Une Approche Méthodologique pour la Modélisation Intentionnelle des Services et leur Opérationnalisation

Rim Samia Kaabi

THESE DE DOCTORAT DE L'UNIVERSITE PARIS I – SORBONNE

Spécialité : **INFORMATIQUE**

Présentée par

Rim Samia Kaabi

Pour l'obtention du titre de :

DOCTEUR DE L'UNIVERSITE PARIS I - SORBONNE

Sujet de la thèse :

Une Approche Méthodologique pour la Modélisation Intentionnelle des Services et leur Opérationnalisation

Soutenue le 13 février 2007 devant le jury composé de

Mme Colette ROLLAND	Directeur de thèse
Mme Carine SOUVEYET	Co-ditecteur de thèse
Mme Corine CAUVET	Rapporteur
M. Michel LEONARD	Rapporteur
Mme Françoise GIRE	Membre du jury

À mes parents

Remerciements

Tout au long de ce travail, et plus généralement de mon parcours réalisé ces dernières années, j'ai pu bénéficier de soutiens, de conseils et d'encouragements d'un très grand nombre de personnes auxquelles je tiens à exprimer toute ma gratitude.

Je voudrais exprimer ma reconnaissance envers les membres du jury. Pouvoir réunir pour cette occasion des chercheurs d'un tel niveau au sein de disciplines aussi diverses est pour moi un très grand honneur et une marque d'encouragement pour la conduite de recherches interdisciplinaires.

En premier lieu, je voudrais exprimer ma reconnaissance envers Madame 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 reconnaissante 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 tiens à remercier Madame Carine Souveyet, Maître de Conférence Habilitée à Diriger des Recherches à l'Université de Paris 1 Panthéon - Sorbonne pour sa disponibilité, ses conseils et son soutien pendant ces quatre années.

Je remercie sincèrement Madame Corine Cauvet, Professeur à l'Université Paul Cézanne Aix-Marseille 3, et Monsieur Michel Léonard, Professeur à l'Université de Genève, qui ont accepté de juger ce travail et d'en être les rapporteurs. Je les remercie pour l'application avec laquelle ils ont lu mon manuscrit et toutes les questions, riches d'intérêt, qu'ils ont pu soulever et qui sont encore vives dans mon esprit.

Je remercie également Madame Françoise Gire, Professeur à l'Université de Paris 1 Panthéon - Sorbonne pour avoir accepté de faire partie du jury de cette thèse.

Mes remerciements vont également à tous les membres de l'équipe du Centre de Recherche en Informatique, qui m'ont permis de passer des années agréables et enrichissantes. J'exprime toute ma gratitude envers tous ceux qui m'ont aidée et encouragée tout au long de mon séjour au sein de l'équipe.

Je voudrais remercier tous mes amis et connaissances. J'adresse tout d'abord mes remerciements aux thésards et aux autres chercheurs qui m'ont accompagnée pendant des mois, que j'ai croisés quasi-quotidiennement et avec qui j'ai eu l'occasion et le plaisir de partager des pauses café sympathiques.

Je suis très reconnaissante à Inès, Rébecca et Anne qui ont eu le courage et la patience de relire mon mémoire.

Je me dois d'exprimer ma profonde gratitude à deux personnes qui ont compté aussi pour l'achèvement de cette thèse, celles qui m'ont accordée leur aide et soutien au cours de ces années. Je remercie donc Naoufel Kraiem et Inès Gam: Naoufel, mon oracle, dont les conseils et les recommandations m'ont aidée à diriger mes idées et pensées ; Ines, qui a été comme une sœur, et qui m'a soutenue dans les périodes les plus difficiles, chaque fois que j'en avais besoin.

Merci à tous mes amis qui m'ont aidée et soutenue tout au long de ce travail et particulièrement : Maha et Ramzi. Merci pour toute votre affection et amitié dévouée qui m'ont longuement et profondément touchée.

Ces quatre années resteront ancrées dans ma mémoire comme une période très enrichissante de ma vie. Cet aboutissement dans mes études ne pouvait se réaliser sans être associé au soutien des personnes les plus proches de moi : ma famille, une valeur à laquelle j'attache énormément d'importance, et particulièrement à mes parents attentifs, affectueux, courageux et admirables ; ma grand-mère qui m'a toujours remontée le moral et encouragée à sa manière ; ma sœur Imen que j'affectionne particulièrement ; et Lotfi qui m'a soutenu avec ses encouragements. Votre patience et l'affection que vous m'avez apportée durant ces années loin de vous m'ont aidée à garder le moral haut, à surmonter les difficultés et à achever ma thèse dans les meilleures conditions. Je tiens à vous dire merci d'avoir été toujours là, pour la patience et l'affection que vous m'avez manifestées durant ces années et pour votre aide morale.

RESUME

Le paradigme SOA est devenu un standard pour la conception technique et le développement d'applications logicielles. Malgré tous les avantages qu'il apporte notamment dans la résolution des problèmes d'hétérogénéité, les solutions associées à ce paradigme sont dédiées aux développeurs et restent difficiles à transposer au monde de l'entreprise. L'hypothèse de cette thèse est que pour tirer le meilleur profit du paradigme SOA, il est nécessaire d'adopter une position dan laquelle les services sont décrits en termes de besoins qu'ils permettent de satisfaire, c'est a dire par les buts organisationnels a satisfaire. Nous qualifions ces services d'intentionnels. La publication, la recherche et la composition de ces services se fait sur la base de ces descriptions intentionnelles. De cette façon, on parle d'un portage de SOA au niveau intentionnel que nous qualifions de ISOA.

La solution proposée dans cette thèse a conduit aux résultats suivants :

- Un modèle de représentation des services intentionnels qui intègre la variabilité et la réflexivité : le modèle $\mathcal{MiS}$ (**M**odèle **I**ntentionnel de **S**ervices),
- Une démarche pour identifier les services intentionnels à partir des besoins des utilisateurs,
- Un modèle de représentation des services opérationnels : le modèle $\mathcal{MoS}$ (**M**odèle **O**pérationnel de **S**ervices),
- Une démarche pour dériver les services opérationnels à partir des services intentionnels,
- Une architecture à agents permettant l'exécution intentionnelle des services.

MOTS CLES : Service**,** Architecture orientée service, Composition de service, Approche orientée but.

ABSTRACT

Despite the growing acceptance of SOA, service-oriented computing remains a computing mechanism to speed-up the design of software applications by assembling ready-made services. We argue that it is difficult for business people to fully benefit of the SOA if it remains at the software level. We propose a move towards a description of services in business terms, i.e. intentions and strategies to achieve them and to organize their publication, search and composition on the basis of these descriptions. In this way, it leverages on the SOA to an intentional level, the ISOA.

The solution developped leads to a methodology based on :

- A goal-based model for representing intentionnal services called $\mathcal{MiS}$ (**M**odèle **I**ntentionnel de **S**ervices),
- A process to identify intentional services from user needs and to represent them into a $\mathcal{MiS}$ model,
- A model for representing operational services called $\mathcal{MoS}$ (**M**odèle **O**pérationnel de **S**ervices),
- A process to identify operational services from intentional services and,
- Agent architecture to support model driven execution of intentional services.

KEYWORDS : Service, Service Oriented Architecture, Service composition, Goal driven approach.

Table Des Matières

Table Des Figures

Table Des Tableaux

CHAPITRE 1

Introduction

1. CONTEXTE DE LA THESE

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

Des composants aux services

Les services sont apparus comme une suite logique des composants logiciels et des approches d'intégration d'applications [Pfister96] à base de composants. Les composants sont des entités logicielles indépendantes utilisant des interfaces pour permettre leur communication au sein d'une même application ou à travers un réseau via une décomposition de la logique applicative en composants distribués [Brown96] [Szyperski97]. L'architecture de composants distribués a permis le développement rapide et évolutif d'applications complexes et distribuées. Cependant, cette architecture, bien qu'utilisant un modèle objet distribué, impose sa propre infrastructure et une liaison forte et figée entre les fonctionnalités offertes par les composants et leurs clients. Les applications construites à base de cette architecture sont donc monolithiques. L'architecture à base de composants est incompatible avec l'ouverture de l'infrastructure Internet.

Le concept de service et les architectures à base de services sont apparus en réaction aux limites des architectures à base de composants de façon à exploiter les possibilités offertes par Internet. Parallèlement, l'avènement du B2B (Business To Business) a renforcé le besoin d'interopérabilité d'applications hétérogènes, développées sur des plate-formes différentes. L'interopérabilité est ainsi devenue une nécessité pour l'entreprise dans le monde du B2B et c'est justement ce que le paradigme *SOC* (Service Oriented Computing) apporte par rapport aux solutions dites monolithiques [Papazoglou03] [Alonso04]. SOC a pour objectif de construire des applications distribuées en réutilisant des briques logicielles hétérogènes appelées *services*.

Bien qu'il n'existe pas une définition consensuelle de ce qu'est un service, celle donnée par [Papazoglou02] sert de référence aux recherches de cette thèse car elle est très largement référencée dans la littérature. D'après [Papazoglou02], un service est défini comme *un ensemble d'applications modulaires auto-contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le web. Un service peut effectuer des actions allant de simples requêtes à des processus métiers complexes. Les services permettent d'intégrer des systèmes d'information hétérogènes en utilisant des protocoles et des formats de données standardisés, autorisant ainsi un faible couplage et une grande souplesse vis-à-vis des choix technologiques effectués*.

Le domaine des services a été et est l'objet de nombreuses recherches au point que de nombreux séminaires et conférences sont consacrés à ce thème de recherche [ICSOC04] [ICSOC05] [Dagstuhl05]. Les sujets traités sont variés et comportent entre autres, l'étude du concept de service, les architectures à base de service, la composition de services, les méthodes de conception d'applications à base de services, et les mécanismes de contrôle de l'exécution distribuée d'applications à base de services.

L'Architecture SOA

Au cœur du domaine des services se trouve l'*architecture orientée services* connue sous l'appellation *SOA* (Service Oriented Architecture) qui permet à une application de faire l'usage d'une fonctionnalité située dans une autre application (cf. Figure 1) [W3C04].

L'architecture SOA vise à transformer le Web en une énorme plate-forme de services faiblement couplés et automatiquement intégrables. Elle offre un modèle à base de service en ligne permettant de construire une application par composition de services définis par des interfaces uniformes et standardisées. Elle utilise des protocoles Internet pour gérer la communication entre les services.

L'architecture SOA propose une perspective globale sur le développement, la gestion et le fonctionnement des services Web [Barry03]. L'architecture SOA est un modèle (abstrait) qui définit un système comme un ensemble d'agents logiciels distribués fonctionnant de concert afin de réaliser une fonctionnalité globale préalablement établie [Heather01]. Les agents dans un système distribué opèrent dans des environnements de traitement hétérogènes et communiquent par échange de messages afin de solliciter les services permettant d'obtenir le résultat souhaité.

Comme le montre la Figure 1, les principaux éléments intervenant dans l'architecture SOA sont :

– Le fournisseur de service : désigne la personne ou l'organisation responsable du service. Il peut également (i) demander à l'annuaire de services de ***publier*** des services et (ii) ***interagir*** avec le client de service.

– Le client de service : représente une personne ou une organisation. C'est un client potentiel des services puisqu'il désigne également l'application cliente qui doit ***interagir*** avec le service une fois que celui-ci a été ***localisé***.

– L'annuaire des services : représente la personne ou l'organisation responsable de ***publier*** des services. Il symbolise l'entité logicielle qui joue le rôle d'intermédiaire entre les clients et les fournisseurs de services. L'annuaire de services fourni aux clients les informations techniques et sémantiques sur le fonctionnement du service une fois ***localisé*** par le client.

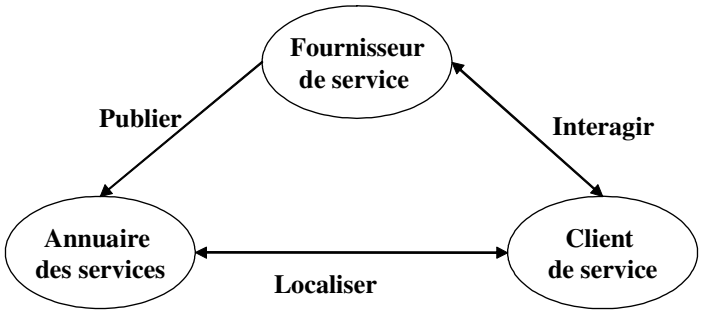


Figure 1. L'architecture orientée service (SOA)

L'architecture SOA donne également des indications sur le cycle de développement des services ainsi que leur cycle de vie [Booth03] :

– Chaque service est défini par un fournisseur. Le fournisseur de services déploie et publie la description de son service dans des annuaires pour en permettre l'usage par des clients ;

– Les clients satisfont leurs besoins en terme de services en effectuant des recherches dans les annuaires de services;

– Une fois le service localisé, le client extrait sa description de l'annuaire;

– Sur la base des informations définies dans la description du service, le client entreprend une interaction.

Le modèle de fonctionnement de l'architecture SOA se base sur un cadre technologique qui constitue son infrastructure. Cette infrastructure offre les fonctionnalités nécessaires à la réalisation des différentes étapes du cycle de vie d'un service. Celles-ci correspondent à des standards. Les trois standards principaux sont [Chauvet02] :

– «S*imple* O*bject* A*ccess* P*rotocol*» (SOAP), assure la communication avec et inter services[1]. SOAP permet l'échange de données structurées indépendamment des langages de programmation ou des systèmes d'exploitation. Le protocole SOAP n'est pas concerné par la nature du programme cible ou le traitement des messages, il ne définit pas un environnement d'exécution mais plutôt un modèle de liaison qui assure l'interopérabilité entre des applications hétérogènes.

– «W*eb* S*ervices* D*escription* L*angage*» (WSDL), offre un langage de description des services[2] [Christensen01]. Contrairement aux architectures monolithiques où la description des composants ainsi que les moyens de les invoquer dépendent de l'infrastructure utilisée, la spécification WSDL permet de décrire l'interface des services de telle façon qu'ils se suffisent à eux-mêmes. La spécification WSDL présente les services comme des boites noires, au moyen d'un certain nombre de ports d'entrée et de sortie.

La spécification WSDL joue un rôle important dans l'interopérabilité des services. Moyennant un schéma uniforme obéissant à une sémantique bien définie, elle permet de définir ce qui est nécessaire à l'invocation des services.

– «U*niversal* D*escription*, D*iscovery* and I*ntegration*» (UDDI) [Bellwood02], offre une manière uniforme de définir des annuaires de services ainsi qu'un schéma uniformément extensible de descriptions des services[3].

UDDI offre deux fonctionnalités au sein de l'architecture globale qui permet aux fournisseurs de publier leurs services selon un modèle de description prédéfini et au client de rechercher et les services dont il a besoin. La spécification UDDI constitue une norme pour les annuaires des services. Les fournisseurs disposent d'un schéma de description permettant de publier des données concernant leurs activités, la liste des services qu'ils offrent et les détails techniques de chaque service. Elle offre aussi une interface aux applications clientes, pour consulter et extraire des données concernant un service et/ou son fournisseur.

[1] Soumit au W3C par UserLand, Ariba, Commerce One, Compaq, Developmentor, HP, IBM, IONA, Lotus, et Microsoft en mai 2000

[2] Proposé au W3C par Ariba, IBM et Microsoft en mars 2001, la première version du standard a été proposé par le W3C en 2002

[3] UDDI est le résultat d'un accord inter-industriel proposé par Dell, Fujitsu, HP, Hitachi, IBM, Intel, Microsoft, Oracle, SAP, Sun, etc. Plus de 220 acteurs sont membres du réseau UDDI

L'Architecture XSOA

L'apparition du concept de service a fait apparaître de nouvelles opportunités de développement d'applications par composition de services. En effet, il est parfois nécessaire de combiner ou de composer un ensemble de services en services plus complexes, qu'on appelle *service composite,* pour répondre à des exigences plus complexes.

La composition de services a donné naissance à une extension de SOA: l'*Architecture Orientée Service* étendue (xSOA – extended Service Oriented Architecture) [Papazoglou03] [Papazoglou05]. Par ailleurs, cette architecture étendue expose un certain nombre de fonctionnalités complémentaires. Par exemple, dans [Papazoglou03], un service doit être décrit et découvert non seulement en fonction de son interface mais aussi en considérant d'autres aspects à savoir, la sécurité, la disponibilité, les performances et tout ce qui est lié à la qualité de services (QoS – Quality of Service).

L'architecture xSOA est structurée en couches :

- La première couche met en place les fondements des services de base en utilisant l'architecture SOA.
- La deuxième couche s'intéresse aux services composites en proposant les méthodes et outils nécessaires à leurs exécutions. De plus, elle comprend des fonctionnalités liées à la qualité des services composites.
- La troisième couche s'intéresse à la gestion, la supervision et la sécurisation des services en mettant en œuvre l'ensemble des outils et ressources nécessaires.

L'architecture xSOA attache à chacune des trois couches des aspects spécifiques d'ingénierie des services. Celle ci est définie comme l'activité qui consiste à analyser, modéliser et développer les techniques et méthodologies nécessaires pour les approches à services basiques et/ou composites.

2. POSITION ADOPTEE DANS CETTE THESE

Il n'y aucun doute sur le fait que SOA 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 plate-formes hétérogènes qui peuvent évoluer sans que leurs architectures de base soient remises en cause. Force est cependant de constater que SOA 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.

En même temps, l'acceptation et la popularité croissantes du paradigme SOA font émerger une demande d'extension du paradigme logiciel au monde de l'entreprise pour automatiser par composition de services, les processus du business intra et inter entreprises tels que les ventes, les chaînes de production, les ressources humaines entre autres. Toutefois, pour que le paradigme SOC (qui, comme son nom l'indique est du niveau du 'calcul') 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. [Arsanjani04], [Zimmermann04].

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 SOA peut porter ses meilleurs fruits.

C'est à cette extension que s'intéresse la thèse.

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*, c'est-à-dire par référence à des buts et stratégies que l'organisation choisit pour les atteindre. Nous les qualifions de *services intentionnels*. 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 en WSDL et celles des besoins organisationnels tels que le monde du business les formule naturellement.

En outre, en ligne avec Sawyer [Sawyer05], notre croyance est que la découverte et la spécification des services ne peut pas être déconnectée de leur usage, c'est-à-dire des processus du business qu'ils servent. Notre position est donc de découvrir et de décrire les services intentionnels nécessaires à une organisation en développant une perspective de modélisation des processus. Par homogénéité entre la notion de service intentionnel et le processus d'élucidation des services du business, nous proposons de modéliser les processus du métier de l'entreprise dans une perspective elle-même intentionnelle.

Finalement, notre position nous conduit à proposer un déplacement de toute la perspective SOA centrée '*fonction*' (cf. Figure 1) vers une position équivalente mais centrée '*intention'* (cf. Figure2). Dans cette perspective, les services sont décrits dans les termes intentionnels du business, c'est-à-dire en termes d'intentions et de stratégies pour les atteindre et leur

publication, leur recherche et leur composition se fait sur la base de ces descriptions intentionnelles. De cette façon, on peut parler d'un portage de SOA au niveau intentionnel que nous qualifions de *iSOA* (intentional Service Oriented Architecture).

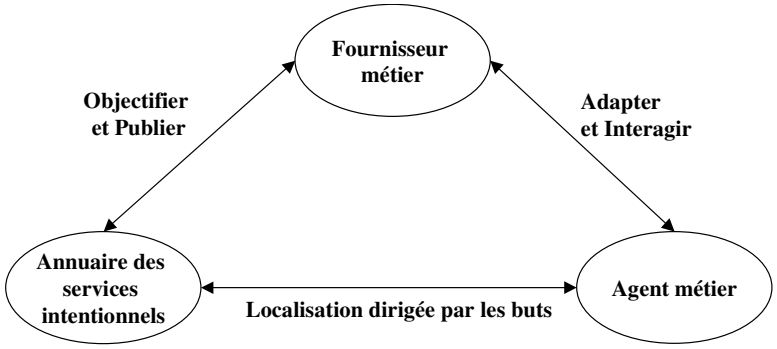


Figure 2. Architecture iSOA

L'architecture iSOA (cf. Figure 2) hérite des mêmes rôles et opérations que SOA mais s'en différencie par les deux points suivants :

(i) les agents métiers remplacent les agents logiciels au niveau des interactions, et

(ii) la description intentionnelle du service remplace la description technique.

Dans la vision iSOA explorée dans cette thèse les organisations qui offrent des e-services centrés 'business' (les fournisseurs métier) les décrivent dans un mode intentionnel (intentions et stratégies) et les publie dans un annuaire de services intentionnels. L'agent métier (le client) qui recherche un ou plusieurs services coïncidant avec ses exigences effectue une localisation dirigée par les intentions. Une fois le service localisé, l'agent métier peut interagir directement avec le fournisseur métier et exécuter les services par adaptation aux conditions spécifiques du moment.

Les trois rôles de l'architecture iSOA structurent la discussion qui suit et servent à introduire les propositions de cette thèse tout en les situant par rapport à l'état de l'art.

L'annuaire iSOA pose la question de la notion de *service intentionnel* et de sa relation avec le concept de service logiciel. Nous verrons que la réponse à cette question prend la forme d'un modèle de service intentionnel, $\mathcal{MiS}$ et nous montrerons qu'une description d'un service intentionnel doit comporter l'expression de *variabilité*, c'est-à-dire des variations de chacun des services composants. Le modèle sert à alimenter l'annuaire en services intentionnels.

Le fournisseur doit définir les services intentionnels à stocker dans l'annuaire. Cet aspect pose la question de la *démarche* à préconiser pour identifier et décrire les services du business qui sont mis à disposition grâce à l'annuaire des services. Nous proposons de modéliser l'intentionnalité d'un business par un graphe d'intentions et de stratégies à partir duquel un ensemble de règles méthodologiques permet de dériver les services intentionnels.

Finalement, le fournisseur doit mettre à la disposition de l'agent métier une architecture d'exécution des services appropriée à leur description intentionnelle. Nous répondons à cette demande par une *architecture iSOA à agent* permettant une exécution des services dirigée par les intentions et permettant l'adaptation de l'application aux conditions spécifiques du moment.

3. DEFIS ET PROBLEMATIQUES DE LA THESE

Nous étudions dans cette thèse les problèmes posés à la mise en place de l'architecture iSOA. Nous nous intéressons à un changement d'échelle d'observation qui se focalise sur les dimensions stratégiques et intentionnelles des clients au niveau (i) des fondements des services, leur composition et leur exécution, et (ii) de la discipline de l'ingénierie des services.

Comme indiqué ci-dessus la thèse s'attaque aux trois problèmes suivants que nous développons dans les sections suivantes :

- le modèle des services intentionnels, $\mathcal{MiS}$
- la démarche de découverte des services intentionnels et,
- l'exécution intentionnelle de services.

3.1. La notion de service intentionnel

La problématique clé de cette thèse est centrée autour de la définition de la notion de service intentionnel, de la variabilité et de la composition intentionnelle. On rappelle les principales positions et définitions de la notion de service et de la composition de services, puis on introduit notre proposition.

3.1.1. Intentionnalité

Situation actuelle

L'architecture x*SOA* considère la définition du concept de service au niveau de la première couche en définissant un service comme étant une application accessible à partir du web, utilisant les protocoles de communication d'Internet et utilisant un langage standard pour décrire son interface.

Plusieurs modèles sont utilisés pour représenter les services, tels que les diagrammes d'états [Mecella02a], les diagrammes d'activités [Benatallah02] ou encore les automates [Berardi03]. La plupart des travaux actuels se concentrent sur l'expression des services à travers les fonctionnalités qu'ils proposent.

Le point de vue qui prime est clairement celui du développeur. En revanche, l'expression des services dans des termes compréhensibles par l'utilisateur final et à fortiori par un gestionnaire ou manager reste encore un problème non résolu. Nous pensons que les notations existantes, reposant par exemple sur les activités, les états ou les automates, ne sont pas adaptées à la description de services pour les utilisateurs du business. Ces modèles décrivent ce qu'une application peut offrir comme fonctionnalités mais sans mentionner en quoi celles ci peuvent répondre aux besoins des clients. Par ailleurs, les modèles cités contiennent de nombreux détails techniques qui n'intéressent pas l'utilisateur final.

Le problème soulevé ici est celui de la mise en correspondance entre les besoins des clients en termes de services et les services logiciels proposés par les développeurs. Comme le montre la Figure 3, la difficulté de la mise en correspondance tient à la disparité des langages dans lesquels les deux parties en présence, les développeurs et les clients, ont l'habitude de s'exprimer. Comme de nombreux auteurs l'ont observé, il y a une « discordance conceptuelle » [Arsanjani04] [RDW95] [Dittrich94] [ObjectMatter05] [Woodfield97] [Lausen97]. En effet, d'un coté le développeur se place à un niveau opérationnel, alors que de l'autre coté les clients se placent au niveau intentionnel.

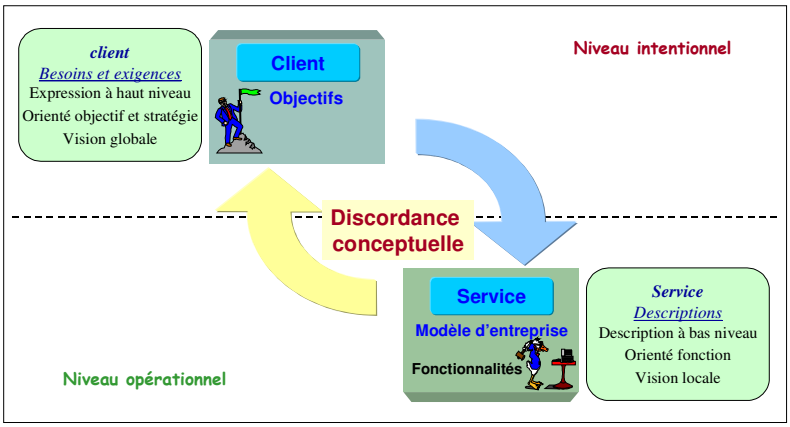


Figure 3. Problème de discordance conceptuelle pour la mise en correspondance du client et des services

La description d'un service au niveau opérationnel se focalise sur le « quoi » et le « comment » des opérations et méthodes ; par exemple, sur les données qu'elles doivent fournir et les opérations qui doivent être effectuées. Rétablir une mise en correspondance entre ces services et les besoins des clients n'est pas une tâche simple. D'un autre coté, les clients qui souhaitent utiliser un service possèdent un certain nombre d'objectifs à satisfaire. Pour cette raison, ils souhaitent faire une mise en correspondance au niveau le plus haut (niveau intentionnel) entre leurs objectifs et les objectifs que les services pourraient aider à réaliser. L'expression des objectifs est intentionnelle, contrairement aux services qui sont exprimés de manière opérationnelle.

Dans [Tansey0x], l'auteur nous invite à utiliser la notion de service selon deux perspectives. La première est celle du métier où le service est considéré comme une interaction métier entre une organisation et un client. La deuxième perspective est technique puisque le service est considéré comme un composant logiciel. Ces deux perspectives sont liées au fait que le service technique permet de répondre aux besoins stratégiques énoncés dans le service métier. [Penserini06a] et [Perini05], de leur coté, démontrent que l'expression des services selon le point de vue de l'utilisateur final est indispensable et que des langages de haut niveau doivent être définis.

Notre position

Pour réduire la discordance conceptuelle et dans la lignée des travaux cités ci-dessus, notre proposition est celle d'une meilleure homogénéité dans l'expression des besoins des clients et des services logiciels. Cela conduit à repenser la notion de service technologique et sa « raison d'être » pour s'adapter à l'échelle stratégique et intentionnelle des clients.

Nous proposons de positionner la description du service au niveau intentionnel avec l'objectif de mettre en avant le but que ce service permet d'atteindre et non plus la fonctionnalité qui permet son atteinte.

La proposition de la thèse repose sur le concept de *service intentionnel* qui abstrait les détails du service logiciel et de ses fonctionnalités pour se concentrer sur son essence, à savoir le but qu'il permet d'atteindre.

En conséquence, la mise en correspondance entre un service et un besoin organisationnel, s'établira au niveau intentionnel au moyen de rapprochement entre modèles de buts (cf. Figure 4.

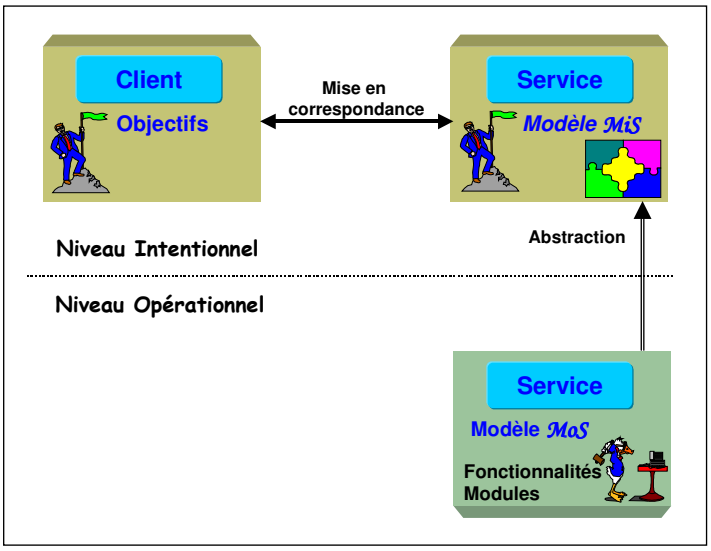


Figure 4. Mise en correspondance

Le concept de service intentionnel est au cœur du modèle $\mathcal{MiS}$ (Modèle Intentionnel de Services) proposé dans cette thèse. Le modèle $\mathcal{MiS}$ définit chaque service intentionnel en tant que brique de construction d'applications en lui associant la connaissance situationnelle et intentionnelle sous la forme d'une interface. Chaque service intentionnel s'applique dans une situation particulière pour réaliser une intention particulière.

Dans l'architecture iSOA, l'annuaire des services permet aux fournisseurs de publier leurs services selon les termes du modèle $\mathcal{MiS}$ et au client, de localiser les services répondant à leurs attentes par coïncidence d'intentions.

3.1.2. Variabilité

Etat de l'art

Dans un environnement SOC, un service peut invoquer plusieurs services comme faisant partie de sa composition. La composition de services est l'instrument permettant de faire face à la diversité des besoins, un même service pouvant être partie de différentes compositions. C'est par la composition que SOC répond aux besoins de variabilité.

Le besoin de variabilité dans le monde des services a émergé à la suite d'un changement de comportement des utilisateurs qui ne veulent plus s'adapter aux capacités des logiciels mais préfèrent que ces derniers soient configurés selon leurs besoins. Des travaux ont permis d'identifier ce que sont les conditions de changement qui peuvent être selon [Tapaloglu04] (i) techniques, lorsque l'invocation d'opérations et des types de paramètres diffèrent, (ii) contextuelles, dans le cas par exemple d'une stratégie de remboursement qui peut varier d'une banque à une autre. [Velasco05] met l'accent sur le contexte d'utilisation et [Chevrin06] [Comerio04] sur le canal de communication utilisé alors que [Penserini06c] introduit les préférences des utilisateurs. Toutefois, la prise en compte des conditions de changement incombe au concepteur qui doit anticiper les variantes d'usage. La solution consiste à développer autant de compositions que de variations d'usage.

Il n'y a donc pas de variabilité dans les applications à base de services au sens où le génie logiciel la définit comme étant « la capacité d'un système ou un artefact à être changé, personnalisé ou configuré selon un contexte d'utilisation particulier » [VanGurp01].

Notre proposition

Notre proposition est d'intégrer la variabilité dans le modèle de service.

Elle se conforme en cela à la définition de la variabilité logicielle qui impose que le design du logiciel comporte des points de variation et les variations qui y sont attachées. Mais en même temps, elle s'adapte à la définition orientée-intention du service $\mathcal{MiS}$.

Notre proposition consiste à introduire la variabilité dans la manière d'atteindre le but du service. Les variantes correspondent aux différentes manières d'atteindre le but. Comme un service intentionnel peut être composé de services intentionnels ayant chacun leur propre but et donc des variantes associées, il en résulte que le service intentionnel se définit comme un réseau de variantes attachées à des points de variation.

Nous pensons que l'encapsulation de services alternatifs dans la définition d'un service intentionnel est d'une part, inhérente au mode intentionnel (prise de décision parmi un ensemble de choix) et d'autre part, apporte la flexibilité nécessaire à l'adaptation dynamique du service intentionnel aux conditions particulières de son exécution.

La variabilité de service proposée dans cette thèse s'apparente à la définition d'un service vu comme une famille de services de façon analogue à une famille de produits logiciels. Une famille de produits est définie comme étant «un ensemble de logiciels partageant un ensemble commun de caractéristiques qui satisfont les besoins d'un segment particulier du marché et qui sont développés à partir d'un ensemble commun d'artefacts logiciels » [Clements02]. Les logiciels d'une famille de produits possèdent des caractéristiques communes mais aussi des variations qui leur permettent de répondre aux besoins particuliers de différents ensembles d'utilisateurs. De manière analogue, la variabilité introduite dans le modèle $\mathcal{MiS}$ permet de mettre en évidence les caractéristiques communes à tout processus d'atteinte d'un but tout en rendant explicites celles qui sont des variations permettant de répondre aux besoins des clients de manière différenciée. Ceci présente deux principaux avantages :

- La réutilisation des parties communes [Ommering02] [Tomphson01] et,
- L'adaptation des produits à différents clients et à différentes situations organisationnelles [Svahnberg01].

3.1.3. Composition intentionnelle de services

Etat de l'art

La composition de services est un des atouts majeurs des approches à base de services. Les services sont conceptuellement limités à des fonctionnalités modélisées par une collection d'opérations qui s'apparentent à des méthodes du monde objet. Pour répondre aux exigences de certains types d'application, il est donc nécessaire de combiner un ensemble de services en services plus complexes appelés services composites.

La composition de services est un des champs de recherche actif du domaine des services. Elle est apparue comme un instrument incontournable du paradigme SOC. De nombreux travaux de recherche ont abordé le problème en proposant des modèles et des notations diverses [Yang02] [Pistore04] [Benatallah03] [Hull03] [Berardi05] [Dijkman03] [Quartel04b] et [Mecella02a]. Il est toutefois constant d'observer que les modèles existants se centrent sur

la définition de l'ordre dans lequel les services sont appelés ainsi que le flux des informations qui transitent d'un service à un autre. La description fait appel à de nombreux détails techniques tels que les opérations à invoquer, les paramètres à fournir ou encore la plateforme de composition choisie.

Une des propriétés des services composites reconnue comme indispensable aujourd'hui est la *réflexivité*. La réflexivité permet de considérer un service composite comme un service à part entière, réutilisable et pouvant faire partie de nouvelles compositions comme tout autre service.

La réflexivité n'a pas fait d'emblée partie de la composition de services. De nombreux travaux comme [Berardi03] [Benatallah03] [Hull03] [Orriëns03] se concentrent plutôt sur les méthodes de composition ainsi que sur la découverte des services à prendre en considération dans une composition.

Nous pensons que, la réflexivité est une propriété importante à préserver dans la composition. Cela a des implications. Par exemple, afin d'assurer la réflexivité des services composites, il est nécessaire d'avoir une séparation claire entre la description de la composition et l'interface. Dans certaines propositions de standard actuelles, comme [WSCI02] [WSCL01], les deux aspects sont confondus, ce qui aboutit à la construction de services composites monolithiques qui ne sont pas directement réutilisables dans d'autres compositions. La propriété de réflexivité fait par conséquent défaut.

A l'opposé certains auteurs, comme [Yang02], démontrent qu'il est crucial de mettre en œuvre des mécanismes supplémentaires au niveau d'un service composite pour le rendre réflexif et par conséquent réutilisable, dans d'autres compositions de services. Notre travail s'inscrit dans cette lignée.

Notre proposition

La composition dans le modèle $\mathcal{MiS}$ est forcément de nature différente des solutions évoquées ci-dessus. Elle doit s'adapter à la perspective intentionnelle du modèle et débouche sur une composition dirigée par les buts. Notre proposition prend donc la forme d'une composition de services dirigée par les buts/intentions qui permet d'exprimer la composition à l'échelle stratégique et intentionnelle des clients.

On entend par composition de services dirigée par les buts le fait que le service composite est associé à un but de haut niveau et sa composition suit la décomposition du but père en sous buts.

On verra que cette composition s'inspire des graphes ET/OU des arbres de buts utilisés en particulier en ingénierie des besoins [Dardenne91], [Dardenne93] [Yu94] [Rolland98b]. La composition de services dirigée par les buts introduit une composition à plusieurs niveaux : le

but du service de plus haut niveau qui peut être de nature stratégique est décomposé en sous buts/services qui eux-mêmes, peuvent nécessiter une nouvelle dé/composition jusqu'à atteindre des buts/services opérationnalisables. Par conséquent, une composition de services à un niveau n+1 est un service à un niveau n. Il y a donc une composition récursive des services.

Comme mentionné précédemment nous avons retenu la propriété de réflexivité et donc un service intentionnel composite est un service intentionnel à part entière.

3.2. Démarche de découverte des services intentionnels

La seconde problématique de la thèse est celle du fournisseur métier qui développe des services intentionnels et les met à disposition des agents métier dans l'annuaire des services. Cette problématique est d'ordre méthodologique puisqu'il s'agit de découvrir et spécifier les services intentionnels publiables selon les termes du modèle $\mathcal{MIS}$. Complémentairement, elle vise à associer les services intentionnels à des services logiciels exécutables au moyen des standards du marché.

Etat de l'art

Le paradigme SOC fait émerger le besoin d'une nouvelle manière de développer des applications. Au départ, les applications ont été développées de manière intuitive, de façon 'ad- hoc'. Cependant, cette manière de procéder a rapidement démontré son incapacité à faire face à la complexité croissante du développement des applications. Cette complexité est due, d'une part, à l'évolution de la logique métier et d'autre part, à l'apparition de nouvelles technologies.

En conséquence, on a vu émerger des travaux de recherche portant le développement de méthodologies spécifiques pour concevoir, analyser, modéliser et produire des applications à base de services.

[Papazoglou05] met l'accent sur ce problème en intégrant la discipline de l'ingénierie des services à chacune des trois couches de base de l'architecture *xSOA*. Il définit l'ingénierie des services comme l'activité consistant à analyser, modéliser et développer les techniques et méthodologies nécessaires pour les approches à base de services.

Un nombre croissant de recherches s'intéressent aux approches méthodologiques pour le développement d'applications à base de services [Papazoglou06a] [Yang03] [Penserini06a] [Mylopoulos00].

L'intérêt de ces approches est reconnu mais leur mise en œuvre reste limitée. Il y a un accord unanime entre [Zimmermann04], [Colombo05], [Ingolf04] et [Arsanjani04] sur le besoin

d'introduire une méthodologie qui couvre tout le cycle de vie depuis les besoins du client jusqu'au niveau opérationnel du développeur. Mais peu de solutions répondent à ce besoin.

Une approche méthodologique complète pour capturer les besoins des utilisateurs pour définir, composer et exécuter les services fait par conséquent défaut.

Notre proposition

Nous pensons que les services qui constituent la population référencée par l'annuaire des services sont issus des processus du business des organisations. Les services sont en relation avec les objectifs du business et, manifestement, aident à les atteindre. Il semble donc naturel de donner au fournisseur métier les moyens de construire un modèle du business à partir duquel il soit aisé d'identifier les services, de les décrire selon les termes de $\mathcal{MiS}$ et de les publier.

Nous avons choisi d'utiliser le système de représentation de la *Carte* [Rolland00] pour modéliser le business dans des termes intentionnels et de développer un ensemble de règles méthodologiques pour dériver les services d'une carte intentionnelle.

La *Carte* est un système de représentation qui permet de modéliser processus dans des termes intentionnels. Il offre un mécanisme de représentation basé sur un ordonnancement non déterministe d'intentions et de s*tratégies*. Une *carte* est un graphe orienté dans lequel les nœuds sont des intentions et les arcs, des stratégies. Un arc entrant identifie une *stratégie* qui permet de réaliser l'intention associée au nœud cible. Le graphe montre par conséquent quelles intentions peuvent être réalisées au moyen de quelles stratégies une fois qu'une intention a elle-même été préalablement réalisée. Les cartes proposent aussi un mécanisme d'affinement qui peut être modélisé par un lien d'affinement allant d'un couple intention / stratégie vers une autre carte. L'affinement permet de décrire les intentions et stratégies à différents niveaux de détail.

Nous considérons que le modèle de la carte est adapté à la découverte des services. En effet, il met en évidence les différentes stratégies pour satisfaire le même but et aussi les différentes combinaisons de stratégies et de buts pour atteindre un but. Il aide donc aussi à la découverte de la variabilité des services. Chaque combinaison possible de buts et de stratégies identifie une variante de services possible pour atteindre le but du service global.

Nous pensons que la proposition consistant à dériver les services d'un modèle du business centré buts offre les avantages suivants :

(i) Modélisation explicite de l'intentionnalité

Les modèles dirigés par les buts modélisent explicitement l'aspect intentionnel du système (le « pourquoi »). Il a été reconnu récemment que le point de vue intentionnel d'un système

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 buts facilitent la sélection des services.

(ii) Simplicité du langage

Les modèles des besoins et en particulier ceux dirigés par les buts utilisent un langage facilement compréhensible par des utilisateurs non experts d'un domaine à la différence des modèles de services qui exigent des connaissances techniques approfondies du domaine.

La démarche proposée comporte un ensemble de règles méthodologiques pour (a) construire la hiérarchie de cartes et (b) identifier les services de manière systématique à partir de la carte et pour les représenter dans les termes du modèle $\mathcal{MiS}$.

Elle permet également de guider l'identification, au niveau opérationnel, de services qui correspondent aux services intentionnels de type $\mathcal{MiS}$. Ceci est réalisé à l'aide la représentation explicite des services opérationnels qui découlent des services intentionnels dans les termes d'un modèle spécifique $\mathcal{MoS}$ (Modèle Opérationnel des Services) et un ensemble de règles méthodologiques pour identifier les services opérationnels à partir des services intentionnels.

3.3. L'exécution intentionnelle de services

La dernière problématique de cette thèse est relative à l'exécution des services et donc d'une composition de services, dirigée par les buts. Elle est mise en œuvre par le fournisseur de service à l'attention du client.

Etat de l'art

De nombreux modèles et architectures d'exécution des services sont disponibles dans la littérature. Certains sont spécifiques [Yang03] [Shegalov01] [Wodtke97] [Mecella02a] [Benatallah03] [Fauvet01] [Penserini06] [Lopes05] [Melliti04] et [Berardi05].

D'autres reposent sur des standards tels que les langages procéduraux de type BPEL4WS (Business Process Execution Language for Web Services) [Andrews03]. Ces langages spécifient les échanges de messages entre les différents services sans spécifier leur composition interne d'une part, et spécifient l'ordre d'exécution des processus impliqués et les messages échangés, d'autre part. L'exécution des services basée sur BPEL4WS par exemple, s'apparente à l'exécution d'un programme. Ecrire une coordination dans l'un de ces langages est assez similaire au fait de la programmer. On peut prendre la mesure de cette

dimension en observant la correspondance “1 à 1” entre le processus écrit dans un langage de coordination et l’opération du service qui l’implémente. On peut aussi remarquer la rigidité du lien entre l’activité et son implémentation, exprimé "en dur" dans le fichier de description de la coordination. Ceci est proche d’un appel de méthode dans un langage de programmation. Le processus de coordination est essentiellement une succession d’appels à des opérations de services.

Certaines approches utilisent des architectures répandues qu’ils adaptent à leurs besoins. Parmi ces architectures, on retrouve celles orientées peer-to-peer [Akram03] [Schuler03] ou celles orientées « grid computing » [Alpdemir03].

Il faut noter que l’exécution des services suit le modèle de coordination défini, sans aucune déviation. On peut dire que la plupart des plate-formes d’exécution opèrent une exécution statique des services.

Notre proposition

Notre contribution sur ce point prend la forme d’un mécanisme d’exécution complémentaire à celui qu’offre les standards. Nous utilisons ces mécanismes pour le contrôle de l’exécution des services logiciels décrits selon les termes de $\mathcal{MoS}$ et qui sont mis en correspondance avec les services intentionnels $\mathcal{MiS}$ comme on le montre au Chapitre 4. En revanche, un service intentionnel $\mathcal{MiS}$ prend la forme d’une composition à choix, c’est-à-dire qui offre des chemins différents composés de services différents pour atteindre le but du service. Pour tirer profit de la variabilité offerte dans la manière de fournir un service, il nous a semblé utile de développer un mécanisme d’exécution des services $\mathcal{MiS}$ qui guide l’agent métier dans le choix des variantes qui sont le mieux adaptées à la situation présente.

Le modèle $\mathcal{MiS}$ sert donc à l’implémentation d’une application orientée services *adaptable* c’est à dire *personnalisée par l’utilisateur*. Cette personnalisation est dynamique puisqu’elle s’opère au moment de l’exécution du service composite.

Nous proposons une architecture spécifique pour implémenter une application adaptable. La solution retenue consiste à implémenter les services intentionnels sous forme de composants architecturaux que nous appelons *Agents*. Chaque agent implémente un service particulier du modèle $\mathcal{MiS}$. Nous distinguons deux sortes d’agents : des agents qui gèrent les services atomiques et des agents qui gèrent les services composites en permettant l’exécution par navigation dans l’arbre d’affinement ET/OU qui est le fondement de la composition.

Nous avons défini des règles méthodologiques permettant au développeur de construire une application adaptable à partir d’un modèle $\mathcal{MiS}$. Ces règles définissent (i) comment les

services sont transformés en composants architecturaux (agents), (ii) comment ces composants doivent être structurés et (iii) comment ils doivent interagir.

Lorsque l'application adaptable s'exécute, elle offre à l'agent métier (le client) les variantes possibles à chaque point de variation rencontré. Il y a donc personnalisation guidée et dynamique, opérée par l'utilisateur final lui-même. Un prototype de démonstration est implémenté.

4. RESULTATS DE LA THESE

Ce mémoire de thèse présente cinq résultats de recherche principaux :

- Un modèle de représentation des services intentionnels qui intègre la variabilité et la réflexivité : le modèle $\mathcal{MiS}$,
- Une démarche pour identifier les services intentionnels à partir des besoins des utilisateurs,
- Un modèle de représentation des services opérationnels : le modèle $\mathcal{MoS}$,
- Une démarche pour dériver les services opérationnels à partir des services intentionnels,
- Une architecture à agents permettant l'exécution intentionnelle des services.

La Figure 5 résume l'approche proposée dans cette thèse. Un des éléments principaux de cette approche est le modèle de services intentionnels $\mathcal{MiS}$ qui définit tout service intentionnel comme un élément réutilisable dans la construction d'autres applications.

Le deuxième élément de l'approche est le processus de génération de services intentionnels. Ce processus utilise en entrée le recensement des besoins du monde du business à l'aide de cartes d'intentions et de stratégies. Le résultat est l'ensemble des services intentionnels qui sont publiés dans l'annuaire des services.

Le troisième élément à prendre en considération est le processus d'opérationnalisation des services intentionnels en services logiciels Ce processus utilise en entrée les services intentionnels de type atomique. Il génère en sortie un modèle opérationnel de services $\mathcal{MoS}$ qui correspond à la caractérisation de l'opérationnalisation du service intentionnel en un ensemble de services logiciels. Ceux-ci sont par la suite traduits dans un langage d'implémentation standard.

Pour tirer le meilleur profit de l'intentionnalité de l'approche et la variabilité, l'exécution est basée sur une architecture orientée agents. L'architecture agents permet une orchestration contrôlée par le client à qui on offre les choix possibles à chaque point de variation et qui

choisi dynamiquement l'option de service qui correspond le mieux à la situation courante. Ainsi, l'architecture par agents nous permet de réaliser des applications à base de services adaptable et personnalisable.

L'ensemble de ces éléments permet de réaliser un prototype facilitant la sélection dynamique des services basée sur le modèle $\mathcal{MiS}$.

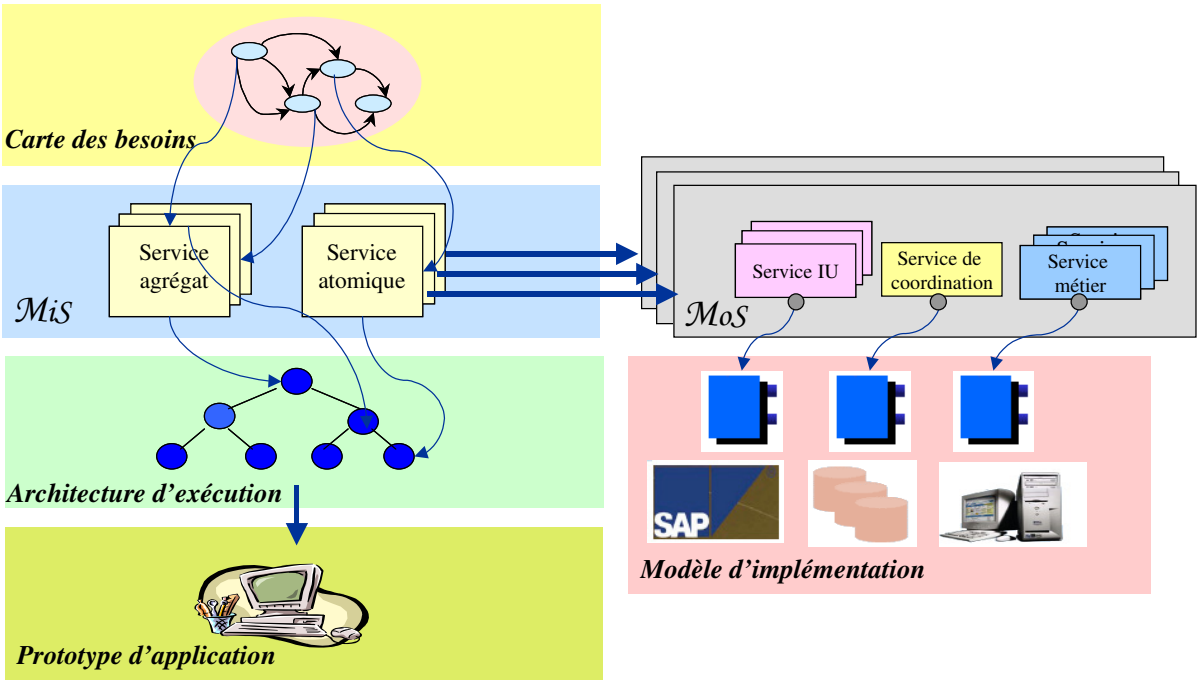


Figure 5. Aperçu de l'approche proposée

5. PLAN DE LA THESE

Le chapitre 2 présente un état de l'art des approches à base de 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.

Le chapitre 3 définit le méta-modèle de services intentionnels $\mathcal{MiS}$. Ce méta-modèle met l'accent sur les manières d'exprimer les différents types de services (les alternatifs, les composites,…) dans une perspective dirigée par les buts.

Le chapitre 4 présente le méta-modèle des services opérationnels MoS. Ce méta-modèle définit les éléments caractérisant les services opérationnels et leurs attributs. Ce chapitre décrit également la démarche d'opérationnalisation des services intentionnels en services opérationnels et les règles à appliquer en vue d'une traduction des services opérationnels dans un langage d'implémentation standard.

Le chapitre 5 décrit le processus qui conduit à l'obtention d'un modèle $\mathcal{MiS}$ à partir d'une modélisation des objectifs du business. Nous expliquons comment modéliser les besoins des utilisateurs dans le modèle de la Carte, et comment les différents concepts du modèle de services sont identifiés à partir d'une carte.

Le chapitre 6 présente l'architecture, que nous avons adoptée pour concevoir et implémenter la composition de services dirigée par un modèle de buts. Cette architecture permet d'exécuter et de contrôler une composition intentionnelle de services.

Le chapitre 7 illustre notre approche par une étude de cas nommée e-Pension.

Le chapitre 8 conclut la thèse et propose des perspectives de recherche faisant suite à ce travail.

CHAPITRE 2

Etat de l'art

1. INTRODUCTION

Ce chapitre est consacré à l'état de l'art du domaine de l'ingénierie des systèmes à base de services. Nous nous intéressons plus particulièrement aux différents modèles concernant la représentation et l'exécution des services proposés dans la littérature.

Ce chapitre a pour but de définir un cadre multidimensionnel permettant l'analyse des approches d'ingénierie à base de services. Ce cadre de référence inspiré de [Rolland98a] est composé de quatre vues. Chacune des vues explore des caractéristiques d'un service. Ainsi, ce cadre permet de (1) identifier des problèmes impliquant les approches à base de services et (2) positionner les approches de recherche les unes par rapport aux autres.

La suite de ce chapitre est organisée comme suit. La section 2 décrit le cadre de référence. Ensuite, la section 3 présente une étude des différentes approches suivant le cadre de référence. Enfin, nous concluons avec la section 5 par un résumé de l'évaluation.

2. CADRE DE REFERENCE POUR LES SERVICES

Selon la Figure 6, le cadre de référence considère la notion de service suivant quatre vues. Chaque vue permet d'analyser un aspect particulier du service en posant une question fondamentale. A l'inverse, chaque service peut être analysé selon les quatre vues. Les quatre questions posées sont celles du « quoi », du « pourquoi », du « comment », et enfin du « par quel moyen ». Ces questions permettent de s'intéresser (1) à l'objet c'est à dire aux éléments utilisés, (2) aux buts de l'approche à base de services, (3) à la méthode mise en œuvre pour atteindre ces buts et (4) aux outils utilisés.

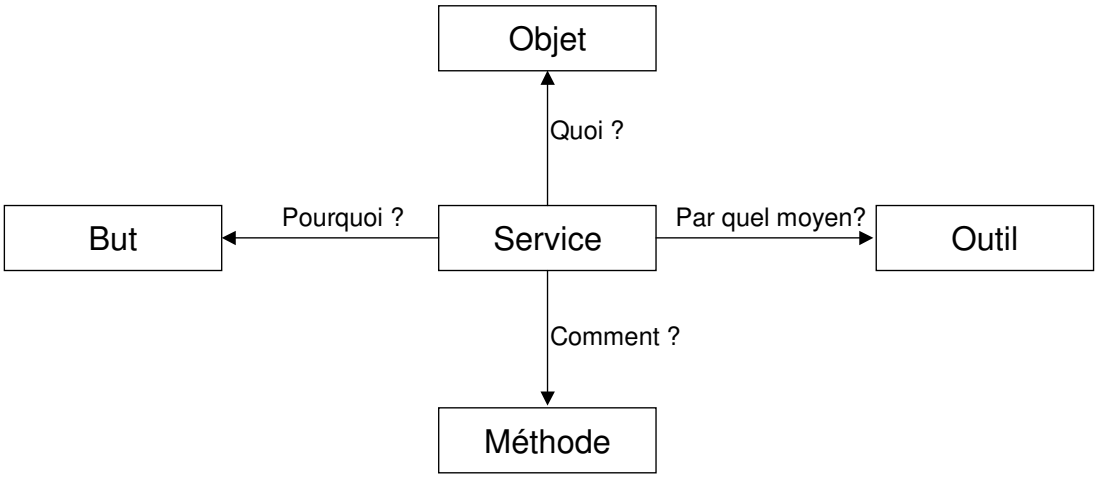


Figure 6. Les quatre vues du cadre de référence

Chaque vue est caractérisée et mesurée à l'aide d'un ensemble d'attributs. Les attributs 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 ()).

