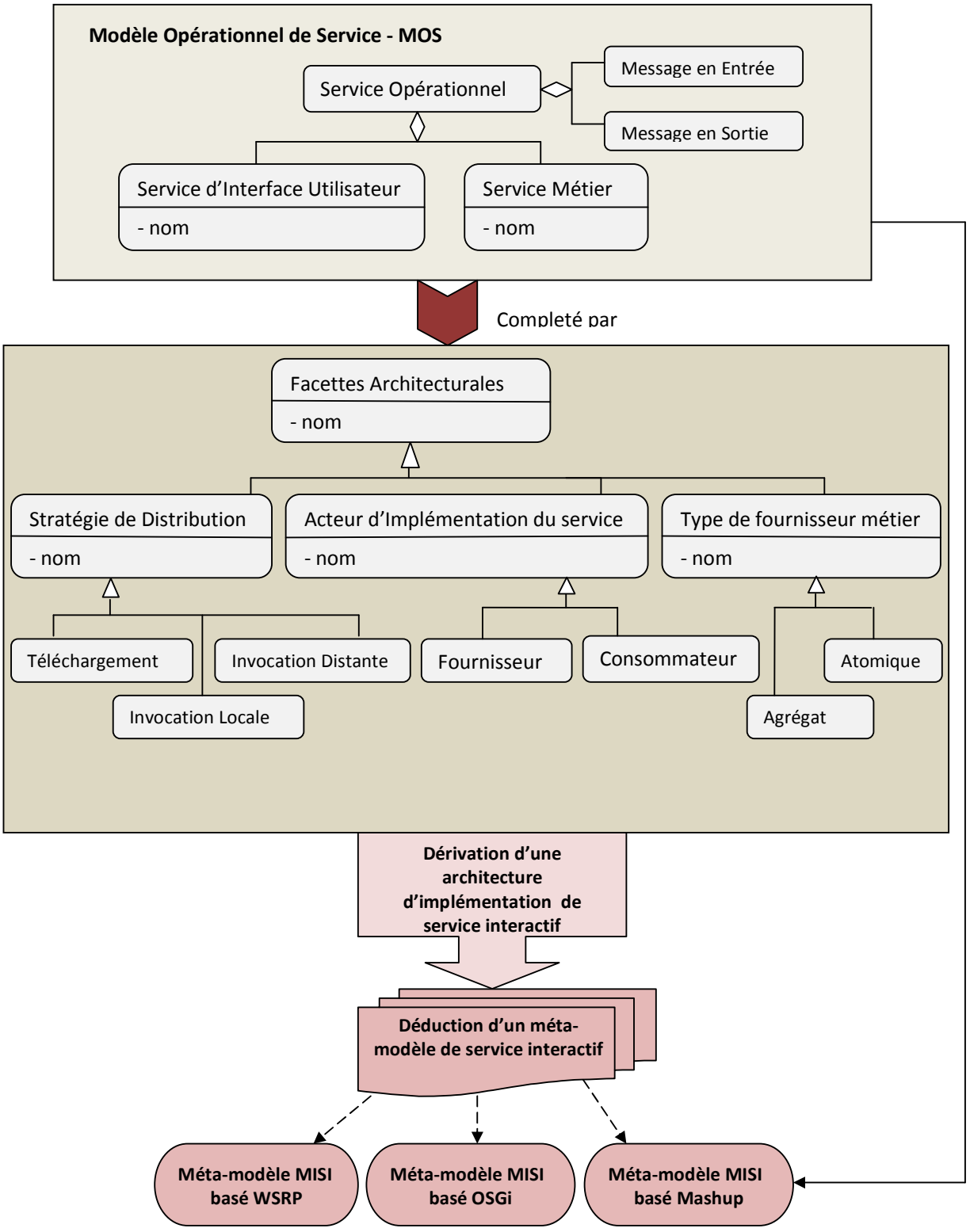


Figure 5.6. : Etape de sélection de l'architecture et plateforme d'implémentation

La figure 5.6 résume le schéma de sélection du méta-modèle de type PSM. Ces méta-modèles sont basés sur un processus de déductions à partir d'un méta-modèle de facettes architecturales. Les dérivations sont nombreuses et vont du plus spécifique pour la détermination de services interactifs comme le méta-modèle MISI du WSRP au moins spécifique pour la détermination de services non interactifs mais ayant la notion de composition de services par le consommateur comme le méta-modèle MISI du Mashup. Le plus spécifique permet d'avoir un modèle de service interactif identique aux objectifs d'implémentation de l'approche de service intentionnel. C'est le cas des architectures orientées services interactifs ayant des fournisseurs de services interactifs qui mettent à la disposition des utilisateurs les services d'interface utilisateur et de coordination. Ces architectures possèdent également des fournisseurs de services web. Le moins spécifique et conforme à notre approche de service intentionnel permettent d'avoir, certes, une composition de services interactifs, mais la partie interactive avec l'utilisateur est développé de manière rapide et aisée par le client. C'est le cas du Mashup qui permet de concevoir une interface utilisateur combinant les données de plusieurs services web. Généralement, lorsque la coordination des services web est chez le client en même temps que le service d'interface utilisateur, la composition est en effet facile à effectuer puisque c'est le client qui effectue la l'orchestration de service. Les différentes architectures sont expliquées dans les sections suivantes. La plateforme d'implémentation est déterminée par les deux facettes : stratégie de distribution du service interactif et acteur implémentant le service interactif. En effet, le tableau suivant donne les correspondances avec les trois plateformes existantes : WSRP, OSGI et Mashup en fonction des valeurs des deux premières facettes.

Tableau 5.1. Tableau de correspondance

Tableau de correspondances		**Facette : stratégie de distribution**		
		Invocation distante	téléchargement	Invocation locale
Facette : acteur d'implémentation du service interactif	consommateur			Mashup
	fournisseur	WSRP	OSGI	Portlet - application web

Dans le tableau de correspondance (cf Tableau 5.1), nous pouvons voir que les deux plateformes WSRP et OSGI permettent de développer des services interactifs, d'avoir un fournisseur qui le publie et un consommateur distant qui le consomme. Néanmoins, WSRP

permet de l'invocation à distance du service interactif alors qu'OSGI a opté pour une stratégie de distribution par téléchargement. Les plateformes de type Mashup disponibles actuellement sur le marché permettent aux clients de services web de développer de manière plus facile des applications graphiques combinant des données provenant de plusieurs services web métier.
La troisième et dernière facette permet de définir si la partie coordination du service métier peut être exposée comme un web service agrégat (valeur *agrégat*) ou au contraire est-elle cachée dans l'implémentation du service interactif (valeur *atomique*).

2.2.4. Les différentes variantes d'architectures d'implémentation possibles

Les différentes architectures d'implémentation sont issues du processus de dérivation à travers le modèle des facettes. Le processus de dérivation est fonction de l'emplacement du service métier et du service d'interface utilisateur. Lorsque le service d'interface utilisateur et le service de coordination sont invocables chez le fournisseur, alors on parle de service interactif ; mais lorsque le service d'interface utilisateur est développé chez le consommateur, alors on parle de composition interactive de service web, dans le dernier cas, nous sommes en présence d'une architecture de service non interactif car le service d'interface utilisateur est développée une seule fois chez le client. La partie interactive du service est dépendante du client qui consomme le service.

Ce processus nous a permis de dériver quatre architectures selon la stratégie de distribution, l'acteur d'implémentation et le type de fournisseur métier.

2.2.4.1. Architecture 3-tier orientée service interactif

L'architecture 3-tier orientée service interactif est le type d'architecture le plus spécifique à la notion d'approche de service interactif. Au sein de l'architecture de service interactif le service interface utilisateur et le service de coordination sont fournis par un fournisseur de service interactif. Le consommateur de service est quant à lui une sorte de portail qui encapsule les différents services qui sont invoqués. Le portail par définition est un agrégateur de services fournis. Les services web fournis également par un fournisseur de services sont coordonnés par un moteur de coordination chez le fournisseur de service interactif. L'architecture de service interactif 3-tiers comprend donc trois acteurs qui sont le consommateur de service, le fournisseur de service d'interface utilisateur et de coordination et le fournisseur de service web.

L'avantage d'une telle architecture est de pouvoir distribuer pour un fournisseur, (1) le service interactif dans sa totalité car il convient à l'environnement du client et (2) le service métier agrégat (coordination) si les clients ont besoin d'une interface utilisateur plus spécifique. Cette architecture est déductible lorsque les facettes suivantes sont :

- Stratégie de distribution : *invocation à distance, téléchargement*,
- Acteur implémentant le service interactif : *fournisseur*
- Type de fournisseur métier : *agrégat*.

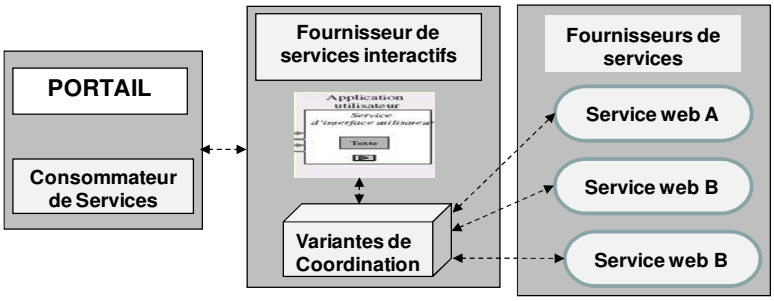


Figure 5.7. : Présentation d'une architecture 3-tier de service interactif

2.2.4.2. Architecture 2-tier orientée service interactif

L'architecture 2-tier de service interactif est quasi semblable à la structure de l'architecture 3-tier de service interactif sauf qu'elle ne possède pas de coordination de service.

Le client ne peut invoquer que le seul fournisseur de service interactif : le service web accompagné de son service d'interface utilisateur. Cette architecture garde également la structure interactive de service. Le client est totalement indépendant du développement des deux services d'interface utilisateur et web.

Cette architecture est déductible avec les mêmes valeurs que l'architecture précédente exceptée que le type de fournisseur métier est *atomique*.

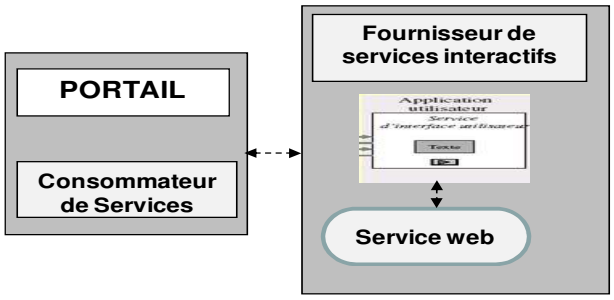


Figure 5.8. : Présentation d'une architecture 2-tier de service interactif

2.2.4.3. Architecture 3-tier orientée service non interactif (composition interactive de services)

Les deux architectures qui suivent ne font pas intervenir une architecture de services interactifs puisque la partie interactive est développée par le client. L'usage des plateformes de type Mashup permettent de rendre plus facile ce type de développement. Dans ce cas, la composition des services web est réalisée au niveau de l'interface utilisateur par le client. Nous appelons cela composition interactive de services webs. L'architecture 3-tier de composition interactive de service est une exposition de trois composants de services qui sont le consommateur de services, le fournisseur de coordination de services et le fournisseur de services web. En résumé, le fournisseur métier expose la coordination de services web et le client développe la partie interface utilisateur. Le site web d'une agence de voyage de la SNCF est un cas d'exemple de cette architecture. Le site web de l'agence est considéré en même temps comme un consommateur de services et un fournisseur de coordination de service.

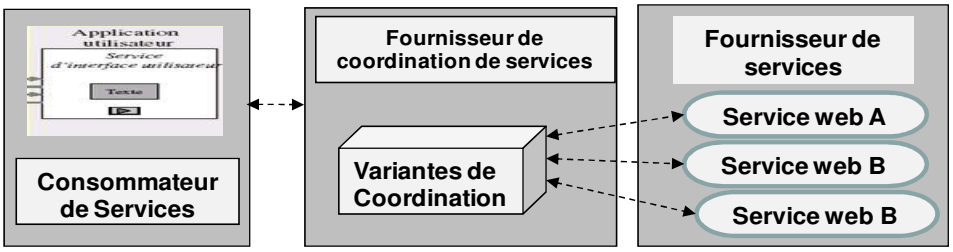


Figure 5.9. : Architecture 3-tier de composition interactive de service

Cette architecture est déductible lorsque les facettes suivantes sont :

- Stratégie de distribution : *invocation locale*,
- Acteur implémentant le service interactif : *consommateur*
- Type de fournisseur métier : *agrégat*.

2.2.4.4. Architecture 2-tier orientée service non interactif (composition interactif de service)

Dans cette architecture de composition interactive de service, la coordination de services n'est pas exposée comme un service web. Il n'y a donc pas de fournisseur de coordination de services. C'est le cas d'exemple traditionnel des technologies à base de mashup. Le client doit connaitre l'ensemble des services web métier atomiques et il prend en charge la partie coordination et la partie interface utilisateur.

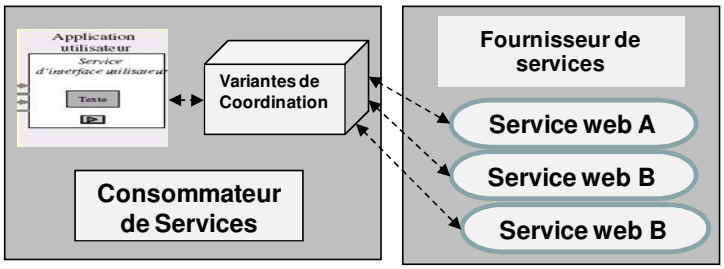


Figure 5.10. : Architecture 2-tier de composition interactive de service

Cette architecture est déductible lorsque les facettes suivantes sont :

- Stratégie de distribution : *invocation locale*,
- Acteur implémentant le service interactif : *consommateur*
- Type de fournisseur métier : *atomique*.

Nous venons d'illustrer comment l'usage des trois facettes architecturales permet de sélectionner l'architecture et la plateforme d'implémentation des services interactifs. Cette étape préliminaire permet également de choisir le méta-modèle de type PSM qu'il faut utiliser pour développer le service interactif. Pour illustrer la transformation du modèle MOS en Modèle de niveau MISI, nous avons choisi la plateforme WSRP et par conséquent le modèle MISI-WSRP.

2.2.4.5. Le processus de transformation d'un modèle MOS en Modèle MISI-WSRP

Le modèle MISI-WSRP permet de décliner deux architectures possibles : 3tier ou 2 tier. Les transformations pour ces deux architectures sont décrites dans cette section.

Le schéma ci-dessous présente le processus de transformation du modèle MOS modèle MISI-WSRP pour une architecture 3 tier. La figure 5.11 permet de décrire les règles de transformation qui sont appliquées.

Concepts du modèle MOS	Facettes architecturaux	Architecture possible	Traduction dans le modèle MISI
Service d'Interface Utilisateur			Vue interface utilisateur invoqué/téléchargé chez le fournisseur
Service Métier	Stratégie de distribution de type *invocation à distance*.		Vue coordination invoqué/télécharger depuis chez le fournisseur
	Stratégie de distribution de type *téléchargement.*		Vue service web invoqué/télécharger chez le fournisseur
Service d'Interface Utilisateur	Acteur d'implémentation de type *Fournisseur*	Architecture 3-tier de service interactif	Vue interface utilisateur implémenté chez le fournisseur de service interactif
Service Métier	Acteur d'implémentation de type *Fournisseur*		Vue coordination implémenté chez le fournisseur de service interactif
	Acteur d'implémentation de type *Fournisseur*		Vue service web implémenté chez le fournisseur.
Service Métier	Type de fournisseur métier *agrégat*		Agrégation

Figure 5.11. : Processus de génération du modèle d'implémentation de système interactif basé WSRP

La figure 5.12 présente le processus de génération du modèle MISI ayant un fournisseur atomique à partir du modèle MOS.

Concepts du modèle MOS	Facettes architecturaux	Architecture possible	Traduction dans le modèle MISI
Service d'Interface Utilisateur			Vue interface utilisateur invoqué/téléchargé chez le fournisseur
Service Métier	Stratégie de distribution de type *invocation à distance*.		Vue coordination invoqué/télécharger depuis chez le fournisseur
	Stratégie de distribution de type *téléchargement.*		Vue service web invoqué/télécharger chez le fournisseur
Service d'Interface Utilisateur	Acteur d'implémentation de type *Fournisseur de service interactif*	Architecture 3-tier de service interactif	Vue interface utilisateur implémenté chez le fournisseur de service interactif
Service Métier	Acteur d'implémentation de type *Fournisseur de service interactif*		Vue coordination implémenté chez le fournisseur de service interactif
	Acteur d'implémentation de type *Fournisseur de service*		Vue service web implémenté chez le fournisseur.
Service Métier	Type de fournisseur de service		Atomique

Figure 5.12. : Processus de génération du modèle d'implémentation de système interactif basé WSRP

2.2.5. Exemple de sélection de l'architecture et du modèle MISI associé

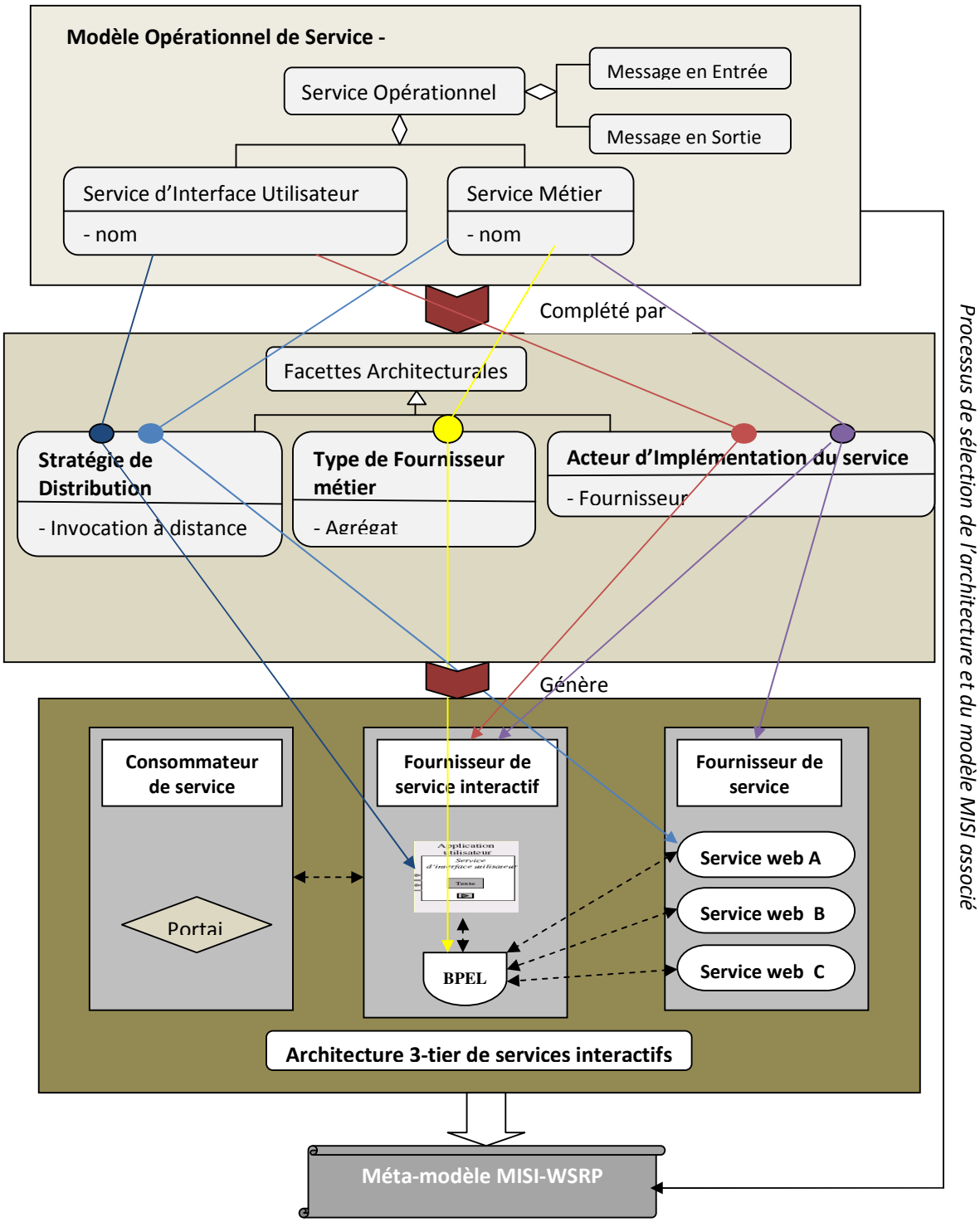


Figure 5.13. : Processus de sélection de l'architecture et du méta-modèle MISI basé WSRP

La figure 5.13 montre qu'à partir du modèle MOS, il faut définir les valeurs des facettes architecturales. Pour l'exemple, nous avons choisi les valeurs suivantes :

- Stratégie de distribution : *invocation à distance*,
- Acteur implémentant le service interactif : *fournisseur*,
- Type de fournisseur métier : *agrégat*.