Pour un type structuré, il est possible de caractériser les valeurs x et y en ajoutant une valeur explicative dont le domaine est une chaîne de caractères. La valeur explicative est propre à l'approche.

Par exemple, la vue objet comprend les cinq attributs suivants :

- L'entité à laquelle on s'intéresse qui est soit technologique, métier, applicative, interactionnelle ou intentionnelle.
- L'adaptabilité du service à utiliser définie par un booléen.
- L'utilisation réflexive du service définie par un booléen.
- Le type de service qui est soit une boite noire soit boite blanche.

Une approche à base de services est positionnée selon la vue objet suivant les valeurs d'attributs suivantes (cf. Tableau 1):

Tableau 1. Vue objet: attributs, domaines et exemples de valeurs

Attribut	Domaine	Exemple de valeurs
Entité	Enum {technologique, métier, applicatif, interactionnel, intentionnel}	Intentionnel
Adaptabilité	Booléen	Vrai
Réflexivité	Booléen	Vrai
Type de service	Enum {boite noire, boite blanche}	Boite noire

Nous détaillons les attributs des différentes vues dans la section suivante.

2.1. La vue objet

On constate aujourd'hui que la littérature scientifique traitant des services ou même des e-services (on parlera plus simplement d'entités) 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.

Nous proposons d'associer à la vue objet les cinq attributs suivants : *(i)* l'entité, *(ii)* l'adaptabilité, *(iii)* la réflexivité, *(iv)* le type du service et *(v)* la nature du service. Nous détaillons chacun des attributs dans ce qui suit.

2.1.1. Entité

[Baida04] suggère qu'au moins trois perspectives sur les services doivent être comprises afin de fournir une terminologie partagée pour les services :

- Un point de vue technologique (informatique) ;
- Un point de vue métier (business) ;
- Un point de vue applicatif.

En se référant à [Chevrin06], nous pouvons ajouter le point de vue interactionnel d'un service. Il prend en compte l'aspect interaction (IHM) qui expose le point de vue du client au travers d'interfaces utilisateur.

Nous introduisons dans cette thèse, un point de vue intentionnel qui se traduit par le fait qu'un service exhibe une intentionnalité formulée par le *but* qu'il permet à ses clients d'atteindre [Rolland05].

Les différents points de vue déjà cités sont détaillés dans ce qui suit :

- ***La perspective technologique :***

La définition d'un service est fortement influencée par les domaines proches de l'ingénierie du web et le développement de standards liés aux services web. Toutefois, dans ce domaine, le terme service est utilisé comme synonyme de web service ou e-service. [Papazoglou02] définit un service comme un ensemble d'applications modulaires auto-contenues et auto-descriptives qui peuvent être publiées, localisées et invoquées depuis le web. Un service peut effectuer des actions allant de simples requêtes à des processus métiers complexes. Les services permettent d'intégrer des systèmes d'information hétérogènes en utilisant des protocoles et des formats de données standardisés, autorisant ainsi un faible couplage et une grande souplesse vis-à-vis des choix technologiques effectués.

Selon [Papazoglou02], la dernière citation apporte trop peu d'information pour comprendre précisément la nature d'un service. En synthétisant, l'auteur donne les deux définitions suivantes :

- Une définition non informatique : le terme « service » décrit une fonctionnalité commerciale exposée par une entreprise sur Internet afin de fournir un moyen d'utiliser ce service à distance [Piccinelli03] ;
- Une définition détaillée tenant compte des aspects opérationnels. Les services sont des applications modulaires qui peuvent être décrites, publiées, localisées et invoquées dans un réseau [W3C04] [Gustavo04]. L'objectif initial d'un service est de permettre l'utilisation d'un composant applicatif de manière distribuée. Plus clairement, cela consiste à permettre l'utilisation d'une application à distance. Les services facilitent l'invocation de certains traitements via Internet. Le modèle logiciel proposé contient à la fois un nouveau mode de développement et une nouvelle méthode de fourniture de services.

Les services s'appuient sur le triplet de standards suivant : « Web Service Description Language » (WSDL), « Simple Object Access Protocol » (SOAP) et « Universal Description, Discovery and Integration » (UDDI).

- WSDL offre un schéma formel de description des services[4]. C'est toujours dans le but de rendre les services faiblement couplés et autonomes, que la spécification WSDL a vu le jour [CCMW01]. Contrairement aux architectures monolithiques où la description des composants ainsi que les moyens de les invoquer dépendent fortement de l'infrastructure utilisée, la spécification WSDL offre une grammaire qui décrit l'interface des services de telle façon qu'ils se suffisent à eux-mêmes. La spécification

[4] Proposé au W3C par Ariba, IBM et Microsoft en mars 2001, la première version du standard a été proposé par le W3C en 2002

WSDL présente les services comme des boites noires, présentant l'apparence d'un certain nombre de ports d'entrée et de sortie.

- SOAP assure la communication avec et inter services[5]. SOAP permet l'échange de données structurées indépendamment des langages de programmation ou des systèmes d'exploitation. Le protocole SOAP n'est pas concerné par la nature du programme cible ou le traitement des messages, il ne définit pas un environnement d'exécution mais plutôt un modèle de liaison qui assure l'interopérabilité au niveau applicatif entre des applications hétérogènes.
- UDDI offre une manière uniforme de définir des registres des services et aussi un schéma uniformément extensible de descriptions des services[6].

- ***La perspective métier :***

Selon [Pallos01], un service est un groupement logique de composants requis pour satisfaire une demande métier particulière. Par conséquent, 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 dont les fonctions sont généralement liées au commerce électronique et visent finalement à reproduire dans le monde virtuel les transactions commerciales du monde réel (transactions, contrats, facturation, paiement, etc.).

- ***La perspective applicative :***

Le point de vue applicatif permet d'établir un pont entre la perspective métier et la perspective technologique. Le concept de service dans cette perspective n'est pas encore mature, et il existe peu de définitions. Ces définitions oscillent entre la perspective business et la perspective technologique.

Par exemple, [Quartel04] définit le service applicatif comme le service d'une application qui permet de supporter un processus métier. Le service applicatif est exposé par le système d'information pour qu'il soit invocable par les clients ou par d'autres systèmes d'information.

De façon analogue, [Penserini06] suggère de modéliser les services sous la forme de *capacités*. Une capacité est associée à un but donné et définit la manière de le réaliser. La définition du service reste donc au niveau applicatif.

- ***La perspective interactionnelle :***

[5] Soumit au W3C par UserLand, Ariba, Commerce One, Compaq, Developmentor, HP, IBM, IONA, Lotus, et Microsoft en mai 2000

[6] UDDI est le résultat d'un accord inter-industriel proposé par Dell, Fujitsu, HP, Hitachi, IBM, Intel, Microsoft, Oracle, SAP, Sun, etc. Plus de 220 acteurs sont membres du réseau UDDI

Un service de point de vue interactif est vu d'après [Chevrin06] 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.

- ***La perspective intentionnelle***

Le service intentionnel 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 ou par la composition de services dirigée par les intentions. L'avantage de ce modèle est qu'il offre une définition orientée but, compréhensible par les utilisateurs, un mode de composition lui-même dirigé par les buts et un mécanisme d'affinement pour atteindre des buts opérationalisables implémentés par des services applicatifs.

L'entité mise en jeu peut donc être de cinq types différents comme le montre la définition ci-dessous :

Entité : Enum {technologique, métier, applicatif, interactionnel, intentionnel}

2.1.2. Adaptabilité

Un e-Service est défini par [Rust03] comme *« the provision of services over electronic network »* où les réseaux électroniques ont une signification plus large qu'Internet, et incluent par exemple, les réseaux sans fil. Progressivement, le commerce électronique s'est centré vers les services. Cela est justifié par le fait que c'est une approche davantage orientée vers le client, permettant une plus grande efficacité en ce qui concerne la satisfaction de leurs besoins.

Pour [Rust03], cela est réalisé comme suit « *uses of two ways dialogues to build customized services offerings, counting on knowledge about the customer to build strong customer relationships* ». Le service doit donc s'adapter aux besoins des clients afin d'assurer sa pérennité et faciliter son utilisation. Il peut donc être potentiellement utilisable par des utilisateurs de profils hétérogènes et ayant des besoins différents.

Nous pouvons retenir que :

- D'une part, les services peuvent êtres composés et adaptés dynamiquement soit à l'initiative de l'application à partir de connaissances acquises sur le client (par une relation d'apprentissage [Peppers97]), soit simplement par le client/utilisateur de cette application. Par ailleurs, le client/utilisateur peut être assimilé, par extension, à un co-producteur du service (notion de self-services [Benatallah03]).

- D'autre part, la portée d'une approche à base de services pour le commerce électronique est d'étendre les canaux en amont et en aval d'une organisation en direction de ses clients, ou utilisateurs, où les canaux ne sont pas seulement le web ni même supportés par Internet.

Les travaux de recherche existant sur l'adaptabilité des services mettent l'accent sur le contexte d'utilisation mais peuvent couvrir aussi le profil de l'utilisateur. [Velasco05] propose d'intégrer le profil de l'utilisateur dans la description WSDL du service. Ce profil comprend tant les caractéristiques personnelles de l'utilisateur que les caractéristiques du contexte d'utilisation du service. Dans ce cas, on parle de service adaptable en fonction du *profil*.

[Chevrin06] et [Comerio04] proposent un service adaptable en fonction des interactions et particulièrement en fonction du canal de communication utilisé. Il peut être réutilisé dans de nombreuses configurations de dispositifs (canal de communication ou couplage de canaux) pour des cibles différentes et pour des organisations différentes. Dans ce cas, on parle de service adaptable en fonction du *canal*.

[Penserini06c] propose un service adaptable en fonction des préférences des utilisateurs. L'utilisateur dans ce cas, sélectionne le service qui correspond le mieux à ses préférences. Dans ce cas, on parle de service adaptable en fonction du *besoin*.

L'attribut *adaptabilité* est défini donc comme suit :

Adaptabilité : enum {profil, canal, besoin}

2.1.3. Réflexivité

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.

La réflexivité est le fait que le service composite obtenu soit réutilisable dans d'autres compositions. [Yang03] 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.

L'attribut de la réflexivité est défini comme suit :

Réflexivité : Booléen

2.1.4. Type de service

Selon la définition d'un service proposée dans [Quartel04] « *a regulary interacting or independent group of items forming a unified whole* », on peut distinguer deux perspectives

du service : externe référencée par « *unified whole* » et interne référencée par « *interacting or independent goup of items* ».

La perspective externe tient à présenter le service comme une *boîte noire*. Seulement les entrées/sorties sont gérées sans donner les spécifications internes. Ceci permet un haut niveau de modularité et d'interopérabilité. L'avantage du modèle de message est qu'il permet de s'abstraire de l'architecture, du langage ou encore de la plate-forme 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 couplage lâche au travers d'un échange asynchrone de messages.

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.

Le service mis en jeu peut être de deux types différents comme le montre la définition ci-dessous :

Type de service : Enum {boîte noire, boîte blanche}

2.2. La vue but

Les buts recherchés par les approches à base de services sont peu diversifiés. La vue but compte un seul attribut, intitulé *but*. Cet attribut répond à la question « pourquoi un service est-il utilisé ? ».

L'utilisation des services fait l'objet de méthodes essentiellement centrées autour de trois problématiques :

- *Modéliser* les systèmes d'information à base de services,
- *Composer* des services, et
- *Exécuter* des services.

1- *Modéliser les systèmes d'information à base de services.* De nombreux chercheurs s'intéressent aux différentes approches utilisées pour la *modélisation des systèmes d'information à base de services* [Quartel04] [Papazoglou06b] [Mecella02a] [Henkel04] [Yang03b]. Ces approches méthodologiques permettent de passer graduellement des processus métiers aux services à l'aide d'un ensemble de principes et de directives.

[Tut02] utilise un ensemble de patrons génériques pour identifier les exigences du système. Les patrons sont ensuite instanciés afin d'identifier les services à mettre en œuvre pour réaliser le but du système.

[Penserini06] et [van Lamsweerde00] 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 qui réalisent ces buts.

2- *Composer des services*. La composition des services est un processus qui crée un service à partir d'autres. 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.

La composition de services est modèlisée par l'ordre dans lequel les services seront appelés. Parmi les exemples de modèles, citons les diagrammes d'activités, les réseaux de Petri ou encore les statecharts.

La composition de services peut être dynamique [Yang04]. [Charfi04] et [Yang04] proposent une composition modulaire et [Solanki04] donne les moyens de vérifier la conformité de la composition de services.

[Berardi03] propose un cadre de développement pour la composition de services qui repose sur un ensemble d'automates. Les automates sont étiquetés par un alphabet représentant les actions du service.

[Benatallah03] utilise les diagrammes d'états pour représenter le flux d'invocation des opérations d'un service composite ainsi que les données échangées.

3- *Exécuter des services*. L'exécution des services requiert d'orchestrer l'enchaînement des services selon un canevas prédéfini, et de les exécuter à travers des "scripts d'orchestration". Ces scripts peuvent représenter des processus métier ou des workflows inter/intra-entreprise. Ils décrivent les interactions entre applications en identifiant les messages, et en déterminant la logique et les séquences d'invocation [SOE].

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. Ces interactions peuvent couvrir des applications et/ou des organisations [Peltz03].

[Benatallah02] propose un modèle de coordination d'égal à égal comme support d'exécution des services formant un système d'information. L'exécution des services est distribuée entre un ensemble d'entités logicielles générées pour chacun des services.

[Mecella02b], suggère l'utilisation des réseaux de Petri pour l'exécution des services. Chaque service est contrôlé par un mécanisme d'exécution appelé un moteur d'orchestration. Le moteur fonctionne suivant un ensemble de règles d'exécution. Il contrôle la sélection et l'exécution des services en interprétant leurs modèles respectifs.

En conclusion l'attribut *but* de la vue but est défini comme suit :

But : set of (concevoir, composer, exécuter)

2.3. La vue méthode

La méthode utilisée dans les approches à base de services dépend du but à atteindre. Nous avons donc identifié quatre attributs : (i) méthode de conception, (ii) méthode de composition, (iii) description de la méthode et (iv) processus de la démarche.

Chacun de ces attributs est détaillé dans les sous sections suivantes.

2.3.1. Méthode de conception

Les méthodes de conception peuvent être classées en trois catégories : les approches top-down, les approches bottom-up et les approches mixtes.

Certaines méthodologies de développement des applications à base de 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 [Penserini06a] et [Traverso04] proposent des approches qui utilisent des modèles de buts pour comprendre les besoins des usagers et pour identifier les services qui permettent leur satisfaction. [Quartel04a] propose un ensemble d'étapes qui couvrent la modélisation de l'approche allant du processus métier jusqu'aux services applicatifs qu'il compose. 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 ou bien des parties des processus métier existants.

A l'inverse, il est possible d'adopter une démarche bottom-up permettant la ré-ingénierie 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 'legacy' système et de les présenter sous la forme d'un ou plusieurs services. [Penserini06b] considère par exemple, qu'il est possible de dériver les modèles *i** représentant les buts et acteurs de l'organisation à partir d'un ensemble de plans et d'activités.

Certaines approches combinent les deux types d'approches précédentes. Ainsi, [Quaretl04b] propose une approche top-down sur le plan global pour l'identification des services applicatifs à partir du processus métier. En revanche, la vérification de la conformité des services aux exigences du processus est analysée de façon bottom-up. Nous qualifions ce type d'approche de *mixte*.

L'attribut *méthode de conception* est défini comme suit :

Méthode de conception : Enum {top-down, bottom-up, mixte}

2.3.2. Méthode de composition

Les méthodes de composition de services peuvent être classées : composition *automatique*, composition *semi-automatique* et composition *fixe*.

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 [Benatallah02] supporte la composition automatique des services en utilisant la notion de conteneur de services. Un conteneur est un service qui représente 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* de services est une combinaison des deux approches de composition à savoir automatique et fixe. En effet, les services à composer sont générés automatiquement « 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. D'un autre coté, le processus de mise en correspondance peut générer une série d'alternatives de compositions de services. L'utilisateur doit choisir dans ce cas, les alternatives qui correspondent au mieux à ses attentes.

[Schaffner06] [Sirin04] [Kim04] et [Meyers02] présentent des outils interactifs dédiés à la composition semi-automatique des services. Le comportement des outils se résume 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 *fixe* requiert que les services constituants soient combinés d'une manière fixe (prédéfinie).

L'attribut *méthode de composition* est défini comme suit :

Méthode de composition : set of (automatique, semi-automatique, fixe)

2.3.3. Description de la méthode

La description de la démarche fait appel à des notations. Nous avons identifié trois types de notations.

La première est la notation *informelle* comme par exemple le langage naturel. [Papazoglou06a] suggère une démarche structurée en étapes, directives et principes méthodologiques pour spécifier, construire et implémenter des applications supportant une composition de services entièrement décrite en langage naturel.

La deuxième est la notation *semi-formelle* comme les diagrammes. [Orriëns03] introduit une approche structurée en étapes pour la composition de services qui part de la définition et débouche sur l'exécution des services. L'approche respecte un ensemble de règles de composition qui sont classifiées en cinq catégories. La composition des services se base sur un modèle spécifique constitué d'un ensemble d'éléments abstraits.

La troisième est la notation *formelle.* Ce type de notation se base sur une sémantique formelle bien définie qui permet d'une part, de vérifier les propriétés que la démarche doit remplir et, d'autre part, de valider le processus en utilisant des techniques comme par exemple l'instanciation, la simulation ou l'exécution. [Berardi03] propose un cadre de développement formel pour composer les services. Les techniques utilisées vont de la spécification des services en utilisant des automates jusqu'à la composition automatique de ces services en suggérant des algorithmes reposant sur le langage PDL (Propositional Dynamic Logics).

L'attribut *description de la méthode* est donc défini comme suit :

Description de la méthode : Enum {formelle, semi-formelle, informelle}

2.3.4. Processus de la démarche

Le processus est « un ensemble d'activités inter-reliées et menées dans le but de définir un produit » [Franckson91]. 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 [Dowson88], les modèles de processus peuvent être classés en trois groupes:

- Les modèles *orientés-activité* se focalisent sur les activités exécutées pour élaborer un produit et sur leur ordonnancement. Par exemple, [Papazoglou06a] 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* couplent l'état du produit à l'activité qui génère cet état. Ils visualisent le processus comme un diagramme de transition d'états. Les travaux de [Quartel04a] s'inscrivent dans ce groupe. En effet, il propose un processus méthodologique pour la modélisation des services structurée en étapes. A chaque étape, l'état du service change.

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

Une nouvelle catégorie de modèles processus a été ajoutée : les *modèles contextuels*

Les modèles contextuels définissent les processus à travers la combinaison de situations observables à un ensemble d'intentions spécifiques. 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 où l'on se trouve au moment de le réaliser.

L'attribut processus s'appuie sur cette typologie établie des processus et est défini comme suit :

Processus de la démarche : Enum {activité, produit, décision, contexte}

2.4. La vue outil

La vue outil comporte trois attributs : (i) outil de composition, (ii) outil d'exécution et (iii) architecture d'exécution.

Chacun de ces attributs est détaillé dans l'une des trois sous sections ci-dessous.

2.4.1. Outil de composition

La composition de services est présentée par un modèle qui définit l'ordre dans lequel les services seront appelés. Parmi les exemples de modèles, citons les diagrammes d'activités, les réseaux de Petri ou encore les statecharts.

[Yang03] propose de réutiliser et composer des services existants dans un « composant service ». Ainsi, l'ensemble des opérations offertes est décrit à l'aide d'une interface uniforme. Les « composant services » sont spécifiés en utilisant le langage SCSL (Service Composition Specification Language).

[Pistore04] et [Benatallah03] définissent le service par un diagramme d'états qui représente le flux d'invocation des opérations des services composants ainsi que les données échangées.]

[Hull03] et [Berardi04] modélisent les services à l'aide d'automates. Chaque automate consiste à définir le comportement du service dans des situations données. L'automate est constitué d'un ensemble d'états et d'un ensemble de transitions étiquetées entre ces états qui décrivent l'évolution du service. Bien que structurellement simple, ce formalisme permet de modéliser la plupart des aspects des services avec une précision souvent suffisante Il joue un rôle important dans la définition de leur sémantique.

[Dijkman03] suggère d'utiliser un modèle conceptuel générique exprimé avec le langage *ISDL* (Interaction Systems Design Language) [Quartel04b] afin de modéliser les concepts

caractérisant un service. Cet outil de composition couvre plusieurs niveaux d'abstraction de définition d'un service.

[Mecella02b] représente le service par un modèle conceptuel constitué de deux parties : l'interface et l'ensemble des interactions dans lesquelles l'e-service est impliqué. L'interface du e-service est spécifiée par un diagramme de classes qui représente les données utilisées en entrée et en sortie. Les interactions sont spécifiées par un diagramme d'états qui représente le flux d'invocation des opérations ainsi que les données échangées.

L'attribut *outil de composition* est défini comme suit :

Outil de composition : Set of (langage (valeur), notation (valeur))

2.4.2. Outil d'exécution

[Yang03] utilise une structure d'exécution appelée SCEG (Service Composition Execution Graph). SCEG est un arbre où chaque nœud est un service composite et dont les fils respectifs en sont les constituants. La racine du graphe forme l'application qu'on veut développer. Le type de composition ainsi que la dépendance entre les messages sont indiqués au niveau de chaque nœud.

[Shegalov01] propose un moteur pour le support d'exécution des services. Les services sont spécifiés par un diagramme d'états qui représente le flux d'invocation des opérations des services composants ainsi que les données échangées [Wodtke97].

D'après [Mecella02b], l'exécution des services est définie à l'aide d'un réseau de Petri. Le service est sous le contrôle d'un mécanisme d'exécution appelé moteur d'orchestration. Chaque moteur fonctionne suivant un ensemble de règles d'exécution communes et contrôle la sélection et l'exécution des services en interprétant leurs modèles respectifs.

[Benatallah03] et [Fauvet01] proposent un modèle d'exécution qui repose sur les interactions d'égal à égal entre les services participants à la composition. La responsabilité de la coordination de l'exécution d'un service composite est distribuée entre les composants appelés coordinateurs. Un coordinateur est généré pour chacun des services composés ainsi que pour les composants.

[Penserini06] et [Lopes05] utilisent les architectures dirigées par les modèles (MDA – Model Driven Architecture) comme moyen d'exécution des services. L'approche consiste en la transformation de modèles qui est organisée en deux étapes : la spécification de la correspondance et la définition de la transformation. Dans la spécification de la correspondance, les auteurs se sont concentrés sur le problème d'identification des éléments de correspondance. Dans la définition de la transformation, ils ont décrit les étapes opérationnelles d'une transformation. La spécification de la correspondance est vue comme un modèle indépendant de la plate-forme (PIM – Platform Independent Model) et la

transformation comme un modèle spécifique à la plate-forme (PSM – Platform Specific Model). [Penserini06] propose des spécifications de correspondance entre le méta-modèle du service exprimé à l'aide d'un diagramme d'activités d'UML (comme PIM) et le méta-modèle en JADE (comme PSM) qui est directement opérationnel.

L'attribut *outil d'exécution* est défini comme suit :

Outil d'exécution : set of (langage (valeur), notation (valeur))

2.4.3. Architecture d'exécution

L'architecture à mettre en œuvre pour l'exécution des services peut être centralisée ou distribuée.

Une architecture d'exécution centralisée est utilisée principalement pour la coopération entre entreprises où la coordination est effectuée par un moteur d'exécution centralisé.

Une architecture d'exécution distribuée se base sur la communication égal-à-égal entre partenaires. Ce type d'architecture est utilisé pour la coopération entre entreprises égales comme dans [Benatallah03] et [Mecella02b].

L'attribut *architecture d'exécution* est défini comme suit :

Architecture d'exécution : set of (centralisé, distribué)

En synthèse, le cadre de référence montre l'ensemble des apports offerts par les approches à base de services. Ces apports sont décrits dans la perspective de quatre vues à savoir objet, but, méthode et outil dont les détails sont récapitulés à la Figure 7.

Vue Objet

Entité : Enum {technologique, métier, applicatif, interactionnel, intentionnel}

Adaptabilité : Enum {profil, canal, besoin}

Réflexivité : Booléen

Type de service : Enum {boite noire, boite blanche}

Vue But

But : Set of (concevoir, composer, exécuter)

Vue Méthode

Méthode de modélisation : Enum {top-down, bottom-up, mixte}

Méthode de composition : Set of (automatique, semi- automatique, fixe)

Description de la démarche : Enum {formelle, semi-formelle, informelle}

Processus de la démarche : Enum {activité, produit, décision, contexte}

Vue Outil

Outil de composition : Set of (langage (valeur), notation (valeur))

Outil d'exécution : Set of (langage (valeur), notation (valeur))

Architecture d'exécution : Enum {centralisée, distribuée}

Figure 7. Résumé du cadre de référence

3. POSITIONNEMENT DE SEPT APPROCHES AU MOYEN DU CADRE DE REFERENCE

Cette section présente une évaluation de sept approches à base de services :

- Un support méthodologique pour la modélisation des services [Dijkman03] [Quartel04].
- Une approche pour la composition des services et leur suivi d'exécution [Benatallah02] [Benatallah03] [Fauvet02].
- Une approche pour la composition des e-services reposant sur les automates [Berardi03] [Berardi04].
- Une approche méthodologique pour la conception et le développement des systèmes à base de services [Papazoglou06a].
- Une approche pour la construction d'applications coopératives à base de e-services [Mecella02a] [Mecella02b].
- Une approche méthodologique orientée but pour le développement d'application à base de services [Penserini05] [Penserini06a].

- Une approche méthodologique pour la gestion du cycle de vie de la composition de services [Yang03].

Ces sept approches ont été sélectionnées parmi toutes celles citées dans cette section, d'une part pour leur intérêt particulier, d'autre part parce qu'elles nous ont paru utilisables en pratique, et enfin parce qu'elles forment un ensemble représentatif de l'état de l'art. Cette partie décrit ces approches et les situe par rapport au cadre de référence décrit ci-dessus.

3.1. Un support méthodologique pour la modélisation des services

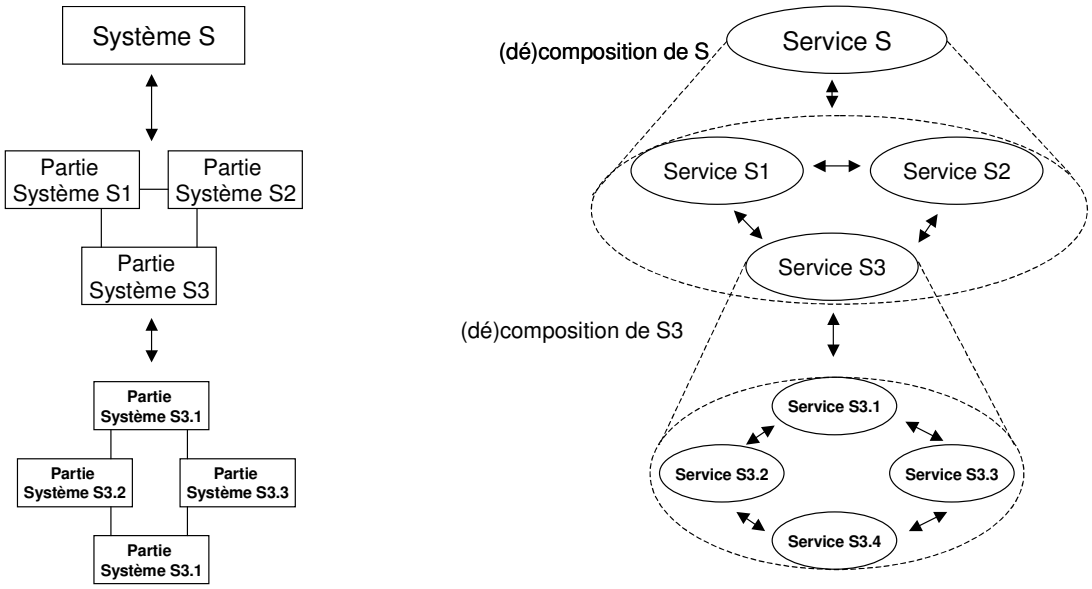
[Dijkman03] se place dans le contexte de la modélisation orientée services (ou encore SoD pour Service oriented Design). Il justifie son travail par le fait qu'il y a non seulement une dimension technologique évidente dans la définition des services mais aussi la nécessité d'introduire des langages, méthodes et techniques de modélisation de services. Ces derniers permettent de répondre aux problèmes de transformation d'un processus métier en services applicatifs.

L'auteur fait la distinction entre SoC (pour Service oriented Computing) et SoD. En effet, SoC est un paradigme qui utilise les services comme brique logicielle de base afin de développer des services plus complexes à forte valeur ajoutée, facilement accessibles par Internet. Ces services sont appelés services composites [Casati01] [Benatalllah02]. Le paradigme SoC est associé aux technologies de description, publication et recherche des services (SOAP, WSDL, BPEL4WS, …).

SoD en revanche, est le paradigme mettant en œuvre l'ensemble des langages, méthodes et techniques de modélisation pour définir les services. Ces derniers peuvent être ensuite développés en utilisant les technologies de SoC.

Djikman introduit une définition de service plus générique qu'usuellement : le service n'est plus défini par référence aux technologies de SoC mais peut être un processus métier dans son ensemble, des services applicatifs ou encore un ensemble de communications au sens d'un composant logiciel.

Il considère qu'un système présente deux perspectives : la perspective interne et la perspective externe. La perspective externe correspond à la perspective des utilisateurs. Le système est alors considéré comme une boîte noire et le comportement observable du système est appelé service. La perspective interne correspond à la perspective des concepteurs. Le système est considéré comme une composition de services qui interagissent ensemble. Ces services peuvent être à leur tour décomposés. La Figure 8 montre les deux perspectives d'un système.



3a. La perspective externe **3b. La perspective interne**

Figure 8. La perspective externe (3a) et interne (3b) d'un système

Quartel [Quartel04a] étend les travaux de Djikman en proposant un processus méthodologique pour la modélisation des services définis par [Djikman03] à partir de la définition du processus métier global.

Le processus comprend quatre étapes méthodologiques à savoir :

1) *La spécification du processus métier*. Cette étape permet de définir les exigences métier que l'application doit supporter. Le processus métier est composé d'un ensemble d'activités.

2) *La spécification des services applicatifs*. Elle suggère de déterminer les fonctionnalités requises par l'application ou encore les services applicatifs. Les activités du processus métier sont ainsi décomposées en parties et supportées par des applications ou par des utilisateurs (environnement de l'application).

3) *La modélisation des services applicatifs*. Elle permet de modéliser chaque service applicatif sous la forme d'une composition de sous services. Le processus de décomposition est réitéré jusqu'à atteindre des services directement implémentables à l'aide les technologies de SoC.

4) *L'implémentation des services applicatifs*. L'objectif est d'implémenter les services applicatifs modélisés dans les étapes précédentes en utilisant les technologies ou les plates-formes spécifiques.

Les trois premières étapes couvrent la modélisation de l'approche allant du processus métier jusqu'aux services applicatifs qu'il compose. [Quartel04a] propose de ce fait un modèle

conceptuel générique exprimé avec le langage *ISDL* (Interaction Systems Design Language) [Quartel04b] afin de modéliser ces étapes.

Le support méthodologique pour la modélisation des services en utilisant ISDL [Dijkman03] [Quartel04] se positionne de la façon suivante par rapport au cadre de référence :

Vue Objet

Entité : applicatif

Type de service : boite blanche

Vue But

But : concevoir, composer

Vue Méthode

Méthode de modélisation : mixte

Méthode de composition : fixe

Description de la démarche : semi-formelle

Processus de la démarche : produit

Vue Outil

Outil de composition : Langage (ISDL)

3.2. Une approche pour la composition des services et leur suivi d'exécution

L'approche Self-Serv (compoSing wEb accessibLe inFormation and buSiness sERVices) propose un modèle de coordination d'égal à égal pour le support de la composition dynamique de services Web inter-entreprise [Benatallah02] [Benatallah03]. Un service composé relie des services élémentaires et permet la réalisation d'un certain nombre d'opérations. Les opérations du service composé sont spécifiées par un diagramme d'états qui représente le flux d'invocation des opérations des services composants ainsi que les données échangées. Self-Serv supporte le choix dynamique des composants en utilisant la notion de conteneur de services. Un conteneur est un service qui représente 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.

Le modèle de coordination de Self-Serv repose sur les interactions d'égal à égal entre les composants participant à la composition. La responsabilité de la coordination de l'exécution d'un service composé est distribuée entre les composants appelés coordinateurs. Un

coordinateur est généré pour chacun des services composés ainsi que pour les composants. Ils sont hébergés par le prestataire du service correspondant. Les coordinateurs sont des planificateurs légers qui reçoivent les notifications de terminaison des autres coordinateurs et invoquent le service correspondant. Une fois le service accompli, le coordinateur transmet les résultats obtenus à l'invocateur du service et envoie une notification de terminaison aux coordinateurs des états suivants. De cette manière, la gestion des flux entre les différents composants d'un service est réalisée par l'échange de messages de notification entre coordinateurs. Les connaissances nécessaires au fonctionnement d'un coordinateur sont extraites statiquement du diagramme d'états spécifiant les opérations du service composé.

Le modèle Self-Serv est mieux adapté au passage à l'échelle que les modèles centralisés grâce au partage du traitement des messages pendant l'exécution entre plusieurs serveurs. Il respecte l'autonomie des services mais les interactions entre les composants (services) sont préalablement définies. Il ne permet pas l'échange de résultats intermédiaires. Il supporte le choix du prestataire de service. Le modèle est destiné à la composition automatique des services et ne semble pas approprié aux interactions humaines. Il n'y a pas de support pour des exécutions transactionnelles.

Self-Serv permet le suivi de l'exécution de services composés [Fauvet02]. Pour ce faire, chaque prestataire de service enregistre des traces liées à ses exécutions. Une trace contient l'historique des états du service, l'ensemble des valeurs effectives de ses paramètres d'entrée et de sortie et la localisation du réalisateur du service. L'historique des exécutions des services représente un répertoire de traces qui propose une interface pour les requêtes. La collection des traces est réalisée par des requêtes distribuées. Lors de la réception d'une requête, si le service n'est pas élémentaire, des sous requêtes sont envoyées aux prestataires de ses composants. Les résultats des sous requêtes sont ensuite fusionnés avec les résultats locaux et renvoyés au demandeur. Les traces d'exécution des services sont donc stockées à un endroit unique (le prestataire) et sont collectées de manière distribuée. Elles peuvent être utilisées pour supporter la prise de décision concernant l'amélioration de la structure ou la dynamique des services. Cependant, la méthode proposée n'est opérationnelle que si les partenaires n'exigent pas la confidentialité de leurs exécutions.

L'approche pour la composition des services et leur suivi d'exécution Self-Serv [Benatallah02] se positionne de la façon suivante par rapport au cadre de référence :

Vue Objet

Entité : métier

Adaptabilité : besoin

Type de service : boite noire

Vue But

But : composer, exécuter

Vue Méthode

Méthode de modélisation : bottom-up

Méthode de composition : automatique

Description de la démarche : semi-formelle

Processus de la démarche : activité

Vue Outil

Outil de composition : Modèle (diagramme d'états)

Outil d'exécution : Modèle (arbre)

Architecture d'exécution : distribuée

3.3. Une approche pour la composition des e-services reposant sur les automates

[Berardi03] propose un cadre de développement et de composition de e-services reposant sur un ensemble d'automates. Ces automates sont étiquetés par un alphabet représentant les actions du e-service.

La modélisation des e-services par des automates consiste à définir le comportement du e-service face à un ensemble d'actions. L'automate est constitué d'un ensemble d'états et d'un ensemble de transitions étiquetées entre ces états qui retranscrivent l'évolution du e-service. Bien que structurellement simple, ce formalisme permet de modéliser la plupart des aspects des e-services avec une précision souvent suffisante, et joue un rôle important dans la définition de leur sémantique.

L'auteur classifie les e-services suivant trois rôles à savoir : servant, initiateur et mixte.

Le e-service *servant* (noté >>*s*) consiste en un ensemble d'actions fournies par le fournisseur. Il est directement exploitable par le client et il est capable de satisfaire sa requête. L'automate correspondant à ce e-service est un automate où les actions sont de type action de sortie.

Le e-service *initiateur* (noté *s*>>) reflète les actions que le client désire effectuer. L'automate correspondant à ce e-service est un automate où les actions sont de type action d'entrée.

Quant au e-service *mixte*, il résulte de la combinaison de e-services de type initiateur et servant. Le modèle correspondant à ce e-service est un automate où les actions sont subdivisées en deux catégories : les actions d'entrée (c'est à dire de l'initiateur) et les actions de sortie (c'est à dire du servant). Cet automate considère différemment les actions de réception et celles d'émission.

La Figure 9 montre un exemple d'automate représentant un e-service de rédaction de carte postale de type mixte. Il consiste à (i) valider les informations de l'utilisateur reçus lors de son authentification (*authentification*), (ii) rédiger un message et vérifier l'orthographe (*rédiger/vérifier*) ou bien proposer un canevas prédéfini pour la rédaction du message : le canevas peut être soit (iii) en anglais (*sélectionner-canevas-anglais*), ou (iv) en italien (*sélectionner_canevas_italien*) ; une fois que l'un de ces canevas a été sélectionné, il est édité (*éditer_canevas*). L'e-service de rédaction de carte postale est servant pour les opérations *authentification* et *rédiger/vérifier* et initiateur pour le reste des opérations. D'où la nature *mixte* du e-service.

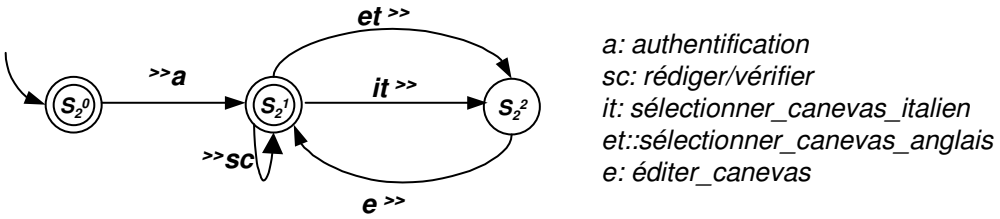


Figure 9. Un e-service de type mixte

L'auteur présente la composition de e-services sous la forme d'un arbre de composition. Les nœuds de l'arbre représentent la séquence d'actions déjà exécutées ainsi que la spécification de leurs initiateurs respectifs. Les arcs montrent l'ensemble des transitions d'un nœud à un autre. Ils sont étiquetés par *(a, I)/S* indiquant que l'action *a* est initialisée par *I* et exécutée par l'ensemble des servants *S*.

L'arbre de composition est généré à partir de la spécification du client transformée en un automate au moyen d'un ensemble de règles de transformation.

[Berardi04] présente une approche d'exécution des e-services en utilisant des formules en PDL_{gm} [Harel00] (une variante de PDL – Propositional Dynamic Logic). L'approche consiste à traduire en premier les automates des e-services à composer ainsi que l'e-service initiateur sous la forme de formules $PDL_{gm.}$ L'auteur propose ensuite un ensemble de théorèmes à appliquer en vue de trouver la coordination de la composition d'e-services qui satisfait la demande utilisateur.

L'approche pour la composition des e-services reposant sur les automates [Berardi03] [Berardi04] se positionne de la façon suivante par rapport au cadre de référence :

Vue Objet

Entité : applicatif

Type de service : boite blanche

Vue But

But : composer, exécuter

Vue Méthode

Méthode de modélisation : bottom-up

Méthode de composition : automatique

Description de la démarche : formelle

Processus de la démarche : produit

Vue Outil

Outil de composition : modèle (automate)

Outil d'exécution : modèle (PDL_{gm})

3.4. Une approche méthodologique pour la conception et le développement des systèmes à base de services

[Papazoglou06a] propose un cycle de vie méthodologique pour la modélisation conceptuelle et le développement des applications à base de services. Le cycle de vie est un ensemble de directives et principes méthodologiques pour spécifier, construire et implémenter des applications supportant une composition de services.

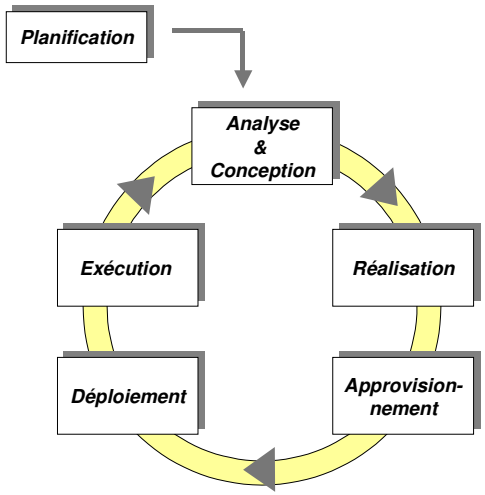


Figure 10. Les phases du cycle de vie méthodologique

Le cycle de vie méthodologique repose sur un processus itératif et incrémental qui comporte six phases :

1- Planification. Cette étape commence par déterminer l'approche de développement à mettre en œuvre (top-down, bottom-up ou mixte). Ensuite, une analyse des écarts est faite en comparant les services planifiés aux implémentations existantes de services. La phase de planification se base sur des scénarios pour concevoir des processus métier à base de services suivant trois manières (i) la réutilisation de services existants, (ii) le développement de nouveaux services ou processus à partir de rien ou (iii) la réutilisation des composants ou d'applications 'legacy' et encapsule sous forme de services.

2- Analyse et conception des services et des processus. L'analyse des services permet d'identifier et de concevoir les processus métier en termes de services. Cette étape est généralement suivie par la modélisation des services qui permettent de définir les interfaces de services à utiliser indépendamment de la plate-forme de développement.

Selon la Figure 11, l'analyse et la conception des services et des processus ont des caractéristiques indépendantes.

Premièrement, l'analyse et la conception des services consistent à identifier les services à utiliser et à les spécifier. La spécification des services comporte quatre activités (i) la description de l'interface en WSDL, (ii) la spécification des paramètres des opérations en entrée et en sortie, (iii) la spécification du protocole de transport et d'échange de données en utilisant le plus souvent le standard SOAP et (iv) la définition des opérations, des liaisons à un protocole concret et l'emplacement du service à invoquer.

Deuxièmement, l'analyse et la conception des processus métier consistent à identifier les processus et à les spécifier. L'identification du processus métier permet d'identifier les services à composer. En outre, la spécification du processus comporte trois activités (i) la définition de l'objectif et la celle de la structure du processus qui spécifie l'ordre d'exécution des services, des partenaires impliqués et les données échangées entre eux, (ii) la description des rôles que jouent les partenaires dans une conversation et (iii) l'identification des besoins non fonctionnels comme la sécurité des processus.

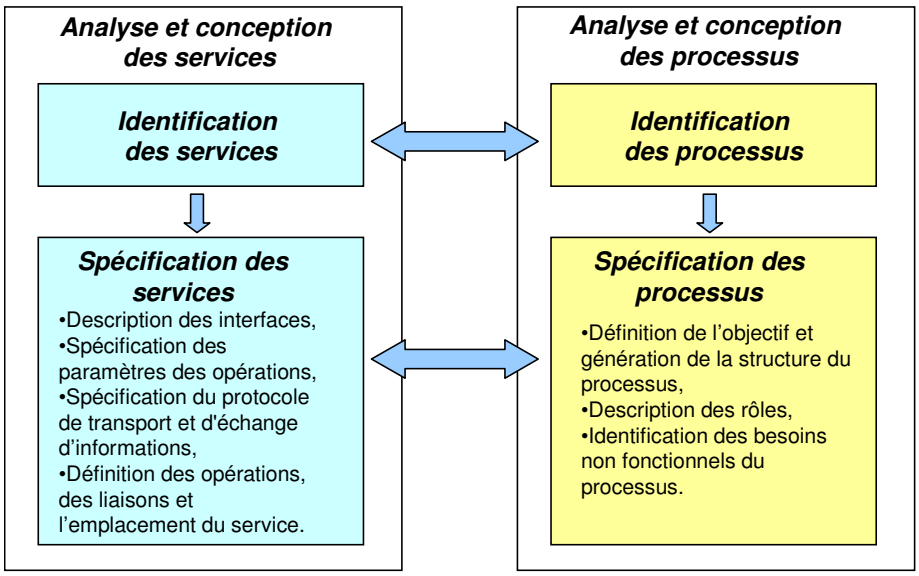


Figure 11. La phase d'analyse et de conception

3- Réalisation. La phase de réalisation comporte trois étapes :

(i) le choix du style de programmation des services. On distingue deux types notamment : rpc (remote procedure call) et message/document.

Dans le style rpc, les services s'apparentent aux modèles d'objets distribués traditionnels. La communication entre le client et le service est centrée sur l'interface. Par contre, dans le style message/document, la communication est considérée comme une standardisation de l'échange de documents. Le client et les services ne sont couplés que par l'utilisation des messages XML utilisant un format standard et par le protocole de transport utilisé.

(ii) le codage des services par la conception d'une interface avec un fichier WSDL. La publication de l'interface du service est réalisée ensuite au travers d'un annuaire spécifique. L'étape suivante consiste à générer le squelette du service à partir de l'interface dans le langage d'implémentation. Enfin, les méthodes du service prédéfinies dans le squelette sont implémentées.

(iii) le codage du processus métier supportant les services par la conception de sa logique métier à travers des spécifications exprimées par exemple en BPEL4WS (Business Process Execution Langage for Web Service).

4- Approvisionnement. Le choix du service à adopter varie d'un utilisateur à un autre. Cependant, plusieurs alternatives influencent le choix tels que le paiement de l'utilisation du service ou son utilisation gratuite.

5- Déploiement. L'objectif de cette étape consiste à (i) compiler et déployer le service, (ii) générer l'implémentation du service (liaisons à un protocole concret et au point d'entrée du service) et (iii) publier l'implémentation dans un annuaire de services.

6- Exécution. Dans cette phase, les services supportant le processus métier sont déployés et opérationnels. Le demandeur de service peut invoquer les opérations publiques du service en se référant à la définition de l'interface de ce dernier.

L'approche méthodologique pour la conception et le développement des systèmes à base de services [Papazoglou06a] se positionne de la façon suivante par rapport au cadre de référence :

Vue Objet

Entité : technologique

Type de service : boite noire

Vue But

But : concevoir, composer, exécuter

Vue Méthode

Méthode de modélisation : top-down

Description de la démarche : informelle

Processus de la démarche : activité

Vue Outil

3.5. Une approche pour la construction d'applications coopératives à base de e-services

L'approche PARIDE (Process-based frAmework for oRchestratIon of dynamic E-services) [Mecella02a] propose une méthodologie pour la définition et la modélisation des processus coopératifs à base de e-services. Chaque e-service est représenté à l'aide d'un modèle conceptuel constitué de deux parties : l'interface et l'ensemble des interactions dans lesquelles l'e-service est impliqué. L'interface du e-service est spécifiée par un diagramme de classes qui représente les données utilisées en entrée et en sortie. Les interactions sont spécifiées par un diagramme d'états qui représente le flux d'invocation des opérations ainsi que les données échangées.

Le modèle d'orchestration dynamique des e-services de PARIDE définit l'enchaînement des e-services selon un canevas prédéfini et leur exécution à travers un schéma d'orchestration. Ces schémas décrivent souvent l'interaction entre e-services en identifiant les messages échangés et les séquences d'invocation de e-services.

Dans l'approche proposée par [Mecella02b], l'orchestration des e-services est définie à l'aide d'un réseau de Petri. Chaque e-service est sous le contrôle d'un mécanisme d'exécution appelé par moteur d'orchestration. Chaque moteur fonctionne suivant un ensemble de règles

d'exécution et contrôle la sélection et l'exécution des e-services en interprétant leurs modèles respectifs.

Chaque moteur de e-service fonctionne suivant son propre modèle. Toutefois, pour assurer l'orchestration globale suivant un mode distribué, il est nécessaire de passer des informations d'un moteur à un autre. Ces informations reposent sur des liens entre les e-services du réseau de Petri. Ces liens permettent de garantir l'orchestration des différents e-services composant le processus coopératif.

L'approche pour la construction d'applications coopératives à base de e-services [Mecella02a] [Mecella02b] se positionne de la façon suivante par rapport au cadre de référence :

Vue Objet

Entité : applicatif

Type de service : boite blanche

Vue But

But : concevoir, composer, exécuter

Vue Méthode

Méthode de modélisation : bottom-up

Méthode de composition : fixe

Description de la démarche : semi-formelle

Processus de la démarche : activité

Vue Outil

Outil de composition : modèle (diagramme d'états)

Outil d'exécution : modèle (réseau de Petri)

Architecture d'exécution : distribuée

3.6. Une approche méthodologique orientée but pour le développement d'application à base de services

[Penserini06a] propose d'utiliser la méthodologie de développement orientée agent TROPOS [Mylopoulos00] afin de couvrir entièrement le cycle de développement des applications à base de services : de l'ingénierie des exigences à l'identification des services qui répondent à ces exigences.

L'approche est structurée en quatre étapes à savoir :

- Identification et expression des besoins des utilisateurs.

- Identification des capacités.
- Identification des préférences des utilisateurs.
- Evaluation des capacités par rapport aux préférences des utilisateurs.

Nous détaillons ci-dessous chacune de ces étapes.

1- Identification et expression des besoins des utilisateurs. Les auteurs proposent d'exprimer les besoins par des buts. L'analyse des besoins correspond alors à l'identification des buts et l'exploration d'alternatives. Le résultat de cette analyse est un graphe ET/OU de buts. Chaque but du graphe peut se décomposer en sous buts selon la manière dont il doit être satisfait.

Des sous buts reliés par une relation ET (« AND ») indiquent qu'ils doivent tous être réalisés afin que le but père soit atteint. Des sous buts reliés par une relation OU (« OR ») impliquent que la satisfaction du but père se réalise suite à la satisfaction d'un ou plusieurs de ses sous buts.

La décomposition d'un but en sous buts est poursuivie jusqu'à atteindre des sous buts qui peuvent être satisfaits par une fonctionnalité du système. Ces sous buts constituent les feuilles du graphe et sont appelés des *tâches*.

Un sous arbre constitué d'un nœud racine et plusieurs nœuds satisfaisant le nœud racine constitue *une alternative*.

2- Identification des capacités. Le but de cette étape est d'identifier les services qui permettent de réaliser les besoins des utilisateurs. [Penserini06a] propose de modéliser les services sous la forme de *capacités*.

Une capacité est un couple <compétences, opportunités>. Une compétence est une condition nécessaire à la réalisation d'un but. Elle est offerte par un plan. Une opportunité est une condition suffisante pour la réalisation d'un but. Elle caractérise le contexte et est formulée comme une pré-condition.

Afin d'exprimer la capacité dans les termes de TROPOS : (1) la compétence est décrite à l'aide de la relation « means-end » entre un but et un plan (dans TROPOS un plan correspond à une tâche) et (2) l'opportunité est décrite à l'aide des concepts plan/softgoal, les influences associées (une influence $\in$ {+, -, ++, --}) ainsi que les contraintes correspondantes (contraintes temporelles).

Durant cette étape, chaque plan est évalué par rapport à l'ensemble des préférences qui sont exprimées par des besoins non fonctionnels (« softgoals »). Certaines préférences peuvent favoriser des plans particuliers ou les défavoriser. Ceci s'exprime à l'aide de métriques.

[Penserini06] développe le concept de *capacité* comme suit :

$$Cap = \langle means\_end(but, plan), \cup_i contribution\ (plan, softgoal, métrique), \{contraintes\_du\_domaine\}\rangle$$

Les capacités offertes sont par la suite, associées aux alternatives identifiées durant la tâche précédente (*Identification et expression des besoins des utilisateurs*). Une alternative peut être une composition de capacités.

Les capacités sont donc un moyen de modéliser les services afin de satisfaire les buts des utilisateurs.

[Penserini06b] enrichit le concept de capacité en introduisant une dimension dynamique afin de réaliser une capacité. Chaque capacité est alors exprimée à l'aide (i) d'un diagramme d'activités qui représente le séquencement d'activités à mettre en oeuvre et (ii) d'un diagramme d'interactions pour chaque activité qui s'appuie sur une représentation graphique de la conversation entre les agents. Ces schémas caractérisent chaque interaction avec les différents agents participants, la nature des messages échangés et enfin le déroulement temporel de l'interaction.

3- Identification des préférences des utilisateurs. Les préférences sont exprimées par des besoins fonctionnels et des besoins non fonctionnels (*« softgoals »).* C'est un moyen qui permet d'identifier les besoins des utilisateurs en terme de buts à atteindre ainsi que l'ensemble des besoins non fonctionnels associés.

Formellement, les préférences d'un utilisateur sont exprimées à l'aide du service isn (initial service need) [Penserini05] :

$$isn : \langle (goal_1, soft\text{-}goal_{1.1}, soft\text{-}goal_{1.2}\ldots),\ldots, (goal_m, soft\text{-}goal_{m.1}, soft\text{-}goal_{m.2}\ldots)\rangle$$

4- Evaluation des capacités par rapport aux préférences des utilisateurs. Les capacités sont étudiées par rapport aux préférences des utilisateurs. L'objectif de cette évaluation est de déterminer les couples <capacités, préférences> qui ont le plus de similitudes. Les préférences d'un utilisateur peuvent être réalisées à l'aide d'une composition de capacités c'est à dire de services.

[Penserini06b] propose d'utiliser l'architecture dirigée par les modèles (MDA – Model Driven Architecture) pour développer les capacités identifiées à l'étape 2 (Identification des capacités). Tout d'abord les capacités sont décrites sans tenir compte des caractéristiques technologiques. L'auteur propose d'utiliser le méta-modèle des diagrammes d'interactions et d'activités d'AUML représentant une capacité. Puis la description des capacités est décrite avec une technologie spécifique : le meta-modèle de JADE (Java Agent Development Framework). Des règles de transformation sont ensuite spécifiées afin de permettre le passage du méta-modèle d'AUML au méta-modèle de JADE.

L'approche méthodologique orientée but pour le développement de services web [Penserini05] [Penserini06a] se positionne de la façon suivante par rapport au cadre de référence :

Vue Objet

Entité : métier

Adaptabilité : vrai

Réflexivité : Booléen

Type de service : boite blanche

Vue But

But : concevoir, composer, exécuter

Vue Méthode

Méthode de modélisation : top-down

Méthode de composition : semi-automatique

Description de la démarche : semi-formelle

Processus de la démarche : activité

Vue Outil

Outil de composition : modèle (TROPOS)

Outil d'exécution : modèle (JADE)

3.7. Une approche méthodologique pour la gestion du cycle de vie de la composition de services

[Yang03] propose un cadre de développement des compositions de services. L'auteur considère que l'activité de conception et de composition des services web demande des efforts considérables de développement et de ré-ingénierie des fonctionnalités existantes.

Le cadre de développement introduit un cycle de vie en cinq phases pour le développement d'applications et la composition de services basés sur le nouveau concept de « composant service ».

Les phases du cycle de vie d'une composition sont : la planification, la définition, la classification, la construction et enfin l'exécution.

La phase de planification aide à déterminer les services à choisir pour satisfaire une demande utilisateur. Elle explique la manière de vérifier la compatibilité des services candidats à composer avec les besoins des utilisateurs en terme de services. Le langage SCPL (Service Composition Planning Language) est utilisé pour réaliser cette phase.

La phase de définition permet de définir les services composites d'une manière abstraite. Pour cela, l'auteur définit un langage reposant sur XML : SCSL (Service Composition Specification Language) pour la définition des services composites ou encore des « composant services ».

La phase de classification est responsable pour déterminer la manière dont les services sont exécutés. Son objectif est de fournir une définition concrète des services définis durant la phase précédente. Cette phase permet de générer plusieurs alternatives de composition. Le langage SSL (Service Scheduling Language) est utilisé comme moyen de spécifier la manière de construire un « composant service » en terme de composition de services en précisant leurs liens et dépendances.

La phase de construction et la phase d'exécution implémentent la composition de services obtenue. Pour cela, la structure d'exécution SCEG (Service Composition Execution Graph) est utilisée. SCEG est un arbre où chaque nœud est un service composite et les fils respectifs en sont les constituants. La racine du graphe forme l'application qu'on veut développer. Le type de composition ainsi que la dépendance entre les messages sont indiqués au niveau de chaque nœud.

Le concept de « composant service » est un mécanisme qui permet de combiner les web services existants afin de développer de nouvelles applications. De nature réflexive, un « composant service » peut être réutilisé dans d'autres compositions.

Comme le montre la Figure 12, un « composant service » est composé de deux parties : l'interface et la logique de composition. L'interface est la partie publique qui permet d'invoquer et de réutiliser le « composant service ». La logique de composition comporte le type de composition c'est à dire la forme que la composition peut prendre et la nature de dépendance entre les messages de services. Elle expose la manière de combiner et de synchroniser des services sous la forme d'une séquence d'interactions.

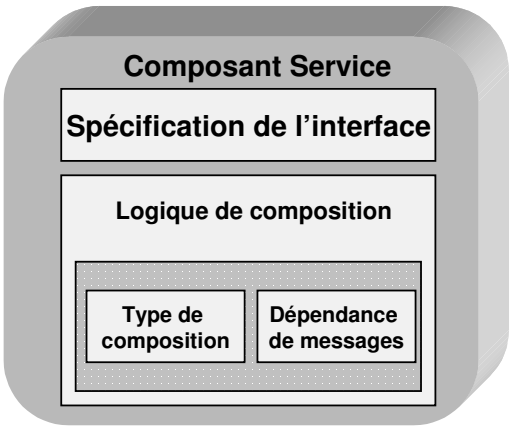


Figure 12. La structure d'un composant service

L'approche méthodologique pour la gestion du cycle de vie de la composition de services [Yang03] se positionne de la façon suivante par rapport au cadre de référence :

Vue Objet

Entité : applicatif

Réflexivité : Vrai

Type de service : boite noire

Vue But

But : composer, exécuter

Vue Méthode

Méthode de modélisation : bottom-up

Méthode de composition : fixe

Description de la démarche : semi-formelle

Processus de la démarche : activité

Vue Outil

Outil de composition : langage (SCSL)

Outil d'exécution : langage (SCEG)

4. RECAPITULATIF DE L'EVALUATION

Le cadre de référence que nous avons proposé permet d'étudier différents aspects des services. Il est constitué de quatre vues qui correspondent à quatre perspectives selon lesquelles on peut étudier la notion de service : le quoi, le pourquoi, le comment et le moyen. Dans chacune de ces vues sont définis des attributs qui précisent les caractéristiques d'un service donné. Nous avons illustré le cadre en étudiant et en positionnant sept approches représentatives de l'état de l'art. Le Tableau 2 synthétise les résultats de cette analyse.

Tableau 2. Récapitulatif de l'évaluation des sept approches

	[Quartel04]	[Benatallah02]	[Berardi03]	[Papazoglou06a]	[Mecella02a]	[Penserini05]	[Yang03]
Nature de l'entité	Applicatif	Métier	Applicatif	Technologique	Applicatif	Métier	Applicatif
Adaptabilité	--	Besoin	--	--	--	Besoin	--
Réflexivité	--	--	--	--	--	--	Vrai
Type	Boite blanche	Boite noire	Boite noire	Boite noire	Boite blanche	Boite blanche	Boite noire
But	Concevoir,	Composer,	Composer,	Concevoir, composer,	Concevoir, composer,	Concevoir, composer,	Composer

	composer	exécuter	exécuter	exécuter	exécuter	exécuter	, exécuter
Méthode de conception	Mixte	Bottom-up	Bottom-up	Top-down	Bottom-up	Top-down	Bottom-up
Méthode de composition	Fixe	Automatique	Automatique	--	Fixe	Semi-automatique	Fixe
Description de la démarche	Semi-formelle	Semi-formelle	Formelle	Informelle	Semi-formelle	Semi-formelle	Semi-formelle
Processus de la démarche	Produit	Activité	Produit	Activité	Activité	Activité	Activité
Outils de composition	Langage (ISDL)	Modèle (diagramme d'états)	Modèle (automate)	--	Modèle (diagramme d'états)	Modèle (TROPOS)	Langage (SCSL)
Outils d'exécution	--	Modèle (arbre)	Modèle (PDL_{gm})	--	Modèle (réseau de Petri)	Modèle (JADE)	Modèle (SCEG)
Architecture d'exécution	--	Distribuée	--	--	Distribuée	--	--

Comme le montre le Tableau 2, les approches d'ingénierie à base de services mettent en jeu des services de nature très variée : [Quartel04], [Berardi03], [Mecella02a] et [Yang03] s'intéressent aux services applicatifs, [Benatallah02] et [Penserini05] aux services métier, et [Papazoglou06a] aux services technologiques.

Deux approches s'intéressent à l'aspect de l'adaptabilité de services à savoir [Benatallah02] et [Penserini05].

Pour l'aspect de la réflexivité, seul [Yang03] propose le concept de « composant service » qui est 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.

Les services manipulés sont, dans la majorité des approches étudiées, de type boîte noire.

Le but des approches d'ingénierie à base de services varie sans prédominance particulière de la conception [Quartel04], [Papazoglou06a], [Penserini05], de la composition [Quartel04], [Benatallah02], [Berardi03], [Mecella02a], [Penserini05], [Yang03] ou de l'exécution [Benatallah02], [Berardi03], [Papazoglou06a], [Mecella02a], [Yang03].

Les approches, que nous avons détaillées, proposent une méthode pour construire des applications à base de services. Ces méthodes sont « top-down » et utilisent des règles de dérivations pour passer de niveaux d'abstractions élevés aux niveaux plus bas [Papazoglou06a] [Penserini05]. Les approches peuvent être « bottom-up » : on part alors des services élémentaires pour arriver à développer l'application composite [Benatallah02] [Beradi03] [Mecella02a] [Yang03]. [Quartel04] propose une approche « mixte » : l'approche

« top-down » est appliquée à l'aide des étapes qui couvrent la modélisation allant du processus métier jusqu'à l'identification des services applicatifs ; l'approche « bottom-up » est appliquée en introduisant des règles qui permettent de vérifier la conformité des services obtenus par rapport au processus de départ.

Concernant les méthodes de composition, elles sont dans la plus part des cas de type « Fixe » [Quartel04], [Mecella02a], [Yang03]. Ceci est du à la complexité des mécanismes à mettre en œuvre pour une composition automatique. Seuls [Benatallah02] et [Berardi03] proposent une approche de composition automatique. [Penserini05] introduit une approche de composition semi-automatique.

La description de la démarche est « semi-formelle » dans la majorité des approches [Quartel04], [Benatallah02], [Mecella02a], [Penserini05], [Yang03]. [Berardi03] propose une notation formelle avec une sémantique bien définie. [Papazoglou06a] suggère une démarche informelle structurée en étapes, directives et principes méthodologiques pour spécifier, construire et implémenter des applications supportant une composition de services entièrement décrite en langage naturel.

Les outils de composition varient d'une approche à une autre. Certains auteurs comme [Benatallah02], [Berardi03], [Mecella02a], [Penserini05], proposent des modèles. D'autres comme [Quartel04], [Yang03] proposent des langages dédiés.

Concernant les outils d'exécution, la plus part des approches proposent des modèles spécifiques comprenant les structures d'arbre [Benatallah02], [Yang03], les réseaux de Petri [Mecella02a], les langages formels [Berardi03] ou encore des plateformes technologiques [Penserini05].

Deux approches s'appuient sur une architecture d'exécution qui est de nature distribuée. [Banatallah02] propose une architecture qui repose sur les interactions d'égal à égal entre les composants participant à la composition. La responsabilité de la coordination de l'exécution d'un service composé est distribuée entre les composants appelés coordinateurs. [Mecella02a] propose une architecture qui repose sur un mécanisme d'exécution appelé moteur d'orchestration. Chaque moteur fonctionne suivant un ensemble de règles d'exécution et contrôle la sélection et l'exécution des e-services en interprétant leurs modèles respectifs.

La vision iSOA, que nous proposons dans cette thèse, se positionne de la façon suivante par rapport au cadre de référence :

Vue Objet

Entité : intentionnel

Adaptabilité : besoin

Réflexivité : Vrai

Type de service : Enum {boite noire, boite blanche}

Vue But

But : concevoir, composer, exécuter

Vue Méthode

Méthode de modélisation : top-down

Méthode de composition : automatique

Description de la démarche : semi-formelle

Processus de la démarche : contexte

Vue Outil

Outil de composition : notation (Carte)

Outil d'exécution : notation (Carte)

Architecture d'exécution : distribuée

Les valeurs de chacun des attributs sont détaillées au fur et à mesure dans les différents chapitres de cette thèse.

CHAPITRE 3

Modèle Intentionnel de Services *MiS*

1. INTRODUCTION

Ce chapitre est consacré à la définition d'un modèle de représentation des services dénommé *MiS* (**M**odèle **I**ntentionnel de **S**ervices). Il s'agit, en fait, d'un modèle qui permet de représenter, à un niveau intentionnel, les services du système à développer et d'aider un client à exprimer son intention et se concentrer sur les buts qu'un service permet de réaliser.

Ce chapitre est composé de quatre sections. Nous présentons les apports du modèle *MiS* à la section 2. Dans la section 3, nous développons les concepts du méta-modèle *MiS*. Ensuite, dans la section 4, nous proposons une structure de description textuelle des services intentionnels basée sur le standard XML (e**X**tensible **M**arkup **L**anguage). Finalement, nous concluons le chapitre par la section 5.

2. MOTIVATIONS POUR $\mathcal{MIS}$

La proposition du modèle $\mathcal{MiS}$ est motivée par les trois principaux aspects suivants :

1. *Le besoin d'une meilleure homogénéité dans l'expression des besoins des clients, exprimés au niveau intentionnel, et les services logiciels, exprimés au niveau opérationnel*,
2. *Le besoin de variabilité dans le monde des services,*
3. *L'importance de la réutilisation des services.*

Premièrement, le besoin d'une meilleure homogénéité dans l'expression intentionnelle des besoins des clients et celles des services, conduit à repenser la notion de service technologique et sa « raison d'être » pour s'adapter à l'échelle stratégique et intentionnelle des clients.

Le modèle $\mathcal{MiS}$ 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 du modèle $\mathcal{MiS}$. Le modèle $\mathcal{MiS}$ définit chaque service intentionnel en tant que brique de construction d'applications en lui associant les connaissances situationnelle et intentionnelle sous la forme d'une interface. Chaque service intentionnel s'adapte à une situation particulière afin de réaliser une intention particulière.

Dans le cadre de l'architecture iSOA, 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.

Deuxièmement, 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 préfèrent que les logiciels se configurent en fonction de leurs besoins.

Le modèle $\mathcal{MiS}$ introduit la variabilité dans la manière de réaliser le but d'un service. Les variantes correspondent aux différentes manières d'atteindre le but. Etant donné qu'un service intentionnel peut être composé d'autres services intentionnels, ayant chacun son propre but 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.

Enfin, 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 modèle $\mathcal{MiS}$ s'adapte à la perspective intentionnelle du modèle et débouche sur une composition dirigée par les buts. 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 buts introduit une composition à plusieurs niveaux : le but du service de plus haut niveau, qui peut être de nature stratégique, est décomposé en sous buts/services qui eux-mêmes, peuvent nécessiter une nouvelle dé/composition jusqu'à atteindre des buts/services opérationnalisables. Il y a donc une composition récursive des services.

3. PRESENTATION DES CONCEPTS DU MODELE $\mathcal{MiS}$

Cette section est dédiée à la présentation des concepts sur lesquels repose le modèle $\mathcal{MiS}$.

3.1. Survol du méta-modèle

La Figure 13 présente le méta-modèle $\mathcal{MiS}$ en utilisant les notations du diagramme de classes UML. On décrit d'abord les concepts d'une manière générale puis les détaille dans les sections suivantes.

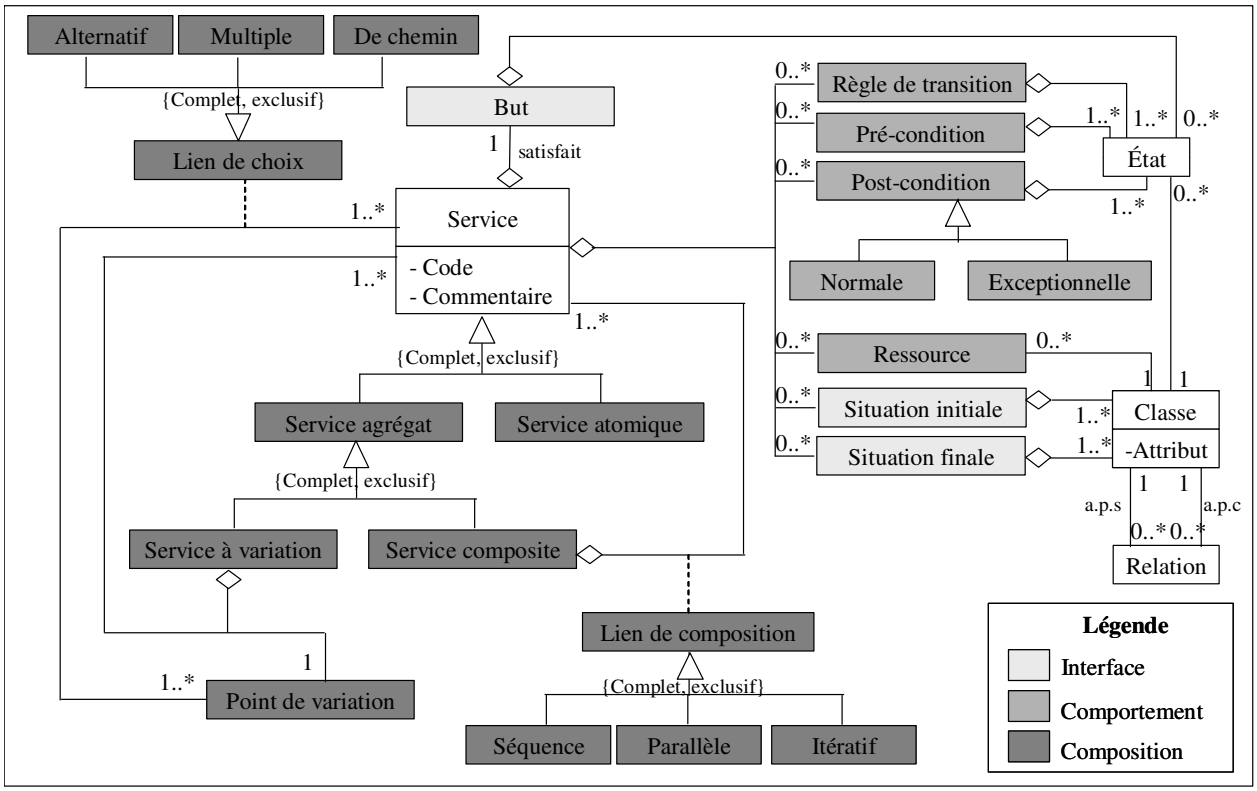


Figure 13. Méta modèle $\mathcal{MiS}$

La Figure 2 met en évidence que le concept central du modèle est celui de *service intentionnel* que désignerons dans la suite de ce mémoire et pour des raisons de simplicité par l'abréviation *service*.

On observe aussi que la perspective intentionnelle choisie dans cette thèse se traduit par le fait qu'un service exhibe une intentionnalité formulée par le *but* qu'il permet à ses clients d'atteindre.

Un but reflète une intention, un objectif que l'on cherche à atteindre sans pour autant dire comment l'atteindre. Un but est associé à un résultat souhaité matérialisé par un ensemble d'états d'objets. Par exemple le but *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'.

On verra par la suite tout le parti que l'on peut tirer d'une expression intentionnelle qui se place au niveau d'expression qui caractérise les acteurs du management et qui évite que l'on ait à entrer dans les détails techniques de la réalisation du service.

Mais on comprend que la recherche d'un service dans cette perspective intentionnelle puisse elle-même être dirigée par les buts. 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).

L'intentionnalité sous entend que le client qui agit d'une manière intentionnelle fasse tout ce qui est en son possible pour atteindre le résultat préfiguré par le but. 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 le but visé. On verra dans les sections suivantes que ce sont les services atomiques qui sont porteurs des actions exécutables nécessaires à la réalisation du but et que ce sont les services agrégats qui orchestrent la composition des actions de base.

La Figure 2 montre, par le jeu des trois couleurs utilisées que la description d'un service intentionnel comporte trois parties correspondant à son *interface*, l'expression de son *comportement* et celle de sa *composition.*

L'*interface* d'un service est la partie coloriée en gris clair sur la figure 2. L'interface, comme dans toute description de service, est la partie visible de l'extérieur, celle qui permet de comprendre ce que sait faire le service sans entrer dans le corps du service. Dans le cas de $\mathcal{MiS}$, la partie clé de l'interface est le *but* 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 du but 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 buts. Pour retrouver le ou les services qui coïncident avec les exigences d'un client exprimées par un ou plusieurs buts, le processus de correspondance opérera non seulement sur l'expression du but lui-même mais aussi sur la situation initiale et la situation finale. La notion d'interface est développée à la section 2.2.2 de ce chapitre.

La deuxième partie descriptive du service est celle relative à son *comportement*. Cette partie est coloriée en gris foncé au niveau de la Figure 13. Dans l'optique intentionnelle de $\mathcal{MiS}$, un service se comporte de façon à satisfaire le but qui lui est associé. Sachant qu'un but est caractérisé par un ensemble d'états d'objets qui traduisent le résultat obtenu par la satisfaction du but, il est usuel de définir le processus de réalisation d'un but par deux ensemble d'états : celui de la situation initiale (au départ du processus de réalisation d'un but) et celui de la situation finale (lorsque le processus de recherche d'atteinte du but est achevé). Nous proposons donc dans $\mathcal{MiS}$, 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 du but du service. La section 2.2.3 de ce chapitre décrit plus en détail la notion de comportement.

La troisième partie descriptive du service est celle de sa *composition*. Comme le montre la Figure 13, 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 un but que l'on qualifie 'd'opérationnalisable', c'est-à-dire pour lequel on est capable de définir une séquence d'actions qui permet de réaliser le but. Dans notre cas, ce sont les actions qui composent le service atomique qui permettent d'assurer l'opérationnalisation du but 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 le but qui lui associé est opérationnalisable dans le sens où l'on sait définir un processus de paiement d'une réservation par carte de crédit.

Les buts de haut niveau tels que les buts stratégiques ne sont pas d'emblée 'opérationnalisables'. Ils nécessitent d'être décomposés en sous buts qui eux-mêmes peuvent nécessiter une décomposition etc. jusqu'à atteindre des buts opérationnalisables. Les services agrégats sont associés à des buts de haut niveau et la composition d'un service agrégat en services suit la décomposition du but père en sous buts. 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 le but *Optimiser le remplissage des hôtels* n'est pas un but opérationnalisable ; il requiert la satisfaction d'autres buts tels que *Gérer une liste d'attente des demandes non satisfaites*, *Suggérer des modifications de demandes, Garantir la mise à jour instantanée des ressources* etc. Chacun de ces buts est associé 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 $\mathcal{MiS}$ est elle aussi dirigée par les buts.

Les approches d'affinement de buts [Dardenne91], [Dardenne93] [Yu94] [Rolland98b] montrent que l'affinement s'apparente à un arbre ET/OU de l'intelligence artificielle [Nilsson71]. Nous pensons que l'affinement OU est utile dans le contexte de $\mathcal{MiS}$ car il permet d'introduire plusieurs décompositions alternatives d'un but. Le méta-modèle comporte la notion de *service à variation* pour satisfaire ce besoin. Un service à variation introduit de la flexibilité dans la manière de satisfaire un but et donc de la flexibilité dans la composition de services et permet d'adapter la manière de rendre un service aux circonstances particulières de sa demande.

La notion de composition est développée à la section 2.2.4 de ce chapitre.

Nous détaillons les concepts de $\mathcal{MiS}$ en respectant leur regroupement selon les trois aspects, *interface*, *comportement* et *composition* dans la section suivante.

3.2. Présentation détaillée des concepts

Dans cette section, nous détaillons chacun des concepts introduits dans le méta-modèle $\mathcal{MiS}$ à la section précédente. Nous choisissons des exemples tirés d'un cas de réservation de chambres d'hôtels pour illustrer les concepts.

3.2.1. Attributs spécifiques du service

Comme le montre la Figure 2 un service intentionnel a deux caractéristiques intrinsèques : son *code* et un *commentaire* explicatif de son contenu.

Pour mettre en évidence la dimension intentionnelle d'un service nous proposons de désigner le service en mettant en exergue le nom du but qu'il permet d'atteindre. Un service réalisant un but *a* est désigné par le code S_a. Par exemple le but $S_{Effectuer\ une\ réservation}$ référence un service qui permet de satisfaire le but *Effectuer une réservation*.

Pour compléter la perception d'un service acquise par le nom du but qu'il permet d'atteindre, on complète sa description par un *commentaire* qui met l'accent sur le résultat qu'il permet d'atteindre et donne quelques indications sur le processus qui sera mis en œuvre pour y parvenir. Ceci permet de comprendre, au moins intuitivement, les conditions d'usage du service. Pour l'exemple ci-dessus, le commentaire serait le suivant :

' Ce service cherche à satisfaire au mieux une demande de réservation de chambres d'hôtel en tenant compte des disponibilités des hôtels du pool offrant ce service. Il génère une réservation lorsque les disponibilités le permettent et met la demande en attente dans le cas contraire. Il informe le client des décisions prises et prend en compte son accord ou son désaccord. Le résultat peut donc être une nouvelle réservation 'créée' ou une demande 'en attente'.

3.2.2. Interface

L'*interface* d'un service $\mathcal{MiS}$ a la même sémantique que celle de l'interface d'un service logiciel classique. Elle permet de caractériser le service d'un point de vue externe, celui du client du service. L'interface a pour vocation de permettre qu'un service soit une brique réutilisable dans la définition de nouvelles compositions de services. Elle sert à extraire d'une base de services publiés par un fournisseur, le ou les services correspondant aux besoins des clients.

Dans le cas de $\mathcal{MiS}$, l'interface a la particularité d'associer à chaque service intentionnel (atomique ou agrégat) la connaissance intentionnelle (i.e. le but) et situationnelle (les situations) permettant la réalisation du but. L'interface du service comporte trois attributs descriptifs, le but qui donne son identité au service, la situation initiale qui définit la structure des paramètres d'entrée et la situation finale qui fournit les paramètres de sortie.

La Figure 14 est un extrait du méta-modèle de la Figure 13 qui montre les différents éléments composant l'interface d'un service.

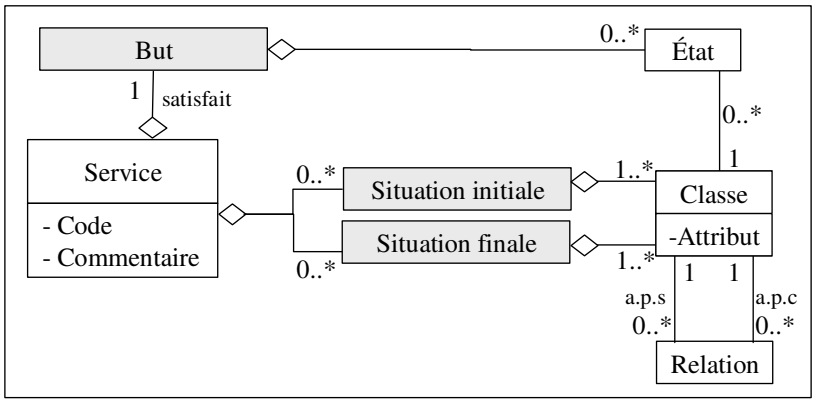


Figure 14. L'interface d'un service

On décrit ces trois éléments successivement.

(i) But

Le concept de but a montré son utilité dans de nombreux domaines : En BPR (Business Process Reengineering) [Yu94] [Anton94] et en Ingénierie des besoins [Rolland99] [Rolland98b] [Yu97] [Dardenne93] [Anton96a] [van Lamsweerds95]. Cette utilité tient à ses caractéristiques :

- Un *but* exprime une *intention*, un objectif que l'on souhaite atteindre. Selon [Jackson95] un but est une déclaration 'optative' qui exprime ce que l'on veut, un état ou un résultat que l'on cherche à atteindre. Elle s'oppose à une expression 'descriptive' qui énonce un fait. Cette caractéristique du but permet d'éviter de rentrer dans les détails du processus d'atteinte du but tout en permettant d'en comprendre les tenants et les aboutissants. Par exemple, l'expression du but *Payer une réservation par carte de crédit* est suffisante pour comprendre la finalité du processus de paiement d'une réservation donné sans avoir à entrer dans les différentes étapes du paiement et les différentes actions à entreprendre pour aboutir au résultat escompté : l'état *réservation.statut = 'payée'*.
- L'expression du but rend *explicite* ce que l'on souhaite réaliser et évite les erreurs d'interprétation et de communication que l'on observe lorsqu'on utilise des énoncés procéduraux et fonctionnels qui laissent *implicite* ce que l'on veut faire. En ingénierie des besoins l'analyse des difficultés rencontrées et observées dans la pratique des approches par scénarios montre qu'elles sont imputables à leur caractère procédural et descriptif qui laisse la finalité non décrite alors que l'enjeu est celui de la finalité.
- Les buts s'expriment à différents *niveaux* : *stratégique, tactique, opérationnalisable* [Ralyté01] et permettent ainsi de capturer dans des expressions de la même nature

des intentions de différents types d'acteurs ayant des enjeux différents mais pourtant énoncés de manière homogène. Les buts stratégiques adressent des intentions du monde stratégique de l'entreprise; ils ont une portée globale alors que les buts tactiques sont plus 'locaux' et spécifiques. Les buts opérationnalisables sont ceux qui peuvent être associés sans difficulté à une séquence d'actions qui permet leur réalisation. *Augmenter le nombre de contrats de 10%* ou bien *Mettre en place un service de distribution d'argent ubiquitaire* sont des buts stratégiques. *Installer des DAB* ou *Effectuer une réservation* sont des buts tactiques. *Contrôler la vitesse du train* ou *Authentifier le client* sont des buts opérationnalisables. Ce sont ceux que dans le BPM on a l'habitude d'associer à chaque processus d'entreprise.

- L'adaptation des graphes de décomposition de buts de l'Intelligence Artificielle [Farreny87] constitue un mécanisme puissant de raisonnement sur les buts. La décomposition ET d'un but père *B1* par des buts fils *B1.1*, *B1.2*, *B1.3* signifie que si *B1.1*, *B1.2* et *B1.3* sont atteints, alors *B1* l'est. La décomposition OU d'un but père *B2* en des buts fils *B2.1* et *B2.2* signifie que si *B2.1* ou *B2.2* est atteint alors *B2* est lui-même atteint. La représentation graphique d'une décomposition ET et OU de buts sont illustrées à la Figure 15.

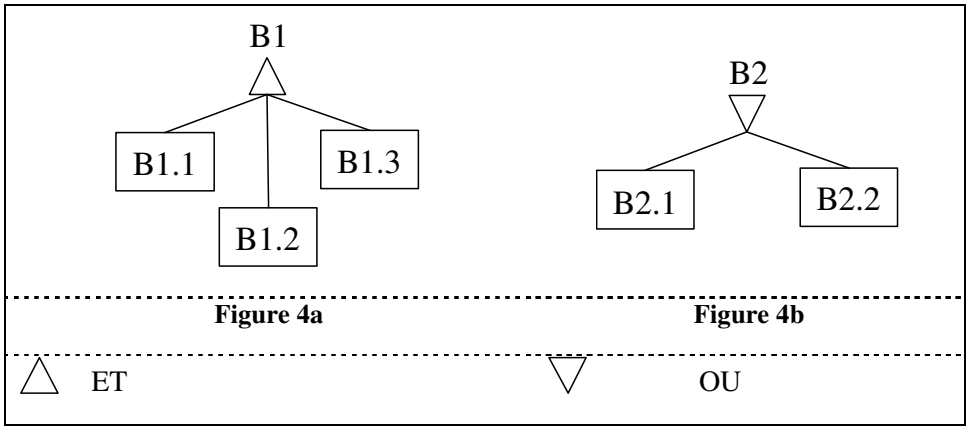


Figure 15. Représentations graphiques des décompositions ET (Figure 4a) et OU (Figure 4b) de buts

- Le rôle des deux opérateurs ET et OU est différent. L'opérateur ET permet d'affiner un but de niveau d'abstraction *i* en un but de niveau d'abstraction inférieur *i+1* ; par exemple de décomposer un but stratégique en buts tactiques ou un but tactique en des buts opérationnalisables. L'opérateur OU introduit des voies alternatives pour atteindre un but. La combinaison des deux opérateurs aboutit à la construction d'arbres d'affinement ET/OU qui permettent de passer de buts stratégiques à des buts opérationnalisables tout en introduisant de la variabilité dans la manière de satisfaire un but donné. On verra que $\mathcal{MiS}$ tire parti de ces propriétés des buts et du mécanisme d'affinement des buts.

Nous avons adopté dans $\mathcal{MiS}$ une formulation des buts qui repose sur une approche linguistique initialement développée par [Prat97], [Prat99]. Cette approche inspirée de la grammaire des cas de Fillmore [Fillmore68] et des extensions de [Dik89] se fonde sur le fait que la sémantique d'un but est capturée par un verbe et des paramètres qui correspondent à des cas ou rôles associés au verbe. Suivant cette approche, un but est donc formulé par un verbe et un ou plusieurs paramètres. Par exemple, le paramètre *qualité* précise un aspect qualitatif requis pour la satisfaction du but. La Figure 16 montre la structure de but retenue dans $\mathcal{MiS}$.

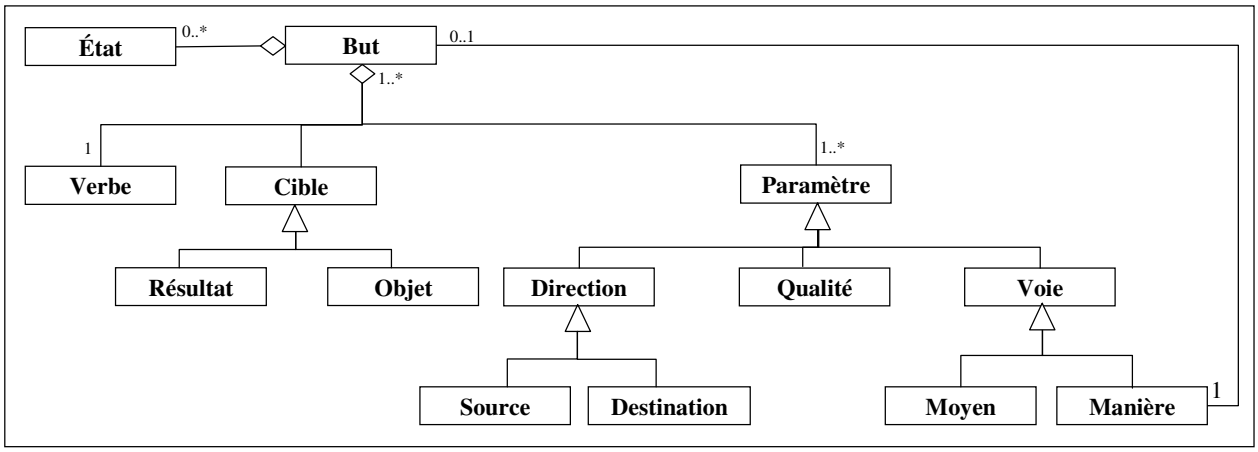


Figure 16. Structure d'un but

Comme le montre la Figure 16, la formulation d'un but comprend un verbe, une cible et des paramètres dits de 'direction'; de 'qualité' et de 'voie'. Les deux premiers éléments sont obligatoires alors que les autres paramètres sont optionnels. On les commente successivement.

La classe du verbe détermine son sens ; la cible qui est le complément d'objet du verbe fait référence à l'entité impactée par le but. La formulation minimale d'un but comporte un verbe et le paramètre cible.

Effectuer une réservation, *Payer une réservation*, *Confirmer une réservation*

sont des expressions minimales de buts assises sur le verbe et la cible.

La cible désigne la ou les entité(s) affectée(s) par le but. Dans les exemples ci dessus : *une réservation* est la cible des verbes *Effectuer, Payer et Confirmer*

Effectuer$_{verbe}$ *(une réservation)* $_{cible}$, *Payer*$_{verbe}$ *(une réservation)* $_{cible}$, *Confirmer* $_{verbe}$ *(une réservation)* $_{cible}$

Le modèle de but montre qu'il y a deux types de cible : *l'objet* et le *résultat*. Un objet existe avant la réalisation du but alors qu'un résultat découle de la satisfaction du but. Ainsi dans

Effectuer$_{verbe}$ *(une réservation)* $_{résultat}$ la réservation est le résultat de la réalisation du but alors que dans *Confirmer* $_{verbe}$ *(une réservation)* $_{objet}$ la réservation existe de manière préalable à la satisfaction du but.

Les *paramètres de 'direction'* caractérisent la source et la destination des entités liées au but.

Effectuer $_{verbe}$ *(une réservation)*$_{objet}$ *(à partir d'une demande)*$_{source}$

est un exemple de formulation comportant une source. La source du but *Effectuer une réservation* est une demande de réservation.

La destination identifie l'emplacement des entités produites par la réalisation du but. Dans l'exemple

Stocker (une réservation)$_{objet}$ *(dans la base de données)*$_{destination}$.

la base de données est la destination de l'objet 'une réservation'.

On comprend à travers ces exemples que les paramètres tels que la destination et la source précisent l'expression minimale d'un but (verbe+cible) et en facilite sa compréhension sans ambiguïté.

Les *paramètres de 'voie'* font référence aux instruments et approches de réalisation d'un but.

Le moyen décrit l'entité qui sert d'instrument pour atteindre le but. Dans,

Payer$_{verbe}$ *(une réservation)*$_{objet}$ *(par carte bleue)*$_{moyen}$.

la carte bleue est le moyen choisi pour satisfaire le but *Payer une réservation*.

La manière identifie une approche selon laquelle le but peut être satisfait. Dans,

Annuler$_{verbe}$ *(une réservation)*$_{résultat}$ *(par expiration du délai d'attente)*$_{manière}$.

Par expiration du délai d'attente est la manière pour satisfaire le but *Annuler une réservation*.

Dans les deux cas, de moyen et manière on précise la façon dont le but va chercher à être atteint. Ces paramètres comme les précédents apportent de la précision dans la formulation du but. On comprend qu'il puisse être important lorsque le but qualifie un service de rendre la manière ou le moyen explicite car les valeurs de ces paramètres renseignent sur le processus d'atteinte du but qui sera exécuté par les services logiciels sous jacents au service intentionnel.

La Figure 16 montre enfin que seule la manière est un paramètre complexe puisqu'une manière peut être un but. Par exemple,

Améliorer (les services offerts aux clients d'une agence de voyages)$_{\text{cible}}$ *(en mettant en place un système de réservation sur Internet)*$_{\text{manière}}$

est un but dont la manière '*en mettant en place un système électronique de réservation'* est un but : *Mettre en place* $_{\text{verbe}}$ un (*système de réservation*) $_{\text{résultat}}$ *(sur Internet)* $_{\text{moyen}}$.

On voit que cette possibilité est utile pour décrire un but en le situant dans le contexte plus général d'un but de plus haut niveau. Dans l'exemple précédent, le but stratégique *Améliorer les services offerts aux clients d'une agence de voyages est le contexte qui justifie le but tactique Mettre en place système de réservation sur Internet*.

La Figure 16 montre enfin que la formulation linguistique du but est complétée par une définition basée sur un *ensemble d'états*. Comme on l'a mentionné précédemment un but peut être interprété comme un *ensemble d'états désirés*. Le lien agrégatif entre but et état de la Figure 16 traduit cet aspect. Les états sont considérés comme étant ceux d'objets de gestion appartenant à des classes. On utilise la notation pointée usuelle

NomClasse•NomAttribut = 'valeur'

pour formuler les états.

Par exemple, le but *Annuler Réservation* correspond à l'état *Réservation• Statut = 'annulée'* où Réservation est le nom de la classe d'objets et Statut est un attribut caractéristique de réservation.

Il faut cependant noter qu'une action visant à satisfaire un but ne permet pas nécessairement de l'atteindre. Certains buts requièrent de nombreuses actions avant d'être satisfaits. Dans d'autres cas, une action peut ou non permettre la satisfaction du but et, sur la base des actions accomplies, cette action peut être répétée ou une autre action être exécutée. Ceci concourt à définir un but par un ensemble d'états dont un sous ensemble peut être jugé acceptable pour considérer que le but a au moins partiellement été atteint. Dans le cas des réservations hôtelières, le but *Effectuer réservation* à partir d'une demande se définit par l'ensemble d'états suivant :

{demande.étatDem = 'traitée', réservation.étatRes = 'OK', demande.étatDem = 'en attente'}

qui comporte deux sous ensembles d'états de satisfaction du but : une réservation coïncidant avec la demande a été faite, et la demande a été mise en attente.

(ii) Situation initiale et situation finale

L'interface du service intentionnel comporte la description de la situation initiale définissant les paramètres d'entrée et la situation finale correspondant aux paramètres de sortie.

La *situation initiale* caractérise la situation de départ dans laquelle le service peut être exécuté pour satisfaire le but qui lui est associé.

La *situation finale* caractérise la situation d'arrivée lorsque le service a été exécuté.

Dans les deux cas, et comme le montre la Figure 3, les situations définissent la structure d'accueil des paramètres d'entrée et de sortie respectivement. On peu comprendre que compte tenu de la granularité élevée que peuvent avoir les services intentionnels, les paramètres aient une structure complexe. On a chois de modéliser ces structures comme des structures de classes d'objets. Par exemple, la structure de situation initiale du service $S_{\textit{Effectuer une réservation}}$ permettant de réaliser le but *Effectuer réservation* manipule la classe *demande* et la structure de la situation finale de ce même service comporte la classe *réservation* dans le cas où la demande est satisfaite et est transformée en une réservation ou la classe *demande en attente* dans le cas où la demande du client ne peut pas être satisfaite immédiatement. La Figure 17 et la Figure 18 présentent ces structures en détail.

Demande

{[Nom-client /* le nom de la personne qui fait la demande*/ ;
Prenom-client /* le prénom de la personne*/ ;
e-mail-client /* l'adresse électronique de la personne*/ ;
Adresse-client /* l'adresse de la personne*/ ;]
[Num-client] /* le numéro de la personne, dans le cas où elle serait déjà enregistrée dans le système comme client*/ ;
Cat-hotel /* la catégorie d'hôtel demandée*/ ;
Nb-chambre /* le nombre de chambres demandé*/ ;
Periode: {DateDeb: /* date de début de réservation souhaitée*/ ;
DateFin} /* date de fin de réservation souhaitée*/ ;
Prix-plafond /* le prix plafond souhaité*/ ;
Liste-att /* cet attribut (à valeur O/N) précise, dans le cas où la demande ne pourrait être satisfaite immédiatement, si le demandeur accepte ou non d'être mis en liste d'attente*/ ;}

Réservation

{Num-résa /* le numéro de réservation généré suite à la demande du client*/ ;
Période-résa : {DateDeb: /* date de début de réservation */ ;
DateFin} /* date de fin de réservation */ ;
Nb-chambre /* le nombre de chambres réservé*/ ;
[Num-hotel /* le numéro de l'hôtel*/ ;
cat-hotel /* la catégorie de l'hôtel*/ ;
Adresse-hotel /* l'adresse de l'hôtel*/ ;
e-mail-hotel /* l'adresse électronique de l'hôtel*/ ;]
Prix-resa /* le prix des chambres réservées*/ ;
Num-client /* le numéro du client qui a effectué la demande de réservation*/ ;}

Figure 17. Spécification de la situation initiale du service S *Effectuer réservation*

Demande en attente	
Num-demande-att	/* le numéro de la demande en attente*/ ;
[Num-client]	/* le numéro de la personne enregistrée dans le système*/ ;
[Num-hotel	/* le numéro de l'hôtel*/ ;
cat-hotel	/* la catégorie de l'hôtel*/ ;
Adresse-hotel	/* l'adresse de l'hôtel*/ ;
e-mail-hotel	/* l'adresse électronique de l'hôtel*/ ;]
Nb-chambre	* le nombre de chambres demandé*/;
Periode: {DateDeb:	/* date de début de réservation */ ;
DateFin	/* date de fin de réservation */ ;}
Prix-plafond	/* le prix plafond souhaité*/ ;

Figure 18. Spécification de la situation finale du service S *Effectuer réservation*

3.2.3. Comportement

Le comportement d'un service est spécifié à travers la *pré-condition* et la *post-condition* qui sont respectivement des conditions sur les états des objets de la situation initiale et la situation finale.

Selon le méta-modèle, un service peut être vu comme la transition d'une pré-condition vers une post-condition résultant de la réalisation du but du service. La Figure 19 présente un extrait du méta-modèle.

La *pré-condition* : permet de préciser les états des classes constituant les conditions devant être satisfaites pour que le but du service puisse être satisfait.

La *post-condition* : permet de préciser les états des classes obtenus suite à la réalisation du but du service. Les post-conditions peuvent être de deux types. Elles sont normales dans le cas où le service s'exécuterait normalement. Par contre, elles sont exceptionnelles lorsqu'une erreur intervient au cours de la réalisation du service.

Selon le méta-modèle, 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. Cet aspect est spécifié par la propriété *Règle de transition qui* permet de préciser les contraintes régissant la réalisation d'opérations entraînant le changement d'états d'objets.

Par exemple, la *pré-condition* du service *Effectuer réservation en vérifiant les disponibilités* consiste en l'existence d'une demande de réservation déjà créée. La *post-condition* correspond dans le cas normal au fait que la demande est archivée et la réservation est valide. Par contre dans le cas exceptionnel, elle correspond au fait que la réservation n'est pas possible. La règle de transition permet de préciser que seules les demandes créées peuvent être archivées ou mises en attente.

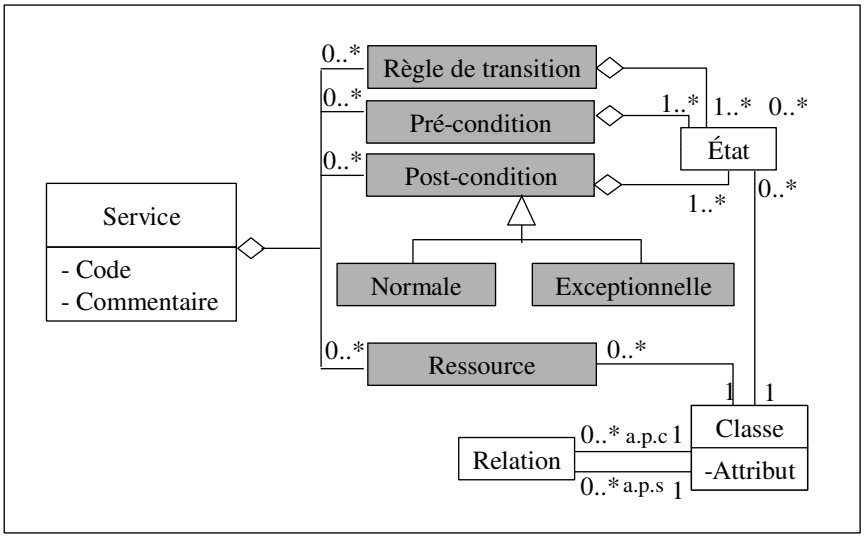


Figure 19. Le comportement d'un service

Les *pré-conditions* et *post-conditions* font référence à des états de classes. Pour un service *S* dont la situation initiale est *I*, la situation finale est *J*, la pré-condition du service *S* porte sur les états de la structure de classes de *I*. De la même façon, la post-condition du service *S* porte sur les états de la structure de classes de *J*.

Ainsi, si nous prenons l'exemple du service *Effectuer réservation en vérifiant les disponibilités* :

- La *pré-condition* s'écrit *Demande.état = 'créée'* ce qui signifie que l'état de la classe demande est « créée ».
- La *post-condition* dans le cas normal s'écrira *Demande.état = 'archivée'* $\wedge$ *Réservation.état = 'valide'*. Par contre dans le cas exceptionnel, la post-condition s'écrira *Demande.état = 'résa-impossible'*.
- La règle de transition s'écrit (< *Demande.état = 'créée'*, *Demande.état = 'archivée' AND <-,Réservation.état = 'valide'> OR < Demande.état = 'créée'*, *Demande.état = 'en attente).* Ce qui signifie qu'une transition d'états est possible entre (i) les états '*créée'* et *'archivée'* d'une demande ou (ii) les états *'créée'* et *'en attente'* d'une demande.

Les ressources correspondent aux classes dont le service a besoin pour sa réalisation. Dans notre exemple de réservation, le service *Effectuer réservation* utilise l'hôtel comme ressource. En effet, l'existence d'un hôtel est indispensable à la réservation.

3.2.4. Composition

La troisième partie descriptive d'un service caractérise le type du service, atomique ou agrégat et dans ce dernier cas décrit la forme que prend la composition.

Comme le montre le méta-modèle à Figure 13, il y a différents types de services intentionnels. Au premier niveau de spécialisation les services sont soit *atomiques* soit *agrégats*. Un service atomique est indécomposable en services ; il est associé à une orchestration de services logiciels. On verra au chapitre 4 de quelle façon le pont est établi entre le service intentionnel atomique et la composition de services logiciels qui permet de l'exécuter. L'atomicité du service tient à la nature du but qui le caractérise : il est 'opérationnalisable' au sens introduit en section 3.1, c'est-à-dire qu'il est possible d'associer une séquence d'actions qui en permette sa réalisation. Dans ce sens, on peut dire qu'un service atomique est *exécutable*.

A l'opposé, un service agrégat cherche à satisfaire un but tactique ou stratégique (c.f. section 3.1) et sa composition est calquée sur l'affinement ET/OU du but. La composition d'un service intentionnel introduit deux types de services pour cette raison : ceux qui sont justifiés par une décomposition OU du but, les variants et ceux qui correspondent à une décomposition ET, les composite. Un service agrégat se décompose jusqu'à obtenir des services atomiques exécutables. Il n'est pas exécutable mais permet l'exécution par navigation dans l'arbre d'affinement ET/OU qui est le fondement de sa composition ; nous le qualifions pour cette raison de service *navigationnel*.

Nous détaillons chacun des types de service introduits à la Figure 13 dans la suite de cette section.

3.2.4.1. Service atomique

Un *service atomique* n'est pas décomposable en d'autres services intentionnels. Le but qui lui est associé ne nécessite pas d'être décomposé en sous buts. Dans ce cas, le but est 'opérationnalisable' et son opérationnalisation est assurée par les actions qui composent le service atomique. Un service atomique peut donc être qualifié d'exécutable.

Il est important de noter que l'atomicité d'un service intentionnel est justifiée dans la perspective intentionnelle : il est atomique parce qu'intentionnellement, il correspond à un but opérationnalisable. En revanche, il est possible que sa réalisation requière un flux d'actions complexes qui sera modélisé, par exemple en BPEL4WS [Andrews03] comme une composition des services logiciels complexe. La granularité opératoire d'un service atomique

peut être plus ou moins grande mais sa granularité intentionnelle est faible. Du point de vue intentionnel, le service atomique a vocation à être réutilisé comme brique de construction dans la construction de services intentionnels de granularité plus importante.

La Figure 20 présente la partie du méta-modèle concernant la spécialisation d'un service intentionnel en service atomique.

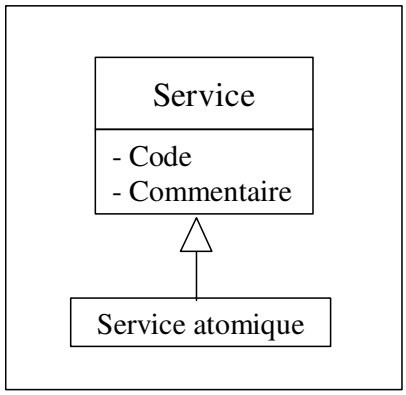


Figure 20. Type: Service atomique

Notation graphique

La notation graphique d'un service atomique est présentée à la Figure 21. Elle a la forme d'un rectangle contenant le code du service dans la forme introduite : S $_{indice}$, l'indice énonçant le but.

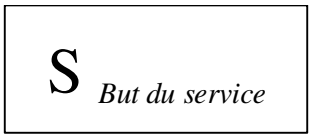


Figure 21. Représentation graphique d'un service atomique

Exemples

Dans l'exemple du système de réservations hôtelières le but *Effectuer une réservation* délimite le service atomique S $_{Effectuer\ une\ réservation}$ ou encore le but *Payer une réservation* peut être associé au service atomique S $_{Payer\ une\ réservation}$, graphiquement présentés comme suit :

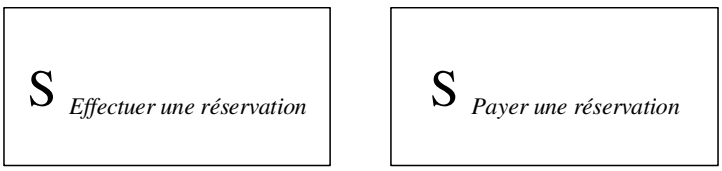


Figure 22. Exemples de représentations graphiques de services atomiques

3.2.4.2. Service agrégat

Comme le montre l'extrait du méta-modèle $\mathcal{MiS}$ présenté à la Figure 24, un *service agrégat* est composé d'autres services qui peuvent être atomiques ou eux-mêmes des services agrégats. Il y a donc une composition récursive des services. Comme introduit précédemment la composition d'un agrégat intentionnel est dirigée par les buts ; elle est conforme à la décomposition que requiert le but du service pour se donner les moyens de passer d'un but/service à portée stratégique/tactique à des buts/services opérationnalisables. La décomposition est de type ET/OU et se traduit par l'introduction dans le méta-modèle $\mathcal{MiS}$ par deux types de services agrégats, les *services à variation* et les *services composites*. Les premiers établissent des liens OU entre services composants et offrent des choix dans la manière d'atteindre le but du service ; les seconds établissent des liens ET entre services composants et assurent le passage d'un niveau intentionnel à un autre niveau plus proche de l'opérationnalisation. On présente les deux types avec plus de détails.

3.2.4.2.1.Service composite

Un service composite est un service qui s'appuie sur une décomposition de type ET du but du service en sous buts et services associés. Rappelons que si un but *B1* se décompose en buts *B1.1*, *B1.2*, *B1.3* (cf. Figure 15a) l'acception est que *B1* est réputé réalisé si *B1.1*, *B1.2* et *B1.3* le sont. Il y a donc un parallèle comme le montre la Figure 23, entre la décomposition du but *B1* et celle du service agrégat S_{B1}. Le service S_{B1} sera fourni si les services composants $S_{B1.1}$, $S_{B1.2}$, $S_{B1.3}$, le sont.

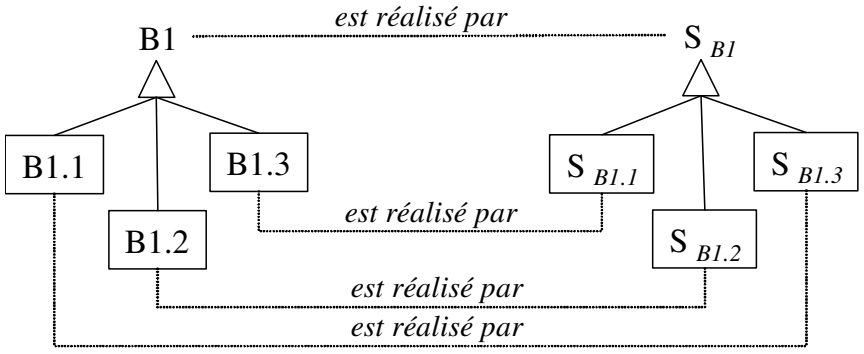


Figure 23. Réalisation des buts par des services

L'extrait du méta-modèle $\mathcal{MiS}$ présenté à la Figure 24 montre que les composants d'un service agrégat sont des services par conséquent atomiques ou agrégats. Il y a donc composition récursive de services dans $\mathcal{MiS}$.