A partir de ces valeurs de facettes architecturales, l'architecture applicable est une architecture 3 tier orientée service interactif comme le décrit le troisième encadré en noir. Le fournisseur de service interactif est également un fournisseur métier de type agrégat.
La vue coordination est implémentée grâce à la technologie BPEL et la vue interface utilisateur est implémentée comme un « remote portlet » avec la technologie WSRP et enfin la vue « service » est composée des web services utilisés par la vue coordination. Chaque web service est identifié par l'adresse du descripteur WSDL et du web service hébergé chez un fournisseur métier qui est considéré comme atomique.
Le modèle PSM associé à cette architecture est le modèle MISI-WSRP décrit en section 5.3.

3. Règles de transformation pour l'implémentation sur la plateforme de service interactif WSRP

3.1. Introduction

Cette dernière étape de la démarche MeTSI fournit un mécanisme pour traduire les modèles à travers les différents langages de modélisation intégrés. Ce mécanisme est basé sur un ensemble de règles de transformation qui sont générées en utilisant les informations de correspondance obtenues. Ces règles de transformations sont implémentées à travers un langage de transformations de modèle à code. Dans notre cas, nous avons utilisé le langage de transformation ATL : Atlas Transformation Language [Jouault 2006].

3.2. Architecture technique générale

3.2.1. Le choix des langages et outils de transformation

Le choix d'un bon langage de transformation permet de répondre parfaitement à nos exigences.

Durant ces dernières années, la création d'un langage de transformation normalisé est devenue le centre de réflexions autour de MDA [Lopes 2005]. L'OMG a lancé en 2002 un appel à travers son RFP QVT [OMG 2002], pour que des propositions d'un langage de transformation soient faites. Le RFP QVT pour *Query, View and Transformation* n'est pas seulement destiné à la transformation, mais aussi à l'expression des requêtes (*Query*) et la définition de vue (*View*). Plusieurs langages s'inscrivent dans la recommandation QVT. Nous citons *MOLA* (Model transformation LAnguage) [Kalnins *et al. 20*04], *ATL*© (Atlas Transformation Language) [Jouault 2006], *GreAT* [Karsai *et al.* 2003], *AndroMDA* [Andro 2007].

Ces outils de transformation peuvent être classés suivant plusieurs critères : la nature des règles (impératives, déclaratives ou hybrides), la gestion des modèles (textuels ou graphiques), la gestion de la traçabilité des transformations, le fait que l'outil soit libre ou pas, etc. Il est alors possible qu'un langage de transformation soit plus adapté qu'un autre dans un contexte spécifique. En conclusion, Il apparaît difficile de converger vers un langage unique. Nous fixons deux critères liés aux transformations de modèles que nous souhaitons établir : la simplicité d'établissement des règles et la possibilité de gérer du code de service interactif.

ATL© semble répondre à ces deux critères. En plus d'être un outil libre, les règles ATL© sont simples à mettre en œuvre. Son langage à base de variables et d'affectation le rend accessible.

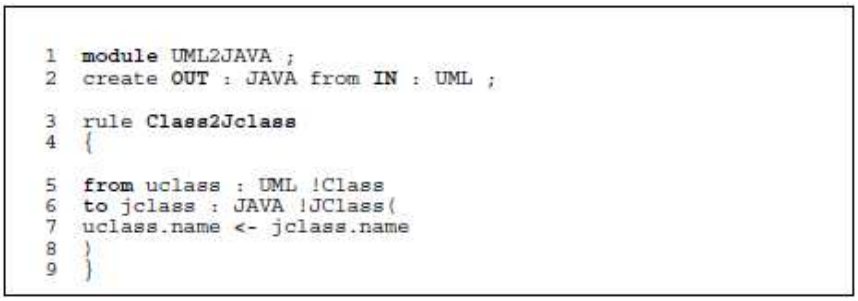
3.2.2. La transformation de modèles avec ATL©

Dans cette section, nous présentons uniquement les caractéristiques de l'outil ATL©, qui nous ont été utiles dans l'implémentation de notre atelier. Une présentation plus détaillée du langage ATL©, est fournie dans [Jouault 2006].

3.2.2.1. Un langage déclaratif et impératif

Une des caractéristiques du langage ATL©, relève de son langage de programmation qui présente la particularité d'être hybride (déclaratif et impératif).

- La partie déclarative permet de faire correspondre directement un élément du méta-modèle source de la transformation avec un élément du méta-modèle cible de la transformation (*matched rule*).



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

3  rule Class2Jclass
4  {

5  from uclass : UML !Class
6  to jclass : JAVA !JClass(
7  uclass.name <- jclass.name
8  )
9  }
```

Figure 5.14. : Une transformation déclarative

- En ATL© une transformation s'appelle *module*. Le mot-clé OUT montre le méta-modèle cible *JAVA* (ligne 2). Le mot clé IN montre le méta-modèle source (*UML*) (ligne 2). La règle (*Class2Jclass*) du fichier de transformation (*UML2JAVA*) est déclarative. L'élément *class* (classe UML) du méta-modèle source va être traduit vers l'élément *jclass* du méta-modèle cible (ligne 6). On précise ensuite en utilisant l'opérateur de liaison←, que l'attribut « *name* » de la nouvelle classe va être égal au nom de la classe UML source (ligne 7).
- La partie impérative d'ATL© complète ces correspondances directes entre éléments. Elle comporte des déclarations conditionnelles (*if…then…else...endif*), des déclarations de variables (*let VarName : varType = initialValue*), des déclarations de boucle (*while* (*condition*)…*do*), etc. Cette partie permet également de manipuler les éléments générés par les règles déclaratives (modification d'attributs, etc.).

3.2.2.2. Gestion de méta-modèles avec ATL©

ATL© doit permettre de gérer les méta-modèles liés à la transformation. Il gère notamment les méta-modèles écrit sous *Ecore* [Budinski *et al. 20*03]. Ce dernier est un langage de métamodélisation qui fait partie d'EMF (*Eclipse Modeling Framework*), résultat des efforts du projet Eclipse (*Eclipse Tools Project*). *Ecore* ressemble dans sa structure à un diagramme de classe UML. Il est basé sur des classes, des attributs, des associations pour lier les classes, des généralisations / spécifications entre classes, etc. *Ecore* présente donc un outil performant pour la réalisation de méta-modèles pour la transformation avec ATL©.

Quelques outils permettent l'édition de méta-modèles conformes à *Ecore,* tels que : *Omondo* ou bien l'éditeur *Ecore* fourni par EMF [Budinski *et al. 20*03].

Une fois les deux méta-modèles de la transformation (source et cible) et le modèle source réalisés, il faut les déclarer en utilisant une fenêtre de configuration de la transformation. La figure (Fig V.6) montre l'utilisation de la fenêtre de configuration pour l'initialisation de la transformation *ServiceCoordination2BPEL.* Les champs du haut *IN* et *OUT* font correspondre les modèles et méta-modèles de la transformation à leurs déclarations dans le code ATL©, alors que le champ du bas *Path Editor* fait correspondre ces déclarations à des fichiers portant une extension *.ecore* .

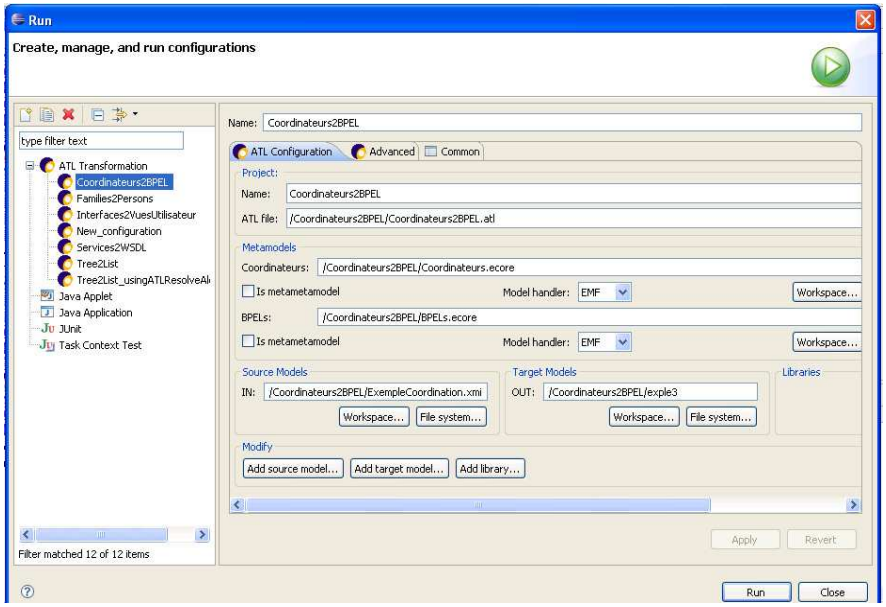


Figure 5.15. : Exécution d'une transformation avec ATL©

3.2.2.3. Structure globale des programmes de transformation

En ATL, une transformation s'appelle un module. Un module contient un en-tête, un ensemble d'importation de bibliothèques de fonctions et un ensemble de fonctions et de règles de transformation. Les fonctions sont appelées helper en ATL.

L'en-tête donne le nom du module de transformation et déclare les modèles source et cible. La figure 5.16 donne un exemple d'en-tête.

```
module Coordinateurs2BPEL; -- Module Template
create OUT : BPELs from IN : Coordinateurs;
```

Figure 5.16 : Entête d'un programme ATL

L'en-tête commence par le mot-clé module suivi du nom du module. Ensuite, les modèles source et cible sont déclarés comme des variables typées par leurs méta-modèles. Le mot-clé *create* indique les modèles *cible*. Le mot-clé *from* indique les modèles *source*. Dans notre

exemple, le *modèle cible* est représenté par le variable *OUT* à partir du *modèle source* représenté par *IN*. Les *modèles source* et *cible* sont respectivement conformes aux méta-modèles BPEL et Coordinateurs. En général, plus d'un modèle source et d'un modèle cible peuvent être listés dans l'en-tête.
Les fonctions et règles de transformation sont les constructions utilisées pour définir une transformation. Elles sont expliquées dans les deux sections suivantes : *les helpers et les règles de transformations*.

3.2.2.4 Les Helpers

Les fonctions ATL sont appelées *helpers* d'après le standard OCL sur lequel ATL se base. OCL définit deux sortes de *helpers* : opération et attribut.

En ATL, un *helper* peut être spécifié dans le contexte d'un type OCL (par exemple String ou Integer) ou d'un type source (venant de l'un des méta-modèles source). Les modèles cible ne sont en effet pas navigables. Les *helpers opération* peuvent être utilisés pour définir des opérations dans le contexte d'un élément de modèle ou du module de transformation. Le rôle principal des *helpers opération* est de réaliser la navigation des modèles source. Ils peuvent avoir des paramètres et peuvent utiliser la récursivité. Les *helpers opération* définis dans le contexte d'éléments de modèles permettent les appels polymorphiques.
Puisque la navigation n'est autorisée que sur les modèles source en lecture seule, une *opération* retourne toujours la même valeur pour un contexte et un ensemble d'arguments donnés.
Les *helpers attribut* sont utilisés pour associer des valeurs nommées en lecture seule sur les éléments de modèles source. Comme les opérations, ils ont un nom, un contexte et un type. La différence est qu'ils ne peuvent pas avoir de paramètre. Leur valeur est définie par une expression OCL. Comme les opérations, les attributs peuvent être définis récursivement avec les mêmes contraintes de terminaison et de cycles.
Les *helpers attribut* peuvent être considérés comme un moyen de décorer les modèles source avant l'exécution de la transformation. La décoration d'un élément de modèle peut dépendre de la déclaration d'autres éléments.

3.2.2.5. Règles de transformation

La règle de transformation est la construction élémentaire en ATL pour exprimer la logique de transformation. Les règles ATL peuvent être soit déclaratives soit impératives. Nous considérons ici les premières, appelées *matched rules*.

Une *matched rule* est composée d'un motif source et d'un motif cible. Le motif source d'une règle définit un ensemble de types source (venant des méta-modèles source) et une garde sous la forme d'une expression OCL booléenne. Un *motif source* est évalué en un ensemble de tuples dans les modèles source. Le *motif cible* est composé d'un ensemble d'éléments. Chacun de ces éléments définit un type cible (venant d'un méta-modèle cible) et un ensemble d'affectations appelées *bindings*. Un *binding* fait référence à une propriété (attribut ou référence) du type et spécifie une expression dont la valeur est utilisée pour initialiser la propriété.

3.3. Implémentation des règles de transformation pour la mise en œuvre de la plateforme de service interactif

Le schéma ci-dessous défini un exemple de règles de transformation qui permet d'implémenter le service logiciel interactif à travers la plateforme WSRP. Les autres éléments de la transformation seront présentés dans le chapitre 6 dans une étude de cas.

```
<config
xmlns : xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns ='http://xml.netbeans.org/schema/Config'>
   <welcome> vueIHM</welcome>
   <model> fonctionMetier </model>
   <rules>
          <rule>
             <event>évènement</event>
             <controller> contrôleur </controller>
             <inputview> vueIHM </inputview>
             <type>règle de navigation</type>
             <forwards>
               <forward name= "évènement" type="v" forward=" vueForward" />
               <forward name= "évènement" type="v"
                forward=" vueFoward" />
               </forwards>
            </rule>
            <rule>
               <event>évènement</event>
               <controller>contrôleur</controller>
                <inputview> vueIHM </inputview>
                <type>règle de navigation</type>
                <forwards>
```

```
                <forward name= ″évènement” type=”v”
                 forward=”vueForward” />
                 <forward name= ″évènement” type=”v” forward=”vueIHM” />
                 <forward name= ″évènement” type=”v” forward=” vueIHM” />
               </forwards>
</config >
```

Figure 5.17. : Extrait des règles de transformations pour la mise en œuvre de la plateforme de service interactif

Le chapitre ci-dessous montre un extrait des règles de transformation pour la coordination des services web en BPEL. Nous présenterons dans un cas d'application au chapitre 6, l'exécution de ces règles.

```
module Coordinateurs2BPEL; -- Module Template
create OUT : BPELs from IN : Coordinateurs;
rule C2P {
        from c: Coordinateurs!Coordination
        to p: BPELs!Process (
                name <- 'Service' +  c.name,
                targetNamespace <- 'http://' + p.name + '.org/',
                abstractProcess <- 'false',
                partnerlink <- pt
                ),
                pt : BPELs!PartnerLink (
                        name <- p.name,
                        partnerLinkType <- plt,
                        myRole <- 'ITF_'+ p.name+'Provider',
                        partnerRole <- ''
                        ),
                        plt: BPELs!PartnerLinkType (
                                name <- 'ITF_' + p.name + 'Link',
                                role <- rl
                                ),
                        rl: BPELs!Role (
                                name <- 'ITF_' + p.name + 'Provider',
                                portType <- 'ITF_' + p.name
                                )
}
```

Figure 5.18. : Extrait des règles de transformation pour la coordination des services web

4. Conclusion

Ce chapitre a décrit les transformations permettant de passer d'un service interactif (MOS) à l'implémentation de ce service interactif. Le processus de transformation est en deux phases:

- de sélection de l'architecture et la plateforme d'implémentation du service interactif. Cette phase permet de choisir le Modèle d'implémentation de service Interactif (MISI) en fonction de la plateforme choisie.
- de transformation du modèle MOS en modèle MISI spécifique (MISI-WSRP dans notre cas).

La première phase est basée sur un méta-modèle de facettes architecturales et des règles de correspondances. Il existe trois plateformes d'implémentation : WSRP, OSGI et Mashup et quatre architectures possibles d'implémentation du service interactif. Pour chacune des plateformes, il y a un modèle MISI spécifique. En effet, les concepts d'implémentation sont différents d'une implémentation à l'autre. Le modèle MISI présenté est le modèle MISI spécifique à la plateforme WSRP.

La deuxième phase permet une transformation de type (PIM-PSM). En effet, le modèle MOS est un modèle indépendant d'une plateforme alors que le modèle MISI-WSRP est un modèle spécifique à une plateforme d'implémentation. Le modèle MISI-WSRP est décrit dans ce chapitre et les transformations nécessaires pour passer du niveau MOS au niveau MISI sont présentées en fonction de l'architecture ciblée (2 tier ou 3 tier).

Enfin, le modèle MISI est un modèle de type PSM suffisamment détaillé pour permettre une génération automatique du code et des descripteurs de déploiement composant l'implémentation du service interactif.

L'ensemble des règles de correspondance de la première phase et les règles de transformation de la deuxième phase sont décrites selon une approche dirigée par les modèles. Par conséquent, les méta-modèles utilisés sont méta-décrits selon le standard EMOF et les règles sont décrites à l'aide du langage ATL afin d'être automatisées par un outil d'IDM.

Le chapitre suivant illustre l'approche MeTSI à travers un exemple.

CHAPITRE 6 : CAS D'APPLICATION

1. Introduction

Ce chapitre présente d'une manière détaillée l'application de l'approche MeTSI sur un cas d'étude, appelé *e-Pension*, dont l'objectif est d'aider les personnes handicapées en Italie à obtenir une pension. Les dirigeants des organisations de traitement des pensions dans ce pays ont pris conscience des difficultés rencontrées au quotidien par les citoyens d'une part, et par les employés des administrations, d'autre part. Pour les réduire, ils souhaitent automatiser le processus global en mettant en place une application qui ouvre la voie des interactions B2B[1] (entre administrations) et B2C[2] (entre administrations et citoyens).

Ce chapitre est organisé comme suit : la section 2 introduit le cas d'étude e-Pension. Ensuite, la section 3 présente une vue d'ensemble de la mise en œuvre du processus méthodologique MeTSI pour générer les services intentionnels. A la section 4, nous donnons un aperçu de la solution du modèle MIS qui a déjà été traité dans [Kaabi 2007]. La section 5 se focalise sur la construction du modèle MOS de l'application qui génère le service opérationnel correspondant. La section 6 illustre le choix de l'architecture et de la plateforme interactive d'implémentation. Le choix du méta-modèle MISI a été porté sur le méta-modèle orientée WSRP. Nous appliquons donc notre application au modèle MISI de la plateforme interactif WSRP. La section 7 présente la génération de code de service interactif du projet *e-Pension*.

2. Présentation du cas d'étude

Cette section présente l'étude de cas dans les grandes lignes et introduit les axes d'évolution que les utilisateurs souhaitent prendre.

[1] *Business-to-Business*
[2] *Business-to-Consumer*

2.1. Cas d'étude

Le cas d'étude utilisé dans ce chapitre, extrait de [Batini 2001], propose une version simplifiée d'un processus qui se déroule en Italie et dont l'intention est d'*aider les personnes handicapées à obtenir une pension*.

A l'origine, l'étude de cas considère qu'un citoyen présentant un handicap peut demander une pension du gouvernement italien. Afin de commencer le processus, la personne a besoin : (*i*) d'une attestation de domiciliation fournie par la mairie de son arrondissement ; et (*ii*) de remplir un formulaire de demande de pension.

Ces documents sont présentés ensuite au *Local Health Authority* (LHA, entité médicale) qui après avoir négocié un rendez-vous avec le citoyen, l'examine et prépare un rapport formulant une pré-décision. Le rapport est ensuite envoyé à la Préfecture qui prend la décision finale. Le citoyen doit entre-temps communiquer à la Préfecture sa demande de pension, suite à quoi, il reçoit un accusé de réception. Dans le cas où la demande de pension serait acceptée, la Préfecture prépare tous les documents nécessaires à l'établissement du dossier de paiement de la pension et les transmet ensuite au citoyen, qui peut alors percevoir sa pension chaque mois.

2.2. Dysfonctionnements

Le processus de demande de pension est un processus long et compliqué. On observe qu'un temps considérable est gaspillé à transmettre des documents d'une administration à une autre (du LHA à la préfecture, par exemple). En outre, il incombe au citoyen de suivre son dossier. De plus, de nombreuses activités doivent être distribuées entre les organisations ce qui pose un problème d'hétérogénéité de données.

L'ensemble de ces problèmes a des conséquences sur la performance des administrations. Le diagnostique de ces dysfonctionnements (voir [Batini 2001] pour plus de détails) montre qu'il serait utile d'automatiser le processus pour aboutir à un état de fonctionnement satisfaisant pour les administrations affectées, afin d'améliorer le rendement en termes de dossiers traités et, finalement, pour améliorer la qualité des services fournis aux citoyens.

2.3. Axes d'évolution

Afin de faciliter le processus d'obtention des pensions, les dirigeants des administrations italiennes souhaitent implanter une solution mettant en œuvre une application qui permettrait la coopération du LHA, de la Préfecture et de la Mairie afin de réaliser l'intention « *Fournir*

une aide aux personnes handicapées » dans des conditions plus aisées. Nous appelons cette application *e- Pension*. La solution proposée a un double objectif :

- Du point de vue des utilisateurs de l'application, l'objectif est de mettre en place une politique orientée client, et non plus orientée produit, afin de mieux satisfaire les exigences des citoyens ;
- Du point de vue des administrations coopératives, l'objectif est de mettre en place une application coopérative, qui ouvre la voie des interactions B2B et B2C, afin de réaliser l'intention qu'ils partagent.

Le deuxième objectif est mis en œuvre par le choix de l'utilisation des services Web comme briques de base de l'application e-Pension. En effet, les services Web garantissent les interactions B2B et B2C. Les interactions entre les différentes administrations ne sont pas figées à l'avance, elles sont au contraire instanciées à la demande. Les relations B2B sont plutôt effectuées en partenariat et les services Web jouent alors le rôle d'intégration entre les différents systèmes d'information. Ainsi un collaborateur du système de gestion des pensions peut consulter le LHA ou la Préfecture au sein d'une application développée en interne, mais fédérant les appels aux systèmes d'information partenaires.

D'autres perspectives sont ouvertes par les services Web. Le Web étant par nature très versatile, l'intégration de systèmes d'information pour le B2B a besoin d'un couplage aussi faible que possible. En effet, la croissance du nombre potentiel de partenaires implique qu'on ne peut pas raisonnablement suivre les mises à jour effectuées par les uns et par les autres. Les services Web permettent justement de se prémunir de ce type de problèmes tant que les interfaces restent stables. La capacité de ces services à se décrire eux-mêmes permet d'envisager l'automatisation de leur intégration.

Enfin, l'application e-Pension voudrait mettre en place une solution d'entreprise automatisée *intelligente,* dans le sens où elle pourrait prendre en charge certaines opérations fastidieuses pour un citoyen. Par exemple, le système d'information e-Pension possédant un service de gestion des pensions pourrait transmettre automatiquement une demande de pension auprès de ses partenaires en vue d'une étude. Pour cela, il suffirait d'employer les services Web proposés par les partenaires. En retour, les services des partenaires vont pouvoir effectuer automatiquement l'envoi d'une décision auprès du service de paiement. A la réception de la décision, le système d'information pourra notifier le service de traitement des pensions du suivi du paiement.

Ceci illustre comment les services Web peuvent automatiser nombre de tâches tout en limitant le couplage des systèmes d'information ainsi que leur complexité.

3. Vue d'ensemble de la mise en œuvre du processus méthodologique.

Nous avons choisi l'étude de cas de l'e-Pension pour la mise en œuvre la démarche méthodologique proposée dans cette thèse afin de réaliser une application orientée services en partant des besoins et des attentes des citoyens. La démarche appliquée à l'exemple met en œuvre quatre étapes, à savoir :

- Construction du modèle MIS, décris précédemment dans [Kaabi 2007] ;
- Construction du modèle MOS ;
- Construction du modèle MISI ;
- Génération du code relatif à la plateforme de services interactifs WSRP qui permet d'exécuter les services invoqués.

Le modèle MIS représente les services qui exhibent une intentionnalité formulée par l'*intention* qu'ils permettent à ses clients d'atteindre dans une perspective intentionnelle.

Pour construire le modèle MIS, nous nous basons sur le modèle de la Carte pour la capture des besoins des utilisateurs d'e-Pension ainsi que d'un ensemble de règles. Dans cette partie, la carte e-Pension, ainsi que le modèle MIS qui en découle, ont été extraits de [Kaabi 2007].

La seconde étape du processus consiste à dériver le modèle MOS à partir du modèle MIS. Le modèle MOS décrit, à un niveau opérationnel, le service logiciel interactif qui opérationnalise le service intentionnel atomique du modèle MIS. Dans cette optique, un service logiciel interactif constitue une entité opérationnelle ayant une responsabilité spécifique et décrite indépendamment d'une plateforme spécifique. Le passage du modèle MIS au modèle MOS se fait de manière sémantique à travers le modèle de scénario.

L'étape suivante consiste à passer du modèle MOS au modèle MISI. Le modèle MISI permet de guider l'identification, au niveau logiciel, de services qui correspondent aux services opérationnels de type MOS. Ceci est réalisé à l'aide de la représentation explicite des services logiciels qui découlent des services opérationnels dans les termes d'un modèle spécifique MISI (Modèle d'Implémentation de Service Interactif) et un ensemble de règles

méthodologiques prenant en compte des paramètres caractérisant les différents plateformes de services interactifs.

Le processus de transformation s'appuie sur des règles spécifiques à chaque plateforme. Ces règles sont automatisées. Dans ce travail, nous avons concrètement utilisé la plateforme de service WSRP qui permet d'exécuter les services interactifs invoqués.

La section suivante montre de manière détaillée l'application du processus pour construire l'application orientée de services e-Pension.

4. Construire le modèle MIS

La construction du modèle MIS met en œuvre un ensemble d'étapes méthodologiques. Ces étapes ont été prises en compte dans l'élaboration de l'application e-Pension. La construction du modèle MIS a été traitée dans [Kaabi 2007]. Nous nous aidons du résultat pour construire le modèle MOS en nous basant sur le modèle de scénario.

La première étape dans la construction du modèle MIS consiste à construire le modèle de la carte pour la capture des besoins des utilisateurs d'e-Pension. Le modèle de la Carte exprime les besoins des utilisateurs par des intentions et des stratégies pour les atteindre. La seconde étape consiste à identifier les services intentionnels à partir du modèle de la carte e-Pension.

Afin de comprendre le début de notre cas d'application, nous intégrons quelques extraits du modèle MIS du cas e-Pension traité dans [Kaabi 2007]. Nous ne traiterons pas tout le modèle MIS correspondant à ce cas d'étude dans cette thèse.

Dans la suite de cette section, nous présenterons la carte pour la capture des besoins des utilisateurs d'e-pension et la description des services du modèle MIS.

4.1. Le modèle de capture des besoins : carte e-Pension

La première étape de l'approche MeTSI consiste à analyser les besoins des utilisateurs concernant un certain processus métier. Dans la présente étude de cas, le processus vise à aider les personnes handicapées à obtenir une pension, suivant les démarches mises en place par le gouvernement Italien. Ce processus est décrit en termes intentionnels, à travers le formalisme de la Carte. La description intentionnelle du processus ainsi produite est représentée par une carte nommée e- Pension.

Comme indiqué précédemment, l'objectif du processus s'étend à aider les personnes handicapées à obtenir une pension. Les deux intentions principales assignées au processus sont alors *Formuler la demande* et *Statuer sur la demande*. La carte de la Figure 6.1 décrit la manière de procéder pour réaliser les objectifs identifiés. Comme le montre la carte *e-Pension*, un ensemble d'intentions sont identifiées, ainsi qu'un ensemble de stratégies. Les premières doivent ou peuvent être accomplies afin d'aider les personnes handicapées à obtenir une pension, tandis que les secondes définissent la manière d'accomplir ces intentions.

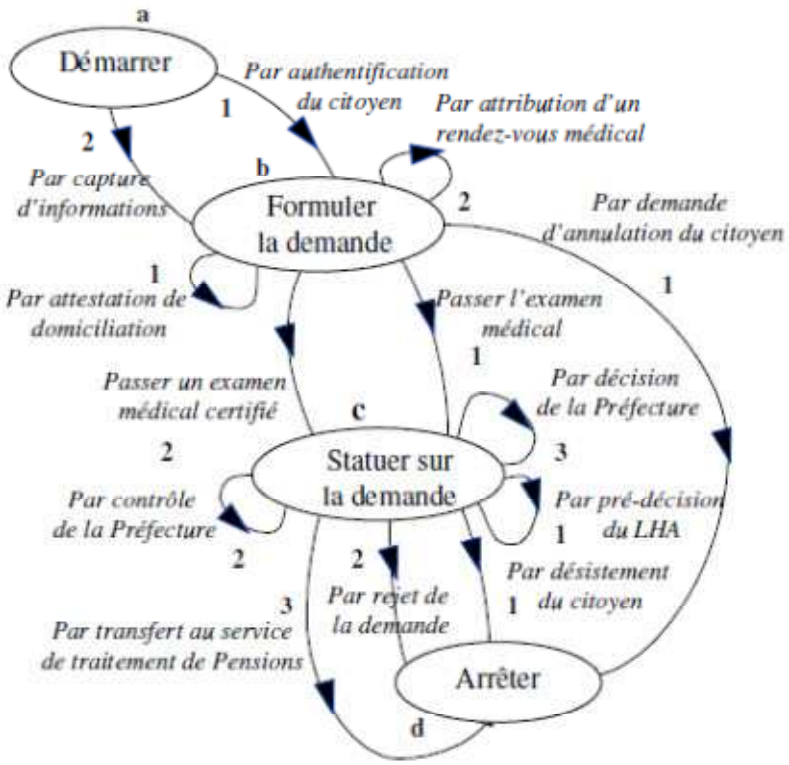


Figure 6.1. La Carte *e-Pension*

Les quatre intentions de la carte e-Pension (*Démarrer*, *Formuler la demande*, *Statuer sur la demande* et *Arrêter*) sont détaillées ci-dessous :

- *Démarrer* : indique le point de départ de l'application du processus ;
- *Formuler la demande* : fait référence à la formulation de la requête de demande d'une pension ;
- *Statuer sur la demande* : correspond à la prise de décision relative à la demande de-pension en question ;

- *Arrêter* : le processus d'aide des personnes handicapées peut se terminer par une demande d'annulation ou de désistement du citoyen, par rejet de la demande de pension ou par transfert de la demande au service de traitement des pensions.

4.2. Description des services du modèle MIS

A partir de la carte e-Pension (Figure 6.1), plusieurs services intentionnels atomiques et agrégats peuvent être identifiés. Chacun de ces services correspond à un fragment de la carte, comprenant des intentions et au moins une stratégie les reliant.

La Figure 6.2 montre un exemple de description formulée en XML (*e**X**tensible **M**arkup **L**anguage*) du service atomique qui permet au citoyen de s'authentifier S $_{Authentifier\ le\ citoyen}$.

```
<Service_Intentionnel Code = S Authentifier le citoyen >
    <Service_Atomique>
      <Interface>
        <But> Authentifier le citoyen </But>
        <Situation_Initiale> citoyen </Situation_Initiale>
        <Situation_Finale> notification </Situation_Finale>
      </Interface>
      <Comportement>
        <Pre_condition> --</Pre_condition>
        <Post_condition> citoyen.état= 'authentifié' </Post_condition>
      </Comportement>
      <Commentaire> Ce service permet sécuriser l'accès aux données
      personnelles des citoyens </Commentaire>
    </Service_Atomique>
</Service_Intentionnel>
```

Figure 6.2. Description du service atomique S *Authentifier le citoyen*

La Figure 6.3 montre un exemple de description formulée en XML d'un second service atomique S $_{Formuler\ la\ demande\ par\ capture\ d'informations}$, lequel permet de capturer les informations personnelles constituant la requête du citoyen. Nous pouvons observer, dans la Figure 6.3, les différents éléments qui caractérisent un service intentionnel atomique, à savoir l'intention à satisfaire, ses situations initiale et finale, ainsi que ses pré- et post-conditions. Ces conditions portent, dans le cas de l'application e-Pension, sur l'état de l'objet *citoyen*, qui doit être *authentifié* au départ, et de l'objet *demande*, qui passe à l'état *créé* suite à ce service.

```
<Service_Intentionnel Code = S Formuler la demande par capture d'informations >
        <Service_Atomique>
          <Interface>
            <But> Formuler la demande par capture d'informations </But>
              <Situation_Initiale> citoyen </Situation_Initiale>
              <Situation_Finale> demande </Situation_Finale>
          </Interface>
          <Comportement>
            <Pre_condition> citoyen.état= 'authentifié' </Pre_condition>
            <Post_condition> demande.état= 'créée' </Post_condition>
            <Regle_transition>citoyen.état='authentifié' Λ demande.état='créée'
            </Regle_transition>
          </Comportement>
            <Commentaire> Ce service permet de capturer les informations
             personnelles constituant la requête du citoyen </Commentaire>
        </Service_Atomique>
</Service_Intentionnel>
```

Figure 6.3. Description du service atomique S *Formuler la demande par capture d'informations*

La figure 6.4 montre un exemple de description formulée en XML d'un service agrégat à choix multiple. Celui-ci, appelé S *Formuler la demande de pension*, permet de formuler une demande de pension en proposant d'authentifier le citoyen et ensuite l'aider à saisir les informations personnelles constituant sa requête. Il est défini donc autour de l'intention *Formuler la demande de pension* et, outre la définition des situations et des conditions, il compte aussi la description de sa composition, avec la nature de la composition (service à variation) et le pont de variation.