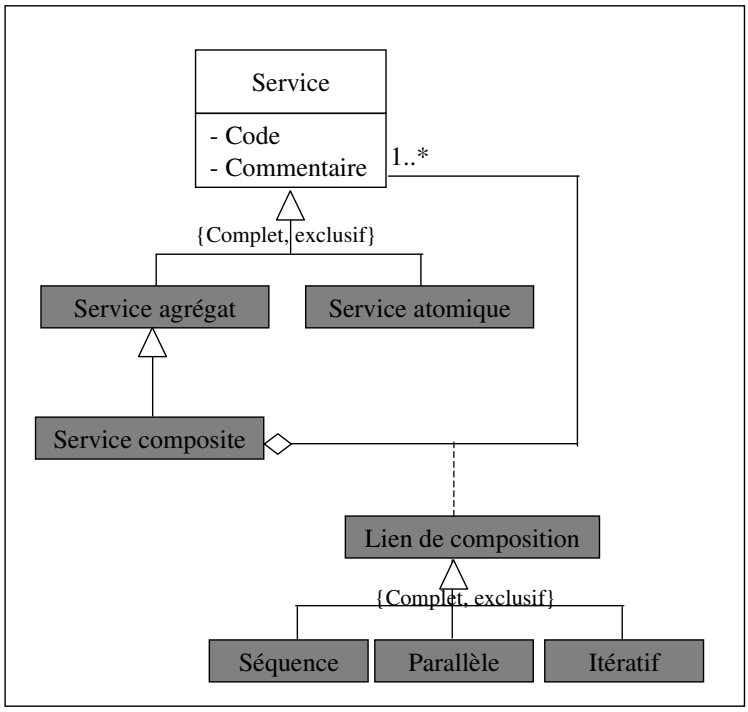


Figure 24. Type: Service composite

Par ailleurs la Figure 24 montre qu'il y a trois opérateurs de composition : la *séquence,* le *parallélisme* et *l'itération*. Les contraintes de la spécialisation montrent la couverture est complète et exclusive.

Composition séquentielle

Il s'agit du cas le plus typique dans lequel le service agrégat requiert la livraison séquentielle des services composants. Le service composite séquentiel construit ainsi un nouveau service de même niveau d'abstraction que ceux qu'ils composent mais de niveau de granularité intentionnelle plus élevé.

Notation

Nous associons au lien de composition séquentiel le symbole « • ». Ainsi, le service composite séquentiel S_b se note :

$$S_b = \bullet(S_i, \ldots, S_n)$$

où S_i désigne un service composant lié à S_b par un lien de séquence.

Dans le cas des réservations hôtelières, la confirmation d'une réservation nécessite satisfaction du but *Accepter paiement* d'une réservation qui ne peut se faire que si la réservation a été effectuée, c'est-à-dire si le but *Effectuer Réservation* a été satisfait. Le paiement se fait ensuite suivant le moyen choisi par le client. Les deux buts sont soumis à un

enchaînement : il faut d'abord réaliser le but *Effectuer réservation* pour pouvoir satisfaire le but *Payer réservation*. Ceci se traduit par la composition séquentielle de services suivante :

$$S_{Confirmer\ réservation} = \bullet\ (S_{Effectuer\ réservation,}\ S_{Payer\ réservation})$$

Représentation graphique

Comme le montre la Figure 25, la composition séquentielle de services est représentée comme le montre la Figure 25 par un connecteur comportant le symbole « • » de composition séquentielle reliant les rectangles figurant les services composants. Le rectangle global représente le service composite. Ainsi la figure ci-dessous visualise la composition :

$S_b = \bullet(S_i, \ldots, S_n)$

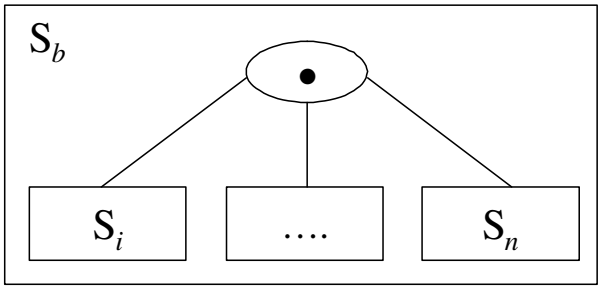


Figure 25. Représentation graphique d'un service composite

La Figure 26 est la représentation graphique du service composite $S_{Confirmer\ une\ réservation}$ présenté plus haut dans cette section. Ce service est la composition de deux services à savoir : $S_{Effectuer\ réservation}$ et $S_{Payer\ réservation}$. Ces deux services sont attachés par le symbole du lien de séquence « • » afin de montrer qu'ils sont réalisés d'une manière séquentielle.

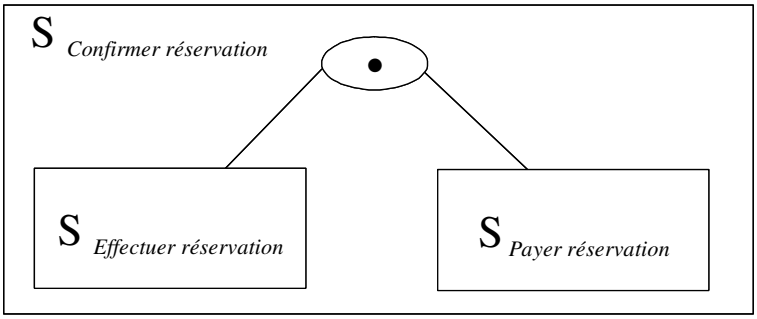


Figure 26. Représentation graphique du service composite $S_{Confirmer\ une\ réservation}$

Composition parallèle

Les services réalisés parallèlement peuvent être exécutés selon un ordre quelconque et non dans un ordre impératif comme précédemment.

La Figure 24 illustre le service composite parallèle. Il est constitué d'un lien de composition parallèle qui relie les services qui s'exécutent dans un ordre quelconque.

Notation

Nous associons au lien de composition parallèle le symbole « // ». Ainsi, le service composite parallèle S_p se note :

$$S_p = // (S_i,\ldots, S_n)$$

où les S_i désignent les services composants qui peuvent être réalisés en parallèle.

Dans le cas des réservations hôtelières, la recherche d'un hôtel satisfaisant certains critères dans un lieu donné et celle d'agences de location de bicyclettes dans le même lieu peuvent être satisfaites en parallèle. Le service composite $S_{Rechercher\ package}$ est donc une composition dans laquelle les deux services composants $S_{Chercher\ location\ bicyclette}$ et $S_{Chercher\ hôtel}$ peuvent être exécutés en parallèle, soit :

$$S_{Rechercher\ package} = // (S_{Chercher\ location\ bicyclette}, S_{Chercher\ hôtel})$$

Notation graphique

La Figure 27 présente la représentation graphique de la composition parallèle de services que nous avons retenue. Le symbole « // » est dans une icône ovale reliant les services S_i tels que i $\in$ {1,..,n} qui composent le composite S_p et peuvent être exécutés dans n'importe quel ordre.

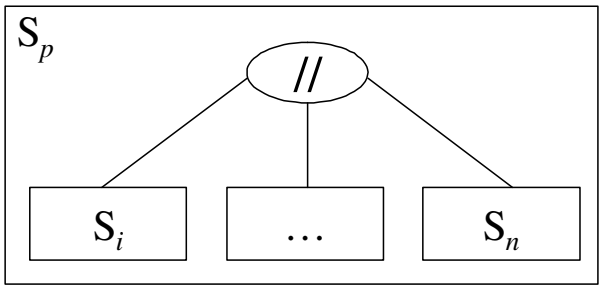


Figure 27. Représentation graphique d'un service composite parallèle

La représentation graphique du service parallèle

$$S_{Rechercher\ package} = // (S_{Chercher\ location\ bicyclette}, S_{Chercher\ hôtel})$$

est la suivante :

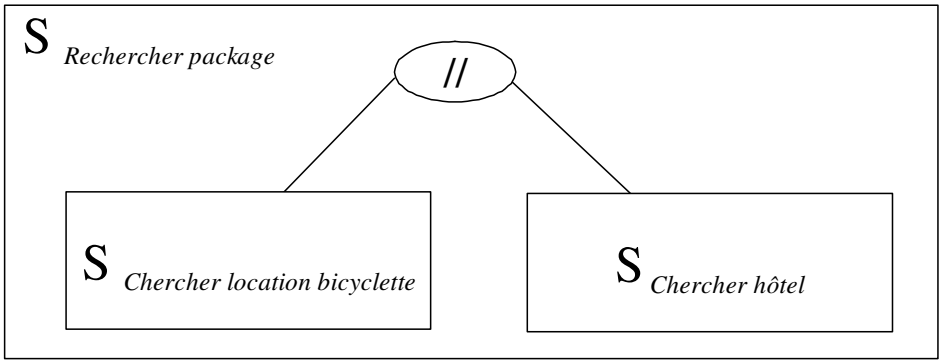


Figure 28. Représentation graphique du service parallèle S *Rechercher package*

Composition itérative

La satisfaction d'un but peut requérir l'exécution itérative d'un même ensemble d'actions ; Cela se traduit par l'introduction de l'opérateur d'itération dans la composition. L'itération peut s'appliquer à un service entrant dans une composition séquentielle ou parallèle ou bien à un service agrégat dans son ensemble.

Notation

Nous associons à l'itération le symbole « * ». Ainsi, la répétition du service S_1 dans le service composite séquentiel S_s se note :

$$S_s = \bullet (S^*_1, \ldots, S_n)$$

où les S_i désignent les services composants qui doivent être réalisés en séquence.

Dans le cas des réservations hôtelières, le service permettant d'*Effectuer réservation avec attente* est un service composite $S_{\textit{Effectuer réservation avec attente}}$ à deux composants $S_{\textit{Mettre demande en attente}}$ et $S^*_{\textit{Exploiter nouvelle disponibilité}}$ dont l'un, le second est exécuté de manière répétitive. Chaque fois qu'une disponibilité de chambre se dégage le service $S_{\textit{Exploiter nouvelle disponibilité}}$ cherche à utiliser la ou les nouvelles disponibilités pour satisfaire les demandes en attente. En effet à faire sortir de l'attente. C'est donc l'exécution itérative de ce service qui peut déboucher sur la transformation d'une demande de la file d'attente en réservation. Cette composition incluant une itération se note :

$$S_{\textit{Effectuer réservation avec attente}} = \bullet (S_{\textit{Mettre demande en attente}}, S^*_{\textit{Exploiter nouvelle disponibilité}})$$

Représentation graphique

La Figure 29 présente la représentation graphique d'un service composite S_a faisant appel à l'opérateur d'itération. Dans ce cas, le symbole « * » est mis au-dessus du rectangle désignant le service itéré. S_j un service agrégat est itératif, le service est encadré par un rectangle en pointillé et le symbole « * » est mis en avant. Le cas de la Figure 17a est celui où l'un des services atomiques de la composition est itératif tandis que le cas de la Figure 17b adresse celui où l'un des services composant est lui-même un agrégat (à variation) qui nécessite une exécution itérative.

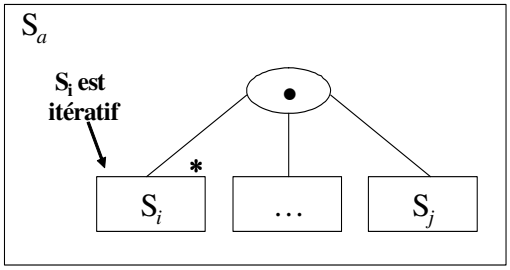


Figure 17a

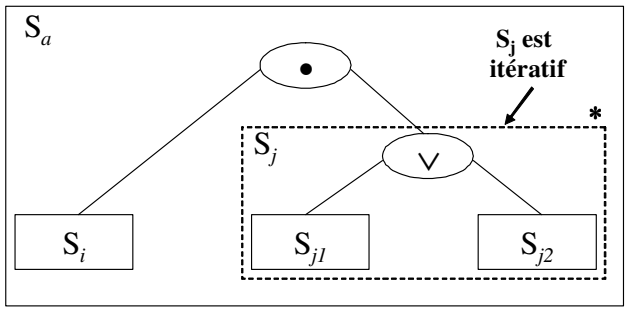


Figure 17b

Figure 29. Exemples de représentation graphique de compositions comportant des itérations

La représentation graphique de l'exemple précédent de service composite à itération du cas des réservations :

$$S_{\textit{Effectuer une réservation avec attente}} = \bullet\,(S_{\textit{Mettre demande en attente}}, S^*_{\textit{Exploiter nouvelle disponibilité}})$$

est la suivante :

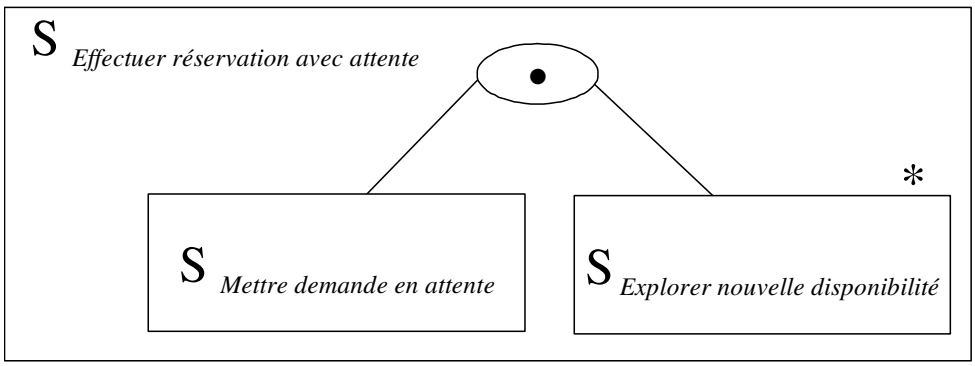


Figure 30. Représentation graphique du service $S_{\textit{Effecteur une réservation avec attente}}$ comportant un service itératif $S_{\textit{Explorer nouvelle disponibilité}}$

3.2.4.2.2.Service à variation

La Figure 15 montre que la composition permet d'introduire de la *variabilité* dans la manière d'atteindre le but du service agrégat. Cette forme d'agrégation de services permet de mettre en évidence les caractéristiques communes à tout processus d'atteinte d'un but (services

communs) mais aussi celles qui sont des variations qui permettent de répondre aux besoins des clients de manière différenciée (services alternatifs). Ces services alternatifs sont appelés des *services à variation.*

Le concept de variabilité a prouvé sa pertinence et son utilité dans plusieurs domaines d'ingénierie lorsque des compagnies ne sont plus confrontées au développement d'un unique produit mais à celle du développement de lignes de produits et de familles de produits. Le premier cas est justifié par l'évolution du produit, par exemple un lecteur DVD, tandis que le second correspond à l intégration de différentes lignes de produit telles les lignes de DVD et de MP3. Le concept de variabilité a été introduit pour permettre la distinction entre les parties communes et les parties spécifiques dans un ensemble de lignes de produits d'une même famille. Expliciter les aspects communs et les aspects variables a deux principaux avantages

(a) l a réutilisation des parties communes [Ommering02], [Tomphson01] et,

(b) l'adaptation des produits à différents clients et différentes situations organisationnelles [Svahnberg01]

Notre position est que la notion de variabilité s'applique parfaitement au domaine des services. Elle enrichit la manière de rendre un service en permettant l'adaptation du service rendu au profil du client qui le demande. Pour parvenir à cet effet, il est nécessaire d'introduire la variabilité dans la description des services et c'est ce qui est proposé dans $\mathcal{MiS}$ grâce à la notion de *service à variation*.

Un service à variation correspond à une situation qui nécessite l'exploration de différentes possibilités alternatives car ce sont des situations dans lesquelles il existe différentes façons d'atteindre un but. Le service à variation décrit le point auquel la variation est possible [Czarnecki00], le *point de variatio*n (c.f Figure 31) ainsi que chacune des solutions alternatives par un service. L'exécution du service exhibe l'ensemble de possibilités permettant de satisfaire le but et demande au client de choisir celle qui correspond le mieux à sa situation. On verra au chapitre 6 les mécanismes que nous avons développé pour faciliter le choix et donc l'exécution différenciée des services.

Notons que l'on retrouve ici la dimension intentionnelle qui caractérise $\mathcal{MiS}$ puisqu'il est évident qu'un service à variation correspond à une décomposition de type OU du but du service agrégat en services représentant des alternatives de satisfaction du but. En d'autres termes, le but du service à variation est affiné par des buts plus précis proposant chacun une manière différente pour satisfaire le but de départ. La réalisation du but d'un service à variation consiste (1) à choisir l'une des alternatives la plus adaptée à la situation donnée et (2) à l'exécuter.

La Figure 15 montre que les variations sont de trois types : *choix alternatif*, *choix multiple* ou bien *choix de chemin*. Le premier cas correspond à un XOR (OU exclusif), le second à un OR (OU inclusif) entre services; le troisième cas traduit une variation non plus de service mais d'une composition de services. On détaille ces trois types dans la suite.

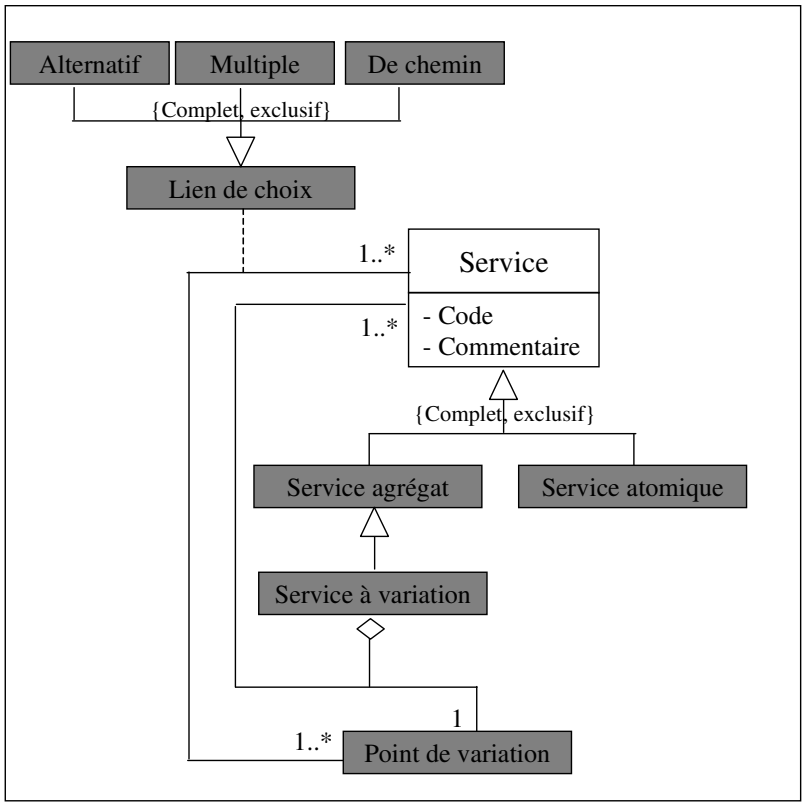


Figure 31. Type : Service à variation

Service à choix alternatif

Un service à choix alternatif exprime un choix d'alternatives dans la manière de réaliser le but du service agrégat en offrant des services composants qui sont mutuellement exclusifs. Chaque service représente une manière différente pour satisfaire le but de l'agrégat. Ainsi, un service à choix alternatif regroupe plusieurs services mutuellement exclusifs et construit ainsi un nouveau service de même niveau d'abstraction mais de granularité plus élevée.

La Figure 31 illustre la partie du méta-modèle $\mathcal{MiS}$ concernant la structure d'un service à choix alternatif. Un service à choix alternatif est constitué d'un point de variation qui exprime le choix alternatif des services qui le composent. Au moment de l'exécution, un seul service sera choisi parmi l'ensemble d'alternatives offertes.

Notation

Nous associons au lien de choix alternatif le symbole « ⊗ ». Ainsi, le service à choix alternatif S_a s'écrit comme suit :

$$S_a = \otimes (S_1, S_2, \ldots, S_n)$$

où les S_i désignent les codes des services réalisant le but *a* du service à choix alternatif. Le symbole « ⊗ » représente le lien de choix alternatif. Il indique qu'un seul service parmi les S_i doit être choisi.

Dans le cas des réservations hôtelières le but *Payer réservation* peut être satisfait de différentes façons : le paiement de la réservation peut se faire par exemple *par carte de crédit*, *par chèque* ou *par bon de commande*. Chacun de ces modes de paiement suggère d'offrir un service qui lui soit adapté. Cela milite pour l'introduction de trois services $S_{\textit{Payer réservation par carte de crédit}}$, $S_{\textit{Payer réservation par chèque}}$ et $S_{\textit{Payer réservation par bon de commande}}$. Ces services sont exclusifs et composent le service à variation à choix alternatif $S_{\textit{Payer réservation}}$. La composition à variation se note :

$$S_{\textit{Payer réservation}} = \otimes (S_{\textit{Payer réservation par carte de crédit}}, S_{\textit{Payer réservation par chèque}}, S_{\textit{Payer réservation par bon de commande}})$$

Représentation graphique

Un service à choix alternatif est représenté graphiquement par une structure disjonctive constituée de rectangles représentant les services et reliés par le symbole « ⊗ ». Ce dernier représente le point de variation et indique qu'un seul des services parmi les Si peut être choisi (cf. Figure 32).

Par exemple le service à variation à choix alternatif $S_a = \otimes (S_{1i}, S_2, \ldots, S_n)$ est dessiné comme indiqué ci-dessous :

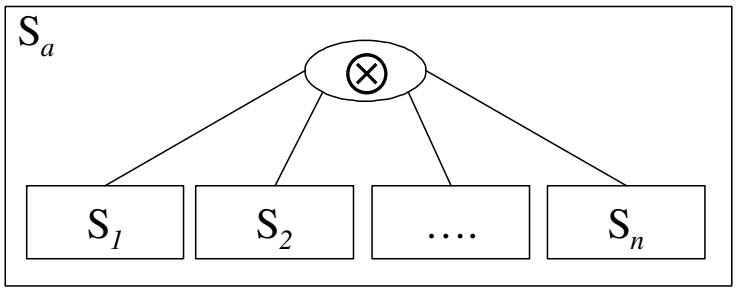


Figure 32. Représentation graphique d'un service à choix alternatif

La Figure 33 est un exemple de représentation du service à choix alternatif $S_{\textit{Payer une réservation}}$

$$S_{\textit{Payer réservation}} = \otimes (S_{\textit{Payer réservation par carte de crédit}}, S_{\textit{Payer réservation par chèque}}, S_{\textit{Payer réservation par bon de commande}})$$

présenté dans cette section et relatif aux moyens de paiement d'une réservation hôtelière.

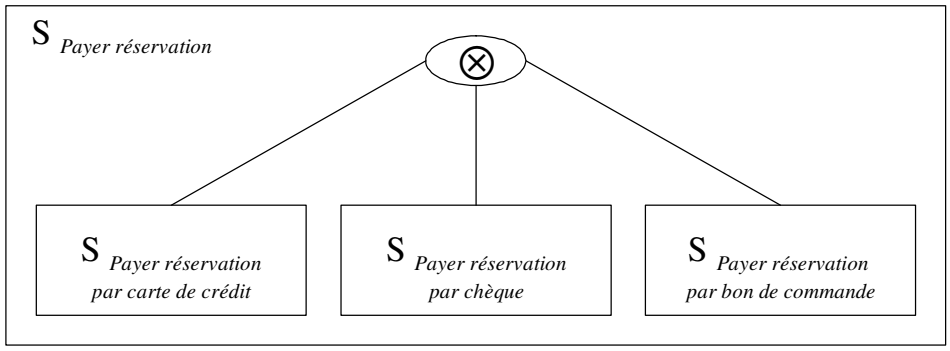


Figure 33. Représentation graphique du service à choix alternatif S *Payer une réservation*

Service à choix multiple

Un service à choix multiple offre un choix non exclusif de réalisation du but du service agrégat en groupant plusieurs services parmi lesquels au moins mais éventuellement plusieurs pourront être choisis.

On peut noter que tous les services formant un service à choix multiple permettent d'atteindre le même but (celui du service à variation) mais par des moyens ou des manières différentes. En d'autres termes c'est souvent le paramètre 'voie' de la structure du but (voir section 3.2.2) qui est à la source des variations. Le service à choix multiple est différent du service à choix alternatif car il offre des services qui ne sont pas exclusifs. L'exécution d'un tel service peut conduire à exécuter plusieurs des services offerts suivant les besoins spécifiques du moment.

La Figure 31 illustre la partie du méta-modèle $\mathcal{MIS}$ concernant la structure d'un service à choix multiple. Un service à choix multiple est constitué d'un point de variation qui exprime le choix multiple des services qui le composent. La satisfaction du but du service à choix multiple se fait à travers le choix d'un ou plusieurs d'entre eux.

Notation

Nous associons au lien de choix multiple le symbole « $\vee$ ». Ainsi, le service à choix multiple S_m s'écrit comme suit :

$$S_m = \vee (S_1, S_2, \ldots, S_n)$$

où les S_i sont les codes des services atomiques réalisant le but *m* du service à choix multiple. Le symbole « $\vee$ » représente le lien de choix multiple. Il indique qu'un ou plusieurs services atomiques parmi les S_i doivent être choisis.

Dans le cas des réservations hôtelières, le service $S_{Choisir\ vol}$ est à choix multiple. Il existe en effet plusieurs manières non exclusives de satisfaire le but *Choisir vol*. On peut utiliser différents critères tels que le prix, la compagnie aérienne ou les horaires pour effectuer le choix. Plusieurs de ces critères ou même tous peuvent être utilisés pour prendre la décision finale. Chacun des processus de choix selon un des trois critères correspond à un service et la panoplie de ces services constitue un service à variation à choix multiple. La définition de ce service est la suivante :

$$S_{Choisir\ vol} = \vee (S_{Choisir\ un\ vol\ par\ compagnie\ aérienne}, S_{Choisir\ un\ vol\ par\ prix}, S_{Choisir\ un\ vol\ par\ horaire})$$

correspondant respectivement aux différentes manières de rechercher un vol à savoir : C*hoisir un vol par compagnie aérienne*, C*hoisir un vol par prix* et C*hoisir un vol par horaire*. Un ou plusieurs services atomiques peuvent être sélectionnés pour choisir un vol. En effet, le client peut procéder la recherche d'un vol par le prix ensuite par le choix d'une compagnie aérienne.

Représentation graphique

Un service à choix multiple est décrit graphiquement, comme le montre la Figure 34 par une structure disjointe constituée de rectangles représentant les services reliés par le symbole « ∨ » indiquant le point de variation où un ou plusieurs services peuvent être choisis.

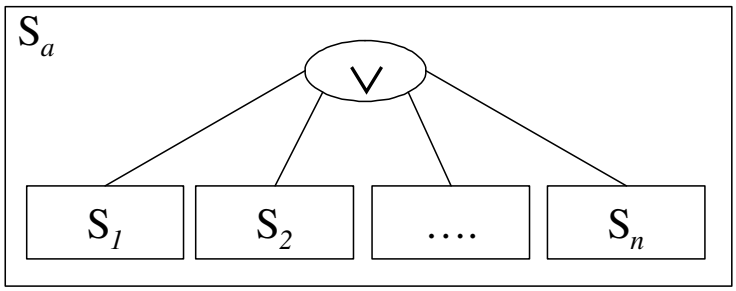


Figure 34. Représentation graphique d'un service à choix multiple

L'exemple du cas des réservations introduit ci-dessus

$$S_{Choisir\ un\ vol} = \vee (S_{Choisir\ un\ vol\ par\ compagnie\ aérienne}, S_{Choisir\ un\ vol\ par\ prix}, S_{Choisir\ un\ vol\ par\ horaire})$$

est présenté graphiquement à la Figure 35.

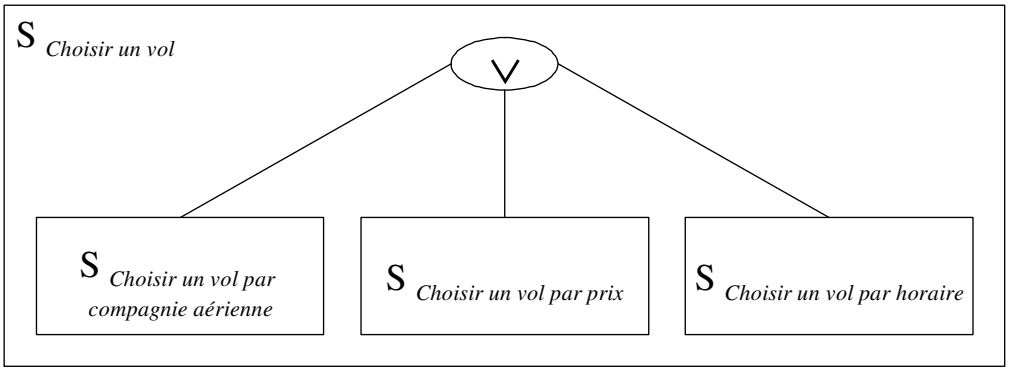


Figure 35. Représentation graphique du service à choix multiple S *Choisir un vol*

Service à variation de chemin

Les deux cas de variation précédents portent sur des alternatives exclusives ou non de services considérés individuellement. On est dans une logique où S_a est alternatif de S_b et de S_c parce que *a, b et c* sont des buts en situation de OR ou de XOR. La variation de chemin introduit une variation qui porte sur un 'chemin' de buts, c'est-à-dire sur des enchaînements alternatifs de buts. La Figure 36 ci dessous illustre cette situation :

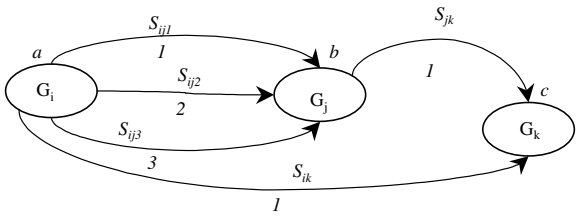


Figure 36. Illustration d'une variation de chemin

Le graphe de la Figure 36 visualise les chemins d'atteinte de trois buts : G_i, G_j et G_k qui sont les nœuds du graphe. Les arcs du graphe visualisent les manières d'atteinte d'un but. Par exemple, il y a trois façons d'atteindre G_j à partir d'une situation initiale qui correspond au fait que G_i est satisfait : elles sont notées S_{ij1}, S_{ij2} et S_{ij3}. Le graphe ci-dessus illustre la notion de variation chemin : on peut satisfaire G_k soit directement à partir de G_i (par S_{ik}), soit par la satisfaction de G_j (par l'une des trois manières S_{ij1}, S_{ij2} et S_{ij3}) d'abord puis celle de G_k par S_{jk}.

La variation de chemin permet de capturer dans la définition d'un service agrégat des chemins multiples et alternatifs permettant d'atteindre le but de l'agrégat. Un service à variation de chemin offre un choix dans la manière de réaliser un but en utilisant des chemins alternatifs et exclusifs.

La Figure 31 illustre la partie du méta-modèle $\mathcal{MiS}$ concernant la structure d'un service à variation de chemin. Un service à variation de chemin comporte un point de variation qui

exprime le choix exclusif des services qui le composent. Un seul service est choisi parmi plusieurs.

Notation

Nous associons au lien de choix le symbole « ∪ ». Ainsi, le service à variation de chemin S_c s'écrit comme suit :

$$S_c = \cup (S_1, S_2, \ldots, S_n)$$

où les S_i sont les codes des services associés par un lien de choix représenté par le symbole « ∪ ». Chaque service composant représente un chemin distinct pour atteindre le but du service à variation.

La Figure 37 utilise la même forme de graphe que ci-dessus pour visualiser deux chemins dans un exemple de situations du cas des réservations hôtelières. Le graphe montre qu'il y a deux chemins aboutissant à l'annulation d'une réservation effectuée. Celui qui passe par son paiement puis son annulation du fait de la décision du client et celui qui conduit à annuler une réservation non payée au-delà d'un certain délai. Cette situation est capturée par un service à variation de type chemin multiple $S_{Annuler\ réservation}$ défini ainsi :

$$S_{Annuler\ réservation} = \cup(S_{Annuler\ réservation\ par\ choix\ du\ client}, S_{Annuler\ réservation\ par\ expiration\ du\ délai\ d'attente})$$

Où $S_{Annuler\ réservation\ par\ choix\ du\ client} = (\bullet\ (S_{Payer\ réservation}, S_{Annuler\ réservation\ par\ demande\ explicite\ du\ client})$

Le service $S_{Annuler\ réservation}$ propose deux chemins : l'un correspond au service composite défini par $S_{Annuler\ réservation\ par\ choix\ du\ client} = \bullet\ (S_{Payer\ réservation}, S_{Annuler\ réservation\ par\ demande\ explicite\ du\ client})$, l'autre est un service atomique $S_{Annuler\ réservation\ par\ expiration\ du\ délai\ d'attente}$.

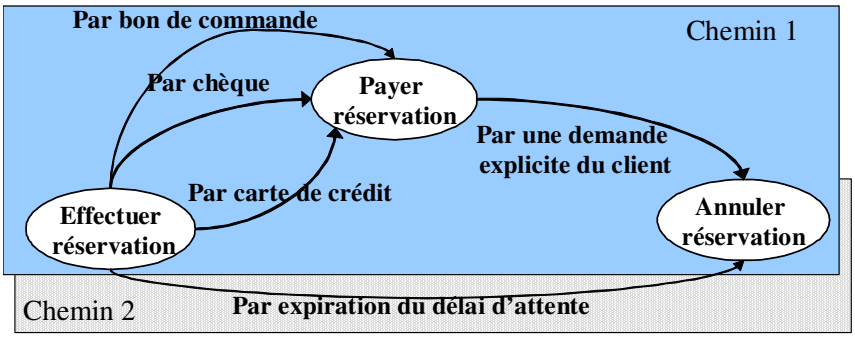


Figure 37. Les chemins formant le service à variation de chemin $S_{Annuler\ réservation}$

Représentation graphique

La Figure 38 présente la représentation graphique d'un service à variation de chemin.

Chaque ensemble de services représentant une composition possible de services pour atteindre le but du service à variation, est relié aux autres compositions par le symbole « ∪ » représentant le point de variation.

Dans la Figure 38, le service $S_c = \cup (S_a, \otimes (S_i, S_j, S_k))$ coffre deux chemins, l'un utilisant S_a, l'autre étant un service à choix alternatif $\otimes (S_i, S_j, S_k)$.

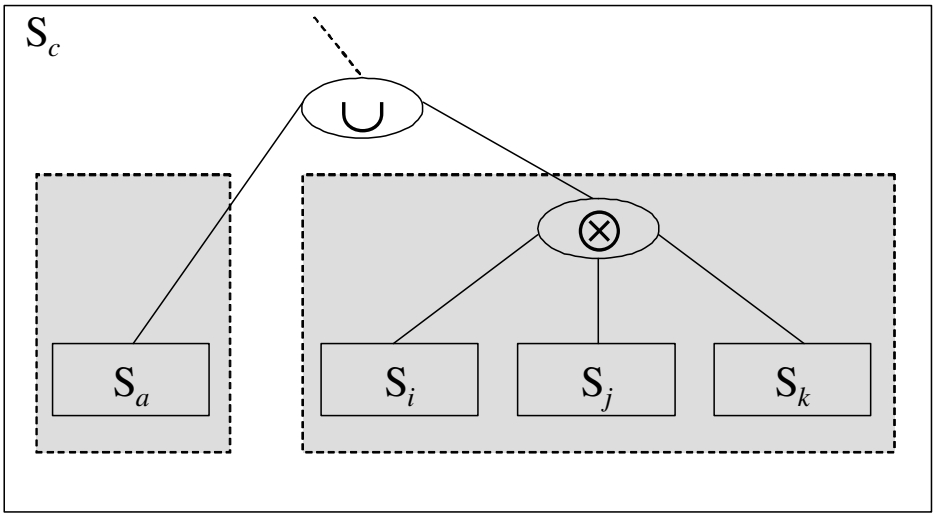


Figure 38. Représentation graphique d'un service à variation de chemin

La Figure 39 est la représentation graphique du service à variation de chemin :

$S_{Annuler\ réservation} = \cup (S_{Annuler\ réservation\ par\ choix\ du\ client}, S_{Annuler\ réservation\ par\ expiration\ du\ délai\ d'attente})$

Comme le montre la figure, ce service est composé de deux chemins possibles identifiés par les services $S_{Annuler\ réservation\ par\ choix\ du\ client}$ et $S_{Annuler\ réservation\ par\ expiration\ du\ délai\ d'attente}$ et reliées par le point de variation « ∪ ».

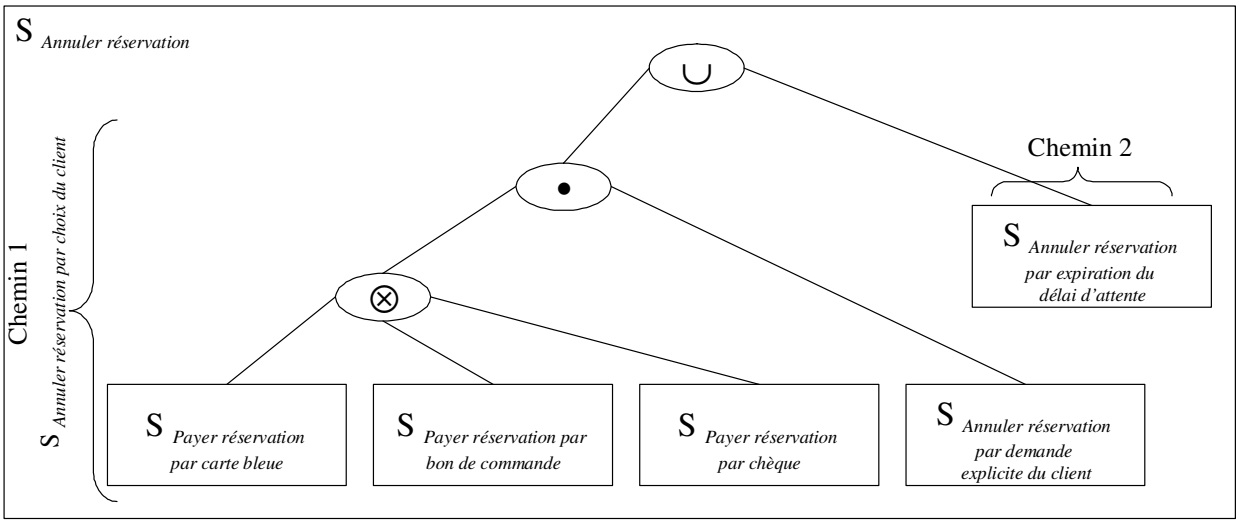


Figure 39. Exemple de représentation graphique du service S *Annuler réservation*

4. DESCRIPTION DES SERVICES

Le méta-modèle introduit à la section précédente définit les attributs descriptifs d'un service intentionnel que nous avons retenus. Il nous a semble nécessaire de compléter le méta-modèle par une structure de description textuelle des services qui soit basée sur un standard. Nous avons choisi XML et présentons dans cette section, la DTD correspondant à la structure descriptive du service intentionnel puis le modèle XML de description textuelle d'un service conforme à cette DTD. Nous donnons quelques exemples de description des services introduits dans la section précédente.

La DTD (**D**efinition **T**ype de **D**onnées) de la Figure 40 montre l'ensemble des attributs, caractérisant un service intentionnel, exprimés à l'aide de la syntaxe XML.

```
<!DOCTYPE Service [
<!ELEMENT Service_Intentionnel (Service_Atomique|Service_Agrégat)>
<!ATTLIST Service_Intentionnel code ID "REQUIRED">
<!ELEMENT Service_Atomique (Interface;Comportement;Commentaire)>
<!ELEMENT Interface (But;Situation_Initiale*;Situation_Finale*)>
<!ELEMENT But (#PCDATA)>
<!ELEMENT Situation_Initiale (#PCDATA)>
<!ELEMENT Situation_Finale (#PCDATA)>
<!ELEMENT Comportement (Pré_condition*; Post_condition* ; Règle_gestion* ;
Ressource*)>
<!ELEMENT Pré_Condition (#PCDATA)>
<!ELEMENT Post_Condition (#PCDATA)>
<!ELEMENT Règle_transition  (#PCDATA)>
<!ELEMENT Ressource(#PCDATA)>
<!ELEMENT Commentaire (#PCDATA)>
<!ELEMENT service_Agrégat (Service_Variation|Service_Composite)>
<!ELEMENT Service_Variation
(Interface;Comportement;Commentaire;Point_Variation;Service_Composant+)>

<!ENTITY Point_Variation (Alternatif|Multiple|De_chemin)>
<!ELEMENT Service_Composant EMPTY>
<!ATTLIST Service_Composant Ref IDREF "REQUIRED">
<!ELEMENT Service_Composite
(Interface;Comportement;Commentaire;Lien_Composition;Service_Composant+)>
<!ENTITY Lien_Composition (Séquentiel|Parallèle|Itératif)>
]>
```

Figure 40. DTD d'un service

La Figure 41 montre le document XML conforme à la DTD de la Figure 40 mettant en évidence les différents attributs d'un service.

```
<Service_Intentionnel Code = "Identifiant unique du service">
      <Service_Atomique>
            <Interface>
                  <But> Le but du service </But>
        <Situation_Initiale> les objets transmis en entrée
        </Situation_Initiale>
        <Situation_Finale> les objets retournés par le service
        </Situation_Finale>
            </Interface>
            <Comportement>
        <Pre_condition> Conditions de déclenchement du service
        </Pre_condition>
        <Post_condition> décrit un cas de sortie possible du service et
        caractérise les changements possibles sur les objets manipulés par le
        service </Post_condition>
```

```
        <Regle_transition > permet de préciser les contraintes régissant la
        réalisation d'opérations entraînant le changement d'états d'objets
        </Regle_transition >
        <Ressource> décrit les classes dont le service a besoin pour sa
        réalisation </Ressource>
            </Comportement>
            <Commentaire> La description du service </Commentaire>
      </Service_Atomique>
</Service_Intentionnel>
```

Figure 41. Attributs d'un service

Notation textuelle

La Figure 42 montre un exemple de description formulée en XML d'un service atomique relatif au paiement d'une réservation hôtelière par carte de crédit $S_{Payer\ réservation\ par\ carte\ de\ crédit}$.

```
<Service_Intentionnel Code = S Payer réservation par carte de crédit>
      <Service_Atomique>
            <Interface>
                  <But> Accepter le paiement </But>
        <Situation_Initiale> réservation </Situation_Initiale>
        <Situation_Finale> réservation, paiement </Situation_Finale>

            </Interface>
            <Comportement>
        <Pre_condition> réservation.état= 'valide'</Pre_condition>
        <Post_condition> réservation.état= 'payée'∧ paiement.état=
        'effectué'</Post_condition>
        <Regle_transition> réservation.état= 'valide'∧ réservation.état=
        'payée'∧ paiement.état= 'effectué'</Regle_transition>
            </Comportement>
      <Commentaire> Ce service permet d'accepter le paiement d'une
      réservation effectuée par un client </Commentaire>
      </Service_Atomique>
</Service_Intentionnel>
```

Figure 42. Exemple de description d'un service atomique

La Figure 43 décrit le service composite séquentiel correspondant $S_{Confirmer\ réservation}$ en utilisant la description formulée en XML.

```
<Service_Intentionnel code = S Confirmer réservation>
      <Service_Agrégat>
            <Service_Composite>
                  <Interface>
```

```
                        <But> Confirmer réservation </But>
                        <Situation_Initiale> demande </Situation_Initiale>
            <Situation_Finale> réservation, paiement </Situation_Finale>

                  </Interface>
                  <Comportement>
                        <Pre_condition> demande.état=
'créée'</Pre_condition>
            <Post_condition> réservation.état= 'confirmée' ∧ paiement.état=
            'efféctué' </Post_condition>
            <Regle_transition> demande.état= 'créée'∧ réservation.état=
         'confirmée' ∧ paiement.état= 'efféctué'</Regle_transition>
                  </Comportement>
         <Commentaire> ce service permet de confirmer une réservation à
         partir d'une réservation effectuée </Commentaire>

                  <Lien_Composition> Séquentiel </Lien_Composition>
                  </Service_composant Ref="S Effectuer réservation, S Payer une
réservation">
             </Service_Composite>
      </Service_agrégat>
</Service_Intentionnel>
```

Figure 43. Exemple de description d'un service composite séquentiel

La Figure 44 décrit le service à choix alternatif correspondant $S_{Payer\ une\ réservation}$ en utilisant la description formulée en XML.

```
<Service_Intentionnel code = S Payer réservation>
      <Service_Agrégat>
            <Service_Variation>
                  <Interface>
                        <But> Accepter paiement </But>
            <Situation_Initiale> réservation</Situation_Initiale>
            <Situation_Finale> réservation, paiement </Situation_Finale>

                  </Interface>
                  <Comportement>
            <Pre_condition> réservation.état= 'valide'</Pre_condition>
            <Post_condition> réservation.état= 'confirmée' ∧ paiement.état=
            'effectué' </Post_condition>
            <Regle_transition> réservation.état= 'valide'∧ réservation.état=
         'confirmée' ∧ paiement.état= 'effectué'</Regle_transition>
                  </Comportement>
         <Commentaire> ce service permet d'accepter le paiement d'une
         réservation en proposant trois manières à savoir : par carte de
```

```
        crédit, par chèque et par bon de commande</Commentaire>
                  <Point_Variation> Alternatif </Point_Variation>

        </Service_Composant Ref="S Payer une réservation par carte de crédit, S Payer
        réservation par chèque, S Payer réservation par bon de commande">
            </Service_Variation>
      </Service_Agrégat>
</Service_Intentionnel>
```

Figure 44. Exemple de description d'un service à choix alternatif

5. CONCLUSION

Le modèle $\mathcal{MiS}$ que nous avons proposé dans ce chapitre, permet la représentation des services à un niveau intentionnel. Ce modèle $\mathcal{MiS}$ 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.

Le modèle $\mathcal{MiS}$ introduit la variabilité dans la manière d'atteindre le but du service avec la notion de service à variation. Un service à variation introduit de la flexibilité dans la manière de satisfaire un but et donc de la flexibilité dans la composition de services et permet d'adapter la manière de rendre un service aux circonstances particulières de sa demande.

Enfin, le modèle $\mathcal{MiS}$ introduit la composition de services dirigée par les buts à l'échelle stratégique et intentionnelle des clients. Nous entendons par composition de services dirigée par les buts, le fait que le service composite est associé à un but de haut niveau et sa composition suit la décomposition du but père en sous buts.

CHAPITRE 4

Modèle Opérationnel de Services $\mathcal{MoS}$

1. INTRODUCTION

Nous avons introduit dans le cadre du Chapitre 3, la modélisation des services dirigée par les buts à un niveau intentionnel.

Nous proposons dans ce chapitre, une démarche pour guider l'identification, au niveau opérationnel, de services qui correspondent aux services intentionnels de type $\mathcal{MiS}$. Ceci est réalisé à l'aide la représentation explicite des services opérationnels qui découlent des services intentionnels dans les termes d'un modèle spécifique $\mathcal{MoS}$ (Modèle Opérationnel des Services) et un ensemble de règles méthodologiques pour identifier les services logiciels à partir des services intentionnels.

Le modèle $\mathcal{MiS}$ 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 logiciels.

Le regroupement de ces éléments logiciels permet de décrire *l'unité applicative opérationnelle* à développer pour supporter l'utilisateur dans la réalisation de son but. L'unité applicative opérationnelle est introduite, dans ce chapitre, par le concept de *service logiciel* dans le modèle $\mathcal{MoS}$.

D'un point de vue architectural, le service logiciel met en œuvre deux types d'éléments logiciels : les éléments d'avant plan (les applications pour l'utilisateur) et les éléments d'arrière plan (les applications centrées métier). Comme le montre la Figure 45, les éléments d'avant plan représentent les applications gérant les interactions avec l'utilisateur à l'aide des interfaces graphiques alors que les éléments d'arrière plan sont les applications métier qui sont invoquées à partir des applications utilisateur.

Dans le cadre de l'ingénierie des systèmes à base de services, les applications métier peuvent être réalisées par la réutilisation et la coordination de plusieurs services métier. Par conséquent, le modèle $\mathcal{MoS}$ introduit le concept de *service métier* pour représenter l'unité logicielle qui est fournie par une organisation et le concept de *service de coordination* pour représenter l'application métier construite par composition de services métier et qui est invoquée par le *service d'interaction utilisateur*.

De ce fait, le service logiciel du modèle $\mathcal{MoS}$ est réalisé par l'assemblage de trois types d'éléments logiciels : le service d'interface utilisateur, le service de coordination et le service métier.

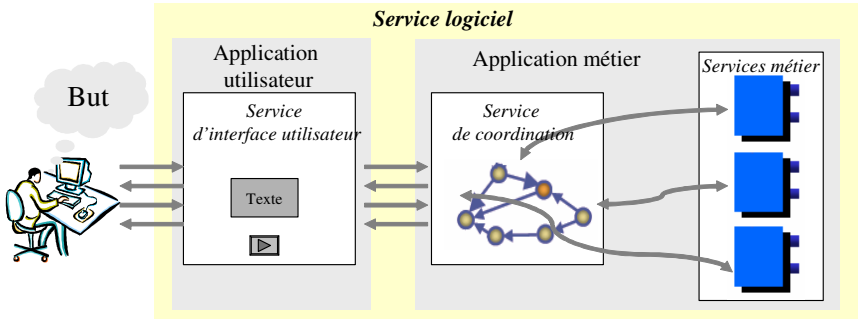


Figure 45. Les applications formant un service logiciel

Le service d'interface utilisateur est développé 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 $\mathcal{MoS}$.

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

Comme nous pouvons le remarquer, le service logiciel assemble des éléments logiciels réalisés séparément avec des techniques de développement différentes. Cette vision de l'unité applicative opérationnelle 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 la coordination de plusieurs services mais aussi pour rendre indépendante l'application d'avant plan de l'application d'arrière plan.

Le service de 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 métier réutilisé peut être différent pour un autre fournisseur.

La démarche de construction du modèle $\mathcal{MoS}$, présentée à la Figure 46, comporte trois étapes complémentaires à savoir :

(1) Ecrire le scénario de base et découvrir les exceptions,

(2) Construire le modèle $\mathcal{MoS}$ par analyse des scénarios et

(3) Identifier le modèle d'implémentation.

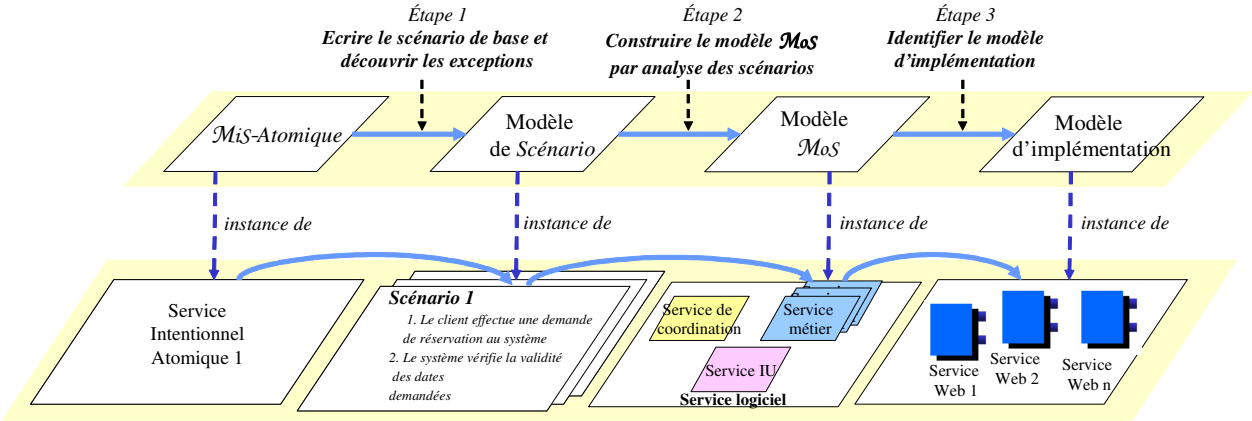


Figure 46. Démarche de construction du modèle $\mathcal{MoS}$

Lors de l'étape 1 (Ecrire le scénario de base et découvrir les exceptions), le comportement du

service atomique est décrit par un ensemble de scénarios de base et exceptionnels.

Nous proposons de réutiliser l'approche Crews L'Ecritoire 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.

Le choix de l'utilisation des scénarios est motivé par le fait qu'ils 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. On reprend la combinaison (but, scénario) utilisée par différents auteurs [Anton96b], [Holbrook90], [Potts94], [Rolland98a], [Ben Achour99].

Selon Ben Achour, un scénario est un comportement possible limité à un ensemble d'interactions entre plusieurs agents [Ben Achour99]. Le scénario comporte une ou plusieurs actions. Une combinaison d'actions dans un scénario décrit un chemin unique menant à l'état final à partir de l'état initial.

La description du comportement du service atomique, à l'aide des scénarios, obtenue à l'étape 1 sert à découvrir les services opérationnels du modèle $\mathcal{MoS}$ lors de l'étape 2 (cf. Figure 46).

L'étape 2 (Construire le modèle $\mathcal{MoS}$ par analyse des scénarios) aboutit à l'opérationnalisation des services intentionnels atomiques.

Le modèle $\mathcal{MoS}$ est une représentation des services logiciels suivant une forme canonique. Les services logiciels 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 modèle $\mathcal{MoS}$ est considéré comme une opérationnalisation du service intentionnel atomique.

Les services logiciels du modèle $\mathcal{MoS}$ sont décrits indépendamment de la plate-forme d'implémentation choisie.

L'étape 3 (Identifier le modèle d'implémentation) (cf. Figure 46), consiste à implémenter les services logiciels du modèle $\mathcal{MoS}$, c'est à dire à les traduire dans le langage du modèle d'implémentation retenu.

Grâce à cette séparation explicite des trois niveaux, la logique métier d'un service intentionnel atomique est protégée contre les changements occasionnés par l'apparition d'une nouvelle technologie. De plus, elle apporte une plus grande réactivité quand les applications doivent évoluer pour utiliser une autre technologie ou satisfaire d'autres exigences.

Le chapitre est composé de trois sections. La section 2 présente le modèle opérationnel de services $\mathcal{MoS}$. La section 3 est consacrée à la présentation des étapes de la démarche de construction du modèle $\mathcal{MoS}$. Finalement, la section 4 conclut ce chapitre.

2. PRESENTATION DU MODELE OPERATIONNEL DE SERVICE $\mathcal{MoS}$

Cette section détaille les concepts du modèle $\mathcal{MoS}$ et notamment la spécification des différents éléments constituant le service logiciel. Préalablement à cette description, nous exposons deux caractéristiques du modèle $\mathcal{MoS}$ à savoir l'interconnexion des modèles $\mathcal{MiS}$ et $\mathcal{MoS}$ (cf. section 2.1) et l'indépendance du modèle $\mathcal{MoS}$ à une plate-forme spécifique d'implémentation (cf. section 2.2).

2.1. Interconnexion des modèles $\mathcal{MiS}$ et $\mathcal{MoS}$

Les services intentionnels du modèle $\mathcal{MiS}$ offrent le moyen de satisfaire les buts de haut niveau tels que des buts stratégiques. Ceci est réalisé en décomposant le but stratégique en sous buts d'autres clients qui eux-mêmes peuvent nécessiter une décomposition etc. jusqu'à atteindre des buts opérationnalisables.