```
<Service_Intentionnel code = S Formuler la demande de pension >
      <Service_Agrégat>
        <Service_Variation>
          <Interface>
            <But> Formuler la demande de pension </But>
            <Situation_Initiale> citoyen</Situation_Initiale>
            <Situation_Finale> demande </Situation_Finale>
          </Interface>
          <Comportement>
            <Pre_condition> demande.état= 'créée' </Pre_condition>
            <Post_condition> demande.état= 'formulée' </Post_condition>
```

<Regle_transition>demande.état='créée'Ùdemande.état= 'formulée'
</Regle_transition>
</Comportement>
<Commentaire> Ce service permet de formuler une demande de pension en proposant d'authentifier le citoyen et ensuite l'aider à saisir les informations personnelles constituant sa requête </Commentaire>
<Point_Variation> multiple </Point_Variation>
</Service_Composant Ref="S Authentifier le citoyen, S Formuler la demande par capture d'informations ">
</Service_Variation>
</Service_Agrégat>
</Service_Intentionnel>

Figure 6.4. Description du service à choix multiple S *Formuler la demande de pension*

Le processus de traitement complet du modèle MIS se trouve dans [Kaabi 2007]. Nous nous basons sur ce modèle MIS pour la première étape de l'étude de cas applicatif *e-Pension*. La suite consiste à appliquer à ce cas d'étude les autres modèles de notre approche MeTSI. L'étape suivante consiste alors à construire le modèle MOS de l'application e-Pension à partir des services intentionnels représentés à l'aide du modèle MIS.

5. Construction de l'application E-Pension orientée services du modèle MOS

La construction du modèle MOS de l'application orientée services *e-Pension* consiste à mettre en œuvre la démarche méthodologique expliquée dans le chapitre 4. Elle consiste à guider l'identification, au niveau opérationnel, des services du modèle MOS, qui opérationnalisent de chaque service intentionnel atomique du modèle MIS.

La démarche de construction du modèle MOS comporte deux étapes :

- Etape 1 : Ecrire le scénario de base et découvrir les exceptions,
- Etape 2 : Construire le modèle MOS par analyse des scénarios et,

Les étapes de la démarche ont été prises en compte dans l'élaboration de l'application e-Pension. Parmi les services intentionnels identifiés par [Kaabi 2007] pour l'étude de cas e-Pension, nous avons choisi le service intentionnel atomique S *Attribuer un rendez-vous médical* pour illustrer le déroulement de chacune de ces étapes.

Le déroulement de chacune de ces sous étapes, appliqué au service S *Attribuer un rendez-vous médical* de e-Pension, est détaillé dans les sections ci-après de la manière suivante : La section 5.1 illustre la construction des scénarios permettant de décrire le comportement d'un service atomique dans la satisfaction de l'intention associée ; La section 5.2 illustre la découverte des éléments constituant le service logiciel du modèle MOS à partir des scénarios obtenus (scénario de base et scénarios alternatifs de succès ou d'échecs).

5.1. Ecrire le scénario de base et découvrir les exceptions

Considérons le service intentionnel atomique S *Attribuer un rendez-vous médical* ayant comme intention *Attribuer* un *rendez-vous médical*. Pour la production des scénarios correspondants à ce service, nous nous basons sur les directives de style et celles de contenu présentées au chapitre 4. Ces directives guident notamment l'identification et l'écrire des flux d'actions. Puis, nous nous servons de l'Ecritoire pour aboutir au scénario conceptualisé suivant :

Expression textuelle du scénario S *Attribuer un rendez-vous médical*

1. Le citoyen formule une demande comportant des dates de rendez-vous médical au système e-Pension.
2. E-Pension enregistre la demande d'attribution de rendez-vous médical.
3. E-Pension demande une liste de rendez-vous au LHA.
4. Le LHA vérifie les disponibilités
5. S'il existe des disponibilités alors
    6. Le LHA confirme la liste de rendez-vous à e-Pension.
    7. E-Pension notifie la liste de rendez-vous au citoyen.
    8. Le citoyen choisit une date de rendez-vous à e-Pension.
    9. E-Pension confirme le rendez-vous au LHA.
    10. Le LHA met à jour la base des rendez-vous
    11. Le LHA envoie une confirmation du rendez-vous à e-Pension.
    12. E-Pension envoie la confirmation de rendez-vous au citoyen

Figure 6.5. Le scénario relatif au service S *Attribuer un rendez-vous médical*

5.1.1. Recherche d'exception

La découverte des scénarios alternatifs à partir d'un scénario de base permet de trouver les nouveaux scénarios qui représentent des manières alternatives de réalisation de l'intention

associée au service atomique. Les scénarios ainsi découverts sont liés par un lien 'OU' à l'intention du service atomique.

La découverte de scénarios alternatifs à partir du scénario associé au service S *Attribuer un rendez-vous médical* consiste à :

- Explorer la description du scénario pour identifier les conditions imbriquées ;
- Trouver les imbrications alternatives possibles ; et
- Sélectionner les imbrications et associer à chacune d'entre elles une manière spécifique afin d'identifier de nouveaux scénarios.

Dans le scénario associé au service S *Attribuer un rendez-vous médical*, il y a une condition imbriquée mise en gras dans le texte du scénario :

Expression textuelle du scénario S *Attribuer un rendez-vous médical*

1. Le citoyen formule une demande comportant des dates de rendez-vous médical au système e-Pension.
2. E-Pension enregistre la demande d'attribution de rendez-vous médical.
3. E-Pension demande une liste de rendez-vous au LHA.
4. Le LHA vérifie les disponibilités

5. S'il existe des disponibilités alors

6. Le LHA confirme la liste de rendez-vous à e-Pension.
7. E-Pension notifie la liste de rendez-vous au citoyen.
8. Le citoyen choisit une date de rendez-vous à e-Pension.
9. E-Pension confirme le rendez-vous au LHA.
10. Le LHA met à jour la base des rendez-vous
11. Le LHA envoie une confirmation du rendez-vous à e-Pension.
12. E-Pension envoie la confirmation de rendez-vous au citoyen

Nous pouvons alors suggérer la construction des différentes combinaisons possibles avec les négations des conditions. Chaque imbrication générée à partir des négations des conditions identifie une manière alternative de satisfaction ou non satisfaction d'une intention.

Pour la négation de condition identifiée, il faut déterminer s'il s'agit d'un cas de succès ou d'échec. La manière correspondante au cas identifié doit permettre cela.

Manière	Type de cas
Il n'existe pas de disponibilités	Echec

Le résultat obtenu indique la nécessité d'un nouveau scénario alternatif d'échec lorsqu'il est impossible de satisfaire l'intention *Attribuer un rendez-vous médical.*

Le scénario suivant (cf. Figure 6.6) est le scénario associé à l'intention *Attribuer un rendez-vous médical* dans le cas où il n'existerait pas de disponibilités de rendez-vous. Les interactions notées en italique proviennent du scénario de la Figure 6.5. Les interactions notées en gras sont spécifiques à ce scénario exceptionnel.

Expression textuelle du scénario S ***Attribuer un rendez-vous médical***

1. Le citoyen formule une demande d'attribution d'un rendez-vous médical au système e-Pension.

2. E-Pension enregistre la demande d'attribution de rendez-vous médical.

3. E-Pension demande rendez-vous au LHA.

4. Le LHA vérifie les disponibilités

5. S'il n'existe pas de disponibilités alors

6. E-Pension demande au citoyen de reformuler sa demande.

Figure 6.6. Le scénario alternatif relatif au service S ***Attribuer un rendez-vous médical***

5.2. Construire le modèle MOS par analyse des scénarios

L'objectif de cette étape est de découvrir les éléments constituant le service logiciel du modèle MOS à partir des scénarios obtenus lors de l'étape précédente (Ecrire le scénario de base et découvrir les exceptions).

La génération du modèle MOS, à partir des scénarios, est réalisée par l'application des deux sous étapes :

- Etape 1 : Construire le modèle MOS par analyse du scénario de base et ;
- Etape 2 : Construire le modèle MOS par analyse des scénarios alternatifs d'échec.

Les prochaines sections décrivent l'application de chacune de ces sous-étapes pour le service intentionnel S *Attribuer un rendez-vous médical*.

5.2.1. Construire le modèle MOS par analyse du scénario de base

La génération du modèle MOS, à partir du scénario de base associé au service intentionnel S *Attribuer un rendez-vous médical*, consiste à identifier les différents éléments qui composent le modèle MOS à partir des renseignements mis en évidence par le scénario. Ceci est réalisé par application des quatre sous étapes décrites dans les sections suivantes :

- Spécialiser les agents du scénario ;
- Modéliser le service d'interface utilisateur ;
- Découvrir le service métier ;
- Découvrir les opérations et les ressources pour l'invocation du service métier

a. Spécialiser les agents du scénario

L'analyse du scénario de base relatif au service S *Attribuer un rendez-vous médical* aboutit à identification de trois agents :

- Le *Citoyen* correspond au *Demandeur* du service ;
- L'application *e-Pension* est découpée en deux rôles principaux : *Interface* et *Fournisseur principal* ;
- LHA correspond au *Fournisseur secondaire* du service.

Le Citoyen dépend d'e-Pension pour la formulation de la demande d'un rendez-vous médical. L'e-Pension, de son coté, dépend du LHA pour lui fournir le service demandé par le Citoyen. Il dépend aussi du Citoyen pour la décision d'acceptation ou de refus du rendez-vous médical demandé. Enfin, le LHA dépend d'e-Pension pour l'acceptation ou le refus du rendez-vous choisi. Le Citoyen est donc l'utilisateur de l'application. La prochaine étape se concentre sur cet acteur afin d'identifier les services d'interface nécessaire à son interaction avec l'application.

b. Découverte du Service d'Interface Utilisateur du modèle MOS à partir du modèle MIS

La modélisation du service d'interface utilisateur, à partir du scénario S *Attribuer un rendez-vous médical*, est guidée par un ensemble d'étapes méthodologiques. Ces étapes sont présentées en détail au Chapitre 4.

- L'étape 1 consiste à extraire les communications utilisateur à partir du scénario de base S *Attribuer un rendez-vous médical*. On distingue quatre actions numérotées 1, 7, 9 et 12 dans le scénario de la Figure 6.5 :

1. *Le citoyen formule une demande d'attribution d'un rendez-vous médical au système e-Pension.*
7. *E-Pension notifie la liste de rendez-vous au citoyen.*
9. *Le citoyen confirme le rendez-vous à e-Pension.*
12. *E-Pension envoie la confirmation de rendez-vous au citoyen.*

Ces quatre communications utilisateur représentent les interactions de base qui s'effectuent entre le demandeur et le service d'interface utilisateur.

Les interactions 1 et 9 ont comme agent source le citoyen qui a le rôle de *Demandeur*. Dans ce cas, on considère que ces interactions sont de type *"entrée"*. Par contre, les interactions 7 et 12 sont de type *"sortie"*(le citoyen en est le destinateur).

- L'étape 2 permet de confirmer, à partir des communications utilisateur, les rôles de *Demandeur*, *Interface* et *Fournisseur principal*.

En appliquant l'étape 2 aux communications utilisateur du scénario S *Attribuer un rendez-vous médical*, nous identifions deux agents : le citoyen qui joue le rôle de *Demandeur* et e-Pension qui joue à la fois le rôle *d'Interface* et de *Fournisseur principal*.

- L'étape 3 vise à identifier, à partir des communications utilisateur, les interactions de base, à savoir les interactions utilisateur de type ″entrée″ et ″sortie″, ainsi que les interactions métier de type ″demande utilisateur″ et ″réponse utilisateur″.

L'application de l'étape 3 aux quatre communications utilisateur permet d'identifier les interactions de base du service d'interface utilisateur suivantes (cf. Tableau 6.1) :

Tableau 6.1. Liste des interactions de base

N°	Interaction	Agent source	Agent cible	Type	Sens
1	FormuleDemandeRDVMédical	Citoyen	e-Pension (Interface)	utilisateur	entrée
	EnvoieDemandeRDVMédical	e-Pension (Interface)	e-Pension (Fournisseur principal)	métier	demande utilisateur
	ReçoitListeRDV	e-Pension (Fournisseur principal)	e-Pension (Interface)	métier	réponse utilisateur
7	NotifieListeRDV	e-Pension (Interface)	Citoyen	utilisateur	sortie
9	SélectionRDV	Citoyen	e-Pension (Interface)	utilisateur	entrée
	EnvoieRDVSelectionné	e-Pension	e-Pension	métier	demande

		(Interface)	(Fournisseur principal)		utilisateur
	RéçoitAccuséReception	e-Pension (Fournisseur principal)	e-Pension (Interface)	métier	réponse utilisateur
12	AfficheAccuséReception	e-Pension (Interface)	Citoyen	utilisateur	Sortie

A ce stade, il est possible d'identifier dans un service d'interface utilisateur (i) l'interaction structurée de type séquence, et (ii) les interactions de base qui la composent. Par exemple, comme nous pouvons observer dans le Tableau 6.1, l'action n° 1 est, en réalité, une interaction structurée, formée par la séquence de plusieurs interactions (*FormuleDemandeRDVMédical*, *EnvoieDemandeRDVMédical* et *ReçoitListeRDV*).

- L'étape 4 aide à compléter chaque interaction de base par l'identification des ressources échangées. Ceci est réalisé en associant à chaque interaction, les ressources attachées aux communications utilisateur.

En appliquant l'étape 4 aux communications utilisateur du scénario S *Attribuer un rendez-vous médical*, nous obtenons la liste des ressources présentées au Tableau 6.2.

Tableau 6.2. Liste des ressources dans un service d'interface utilisateur

Communication utilisateur	Ressources identifiées
1. Le citoyen formule une demande d'attribution d'un rendez-vous médical au système e-Pension	Demander_rendez_vous_médical (nomPatient, prénomPatient, dateRdv)
6. Le LHA confirme la liste de rendez-vous à e-Pension.	Liste_rendez_vous (dateRdv, établisement, num_tel, nom_medecin)
9. Le citoyen confirme le rendez-vous à e- Pension.	Rendez_vous (dateRdv, établisement, num_tel, nom_medecin)
12. Le LHA envoie une confirmation du rendez-vous à e-Pension.	Confirmation_rendez_vous

Le Tableau 6.3 présente les valeurs requises par le modèle MOS pour chacune des interactions de base du service d'interface utilisateur.

Tableau 6.3. Valeurs requises par MOS pour chacune des interactions

N°	Interaction	Agent source	Agent cible	Type	Sens	Ressource
1	FormuleDemandeRDV Médical	Citoyen	e-Pension (Interface)	utilisateur	entrée	Demande_ Rendez Vous
	EnvoieDemandeRDV Médical	e-Pension (Interface)	e-Pension (Fournisseur principal)	métier	demande utilisateur	Demande_ Rendez-Vous

	ReçoitListeRDV	e-Pension (Fournisseur principal)	e-Pension (Interface)	métier	réponse utilisateur	Liste (Rendez_vous)
7	NotifieListeRDV	e-Pension (Interface)	Citoyen	utilisateur	sortie	Liste (Rendez_vous)
9	SélectionRDV	Citoyen	e- Pension (Interface)	utilisateur	entrée	Rendez_Vous_ Séléctionné
	EnvoieRDVSelectionné	e-Pension (Interface)	e-Pension (Fournisseur principal)	métier	demande utilisateur	Rendez_Vous_ Séléctionné
	RéçoitAccuséReception	e-Pension (Fournisseur)	e-Pension (Interface)	métier	réponse utilisateur	Confirmer_ Rendez_Vous
13	AfficheAccuséReception	e-Pension (Interface)	Citoyen	utilisateur	sortie	Confirmer_ Rendez_Vous

- L'étape 5 (voir chapitre 4) sert à compléter la modélisation du service d'interface utilisateur en identifiant les interactions structurées de type contrainte. Celle-ci commence par l'extraction des cas de contrainte existant dans le scénario.

5. *S'il existe des disponibilités (C1)*
8. *Si le citoyen accepte un rendez-vous (C2)*

Chacune des contraintes est, potentiellement, une contrainte qu'il faut positionner dans les interactions de base du service d'interface utilisateur compte tenu de leur impact. Le Tableau 6.4 dispose les contraintes par rapport aux interactions de base.

Tableau 6.4. Disposition des contraintes par rapport aux interactions de base

N°	Interaction	Agent source	Agent cible	Type	Sens	Ressource
1	FormuleDemandeRDV Médical	Citoyen	e-Pension (Interface)	utilisateur	entrée	Demande_ Rendez Vous
	EnvoieDemandeRDV Médical	e-Pension (Interface)	e-Pension (Fournisseur principal)	métier	demande utilisateur	Demande_ Rendez-Vous
	Contrainte C1	**Contrainte**				
	ReçoitListeRDV	e-Pension (Fournisseur principal)	e-Pension (Interface)	métier	réponse utilisateur	Liste (Rendez_vous)
7	NotifieListeRDV	e-Pension (Interface)	Citoyen	utilisateur	sortie	Liste (Rendez_vous)
9	**Contrainte C2**	**Contrainte**				
	SélectionRDV	Citoyen	e- Pension (Interface)	utilisateur	entrée	Rendez_Vous_ Séléctionné
	EnvoieRDVSelectionné	e-Pension (Interface)	e-Pension (Fournisseur principal)	métier	demande utilisateur	Rendez_Vous_ Séléctionné
	RéçoitAccuséReception	e-Pension (Fournisseur principal)	e-Pension (Interface)	métier	réponse utilisateur	Confirmer_ Rendez_Vous
13	AfficheAccuséReception	e-Pension (Interface)	Citoyen	utilisateur	sortie	Confirmer_ Rendez_Vous

c. Découverte du Service Métier du modèle MOS à partir du modèle MIS

La modélisation du service métier, à partir du scénario S *Attribuer un rendez-vous médical*, est également guidée par un ensemble d'étapes méthodologiques. Ces étapes sont présentées en détail au Chapitre 4. Chacune de ces étapes permet d'identifier un des aspects de la description d'un service métier dans e-Pension selon les termes du modèle MOS.

- L'étape 1 (des directives de transformation) consiste à extraire, à partir du scénario, les actions de communication qui ont comme source ou cible un agent qui a le rôle de Fournisseur principal.

Les communications Fournisseur principal correspondent aux actions échangées à partir de et vers le Fournisseur principal pour réaliser l'objectif de la coordination. Par conséquent, les actions de communication correspondent aux activités de base du service métier (coordination de service).

Le Tableau 6.5 résume les communications Fournisseur principal relatives au scénario S *Attribuer un rendez-vous médical* et les actions de base correspondantes.

Tableau 6.5. Liste des communications et leurs activités de base

Liste des communications Fournisseur principal	Activité de base
1. Le citoyen formule une demande d'attribution d'un rendez-vous médical au système e-Pension	FormuleDemandeRDVMédical
2. E-Pension enregistre la demande d'attribution de rendez-vous médical	Activité interne
3. E-Pension demande rendez-vous au LHA	Demander rendez-vous
6. Le LHA confirme la liste de rendez-vous à e-Pension.	
7. E-Pension notifie la liste de rendez-vous au citoyen.	NotifieListeRDV
9. Le citoyen confirme le rendez-vous à e-Pension.	SélectionRDV
10. E-Pension confirme le rendez-vous au LHA	Confirmer rendez-vous
12. Le LHA envoie une confirmation du rendez-vous à e-Pension.	
13. E-Pension envoie la confirmation de rendez-vous au citoyen	RéçoitAccuséReception

Une fois les communications fournisseur principal identifiées, leur type est spécifié comme suit (cf. Tableau 6.6).

Tableau 6.6. Liste des communications Fournisseur principal et leurs types respectifs

Liste des Communications Fournisseur Principal	Type	Relation avec les services (services web et interface utilisateur (Interface $_i$))

1. Le citoyen formule une demande d'attribution d'un rendez-vous médical au système e-Pension	réception	FormuleDemandeRDVMédical (Interface *Attribuer un rendez-vous médical*)
2. E-Pension enregistre la demande d'attribution de rendez-vous médical	Activité interne	
3. E-Pension demande rendez-vous au LHA	Invocation dynamique de services web	Demander rendez-vous (service web ServiceLHA)
6. Le LHA confirme la liste de rendez-vous à e-Pension.		
7. E-Pension notifie la liste de rendez-vous au citoyen.	réponse	NotifieListeRDV (Interface *Attribuer un rendez-vous médical*)
9. Le citoyen confirme le rendez-vous à e- Pension.	réception	SélectionRDV (Interface *Attribuer un rendez-vous médical*)
10. E-Pension confirme le rendez-vous au LHA	Invocation dynamique de services web	Confirmer rendez-vous (service web ServiceLHA)
12. Le LHA envoie une confirmation du rendez-vous à e-Pension.		
13. E-Pension envoie la confirmation de rendez-vous au citoyen	réponse	RéçoitAccuséReception (Interface *Attribuer un rendez-vous médical*)

- L'étape 2 propose de déterminer : (i) les agents participant au Service Métier ; (ii) les services Web à invoqués dynamiquement ; et (iii) les services d'interface utilisateur invoquant le service métier.

L'étape 2 appliquée au scénario S *Attribuer un rendez-vous médical* aide à déterminer les éléments de composition à savoir :

- Le participant dans la coordination, tel que le LHA ;
- Les services Web qui devront être fournis par le participant, tel que le ServiceLHA ;
- Le service d'interface utilisateur nécessaire, à savoir Interface *Attribuer un rendez-vous médical*

Dans le cadre de cette règle, il faut relier chaque *activité d'appel* à *une opération* d'un service Web (Invocation dynamique de service), chaque *activité de type réception* à *l'interaction métier de type demande utilisateur* du service d'interface, et chaque *activité de type réponse* à *l'interaction métier de type réponse utilisateur* du service d'interface.

Tableau 6.7. Les activités de base et leurs relations avec les autres services

Liste des Communication Fournisseur principal	**Type**	**Relation avec les services web ou d'interface utilisateur**
1. Le citoyen formule une demande d'attribution d'un rendez-vous médical au système e-Pension	réception	Interface *Attribuer un rendez-vous médical*
2. E-Pension enregistre la demande d'attribution de rendez-vous médical	Activité interne	
3. E-Pension demande rendez-vous au LHA	Invocation	Service ServiceLHA

6. Le LHA confirme la liste de rendez-vous à e-Pension.	dynamique de services web	
7. E-Pension notifie la liste de rendez-vous au citoyen.	réponse	Interface *Attribuer un rendez-vous médical*
9. Le citoyen confirme le rendez-vous à e- Pension.	réception	Interface *Attribuer un rendez-vous médical*
10. E-Pension confirme le rendez-vous au LHA	Invocation dynamique de services web	Service ServiceLHA
12. Le LHA envoie une confirmation du rendez-vous à e-Pension.		
13. E-Pension envoie la confirmation de rendez-vous au citoyen	réponse	Interface *Attribuer un rendez-vous médical*

- L'étape 3 aide à construire le modèle de données d'un Service Métier (service de composition).

En appliquant l'étape 3 au scénario S *Attribuer un rendez-vous médical*, nous obtenons le modèle de données suivant (cf. Tableau 6.8) :

Tableau 6.8. Liste des ressources formant le modèle de données

Liste des communications métier	Ressources identifiées
1. Le citoyen formule une demande d'attribution d'un rendez-vous médical au système e-Pension	Demander_rendez_vous_médical (nomPatient, prénomPatient, dateRdv)
6. Le LHA confirme la liste de rendez-vous à e-Pension.	Liste_rendez_vous (dateRdv, etablissement, num-tel, nom_medecin)
9. Le citoyen confirme le rendez-vous à e- Pension.	Rendez_vous (dateRdv, etablissement, num-tel, nom_medecin)
12. Le LHA envoie une confirmation du rendez-vous à e-Pension.	Confirmation_rendez_vous

- L'étape 4 définit le modèle d'orchestration du service métier en considérant l'ordre d'exécution des activités de base. Le résultat obtenu correspond à l'identification de l'activité structurée.

Dans l'étape 4, l'observation des directives de transformation des communications métier (voir chapitre 4) appliquées au scénario S *Attribuer un rendez-vous médical* montre que les communications sont structurées en séquence. Par conséquent, les activités de base correspondantes sont structurées en séquence aussi.

- Nous n'appliquons pas l'étape 5 pour le contexte de l'exemple e-Pension vu le nombre réduit des services web impliqués.

En effet, le seul service web que nous possédons dans le scénario S *Attribuer un rendez-vous médical* est le service serviceLHA. Dans ce cas, nous ne pouvons pas avoir d'orchestration de service.

d. Découverte des opérations et des ressources permettant l'invocation dynamique de services Web

La modélisation des opérations et des ressources permettant l'invocation dynamique des services Web, à partir du scénario S *Attribuer un rendez-vous médical*, est guidée par un ensemble d'étapes méthodologiques. Comme pour les précédentes, ces étapes sont présentées en détail au Chapitre 4. L'application de ces étapes aide à modéliser progressivement les éléments des services métier d'e-Pension selon les termes de MOS.

- L'étape 1 consiste à extraire les actions de communication qui représentent une invocation de service Web.

En appliquant l'étape 1 au scénario S *Attribuer un rendez-vous médical*, nous obtenons les deux communications numérotées 3 et 10 dans le scénario présenté dans la Figure 6.5 :

3. E-Pension demande rendez-vous au LHA.

10. E-Pension confirme le rendez-vous au LHA.

- L'étape 2 aide à découvrir les opérations du service Web de type demande/réponse à partir des couples de communications qui ont pour objet la demande/provision d'informations ou la production/consommation de ressources.

L'application de l'étape 2 au scénario S *Attribuer un rendez-vous médical* permet alors d'obtenir les couples suivants :

- *3. E-Pension demande rendez-vous au LHA : 6. Le LHA confirme la liste de rendez- vous à e-Pension.*
- *10. E-Pension confirme le rendez-vous au LHA: 12. Le LHA envoie une confirmation du rendez-vous à e-Pension.*

Le Tableau 6.9 présente les opérations obtenues à partir des couples de communications déjà identifiés :

Tableau 6.9. Liste des opérations de type demande/réponse

Liste des communications Métier	Liste des opérations identifiées
3. E-Pension demande rendez-vous au LHA.	Demander rendez-vous ()
10. E-Pension confirme le rendez-vous au LHA.	Confirmer rendez-vous ()

- Il est à noter que l'étape 3 n'est pas appliquée étant donné que le scénario S *Attribuer un rendez-vous médical* ne présente aucune opération de type unidirectionnel.
- L'étape 4 aide à déterminer les messages en entrée et en sortie de chacune des opérations.

L'application de l'étape 4 au scénario S *Attribuer un rendez-vous médical*, donne la spécification de chaque opération présentée au Tableau 6.10.

Tableau 6.10. Les messages en entrée/sortie des opérations

Opérations identifiées	**Liste des messages en entrée/sortie**	
Demander rendez-vous ()	*Message en entrée*	Demande_rendez_Vous
	Message en sortie	Liste_Rendez_vous
Confirmer rendez-vous ()	*Message en entrée*	Rendez_Vous
	Message en sortie	Confirmation

- L'étape 5 permet de décrire le service Web en fonction des opérations qu'il fournit. Ceci est exprimé à l'aide du concept d'interface, dans lequel le service est décrit d'une manière abstraite suivant les fonctionnalités qu'il présente.

L'application de l'étape 6 au scénario S *Attribuer un rendez-vous médical*, permet d'obtenir l'interface présentée au Tableau 6.11.

Tableau 6.11. Liste des interfaces relatives à l'invocation dynamique des services web

Liste des agents fournisseur secondaire	**Liste des opérations**	**Les interfaces des services web**
LHA	Demander rendez-vous () Confirmer rendez vous ()	ServiceLHA

3.5.2. Découverte des scénarios alternatifs d'échec

Cette section permet de construire le modèle MOS par analyse des scénarios alternatifs d'échec. Un scénario alternatif d'échec représente un scénario qui se termine par la non satisfaction de l'intention.

La construction du modèle MOS, à partir du scénario alternatif d'échec, ne considère que les actions présentées à la Figure 6.6 et générées à partir de la négation de la contrainte, à savoir :

5. S'il n'existe pas de disponibilités alors

6. E-Pension demande au citoyen de reformuler sa demande

Ceci consiste à réappliquer le processus proposé à la section 5.2.1 et à l'étendre avec des règles supplémentaires.

a. Spécialisation les agents du scénario

L'application de la sous étape A de la section 5.2.1 pour le scénario alternatif illustré par la Figure 6.6 précise les agents et les rôles suivants :

- Le citoyen correspond au rôle de *Demandeur* du service
- E-Pension est découpé en deux rôles, celui de l'*Interface* et celui de *Coordonnateur*.

b. Modéliser le service d'interface utilisateur

L'application de l'étape B de la section 5.2.1 modélise le service d'interface utilisateur.

- L'étape 1 consiste à extraire les communications utilisateur à savoir :

6. E-Pension demande au citoyen de reformuler sa demande

qui représente une interaction de base, *ReformulationDemande*, entre le service d'interface utilisateur et le demandeur.

- L'étape 2 identifie, à partir des communications utilisateur, les rôles Demandeur, Interface et Fournisseur principal. Cette étape sert, par la suite, à déterminer le type de chacune des interactions de base.

Les rôles identifiés, à partir de la communication 6 (cf. Figure 6.6), sont les suivants : le citoyen qui joue le rôle de Demandeur et e-Pension qui joue à la fois le rôle d'Interface et de Fournisseur principal.

- L'étape 3 détermine le type de l'interaction de base *ReformulationDemande* comme le montre le Tableau 6.12 :

Tableau 6.12. Type des interactions de base

N°	Interaction	Agent source	Agent cible	Type	Sens
6	ReformulationDemande	e-Pension (Interface)	Citoyen	utilisateur	sortie

- L'étape 4 complète la description de chaque interaction de base par l'identification des ressources échangées. Nous présentons le résultat obtenu dans le Tableau 6.13.

Tableau 6.13. Liste des ressources

N°	Interaction	Agent source	Agent cible	Type	Sens	Ressources
6	ReformulationDemande	e-Pension (Interface)	Citoyen	utilisateur	sortie	Demande Reformulation

- L'étape 5 complète la description du service d'interface utilisateur par l'identification des interactions structurées de type contrainte à savoir :

5. S'il n'existe pas de disponibilités alors (C1)

Le Tableau 6.14 montre la disposition de la contrainte identifiée par rapport à l'interaction de base *ReformulationDemande*.

Tableau 6.14. Disposition des contraintes par rapport aux interactions de base

N°	Interaction	Agent source	Agent cible	Type	Sens
6	Contrainte C1	Contrainte			
	ReformulationDemande	e-Pension (Interface)	Citoyen	utilisateur	sortie

- L'étape 6 sert à alimenter les interactions de base obtenues suite à la contrainte C1 à savoir :

5. S'il n'existe pas de disponibilités ***alors*** *6. ReformulationDemande*

6. Construction de l'application e-Pension orientée services du modèle MISI

Pour la construction du modèle MISI de l'application e-Pension, différents choix doivent être fait. Ces choix contribuent à la découverte des différentes vues du modèle MISI, expliqué dans le chapitre 5.

6.1. Choix des paramètres architecturaux

Le choix des facettes a été porté sur les paramètres qui ont été décris dans le chapitre 5. Ces paramètres sont issus du méta-modèle de *facettes architecturales*. Ces paramètres représentent la base pour permettre de bâtir son application de service. Pour cette application d'e-Pension, les facettes choisies sont décris de la manière suivante :

- **Stratégie de distribution** : *Invocation à distance*

Dans le cas d'étude e-Pension, le client est intéressé par une stratégie de distribution de type invocation à distance qui lui permet d'invoquer les services qui sont conformes au modèle

MOS de l'application e-Pension obtenus par le processus de transformation sémantique décris dans le chapitre 4. Suivant la découverte du service d'interface utilisateur obtenue dans la section 5 du chapitre 6, nous distinguons quatre interfaces utilisateur qui sont : *FormuleDemandeRDVMédical*, *NotifieListeRDV*, *SélectionRDV*, *RéçoitAccuséReception*. Ces quatre interfaces utilisateur sont communes au service d'interface utilisateur Interface *Attribuer un rendez-vous médical*.

Le service métier du modèle MOS correspondant à l'application e-Pension ne possède qu'un seul service Web. Il n'existe donc pas d'orchestration des services. Le service Web obtenu par transformation MIS – MOS de l'application e-Pension est le service ServiceLHA.

Le service d'interface utilisateur et le service métier (dépourvu de coordination) sont construits chez un fournisseur de service que le client invoque. Ces services, qui sont invoqués à distance, composent le service interactif.

L'avantage d'un tel système est que le client n'a pas besoin de développer les services interactifs (interfaces utilisateur et services Web), mais les utilise à sa guise une fois qu'il en a besoin pour atteindre ses intentions. Il peut être aussi amené à utiliser plusieurs interfaces utilisateur et plusieurs services Web orchestrés par un coordinateur de services. Un tel système engendre une facilité d'utilisation, mais est soumis à certaines contraintes, comme la connectivité qui doit être de haut débit, la taille de la mémoire qui ne doit pas être limités, il doit avoir une grande puissance de traitement et un temps de chargement limité et temps d'exécution rapide.

- **Acteur d'implémentation de service interactif :** *fournisseur de service interactif*

L'acteur d'implémentation des services interactif Interface *Attribuer un rendez-vous médical* et ServiceLHA, est un fournisseur de service interactif. Le client invoque son service interactif depuis un fournisseur de service interactif. Compte tenu du fait que le client ne possède qu'un seul service Web, ServiceLHA, le fournisseur ne fournira pas de coordination de service. Dans ce cas de figure, pour exécuter les différents services interface utilisateur et Web, le client a besoin d'une application de présentation qui puisse regrouper ses services afin de les exécuter. Cette application de présentation est une interface unique, qui est appelé ici Portail. Le portail est un consommateur de services qui assemble tous les services qu'ils soient service interface ou services Web composites. Le portail est quant à lui construit chez le client de services.

- **Type de fournisseur métier** : *Atomique*

Le choix des deux paramètres cités ci-dessus nous permet de déduire un fournisseur atomique de service interactif. En effet, le client n'ayant qu'un seul service Web à invoquer, il n'aura donc pas besoin d'un orchestrateur de service Web. Cependant, dans le futur, s'il a besoin d'invoquer des services composites, alors le client aura besoin d'un fournisseur de coordinateur de services.

6.2. Choix d'une architecture

Le paramétrage ci-dessus nous conduit à choisir une architecture *2-tier* de services interactifs. La figure 6.7 présente l'architecture finale choisie pour mettre en place le modèle d'implémentation de service interactif MISI.

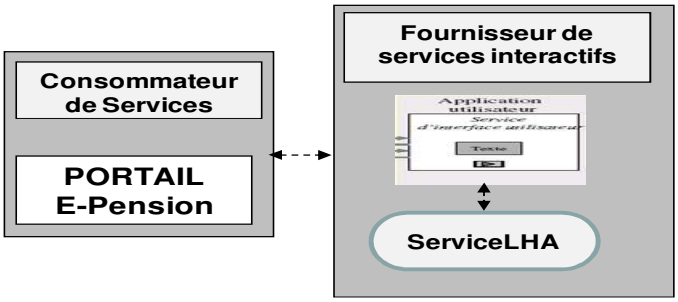


Figure 6.7. Architecture 2-tier de service interactif

Le modèle MISI mis en place est composé d'une vue interface utilisateur et d'une vue service Web. La vue coordination n'existe pas puisqu'il n'existe qu'un seul service Web défini dans le modèle MOS vu dans la section précédente. Ainsi, le méta-modèle MISI déduite est le méta-modèle basé sur le WSRP sans sa partie coordination (orchestration) se services web.

Au sein du modèle MISI, la vue interface utilisateur est composés d'une vue IHM (Interface Homme-Machine), dont la structure est constituée d'un ensemble de composants. La vue interface utilisateur est aussi composée d'un ensemble des règles de navigation, qui représentent un ensemble d'événements. Enfin, la vue interface utilisateur comprend aussi un contrôleur. La partie *fonction_orchestration* du contrôleur est reliée au processus *process,* puis au service Web par *partner* sans toutefois s'intéresser à l'orchestration des services.

Enfin, il convient d'observer que les types de toutes données échangées entre les différents éléments sont représentés en XSD.

6.3. Découverte des différentes vues du modèle d'implémentation de service interactif – MISI-WSRP.

6.3.1. Découverte de la vue Interface Utilisateur et du service web du modèle MISI

Les interactions du modèle MOS de l'application e-Pension trouvées dans la section précédente sont structurées ci-dessous :

FormuleDemandeRDVMédical (**Citoyen – Interface Utilisateur**)
EnvoieDemandeRDVMédical **(Interface Utilisateur – Service Web)**
Si (S'il existe des disponibilités) alors
ReçoitListeRDV **(Service Web – Interface Utilisateur)**
NotifieListeRDV (**Interface Utilisateur - Citoyen**)
Sinon
AfficherMessageReformulationDemandeRDVMédical (**Interface Utilisateur - Citoyen**)
Finsi
Si (Si le citoyen accepte un rendez-vous) alors
SélectionRDV (**Citoyen – Interface Utilisateur**)
EnvoieRDVSelectionné **(Interface Utilisateur – Service Web)**
RéçoitAccuséReception **(Service Web – Interface Utilisateur)**
AffícheAccuséReception (**Interface Utilisateur - Citoyen**)

Les transformations des différentes interactions de base du modèle MOS dans le modèle d'implémentation sont définies ci-dessous :

Tableau 6.15. Traduction de l'interaction de base *FormuleDemandeRDVMédical* entre le citoyen et l'interface utilisateur.

Modèle MOS	**Traduction dans le modèle MISI**				
Interaction Utilisateur	**Vue IHM**	**Composant**	**Conteneur**	**Atomique**	**Règle de Navigation**
					ECV
FormuleDemandeRDV Médical	FormulaireDemande RdvMedical	Formulaire	Formulaire	Bouton, input Text	″entree″

Le tableau 6.15 présente la transformation de l'activité utilisateur *FormuleDemandeRDV Médical* du modèle MOS en différents éléments du modèle MISI. La transformation de cette activité de base permet d'avoir un *composant agrégat* sous la forme d'un formulaire *FormulaireDemandeRdvMedical* composé à son tour d'un ensemble des *composants*

atomiques (*bouton*, *input text…),* permettant de prendre en compte les requêtes du citoyen. Une fois la requête saisie, la règle de navigation *″entree″* de type *ECV (Evènement – Contrôleur - Vue)* est envoyée au contrôleur le plus adapté pour traiter la requête du citoyen.

Tableau 6.16. Traduction de l'interaction de base EnvoieDemandeRDVMédical entre l'interface utilisateur et le service web.

Modèle MOS	**Traduction dans le modèle MISI**					
Interaction Métier	**Règle de Navigation**	**Contrôleur**	**Données en Entrée**	**XSD**	**Fonction Métier**	**Vue Service**
	ECV					
EnvoieDemandeRDV Médical	″entree″	ContrôleurRdv Medical	NomPatient, PrénonPatient, DateRdv	Chaine de caractère, Date	Demande RdvLHA	Service LHA

Le tableau 6.16 présente la transformation de l'activité métier *EnvoieDemandeRDVMédical* du modèle MOS en différents éléments du modèle MISI. Dans cette étape, la règle de navigation *″entree″* venant du formulaire *FormuleDemandeRDV Médical* désigne le contrôleur le plus adapté, ici le contrôleur *ContrôleurRdvMedical*. Ce contrôleur reçoit les données en entrée du citoyen. Ces données sont le nom du patient (*NomPatient*), son prénom (*PrénonPatient*) et la date de rendez-vous (*DateRdv*). Ces données servent à la réservation d'une date de rendez-vous et sont de type *chaîne de caractère et date*. Le contrôleur envoie les différentes données au service ServicesLHA, par l'intermédiaire de *DemandeRdvLHA,* afin qu'ils soient traitées en vue d'avoir une réponse pour le rendez-vous demandé.

Selon la réponse qui sera envoyé par le service *ServicesLHA,* deux cas s'imposent : soit le service répond positivement à la requête de l'utilisateur en envoyant des données en sortie, soit le service ne peut pas répondre à la requête de l'utilisateur, car il n'existe pas de réponse pour sa demande. Le premier cas est un cas de succès, alors que le deuxième est un cas d'échec.

Tableau 6.17. Traduction de l'interaction de base ReçoitListeRDV entre le service web et l'interface : Cas de Succès

Modèle MOS	Traduction dans le modèle MISI				
Interaction Métier	**Contrôleur**	**ViewForward**	**Données en Sortie**	**XSD Schema**	**Vue IHM**
ReçoitListeRDV	ContrôleurRdv Medical	″disponible″	ListeDateRdv (DateRdv, établissement, num_tel, nom_medecin)	Chaine de caractère, date	FormulaireNotifieListeRdv

Le tableau 6.17 présente la transformation de l'activité de métier *ReçoitListeRDV* du modèle MOS en différents éléments du modèle MISI. Dans cette étape, le service ServicesLHA exécute la requête de demande de réservation de rendez-vous du citoyen et la transmet à l'élément contrôleur *ContrôleurRdvMedical* du modèle MISI. En fonction du résultat reçu par le contrôleur, si le service ServicesLHA renvoie des données en réponse à la requête du citoyen, alors un événement *″disponible″* est généré. Cet événement de type *ViewForward* permet de transmettre les données renvoyées par le service ServicesLHA au citoyen par l'intermédiaire de la vue IHM *FormulaireNotifieListeRdv* (vue interface homme-machine), laquelle s'occupe de l'affichage du résultat. Les données résultant de la requête sont organisées sous la forme d'une liste de rendez-vous constituée de la date de celui-ci, du numéro de téléphone et du nom du médecin. Ces éléments sont de type chaîne de caractère et date.

Tableau 6.18. Traduction de l'interaction de base NotifieListeRDV entre l'interface et le citoyen : Cas de succès

Modèle MOS	Traduction dans le modèle MISI			
Interaction Utilisateur	**VueIHM**	**Composant**	**Conteneur**	**Atomique**
NotifieListeRDV	FormulaireNotifieListeRdv	Formulaire	Formulaire	label

Le tableau 6.18 présente la traduction de l'interaction métier *NotifieListeRDV* du modèle MOS dans les éléments leurs correspondant dans le modèle MISI. Dans cette étape, la liste de rendez-vous est envoyée à l'utilisateur citoyen par le biais de la *vueIHM,* laquelle est constituée d'un formulaire qui se compose d'un label sur lequel est écrit la liste des données en sortie pour le citoyen.

Tableau 6.19. Traduction de l'interaction de base AffichеMessageReformulation entre l'interface et le citoyen : Cas de d'échec (S'il n'existe pas de disponibilité)

Modèle MOS	Traduction dans le modèle MISI				
Interaction Utilisateur	**Contrôleur**	**ViewForward**	**Données en Sortie**	**XSD Schema**	**Vue IHM**
AffichеMessage Reformulation	ContrôleurRdv Medical	″indisponible″	MessageReformulation	Chaine de caractère	FormulaireReformu lationRdv

Le tableau 6.19 présente l'interaction utilisateur *AffichеMessageReformulation* du modèle MOS, traduite en différentes éléments du modèle MISI. Dans cette étape, le contrôleur ne reçoit pas de réponse à la requête de l'utilisateur en termes de liste de rendez-vous, car le service *ServicesLHA* n'a pas de rendez-vous pour les dates indiquées par le citoyen. Le contrôleur, dans ce cas d'échec de la demande de l'utilisateur, émet un événement "*indisponible*". Un message en sortie *MessageReformulation* lié à l'événement "indisponible" est alors envoyé à l'utilisateur citoyen, par l'intermédiaire d'un formulaire de reformulation de requête avec une date différente.

Tableau 6.20. Traduction de l'interaction de base SélectionRDV entre le citoyen l'interface.

Modèle MOS	Traduction dans le modèle MISI					
Interaction Utilisateur	**Vue Interface Utilisateur**	**Composant**	**Conteneur**	**Atomique**	**Règle de Navigation**	
					ECV	**EV**
SélectionRDV	FormulaireSél ectionRdv	Formulaire	Formulaire	Bouton, input text, checkbox.	″selection″	″refuse″

Le tableau 6.20 est la suite du tableau 6.18, lorsque le service *ServicesLHA* répond à la requête de l'utilisateur avec une lise des dates. La liste de rendez-vous est envoyée au citoyen (tableau 6.18) pour que ce dernier sectionne sa date de rendez-vous. Dans le tableau 6.20, l'utilisateur citoyen sélectionne la date de rendez-vous désirée et l'envoie au contrôleur par

l'intermédiaire du bouton "envoyé" sur le formulaire *FormulaireSélectionRdv*. La sélection d'une date de rendez-vous amène le formulaire à déclencher la règle de navigation "*selection*" de type ECV (Evènement – Contrôleur - Vue).

Sinon, si les dates de rendez-vous ne conviennent pas au citoyen, alors celui-ci refuse en choisissant le bouton "refuser", ce qui déclenche la règle de navigation "refuse" de type EV (Evènement - Vue). Cette règle de navigation permet d'abandonner l'interaction en cours, en revenant à une autre vue IHM.

Tableau 6.21. Traduction de l'interaction de base EnvoiRDVSelectionné entre l'interface et le service web.

Modèle MOS	**Traduction dans le modèle MISI**					
Interaction Métier	**Règle de Navigation ECV**	**Contrôleur**	**Données en Entrée**	**XSD Schema**	**Fonction Métier**	**Vue Service**
EnvoiRDV Selectionné	″selection″	ContrôleurSelection DemandeRdv	Rdv_vous_selectionné (DateRdv, ets, num_tel, nom_medecin)	Chaine de caractère, Date	demande RdvLHA	Service LHA

Le tableau 6.21 présente l'interaction correspondant au cas dans lequel l'utilisateur sélectionne une date parmi la liste des dates proposées. L'action sur le bouton engendre une règle de navigation "*sélection*", laquelle permet de choir le contrôleur pour le traitement de la requête. Le contrôleur *ContrôleurSelectionDemandeRdv* reçoit le rendez-vous sélectionné *Rdv_vous_selectionné,* puis il achemine ce dernier au service *ServicesLHA* pour le traitement, via la fonction métier *demandeRdvLHA*.

Tableau 6.22. Traduction de l'interaction de base RéçoitAccuséReception entre le service web et l'interface.

Modèle MOS	**Traduction dans le modèle MISI**				
Interaction Métier	**Contrôleur**	**ViewForward**	**Données en Sortie**	**XSD Schema**	**Vue IHM**
RéçoitAccuséReception	ContrôleurSelection DemandeRdv	″affiche″	Confirmation Rdv	Chaine de caractère	FormulaireAfficher ConfirmationRdv

Le tableau 6.22 présente l'interaction métier *RéçoitAccuséReception*, dans laquelle le contrôleur reçoit la confirmation de la réception du rendez-vous sélectionné par le citoyen. Il

déclenche alors un événement "*affiche*" qui permet de choisir la meilleure vue pour afficher la confirmation de la réception au citoyen.

Tableau 6.23. Traduction de l'interaction de base AfficheAccuséReception entre l'interface et le citoyen.

Modèle MOS	Traduction dans le modèle MISI			
Interaction Utilisateur	**Vue Interface Utilisateur**	**Composant**	**Conteneur**	**Atomique**
AfficheAccuséReception	FormulaireAfficher ConfirmationRdv	Formulaire	Formulaire	label

Enfin, le tableau 6.23 présente l'interaction de base *AfficheAccuséReception* du modèle MOS traduite en différentes éléments du modèle MISI. Dans ce tableau, le contrôleur choisi la meilleure vue IHM pour afficher l'accusé de réception.

6.3.2. Définition des types de données

Les types de données utilisées lors des échanges de messages entre le contrôleur, les vues et le service métier sont spécifiés en XSD schéma. Les figures 6.8 et 6.9 représentent les types de données en entrée et les données en sortie de l'application E-Pension.

Les données échangées en entrée par les différentes opérations de l'application e-Pension sont essentiellement du type chaîne de caractère et forment un type complexe nommé *Données_en_Entrée*, lequel représente la liste des rendez-vous.