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

Les fonctionnalités de l'application à développer sont activées lorsque le client atteint un but opérationnalisable d'un service atomique. Ces fonctionnalités sont décrites à l'aide du modèle $\mathcal{MoS}$.

Le modèle $\mathcal{MoS}$ introduit un élément central : *service logiciel* qui est une unité opérationnelle assemblant des éléments logiciels de nature différente. Par conséquent, le service intentionnel atomique du modèle $\mathcal{MiS}$ est opérationnalisé par l'exécution d'un service logiciel.

Il y a donc une relation bidirectionnelle entre le modèle $\mathcal{MiS}$ et le modèle $\mathcal{MoS}$:

- Relation causale de $\mathcal{MiS}$ vers $\mathcal{MoS}$: le service intentionnel atomique du modèle $\mathcal{MiS}$ est opérationnalisé par un service logiciel du modèle $\mathcal{MoS}$.
- Relation causale de $\mathcal{MoS}$ vers $\mathcal{MiS}$: le service logiciel délivre un résultat considéré

comme l'information nécessaire pour que le but du service atomique soit réalisé.

Cette relation bidirectionnelle des modèles $\mathcal{MiS}$ et $\mathcal{MoS}$ permet de faire la transition entre la notion de service à un niveau intentionnel et celle à un niveau opérationnel. Ceci met en avant le fait que les deux notions de service co-existent et coopèrent à la fois au niveau intentionnel et au niveau opérationnel.

La deuxième caractéristique du modèle $\mathcal{MoS}$ est son indépendance à une plate-forme d'implémentation ; c'est le propos de la sous section suivante.

2.2. Indépendance du modèle $\mathcal{MoS}$ à une plate-forme spécifique d'implémentation

Le modèle $\mathcal{MoS}$ propose, à travers le concept de service logiciel, une structure en trois couches : service d'interface utilisateur, service de coordination et service métier. Chacun de ces éléments logiciels joue un rôle architectural particulier et est implémenté à l'aide de techniques spécifiques de développement.

L'objectif du modèle $\mathcal{MoS}$ est de modéliser les spécifications fonctionnelles de l'application à l'aide des différents éléments logiciels indépendamment des spécifications de son implémentation. Par analogie aux approches MDA [Bezivin02][Bezivin04], le modèle $\mathcal{MoS}$ peut être considéré comme un modèle de type PIM (Platform Independent Model) qui doit se transformer ensuite en modèle PSM (Platform Specific Model).

Nous pensons que le choix du modèle $\mathcal{MoS}$ indépendant des plates-formes technologiques est motivé par les raisons suivantes :

- Arrivée de nouvelles technologies : les applications évoluent plus rapidement qu'auparavant, pas seulement parce que les exigences du métier des applications changent mais aussi parce que les plates-formes technologiques sont en constante et rapide évolution. Par conséquent, la protection des investissements en logiciels contre l'obsolescence due à l'intégration de nouvelles plates-formes technologiques est un but important.
- La compatibilité avec les anciennes technologies : les nouvelles technologies naissent à grande vitesse mais les anciennes ne disparaissent pas. Ces technologies sont agrégées dans les systèmes du passé (legacy systems), ce qui pose un problème d'hétérogénéité.
- Le grand nombre de technologies : les applications informatiques distribuées sont généralement construites avec plusieurs technologies. De plus, les systèmes doivent

toujours communiquer avec d'autres systèmes d'une manière transparente.

2.3. Présentation des concepts du modèle MoS

Le modèle MoS met en évidence un concept central qui est celui de *service logiciel*.

La Figure 3 montre, par le jeu des trois couleurs utilisées, que la description d'un service logiciel assemble trois types d'éléments logiciels correspondant au service d'interface utilisateur (gris clair), au service de coordination métier (gris hachuré) et au service métier (gris foncé).

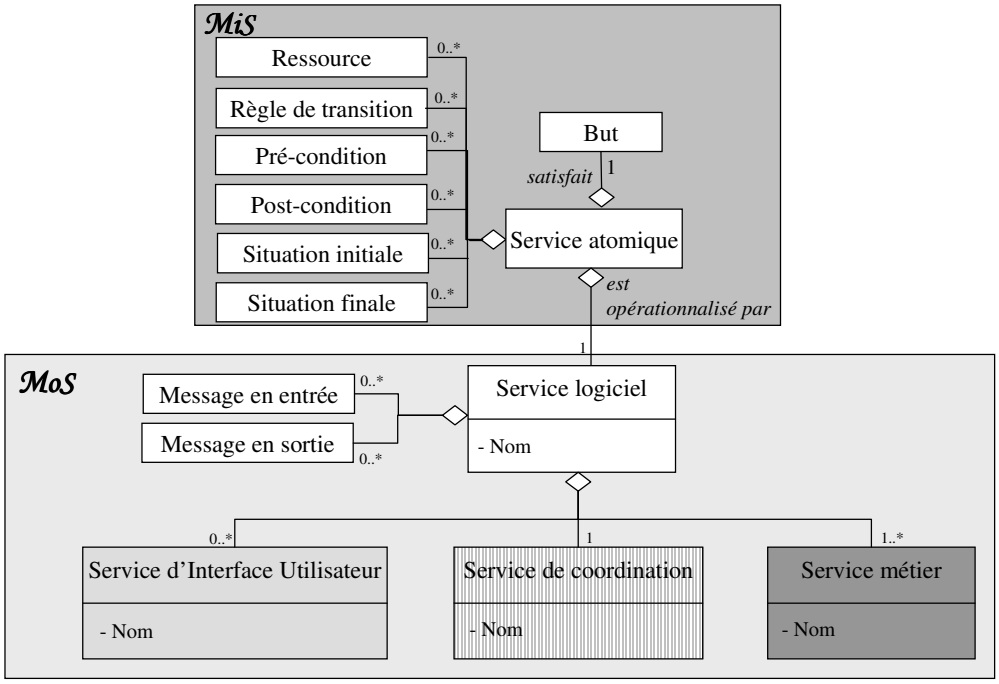


Figure 47. Le modèle MoS

Les sections suivantes présentent les différents concepts du méta-modèle MoS dans l'ordre suivant : service logiciel, service d'interface utilisateur, service métier et enfin, service de coordination.

2.3.1. Service logiciel

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

L'exécution du service logiciel permet d'interagir avec le service d'interface utilisateur, qui délègue au service de coordination la réalisation d'une transaction métier complexe et distribuée, qui à son tour, délègue aux services métier pour la réalisation de transaction 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 logiciel.

Le service logiciel est caractérisé par les éléments suivants :

- un nom permettant de l'identifier de manière unique,
- Message en entrée : est une structure de données passée en entrée pour adapter le comportement du service logiciel au contexte d'utilisation,
- Message en sortie : est 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 logiciel de nom $SL_{Effectuer\ Réservation}$ avec le *nom du demandeur* comme message d'entrée et la *réservation de vol & chambres payée* comme message de sortie.

2.3.2. Service d'interface utilisateur

Le 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 logiciel, 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 récupérer 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 48[7] est un extrait de la Figure 47 montrant les éléments d'un service d'interface utilisateur.

[7] Les éléments coloriés en gris hachuré ne font pas partie du modèle de service d'interface utilisateur mais plutôt du modèle du service de coordination. Nous avons choisi de les ajouter afin d'éclaircir la relation des interactions métier avec le service de coordination.

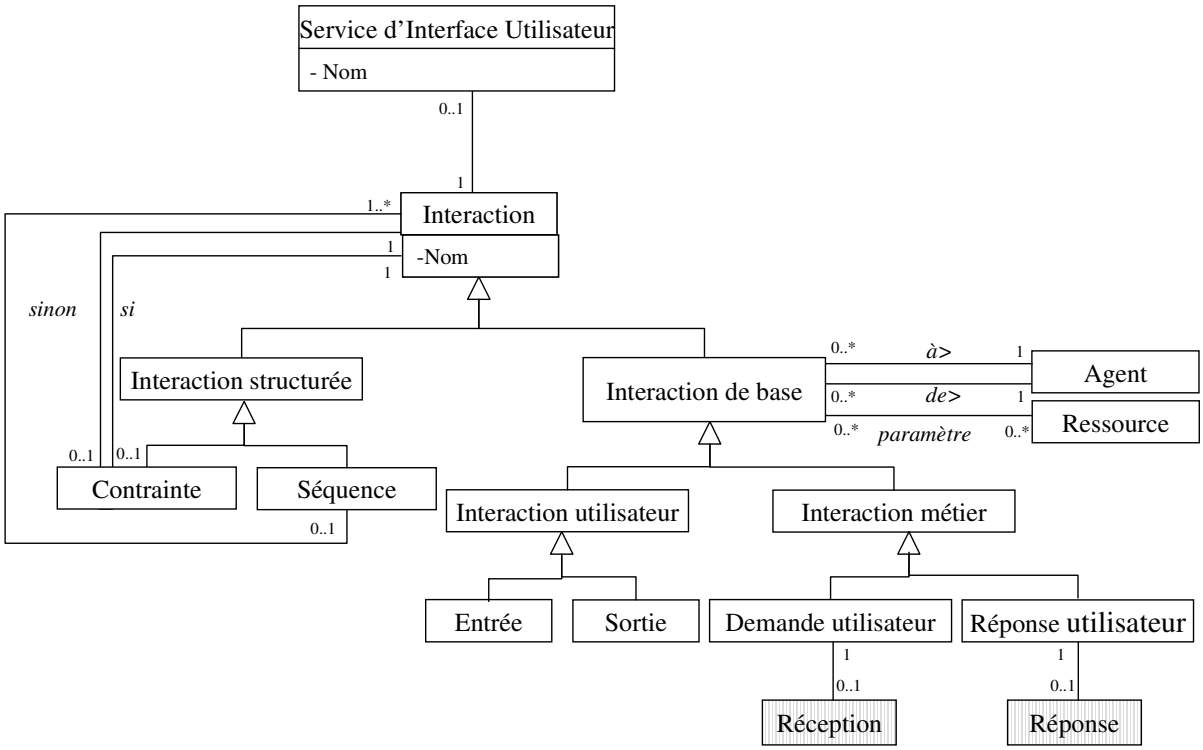


Figure 48. Eléments d'un service d'interface utilisateur

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* alors qu'une interaction structurée permet de modéliser l'enchaînement des interactions de base nécessaire pour formuler le dialogue ou la conversation-type que l'utilisateur doit avoir avec le service d'interface utilisateur.

Une interaction de base est spécialisée en deux sous types : *interaction utilisateur* et *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 logiciel au service de coordination. 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 qui correspond au fait que l'interface envoie une demande de réservation de vol et de chambres au coordonnateur 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 de coordination, les relations qui existent entre les interactions métier du service d'interface et le service de coordination.

Le modèle de service d'interface utilisateur permet donc de décrire le contenu de l'application web en termes d'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. A partir de là, une conception graphique des pages web peut 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.
- Enfin la navigation web entre les pages web et les composants web est décrite à l'aide de l'ordonnancement des interactions utilisateur et métier. Ceci permet de déterminer quel composant web est invoqué à partir d'une page web et quelle page web est affichée à partir de l'exécution du composant web.

2.3.3. Service métier

Le service métier décrit les transactions métier. La spécification ne décrit pas la manière dont les transactions métier doivent être implémentées mais elle s'attache à décrire comment elles sont utilisées indépendamment de leur implémentation. Le service métier est, par conséquent, décrit par son interface.

Notre définition de service métier s'est largement inspirée des services web et considère l'interface d'un service métier comme l'agrégation d'un ensemble d'opérations nécessaires à la délivrance d'un service spécifique.

Dans le cadre du service logiciel, plusieurs services métier peuvent être réutilisés pour pouvoir délivrer le service associé au service logiciel à son utilisateur bénéficiaire.

La Figure 49 est un extrait du méta-modèle $\mathcal{M}_{OS}$ qui montre les différents éléments composant le service métier.

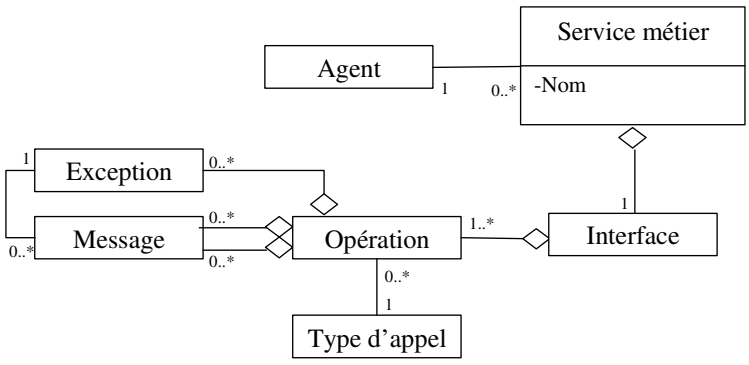


Figure 49. Structure d'un service métier

Dans ce qui suit, nous désignons par *service métier* ces opérations.

Par exemple, en prenant l'exemple du service métier *Effectuer une réservation*, l'objectif est

la création de la réservation qui correspond à l'exécution de l'opération *Effectuer-Reservation ().*

Une *opération* décrit les *types d'appel* de manière abstraite. Un type d'appel peut prendre l'une des valeurs suivantes : Unidirectionnel, Demande/Réponse, et Notification.

Une opération dans un service métier envoie et reçoit des *messages*. Un *message* représente les données en entrée (respectivement en sortie) fournies (respectivement produites) par le service métier. Le message peut être de type *exception*.

Par exemple, l'opération *Effectuer-Reservation ()* requiert en entrée le détail du client ainsi que le détail de la réservation. Le résultat fourni par cette opération est un message comportant la date de la réservation et le montant total à payer.

Un service métier est fourni par un *agent* et est décrit à l'aide d'une *interface*. C'est la partie visible qui permet de comprendre la capacité du service sans entrer dans le corps de celui-ci.

2.3.4. Service de coordination

Le service de coordination permet, d'une part, de coordonner l'ensemble des services métier à travers un ensemble d'activités ordonnées et, d'autre part, de rendre le service d'interface utilisateur indépendant des services métier mis en oeuvre. Ceci permet de développer des applications d'avant plan indépendamment des applications métier.

Un service de coordination repose sur la définition formelle 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.

Un service de coordination 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 métier concerné. Ce dernier renvoie son résultat au service de coordination.

Le rôle d'intermédiaire dans la communication attribue au service de coordination la fonction de synchronisation et de coordination qui sont nécessaires entre les services.

Le service de coordination permet donc 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.

La Figure 50 est un extrait du méta-modèle $\mathcal{MoS}$ et montre les différents éléments composant le service de coordination.

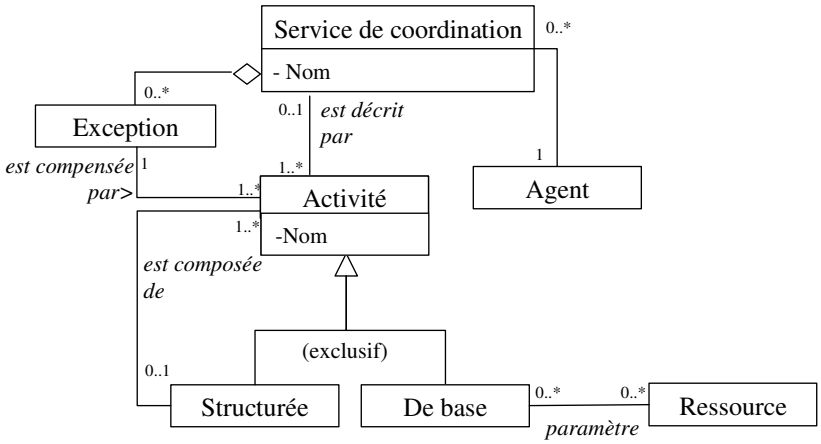


Figure 50. Structure d'un service de coordination

La structure d'un service de coordination est relativement complexe par rapport au service métier [Alonso04].

Nous présentons les différents concepts formant un service de coordination dans les sections suivantes.

2.3.4.1. La notion d'activité

Un service de coordination 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 métier. Les activités structurées sont organisées hiérarchiquement.

Dans le modèle du service de coordination, les activités structurées sont les activités complexes alors que les activités de base sont les activités atomiques. La Figure 50 montre que ces deux notions sont généralisées par la notion d'activité. Cette généralisation est exclusive (la contrainte 'Exclusif' entre les deux liens de spécialisation).

D'un autre côté, une activité fait partie d'une description de service de coordination ou d'une activité structurée. Par conséquent, les activités ne sont pas partagées entre plusieurs services de coordination ni même entre plusieurs activités structurées au sein d'un même service de coordination (d'où la cardinalité 0..1 de *Activité* vers *Service de coordination*).

L'association *est composée de* entre *Activité structurée* et *Activité* à la Figure 50 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 de coordination* et *Activité*, montre qu'un service de coordination est décrit par un ensemble d'activités dans le cas normal (la cardinalité 1..*).

Les deux sous sections suivantes détaillent la notion d'activité de base et la notions d'activité structurée.

2.3.4.2. 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.

La Figure 51[8] présente les différents types d'activités de base.

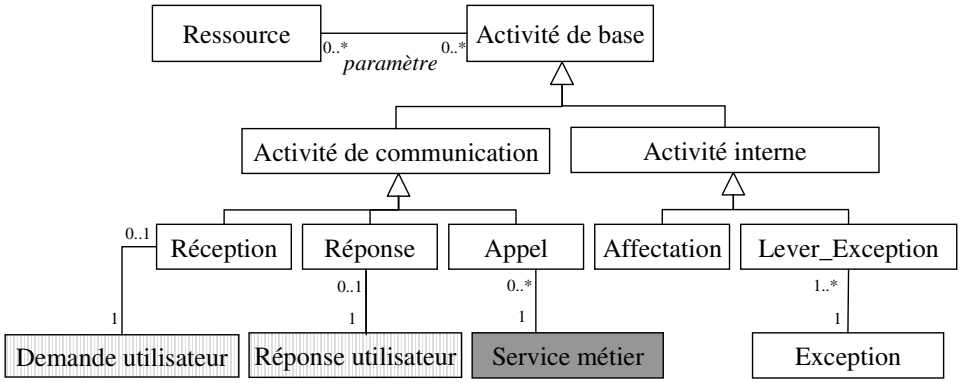


Figure 51. Les types des activités de base

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 : *appel, réception* et *réponse*. Un *appel* correspond à l'invocation d'un service métier, 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* est spécialisée en : *affectation* et *lever_exception*.

Une *affectation* permet de copier les données d'un message dans un autre en introduisant de nouvelles données. Ceci sert à adapter les données échangées entre le service d'interface utilisateur et les services métier. Cette adaptation de données est nécessaire pour rendre le service d'interface utilisateur indépendant des services métier réutilisés.

L'activité *lever_exception* permet de lever une exception lorsqu'une erreur est détectée. Cette activité interrompt l'exécution de la procédure de coordination et est reprise par l'activité de compensation associée à l'exception levée.

La Figure 51 montre aussi que chaque activité de base peut manipuler une ou plusieurs *ressources*.

[8] Les éléments coloriés en gris hachuré ne font pas partie du modèle de service d'interface utilisateur mais plutôt du modèle du service de coordination. Pareillement, l'élément colorié en gris foncé qui fait partie du modèle du service métier. Nous avons choisi de les ajouter afin d'éclaircir la relation des activités de communication avec le service de coordination d'une part et le service métier d'autre part.

Par exemple dans l'activité de base

> *La transformation du message « demande de réservation de vol et d'hôtel » en le message « demande de vol »*

les ressources sont *demande de réservation de vol* et *de chambres d'hôtel.*

2.3.4.3. 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 52 et sont détaillés dans les sections suivantes.

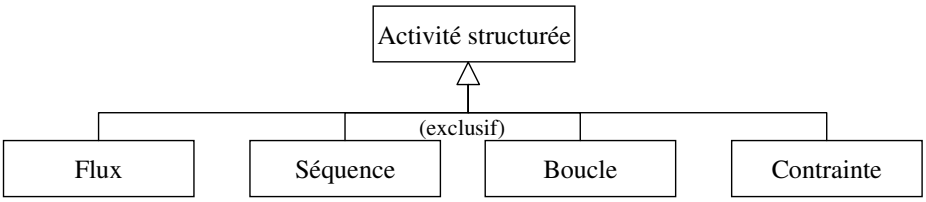


Figure 52. La notion d'activité structurée

2.3.4.3.1.La séquence

Une séquence est composée de deux activités, la deuxième activité se déroulant après la première. L'activité structurée suivante :

'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 ». Appel du service « 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 l'appel du service « 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.

2.3.4.3.2.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 pas 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:

Appel du service « 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 ; en réalité, elles ne commencent ni 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 ; qui peuvent être des activités structurées.

2.3.4.3.3.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'hôtel'*. 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 de coordination décrit le comportement correspondant au cas où la période serait valide et celle où elle ne le serait pas.

2.3.4.3.4.La boucle

La boucle permet d'exprimer une répétition d'activités structurées. Lorsque l'agence de voyage a un partenariat avec plusieurs compagnies aériennes, l'appel du service de recherche 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 où la boucle est composée de trois activités de base:

(a) appel du service « 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 ».

2.3.4.4. la notion d'exception

Une exception, dans un service de coordination, est levée par l'activité de base de type *lever_exception* dès qu'une erreur est constatée. Une erreur peut être détectée suite à l'évaluation d'une condition dans le cadre d'une activité de type contrainte. Si la contrainte n'est pas vérifiée, une exception doit être levée par une activité de type *lever_exception*.

Le service de coordination définit la liste des exceptions identifiées et qu'il peut traiter à l'aide d'une activité de compensation. L'activité de compensation représente le flux d'activités qu'il faut exécuter pour traiter ou compenser l'exception identifiée.

Lever une exception a comme conséquence d'interrompre l'exécution de la procédure de coordination. L'exception est prise en compte par le service de coordination et l'exécution de la procédure de coordination est reprise par l'exécution de l'activité de compensation associée à l'exception levée. Par conséquent, on peut dire que la suite de la procédure de coordination est remplacée par l'activité de compensation.

Ce mécanisme d'exception est utilisé dans le service de coordination pour traiter le problème où les services métier invoqués ne délivrent pas le résultat attendu. A ce moment, la non délivrance d'un des services métier a un impact sur le service de coordination qui se traduit par la levée d'une exception et l'exécution d'une activité de compensation.

Aucune disponibilité n'a été trouvée est un cas d'exception où le service n'est pas satisfait et une activité de compensation en proposant d'autres dates de réservation peuvent être suggérées au client.

3. LA DEMARCHE DE CONSTRUCTION DU MODELE $\mathcal{MoS}$

La démarche de construction du modèle $\mathcal{MoS}$ comporte trois étapes :

- Etape 1 : Ecrire le scénario de base et découvrir les exceptions,
- Etape 2 : Construire le modèle $\mathcal{MoS}$ par analyse des scénarios et,

– Etape 3 : Identifier le modèle d'implémentation.

Les sections suivantes sont consacrées à la description de chacune de ces trois étapes.

3.1. Etape 1 : Ecrire le scénario de base et découvrir les exceptions

Nous introduisons dans cette section le modèle de scénario, présenté comme outil méthodologique permettant de décrire le comportement attendu du service intentionnel atomique menant à la réalisation de son but. La réalisation d'un but peut se faire à l'aide d'un ou plusieurs scénarios.

L'hypothèse de base de notre approche est que la relation entre le but d'un service intentionnel atomique et les scénarios doit être bidirectionnelle : un but sert à écrire un ou plusieurs scénarios. Ces scénarios peuvent être (i) normaux dans le cas où la satisfaction du but serait réalisée avec succès et (ii) exceptionnels dans le cas où la satisfaction du but serait un échec [Cockburn95] [Cockburn00].

Les scénarios obtenus sont utilisés pour découvrir les éléments qui forment le modèle $\mathcal{MoS}$ à savoir : le service d'interface utilisateur, le service de coordination et le service métier.

3.1.1. Pourquoi les scénarios ?

Historiquement, les approches dirigées par les besoins, telles que l'analyse structurée et les techniques de conception, se concentrent sur les modèles et les langages pour la conception des fonctions et des structures des systèmes [Chen76] [Smith77] [Hull87] [Rolland92]. Cependant, alors que les applications augmentèrent en taille et en complexité, d'autres approches furent nécessaires. Les centres d'intérêt se déplacèrent vers les composants de l'application et les interfaces (entre objets, mais aussi entre homme et machine). Cette progression historique amena au paradigme centré utilisateur [Lubars93].

Les utilisateurs dépendent plus que jamais des applications pour réaliser leurs tâches. Pour cette raison, les utilisateurs doivent participer au développement des applications. Les approches dirigées par les besoins centrées utilisateur cherchent à extraire directement des utilisateurs la connaissance de l'environnement de travail, des tâches, de l'organisation, etc. Cependant les utilisateurs n'ayant pas les compétences requises pour exprimer formellement leurs besoins, ils ne sont pas à même de manipuler les concepts tels que: les buts, les fonctions, les objets, etc. Il est donc nécessaire de prendre le point de vue de l'utilisateur selon Carroll [Carroll95] : 'N*ous avons besoin de développer de nouveaux vocabulaires pour permettre de justifier et caractériser la conception en terme des activités que les utilisateurs projettent d'accomplir en utilisant le système'*.

La notion de scénario a été introduite dans les approches dirigées par les besoins pour fournir un moyen de description des perspectives des 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 [Tawbi01], 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.

3.1.2. Présentation du modèle de scénario

La Figure 53 présente le modèle de scénario retenu en utilisant les notations du diagramme de classes UML. Les concepts sont détaillés dans les sections suivantes.

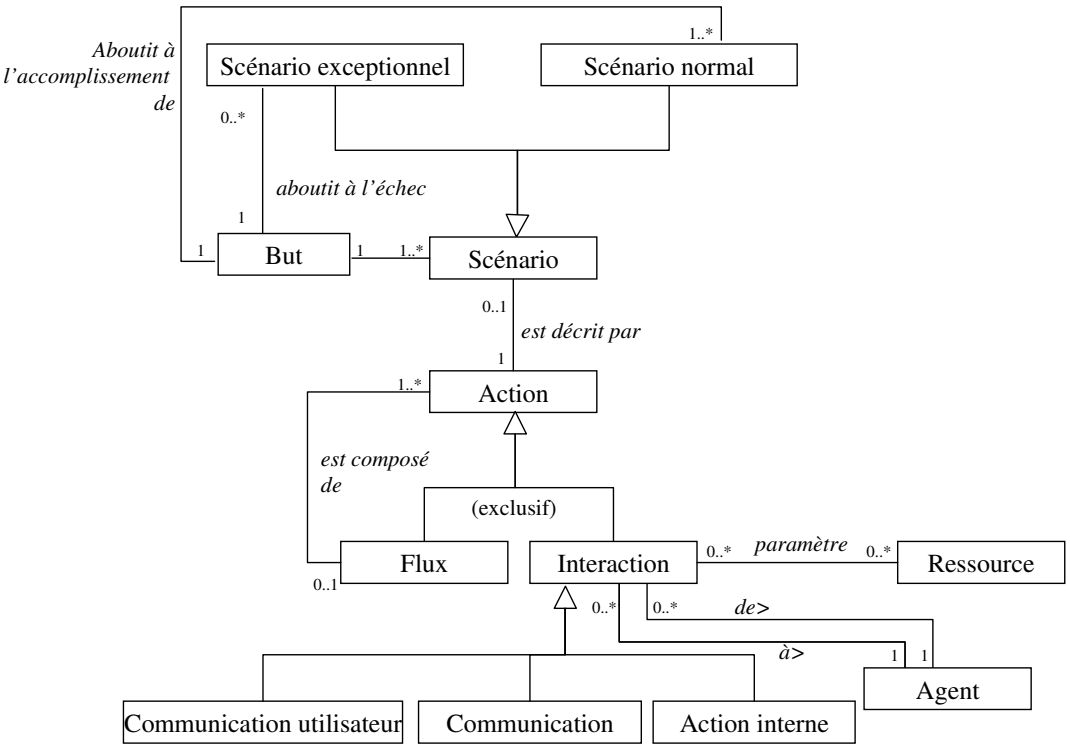


Figure 53. Modèle de scénario

3.1.3. Scénario

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 Achour99] et étendue dans [Tawbi01].

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 [Rolland98a]. Tout scénario encapsule de manière non formelle une partie des connaissances que le système a du système.

La Figure 53 met l'accent sur deux notions :

- La notion d'interaction et,
- La notion de flux.

Dans le modèle de scénario, les flux sont les actions complexes alors que les interactions sont les actions atomiques. La Figure 54 montre que ces deux notions sont généralisées par la notion d'action. Cette généralisation est exclusive (la contrainte 'Exclusif' entre les deux liens de spécialisation).

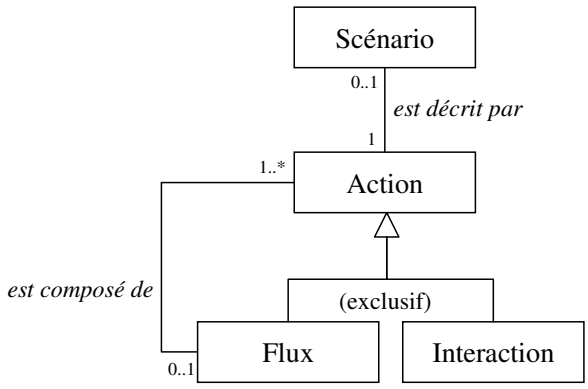


Figure 54. La notion d'action dans le scénario

Le lien *est composé de* entre *Flux* et *Action* à la Figure 54 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 *Scénario* et *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.

3.1.3.1. La notion d'interaction

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.

La Figure 55 présente la notion d'interaction.

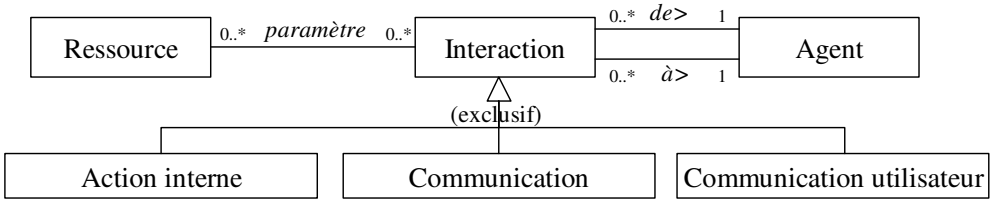


Figure 55. Les paramètres d'une interaction

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 souhaitée*',

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 55 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 une machine 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 55, 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 et la *demande de réservation* représente la ressource échangée.

3.1.3.2. 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 56 et sont détaillés dans les sections suivantes.

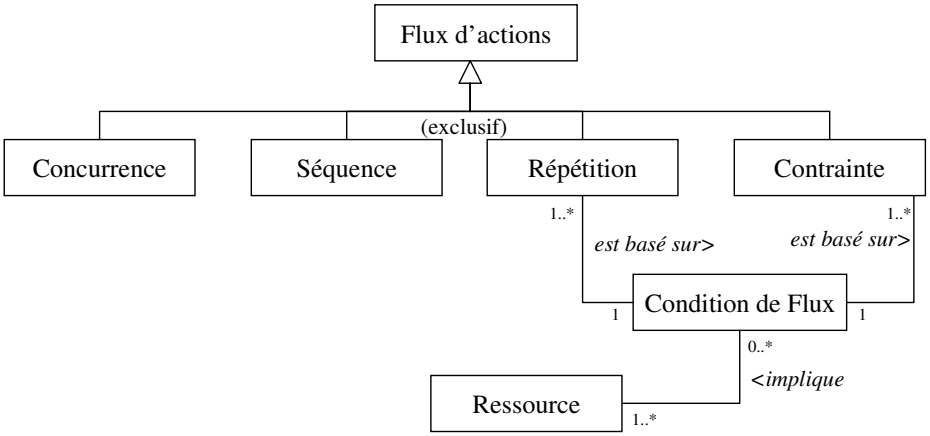


Figure 56. La notion de flux d'interactions

3.1.3.2.1.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 (Figure 56), 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.

3.1.3.2.2.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 pas et ne se terminent pas au même moment. Cependant, du point de vue utilisateur, elles se déroulent au même moment et donc, selon la définition d'un scénario, elles sont concurrentes.

Une concurrence est composée de deux actions, et donc, de la même manière que 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*. Dans les deux sous-sections suivantes, nous détaillons ces deux notions.

3.1.3.2.3.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 à la Figure 56 par l'entité *condition de flux* qui est liée aux deux entités *contrainte* et *répétition.*

La Figure 56 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'.

3.1.3.2.4.La contrainte

Une contrainte décrit les conditions nécessaires au déroulement de l'ensemble des actions qui

suivent la contrainte. 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 de scénario. 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.

3.1.3.2.5.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 que 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:

'l'application affiche un message confirmant le paiement de la réservation'.

3.1.3.3. Les scénarios normaux et les scénarios exceptionnels

Nous distinguons deux types de scénarios, les normaux et les exceptionnels (cf. Figure 57).

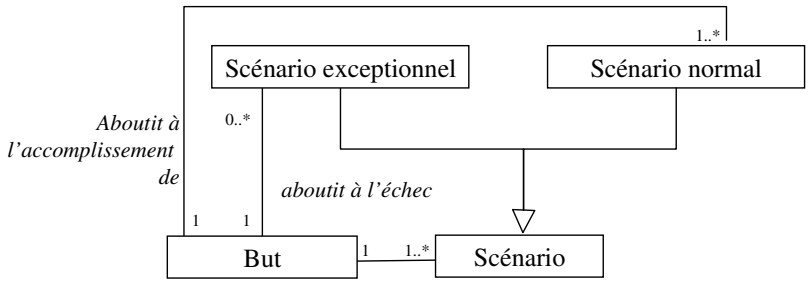


Figure 57. La notion des scénarios normaux et exceptionnels

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 comment '*obtenir une réservation de chambres d'hôtel'* de trois manières différentes :

'dans le cas normal', et

'*dans le cas d'une erreur de période de validité'*

'dans le cas de non disponibilité'.

Le premier scénario est de type normal puisqu'il permet au client d'obtenir une réservation. Alors que le deuxième et troisième cas sont exceptionnels puisqu'ils ne permettent pas de délivrer une réservation au client. En effet, '*dans le cas de non disponibilité'* correspond au cas où l'hôtel n'aurait plus de chambres libres pour la période demandée alors que 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.

3.1.4. 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 [Tawbi01] utilisée comme un guidage méthodologique dans la construction des scénarios en langage naturel.

La construction d'un scénario dans l'approche Crews-L'Ecritoire 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.

3.1.4.1. 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.

3.1.4.1.1. Ecriture d'un scénario

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.

3.1.4.1.2. Les directives de style

Les directives de style (DS) présentées à la Figure 58 sont conçues pour le méta-modèle de scénario présenté à la section 3.1.2.

Les directives de style

DS1: Chaque action atomique doit être déclarée par une clause composée d'un verbe et des 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, à la location, 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 58. Les directives de style

La *DS1* aide à écrire les actions atomiques 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 location 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 écrites.

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.

3.1.4.1.3.Les directives de contenu

Les directives de contenu (DC) sont présentées à la Figure 59.

Les directives de contenu
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 **DS6.**

Figure 59. 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.

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

3.1.4.1.4.Exemple d'application

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

On se base sur les directives de style présentées à la Figure 58 et celles de contenu présentées à la Figure 59 pour écrire les flux d'actions suivant :

Expression textuelle du scénario $S_{Effectuer\ une\ réservation}$
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 60. Le scénario 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.

[Tawbi01] 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 [Tawbi01] 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 61 illustre le scénario S *Effectuer une réservation* de la Figure 60 après avoir été conceptualisé et complété. Des activités de conceptualisation et de complétude, telles que la suppréssion des références anaphoriques ou l'ajout des agents source et cible, ont été appliquées.

Expression textuelle du scénario S ***Effectuer une réservation***

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 61. Le scénario après avoir été complété

3.1.4.2. Recherche d'exceptions

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 4 nouveaux scénarios synthétisés par le tableau suivant (cf. Tableau 3) :

Tableau 3. Liste des scénarios alternatifs

Scénarios	Type	Manière	Résumé
SC2	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
SC3	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
SC4	Echec	Aucune proposition sélectionnée	L'agence de voyage suggère au client de reformuler une nouvelle demande avec une période ou une

			destination différente
SC5	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 62) 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 61. 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.

Expression textuelle du scénario SC2

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 62. Le scénario SC2

3.2. Etape 2 : Construire le modèle $\mathcal{MoS}$ par analyse des scénarios

L'objectif de cette étape est de découvrir les éléments constituant le service logiciel du modèle $\mathcal{MoS}$ à partir des scénarios obtenus (scénario de base et scénarios alternatifs de succès ou d'échecs) lors de l'étape 1 (Ecrire le scénario de base et découvrir les exceptions).

La construction du modèle $\mathcal{MoS}$ par analyse des scénarios est réalisée en procédant par un ensemble de sous étapes. La structure du modèle $\mathcal{MoS}$ générée est conforme à la structure présentée à la section 2.3 (Présentation des concepts du modèle $\mathcal{MoS}$).

La génération du modèle $\mathcal{MoS}$, à partir des scénarios, est réalisée par application des trois sous étapes suivantes :

- Etape 2.1 : Construire le modèle $\mathcal{MoS}$ par analyse du scénario de base,
- Etape 2.2 : Construire le modèle $\mathcal{MoS}$ par analyse des scénarios alternatifs de succès et,
- Etape 2.3 : Construire le modèle $\mathcal{MoS}$ par analyse des scénarios alternatifs d'échec.

qui sont décrites dans les sections suivantes.

Nous illustrerons par la suite l'analyse du scénario de base afin de montrer l'application des règles de construction de l'étape 2.1 et nous alalyserons le scénario d'exception : pas de disponibilité de vol pour montrer les règles de construction de l'étape 2.2 et 2.3. En effet, l'étape 2.3 inclut les règles de construction de l'étape 2.2.

3.2.1. Etape 2.1 : Construire le modèle $\mathcal{MoS}$ par analyse du scénario de base

La génération du modèle $\mathcal{MoS}$, à partir du scénario de base, est réalisée par application des quatre sous étapes suivantes :

- Etape 2.1.1 : Spécialiser les agents du scénario,
- Etape 2.1.2 : Modéliser le service d'interface utilisateur,
- Etape 2.1.3 : Modéliser les services métier et,
- Etape 2.1.4 : Modéliser le service de coordination.

qui sont décrites dans les sections suivantes.

3.2.1.1. Etape 2.1.1 : Spécialiser les agents du scénario

Un agent est décrit comme une entité autonome communicante qui joue des rôles. Dans un service logiciel, 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.

Un agent 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 en :

- *Demandeur* : l'agent utilisateur qui initie la demande d'information ou de service et qui bénéficie du service
- *Interface* : l'agent système qui gère les interactions entre le demandeur et le coordonnateur.
- *Coordonnateur* : agent système qui contrôle tous les échanges entre agents et gère l'ensemble des données partagées.
- *Fournisseur* : propriétaire de l'information et/ou du service demandé.

L'agent Demandeur dépend de l'Interface et du Coordonnateur pour la demande de service

ainsi que pour la demande d'un ensemble de données partagées. L'agent Coordonnateur de son coté dépend du Fournisseur 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 dépend du Coordonnateur 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 les services d'interface utilisateur, de coordination et métier du modèle $\mathcal{MoS}$.

3.2.1.1.1.Exemple d'application

L'application de l'étape 2.1.1 (spécialiser les agents du service atomique) du processus d'identification des services logiciels au scénario de la Figure 61 aboutit ainsi :

- Le client : correspond au rôle de *Demandeur* du service
- L'agence de voyage : est découpé en deux rôles systèmes celui de l'*Interface* et celui de *Coordonnateur*.
- L'hôtel, la compagnie aérienne et la banque : sont les agents ayant le rôle de *Fournisseur* du service.

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 de (i) la compagnie aérienne et de l'hôtel pour recevoir les possibilités de vols et de réservation de chambres (ii) client pour le choix du vol et la chambre qui lui conviennent et pour la récupération des informations de paiement et (iii) la banque pour effectuer la transaction bancaire.

3.2.1.2. Etape 2.1.2 : Modéliser le service d'interface utilisateur

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

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

Les étapes de découverte du service d'interface utilisateur

DSU1: Extraire les communications utilisateur dans leur ordre d'apparition dans le scénario de base.

DSU2 : Identifier, à partir des communications utilisateur, les agents de rôle demandeur, interface et coordonnateur

DSU3 : Identifier, à partir des communications utilisateur du scénario, les interactions de base

DSU4: Compléter chaque interaction de base par l'identification des ressources échangées.

DSU5 : Identifier, à partir des actions de type contrainte du scénario, les interactions structurées de type contrainte afin de compléter la description du service d'interface utilisateur.

Figure 63. Les étapes de découverte du service d'interface utilisateur

- L'étape de découverte des services d'interface utilisateur (DSU1) consiste d'abord 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.

Par exemple, l'application de la *DSU1* au scénario $S_{Effectuer\ une\ réservation}$, distingue les six actions suivantes numérotées 1, 8, 9, 11, 12 et 19 :

> *1. Le client formule une demande de réservation de vol et de chambres d'hôtel à partir de l'agence de voyage.*
> *8. L'agence de voyage notifie les propositions de vol, de chambres et le montant à payer au client.*
> *9. Le client sélectionne l'une des propositions pour l'agence de voyage.*
> *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.*
> *19. L'agence de voyage confirme la réservation au client.*

Les interactions 1, 9 et 12 sont des communications utilisateur qui ont comme agent source le client dont le le rôle de Demandeur. Dans ce cas, on considère que ces communications sont de type entrée. Par contre, les communications 8, 11 et 19 sont de type sortie.

- La DSU2 permet d'identifier, à partir des coomunications utilisateur, les trois rôles (Demandeur, Interface et Coordonnateur).

En appliquant la DSU2 aux communications utilisateur du scénario $S_{Effectuer\ une\ réservation}$, 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 Coordonnateur.

- L'étape 3 (DSU3) 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.

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 à son tour de réagir.

Une fois cette condition vérifiée, les communications utilisateur du scénario sont opérationnalisées par les deux principes suivants :

- Une communication utilisateur de type entrée est opérationnalisée dans le service d'interface utilisateur de $\mathcal{MOS}$ par :
  - une interaction utilisateur de type entrée entre le demandeur et l'agent Interface,
  - une interaction métier de type demande utilisateur entre les agents Interface et Coordonnateur et,
  - une interaction métier de type réponse utilisateur entre les agents Coordonnateur et Interface.
- Une communication utilisateur de type sortie est opérationnalisée dans le service d'interface utilisateur par une interaction utilisateur de type sortie.

L'application de la DSU3 aux six communications utilisateur permet d'identifier les interactions de base suivantes (cf. Tableau 4) :

Tableau 4. Liste des interactions de base

N°	Interaction de base	Agent source	Agent cible	Type	Sens
1	FormuleDemandeRéservationVol &Chambres	client	Agence (Interface)	utilisateur	entrée
	EnvoieDemandeRéservationVol& Hotel	Agence (Interface)	Agence (Coordonnateur)	métier	demande utilisateur
	ReçoitListeProposition	Agence (Coordonnateur)	Agence (Interface)	métier	réponse utilisateur
8	NotifiePropositions	Agence (Interface)	client	utilisateur	sortie

9	SélectionProposition	client	Agence (Interface)	utilisateur	entrée
	EnvoiePropositionSelectionnée	Agence (Interface)	Agence (Coordonnateur)	métier	demande utilisateur
	RéçoitAccuséReception	Agence (Coordonnateur)	Agence (Interface)	métier	réponse utilisateur
11	AfficheInformationsPaiement	Agence (Interface)	client	utilisateur	sortie
12	ConfirmePaiementEnLigne	client	Agence (Interface)	utilisateur	entrée
	EnvoiePaiementEnLigne	Agence (Interface)	Agence (Coordonnateur)	métier	demande utilisateur
	ReçoitConfirmationRéservationPayéeVol&Chambres	Agence (Coordonnateur)	Agence (Interface)	métier	réponse utilisateur
19	ConfirmeRéservationVol&Chambres	Agence (Interface)	client	utilisateur	sortie

- La DSU4 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 fournie à l'utilisateur par l'interaction utilisateur de sortie.

En appliquant la *DSU4* au scénario $S_{\textit{Effectuer une réservation}}$, on obtient la liste des ressources présentées au Tableau 32.

Tableau 5. Liste des ressources

Communication utilisateur	Ressource
1. Le client formule une demande de réservation de vol et de	Demande_Reservation_Vol&Chambres

chambres d'hôtel à partir de l'agence de voyage.	
8. L'agence de voyage notifie les propositions de vol, de chambres et le montant à payer au client	Liste (Proposition_Vol) Liste (Proposition_Chambre)
9. Le client sélectionne l'une des propositions pour l'agence de voyage.	Proposition_Vol_Sélectionnée Poposition_Chambre_Sélectionnée MontantàPayer
11. L'agence de voyage affiche le formulaire de saisie pour compléter la réservation et effectuer le paiement en ligne au client.	Formulaire_InfoPerso&Paiement
12. Le client confirme en envoyant ses informations personnelles et ses données bancaires à l'agence de voyage	InfoPerso&Paiement
19. L'agence de voyage confirme la réservation au client	Confirmation_Reservation

A ce stade, il est possible d'identifier (a) que l'interface comporte une interaction structurée de type séquence et (b) les interactions de base qui la composent.

Le Tableau 6 présente les valeurs requises par le modèle $\mathcal{MoS}$ pour chacune des interactions de base du service d'interface utilisateur.

Tableau 6. Valeurs requises par les interactions de base

N°	Interaction de base	Type	Sens	Agent source	Agent cible	Ressources
1	FormuleDemandeRéservationVol&Chambres	utilisateur	entrée	client	Agence (Interface)	Demande_Reservation_Vol&Chambres
	EnvoieDemandeRéservationVol&Hotel	métier	requête	Agence (Interface)	Agence (Coordonnateur)	Demande_Reservation_Vol&Chambres
	ReçoitListeProposition	métier	réponse	Agence (Coordonnateur)	Agence (Interface)	Liste (Proposition_Vol) Liste (Proposition_Chambre)
8	NotifiePropositions	utilisateur	sortie	Agence (Interface)	client	Liste (Proposition_Vol) Liste (Proposition_Chambre)

9	SélectionProposition	utilisateur	entrée	client	Agence (Interface)	Proposition_Vol_Sélectionnée Poposition_Chambre_Sélectionnée MontantàPayer
	EnvoiePropositionSelectionnée	métier	requête	Agence (Interface)	Agence (Coordonnateur)	Proposition_Vol_Sélectionnée Poposition_Chambre_Sélectionnée MontantàPayer
	RéçoitAccuséReception	métier	réponse	Agence (Coordonnateur)	Agence (Interface)	
11	AfficheInformationsPaiement	utilisateur	sortie	Agence (Interface)	client	Formulaire_InfoPerso&Paiement
12	ConfirmePaiementEnLigne	utilisateur	entrée	client	Agence (Interface)	InfoPerso&Paiement
	EnvoiePaiementEnLigne	métier	requête	Agence (Interface)	Agence (Coordonnateur)	InfoPerso&Paiement
	ReçoitConfirmationRéservationPayéeVol&Chambres	métier	réponse	Agence (Coordonnateur)	Agence (Interface)	Confirmation_Reservation
19	ConfirmeRéservationVol&Chambres	utilisateur	sortie	Agence (Interface)	client	Confirmation_Reservation

- La règle DSU5 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 $\mathcal{MoS}$ (cf. Figure 47).

Cette règle 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. Le tableau suivant dispose les contraintes par rapport aux interactions de base.

Tableau 7. Positionnement des contraintes par rapport aux interactions de base

N°	Nom de l'interaction	Type Interaction	Type	Sens	Agent source	Agent cible
1	FormuleDemandeRéservationVol&Chambres	De base	utilisateur	entrée	client	Agence (Interface)
	EnvoieDemandeRéservationVol&Hotel	De base	métier	requête	Agence (Interface)	Agence (Coordonnateur)
	Contrainte (C1 and C2 and C3)	Contrainte				
	ReçoitListeProposition	De base	métier	réponse	Agence (Coordonnateur)	Agence (Interface)
8	NotifiePropositions	De base	utilisateur	sortie	Agence (Interface)	client
9	SélectionProposition	De base	utilisateur	entrée	client	Agence (Interface)
	Contrainte C4	Contrainte				
	EnvoiePropositionSelectionnée	De base	métier	requête	Agence (Interface)	Agence (Coordonnateur)
	RéçoitAccuséReception	De base	métier	réponse	Agence (Coordonnateur)	Agence (Interface)
11	AfficheInformationsPaiement	De base	utilisateur	sortie	Agence (Interface)	client
12	ConfirmePaiementEnLigne	De base	utilisateur	entrée	client	Agence (Interface)
	EnvoiePaiementEnLigne	De base	métier	requête	Agence (Interface)	Agence (Coordonnateur)
	ReçoitConfirmationRéservationPayéeVol&Chambres	De base	métier	réponse	Agence (Coordonnateur)	Agence (Interface)
	Contrainte C5	Contrainte				

19	ConfirmeRéservationVol& Chambres	De base	utilisateur	sortie	Agence (Interface)	client

Le dernier point est la simplification des conditions formulées dans les contraintes. La première contrainte nécessite une simplification parce que *C1 and C2* sont incluses dans C3. Par conséquent, seule la condition C3 subsiste.

La représentation graphique indentée des interactions de base et structurées 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.

3.2.1.3. Etape 2.1.3 : Modéliser les services métier

Un service métier encapsule les règles du métier. Afin de modéliser les services métier selon les termes du modèle $\mathcal{MoS}$, nous proposons les étapes méthodologiques (DSM) suivantes (cf. Figure 64) :

Les étapes de découverte de services métier

DSM1: Extraire les communications du scénario dont l'agent cible a le rôle de fournisseur. Ces communications sont potentiellement des invocations de service appelées C_{Appel}.

DSM2: Coupler les communications demande/provision d'informations (ou de services) ou production/consommation de ressources afin d'identifier les opérations de type demande/réponse.

DSM3: Les communications orphelines de type C_{Appel} non couplées sont potentiellement des opérations de type unidirectionnel.

DSM4 : Identifier les messages en entrée et en sortie de l'opération.

DSM5 : L'ensemble des opérations d'un même agent forme l'interface du service métier.

Figure 64. Les étapes de découverte de services métier

- L'étape de découverte des services métier (DSM1) consiste à extraire les actions de communication représentant une invocation de service métier. On appelle ces communications C_{appel}.

En appliquant la DSM1 au scénario $S_{\textit{Effectuer une réservation}}$, on obtient les cinq communications numérotées 3, 3bis, 13, 13bis et 14 :

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.

- La DSM2 a pour but d'identifier les opérations de type demande/réponse d'un service métier. 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 que l'on appelle $C_{Réponse}$.

Le couplage des actions de communications C_{Appel} 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_{Appel} qui a pour objet une demande d'information est exprimée, comme nous l'avons détaillé à la section 3.1.4.1.2, à l'aide du modèle de clauses suivant :

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

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

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

<Action><Ressources>

L'application de la DSM2 au scénario $S_{\textit{Effectuer une réservation}}$, 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.*

Le Tableau 35 présente les opérations, de type demande/réponse, obtenues par application de la DSM2 aux couples de communications du scénario $S_{\textit{Effectuer une réservation}}$:

Tableau 8. Liste des opérations de type demande/réponse

Liste des C_{Appel}	**Liste des opérations identifiées**
3. L'agence de voyage envoie une demande de réservation de vol à la compagnie aérienne.	Demander réservation vol ()
3bis. L'agence de voyage envoie une demande de réservation de chambres à l'hôtel.	Demander réservation chambres ()
14. L'agence de voyage envoie les données bancaires à la banque.	Envoyer données bancaires ()

- La DSM3 aide à identifier, à partir des communications orphelines, les opérations de type unidirectionnel.

L'étape DSM3, appliquée aux communications du scénario $S_{\textit{Effectuer une réservation,}}$ identifie les C_{Appel} ayant les numéros 13 et 13bis qui sont des communications orphelines puisqu'il

n'existe aucune action complémentaire de type $C_{Réponse}$. Nous présentons le résultat obtenu au Tableau 9

Tableau 9. Liste des opérations de type unidirectionnel

Liste des C_{Appel}	**Liste des opérations identifiées**
13. L'agence de voyage confirme la réservation de vol à la compagnie aérienne.	Confirmer réservation vol ()
13bis. L'agence de voyage confirme la réservation de chambres à l'hôtel	Confirmer réservation chambres ()

- La DSM4 détermine les messages en entrée et en sortie afin d'avoir une spécification complète de chacune des opérations. Ceci consite à :

- Les ressources attachées à une action de communication C_{Appel} définissent les ressources en entrée de l'opération identifiée.
- Les ressources attachées à une communication $C_{Réponse}$ définissent les ressources en sortie.

L'application de l'étape DSM4 au scénario S *Effectuer une réservation*, donne la spécification de chaque opération présentée au Tableau 36.

Tableau 10. Les messages en entrée/sortie relatifs aux opérations

Opérations identifiées	**Liste des messages en entrée/sortie**	
Demander réservation vol ()	*Message en entrée*	Réservation_vol
	Message en sortie	Liste_vols Liste_prix
Demander réservation chambres ()	*Message en entrée*	Réservation_chambre_hôtel
	Message en sortie	Liste_Chambres Liste_Montant_total
Confirmer réservation vol ()	*Message en entrée*	Réservation_vol
	Message en sortie	--
Confirmer réservation chambres ()	*Message en entrée*	Réservation_chambre
	Message en sortie	--
Envoyer données bancaires ()	*Message en entrée*	Données_bancaires
	Message en sortie	Transaction

- La DSM5 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 pouvant être encapsulées dans des services métier 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 la DSM2 et la DSM3) d'un même agent fournisseur.

L'application de l'étape DSM5 au scénario $S_{Effectuer\ une\ réservation}$, permet d'obtenir les interfaces présentées au Tableau 11.

Tableau 11. Liste des interfaces relatives aux services métier

Liste des agents fournisseur	Liste des opérations	Les interfaces des services métier
Compagnie aérienne	Demander réservation vol () Confirmer réservation vol ()	ServiceCompAérienne
Hôtel	Demander réservation chambres () Confirmer réservation chambres ()	ServiceHôtel
Banque	Envoyer données bancaires ()	ServiceBanque

3.2.1.4. Etape 2.1.4 : Modéliser les services de coordination

La modélisation des services de coordination selon les termes du modèle $\mathcal{MoS}$, s'opère en cinq étapes :

Les étapes de découverte de services de coordination
DSC1 : Extraire, à partir du scénario, les actions ayant comme source ou cible un agent qui a le rôle de Coordonnateur. Chaque action identifiée correspond à une activité de base du service de coordination dont on déterminera la nature
DSC2 : Déterminer les agents participant à la coordination, les services métier invoqués et les services d'interface utilisateur invoquant le service de coordination.
DSC3 : Déterminer les données nécessaires à la coordination.
DSC4 : Identifier l'ordre de séquencement des actions.
DSC5 : Identifier les activités de type affectation nécessaires pour adapter les données entre le service d'interface utilisateur et les services métier.