```
<?xml version="1.0" encoding="UTF-8"?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Données_en_Entrée">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="nom" type="xs:string" />
     <xs:element name="prenom" type="xs:string" />
     <xs:element name="date_rendez-vous" type="xs:date" />
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:schema>
```

Figure 6.8. Type de données en entrée défini en XSD schéma

De même, les données échangées en sortie par les différentes opérations de l'application e-Pension forment aussi un type complexe, nommé *Données_en_Entrée*, lequel inclut les informations sur le rendez-vous (date, établissement, nom du médecin et son numéro de téléphone).

```
<?xml version="1.0" encoding="UTF-8"?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Données_en_Sortie">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="date_rendez-vous" type="xs:date" />
     <xs:element name="etablissement" type="xs:string" />
     <xs:element name="num_tel" type="xs:string" />
     <xs:element name="nom_medecin" type="xs:string" />
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:schema>
```

Figure 6.9. Type de données en sortie défini en XSD schéma

7. Génération du code de l'application E-Pension orientée service intentionnel

7.1. Génération des JSP (Java Server Face) correspondants aux différentes vues IHM.

Dans cette section, nous illustrons la génération de code pour les vues à partir de l'exemple d'un seul formulaire. Il s'agit du formulaire *formulaireDemandeRDVMédical*. Ce formulaire permet de prendre en compte les requêtes de l'utilisateur citoyen pour l'obtention d'un rendez-vous, c'est-à-dire le nom, le prénom et la date de rendez-vous. Ce formulaire, présenté dans la Figure 6.10, est généré en *Java Server Face* (JSP). Les pages JSP générés proviennent d'un même format de page qui contient les balises des composants telles que <form>, <input> etc. Ces balises contiennent des valeurs telles qu'action, type, name, etc. les fonctions renderRequest, getParameter permettent de récupérer les valeurs des données renseignées par les utilisateurs. Le choix de ces composants permet la génération des pages JSP.

Même si, dans cette section, nous ne présentons que de l'exemple d'un seul formulaire, il est important de souligner que d'autres formulaires sont également générés, ces sont les formulaires de *FormulaireNotifieListeRdv.jsp*, *FormulaireReformulationRdv.jsp* et *FormulaireAfficher ConfirmationRdv.jsp*.

```
<%@ taglib uri=″http://java.sun.com/portlet_2_0″ prefix=″portlet″%>
<portlet:defineObjects />
<%PortletPreferences prefs = renderRequest.getPreferences (); %>
<form action=<portlet:actionURL/> method=″POST”>
      <% String nom=renderRequest.getParameter (“nom”) %>
       If (nom==null) nom= ″″;%>
       <input type=”text” name=”nom” value=”<%=nom%>” />

      <% String prenom=renderRequest.getParameter (“prenom”) %>
      If (prenom==null) prenom= ″″;%>
      <input type=”text” name=”prenom” value=”<%=prenom%>” />

      <% String dateRdv=renderRequest.getParameter (“dateRdv”) %>
      If (dateRdv==null) dateRdv= ″″;%>
      <input type=”text” name=”dateRdv” value=”<%=dateRdv%>” />

      <input type=”submit” value=”validate” name=”submitApplication” class=”buttonStyle” />
       <input type=”hidden” value=”login” name=”event” />
</form>
```

Figure 6.10. FormulaireDemandeRDVMedical.jsp

7.2. Génération des règles de l'application E-Pension

Les règles présentées en XML dans la Figure 6.11 permettent de configurer de manière automatique l'application e-Pension avec les objets générés : le modèle (*Model*) qui fait appelle aux services composites à travers leurs différents éléments (*event, controller, inputview, type, forward*). L'élément *<rules>* dans la figure 6.11 permettent de décrire comment les différents éléments s'enchainent entre eux en fonction des traitements. L'élément *<controller>* permet de faire le pont entre la « vue Interface Utilisateur » et la « vue Service ». Il analyse la requête du client et se contente d'appeler le modèle adéquat et de renvoyer la vue correspondante à la demande. Le *controller* est traduit par *perform* dans le code généré. Il peut avoir plusieurs contrôleurs dans une application. L'élément *<inputView>* représente une page web pour acquérir les interactions utilisateurs de type entrée et sortie. Cet écran correspond à l'interface avec laquelle l'utilisateur interagit. Dans le cas de l'application basée sur le WSRP, la page web est un portlet qui s'affiche sur un portail. L'élément *<forwards>* est un événement qui a un nom et un type et permet de diriger les événements issus du contrôleur vers la vue la plus adéquate. Les types d'événement peuvent être soit des types *ecv* (événement – controller - view) ou des types *ev* (événement – view) ou des types *v*.

Les types *ecv* permettent d'envoyer les réponses du service web issues des contrôleurs soit vers un autre contrôleur pour traitement ou soit vers une autre page web JSP pour l'affichage de la réponse. Les types *ecv* permettent d'envoyer les réponses directement vers une page web JSP.

Les règles de l'application ci-dessous sont générées à partir l'EDI (Environnement de Développement Intégré) NetBeans [Netbeans]. Elles permettent de construire facilement notre application e-Pension à travers n'importe quel outil de développement.

```
<config
xmlns : xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns ='http://xml.netbeans.org/schema/Config'
<welcome> Portlet_FormulerDemandeRDVMedical.jsp <welcome>
<model> FonctionDemandeRDVLHA </model>
<rules>
        <rule>
            <event>entree</event>
            <controller> performEnvoieRDVMedical</controller>
            <inputview> Portlet_FormulerDemandeRDVMedical.jsp </inputview>
            <type>ecv</type>
            <forwards>
               <forward name= ″disponible" type="v" forward=" Portlet_NotifieListeRDV.jsp" />
               <forward name= ″indisponible" type="v" forward=" Portlet_FormulerDemandeRDV.jsp" />
            </forwards>
        </rule>
        <rule>
            <event>disponible</event>
            <controller>performSelectionDemandeRDV</controller>
            <inputview> Portlet_NotifieListeRDV.jsp</inputview>
            <type>ecv</type>
            <forwards>
               <forward name= ″affiche" type="v" forward="Portlet_AfficherConfirmationRdv.jsp" />
               <forward name= ″refuse" type="v" forward="Portlet_AnnulerRDV.jsp" />
               <forward name= ″erreur" type="v" forward="Portlet_DemandeRDVErreur.jsp" />
            </forwards>
        </rule>
        <rule>
            <event>erreur</event>
            <controller>default</controller>
```

```
            <inputview> Portlet_NotifieListeRDV.jsp </inputview>
            <type>ev</type>
            <outputview>RDVPortlet_global_event_error.jsp</outputview>
        </rules>
</config >
```

Figure 6.11. Génération des règles de l'application E-Pension

7.3. Génération du contrôleur : *performEnvoieRDVMedical*

Chaque contrôleur utilisé dans l'application e-Pension est généré de manière automatique à partir d'un ensemble de règles. La Figure 6.12 illustre le contrôleur *performEnvoieRDVMedical* généré pour le cas d'étude e-Pension. Le fichier ci-dessous est écrit dans le langage Java. Le corps des fichiers *controller* sont structurés de la même manière. C'est au développeur d'application de compléter la structure avec certaines informations Cependant, avec la structure du fichier *controller*, la tâche du développeur est beaucoup réduite. Dans la figure 6.12 ci-dessous, la définition de la fonction *Protected void performEnvoieRDVMedical (ActionRequest request, ActionResponse response) {}* reste le même pour tout type de contrôleur. L'élément *request* qui a pour type ActionRequest prend en compte les données renseignées par l'utilisateur (*nom, prenom, date_rdv*) dans la page JSP de la vue IHM tandis que l'élément *response* qui à pour type *ActionResponse* renvoie la réponse du service web dans notre cas actuel service *serviceLHA* (*date_rendez-vous, etablissement, num_tel, nom_medecin*) à la page web. Dans la suite du fichier, l'invocation du service web serviceLHA se fait à l'aide de la fonction *getModel ()*. Puis la réponse du service web est obtenue à l'aide de la fonction *modeleDemandeRDVLHA.getRdv*, cette fonction étant l'exemple du cas e-Pension.

La génération de ce fichier réduit considérablement le travail du développeur d'application.

```
//contrôleur pour performEnvoieRDVMedical
Protected void performEnvoieRDVMedical (ActionRequest request, ActionResponse response) {

// récupération des données
String nom=request.getParameter (″nom″) ;
String prenom=request.getParameter (″prenom″) ;
Date date_rdv=request.getParameter (″date_rdv ″) ;

// Tester la validité des données
If ( ((login != null) && (prenom != null)  && (date_rdv !=null)))

//Invocation du modèle
Object O = this.getModel();
ModeleInterface modeleDemandeRDVLHA =  (ModeleInterface) O;

// Affectation des paramètres
```

```
String demandeRdv=  modeleDemandeRDVLHA.getRdv (nom, prenom, date_rdv);

//Cas Normal
If (demandeRdv) {
Forward= ″disponible″
response.setRenderParameter(″date_rendez-vous″, date_rendez-vous) ;
response.setRenderParameter(″etablissement″, etablisssement) ;
response.setRenderParameter(″num_tel″, num_tel) ;
response.setRenderParameter(″nom_medecin″, nom_medecin) ;
}
Else {
// cas échec
  }
}
else {
// Cas d'erreur
  }
}
```

Figure 6.12. Génération d'un exemple de contrôleur : *performEnvoieRDVMedical*

7.4. Transformation ATL du méta-modèle de la vue Coordination de MISI au méta-modèle du langage BPEL

7.4.1. Définition des transformations en ATL

La Figure 6.13 ci-dessous représente les règles de transformation du méta-modèle de la vue coordination de MISI vers le méta-modèle du langage BPEL.

Compte tenu qu'il existe qu'un seul service dans le cas d'étude e-Pension, nous n'avons normalement pas besoin d'une orchestration de services. Cependant dans le but de montrer son fonctionnement, nous mettons ces règles, car elles sont les mêmes pour toutes applications ayant des services web à orchestrer.

Les règles de transformation ATL ont été décrites au chapitre 5.

L'exemple de la figure 6.15 montre le document du modèle (instance) BPEL obtenu par la transformation du méta-modèle de la vue coordination de MISI au méta-modèle du langage BPEL. Cet exemple est relatif au service Intentionnel S *Attribuer un rendez-vous médical* de notre étude de cas e-pension.