Figure 65. Les étapes de découverte du service de coordination

Chacune de ces étapes identifie un des aspects du service de coordination selon les termes du modèle $\mathcal{MoS}$.

- L'étape de découverte du service de coordination numéro 1 (*DSC1*) consiste à extraire, à

partir du scénario, les actions de communication qui ont comme source ou cible un agent dont le rôle est Coordonnateur. On appelle ces actions $C_{Coordination}$.

Les $C_{Coordination}$ correspondent aux actions échangées à partir de et vers le Coordonnateur pour réaliser l'objectif de la coordination. Par conséquent, les $C_{Coordination}$ correspondent aux activités de base du service de coordination.

Le Tableau 38 résume les communications $C_{Coordination}$ relatives au scénario $S_{\textit{Effectuer une réservation}}$ et les actions de base correspondantes.

Tableau 12. Liste des $C_{Coordination}$ et leurs activités de base

Liste des $C_{Coordination}$	Liste des activités de base
1. Le client formule une demande de réservation de vol et de chambres d'hôtel à partir de l'agence de voyage	EnvoieDemandeReservationVol&Chambres
2. L'agence de voyage enregistre la demande réservation de vol et de chambres d'hôtel du client	affectation
3. L'agence de voyage envoie une demande de réservation de vol à la compagnie aérienne	DemanderReservationVol
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.	DemanderReservationChambres
6bis. L'hôtel confirme les chambres et le montant total à payer à l'agence de voyage.	
8. L'agence de voyage notifie les propositions de vol, de chambres et le montant à payer au client	ReçoitListeProposition
9. Le client sélectionne une des propositions.	EnvoiePropositionSélectionnée
11. L'agence de voyage affiche le formulaire de saisie pour compléter la réservation et effectuer le paiement en ligne au client.	ReçoitAccuséRception
12. Le client confirme en envoyant ses données personnelles et ses données bancaires à l'agence de voyage.	EnvoiePaiementEnLigne
13. L'agence de voyage confirme la réservation de vol à la compagnie aérienne.	ConfirmerReservationVol
13bis. L'agence de voyage confirme la réservation de chambres à l'hôtel.	ConfirmerReservationChambres
14. L'agence de voyage envoie les données bancaires à la banque.	EnvoyerDonnéesBancaires
18. La banque notifie la transaction à l'agence de voyage.	
19. L'agence de voyage confirme la réservation au client.	ReçoitConfirmationRéservationPayéeVol&Chambres

Une fois les $C_{Coordination}$ identifiées, leur type est spécifié. Les $C_{Coordination}$ peuvent être : un appel pour invoquer une opération dans un service métier, une réception pour attendre un message d'un service métier, une réponse pour répondre au service métier ou une affectation pour copier les données d'un message à un autre en introduisant de nouvelles données.

Les communications de type $C_{Coordination}$ qui se couplent en C_{Appel} et $C_{Reponse}$ sont traitées, dans

le service de coordination, 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 de coordination, par les activités de type réception alors que les communications utilisateur en sortie sont considérées comme des activités de type réponse.

Tableau 13. Liste des $C_{Coordination}$ et leurs types respectifs

Liste des $C_{Coordination}$	Type
1. Le client formule une demande de réservation de vol et de chambres d'hôtel à partir de l'agence de voyage	réception
2. L'agence de voyage enregistre la demande réservation de vol et de chambres d'hôtel du client	affectation
3. L'agence de voyage envoie une demande de réservation de vol à la compagnie aérienne	appel
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.	appel
6bis. L'hôtel confirme les chambres et le montant total à payer à l'agence de voyage.	
8. L'agence de voyage notifie les propositions de vol, de chambres et le montant à payer au client	réponse
9. Le client sélectionne une des propositions.	réception
11. L'agence de voyage affiche le formulaire de saisie pour compléter la réservation et effectuer le paiement en ligne au client.	réponse
12. Le client confirme en envoyant ses données personnelles et ses données bancaires à l'agence de voyage.	réception
13. L'agence de voyage confirme la réservation de vol à la compagnie aérienne.	appel
13bis. L'agence de voyage confirme la réservation de chambres à l'hôtel.	appel
14. L'agence de voyage envoie les données bancaires à la banque.	appel
18. La banque notifie la transaction à l'agence de voyage.	
19. L'agence de voyage confirme la réservation au client.	réponse

- La DSC2 propose de déterminer (i) les agents participant à la coordination, (ii) les services métier invoqués et (iii) les services d'interface utilisateur invoquant le service de coordination.

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

Le service de coordination a un rapport étroit avec les services métier en faisant référence à leurs interfaces respectives lors de la coordination. Par conséquent, les services métier sont identifiés à partir des scénarios en appliquant les étapes proposées à l'étape 2.1.3 (cf. section 3.2.1.3).

Les services d'interface utilisateur ont un rapport avec le service de coordination à travers les interactions métier de type Demande utilisateur. Ces interactions correspondent aux activités de base de type Réception dans le service de coordination. Par conséquent, les services d'interface utilisateur sont identifiés à partir des scénarios en appliquant les étapes proposées à l'étape 2.1.2 (cf. section 3.2.1.2).

L'étape DSC2 appliquée aux $C_{Coordination}$ du scénario S *Effectuer une réservation* aide à déterminer les éléments de composition à savoir :

- Les participants dans la coordination tels que la compagnie aérienne, l'hôtel et la banque.
- Les services métier 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 Coordonnateur.

Tableau 14. Les activités de base et leurs relations avec les autres services

Liste des $C_{Coordination}$	**Type**	**Relation avec les services métier ou d'interface utilisateur**
1. Le client formule une demande de réservation de vol et de chambres d'hôtel à partir de l'agence de voyage	réception	Interface *Effectuer Réservation*
2. L'agence de voyage enregistre la demande réservation de vol et de chambres d'hôtel du client	affectation	
3. L'agence de voyage envoie une demande de réservation de vol à la compagnie aérienne	appel	service ServiceCompAérienne
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.	appel	service ServiceHotel
6bis. L'hôtel confirme les chambres et le montant total à payer à l'agence de voyage.		
8. L'agence de voyage notifie les propositions de vol, de chambres et le montant à payer au client	réponse	Interface *Effectuer Réservation*

9. Le client sélectionne une des propositions.	réception	Interface *Effectuer Réservation*
11. L'agence de voyage affiche le formulaire de saisie pour compléter la réservation et effectuer le paiement en ligne au client.	réponse	Interface *Effectuer Réservation*
12. Le client confirme en envoyant ses données personnelles et ses données bancaires à l'agence de voyage.	réception	Interface *Effectuer Réservation*
13. L'agence de voyage confirme la réservation de vol à la compagnie aérienne.	appel	service ServiceCompAerienne
13bis. L'agence de voyage confirme la réservation de chambres à l'hôtel.	appel	service ServiceHotel
14. L'agence de voyage envoie les données bancaires à la banque.	appel	service ServiceBanque
18. La banque notifie la transaction à l'agence de voyage.		
19. L'agence de voyage confirme la réservation au client.	réponse	Interface *Effectuer Réservation*

- La DSC3 aide à construire le modèle de données relatif à un service de coordination. Nous proposons d'extraire les ressources relatives aux communications $C_{Coordination}$ qui ont comme cible un agent dont le rôle est Coordonnateur. Ces ressources constituent les messages en entrée nécessaires à la coordination. Les messages en sortie ne sont dans ce cas, qu'une synthèse des messages en entrée.

En appliquant la DSC3 aux $C_{Coordination}$ du scénario S *Effectuer une réservation*, nous obtenons les ressources relatives au service de coordination présenté au Tableau 41 :

Tableau 15. Liste des ressources

Liste des $C_{Coordination}$	**Ressources identifiées**
1. Le client formule une demande de réservation de vol et de chambres d'hôtel à partir de l'agence de voyage.	Demander_réservation_vol_chambres
6. La compagnie aérienne confirme les vols et les prix à l'agence de voyage.	Liste_Vols Liste_Prix
6bis. L'hôtel confirme les chambres et le montant total à payer à l'agence de voyage.	Liste_Chambres Liste_Montant_total
10. Le client confirme les données bancaires à l'agence de voyage.	Données_Bancaires
16. La banque notifie la transaction à l'agence de voyage.	Transaction

- La DSC4 définit le modèle d'orchestration du service de coordination en considérant l'ordre d'exécution des actions de communication $C_{Coordination}$. Les $C_{Coordination}$ peuvent être structurées en séquence, concurrence, répétition ou contrainte.

Dans un service de coordination, les $C_{Coordination}$ structurées en séquence correspondent aux activités structurées de type séquence. Les $C_{Coordination}$ 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 de coordination. Enfin, la contrainte correspond à la structure de condition dans un service de coordination.

En appliquant la DSC4 aux $C_{Coordination}$ du scénario $S_{Effectuer\ une\ réservation}$ et en se référant à la section 3.1.3.2 (La notion de flux), on note que les communications $C_{Coordination}$ sont structurées en séquence. Par conséquent, les activités générées sont structurées de la même manière.

- La DSC5 permet de suggérer au concepteur de vérifier la nécessité des activités de type affectation.

La transformation de données est nécessaire dans les deux cas suivants :

- Avant chaque activité de type appel de service métier afin de construire les données nécessaires à l'appel du service métier
- Avant chaque activité de type réponse afin de construire le message envoyé au service d'interface.

Des activités de type affectation sont introduites afin de rendre possible le passage de données entre (i) le service de coordination et le service métier et (ii) le service de coordination et le service d'interface utilisateur.

3.2.2. Etape 2.2 : Construire le modèle $\mathcal{MoS}$ par analyse des scénarios alternatifs de succès

Un scénario alternatif de succès représente une manière alternative de réalisation du but du service atomique.

Le scénario alternatif et le scénario de base ont en commun les actions qui précédent la contrainte. L'imbrication générée à partir de la négation de la contrainte identifie des actions alternatives de satisfaction du but initial.

La construction du modèle $\mathcal{MoS}$, à partir du scénario alternatif de succès, ne considère que les actions générées à partir de la négation de la contrainte. Ceci consiste à re-appliquer le processus proposé à l'étape 2.1 (cf. section 3.2.1) et à l'étendre avec des règles supplémentaires.

3.2.2.1. Spécialiser les agents du scénario

L'application de la sous étape 2.1.1, précise les agents suivants et leurs rôles respectifs :

- *Demandeur* : l'agent utilisateur qui initie la demande d'information ou de service et qui bénéficie du service
- *Interface* : l'agent système qui gère les interactions entre le demandeur et le

coordonnateur.

- *Coordonnateur* : agent système qui contrôle tous les échanges entre agents et gère l'ensemble des données partagées.
- *Fournisseur* : propriétaire de l'information et/ou du service demandé.

3.2.2.2. Modéliser le service d'interface utilisateur

L'application de la sous étape 2.1.2, selon la séquence des règles (DSU1 à DSU5), modélise le service d'interface utilisateur.

Nous étendons notre processus avec la règle DSU6. Le but est d'alimenter le service d'interface utilisateur par la séquence structurée ou l'interaction de base obtenue.

3.2.2.3. Modéliser les services métier

L'objectif de cette étape consiste à modéliser les services métier invoqués à partir du scénario alternatif en appliquant la séquence des règles (DSM1 à DSM5).

Les services obtenus peuvent être les mêmes que ceux obtenus à partir du scénario de base mais aussi :

(i) Des services métier nouveaux ou,

(ii) Des opérations nouvelles pour un même service métier.

3.2.2.4. Modéliser le service de coordination

L'application de la sous étape 2.1.4, selon la séquence des règles (DSC1 à DSC6), complète la modélisation du service de coordination.

Nous étendons notre processus avec la règle DSC7. Le but est d'alimenter le service de coordination par les activités identifiées en procédant comme suit :

- Le scénario alternatif diverge du scénario de base à un point de divergence correspondant à une contrainte que l'on appelle $\text{Contrainte}_{\text{divergente}}$.
- L'activité structurée du scénario alternatif ($\text{AS}_{\text{alternatif}}$) représente le traitement de coordination à appliquer lorsque la condition de la contrainte $\text{Contrainte}_{\text{divergente}}$ n'est pas vérifiée. Par conséquent, l'activité structurée $\text{AS}_{\text{alternatif}}$ est ajoutée comme cas 'sinon' de la contrainte $\text{Contrainte}_{\text{divergente}}$ incluse dans l'activité structurée identifiée à partir du scénario de base.

3.2.3. Etape 2.3 : Construire le modèle $\mathcal{MoS}$ par analyse des scénarios alternatifs d'échec

Cette section permet de construire le modèle $\mathcal{MoS}$ 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 $\mathcal{MoS}$, à partir du scénario alternatif d'échec, ne considère que les actions précédemment présentées à la Figure 62. 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 re-formulation d'une nouvelle demande au client.

Ceci consiste à re-appliquer le processus proposé à l'étape 2.1 (cf. section 3.2.1) et à l'étendre avec des règles supplémentaires.

L'application de la sous étape 2.1.1, précise les agents suivants et leurs rôles respectifs :

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

L'application de la sous étape 2.1.2, selon la séquence des règles (DSU1 à DSU5), modélise le service d'interface utilisateur. Nous présentons le résultat obtenu dans le Tableau 16.

Tableau 16. Liste des éléments caractérisant le service d'interface utilisateur

N°	Interaction de base	Type	Sens	Agent source	Agent cible	Ressource
8	AfficheMessageReformulation	utilisateur	sortie	Agence (Interface)	Client	MessageReformulation

Nous étendons notre processus avec la règle DSU7. 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 2.1.3, complète la description de la modélisation du service métier ServiceCompAérienne correspondant à son opération *Demander réservation vol ()* 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 2.1.4, selon la séquence des règles (DSC1 à DSC6), complète la modélisation du service de coordination par la définition des activités de compensation. Nous considérons le scénario alternatif d'échec comme étant une exception au niveau du service de coordination. 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 17.

Tableau 17. Liste des activités de base

Liste des $C_{Coordination}$	**Liste des activités de base**	**Type**	**Relation avec les autres services**	**Ressources**
8- L'agence de voyage envoie un message de re-formulation d'une nouvelle demande au client	ReçoitPropositionReformulation	réponse	ServiceInterface	PropositionReformulation

Nous étendons notre processus avec la règle DSC8. Le but est d'alimenter le service de coordination par les activités de compensation déjà identifiées. L'application de la règle DSC8 permet d'alimenter le service de coordination par l'activité 8 comme étant une activité de compensation produite suite à l'exécution du message d'exception *non disponibilité*.

L'activité de type lever_exception *non disponibilité* est créée et ajoutée comme une activité du cas sinon de la contrainte (i.e. il existe au moins une proposition de vol et au moins une proposition de chambre) du service de coordination.

3.3. Etape 3 : Identifier le modèle d'implémentation

Nous présentons dans cette section un ensemble de correspondances entre le modèle $\mathcal{MoS}$ et WSDL (pour Web Service Description Language) [WSDL01] d'une part et BPEL4WS (pour

Business Process Execution Language for Web Services) [Andrews03] d'autre part. Une fois les correspondances sont réalisées, le code source, à savoir le document WSDL et le document BPEL4WS, peut être généré à l'aide d'un ensemble d'outils spécifiques [Lopes05].

Nous avons choisi les technologies de services web les plus importantes à savoir WSDL et BPEL. D'une part, WSDL permet la modélisation des aspects statiques d'un service, et d'autre part, BPEL permet la modélisation des aspects dynamiques d'un service à partir de la composition d'autres services. Le choix du langage BPEL est motivé par le fait de son adoption par les milieux académique et industriel.

Dans le modèle $\mathcal{MoS}$, chaque service métier du modèle $\mathcal{MoS}$ constitue l'interface à utiliser pour le développement d'applications à base de services. Il peut être implémenté en utilisant le langage WSDL qui fournit un modèle et un format XML pour décrire des services web.

De même, le service de coordination du modèle $\mathcal{MoS}$ définit le processus par lequel les services métier sont exécutés. Il peut être implémenté en utilisant le langage BPEL4WS qui a un rapport étroit avec WSDL car il fait référence à un Port Type de services utilisés dans la coordination.

Enfin, le service d'interface utilisateur du modèle $\mathcal{MoS}$ peut représenter des formulaires ou des composants graphiques de type javaBeans par exemple.

3.3.1. Les correspondances entre le service métier du modèle $\mathcal{MoS}$ et WSDL

WSDL fournit un modèle et un format XML pour décrire des services web. WSDL permet de séparer la description de la fonctionnalité abstraite d'un service des détails concrets d'une description de service, c'est à dire la séparation de «quelle» fonctionnalité est fournie de «comment» et «où» celle-ci est offerte [W3C03]. Un document WSDL est composé essentiellement de définitions. Chaque définition est composée d'interfaces[9], messages, liaisons (bindings) et services.

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

Dans cette section, nous nous intéressons à spécifier les correspondances entre les éléments du modèle $\mathcal{MoS}$ et en particulier les services métier et la partie abstraite de WSDL.

Nous introduisons tout d'abord les différents concepts formant la partie abstraite de WSDL à savoir :

[9] Le PortType a été nommé `Interface` à partir de WSDL 1.2.

- `Type`: utilisé pour définir un type de données abstraites simples ou complexes en conformité avec un schéma XML.
- `Message`: décrit un format abstrait d'un message particulier que le service Web envoie ou reçoit. Il contient des parties (par exemple Part) décrivant chaque partie d'un message.
- `Operation` : décrit les types d'appels de manière abstraite. Les types d'appel sont caractérisés par les éléments One-WayOperation, RequestResponseOperation, et NotificationOperation. L'élément ParamType identifie quels sont les messages en entrée, en sortie et d'exception.
- `PortType`: définit l'interface d'un service. Il contient un ensemble d'opérations qu'un service envoie et/ou reçoit.

Dans l'annexe A (Figure 1), un méta-modèle complet de WSDL est fourni.

Les règles de correspondance entre le modèle $\mathcal{MOS}$ et WSDL se résument comme suit : l'élément Service métier du modèle $\mathcal{MOS}$ est l'équivalent de l'élément `Definition` de WSDL. L'élément Interface est l'équivalent de l'élément `PortType` de WSDL. L'élément Opération et Type d'appel du modèle $\mathcal{MOS}$ est l'équivalent de l'élément `Operation` de WSDL. L'élément message du modèle $\mathcal{MOS}$ est l'équivalent de l'élément `Message` et `Type` de WSDL. L'élément Exception du modèle $\mathcal{MOS}$ est l'équivalent de l'élément `Fault` de WSDL.

Le Tableau 18 présente un récapitulatif des correspondances entre le service métier du modèle $\mathcal{MOS}$ et WSDL.

Tableau 18. Une correspondance entre le service métier et WSDL

Modèle $\mathcal{MOS}$	**WSDL**
Service métier	Definition
Interface	PortType
Opération et Type d'appel	Operation
Message	Message et Type
Exception	Fault

3.3.2. Les correspondances entre le service de coordination du modèle $\mathcal{MOS}$ et BPEL4WS

Dans cette section, nous nous intéressons à spécifier les correspondances entre les éléments

du modèle $\mathcal{MoS}$ et en particulier les services de coordination et BPEL4WS.

Nous introduisons tout d'abord les différents éléments formant BPEL4WS, à savoir :

- `Agent` : définit les différentes parties qui agissent les unes sur les autres lors de l'exécution d'un processus. Il est caractérisé par un ou plusieurs PartnerLink qui spécifient les relations de conversation entre deux services par le biais de la déclaration de leurs rôles. Chaque rôle spécifie un port Type de WSDL qu'un partenaire réalise.
- `Variable` : définit les variables de données utilisées par le service de coordination, et permet au service de coordination de maintenir les données et les historiques basés sur l'échange de données.
- `FaultHandlers` : gestionnaire de défauts de l'exécution de services définissant les activités qui seront exécutées en réaction à la faute.
- `Activity` : activité structurée en plusieurs parties comme le flux de contrôle (par exemple, Switch et While) et de messages (données transférées entre les activités).

Dans l'annexe A (Figure 2), un méta-modèle complet de BPEL4WS est fourni.

BPEL repose sur un modèle constitué d'activités qui peuvent être de deux types : de base ou structuré. Les activités de base représentent l'invocation d'une opération réalisée par un service web, par exemple invoke, receive et reply. Chaque activité de base est réalisée par une opération du service et chaque service est associé à un PartenerLink. Celui-ci contient les informations sur le rôle et l'interface (PortType) du service qui réalise cette opération.

Les activités structurées sont basées sur l'approche des hiérarchies d'activités dans laquelle un processus est spécifié par le raffinement progressif d'une activité dans un arbre d'activités. Les nœuds représentent les actions à exécuter, et les nœuds intermédiaires définissent une contrainte entre les activités [Alonso04].

Dans le modèle $\mathcal{MoS}$, chaque activité de base peut être liée à une opération du service métier et correspond soit à un appel (invoke), soit à une réception (receive), soit à une réponse (reply).

Les règles de correspondances entre le modèle $\mathcal{MoS}$ et BPEL se résument comme suit : l'élément Service de coordination est l'équivalent de l'élément `Process` de BPEL. L'élément Agent du modèle $\mathcal{MoS}$ est l'équivalent de l'élément `Agent` de BPEL et le rôle de cet agent est équivalent à l'élément `Role` en BPEL. L'élément Ressource du modèle $\mathcal{MoS}$ est l'équivalent de l'élément `Variable` de BPEL. L'élément Activité de base de type appel du modèle $\mathcal{MoS}$ est l'équivalent de l'élément `Invoke` de BPEL. L'élément Activité de base de type réponse

du modèle $\mathcal{MOS}$ est l'équivalent de l'élément `Reply` de BPEL. L'élément Activité de base de type reception du modèle $\mathcal{MOS}$ est l'équivalent de l'élément `Receive` de BPEL. L'élément Activité de base de type affectation du modèle $\mathcal{MOS}$ est l'équivalent de l'élément `Assign` de BPEL. L'élément Activité de base de type throw du modèle $\mathcal{MOS}$ est l'équivalent de l'élément `Lever_Exception` de BPEL L'élément Séquence est l'équivalent de l'élément `Sequence` de BPEL. L'élément Flux du modèle $\mathcal{MOS}$ est l'équivalent de l'élément `Flow` de BPEL. L'élément Boucle est l'équivalent de l'élément `While` de BPEL. L'élément Condition est l'équivalent de l'élément `Switch` de BPEL. L'élément Exception est l'équivalent de l'élément `FautHandlers` de BPEL. Enfin, les activités de compensation sont équivalentes à l'élément `CompensationHandler` de BPEL.

Le Tableau 19 présente un récapitulatif des correspondances entre le service de coordination du modèle $\mathcal{MOS}$ et BPEL.

Tableau 19. Une correspondance entre le service de coordination et BPEL

Modèle $\mathcal{MOS}$	**BPEL**
Service de coordination	Process
Agent	Agent
Ressource	Variable
Activité	Activity
Appel	Invoke
Réponse	Reply
Réception	Receive
Affectation	Assign
Lever_Exception	Throw
Séquence	Sequence
Concurrence	Flow
Boucle	While
Contrainte	Switch
Exception	FautHandlers
Activité de compensation	CompensationHandler

4. 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 $\mathcal{M}_{oS}$ et d'une démarche méthodologique conduisant à sa construction.

Le modèle $\mathcal{M}_{oS}$ décrit à un niveau opérationnel, les services logiciels 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 d'une plate-forme spécifique.

La démarche, que nous avons défini dans ce chapitre, permet de (i) guider l'utilisateur dans l'ecriture des scénarios où chaque scénario représente manière possible de réalisation du service atomique, (ii) générer de façon systématique l'opérationnalisation des service intentionnel atomique à l'aide du modèle $\mathcal{M}_{oS}$ et (iii) établir les correspondances adéquates avec les modèles des standards des 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 $\mathcal{M}_{iS}$ et les services logiciels proposés par les développeurs, à un niveau opérationnel, à l'aide des services du modèle $\mathcal{M}_{oS}$.

Certains avantages de l'approche proposée sont les suivants :

- Le même modèle $\mathcal{M}_{oS}$ peut être utilisé plusieurs fois pour générer des modèles sur des plateformes différentes (service web ou autre).
- L'indépendance de la logique métier aux technologies d'implémentation.
- L'évolution simultanée du niveau intentionnel et opérationnel des services.

CHAPITRE 5

Processus de construction du modèle $\mathcal{MiS}$

1. INTRODUCTION

Ce chapitre est consacré à la définition du processus qui permet d'identifier les différents types de services et de les représenter dans les termes du modèle $\mathcal{MiS}$ que nous avons présenté au chapitre 3.

Dans le cadre de l'architecture iSOA, le processus à préconiser permet au fournisseur d'identifier et décrire les services du business qui sont mis à disposition grâce à l'annuaire des services.

Ce chapitre est composé de quatre sections. La première introduit le processus de construction du modèle $\mathcal{MiS}$. Ensuite, les sections 2 et 3 présentent les étapes du processus. Finalement, la section 4 conclue ce chapitre.

Le processus de construction du modèle $\mathcal{MiS}$ est constitué de deux étapes :

- Etape 1 : construction du modèle de capture des besoins.

- Etape 2 : génération du modèle de représentation des services.

Les besoins utilisateurs sont identifiés à la première étape. Ces besoins sont décrits avec un modèle orienté buts appelé carte [Rolland00]. La carte exprime les besoins des utilisateurs par des buts et des stratégies pour les atteindre. Le formalisme de la carte représente les besoins par un ensemble de stratégies pour atteindre un même but et par des compositions de buts et de stratégies pour satisfaire un but final. Dans ce qui suit, nous détaillons les deux étapes, à savoir la construction du modèle de capture des besoins et la génération du modèle de représentation des services.

2. ETAPE 1 : CONSTRUCTION DU MODELE DE CAPTURE DES BESOINS

Nous présentons dans cette section le formalisme de la carte que nous avons choisi pour construire le modèle de capture des besoins des utilisateurs.

2.1. Présentation de la carte

Le modèle de la carte est un système de représentation des processus dans un mode intentionnel. Il fait partie de la classe des modèles de buts. Le système de représentation de la carte utilise le concept de *but* et se différencie des autres modèles par l'introduction du concept de *stratégie* pour atteindre un but. Dans ce système de représentation, un *but* est ce qu'on cherche à atteindre. Une *stratégie* est une manière de réaliser un but. Le modèle de la carte repose sur un ordonnancement déclaratif et flexible de buts et de stratégies.

La Figure 66 présente le méta-modèle de la carte, ses concepts clés et leurs relations en utilisant la notation UML.

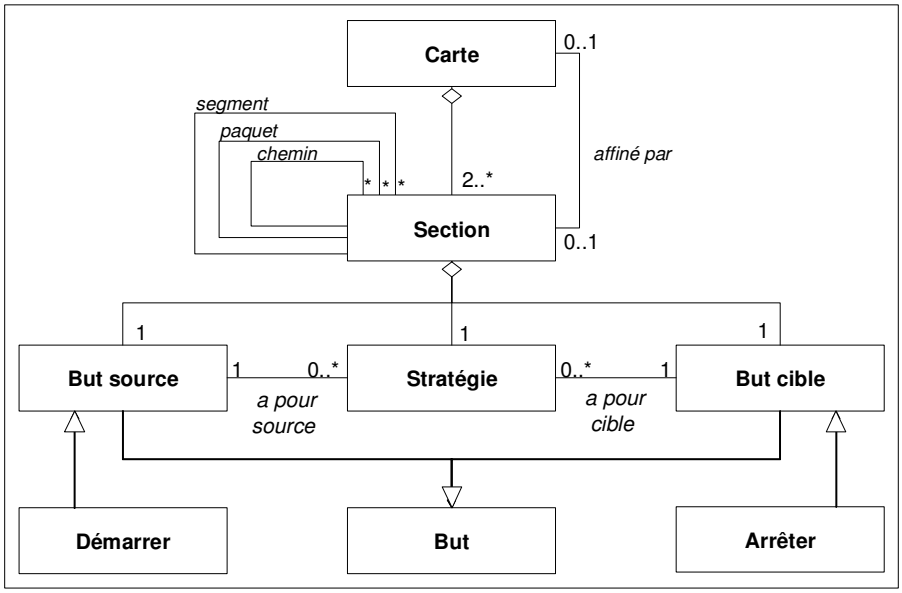


Figure 66. Méta modèle de la Carte

Les sous-sections suivantes présentent les différents concepts de ce méta-modèle. A savoir, *carte, but, stratégie, section, multi-segment, multi-chemin, paquet* et *affinement*.

2.1.1. Carte

Le méta-modèle de la carte (cf. Figure 66) montre qu'une carte est composée de deux ou plusieurs sections. Une section est une agrégation de deux types de buts, un but source et un but cible, et d'une stratégie. Chaque section correspond à une stratégie qui peut être utilisée pour réaliser un but cible, une fois que le but source a été atteint.

La carte est représentée par un graphe orienté et étiqueté. Les buts sont les nœuds et les stratégies en sont les arcs. La nature orientée de la carte traduit le flux du but source au but cible via la stratégie. Une section est ainsi représentée par deux nœuds reliés par une flèche. Dans l'exemple de la Figure 67, nous proposons un exemple de carte qui a pour but *Satisfaire efficacement les besoins en produits.*

L'exemple de la Figure 67 comporte quatre buts (*Démarrer*, *Acquérir des produits*, *Contrôler le stock* et *Arrêter*), douze stratégies (*Par seuil de réapprovisionnement*, *Par prévisions stratégiques*, *Manuellement*, *Entrer en stock des produits livrés*, *Entrer les produits directement*, *Périodique*, *Echantillonnage*, *En continu*, *Par valuation*, *Par inspection de la qualité*, *Par transfert des produits*, *Par contrôle du paiement des factures*) et douze sections.

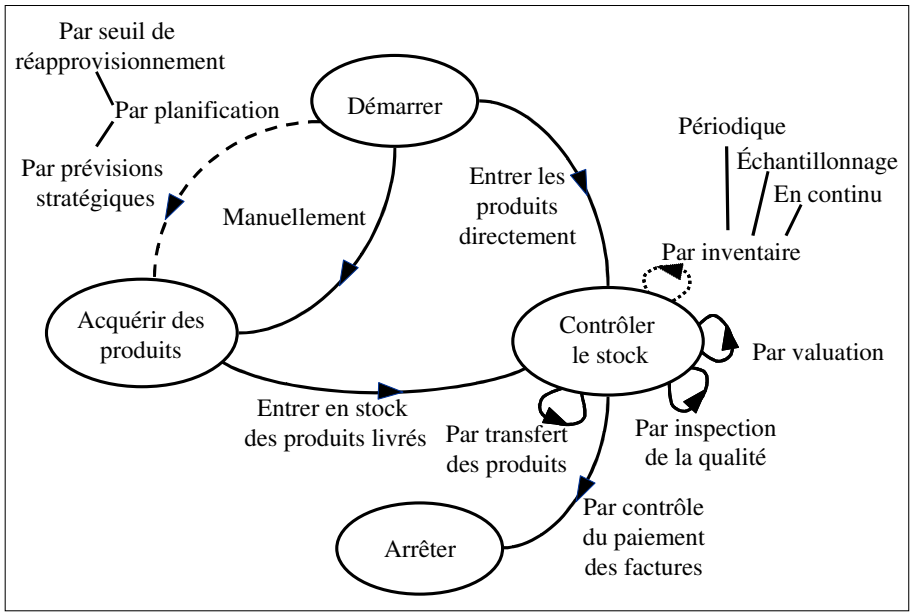


Figure 67. Exemple d'une carte *Satisfaire efficacement les besoins en produits*

2.1.2. But

Un *but* représente ce qu'on cherche à atteindre. Selon [Jackson95], un but est une déclaration 'optative' qui exprime ce que l'on veut, un état ou un résultat que l'on cherche à atteindre. Par exemple, comme le montre la Figure 67, *Acquérir des produits* et *Contrôler le stock* sont deux buts dans le domaine de la gestion de stock.

Chaque carte possède deux buts particuliers : *Démarrer* et *Arrêter* pour respectivement commencer et terminer l'exécution de la carte.

De plus, un but ne peut apparaître qu'une seule fois dans la même carte et chaque but a les caractéristiques suivantes :

- Il fait abstraction du processus de sa réalisation ;
- Il capture « l'essence » du processus ;
- Il traduit un objectif du métier ;
- Son expression est plus ou moins abstraite.

2.1.3. Stratégie

Une *stratégie* est une approche, une manière ou un moyen pour réaliser un but cible à partir du but source. Dans une carte, les stratégies correspondent aux différentes façons de réaliser les buts. Une stratégie s'associe au but auquel elle s'applique. Elle a pour objectif principal d'extérioriser la façon d'atteindre ce but puisqu'elle permet de distinguer le but et la façon de le réaliser. En outre, fournir plusieurs stratégies pour atteindre le même but permet de

suggérer des façons alternatives de réaliser ce but. Ceci permet plus d'adaptabilité et de souplesse dans l'exécution des processus métier.

Dans l'exemple de la Figure 67, les demandes d'achats de produits peuvent être créées *Manuellement*. Les demandes peuvent être générées automatiquement et sont transformées en ordres d'achat à l'aide de la stratégie *Par planification.* Les deux stratégies *Manuellement* et *Par planification* sont des manières différentes de réaliser le but *Acquérir des produits*.

Une même stratégie peut apparaître dans une ou plusieurs sections de la même carte. Deux sections reposant sur une même stratégie peuvent donc être similaires ou différentes selon les processus qu'elles référencent. Enfin, les stratégies sont des éléments discriminants du fait qu'elles prennent la forme d'approches précises pour réaliser un but.

2.1.4. Section

Une *section* est un triplet $<B_s,B_c,S>$ composé d'un but source B_s, d'un but cible B_c et d'une stratégie S. Une section exprime la réalisation du but cible en utilisant la stratégie une fois que le but source a été réalisé. Par exemple, dans la Figure 67, l'agrégation du but source *Acquérir des produits*, du but cible *Contrôler le stock* et de la stratégie *Entrer en stock des produits livrés* définit la section <*Acquérir des produits, Contrôler le stock, Entrer en stock des produits livrés*>. Ici la stratégie *Entrer en stock des produits livrés* caractérise le flux du but source *Acquérir des produits* au but cible *Contrôler le stock* et la manière de réaliser la cible. Ceci signifie que les livraisons sont contrôlées par rapport aux ordres d'achat, et envoyées sur le lieu de stockage ou de consommation.

2.1.5. Conventions de codification des cartes

Nous adoptons l'une des conventions suivantes pour codifier les cartes : la codification locale ou la codification absolue.

Codification locale

- Chaque but est codé par une lettre de l'alphabet. Cela permet de manipuler plusieurs buts dans une même carte. A la Figure 68 par exemple, les quatre buts de la carte à savoir *Démarrer*, *Acquérir des produits*, *Contrôler le stock* et *Arrêter* sont respectivement codés par a, b, c et d.
- Les stratégies sont numérotées relativement à leurs buts cible et source. Ainsi, deux stratégies ayant les mêmes buts source et cible seront respectivement numérotées par deux numéros distincts. Une carte peut donc avoir plusieurs stratégies numérotées de la même manière.

Par exemple, à la Figure 68, les stratégies de la carte sont codées comme suit : 1, 2 et 3 (ab) ; 1 (bc) ; 1, 2, 3, 4, 5 et 6 (cc) ; 1 (cd) ; 1(ac).

- Les sections sont codées par juxtaposition de trois éléments (i) la lettre du but source, (ii) la lettre du but cible et (iii) le numéro de la stratégie. Par exemple, la section ab_1 permet, en partant du but *a*, d'atteindre le but *b* en suivant la stratégie *1*.

La Figure 68 représente un exemple de codification locale d'une carte. Nous avons les douze sections suivantes : ab_1, ab_2, ab_3, bc_1, cc_1, cc_2, cc_3, cc_4, cc_5, cc_6, cd_1 et ac_1.

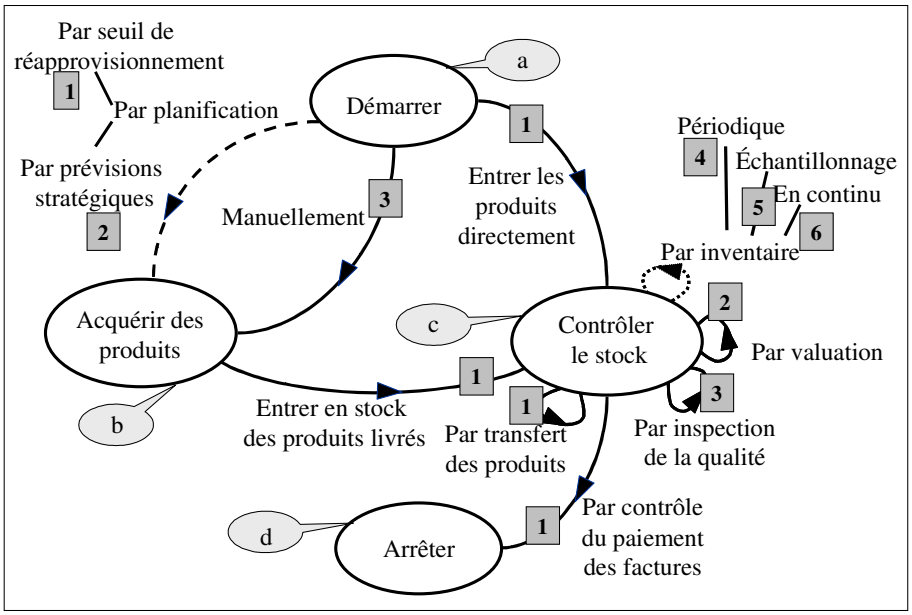


Figure 68. Exemple de codification d'une carte

Codification absolue

Avec la codification absolue, la carte racine est appelée C. Ses buts sont nommés C.a, C.b… et ses sections $C.ab_1$, $C.ab_2$, $C.ab_3$…

2.1.6. Relations dans le modèle de la Carte

Le méta-modèle de la carte, présenté à la Figure 66, montre trois relations qui relient des sections entre elles (*segment*, *chemin* et *paquet*). Dans les sous-sections suivantes nous présentons ces trois relations.

2.1.6.1. Paquet

On a une relation de type *paquet* entre plusieurs sections quand :

- ces sections ont le même couple de buts (source et cible) ; et
- le choix d'une de ces sections pour la réalisation du but cible empêche la sélection des autres sections.

Il s'agit de plusieurs sections mutuellement exclusives. En d'autres termes la relation entre ces sections est de type *OU Exclusif*. Un paquet est représenté graphiquement par une flèche en pointillés (cf. Figure 69).

La relation de type paquet entre deux buts de codes respectifs k et l est notée par $P_{kl} = \otimes(kl_1, kl_2, \ldots, kl_n)$ où kl_i i∈ {1..n} et « ⊗ » l'opérateur d'exclusion qui relie les sections exclusives kl_i.

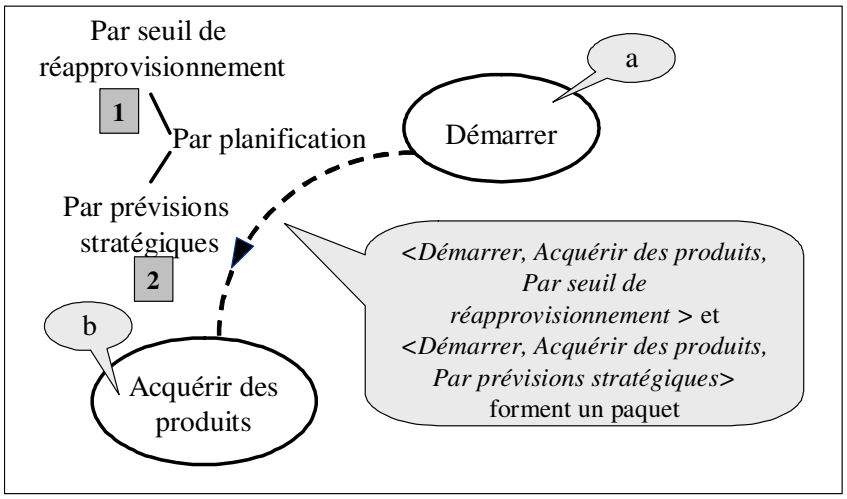


Figure 69. Exemple de paquet

La Figure 69 montre un exemple de paquet entre les deux buts *Démarrer* et *Acquérir des produits*. Le nom de ce paquet est *Par planification*. Il regroupe deux moyens alternatifs pour l'acquisition des produits : *Par seuil de réapprovisionnement* ou *Par prévisions stratégiques*. L'utilisation de l'un de ces deux moyens exclut l'utilisation de l'autre.

Le paquet *Par planification* (cf. Figure 69) est noté par $P_{ab} = \otimes(ab_1, ab_2)$ où ab_1 et ab_2 sont des sections mutuellement exclusives.

2.1.6.2. Segment

Dans une carte, il est possible de réaliser un but cible à partir d'un but source en utilisant plusieurs stratégies complémentaires. Chacune de ces stratégies, couplée avec le but source et le but cible, définit une section dans la carte. Cette topologie est appelée *multi-segment* (multi-thread). Elle peut être vue comme une relation logique *ET/OU*. Les sections appartenant à un multi-segment possèdent le même but source et le même but cible.

La relation de type multi-segment entre deux buts de codes respectifs k et l est notée par $MS_{kl} = \vee(kl_1, kl_2, \ldots, kl_n)$ où kl_i i∈ {1..n} et « ∨ » l'opérateur qui relie les sections complémentaires kl_i.

Dans l'exemple de la Figure 70, il existe deux manières différentes pour l'acquisition des produits, manuellement ou par planification. Les deux stratégies *Manuellement* et *Par planification* ont le même but source *Démarrer* et le même but cible *Acquérir des produits*. Les deux sections <*Démarrer, Acquérir des produits, Manuellement*> et <*Démarrer, Acquérir des produits, Par planification*> forment une topologie *multi-segment*.

Le muli-segment présenté à la Figure 70 est noté par $MS_{ab} = \vee(ab1, ab3)$ où ab1 et ab3 sont des sections complémentaires.

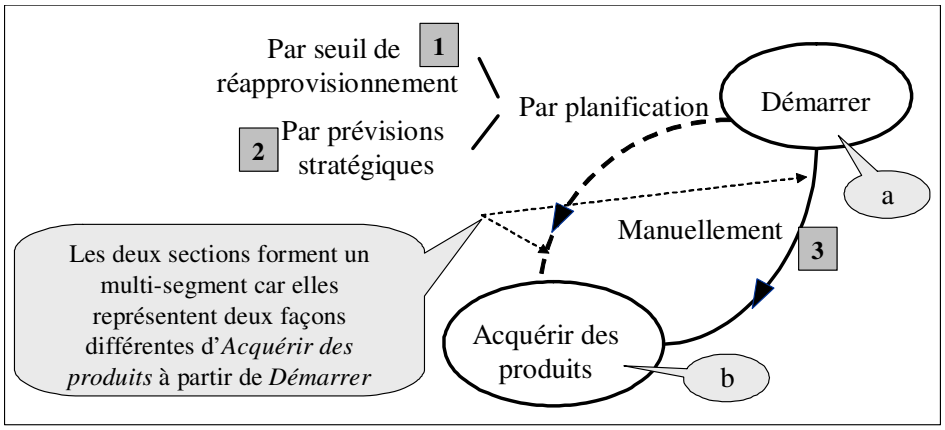


Figure 70. Exemple de multi-segment

2.1.6.3. Chemin

La relation entre les sections d'une carte établit quels buts précèdent ou succèdent tels autres. La relation de précédence indique qu'un but ne peut être réalisé que si un autre but a été réalisé. La relation de précédence conduit à ordonner les sections dans un *chemin*. Dans un chemin, le but cible de la section qui précède est le but source de la section qui lui succède. Un chemin est donc un sous-ensemble de sections de la carte où les sections sont reliées par une relation de précédence.

La relation de type chemin entre deux buts k et l est notée par $C_{k,Q,l}$ où Q désigne l'ensemble des buts intermédiaires utilisés pour la réalisation du but cible de code l à partir du but source de code k. Une relation de type chemin repose sur le lien de précédence « • » entre les sections et les relations entre elles.

Si, à partir d'un but source, il est possible d'atteindre un même but cible en suivant plusieurs chemins de la carte, on parle alors de topologie *multi-chemin*.

La relation de type multi-chemin entre deux buts k et l est notée: $MC_{k,Q,l}$ où Q désigne l'ensemble des buts intermédiaires utilisés pour la réalisation du but cible l à partir de du but source k. Une relation de type multi-chemin repose sur le lien union « ∪ » entre les chemins alternatifs.

Par exemple, à la Figure 67 et pour réaliser le but *Contrôler le stock* à partir du but *Démarrer*, il existe plusieurs chemins:

(1) *Acquérir des produits Manuellement*, puis *Contrôler le stock* en utilisant la stratégie *Entrer en stock les produits livrés* et enfin *Arrêter Par contrôle du paiement des factures* (cf. Figure 71). Ce chemin est noté par $C_{a,\{b\},c} = \bullet(ab_3, bc_1, cd_1)$.

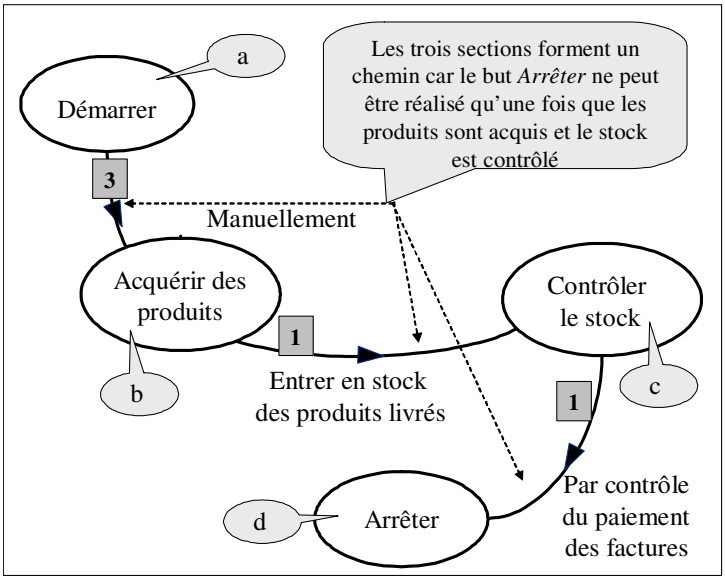


Figure 71. Relation multi-chemin – Premier chemin possible

(2) *Contrôler le stock* en utilisant la stratégie *Entrer les produits directement*, puis *Arrêter Par Contrôle du paiement des factures* (cf. Figure 72). Ce chemin est noté par $C_{a,\{c\},d} = \bullet(ac_1, cd_1)$.

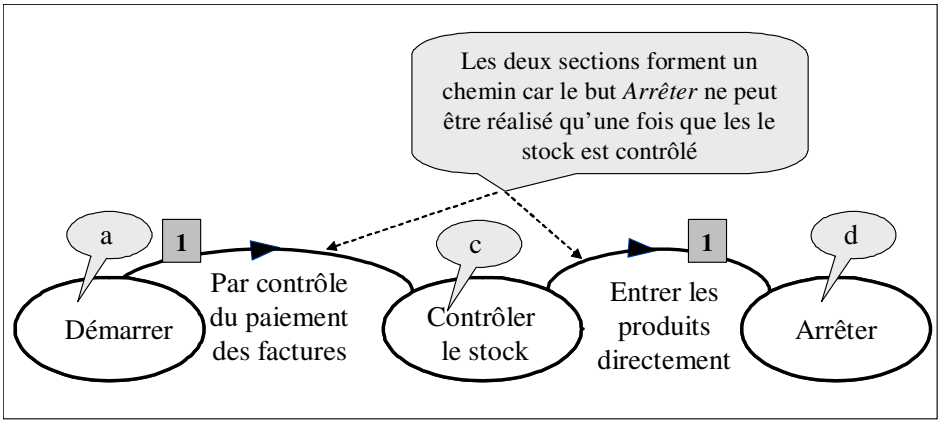


Figure 72. Relation multi-chemin – Deuxième chemin possible

Le multi-chemin qui permet de *Contrôler le stock* est noté par $MC_{a,\{b,c\},d} = \cup(C_{a,\{b\},c}, C_{a,\{c\},d})$.

Une carte intègre en général plusieurs chemins de *Démarrer* à *Arrêter*. C'est une structure multi-chemin et elle peut contenir des multi-segments. Chaque chemin décrit une manière d'atteindre le résultat. Il représente donc un processus, ce qui conduit à dire qu'une carte est un modèle multi-processus.

2.1.7. Lien d'affinement

Le méta-modèle de la carte de la Figure 66 précise qu'une section à un niveau i peut être affinée par une carte entière à un niveau d'affinement plus élevé i+1 (c'est à dire un niveau d'abstraction moins élevé). L'affinement produit ainsi une structure multi-chemin, multi-segment au niveau i+1. En conséquence, pour une section donnée du niveau i, non seulement : (i) la structure multi-segment décrit une alternative de sous sections au niveau i+1, mais aussi (ii) la structure multi-chemin introduit plusieurs combinaisons différentes de sous sections. Ainsi, l'affinement d'une section est une structure plus complexe qu'une simple décomposition ET/OU de buts.

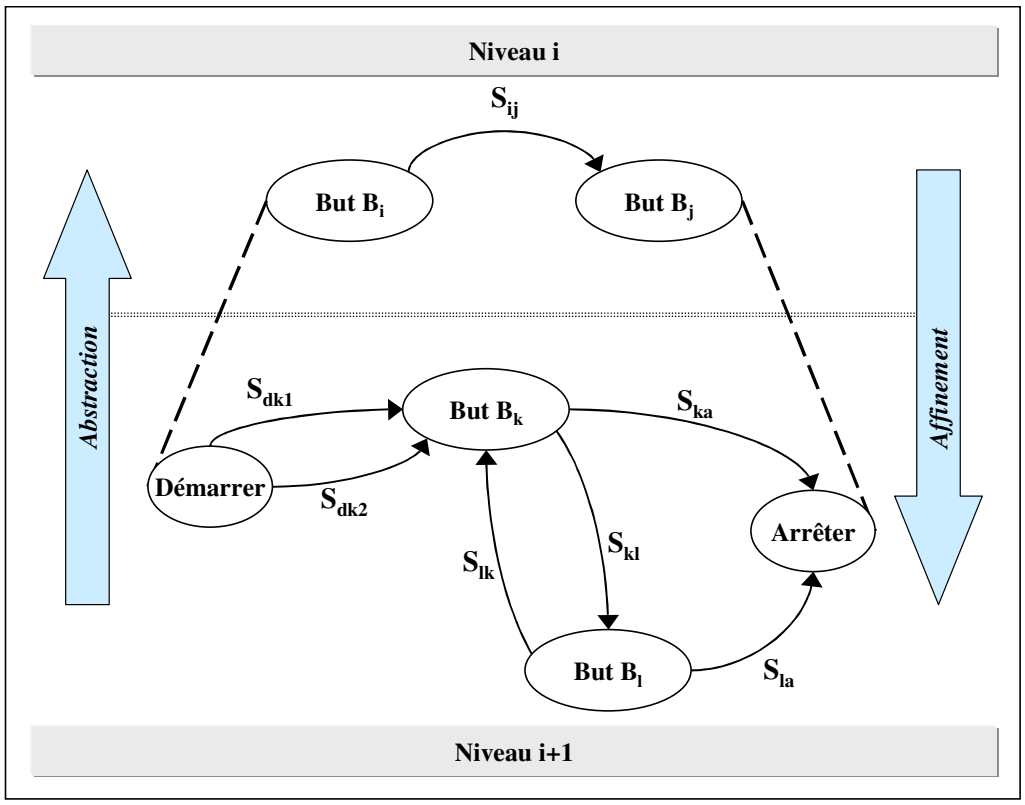


Figure 73. Affinement des sections

La Figure 73 montre les deux niveaux (i) et (i+1). La carte de niveau (i+1) propose plusieurs chemins entre *Démarrer* et *Arrêter*. Chacun de ces chemins est équivalent à la section du niveau (i). Le passage du niveau (i+1) au niveau (i) se fait par *abstraction*.

Une section ne peut être affinée que par une carte au maximum. Il est possible qu'une section ne soit pas affinée, on dit dans ce cas, qu'elle est opérationnalisable.

Le mécanisme d'affinement débouche ainsi sur une modélisation des besoins à plusieurs niveaux de détail. Par exemple, dans la carte de la Figure 68, on peut affiner la section <*Acquérir des produits, Contrôler le stock, Entrer en stock des produits livrés*> par la carte de la Figure 74. Cette carte est de nature plus opérationnelle. Par exemple, le choix au niveau i+1 du chemin *(<Démarrer, Accepter livraison, Par ajustement des quantités> ; <Accepter livraison, Entrer les produits en stock, Directement sur le lieu d'utilisation> ; < Entrer les produits en stock, Arrêter, Par stratégie de finalisation>*) est équivalent au choix de la section <*Acquérir des produits, Entrer les produits en stock, Entrer en stock des produits livrés* > du niveau i.

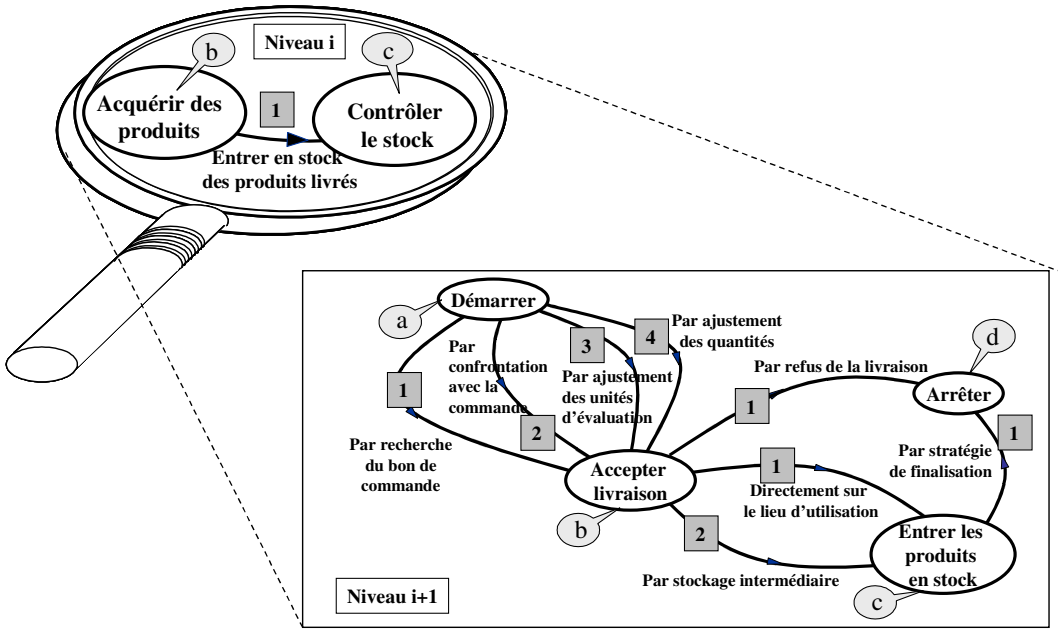


Figure 74. Affinement de la section <*Acquérir des produits, Contrôler le stock, Entrer en stock des produits livrés*>

Cette relation d'affinement entre sections et cartes conduit à un modèle prenant la forme d'une hiérarchie de cartes qui forme le modèle intentionnel. Ceci permet une modélisation à différents niveaux d'abstraction, offrant ainsi une lecture des processus allant du niveau le plus stratégique au niveau le plus opérationnel.