```
module Coordinateurs2BPEL; -- Module Template
create OUT : BPELs from IN : Coordinateurs;
rule C2P {
        from c: Coordinateurs!Coordination
        to p: BPELs!Process (
                name <- 'Service' +  c.name,
                targetNamespace <- 'http://' + p.name + '.org/',
                abstractProcess <- 'false',
                partnerlink <- pt
                ),
```

```
                pt : BPELs!PartnerLink (
                        name <- p.name,
                        partnerLinkType <- plt,
                        myRole <- 'ITF_'+ p.name+'Provider',
                        partnerRole <- ''
                        ),
                        plt: BPELs!PartnerLinkType (
                                name <- 'ITF_' + p.name + 'Link',
                                role <- rl
                                ),
                        rl: BPELs!Role (
                                name <- 'ITF_' + p.name + 'Provider',
                                portType <- 'ITF_' + p.name
                                )
}
rule R2V {
        from r: Coordinateurs!Ressource
        to v:BPELs!Variable (
                name <- r.name
                )
}
rule Se2Seq {
        from se: Coordinateurs!Sequences
        to Seq: BPELs!Sequences (
           )
}
rule Re2Rec {
                from re: Coordinateurs!Reception
                to rec: BPELs!Receive(
                        name <- re.name,
                        operations <- re.operations,
                        variable <- re.ressource,
                        createInstance <- 'true'
                        )
}
rule C2F {
        from c: Coordinateurs!Concurrence
        to f: BPELs!Flow (
                name <- c.name
                )
}
rule Ses2Seqs {
        from ses: Coordinateurs!"Sequence"
        to Seqs: BPELs!"Sequence" (
)
}
rule A2As {
        from a: Coordinateurs!Affectation
        to as: BPELs!Assign (
                name <- a.name
                )
}
rule A2I {
        from a: Coordinateurs!Appel
        to i: BPELs!Invoke (
                name <- a.name,
                operations <- a.operations,
                inputVariable <- a.ressourceEnEntree,
                outputVariable <- a.ressourceEnSortie
                )
```

```
}
rule Re2Ry {
        from re: Coordinateurs!Reponse
        to ry: BPELs!Reply (
                name <- re.name,
                operations <- re.operations,
                variable <- re.ressource
                )
}
rule F2T {
        from f: Coordinateurs!Fin
        to t: BPELs!Terminate (
                name <- 'EndProcess'
                )
}
```

Figure 6.13. Règles de transformation du méta-modèle de la vue coordination de MISI au méta-modèle du langage BPEL

La figure 6.14 est une instance conforme au méta-modèle de la vue coordination du MISI. Cet instance est relatif à l'exemple au service Intentionnel S *Attribuer un rendez-vous médical* de l'application e-Pension.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="Coordinateurs">
<Coordination name="ServicePension">
<ressource name="demandeRdv"/>
<sequences>
<reception name="recevoirDemandeRdv" operation="demandeRdv" ressource="demandeRdv"/>
<sequence>
<affectation name="affecterDemandeRdv"/>
<appel name="demandeRdv" operation ="trouverVoyage" ressourceEnEntree="bb" ressourceEnSortie="cc"/>
</sequence>
<sequence>
<affectation name="affecterDemandeRdv"/>
<appel name="ServiceVol" operations ="demandeRdv" ressourceEnEntree="demandeRdv"
ressourceEnSortie="ListeRdv"/>
<reponse name="reponseDemandeRdv" operation="demandeRdv" ressource="ListeRdv"/>
<affectations name="affecterDemandeRdv"/>
<fin/>
</sequences>
</Coordination>
</xmi:XMI>
```

Figure 6.14. Document d'un modèle (une instance) conforme au méta-modèle de la vue coordination de MISI relatif au service Intentionnel S *Attribuer un rendez-vous médical* de l'application e-Pension

L'exemple de la Figure 6.15 montre une instance BPEL conforme au méta-modèle du langage BPEL relatif au service Intentionnel S *Attribuer un rendez-vous médical*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="BPELs">
  <Process name="ServicePension" targetNamespace="http://ServicePension.org/" abstractProcess="false">
```

```
      <partnerlink name="ServiceLHA" myRole="ITF_ServiceLHAProvider" partnerRole="">
      <partnerLinkType name="ITF_ServiceLHALink">
       <role name="ITF_ServiceLHAProvider" portType="ITF_ServicePension"/>
       </partnerLinkType>
       </partnerlink>
       <Variable name="demandeRdv"/>
      <Sequences>
       <Receive name="receptionDemandeRdv" operation="demandeRdv" variable="demandeRdv"
        createInstance="true"/>
       <Assign name="assignDemandeRdv"/>
      <Invoke name="demandeRdv" operation="ServiceLHA" inputVariable="demandeRdv"
     outputVariable="ListeRdv"/>
       <Reply name="reponseTrouverRdv" operations="demandeRdv" variable="listeRdv"/>
      <Terminate name="EndProcess"/>
    </Sequence>
</process>
</xmi:XMI>
```

Figure 6.15. Document du modèle BPEL conforme au méta-modèle du langage BPEL relatif au service Intentionnel S *Attribuer un rendez-vous médical* **de l'application de pension**

8. Conclusion

Ce chapitre a présenté l'application de la démarche méthodologique MeTSI à l'étude de cas *e-Pension* permettant d'aider les personnes handicapées à obtenir une pension. L'application a permis d'illustrer les étapes à appliquer pour trouver les services opérationnels et services logiciels interactifs qui correspondent aux mieux aux exigences des clients. L'application générée est conforme à la plateforme de service interactif WSRP.

CHAPITRE 7 : CONCLUSION

1. Contribution

Cette thèse a exploré un certain nombre de problèmes posés par un changement d'échelle visant à permettre une exploitation des services intentionnels dans une méthode de transformation de modèle en application exécutable à base de services logiciels. Un service intentionnel MIS se définit par rapport au but que les services logiciels correspondants permettront d'atteindre.

L'approche que nous avons proposée dans cette thèse et qui se nomme MeTSI (pour Méthode de Transformation de Service Intentionnel) a permis de décrire une démarche de développement d'une application à base de services interactifs par transformations successives des services intentionnels en services logiciels interactifs exécutables.

La méthode applique les principes de l'ingénierie dirigée par les modèles. Elle adopte une approche transformationnelle, dans le sens où, à chaque étape, les modèles sont transformés et affinés. Elle permet d'aboutir à une solution logicielle exécutable sous la forme d'une application à base de services qui contient toutes les dimensions d'une application, à savoir, l'aspect interface de l'utilisateur avec l'application, l'aspect métier réalisé par des services web et leur coopération dans une composition orchestrée de services.

Elle opère par transformations successives de façon à séparer les sujets d'intérêt pour apporter de la flexibilité et permettre l'adaptation, en particulier, à différentes plateformes d'implémentation. Elle réutilise des services web déjà réalisés lorsque c'est possible.
Les propositions que nous avons présentées dans ce mémoire, comportent les éléments suivants :

- Un méta-modèle de représentation des services opérationnels : le modèle MOS,
- Une démarche pour dériver les services opérationnels à partir des services intentionnels,
- Un méta-modèle d'implémentation des services interactifs dépendant de la plateforme : le modèle MISI,

- Une démarche pour transformer les services opérationnels en service dépendante d'une plateforme suivant différents paramétrages caractérisant trois différentes plateformes de services interactifs,
- La génération de code spécifique à une plateforme.

Nous avons retenu la définition de la notion de service intentionnel fondée sur le modèle MIS de Kaabi [2007] comme point de départ de notre travail

Le méta-modèle de représentation des services intentionnels dénommé MIS (Modèle intentionnel de Services) définit chaque service intentionnel en tant que brique de construction d'applications visible au travers de son interface qui apporte la connaissance situationnelle et intentionnelle. Chaque service intentionnel s'applique dans une situation particulière pour réaliser une intention particulière.

Le méta-modèle MIS est détaillé au Chapitre 3. Il classe les services intentionnels en atomiques et agrégats. Un *service atomique* n'est pas décomposable en d'autres services intentionnels alors qu'un *service agrégat* l'est. Cette classification prend sa source dans la nature des buts associés aux services.

La vue opérationnelle des services intentionnels est développée au Chapitre 4. Celle-ci est réalisé, d'une part, par la présentation du méta-modèle opérationnel de services (MOS) et, d'autre part, par la proposition d'une démarche méthodologique qui permet de guider l'identification, au niveau opérationnel, des services qui correspondent au mieux aux attentes et aux exigences des clients tels qu'exprimés, au niveau intentionnel, par des services intentionnels.

Le méta-modèle MOS propose, à travers le concept de *service opérationnel*, un service structuré en deux couches: service d'interface utilisateur et service métier. Chacun de ces éléments opérationnels joue un rôle architectural particulier et est implémenté à l'aide de techniques spécifiques de développement.

Les caractéristiques du méta-modèle MOS sont les suivantes :

- L'interconnexion avec le méta-modèle MIS. Ceci permet de passer de la perspective intentionnelle à la perspective opérationnelle des services.
- L'indépendance à une plateforme d'implémentation spécifique.

Le chapitre 5 développe le méta-modèle d'implémentation de service interactif – MISI.

Le méta-modèle MISI permet de guider l'identification, au niveau logiciel, de services qui correspondent aux services opérationnels de type MOS. Ceci est réalisé à l'aide de la représentation explicite des services logiciels qui découlent des services opérationnels dans

les termes d'un modèle spécifique MISI (Modèle d'Implémentation de Service Interactif) et un ensemble de règles méthodologiques prenant en compte des paramètres caractérisant les différents plateformes de services interactifs.

Les paramètres caractérisant les plateformes de services interactifs sont modélisés dans un méta-modèle architectural d'infrastructure appelé facettes architecturales et comportent plusieurs facettes liées à plusieurs attributs. Les facettes d'architecture ont été mises en place pour aider l'agent opérationnel à faire un choix entre les différentes plateformes de services.

Le processus de transformation MOS-MISI est motivé par le besoin de définir une solution architecturale qui est déterminée en fonction de paramètres qui caractérisent une classe d'architectures de même nature. L'effort fait par MISI est d'avoir identifié les paramètres qui caractérisent les différentes classes architecturales et de les avoir modélisées dans un méta-modèle. Le processus de transformation tient évidemment compte des valeurs des paramètres choisis par l'ingénieur d'application pour configurer la solution architecturale en fonction de ces valeurs.

Enfin, le chapitre 6 décrit également la dernière étape de mise en œuvre de la méthodologie MeTSI qui consiste en la génération de code de services logiciels interatcifs. Trois plateformes d'implémentation possibles : OSGi [OSGi 2009], WSRP [WSRP 2006] et Mashup [Wikipedia, Mashup 2006], ont été choisi et représentent un panel représentatif de l'ensemble des solutions disponibles à ce jour.

Le processus de génération s'appuie sur des règles spécifiques à chaque plateforme. Ces règles sont automatisées. On peut donc parler de génération de code. Au bout de compte, l'application est générée dans une forme exécutable. Elle comporte les aspects relatifs à l'interaction avec l'usager, de composition des services et, d'encapsulation des règles métiers dans les services.
Dans ce travail, nous avons choisi la plateforme de service WSRP pour générer le code de services logiciels interactifs.

2. Perspectives

Le travail présenté dans cette thèse peut être poursuivi dans plusieurs directions :

- **Recherche dans l'annuaire des services intentionnels :**

La méthodologie proposée dans cette thèse ne présente pas de mécanismes pour la recherche des services, au niveau intentionnel, dans l'annuaire.

La proposition d'un processus qui vise à une localisation de services dirigée par les buts peut contribuer à l'enrichissement de la méthodologie. Chercher un service intentionnel revient à établir la coïncidence entre le but recherché (celui qu'un client d'un annuaire de services cherche à atteindre) et le but affiché par le service (celui que le service garantit de satisfaire).

Le processus de recherche peut intégrer plusieurs aspects à savoir :

- *L'extraction et l'assemblage des services intentionnels*. Il doit y avoir la possibilité de sélectionner des services dans l'annuaire sur la base des caractéristiques de l'interface d'un service intentionnel MIS. Un langage d'interrogation spécifique à la formulation de ces caractéristiques pour accéder au contenu de l'annuaire doit être défini.
- *Des mesures de similarité*. Elles doivent permettre de mesurer à quel point deux buts sont proches sémantiquement même s'ils ne sont pas exprimés de manière identique. L'approche linguistique de formulation des buts devrait aider à résoudre ce problème.

- **Enrichissement du modèle MIS pour prendre en compte les exigences non fonctionnelles :**

L'ingénierie des systèmes à base de services distingue les caractéristiques fonctionnelles qui définissent les services à fournir, des caractéristiques non fonctionnelles qui contraignent la manière dont l'application doit satisfaire les exigences fonctionnelles ou la manière de la développer. Le modèle MIS ne permet de prendre en compte que les caractéristiques fonctionnelles. Le modèle MIS pourrait être enrichi par des caractéristiques non fonctionnelles dans la définition des services.

BIBLIOGRAPHIE

[Alanso *et al.* 2004] Alanso, G., Casati, F., Kuno, H., and Machiraju, V. *Web Services: Concepts, Architectures and Applications.* Springer, Heidelberg, 2004.

[Andro 2007] AndroMDA. Disponible sur *htpp://www.andromda.org/.*

[Arjuna Technologies *et al.* 2005] Arjuna Technologies Ltd., BEA Systems, Hitachi Ltd., IBM Corporation, IONA Technologies, and Microsoft Corporation. 2005. Web Services Business Activity Framework. Published online at ftp://www6.software.ibm.com/software /developer/library/WS-BusinessActivity.pdf.

[Arni-Bloch *et al.* 2009] N. Arni-Bloch, J. Ralyté, and L. Michel. *Service-Driven Information Systems Evolution: Handling Integrity Constraints Consistency*. Published in Springer, 2009.

[Arni-Bloch *et al.* 2008] N. Arni-Bloch, J. Ralyté, and L. Michel. MISS: *A Metamodel of Information System Service*. Published in Springer, 2008.

[Azaiez 2007] S. Azaiez. *Approche dirigée par les modèles pour le developpement de systèmes multi-agents*. Thèse de doctorat, Université de Savoie, 2007.

[Baida *et al.* 2004] Z. Baida, J. Gordijn, B. Omelayenko, H. Akkermans. A *Shared Service Terminology for Online Service Provisioning*. Proceedings of the Sixth International Conference on Electronic Commerce (ICEC04),Delft, The Netherlands, 2004.

[Batini 2001] C. Batini. *Enabling Italian E-Government Through a Cooperative Architecture*. IEEE Computer, vol 6, no. 3, 2001.

[Belangour *et al.* 2006] A. Belangour, J. Bézivin, and M. Fredj. *Towards a new software development process for MDA*. In Proceedings of the European Workshop on Milestones, Models and Mappings for Model-Driven Architecture, Bilbao, Spain, 2006.

[Belhajjame *et al.* 2006] K. Belhajjame, S. M. Embury, N. W. Paton, R. Stevens, and C. A. Goble, *Automatic Annotation of Web Services based on Workflow Defintions*, ISWC 2006.

[Bellifemmine 1999] F. Bellifemmine, A. Poggi, and G. Rimassa. *JADE - A FIPA compliant agent framework.* In 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, 1999.

[Ben Achour 1999] C. Ben Achour. *Extraction des Besoins par Analyse des Scénarios Textuels*, Thèse du doctorat à l'Université de Paris6. Janvier 1999.

[Benatallah *et al.* 2002] B. Benatallah, M. Dumas, Q-Z. Sheng et A H. H. Ngu. *Declarative composition and peer-to-peer provisionning of dynamic web services*. Rakesh Agrawal, Klaus Dittrich et Anne H; H. Ngu, éditeurs, 18th international Conference on Data Engineering (ICDE 2002), pages 297- 308, San Jose, California USA, 2002. IEEE Computer Society.

[Benslimane *et al.* 2008] D. Benslimane, S. Dustdar, A. Sheth. Services Mashups: *The New Generation of Web Applications*. IEEE Internet Computing, 2008, vol.12, n°5, pp.13-15.

[Bézivin *et al.* 2005] J. Bézivin, M. Blay, M. Bouzhegoub, J. Estublier, J.M. Favre, S. Gérard, and J.M. Jezequel. *Rapport de Synthèse de l'AS CNRS sur le MDA (Model Driven Architecture).* Rapport CNRS, janvier 2005.

[Bottoni *et al.* 2000] P. Bottoni, M. Koch, F. Parisi-Presicce, and G. Taentzer. *Consistency checking and visualization of OCL constraints*. In The Unified Modeling Language Advancing the strandard, third international conference, pages 294–308, 2000.

[Boulet et al. 2002] P. Boulet, J.-L. Dekeyser, C. Dumoulin, Ph. Kajfasz, Ph. Marquet, and D. Ragot. Sophocles : Cyber-Enterprise for System-On-Chip Distributed Simulation. LIFL02, June 2002.

[Bresciani *et al.* 2004] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini, 2004. *Tropos: An agent-oriented software development methodology*. Autonom. Agents Multi- Agent Syst. 8, 3, 2003-236.

[Brogi *et al.* 2008] A. Brogi, S. Corfini, and R Popescu, *Semantics-Based Composition-Orinted Discovery of web Services*, ACM Transactions on Internet Technology, Vol.8, No. 4, Article 19, Sptembre 2007.

[Budinski *et al.* 2003] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick et T.J. Grose, *Eclipse Modeling Framework: A Developer's Guide*. Addison-Wesley Pub Co, 1st edition, August 2003.

[Castro *et al.* 2002] J. Castro, M. Kolp, and J. Mylopoulos. *Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems.* Elsevier, Amsterdam, the Netherlands.

[Chella *et al.* 2004] A. Chella, M. Cossentino, and L. Sabatucci. *Tools and patterns in designing multiagent systems with PASSI*. In 6th WSEAS International Conference on Telecommunications and Informatics, Cancun, Mexico, 2004.

[Chen et al 2003] H. Chen and T.Finin. *An Ontology for a context Aware Pervasive Computing Environment*. In Internationan Joint Conference On Artificial Intelligence, IJCAI Workshop on Ontologies and Distributed Systems, IJCAI, 2003.

[Chevrin 2006] V. Chevrin. *L'Interaction Usagers/Services, multimodale et multicanale : une première proposition appliquée au domaine du e-Commerce***.** Thèse de doctorat en informatique, Université de Lille1, Avril 2006.

[Corby *et al.* 2008] O. Corby, R. Dieng-Kuntz, C. Faron-Zucker. *Querying the semantic web with the CORESE search engine.* 16th European Conference on Artificial Intelligence (*ECAI/PAIS*), Valencia, Spain, p.705-709, 2008.

[Cossentino *et al.* 2007] M. Cossentino, S. Gaglio, A. Garro, and V. Seidita. *Method fragments for agent design methodologies : from standardization to research*. International Journal on Agent Oriented Software Engineering (IJAOSE), 2007.

[**Cossentino *et al.* 2005**] M. Cossentino, C. Bernon, and J. Pavón. *Modelling and Metamodelling Issues in Agent Oriented Software Engineering: The AgentLink AOSE TFG Approach*. Report of the AOSE TFG Ljubljana meeting, 2005.

[Czarnecki *et* Helsen 2003] K. Czarnecki and S. Helsen. *Classification of model transformation approaches*. In OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, 2003.

[Derniame *et al.* 1999] J.-C. Derniame, B. A. Kaba, and D.G.Wastell, editors. *Software Process : Principles, Methodology, Technology*, volume 1500 of Lecture Notes in Computer Science. Springer, 1999.

[Dey *et al.* 2001] A. K. Dey, D. Salber, and G. D. Abowd: *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications*. Human-Computer Interaction, 16(2-4) (2001) 97–166.

[Dinh *et al.* 2006] T.-L.-A. Dinh, O. Gerbé, and H. Sahraoui. *Un méta-méta-modèle pour la gestion de modèles*. In IDM 06 Actes des 2èmes Journées sur l'Ingénierie Dirigée par les Modèles, Lille, France, 2006.

[Dowson 1988] M. Dowson. *Iteration in the Software Process*, Proc 9th Int. Conf. On Software Engineering, 1988.

[EMOF 2004] OMG. MOF 2.0 Core Final Adopted Specification, Object Management Group, http://www.omg.org/cgi-bin/doc?ptc/03-10-04, 2004.

[Estublier *et al.* 2005] J. Estublier, G. Vega, and A. Ionita. *Composing Domain- Specific Languages for Wide-scope Software Engineering Applications*. In MODELS, 2005.

[Favre 2004] J.-M. Favre, *Toward a Basic Theory to Model Model Driven Engineering*, 3rd Workshop in Software Model Engineering, WiSME 2004, http://www-adele.imag.fr/~jmfavre.

[Franckson *et al.* 1991] M. Franckson, C. Peugeot. *Specification of the object and process modelling language*. ESF Report n° D122-OPML-1.0, 1991.

[Höbler *et al.* 2006] J. Höbler, M. Soden, and H. Eichler. *Coevolution of Models, Metamodels and Transformations*. http ://www.ikv.de/index.php, 2006.

[ICSOC 2008] 6th International Conference on Service Oriented Computing, December 1-5, 2008, University of Technology, Sydney, Ultimo City Campus.

[IEEE SCC 2009] International Conference on Services Computing September 21-25, 2009, Bangalore, India

[Ionita *et al.* 2005] A. D. Ionita, J. Estublier, and G. Vega. *Domaines réutilisables dirigés par les modèles*. In Proc. Of Ingénierie dirigée par les modèles, IDM05, Paris, 2005.

[**Issarny *et al.* 2007**] V. Issarny, M. Caporuscio and N. Georgantas. *A Perspective on the Future of Middleware-based Software Engineering*. In: Briand, L. and Wolf, A. (Eds.), Future of Software Engineering 2007 (FOSE), ICSE (International Conference on Software Engineering), IEEE-CS Press. 2007

[ISM 2003] IBM, BEA Systems, Microsoft, SAP AG, and Siebel Systems. Business Process Execution Language for Web Services. ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, May 2003.

[IST-SPICE 2010] FP6 IST Project SPICE. http://www.ist-spice.org/.

[Jack *et al.* 2004] G. Jack, K. Short, S. Cook, and S. Kent. *Software Factories : Assembling Applications with Patterns, Models, Frameworks, and Tools*. West Sussex, England : John Wiley & Sons, Ltd., 2004.

[Jarke *et al.* 1993] M. Jarke and K. Pohl, *Requirements Engineering: An Integrated View of Representation, Process and Domain*, Proc. 4th European Software Conf., Springer Verlag (1993).

[Jarke *et al.* 1992] M. Jarke and K. Pohl, *Information systems quality and quality information systems*, In Proc. Of the IFIP 8.2 working conference on the impact of computer-supported techniques on information systems development, Mineapolis, NM (june 1992).

[Jonkers *et al.* 2007] H. Jonkers, M. Steen, L. Heerink, and D. Van Leeuwen. *From Business Process Design to Code : Model-Driven Development of Enterprise Applications*. https ://doc.freeband.nl/dsweb/Get/Document-77839/, 2007.

[Jouault 2006] F. Jouault, *Contribution à l'étude des langages de transformation de modèles*, thèse de doctorat, Université de Nantes, 2006.

[Jouault *et al.* 2006] F. Jouault and I. Kurtev. *On the architectural alignment of atl and qvt.* In Proceedings of ACM Symposium on Applied Computing (SAC 06), Model Transformation Track, Dijon (Bourgogne, FRA), April 2006.

[JSR168] JSR 168: Portlet Specification, http://www.jcp.org/en/jsr/detail?id=168.

[Juric 2005] M. Juric. BPEL and Java. *On line journal theserverside.com*. http://www.theserverside.com/articles/article.tss?l=BPELJava.

[Kaabi 2007] R. Kaabi, *Une approche méthodologique pour la modélisation intnetionnel de services et leur opérationalisation*, Thèse de doctorat, Université Paris 1 – Sorbonne, 2007.

[Kadima 2005] H. Kadima. *MDA - conception orientée objet guidée par les modèles*. NFE114, 2005.

[Kalnins *et al.* 2004] A. Kalnins, J. Barzdins et E. Celms, *Basics of Model Transformation Language MOLA*. Workshop on Model Driven Development (WMDD 2004) at ECOOP 2004, June 2004.

[Karsai *et al.* 2003] G. Karsai, A. Agrawal, F. Shi et J. Sprinkle, *On the Use of Graph Transformation in the Formal Specification of Model Interpreters.* J. UCS, 9(11) : 1296.1321, 2003.

[Kazhamiakin *et al.* 2003] R. Kazhamiakin, M. Pistore, and M. Roveri. *T-tool tutorial. Technical report*, University of Trento, 2003.

[Kim *et al.* 2004] J. Kim, M. Spraragen, Y. Gil. *An Intelligent Assistant for Interactive Workflow Composition*. In IUI'04 Proc. On application and Theory of Petri Nets, London, UK, Springer-Verlag (1997), 407-426.

[Kleppe 2003] A. Kleppe, S. Warmer, W. Bast, , *MDA Explained. The Model Driven Architecture : Practice and Promise*, Addison-Wesley, Avr. 2003.

[Larvet 1994] P. Larvet. *Analyse des systèmes : de l'approche fonctionnelle à l'approche objet*. InterEditions., 1994.

[Ledeczi *et al.* 2001] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, and J. Sprinkle. *Composing domain-specific design environments*. Computer Networks and ISDN Systems, pages 44–51, 2001.

[**Link *et al.* 2006**] S. Link, F. Jakobs, L. Neer, S. Abeck,: *Architecture of and Migration to SOA's Presentation Layer.* Cooperation & Management (Prof. Abeck), Research Report, 2006.

[Lopes 2005] D. Lopes, *Étude et applications de l'approche MDA pour des plateformes de Service Web,* thèse de doctorat, Université de Nantes, 2005.

[Maamar *et al.* 2006] Z. Maamar, D. Benslimane and N.C. Narendra. *What can context do for web services?,* Communication of the ACM, vol. 49, n° 12, pages 98-103Dec. 2006,

[Maness 2006] J.M. Maness. Library 2.0 Theory: Web 2.0 and Its Implications for Libraries. Webology [en ligne], 2006, vol.3, n°2. Disponible sur : <http://www.webology.ir/2006/v3n2/a25.html>.

[Mens *et al.* 2005] T. Mens, K. Czarnecki, and P. Van Gorp. *A taxonomy of model transformations.* In Dagstuhl Seminar Proceedings 04101, 2005.

[Mellor 2004] S. J. Mellor, K. Scott, A. Uhl, , D. Weise, *MDA Distilled : Principales of Model-Driven Architecture*, Addison-Wesley, Mar. 2004.

[Meyers *et al.* 2002] H. Meyers, M. Weske. *Automated Service Composition using Heuristic Search.* In S. Dustdar, J-L. Fiadeiro, A. Sheth, eds.: Proc. of the 4th International Conference on Business Process Management (BPM 2006). Volume 4102 of Lecture Notes In Computer Science, Heidelberg, Springer (2002) 81-96.

[Mirbel *et al.* 2009] I. Mirbel, P, Crescenzo, *Improving Collaboration in Neuroscientist Community.* International Conference on Web Intellingence and Intelligent Agent Technology, 2009.

[Najar *et al.* 2009] S.Najar, O.Saidani, M. Kirsch-Pinheiro, C.Souveyet, and S. Nurcan. Semantic representation of context models: a framework for analyzing and understanding. In: J. M. Gomez-Perez, P. Haase, M. Tilly, and P. Warren (Eds), 1st Workshop on Context, information and ontologies (CIAO 09), European Semantic Web Conference (ESWC'2009), (Heraklion, Greece, June 2009). ACM, New York, NY, 1-10. DOI= http://doi.acm.org/10.1145/1552262.1552268.

[Nehan *et* Deneckere 2007] Y-R. Nehan and R. Deneckere, *Component-Based Situation Methods. A framework for understand SME.* Published in SME: Fundamentals and experiences, Geneve: Switzerland (2007).

[NetBeans] www.netbeans.org

[OMG 2006] OMG. Meta Object Facility Core Specification version 2.0. OMG Document, january 2006.

[OMG 2005] OMG. MOF QVT Final Adopted Specification. OMG document, november 2005.

[OMG 2004] OMG. Uml 2 metamodel. ptc/04-10-05 (UML 2.0 Superstructure FTF Rose model containing the UML 2 metamodel), 2004.

[OMG 2003] OMG. MDA GUIDE Version 1.0.1. document number omg/2003-06-01., 2003.

[OMG 2002] 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.

[OSGi 2009] Alliance OSGi, "OSGi Service Platform Release 4.2 : Core Specification", http://www.osgi.org/Specifications/HomePage, September 2009.

[Pallos 2001] M. Pallos. *Service Oriented Architecture: A Primer*, EAI Journal, December 2001.

[Papazoglou *et al* 2007] P. Papazoglou et W-J van den Heuvel: *Service Oriented Architectures: Approaches, Technologies and Research Issues*, The VLDB Journal Springer Berlin / Heidelberg, Vol. 16, No. 4, pages 389-415.

[Papazoglou 2006] M.P. Papazouglou, *Principe et Fondements of Web Services: A holistic view*, Addison-Wesley, 2006.

[Papazoglou *et* Georgakopoulos 2003] M.P. Papazouglou and Georgakapoulos, *Introduction to the special issues about Service-Oriented Computing*, CACM, October 2003, 46 (10): 24-29.

[Penserini *et al.* 2007] L. Penserini, A. perini, and A. Susi: *High Variability Design for Software Agents: Extending Tropos*. ACM Transaction on autonomous And Adaptative Systems, Vol.2, No. 4, Article 16, Novembre 2007.

[Penserini *et al.* 2006a] L. Penserini, A. Perini, A. Susi, and J. Mypoulos, *From stakeholder intentions to software agent implementations*. In Proceedings of the Advanced Information Systems Engineering, 18th International Conference (CAiSE'06). E. Dubois and K. Pohl, Eds. Lecture Notes in Computer Science, vol. 4001. Springer, 465–479.

[Penserini *et al.* 2006b] L. Penserini, A. Perini, A. Susi, J. Mylopoulos. *From Stakeholder needs to service requirements specifications*. Technical Report, ITC-IRST, Automated Reasoning Systems, 2006.

[Quartel et al. 2004] D. Quartel, R-M. Dijkman, M. van Sinderen. *Methodological Support for Service-oriented Design with ISDL*. Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04). New York, USA, 2004.

[Rolland 2007] C. Rolland. *Conceptual Modeling in Information Systems Engineering*, Springer-Verlag, 2007.

[Rolland *et* Kaabi 2005] C. Rolland, R-S. Kaabi. *Designing Service Based Cooperative Systems*, Encyclopedia of ETechnologies and Applications (EET&A), IDEA Group (pub), 2005.

[Rolland *et* Prakash 2000] C. Rolland, N. Prakash. *Bridging the gap between Organizational needs and ERP functionality*. Requirements Engineering Journal, 2000.

[Rolland 1998] C. Rolland, *A comprehensive view of process engineering*, proceeding of CAISE'98, Pisa, Italy, (1998).

[Rolland 1997] C. Rolland, *A primer for method engineering*, proceeding of INFORSID' 97, Toulouse, France (1997).

[Rolland *et al.* 1998] C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N-A. M. Maiden, M. Jarke, P. Haumer, K. Pohl, E. Dubois and P. Heymans. *A Proposal for a Scenario Classification Framework*, Requirements Engineering Journal (REJ), Vol. 3, N°1, pp. 23-47, 1998.

[Rosemann et al. 2007] M. Rosemann, J. Recker and C. Flender, C.. *Contextualization of Business Processes*. International Journal on Business Process Integration and Management, vol. 1, n. 1/2/3, 2007.

[Ryu *et al.* 2008] S. Ryu, F. Casati H. Skogsrud, B. Benatallah, and R. Saint-Paul, *Supporting the Dynamic Evolution of Web Service Protocols in Service-Oriented Architectures*, ACM Transaction on the Web, Vol. 2, No 2, Article 13, April 2008.

[Saidani et al. 2007] O. Saidani and S. Nurcan. *Towards Context Aware Business Process Modeling*, 8th Workshop on Business Process Modeling, Development, and Support (BPMDS'07), CAiSE'07, 2007.

[Sassen *et al.* 2005] A. Sassen and C. Macmillan. *The service engineering area: An overview of its current state and a vision of its future*. Disponible sur ftp://ftp.cordis.europa.eu/pub/ist/docs/directorate_d/stds/sota_v1-0.pdf. European Commission, Network and Communication Technologies, Software Technologies, 2005.

[Sawyer *et al.* 2005] P. Sawyer, J. Hutchison, J. Walkerdine, I. Sommerville. Faceted Service Specification. Proc. Workshop on Servic Oriented Computing Requirements (SOCCER). Paris, August, 2005.

[Schäfer *et al.* 2008] M. Schäfer, P. Dolog, and W. Nejdl, *An Environment for flexible Advances Compensations of Web Services Transactions*, ACM Journal January 2008.

[Schaffner *et al.* 2006] J. Schaffner, H. Meyer. *Mixed Initiative Use Cases for Semi_autoated Service Composition: A Survey*. In Proc. Of International WWorkshop on Service Oriented Software Engineering (IW-SOSE'06), located at ICSE 2006, 27-28 May, 2006, Shangai, China, ACM Press, NewYork, NY, USA, 2006.

[Sebastiani *et al.* 2004] R. Sebastiani, P.Giorgini, and J. Mylopoulos. *Simple and minimum-cost satisfiabilityfor goal models*. In *Proceedings of the 16th (CAiSE 04), LNCS Springer*, 2004.

[Seidewitz 2003] E. Seidewitz, *What Models Mean*, IEEE Software, Vol. 20, Num. 5, Sept. 2003, Page(s) : 26 - 32.

[Sirin *et al.* 2004] E. Sirin, B. Parsia, D. Hendler, J. Nau. *HTN Planning for Web Service Composition Using SHOP2*. Journal of Web Semantics 1 (2004) 377-396.

[Smart 2006] http: //smartqvt.elibel.tm.fr/, 2006.

[Suraci *et al.* 2007] V. Suraci, S. Mignanti and A. Aiuto. *Context-aware Semantic Service Discovery*. 16th IST Mobile and Wireless Communications Summit, 2007, 1-5

[Tawbi 2001] M. Tawbi. *Crews-L'Ecritoire : Un guidage outillé du processus d'ingénierie des besoins.* Thèse de Doctorat à l'Univeristé Paris 1, octobre 2001.

[Toninelli *et al.* 2008] A. Toninelli, A. Corradi, and R. Montanari. *Semantic-based discovery to support mobile context-aware service access*. Computer Communications, vol. 31, n. 5 (2008), 935-949.

[UDDI 2006] UDDI. *Universal Description, Discovery and Integration,* http://www.uddi.org/, 2006.

[UML2Tool] Eclipse: Model Development Tools Project, http://www.eclipse.org/modeling/mdt/

[Van der Aalst *et al.* 2003] W. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. *Workflow Patterns*. Distributed and Parallel Databases, 14(1):5–51, 2003.

[Van der Aalst *et al.* 2000] W. van der Aalst, J. Desel, and A. Oberweis, editors. *Business Process Management: Models, Techniques, and Empirical Studies* - Lecture Notes in Computer Science, volume 1806. Springer-Verlag, 2000.

[W3C 2001] W3C. 2001. http://www.w3.org

[Wahlster *et al.* 2006] Wahlster, W., Dengel, A. Web 3.0*: Convergence of Web 2.0 and the Semantic Web. Technology Radar* Feature Paper, Edition II/2006, Deutsche Telekom Laboratories, pp.1-23.

[Wikipedia, Mashup 2006] Wikipedia , Mashup Homepage: http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid), accessed November 22, 2006.

[Wohed *et al.* 2002] P. Wohed,W. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. *Pattern Based Analysis of BPEL4WS*. Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.

[WSRP 2006] WSRP Web Services for Remote Portlets specification version 2.0. http://www.oasis-open.org/committees/download.php/18617/wsrp-2.0-spec-pr-01.html

[XQuery 2004] XQuery 1.0 and XPath 2.0 formal semantics. W3C Working Draft, Feb. 2004.

[XQuery 2003] XQuery 1.0: An XML query language. W3C Working Draft, Nov. 2003.

[Yang *et al.* 2003] J. Yang, M-P. Papazoglou. *Service Components for Managing the Life-Cycle of Service Compositions*. Information Systems Journal, 2003.

[Yu 1995] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, University of Toronto, 1995.

[Zhang *et al.* 2005] W. Zhang, H. Meiand, H. Zhao, and J. Yang. *Transformation from CIM to PIM : A Feature-Oriented Component-Based Approach.* In *Model Driven Engineering Languages and Systems, 8th Intenational Conference MoDELS*, Montego Bay, Jamaica, 2005.