2.1.8. Conventions de codification d'une hiérachie de cartes

Manipuler un grand nombre de cartes sur plusieurs niveaux de détail est très difficile si l'on ne suit pas une codification précise de ces cartes. Nous adoptons l'une des conventions suivantes pour codifier les cartes : la codification locale ou la codification absolue.

Codification locale

- Chaque but est codé par une lettre de l'alphabet. Cela permet de manipuler plusieurs buts dans une même carte. A la Figure 74 par exemple, les quatre buts de la carte à savoir *Démarrer*, *Accepter livraison*, *Entrer les produits en stock* et *Arrêter* sont respectivement codés par a, b, c et d.
- Les stratégies sont numérotées relativement à leurs buts cible et source. Ainsi, deux stratégies ayant les mêmes buts source et cible seront respectivement numérotées par deux numéros distincts. Une carte peut donc avoir plusieurs stratégies numérotées de la même manière.

Exemple : à la Figure 74, les stratégies de la carte sont codées comme suit : 1, 2, 3 et 4 (ab) ; 1et 2 (bc) ; 1 (cd) ; 1(bd).

- Les sections sont codées par juxtaposition de trois éléments (i) la lettre du but source, (ii) la lettre du but cible et (iii) le numéro de la stratégie. Par exemple, la section ab_1 permet, en partant du but *a*, d'atteindre le but *b* en suivant la stratégie *1*.

La Figure 74 représente un exemple de codification d'une carte. Nous avons les huit sections suivantes : ab_1, ab_2, ab_3, ab_4, bc_1, bc_2, cd_1 et bd_1.

Codification absolue

Avec la codification absolue, la carte racine est appelée C. Ses buts sont nommés C.a, C.b… et ses sections $C.ab_1$, $C.ab_2$, $C.ab_3$… Si l'une de ces sections (par exemple dans la Figure 68 $C.bc_1$) est affinée par une carte, celle-ci a pour code $C.C_{bc1}$ (cf. Figure 74). Ses quatre buts (a, b, c et d) et huit sections (ab_1, ab_2, ab_3, ab_4, bd_1, bc_1, bc_2 et cd_1) sont alors codés de façon absolue comme suit :

- Les buts : $C.C_{bc1}.a$, $C.C_{bc1}.b$, $C.C_{bc1}.c$ et $C.C_{bc1}.d$.
- Les sections : C. $C_{bc1}.ab_1$, C. $C_{bc1}.ab_2$, C. $C_{bc1}.ab_3$, C. $C_{bc1}.ab_4$, C. $C_{bc1}.bd_1$, C. $C_{bc1}.cb_1$, C. $C_{bc1}.bc_2$, C. $C_{bc1}.cd_1$.

Ainsi le code C. $C_{bc1}.ab_2$ correspond à la section 2 entre le but *a* et le but *b* de la carte C_{bc1}.

La Figure 75 présente un exemple de liens entre les différentes cartes.

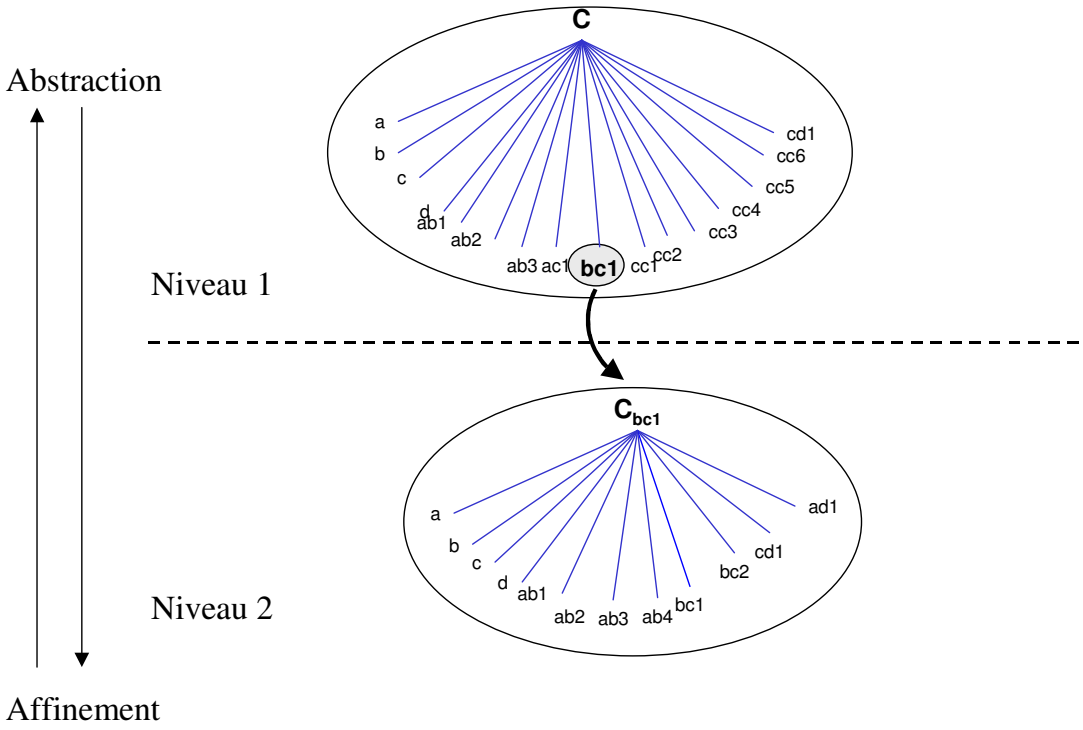


Figure 75. Exemple de liens entre cartes

2.1.9. Invariants et règles de validité de la carte

Pour que les cartes générées soient correctes, elles doivent respecter un certain nombre d'invariants et, pour qu'elles soient valides, elles doivent vérifier des règles de validité.

2.1.9.1. Invariants de la carte

Un *invariant* de la carte est une propriété que la carte doit toujours vérifier [Rolland04] [Banerjee87]. Nous avons identifié les six invariants suivants :

- **I1**. Toute carte a un et seulement un but qui n'est la cible d'aucune stratégie ; c'est le but *Démarrer*.
- **I2**. Toute carte a un et seulement un but qui n'est la source d'aucune stratégie ; c'est le but *Arrêter*.
- **I3**. Tout but dans une carte doit pouvoir se réaliser au moins une fois, c'est-à-dire qu'il existe un chemin qui le relie au but *Démarrer*.
- **I4.** Au sein d'une carte, tout but est unique.
- **I5.** Toute carte a un et un seul but qui peut être vu comme un ensemble vide d'états, le but *Démarrer*.
- **I6.** Dans une carte, une section est la source d'au plus un lien d'affinement.

A partir de ces six invariants, on constate plusieurs corollaires :

- **C1.** Une carte est un graphe connecté ; il n'y a ni but ni stratégie isolé.
- **C2.** Tout but dans une carte est la source d'au moins une stratégie sauf le but *Arrêter*.
- **C3**. Tout but dans une carte est la cible d'au moins une stratégie sauf le but *Démarrer.*
- **C4**. Il existe toujours un chemin de *Démarrer* à *Arrêter*.
- **C5**. Toute section appartient à au moins un chemin entre *Démarrer* et *Arrêter*.
- **C6.** Tout but se voit attribuer un nom différent de celui des autres buts de la carte.
- **C7.** Tout but est constitué d'au moins un état sauf le but *Démarrer*.
- **C8.** Toute section a sa pré-condition qui diffère de l'ensemble vide sauf les sections ayant pour source le but *Démarrer*.
- **C9.** Tout but correspond à un ensemble différent d'états.
- **C10**. Toute carte de niveau i+1 (i≠0) est la cible d'un lien d'affinement ayant pour source le but *Démarrer*.

2.1.10. Règles de validité de la carte

Afin que la carte soit valide, elle doit vérifier les règles suivantes [Rolland01] :

- **R1** : Les buts de la carte doivent être de même niveau d'abstraction.
- **R2** : Aucun but / stratégie de la carte ne doit pouvoir être considéré comme un sous-ensemble d'une autre.
- **R3** : Aucun but ne doit apparaître dans une carte comme une manière d'en réaliser un autre.
- **R4** : Les buts ayant pour résultat la même partie de produit doivent être agrégés.
- **R5** : Les sections représentant des manières mutuellement exclusives de produire un même résultat doivent être regroupées en paquet.
- **R6** : Les buts qui se complémentent mutuellement et vont de pair doivent être agrégés sous la forme d'un but unique qui les abstrait.
- **R7** : Les buts qui se complémentent mutuellement et vont ensemble, doivent être agrégés au sein d'un même but.

3. ETAPE 2 : GENERATION DU MODELE $\mathcal{MIS}$

Le but de cette étape est d'identifier les services intentionnels à partir du modèle de la carte. Nous rappelons que, d'après la typologie de services présentée au chapitre 3, il existe des services atomiques et des services agrégats.

Afin d'identifier ces services, nous proposons un processus structuré en trois sous étapes :

- Etape 2.1 : Définition des services atomiques.
- Etape 2.2 : Définition des services agrégats.
- Etape 2.3 : Définition des services relatifs à une section non opérationnalisable.

3.1. Etape 2.1 : Définition des services atomiques

L'une des caractéristiques du modèle de la carte est l'exploitation des connaissances du métier pour définir les exigences du système. La vue métier et la vue système ne s'opposent pas mais au contraire, doivent être alignées pour que le système d'information soit en concordance avec les besoins de ses futurs utilisateurs [Salinesi03]. Le modèle de la carte a été défini pour qu'une carte puisse être une expression des besoins du monde du business et en même temps, définisse les exigences de services du SI. Une carte a donc une double facette : elle est une expression du métier, de ses buts et des stratégies pour les atteindre mais elle détermine, en conséquence, les services du système qui sont aptes à supporter la réalisation de ces buts et stratégies. Afin d'établir un couplage direct entre les fonctionnalités exigées du système et les besoins du métier, *nous proposons d'associer à chaque section opérationnalisable de la carte un service atomique.*

Une *section opérationnalisable* dans une carte est une section qui a un niveau de détail qui ne permet pas de l'affiner par une autre carte. Pour chaque section opérationnalisable, nous associons un ou plusieurs services logiciels. Les événements servent à déclencher les opérations permettent de réaliser le but cible du service, en partant de son but source, conformément à sa stratégie.

Les services atomiques sont identifiés en appliquant la règle suivante :

R1. Associer à chaque section opérationnalisable de la carte un service atomique

En prenant comme exemple la carte *Satisfaire efficacement les besoins en produits* de la Figure 68, nous identifions onze services atomiques récapitulés dans le Tableau 20.

Tableau 20. Identification des services atomiques à partir de la carte *Satisfaire efficacement les besoins en produits*

Section de la carte	Service atomique
ab_1	S *Commander des produits par seuil de réapprovisionnement*
ab_2	S *Commander des produits par prévisions stratégiques*
ab_3	S *Commander des produits manuellement*

ac_1	S *Entrer des produits en stock directement*
cc_1	S *Transférer des produits*
cc_2	S *Valuer le stock*
cc_3	S *Inspecter la qualité du stock*
cc_4	S *Etablir un inventaire en continu*
cc_5	S *Etablir un inventaire par échantillonnage*
cc_6	S *Etablir un inventaire périodiquement*
cd_1	S *Contrôler le paiement des factures*

Dans la carte *Satisfaire efficacement les besoins en produits* présentée à la Figure 68, la section <*Acquérir des produits, Contrôler le stock, Entrer en stock des produits livrés* > (ayant le code bc_1) n'est pas une section opérationnalisable car elle s'affine par une carte entière à un niveau d'affinement plus élevé que celui de la carte *Satisfaire éfficacement les besoins en produits*. L'étape 2.1 (Définition des services atomiques) ne peut pas être appliquée dans ce cas. L'étape 2.3 relative à l'identification des services relatifs à une section non opérationnalisable (cf. section 3.3) sert dans ce cas.

3.2. Etape 2.2: Définition des services agrégats

Nous rappelons que d'après la typologie de services agrégats, que nous avons présenté au chapitre 3, il existe des services à variation et des services composites.

Afin de définir ces services, nous proposons les deux sous étapes suivantes :

- Etape 2.2.1 : Génération de tous les chemins de la carte,
- Etape 2.2.2 : Identification des services agrégats.

3.2.1. Etape 2.2.1 : Génération de tous les chemins de la carte

Les enchaînements des sections dans une carte identifient diverses sortes de services. Cependant, il est difficile d'identifier de manière exhaustive toutes les possibilités. Nous avons choisi d'utiliser l'algorithme de MacNaughton et Yamada [MacNaughton60] et de l'appliquer au modèle de la carte afin d'identifier tous les chemins possibles entre un but source *Démarrer* et un but cible *Arrêter*.

Durant cette étape, les chemins de la carte sont décrits en premier lieu par des formules mathématiques propres au formalisme de l'algorithme. Par la suite, des mises en correspondance sont appliquées à l'étape 2.2.2 afin de définir les types de services correspondants.

L'algorithme est initialement appliqué pour identifier tous les chemins possibles dans un automate fini. Nous expliquons l'application de cet algorithme dans notre cas pour définir tous les chemins possibles entre les buts ainsi que les relations entre les sections de la carte.

Introduction de l'algorithme de MacNaughton et Yamada

L'algorithme permet d'écrire la formule générale qui représente tous les chemins entre un nœud initial et un nœud final qui, dans notre cas, sont respectivement le but *Démarrer* et le but *Arrêter*.

- soient *s* et *t* les buts sources et cibles désignant respectivement le but *Démarrer* et le but *Arrêter*.
- soit *Q* l'ensemble des buts intermédiaires incluant *s* et *t*.
- soit *P* l'ensemble des buts intermédiaires entre *s* et *t* excluant ces derniers.

La formule initiale utilisée pour découvrir toutes les combinaisons possibles entre les buts *s* et *t* est la suivante :

(1) $Y_{s,\,Q,\,t} = \bullet(X^*_{s,\,Q\setminus\{s\},\,s}, X_{s,\,Q\setminus\{s,t\},\,t}, X^*_{t,\,Q\setminus\{s,t\},\,t})$, où

Le symbole "•" désigne l'opérateur de composition et le symbole "*" désigne l'opérateur d'itération.

La formule $Y_{s,Q,t}$ indique que l'ensemble de tous les chemins possibles entre le but source *s* et le but cible *t* sont les chemins du but *s* vers le but *s* en passant par l'ensemble *Q* des buts intermédiaires sans le but *s*, suivies de tous les chemins entre *s* et *t* sans passer par *s* et *t* et tous les chemins du but final *t* vers le but final *t* sans passer par *s* et *t*.

Chaque X désigne un ensemble de chemins possibles du but source vers le but cible en passant par un but intermédiaire quelconque. Sa formule est la suivante :

Étant donné un but particulier *q* de l'ensemble P, la formule générant l'ensemble des chemins possibles passant par un but intermédiaire *q* est la suivante :

(2) $X_{s,P,t} = \cup((X_{s,\,P\setminus\{q\},\,t}), \bullet(X_{s,\,P\setminus\{q\},\,q}, X^*_{q,\,P\setminus\{q\},\,q}, X_{q,\,P\setminus\{q\},\,t})$

Si $P = \varnothing$ alors $X_{s,P,t} = X_{s,t}$

La formule (2), définissant l'ensemble des chemins entre les buts *s* et *t* en passant par un but intermédiaire *q*, est l'union des chemins entre *s* et *t* sans passer par *q* ainsi que les chemins qui passent par *q*.

Chaque $X_{s,P,t}$ est développé à son tour à l'aide de la formule (2) en prenant au hasard un but intermédiaire de l'ensemble P. Le processus s'arrête lorsque l'ensemble P devient vide.

Nous illustrons dans ce qui suit, le déroulement de l'algorithme de MacNaughton et Yamada à base d'un exemple.

Application de l'algorithme

Considérons la carte décrite à la Figure 68 et l'ensemble *Q* des buts *{a, b, c, d}*, où *a* est le but initial et *d* le but final.

Par application de la formule initiale (1), nous obtenons la formule suivante décrivant tous les chemins entre le but *a* et le but *d* :

$$Y_{a,\{a,b,c,d\},d} = \bullet(X^*_{a,\{b,c,d\},a}, X_{a,\{b,c\},d}, X^*_{d,\{b,c\},d})$$

Afin de découvrir tous les chemins possibles en passant à chaque fois par un but intermédiaire quelconque de la carte, nous appliquons par la suite la formule (2). Nous obtenons ainsi les résultats suivants (cf. Tableau 21) :

Tableau 21. Développement des sous formules $X_{s,P,t}$ à partir de la formule initiale $Y_{a,\{a,b,c,d\},d}$

Formule
$X_{a,\{b,c,d\},a} = \cup(X_{a,\{b,d\}a}, \bullet(X_{a,\{b,d\}c}, X^*_{c,\{b,d\}c}, X_{c,\{b,d\}d}))$
$X_{a,\{b,c\},d} = \cup(X_{a,\{b\}d}, \bullet(X_{a,\{b\},c}, X^*_{c,\{b\},c}, X_{c,\{b\},d}))$
$X_{d,\{b,c\},d} = \cup(X_{d,\{b\},d}, \bullet(X_{d,\{b\},c}, X^*_{c,\{b\},c}, X_{c,\{b\},d}))$

Nous pouvons vérifier sur la carte qu'il n'existe pas de chemins possibles partant de *a* vers *a*, ni de chemins allant de *d* vers *d*. En conséquence, nous déduisons :

$$X_{a,\{b,c,d\},a} = \varnothing$$

$$X_{d,\{b,c\},d} = \varnothing$$

En revanche, il existe plusieurs chemins possibles entre *a* et *d* qui peuvent passer par les buts intermédiaires *{b,c}*. Nous appliquons ainsi à nouveau la formule (2) sur chacun des constituants de la formule suivante :

$$X_{a,\{b,c\},d} = \cup(X_{a,\{b\}d}, \bullet(X_{a,\{b\},c}, X^*_{c,\{b\},c}, X_{c,\{b\},d}))$$

Le Tableau 22 résume les développements de chacun des $X_{s,P,t}$.

Tableau 22. Développement des sous formules de la formule Xa,{b,c},d

Formule initiale	Formule finale[10]
$X_{a,\{b\}d} = \cup(X_{ad,} \bullet(X_{ab} ,X^*_{bb}, X_{bd}))$	$X_{a,\{b\}d} = \varnothing$
$X_{a,\{b\},c} = \cup(X_{ac,} \bullet(X_{ab},X^*_{bb},X_{bc}))$	$X_{a,\{b\},c} = \cup(X_{ac,} \bullet(X_{ab}, X_{bc}))$
$X_{c,\{b\},c} = \cup(X_{cc,} \bullet(X_{cb},X^*_{bb},X_{bc}))$	$X_{c,\{b\},c} = X_{cc}$
$X_{c,\{b\},d} = \cup(X_{cd,} \bullet(X_{cb},X^*_{bb},X_{bd}))$	$X_{c,\{b\},d} = X_{cd}$

L'algorithme s'arrête car nous n'avons que des formules de type X_{st} où l'ensemble P est vide.

La formule finale désignant tous les chemins entre le but initial et le but final de la carte s'écrit alors à l'aide de formules de type X_{st} où *s* et *t* désignent les buts de la carte.

Ainsi, la formule finale décrivant tous les chemins entre le but *a* et le but *d* est :

$$\mathbf{Y_{a,\{a,b,c,d\},d} = \bullet\ (\cup\ (X_{ac,}\ \bullet(X_{ab},\ X_{bc})),\ X^*_{cc},\ X_{cd})}$$

Cette formule nous indique que pour aller de *a* à *d*, il y a une séquence de chemins à suivre. En effet, il faut choisir au début l'un des deux chemins pour satisfaire le but *c* à partir du but *a* : soit aller de *a* à *c* directement (cette possibilité est décrite par la formule X_{ac}), soit aller de *a* à *b* puis de *b* à *c* (ce second chemin est exprimé par la formule $\bullet(X_{ab}, X_{bc})$). Ensuite, aller de *c* à *c* et enfin de *c* à *d*. Les chemins X_{cc} sont itératifs et non obligatoires.

Spécialisation des X_{st} en sections et relations entre elles dans une carte

A ce niveau, nous spécialisons les formules X_{st} obtenues en termes de sections de la carte et les relations entre elles à savoir : chemin, multi-chemin, paquet et multi-segment (cf. section 2.1.5). Ceci est mis en place en appliquant les règles suivantes :

- Affecter à chaque formule X_{st}, la section, le paquet ou le multi-segment correspondant.
- Une formule X_{st} est un chemin de la carte si elle ne correspond pas à une section, un paquet ou un multi-segment. Ce chemin est constitué d'un ensemble de formules reliées par un lien de séquence si l'ordre d'exécution est pré-défini ou bien un lien parallèle dans le cas contraire.

[10] Formule finale par élimination des X_{st} donnant lieu à l'ensemble vide

- Un ensemble de formules reliées par le symbole de composition « • » définit un chemin constitué de sections, paquets ou multi-segments reliés par un lien de séquence.
- Un ensemble de formules reliées par le symbole union « ∪ » définit un multi-chemin. Chaque formule, ou ensemble de formules, précédées du symbole « ∪ » peut être composée de sections, paquets, multi-segments et chemins.
- Une formule X_{st} itérative (ayant le symbole « * ») identifie par défaut une section, paquet, multi-segment, chemin ou multi-chemin itératif et optionnel sauf s'il y a décision pour les rendre obligatoire ou itératif.

Par application des règles énoncées précédemment sur la carte *Satisfaire efficacement les besoins en produits* (cf. Figure 68), nous obtenons les formules illustrées dans le Tableau 23 suivant :

Tableau 23. Liste des relations entre les sections de la carte *Satisfaire efficacement les besoins en produits*

Type de la relation	Relation identifiée
Paquet	$P_{ab}= \otimes(ab_1, ab_2)$ $P_{cc}= \otimes(cc_4, cc_5, cc_6)$
Multi-Segment	$MS_{ab}= \vee(P_{ab}, ab_3)$ $MS_{cc}= \vee(cc_1, cc_2, cc_3, P_{cc})$
Chemin	$C_{a,\{b\},c}= \bullet (MS_{ab}, bc_1)$ $C_{a,\{b,c\},d}= \bullet (MC_{a,\{b\},c}, MS_{cc}^{*}, cd_1)$
Multi-Chemin	$MC_{a,\{b\},c}= \cup (ac_1, C_{a,\{b\},c})$

Nous pouvons vérifier les résultats obtenus dans le Tableau 23 à partir de la carte *Satisfaire efficacement les besoins en produits*. Le chemin $C_{a,\{b,c\},d}$ définit toutes les compositions possibles de services entre le but source *Démarrer* (ayant le code *a*) et le but cible *Arrêter* (ayant le code *b*). Au début, il y a deux chemins pour satisfaire le but *c* à partir du but *a* qui sont décrits par le multi-chemin $MC_{a,\{b\},c}$. Le premier chemin consiste à choisir ac_1. Le deuxième chemin est $C_{a,\{b\},c}$ qui suggère d'atteindre le but *b* en choisissant une ou plusieurs possibilités du multi-segment MS_{ab} puis atteindre le but *c* en choisissant bc_1. L'étape suivante consiste à choisir une ou plusieurs possibilités du multi-segment MS_{cc} puis réaliser cd_1.

3.2.2. Etape 2.2.2 : Identification des services agrégats

Nous proposons l'ensemble des règles suivantes qui permettent d'identifier les types de services agrégats à partir des sections de la carte et les relations entre elles, identifiées à l'étape 2.2.1.

R2. Affecter à chaque chemin de la carte un service composite séquentiel.

R3. Affecter à chaque multi-chemin de la carte un service à variation de chemin.

R4. Affecter à chaque paquet de la carte un service à choix alternatif.

R5. Affecter à chaque multi-segment de la carte un service à choix multiple.

Dans le cas de la carte *Satisfaire efficacement les besoins en produits* (cf. Figure 68), les services agrégats sont résumés au Tableau 24.

Tableau 24. Liste des services agrégats générés à partir de la carte *Satisfaire efficacement les besoins en produits*

Type de la relation		**Service identifié**	
Paquet	$P_{ab}= \otimes(ab_1, ab_2)$	Service à choix alternatif	$S_{\textit{Acquérir des produits par planification}} = \otimes (S_{\textit{Commander des produits par seuil de réapprovisionnement}}, S_{\textit{Commander des produits par prévisions stratégiques}})$
	$P_{cc}= \otimes(cc_4, cc_5, cc_6)$		$S_{\textit{Etablir un inventaire physique}} = \otimes (S_{\textit{Etablir un inventaire physique en continu}}, S_{\textit{Etablir un inventaire physique par échantillonnage}}, S_{\textit{Etablir un inventaire physique périodiquement}})$
Multi-Segment	$MS_{ab}= \vee(P_{ab}, ab_3)$	Service à choix multiple	$S_{\textit{Acquérir des produits}} = \vee (S_{\textit{Commander des produits manuellement}}, S_{\textit{Acquérir des produits par planification}})$
	$MS_{cc}= \vee(cc_1, cc_2, cc_3, P_{cc})$		$S_{\textit{Contrôler le stock}} = \vee (S_{\textit{Etablir un inventaire physique}}, S_{\textit{Inspecter la qualité du stock}}, S_{\textit{Transférer des produits}}, S_{\textit{Valuer le stock}})$
Chemin	$C_{a,\{b,c\},d}= \bullet (MC_{a,\{b\},c}, MS_{cc}^{*}, cd_1)$	Service composite séquentiel	$S_{\textit{Satisfaire efficacement les besoins en produits}} = \bullet(S_{\textit{Recevoir le stock}}, S_{\textit{Contrôler le stock}}^{*}, S_{\textit{Contrôler le paiement des factures}})$
	$C_{a,\{b\},c}= \bullet (MS_{ab}, bc_1)$		$S_{\textit{Entrer les produits en stock normalement}}= \bullet(S_{\textit{Acquérir des produits}}, S_{\textit{Entrer en stock les produits livrés}})$
Multi-Chemin	$MC_{a,\{b\},c}= \cup (ac_1, C_{a,\{b\},c})$	Service à variation de chemin	$S_{\textit{Recevoir le stock}} = \cup (S_{\textit{Entrer les produits en stock directement}}, S_{\textit{Entrer les produits en stock normalement}})$

Le service agrégat qui permet d'acquérir les produits et de contrôler le stock est $S_{\textit{Satisfaire efficacement les besoins en produits}}$. Ce service est une composition de trois services : $S_{\textit{Recevoir le stock}}$, $S_{\textit{Contrôler le stock}}$ et $S_{\textit{Contrôler le paiement des factures}}$. Le premier service est un multi-chemin qui permet de recevoir les produits en stock. Le premier chemin consiste à traiter l'arrivée des produits

dans le stock dans le cas où le processus d'achat aurait été interrompu. Le deuxième chemin consiste en l'ajout de la marchandise en stock lorsqu'une livraison arrive.

Le deuxième service est un service à variation qui permet de s'assurer de la performance de la logistique des produits à travers la réalisation du but *Contrôler le stock*. Ceci consiste à :

1. établir l'inventaire des produits $S_{Etablir\ un\ inventaire\ physique}$,
2. inspecter la qualité des produits en stock $S_{Inspecter\ la\ qualité\ du\ stock}$,
3. réorganiser le stock $S_{Transférer\ des\ produits}$ et
4. Valuer les produits en enregistrant automatiquement la valeur du stock $S_{Valuer\ le\ stock}$.

Enfin, le troisième service $S_{Contrôler\ le\ paiement\ des\ factures}$ est de type atomique et permet de vérifier le traitement des factures par rapport aux bons de commandes et à la réception des produits, envoie ou bloque le paiement si nécessaire.

3.3. Etape 2.3 : Définition des services à partir d'une section non opérationnalisable

Dans une carte donnée, une section peut ne pas être opérationnalisable. Une *section non opérationnalisable* est une section qui est décrite par une carte complète. En d'autres termes, les buts de cette section sont d'un niveau d'abstraction trop élevé pour permettre l'opérationnalisation. Cela signifie en général qu'il existe différentes alternatives locales ou de chemins pour atteindre le but cible à partir du but source.

Pour détailler une section non opérationnalisable, nous utilisons le lien d'affinement (cf. section 2.1.6). Ce lien permet de décrire une section, non opérationnalisable à un niveau d'abstraction donné (i), par une carte à un niveau d'abstraction moins élevé (i+1) (c'est à dire une carte d'un niveau d'affinement plus élevé). L'affinement produit ainsi une structure multi-chemin, multi-segment au niveau i+1.

Une nouvelle carte est construite par conséquent. Afin d'identifier les services intentionnels à partir de cette carte, nous appliquons à nouveau le processus proposé à l'étape 2 (cf. section 3), mais sur la carte affinée et ainsi de suite jusqu'à n'obtenir que des services atomiques et des services agrégats.

Nous définissons les services atomiques ainsi que les services agrégats tels qu'ils ont été précédemment décrits à l'étape 2.1 et à l'étape 2.2.

L'étape 2.3 permet donc au processus d'être cyclique et récursif.

Le Tableau 25 illustre les résultats obtenus pour l'exemple de la carte présenté à la Figure 74. En effet, il représente la liste des services atomiques obtenue en associant à chaque section

opérationnalisable de la carte affinée le service atomique correspondant suite à l'application de la règle 1 de l'étape 2.1.

Tableau 25. Liste des services atomiques à partir de la carte de la Figure 74

Section de la carte	Service atomique
C. $C_{bc1}.ab_1$	S *Accepter les produits par recherche du bon de commande*
C. $C_{bc1}.ab_2$	S *Accepter les produits par confrontation avec la commande*
C. $C_{bc1}.ab_3$	S *Accepter les produits par ajustement des unités d'évaluation*
C. $C_{bc1}.ab_4$	S *Accepter les produits par ajustement des quantités*
C. $C_{bc1}.bd_1$	S *Refuser la livraison*
C. $C_{bc1}.bc_1$	S *Entrer des produits en stock directement*
C. $C_{bc1}.bc_2$	S *Entrer des produits par stockage intermédiaire*
C. $C_{bc1}.cd_1$	S *Finaliser la réception de produits*

Le Tableau 26 définit la liste des services agrégats obtenue par application de l'algorithme de MacNaughton et Yamada ainsi que les règles citées à l'étape 2.2.

Tableau 26. Liste des services agrégats

Type de la relation		Service identifié	
Paquet	$P_{ab}= \otimes(ab_1, ab_2, ab_3, ab_4)$	Service à choix alternatif	S *Accepter livraison* = ⊗ (S *Accepter les produits par recherche du bon de commande*, S *Accepter les produits par confrontation avec la commande*, S *Accepter les produits par ajustement des unités d'évaluation*, S *Accepter les produits par ajustement des quantités*)
	$P_{bc}= \otimes(bc_1, bc_2)$		S *Entrer les produits en stock* = ⊗ (S *Entrer des produits en stock directement*, S *Entrer des produits par stockage intermédiaire*)
Chemin	$C_{a,\{b,c\},d}= \bullet (P_{ab}, MC_{b,\{c\},d})$	Service composite séquentiel	S *Contrôler le stock* = • (S *Accepter la livraison*, S *Gérer la livraison*)
	$C_{b,\{c\},d}= \bullet (P_{bc}, cd_1)$		S *Stocker les produits* = • (S *Entrer les produits en stock*, S *Finaliser la réception de produits*)
Multi-Chemin	$MC_{b,\{c\},d}= \cup(bd_1, C_{b,\{c\},d})$	Service à variation de chemin	S *Gérer livraison* = ∪(S *Refuser la livraison*, S *Stocker les produits*)

Le service S *Contrôler le stock* définit toutes les compositions possibles de services entre le but source *Démarrer* et le but cible *Arrêter*. Ce service agrégat est une composition de deux services : S *Accepter la livraison* et S *Gérer la livraison*.

Le premier service S *Accepter la livraison* est un service à choix alternatif qui donne quatre moyens d'accepter une livraison :

- par recherche du bon de commande S *Accepter les produits par recherche du bon de commande*,
- par confrontation avec la commande S *Accepter les produits par confrontation avec la commande*,
- par ajustements des unités d'évaluation S *Accepter les produits par ajustement des unités d'évaluation* ou,
- par ajustement des quantités S *Accepter les produits par ajustement des quantités*.

Le deuxième service S *Gérer la livraison* est une composition de deux chemins qui permettent de gérer la livraison acceptée :

- Le premier chemin consiste à refuser la livraison en appelant le service atomique S *Refuser la livraison*.
- Le deuxième chemin consiste à stocker les produits en appelant le service composite S *Stocker les produits*. Ce dernier est une composition de deux services : un service à choix alternatif S *Entrer les produits en stock* qui consiste à choisir un moyen parmi deux alternatives proposées afin de réaliser le but *Entrer les produits en stock,* et un service atomique S *Finaliser la réception de produits*.

3.4. Récapitulatif

Nous proposons dans cette section une formalisation du processus de construction du modèle $\mathcal{MiS}$ par un algorithme correspondant à la méthode *GenerationMis()*.

La méthode *GenerationMis()* commence par générer l'ensemble des services atomiques et agrégats à partir d'une carte en appelant la méthode *IdentifierServices()*. L'étape 2.3 du processus (Définition des services à partir d'une section non opérationnalisable) est ensuite appliquée et consiste à appeler la méthode *IdentifierServices()* appliquée sur la carte de niveau i+1 correspondant à la section affinée.

L'algorithme ci dessous décrit la méthode *GenerationMis()* :

```
GenerationMis(Ci : carte) : ListeServicesIdentifies[]
   - Soit Ci la carte de niveau i fournie en entrée et à partir
     de laquelle la liste des services ListeServicesIdentifies
     est générée en sortie. Ci est composée d'un ensemble de
     sections.
   - Soit S la liste des services identifiés à partir de Ci.
```

```
   -  Soit Saff la liste des services identifiés à partir de Ci+1.

//Générer les services à partir de la carte Ci et les
//ajouter dans la liste des services identifiés.
  S <- IdentifierServices(Ci)
  AjouterService(ListeServicesIdentifies, S)
//Application de l'étape 2.3 du processus
  Pour chaque section ∈ Ci
   Si (section est non opérationnalisable) alors
      i<-i+1
      Saff<-IdentifierServices(Ci)
      AjouterService(ListeServicesIdentifies, Saff)
   Fin si
  Fin pour
```

La *GenerationMis()* appelle en particulier les méthodes suivantes:

- *AjouterService(ListeServicesIdentifies, service)* ajoute des services à la liste des services identifiés.
- *IdentifierServices(carte) : ListeServicesIdentifies* prend en entrée une carte de niveau i et retourne les services atomiques et/ou agrégats générés. L'algorithme de cette méthode est le suivant :

```
IdentifierServices(Ci : carte) : ListeServicesIdentifies[]
   -  Soit Ci la carte de niveau i fournie en entrée et à partir
      de laquelle la liste des services ListeServicesIdentifies
      est générée en sortie. Ci est composée d'un ensemble de
      sections.
   -  Soit Sat la liste des services atomiques identifiés à partir
      de Ci.
   -  Soit Sag la liste des services agrégats identifiés à partir
      de Ci.

      Pour chaque section ∈ Ci
//Définition des services atomiques en appliquant l'étape 2.1.
            Si (section est opérationnalisable) alors
                  Sat<- DéfinirServiceAtomique(Ci)
                  AjouterService(ListeServicesIdentifies, Sat)
            Fin si
      Fin pour
//Définition des services agrégats en appliquant l'étape 2.2.
      Sag<- DéfinirServiceAgrégat(Ci)
      AjouterService(ListeServicesIdentifies, Sag)
Fin pour
```

Rappelons que la méthode *DefinirServiceAtomique(carte) : ListeServicesIdentifies* prend en entrée une carte de niveau i et retourne les services atomiques générés. Cette méthode correspond à l'étape 2.1 du processus d'identification des services intentionnels.

Par ailleurs, la méthode *DefinirServiceAgregat(carte) : ListeServicesIdentifies* prend en entrée une carte de niveau i et retourne les services agrégats générés. Cette méthode correspond à l'étape 2.2 du processus d'identification des services intentionnels.

4. CONCLUSION

Le modèle intentionnel de services $\mathcal{MiS}$ est le produit du déroulement d'un processus de construction qui à partir des besoins des utilisateurs exprimés par une hiérarchie de cartes, identifie les services grâce à l'application de l'algorithme de MacNaughton et Yamada et d'un ensemble de règles.

Le choix de la carte comme modèle d'expression des besoins est motivé par l'utilisation d'un formalisme basé sur des buts, stratégies et des relations entre eux. Ces concepts sont en accord avec ceux du modèle $\mathcal{MiS}$.

L'utilisation du modèle de la carte dans la génération des services intentionnels met en avant les besoins des clients. Chaque section de la carte identifie un service. Les relations entre ces sections permettent d'exprimer la variabilité dans la manière d'atteindre le but d'un service via les topologies multi-segment, paquet et multi-chemin. Ceci permet de sélectionner le service qui correspond au mieux à la situation courante. Les relations entre les sections permettent aussi de composer les différents services via la topologie de type chemin.

La relation d'affinement dans une carte permet de décrire une section non opérationnalisable à un niveau d'abstraction donné (i), par une carte à un niveau d'abstraction moins élevé (i+1). Elle produit en conséquence des services intentionnels de niveau d'abstraction moins élevé.

CHAPITRE 6

Orchestration intentionnelle de services

1. INTRODUCTION

L'objectif de ce chapitre est de proposer une architecture qui permet une orchestration intentionnelle des services du modèle $\mathcal{MiS}$. Ceci est réalisé à l'aide d'un contrôle de la navigation dans une composition de services intentionnels et de la gestion de leur sélection à l'exécution.

Dans le cadre de l'architecture iSOA, le fournisseur doit mettre à la disposition de l'agent métier une architecture d'exécution des services appropriée à leur description intentionnelle. Nous répondons à cette demande par une *architecture iSOA à agent* permettant une exécution des services dirigée par les intentions et permettant l'adaptation de l'application aux conditions spécifiques du moment.

L'architecture iSOA à agents se base sur une hiérarchie d'agents qui contrôle le choix des services. La hiérarchie forme la couche de gestion des services intentionnels à différents niveaux de granularité.

L'orchestration intentionnelle, proposée dans ce chapitre, est guidée par les buts des clients. En effet, elle s'adapte aux besoins des clients à travers des sélections qu'ils initient. L'orchestration est, par conséquent, un processus contrôlé par le client qui effectue un choix entre plusieurs options de services. Le client interagit avec l'application afin de formuler explicitement ce qu'il attend de ce dernier. Ainsi, nous pouvons qualifier l'application résultante de personnalisable.

Dans ce qui suit, nous présentons à la section 2 le style d'architecture que nous adoptons dans ce chapitre : l'architecture à agents. Nous présentons à la section 3 l'architecture pour l'orchestration des services intentionnels. Nous détaillons par la suite les constituants de l'architecture à la section 4 et à la section 5. La section 6 est consacrée au processus de construction de la hiérarchie d'agents à partir d'un modèle $\mathcal{MIS}$. La section 7 met l'accent sur la manière dont chacun des contrôleurs définit les services adéquats compte tenu de la situation courante d'exécution. Enfin, nous concluons le chapitre à la section 8.

2. L'ARCHITECTURE A AGENTS

L'architecture que nous proposons repose sur deux principes directeurs : le concept d'agent et la décomposition récursive.

Avant de passer à la présentation des principes de l'architecture, nous faisons quelques clarifications sur la notion d'agent adoptée dans cette architecture.

2.1. La définition de la notion d'agent

La notion d'agent est issue du domaine de l'Intelligence Artificielle. Il n'existe pas, aujourd'hui, une définition universelle de l'agent. L'existence de plusieurs définitions d'un agent est due essentiellement à la multiplicité des facettes et caractéristiques que renferme ce concept. Chaque définition met en avant des aspects différents de l'agent. Elles sont généralement influencées par une vision particulière de l'agent ou par un contexte d'étude.

Plusieurs définitions d'un agent dans la littérature s'accordent à dire qu'il s'agit d'un composant logiciel ayant la capacité d'agir par lui-même (autonomie), de réagir aux stimuli externes (réactivité), de satisfaire ses propres objectifs (proactivité), d'adapter son comportement face aux demandes de changements (apprentissage, adaptation) et de coopérer avec d'autres agents si nécessaire pour réaliser leurs tâches (formant ainsi un système multi-agents) [Weib99] [O'Hare96] [Woolrige99]. Les agents sont créés et déployés dans des plates-formes agents.

Il existe plusieurs sortes d'agents et la classification la plus usuelle distingue les agents réactifs des agents cognitifs [Demazeau91]. Les premiers sont de type stimuli-réponse et ont

un comportement câblé [Drogoul92]. Les seconds, dotés de mécanisme de planification, poursuivent des buts de manière adaptative.

Dans ce mémoire, nous désignons par le terme d'agent des composants logiciels dont le comportement s'apparente à celui des agents réactifs, sans pour autant être implémentés selon une technologie agent ni déployé dans une plate-forme agent. Les agents, tout comme les composants logiciels, sont des entités logicielles modulaires qui peuvent être combinées selon plusieurs configurations afin de produire de nouvelles applications. Le terme agent est utilisé dans ce mémoire par référence au patron dont nous nous sommes inspirés pour définir l'architecture gérant les services à savoir, le patron d'architecture à agents.

2.2. Structure de l'architecture à agents

Comme le montre la Figure 76, l'architecture proposée structure l'application sous forme d'une hiérarchie d'agents. Il existe un agent racine, plusieurs agents intermédiaires et plusieurs agents feuilles. Chaque agent est responsable d'une tâche spécifique de l'application.

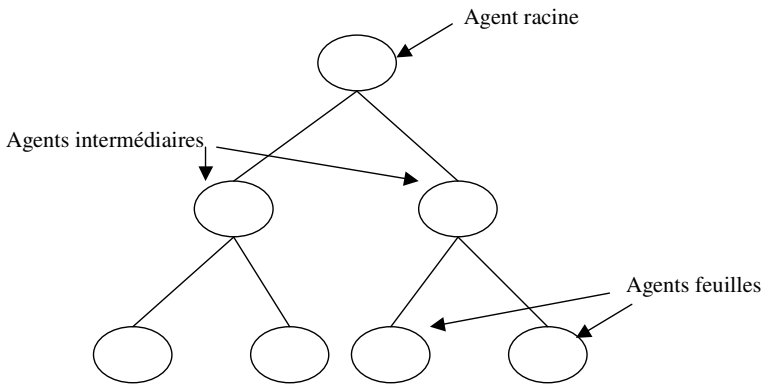


Figure 76. Structure de l'architecture proposée

L'agent racine gère toute la hiérarchie. Les agents feuilles sont des entités autonomes qui fournissent les fonctionnalités de l'application et avec lesquels les utilisateurs interagissent. Les agents intermédiaires gèrent des agents feuilles et/ou d'autres agents intermédiaires. Les liens entre l'agent père et ses fils symbolisent le flux de contrôle entre les deux parties. Les liens peuvent aussi être interprétés comme des relations de composition indiquant que l'agent père est composé de sous agents. Les utilisateurs peuvent être amenés à interagir avec les agents intermédiaires, ce qui implique la présence de la facette Présentation chez certains d'entre eux.

3. PRESENTATION GENERALE DE L'ARCHITECTURE POUR L'ORCHESTRATION DES SERVICES INTENTIONNELS

Afin d'implémenter les services du modèle $\mathcal{MIS}$, il convient de définir une hiérarchie de composants logiciels. Les principes généraux de l'architecture sont définis dans ce qui suit.

3.1. Principes généraux

L'architecture implémente les différentes sortes de services intentionnels en assignant un agent à chaque service identifié du modèle $\mathcal{MIS}$.

L'organisation hiérarchique d'agents est calquée sur les relations des services composites du modèle $\mathcal{MIS}$. En effet, chaque service atomique est implémenté par un agent distinct. Un service de choix alternatif ou multiple permettant de choisir un service intentionnel atomique parmi plusieurs s'implémente par un agent racine relié à chacun des agents représentant les services atomiques possibles. Cet agent racine permet de gérer le choix et l'exécution de l'agent choisi. L'agent racine correspond à un service de choix alternatif ou multiple selon la nature du lien entre les services atomiques.

La Figure 77 est un exemple de réalisation d'un service de choix alternatif sous forme d'agents.

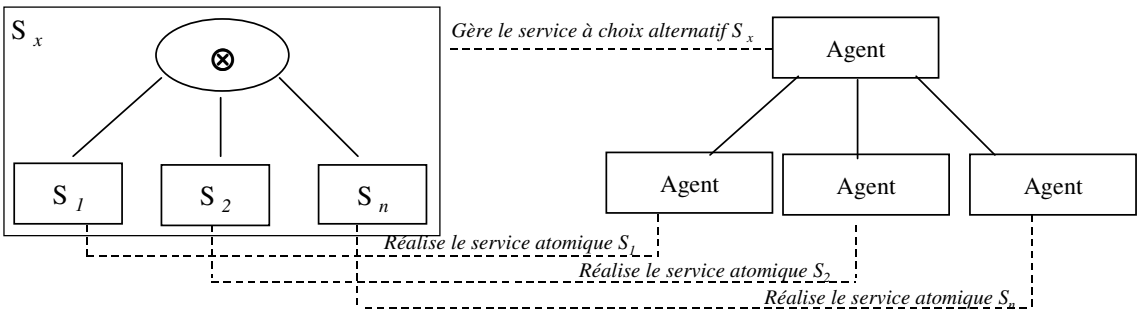


Figure 77. Réalisation d'un service de choix alternatif en agents

En appliquant la même approche, lorsque des services sont reliés les uns aux autres par un lien de composition, ils sont implémentés par un agent racine qui contrôle toute la composition et délègue le contrôle de chaque sous composition à un sous agent. Un sous agent peut, à son tour, être responsable d'autres agents si la composition qu'il contrôle est composée d'autres sous compositions. Ainsi, l'implémentation d'un service composite donne lieu à une hiérarchie d'agents reliés par des liens de délégation. La racine est l'agent qui contrôle toute la composition, plusieurs agents intermédiaires contrôlent les sous compositions et les feuilles sont les agents qui implémentent les services atomiques. La Figure 78 est un exemple de réalisation d'un service composite séquentiel en agents.

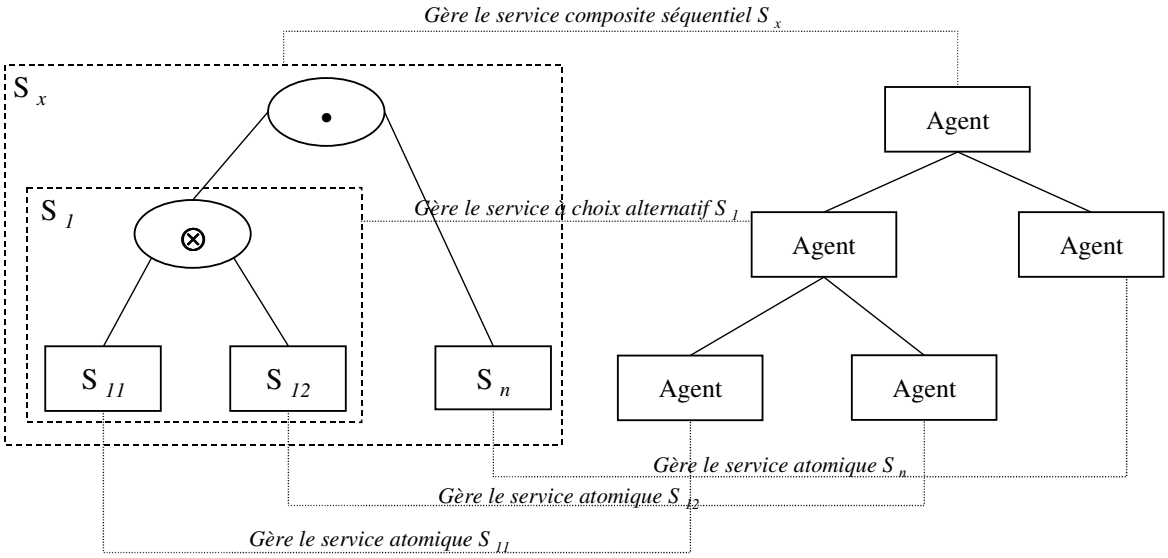


Figure 78. Réalisation d'un service séquentiel en agents

L'implémentation d'un service à variation de chemin suit le même principe de décomposition que celui d'un service composite, un agent racine contrôle le choix du chemin parmi tous les chemins possibles et l'exécution du chemin choisi par délégation. En effet, chaque chemin est contrôlé à son tour par un agent. Ce qui induit une hiérarchie d'agents. La Figure 79 illustre la réalisation d'un service à variation de chemin en agents.

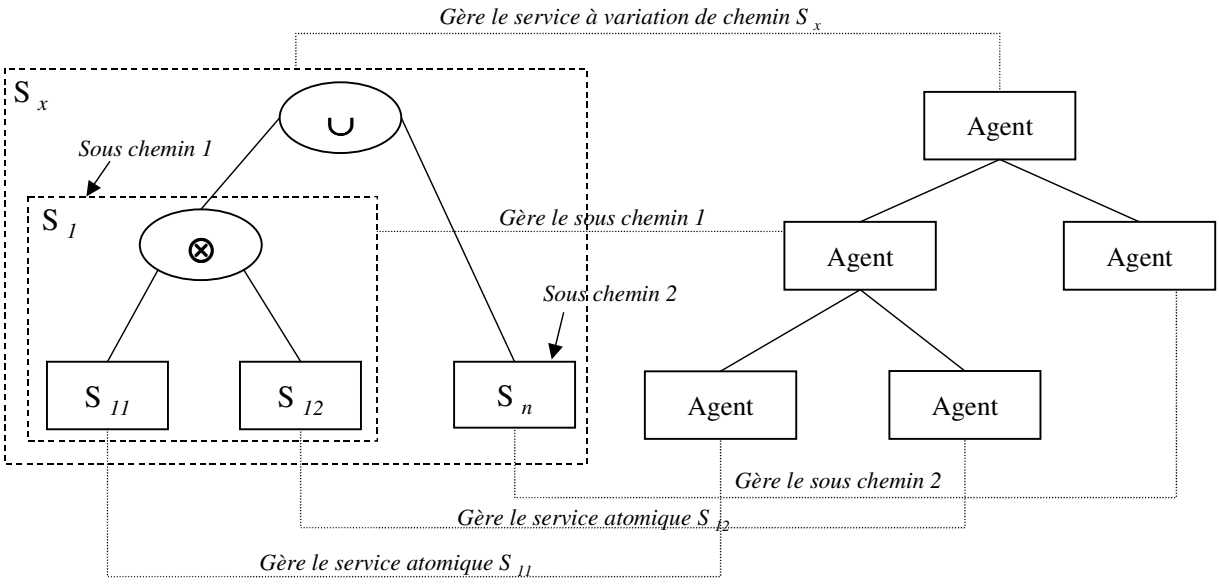


Figure 79. Réalisation d'un service à variation de chemin en agents

De manière générale, tout modèle $\mathcal{MiS}$ est soit un service à variation de chemin qui décrit tous les chemins possibles de services soit un service composite dans le cas où le modèle

définirait une seule composition. Le modèle *MiS* s'implémente ainsi sous la forme d'une hiérarchie d'agents.

3.2. Les catégories d'agents

Nous distinguons plusieurs catégories d'agents pour implémenter les services du modèle *MiS* qui sont représentés à la Figure 80.

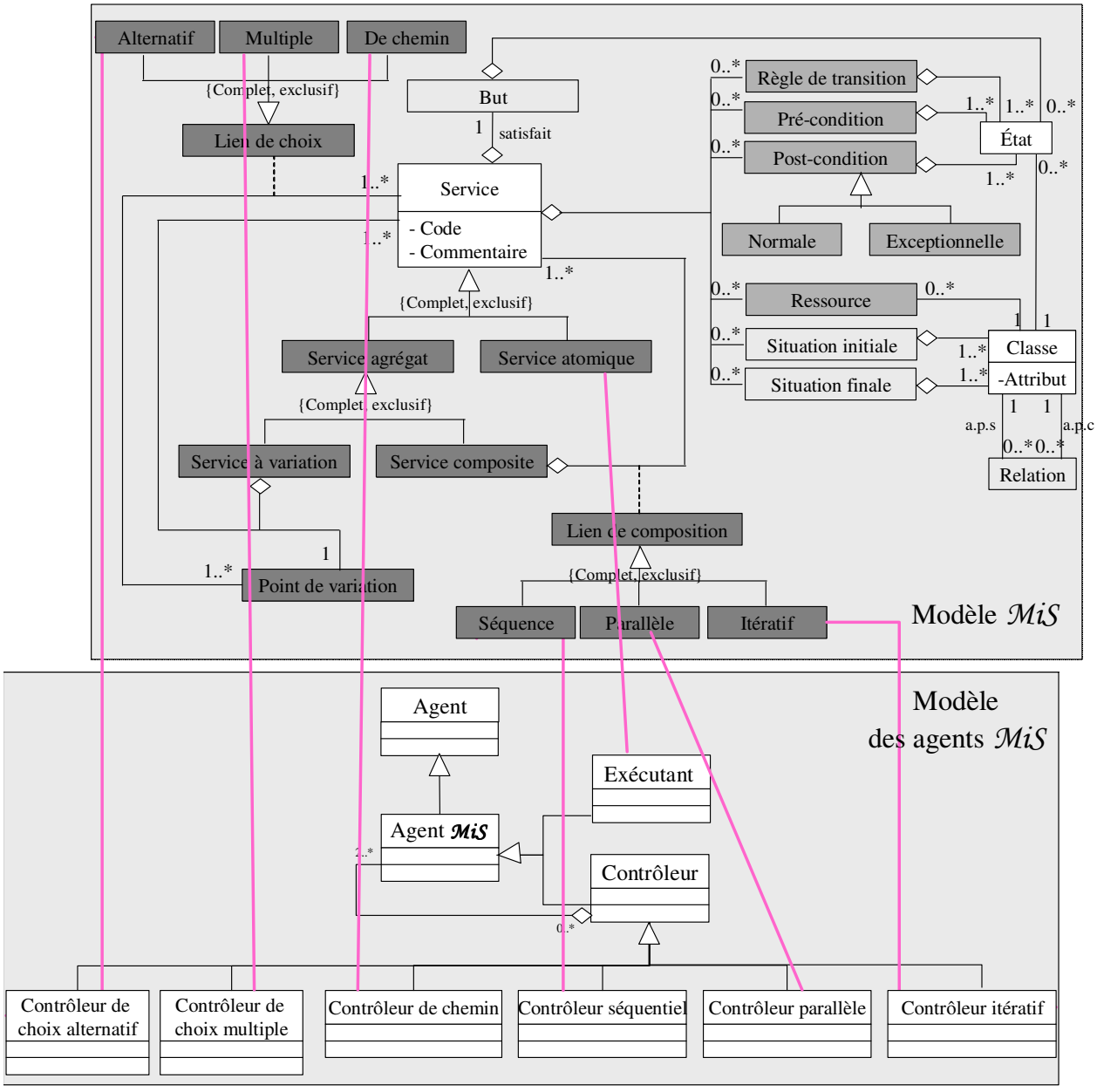


Figure 80. Les composants de l'architecture

Les agents sont classés en deux catégories : des exécutants et des contrôleurs.

- Les **exécutants** sont des entités élémentaires correspondant aux services intentionnels atomiques qui ont comme responsabilité l'invocation du service logiciel concrétisant le service intentionnel atomique qu'il représente.
- Les **contrôleurs** gèrent le choix et l'exécution des services logiciels indirectement à travers l'invocation d'exécutants ou bien d'autres contrôleurs.

Comme le montre la Figure 80, chaque type de service du modèle $\mathcal{MiS}$ est réalisé par un exécutant ou un type de contrôleur. Par exemple, un service atomique est implémenté par un exécutant. Par contre, un service de choix alternatif est implémenté par un contrôleur de choix alternatif, un service de choix multiple est réalisé par un contrôleur de choix multiple et un service à variation de chemin est implémenté par un contrôleur de chemin. Les deux premières sortes de contrôleurs gèrent uniquement des agents exécutants. Le contrôleur de chemin gère d'autres agents contrôleurs.

Un service composite séquentiel est géré par un contrôleur séquentiel. Un service composite parallèle est géré par un contrôleur parallèle et enfin un service composite itératif, constitué d'un lien de composition itératif, est implémenté par un contrôleur itératif. Ces trois sortes de contrôleurs contrôlent des agents exécutants et/ou d'autres agents contrôleurs.

Les exécutants et les contrôleurs sont organisés en plusieurs niveaux. Les exécutants sont les feuilles de la hiérarchie. Le niveau supérieur est constitué d'un seul agent contrôleur qui contrôle toute la hiérarchie. Plusieurs contrôleurs intermédiaires sont définis pour implémenter les liens entre les services dans le modèle $\mathcal{MiS}$. Les relations qui existent entre les divers agents sont des relations de délégation entre l'agent père et ses fils.

3.3. Notations

La Figure 81 définit le formalisme de représentation graphique de la hiérarchie d'agents. Les contrôleurs sont représentés par des rectangles et les exécutants par des cercles ovales. La nature du contrôle qu'exerce l'agent sur ses fils, à savoir s'il implémente un choix multiple, un choix alternatif, une séquence ou autre, est mentionné sur le graphique par des liens entre le père et ses fils et un symbole représentant la nature du contrôle. L'itération est indiquée par une * sur le lien relatif à l'agent itératif.

Pour les conventions d'appellation, par mesure de simplification, nous avons choisi de désigner les exécutants par le code du service atomique correspondant. Les contrôleurs sont identifiés par le code du service à variation ou composite correspondant relatif au lien implémenté par le contrôleur.

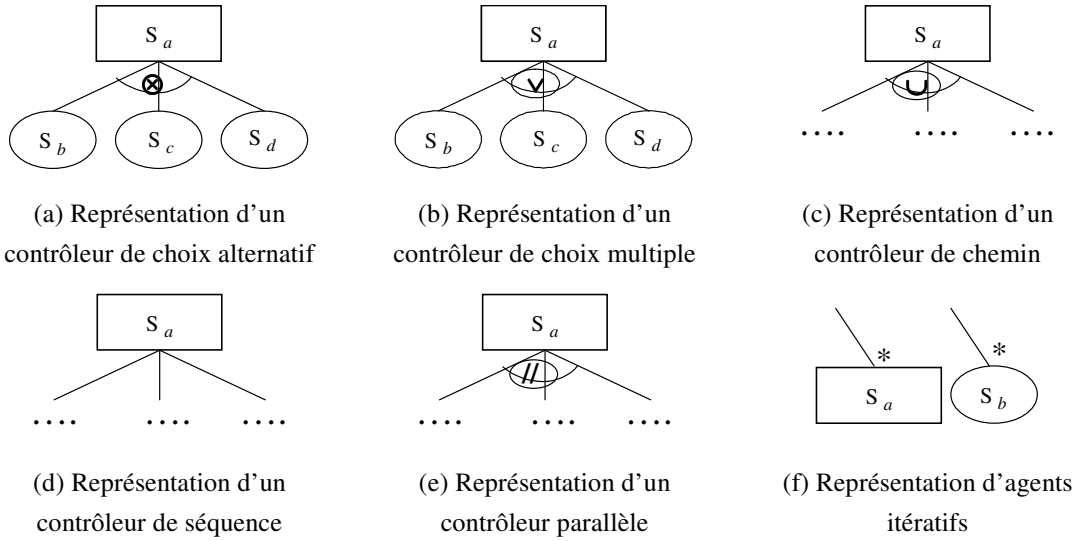


Figure 81. Représentation graphique des agents

3.4. Exemple de hiérarchie d'agents

La Figure 82 est un exemple de hiérarchie d'agents implémentant les services du modèle $\mathcal{MiS}$. Les services et leurs agents respectifs portent le même code.

Modèle $\mathcal{MiS}$ représentant le service S *Satisfaire efficacement les besoins en produits*

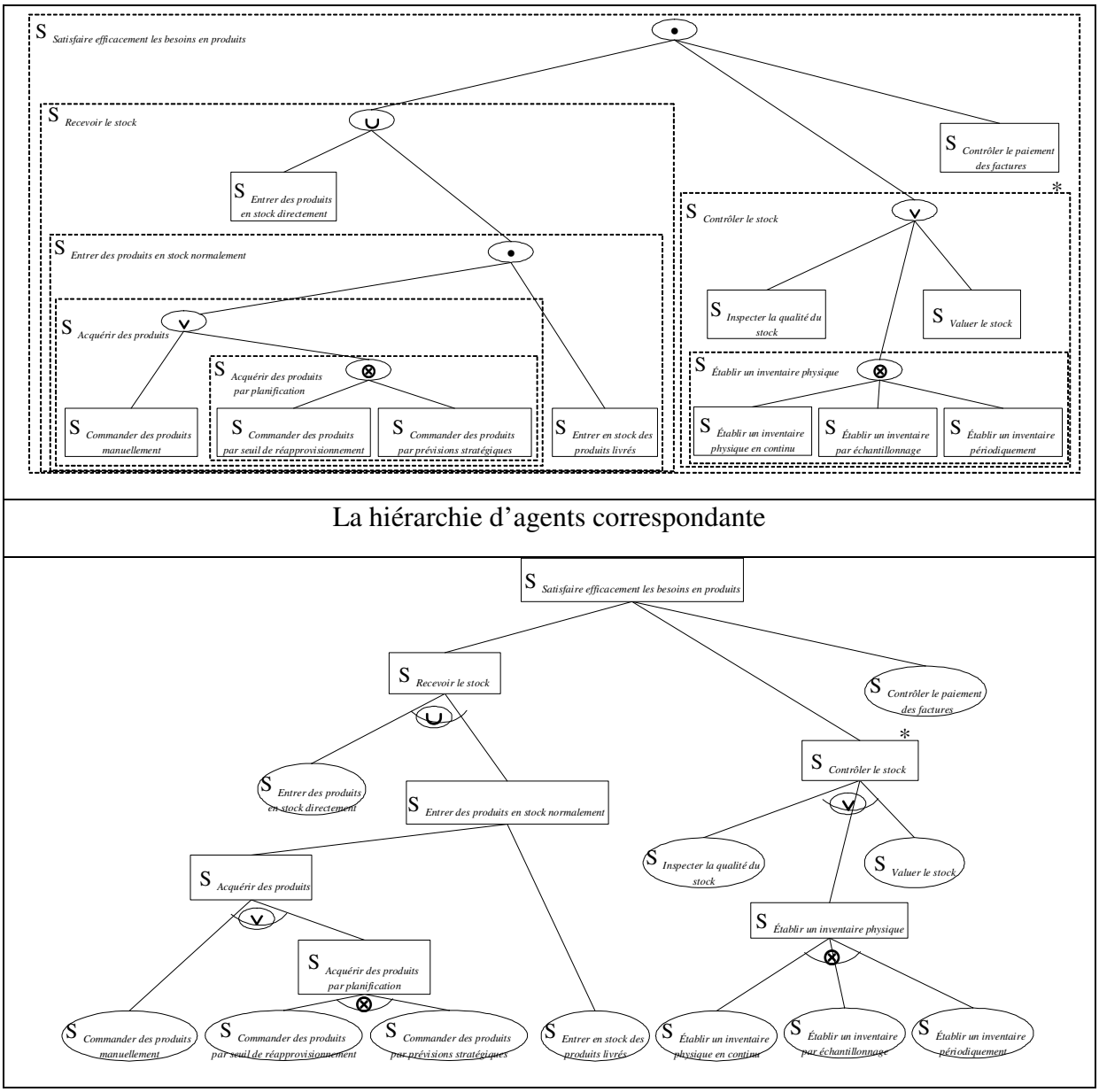


Figure 82. Exemple de hiérarchie d'agents

Le modèle $\mathcal{MiS}$ est composé d'un service composite séquentiel S *Satisfaire efficacement les besoins en produits* dont les sous services son reliés par un lien de composition séquentiel. Ce service est implémenté par un contrôleur de séquence qui est la racine de la hiérarchie. Le service S *Satisfaire efficacement les besoins en produits* est composé d'un service à variation de chemin S *Recevoir le stock*, d'un service de choix multiple S *Contrôler le stock* et d'un service atomique S *Contrôler le paiement des factures*.

Le service S *Recevoir le stock* est implémenté par un contrôleur de chemin. Le service S *Contrôler le stock* est réalisé par un contrôleur de choix multiple. Enfin, le service S *Contrôler le stock* est implémenté par un exécutant. Ces agents sont reliés à l'agent contrôleur père S *Satisfaire efficacement les besoins en produits* par des liens de délégation. Le service S *Recevoir le stock* est composé de

deux chemins, le premier constitué du service atomique S *Entrer des produits en stock directement* et le second est constitué d'un service composite séquentiel S *Entrer des produits en stock normalement*. Chacun des chemins est contrôlé par un agent, le premier par un agent exécutant et le second par un contrôleur de séquence. Les sous services du service S *Entrer des produits en stock normalement* à savoir le service de choix multiple S *Acquérir des produits* et le service atomique S *Entrer en stock des produits livrés* sont à leur tour implémentés respectivement par un contrôleur de choix multiple et un exécutant.

Le service S *Contrôler le stock* est implémenté par un contrôleur de choix multiple. Il est constitué de trois services : le service atomique S *Inspecter la qualité du stock* qui est représenté au niveau de l'architecture par l'exécutant ayant le même code, le service de choix alternatif S *Etablir un inventaire physique* qui est représenté par le contrôleur de choix alternatif et le service atomique S *Valuer le stock* qui est représenté par l'exécutant correspondant.

Enfin, le service S *Contrôler le paiement des factures* est représenté par l'exécutant ayant le même code.

3.5. Comportements des agents

La hiérarchie d'agents permet la navigation à travers les services du modèle $\mathcal{MIS}$. Le choix d'un service revient à satisfaire un besoin de l'utilisateur.

Pour décrire le comportement des agents de la hiérarchie, nous avons choisi de le faire d'abord sur un exemple. Les détails sur le comportement des agents sont fournis aux sections 4 et 5 de ce chapitre.

La Figure 83 est la carte à partir de laquelle le modèle $\mathcal{MIS}$ de la Figure 82 est construit. Cette carte a été présentée en détail dans le chapitre 5 de ce mémoire.

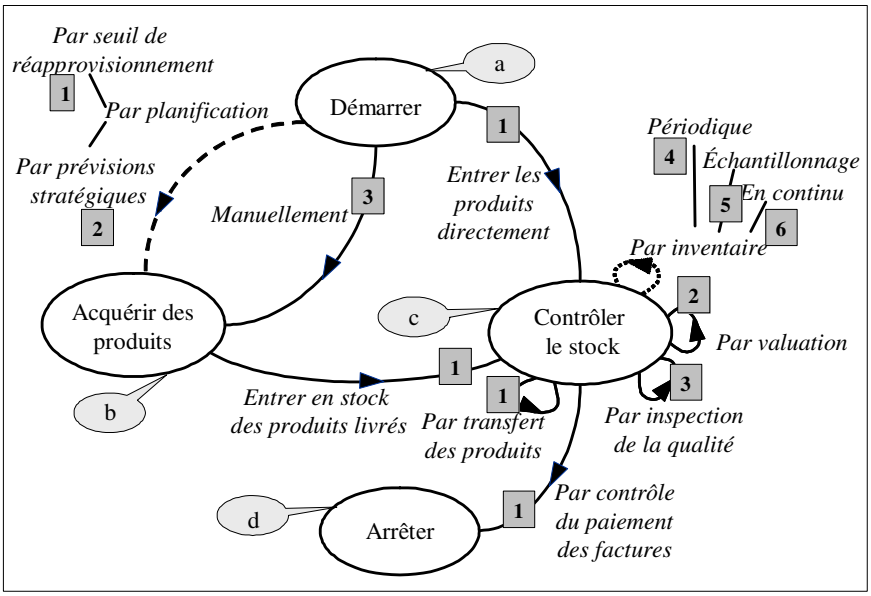


Figure 83. La carte : *Satisfaire efficacement les besoins en produits* relative au modèle $\mathcal{M}_{IS}$ de la Figure 82

Nous allons à présent détailler chacune des étapes de réalisation des buts.

Réalisation du but *Contrôler le stock* (ayant le code c)

D'après le modèle $\mathcal{M}_{IS}$, la satisfaction du but *Contrôler le stock* de code c passe par le choix d'un des deux chemins proposés : aller directement de *a* vers *c* (en choisissant le service atomique $S_{\textit{Entrer des produits en stock directement}}$) ou bien passer par le but *b* (en choisissant le service composite séquentiel $S_{\textit{Entrer des produits en stock normalement}}$). Ceci se traduit au niveau de l'architecture par le comportement des agents suivants :

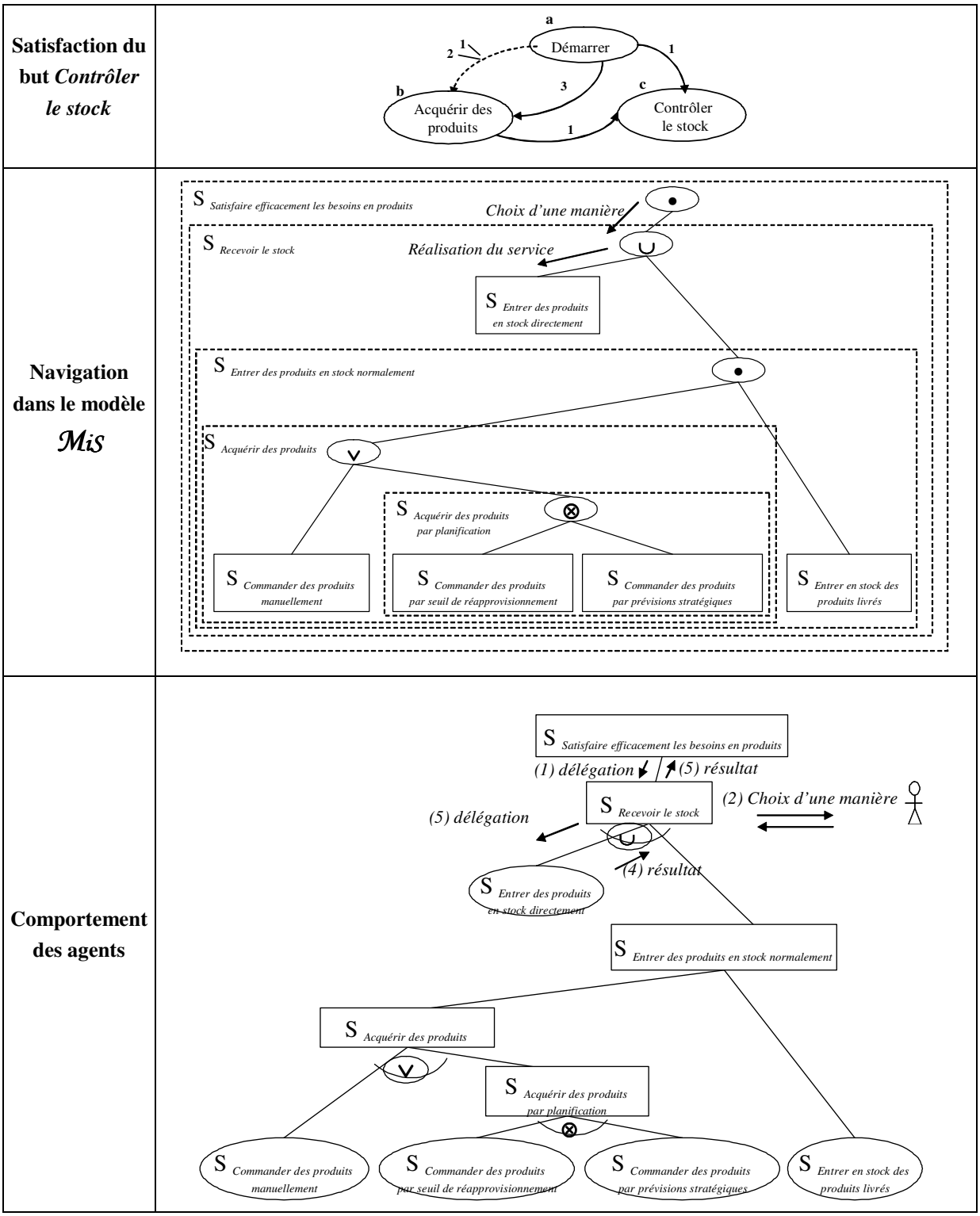


Figure 84. Réalisation du but *Contrôler le stock*

(1) l'agent racine S *Satisfaire efficacement les besoins en produits* qui contrôle la satisfaction du but d à partir du but a délègue la satisfaction du but c au contrôleur de choix multiple S *Recevoir le stock*.

(2) le contrôleur de choix multiple S *Recevoir le stock* propose à l'utilisateur de choisir une manière de satisfaire le but c.

(3) l'agent exécutant S *Entrer des produits en stock directement* ou le contrôleur S *Entrer des produits en stock normalement*, implémentant la manière choisie, est invoqué par le contrôleur S *Recevoir le stock*.

(4) dans le cas où l'exécutant serait choisi, il retourne son résultat à son agent père.

(5) ce dernier ayant atteint le but c, il retourne aussi son résultat à son agent père S *Satisfaire efficacement les besoins en produits* afin que ce dernier puisse continuer la progression vers la satisfaction du but d.

Réalisation du but *Arrêter* (ayant le code d)

A la suite de la réalisation du but c, le modèle $\mathcal{MiS}$ indique par le symbole « • », que la prochaine étape consiste à progresser vers le but c ensuite vers le but d (cf. Figure 85).

Selon le modèle $\mathcal{MiS}$, la progression vers le but c à partir du but c consiste à choisir le service de choix multiple et itératif S *Contrôler le stock* (le nombre d'itérations est compris entre zéro à n fois). L'exécution de S *Contrôler le stock* consiste à choisir au moins un service parmi les trois proposés à savoir S *Inspecter la qualité du stock*, S *Etablir un inventaire physique* et S *Valuer le stock*.

La progression vers le but d à partir du but c consiste à choisir le service atomique S *Contrôler le paiement des factures*.

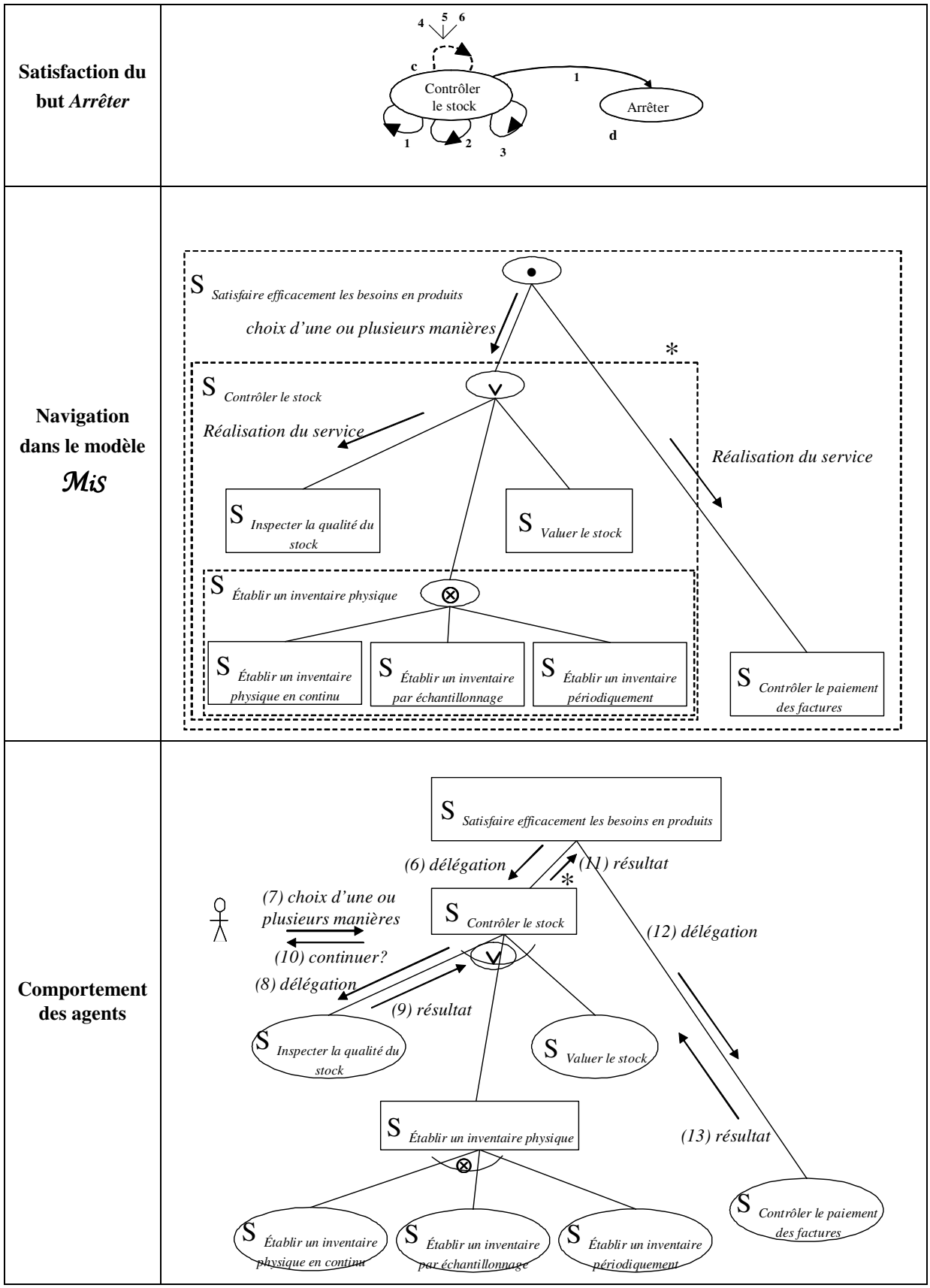


Figure 85. Réalisation du but *Arrêter*

Au niveau de l'architecture, le contrôleur racine identifie le prochain fils à invoquer compte tenu de la situation qui est la réalisation du but c. Il existe deux candidats qui permettent la progression à partir du but c qui sont le contrôleur de choix multiple itératif S *Contrôler le stock* et l'exécutant S *Contrôler le paiement des factures*. Le comportement suivant est par la suite réalisé pour atteindre le but c puis d (cf. Figure 85) :

(6) l'agent racine S *Satisfaire efficacement les besoins en produits* délègue la satisfaction du but c à partir de c au contrôleur de choix multiple S *Contrôler le stock*.

(7) le contrôleur de choix multiple S *Contrôler le stock* propose à l'utilisateur de choisir une ou plusieurs possibilités de progression vers le but c.

(8) en supposant que l'utilisateur a choisi d'aller de c vers c, l'exécutant implémentant cette possibilité S *Inspecter la qualité du stock* est invoquée par le contrôleur S *Contrôler le stock*.

(9) l'agent exécutant retourne le résultat à son agent père.

(10) le contrôleur S *Contrôler le stock* propose à l'utilisateur soit d'explorer d'autres possibilités de réalisation du but c soit d'arrêter.

(11) l'utilisateur décide d'arrêter l'exécution, le contrôleur S *Contrôler le stock* retourne le résultat à son père.

(12) l'agent racine S *Satisfaire efficacement les besoins en produits* délègue la satisfaction du but d à partir de c à l'exécutant S *Contrôler le paiement des factures*.

(13) l'exécutant S *Contrôler le paiement des factures* retourne le résultat à son agent père.

De manière générale le comportement d'un exécutant se résume à :

- L'invocation d'une fonctionnalité.
- La récupération des résultats de la fonctionnalité et leur transmission vers l'agent père.

Le comportement d'un contrôleur indépendamment de son type consiste en :

- Des actions de délégations (invocation) de l'agent père vers ses fils.
- Lorsque l'agent fils termine son exécution, il remonte les résultats obtenus à son père.
- L'agent père détermine les prochains fils candidats ou bien arrête son exécution soit suite à une décision de l'utilisateur ou parce qu'il a atteint son but.

La spécificité de chacun des agents se situe au niveau de la manière de satisfaction du but du service qu'il gère.

Un *exécutant* satisfait son but par l'invocation d'une fonctionnalité.

Un *contrôleur de choix multiple* atteint son but par l'exécution d'un ou plusieurs exécutants et/ou contrôleurs qu'il contrôle. Le contrôleur propose à l'utilisateur de choisir un des fils

dont il a la charge. L'utilisateur sélectionne un agent. Ce dernier est invoqué par le contrôleur. Lorsque l'agent finit son exécution, il retourne le résultat à son père qui redéfinit de nouveau les agents fils candidats et les propose à l'utilisateur. Le processus de satisfaction du but d'un contrôleur de choix multiple s'arrête lorsque l'utilisateur a décidé d'arrêter ou bien lorsqu'il n'y a plus de candidats à invoquer.

Un *contrôleur de choix alternatif*, contrairement à un contrôleur de choix multiple, atteint son but par l'exécution d'un seul exécutant parmi tous ses fils. Dans l'exemple de la Figure 84, le contrôleur S *Acquérir des produits par planification* atteint le but *Acquérir des produits* (ayant le code b) par le choix d'un des exécutants S *Commander des produits par seuil de réapprovisionnement* ou S *Commander des produits par prévisions stratégiques*.

Un *contrôleur de choix multiple* atteint son but cible par l'exécution d'un ou plusieurs des sous chemins qu'il contrôle. Il termine son exécution lorsque l'utilisateur décide d'arrêter ou s'il n'y a plus d'agents fils à invoquer. Dans l'exemple de la Figure 84, le contrôleur S *Recevoir le stock* satisfait le but cible *Contrôler le stock* (ayant le code c) en proposant à l'utilisateur de choisir un agent parmi S *Entrer des produits en stock directement* ou S *Entrer des produits en stock normalement*.

Un *contrôleur de séquence* atteint son but cible par la satisfaction séquentielle de buts intermédiaires jusqu'à atteindre le but final. Les buts intermédiaires sont contrôlés à leur tour par d'autres contrôleurs ou bien par des exécutants. Par exemple, le contrôleur de séquence S *Entrer des produits en stock normalement* réalise dans un premier temps le but *Acquérir des produits* (de code b) en invoquant le contrôleur de choix multiple S *Acquérir des produits* puis réalise le but *Contrôler le stock* (de code c) en invoquant l'exécutant S *Entrer en stock des produits livrés*. Le contrôleur de séquence termine son exécution lorsqu'il a atteint son but cible ou lorsque l'utilisateur suspend son exécution.

Un *contrôleur parallèle*, à l'opposé d'un contrôleur de séquence, invoque ses agents fils selon un ordre non défini à l'avance. C'est l'utilisateur qui décide de l'agent à invoquer.

Enfin, un contrôleur itératif invoque le même agent fils sur des situations d'entrée différentes. L'itération se termine lorsque l'ensemble des situations dérivées de celle de l'agent itératif est traité.

Les spécificités de chaque type de contrôleur sont détaillées plus loin dans ce chapitre. Dans ce qui suit, nous nous limitons à la définition du comportement générique des contrôleurs et des exécutants.

4. SPECIFICATION CONCEPTUELLE D'UN CONTROLEUR

Conceptuellement, un contrôleur est considéré comme une entité cohésive qui a comme objectif la satisfaction d'un but. La Figure 86 représente notre vision d'un contrôleur. Un contrôleur est conceptuellement formé de cinq parties décrites dans les sections suivantes.

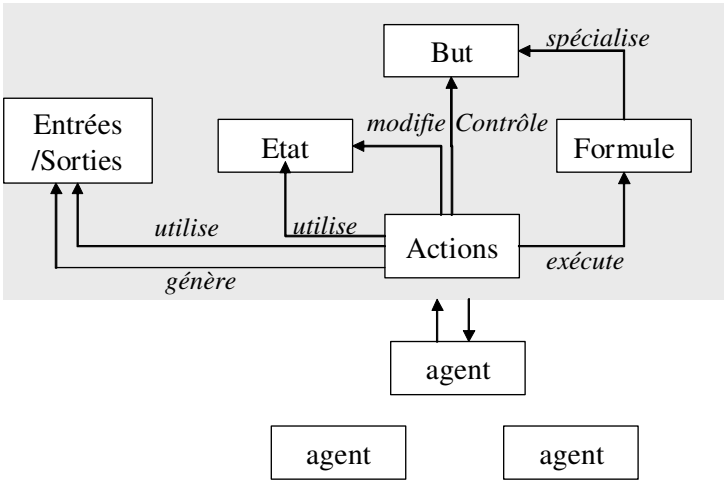


Figure 86. Spécification d'un contrôleur

4.1. Le but

Le **but** est l'objectif que le contrôleur désire atteindre, qui est le contrôle de la satisfaction du but du service implémenté par l'agent, compte tenu d'une situation initiale. Il représente le rôle du contrôleur dans la hiérarchie.

Le but du contrôleur est formulé selon les règles de structuration proposées par [Prat97] [Prat99] à savoir un verbe suivi d'un ou de plusieurs paramètres.

La structure du but est la suivante :

(Contrôler)$_{\text{verbe}}$ (la satisfaction du but <but >)$_{\text{cible}}$

Le verbe *Contrôler* caractérise l'objectif du contrôleur, à savoir le contrôle de l'exécution. La *cible* du verbe précise ce que le contrôleur contrôle, ce qui est la satisfaction du *but* du service correspondant.

Comme exemple de formulation de but, prenons le cas d'un contrôleur de choix alternatif correspondant à un service à choix alternatif permettant d'atteindre le but du service S $_{\textit{Acquérir des produits par planification}}$ par le choix de l'une des stratégies *par réapprovisionnement* ou *par prévisions stratégiques* (voir l'explication dans l'exemple à la section 2.1.6.1 du Chapitre 5).

Le but du contrôleur correspondant est formulé comme suit :

(Contrôler)$_{\text{verbe}}$ (la satisfaction du but Acquérir des produits par planification)$_{\text{cible}}$

4.2. Les actions

Un agent définit l'ensemble des **actions** à exécuter pour atteindre le but de l'agent. Les actions consistent à déterminer les agents candidats à l'exécution, invoquer un agent particulier ou bien demander à l'utilisateur de choisir l'agent qu'il souhaite exécuter.

Un agent contrôleur réagit à des stimuli externes ou encore appelés évènements externes, qui permettent de le faire passer d'un état cohérent vers un autre et, par conséquent, assurent la progression dans la satisfaction de son but. Les évènements externes proviennent de l'environnement de l'agent, c'est à dire à partir de l'utilisateur ou bien de son agent père ou encore des agents fils qu'il gère (cf. Figure 87.).

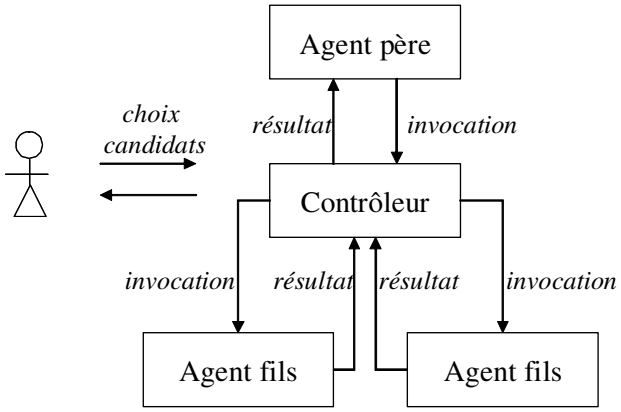


Figure 87. Interaction de l'agent avec son environnement

Les évènements en provenance de l'utilisateur sont des sélections d'agents fils à invoquer. Ceux en provenance de l'agent père sont des demandes d'exécution du contrôleur lui-même auxquels il doit répondre par l'envoi de résultats à la fin de son exécution. Le contrôleur peut lui-même envoyer des évènements vers ses fils pour les invoquer et utilise leurs résultats pour poursuivre son exécution. Le contrôleur peut aussi émettre des évènements vers ses constituants (état, entrées/sorties, …). Ces événements sont appelés évènements internes.

Nous allons à présent nous focaliser sur le comportement du contrôleur face à l'arrivée de stimuli externes de la part d'autres agents ou de l'utilisateur.

La Figure 88 identifie les actions réalisées par l'agent contrôleur.

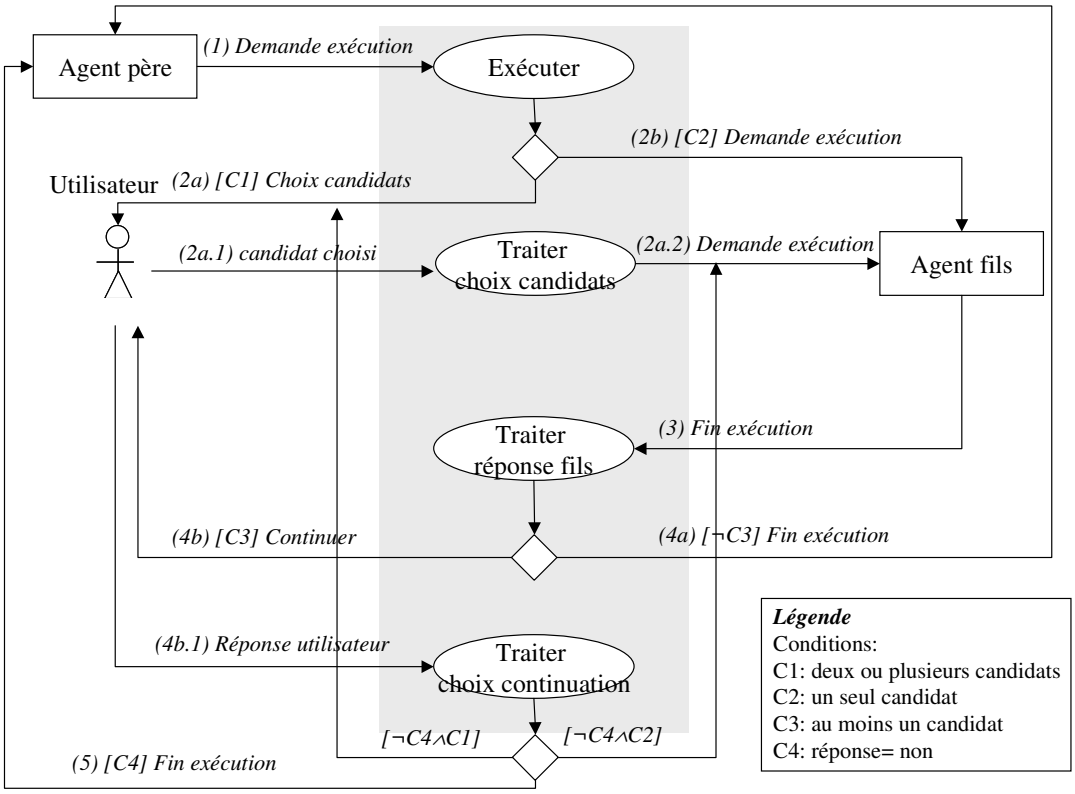


Figure 88. Traitement des stimuli externes par un contrôleur

(1) À l'arrivée d'une demande d'exécution de la part de son agent père, l'action *exécuter* est réalisée. Elle consiste à déterminer les agents fils candidats à invoquer. Deux cas se présentent :

(2b) S'il existe un seul candidat (condition C2), l'action *exécuter* invoque directement l'agent candidat fils.

(2a) S'il existe plusieurs candidats (condition C1), l'agent demande à l'utilisateur de choisir un candidat et celui-ci envoie son choix.

(2a.1) A l'arrivée du choix de l'utilisateur, l'action *traiter Choix Candidat* est exécutée. Elle aboutit à l'invocation de l'agent fils choisi.

A la réception de la notification de fin d'exécution de l'agent fils, l'action *traiter Réponse Fils* est déclenchée. Le résultat du fils est enregistré et les candidats sont de nouveau définis. Plusieurs cas peuvent se présenter :

(4a) S'il n'existe plus de candidats (condition ¬C3), le contrôleur arrête son exécution et envoie son résultat à son père.

(4b) S'il existe au moins un candidat (condition C3), l'utilisateur est invité à choisir s'il désire continuer sur l'exécution de l'agent.

(4b.1) La réponse de l'utilisateur est traitée par l'action *traiter Choix Continuation*.

Si l'utilisateur choisit d'arrêter l'exécution (condition C4), l'action *traiter Choix Continuation* transmet les résultats obtenus à l'agent père (5). Dans le cas contraire, le processus d'exécution reprend par l'invocation d'un fils s'il existe un seul candidat (condition ¬C4∧C2) (revenir à l'étape 2a.2) ou bien par la demande à l'utilisateur à choisir un candidat s'il en a plusieurs (condition ¬C4∧C1) (revenir à l'étape 2a).

Nous détaillons dans ce qui suit le comportement de chacune des actions identifiées à la Figure 88 d'une manière textuelle.

Action exécuter

L'action *Exécuter* reçoit une demande d'exécution matérialisée par plusieurs situations initiales instanciées dont le type est décrit par le nom *NomSituationTypeInitiale* et est formalisé comme un type complexe XML. Cette action initialise le comportement du contrôleur en déterminant les agents fils candidats qui peuvent être invoqués pour la situation fournie en entrée. Le résultat est un ensemble de candidats qui sont des couples <produit, agent candidat>. L'action Exécuter aboutit à l'invocation d'un agent fils.

Action	Exécuter
Description	Cette action permet de déclencher l'exécution du contrôleur
Données en entrée	Situation initiale : <NomSituationTypeInitiale, liste de <situationInitialeInstanciée>>
Prédicat	----
Traitement	Déterminer la liste des candidats S'il existe un seul candidat alors – Mettre à jour l'agent courant qui reçoit le couple <situation en entrée, l'agent candidat> – Lorsque l'agent courant est mis à jour, exécuter l'agent courant Sinon – Afficher l'interface graphique Choix candidat qui aide l'utilisateur à choisir l'agent candidat à traiter.

Action TraiterChoixCandidat

Cette action est déclenchée suite à la réception d'un couple candidat <produit, agent candidat> choisi par l'utilisateur pour être exécuté. Elle aboutit à l'invocation de l'agent sélectionné.

Action	TraiterChoixCandidat
Description	Cette action permet d'invoquer l'agent fils choisi par l'utilisateur
Données en entrée	Candidat :<situationInstanciée, agent candidat >

Prédicat	----
Traitement	Mettre à jour l'agent courant Lorsque l'agent courant est mis à jour, exécuter l'agent courant.

Action TraiterReponseFils

Cette action est déclenchée lorsque l'agent fils informe le contrôleur de sa fin d'exécution. Le contrôleur met à jour sa trace et détermine s'il reste encore des candidats. S'il n'y en a plus, le contrôleur termine son exécution et renvoie à son tour une notification de fin d'exécution à son père. Dans le cas contraire, il demande à l'utilisateur s'il désire continuer l'exécution.

Action	TraiterRéponseFils
Description	Cette action permet de mettre à jour l'état courant de l'agent afin de progresser dans la satisfaction de son but
Données en entrée	Situation finale :<NomSituationTypeFinale, liste de <situationFinaleInstanciée>>
Prédicat	----
Traitement	– Mettre à jour la trace – Mettre à jour le résultat de l'agent – Calculer de nouveau la liste des candidats compte tenu de la nouvelle situation – Si la liste des candidats est vide retourner le résultat au père – Si la liste candidats est non vide alors afficher Choix Continuation

Action TraiterChoixContinuation

L'utilisateur a la possibilité de décider s'il veut continuer l'exécution du contrôleur courant ou bien l'arrêter. Dans le second cas, le contrôleur termine son exécution et informe son père alors que dans le premier cas, l'exécution se poursuit.

La spécification de cette action est la suivante.

Action	TraiterChoixContinuation
Description	Cette action permet de continuer l'exécution du contrôleur ou de l'arrêter selon le désir de l'utilisateur
Données en entrée	Réponse utilisateur : oui/non
Prédicat	----
Traitement	– Si réponse utilisateur=oui alors – Si la liste des candidats contient plus d'un candidat alors afficher

	Choix candidats – Sinon mettre à jour l'agent courant – Lorsque l'agent courant est mis à jour, exécuter l'agent courant – Si réponse utilisateur=non alors envoyer le résultat à l'agent père

Le comportement d'un contrôleur est générique. Il est identique pour tous les types de contrôleurs. Les spécificités de chaque type de contrôleur sont principalement situées au niveau des opérations de calcul des candidats et de mise à jour des candidats.

Nous détaillons les algorithmes de ces deux opérations plus loin dans ce chapitre.

4.3. Les entrées/sorties

L'agent peut être amené à interagir avec l'utilisateur à travers les **entrées/sorties** (interfaces graphiques ou autres) afin de capturer les informations dont il a besoin pour continuer son exécution. Des exemples de telles informations pourraient être le choix du prochain fils à exécuter. L'agent fait par ailleurs appel à d'autres agents lors de la réalisation de son but.

4.4. La formule

La **formule** stocke les informations relatives à un service que le contrôleur implémente. La formule indique quels sont les fils que l'agent doit contrôler, et sert à définir les prochains fils à exécuter. Le contrôleur utilise la formule pour activer et exécuter les actions adéquates.

La formule renferme des informations relatives au service implémenté, à savoir :

- La liste des agents fils : ces agents correspondent aux services composant le service implémenté.
- La situation initiale type : elle est constituée de fragments de produits qui sont formalisés ici comme un type complexe XML.
- La situation finale type : elle est constituée de fragments de produits obtenus suite à la réalisation du but du service et elle est formalisée comme un type complexe XML.
- Pré-condition : elle définit les conditions que la situation initiale doit satisfaire pour que le service associé à l'agent soit candidat. Elle est évaluée pour décider si une situation initiale instanciée est candidate.
- Post-condition : elle définit les conditions que la situation finale doit satisfaire pour affirmer que le but associé au service est atteint.
- La liste des agents fils itératifs pour les contrôleurs séquentiels.

- Pré-traitement : c'est l'ensemble des traitements que l'agent père doit déclencher avant l'exécution d'un agent fils. L'objectif est d'extraire de l'état de l'agent père, l'état de l'agent fils.
- Post-traitement : c'est l'ensemble des traitements que l'agent père doit déclencher pour intégrer le résultat de l'invocation d'un agent fils dans le résultat de l'agent père.

4.5. L'état

Le contrôleur stocke les résultats intermédiaires ainsi que certaines données utiles dans la **trace d'exécution**. Des exemples d'informations contenues dans la trace d'exécution sont l'agent en cours d'exécution et la liste des agents candidats. L'état de l'agent est modifié au fur et à mesure de l'avancement de l'exécution de la formule de réalisation du but.

L'état est composé des éléments suivants :

- La liste des candidats : ce sont des couples <situationInstantiée, agent candidat> où le produit désigne le fragment de produit nécessaire en entrée pour l'exécution de l'agent candidat.
- L'agent en cours : il représente la situation courante de l'exécution, à savoir l'agent en cours d'exécution et le fragment de produit qui lui est fourni en entrée.
- La trace : elle stocke les agents exécutés et les résultats obtenus.
- Le résultat : il stocke les produits obtenus suite à l'exécution des fils.

Ainsi l'entité état comporte des opérations qui sont essentiellement :

- *CalculerCandidats(situation)* : couple[ ]
- *MAJCandidats(situation)*
- *MAJTrace(situation)*
- *MAJRésultat(situation)*

L'opération *CalculerCandidats* permet de déterminer les agents candidats à invoquer. Elle est spécifique à chaque type de contrôleur. L'opération *MAJCandidats* met à jour la liste des candidats, elle fait aussi appel à l'opération *CalculerCandidats*. Les opérations *MAJTrace* et *MAJRésultat* permettent de mettre à jour respectivement la trace et le résultat de l'agent.

5. SPECIFICATION CONCEPTUELLE D'UN EXECUTANT

La spécification d'un agent exécutant est plus simple que celle du contrôleur puisque, d'une part, il n'interagit pas avec l'utilisateur et, d'autre part, il n'a pas d'agents fils. Par

conséquent, il se réduit à invoquer un service logiciel avec la transmission des paramètres appropriés et la récupération du résultat de cette exécution.

Plus précisément, un exécutant joue le rôle d'intermédiaire entre les agents de la hiérarchie et les services logiciels. Il permet l'invocation d'un service logiciel en lui transmettant les paramètres nécessaires et retourne le résultat de cette dernière à son agent père pour poursuivre l'exécution. L'exécutant réalise aussi des opérations de conversion des types de données des paramètres échangés entre les services logiciels et les agents.

Il joue ainsi le rôle d'un adaptateur (comme le patron de conception adaptateur (« Adaptor ») [Gamma95] afin de convertir des services logiciels existants (le format des entrées/sorties, le nom du service,…) selon un autre format compatible à celui des agents de la hiérarchie.

Un exécutant, comme le montre la Figure 89, est conceptuellement constitué de quatre parties.

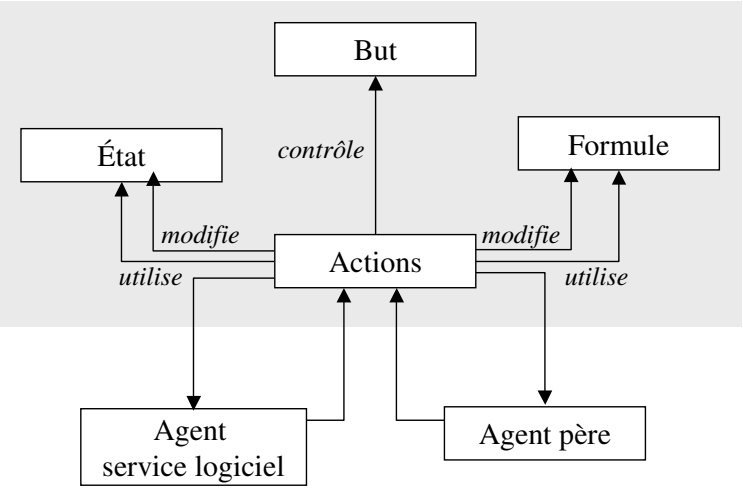


Figure 89. Spécification d'un exécutant

Le **but** définit l'objectif que l'exécutant désire atteindre c'est à dire réaliser le but du service atomique correspondant.

Le but d'un exécutant, tout comme celui d'un contrôleur, respecte des conventions d'appellation. Le but est structuré, selon le même principe <verbe, liste de paramètres> que pour les agents contrôleurs.

Les **actions** permettent la réalisation du but de l'exécution à savoir l'action d'invocation du service logiciel et l'action de traitement du résultat de cette dernière.

L'**état** de l'exécutant renferme principalement le résultat obtenu suite à l'exécution du service logiciel.

La **formule** contient :

- L'agent wrapper permettant d'emballer l'exécution du service logiciel

- La situation initiale type : est le type complexe XML représentant le message d'entrée du service logiciel. L'agent exécutant envoie comme paramètre d'entrée à l'invocation du service logiciel une instance de la situation initiale type à l'agent wrapper du service logiciel.
- La situation finale type : est le type complexe XML représentant le message de sortie du service logiciel. L'agent wrapper a comme rôle de renvoyer une instance de la situation finale type à l'agent exécutant.
- La pré-condition permettant de déterminer si la situation instantiée est une situation intitiale candidate.
- La post-condition permettant de déterminer que le message retourné par le service logiciel permet bien d'atteindre le but associé à l'agent exécutant.
- Le pré-traitement permet de dériver l'état de l'exécutant par rapport à l'état de l'agent père qui l'invoque.
- Le post-traitement permet d'intégrer la situation finale de l'agent exécutant dans le résultat de l'agent père.

La Figure 90 identifie les actions réalisées par l'agent exécutant.

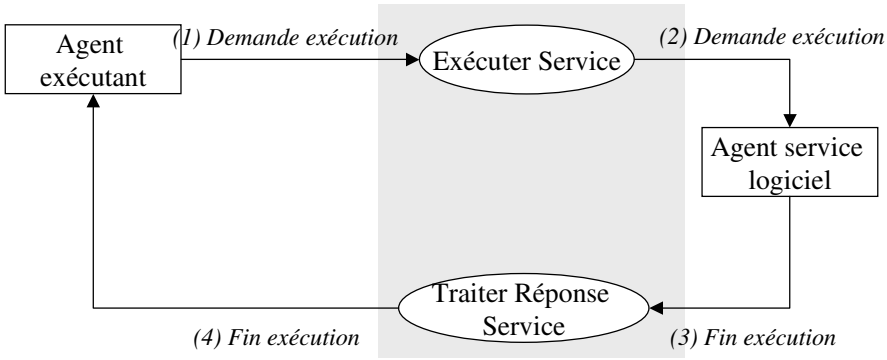


Figure 90. Traitement des stimuli externes par un exécutant

1) À l'arrivée d'une demande d'exécution de la part de l'agent exécutant, l'action *Exécuter Service* est réalisée. Elle consiste à invoquer directement le service logiciel correspondant (2).

3) A la réception de la notification de fin d'exécution du service logiciel, l'action *Traiter Réponse Service* est déclenchée. Le résultat du service logiciel est enregistré.

4) l'action *Traiter Réponse Service* transmet les résultats obtenus à l'agent exécutant.

Nous détaillons dans ce qui suit le comportement de chacune des actions identifiées à la Figure 90 d'une manière textuelle.

Action ExécuterService

L'action *ExécuterService* reçoit une demande d'exécution du service matérialisée par une situation en entrée. Cette action initialise le comportement de l'exécutant en invoquant le service logiciel.

La spécification de cette action est illustrée comme suit :

Action	ExécuterService
Description	Cette action permet de déclencher l'exécution d'un service logiciel
Données en entrée	Situation initiale candidate
Prédicat	----
Traitement	– Transformer la situation fournie en paramètre pour qu'elle soit compatible avce celle du service logiciel. – Invoquer l'agent spécialisé dans l'exécution du service logiciel.

Action TraiterRéponseService

Cette action est déclenchée lorsque le service logiciel informe l'exécutant de sa fin d'exécution. L'exécutant met à jour l'état, termine son exécution et renvoie à son tour une notification de fin d'exécution à son père.

La spécification de cette action est illustrée comme suit :

Action	TraiterRéponseService
Description	Cette action permet de mettre à jour l'état courant de l'exécutant
Données en entrée	Situation finale par le service logiciel.
Prédicat	----
Traitement	– Mettre à jour la trace – Mettre à jour le résultat de l'agent – Calculer la liste des cadidats – Retourner au père.

6. PROCESSUS DE CONSTRUCTION DE L'ARCHITECTURE

Le processus de construction de l'architecture comporte les étapes suivantes :

- Etape 1 : Identification et spécification des agents de la hiérarchie
- Etape 2 : Identification et spécification des services

La première étape prend en entrée le modèle $\mathcal{MIS}$ (tel qu'il a été présenté au Chapitre 3 (Modèle intentionnel de services)) et définit les agents implémentant les différentes sortes de

services qu'il comporte. L'étape 2 consiste à déterminer les services logiciels et à les associer aux exécutants de la hiérarchie. L'architecture obtenue est affinée lors des étapes de conception avancées jusqu'à obtenir une application.

Dans ce qui suit, nous détaillons les deux étapes.

6.1. Etape 1 : Identification et spécification des agents de la hiérarchie

La construction de l'architecture implémentant les services d'un modèle $\mathcal{MiS}$ se fait par une transformation 1 :1 entre les services et les agents correspondants.

La construction de la hiérarchie passe par deux sous étapes :

- Etape 1.1 : Identification des agents et,
- Etape 1.2 : Spécification des agents.

6.1.1. Etape 1.1 : Identification des agents

Les règles suivantes sont appliquées pour identifier les agents :

R1 : un service atomique est associé à un exécutant.
R2 : un service de choix multiple est géré par un contrôleur de choix multiple.
R3 : un service de choix alternatif est implémenté par un contrôleur de choix alternatif.
R4 : un service à variation de chemin est implémenté par un contrôleur de chemin.
R5 : un service composite séquentiel est implémenté par un contrôleur de séquence.
R6 : un service composite parallèle est implémenté par un contrôleur parallèle.

L'exemple présenté à la Figure 91 ainsi que le Tableau 27 illustrent l'application des règles énoncées ci-dessus. Le résultat est une architecture décrivant les agents qui gèrent les servies du modèle $\mathcal{MiS}$.

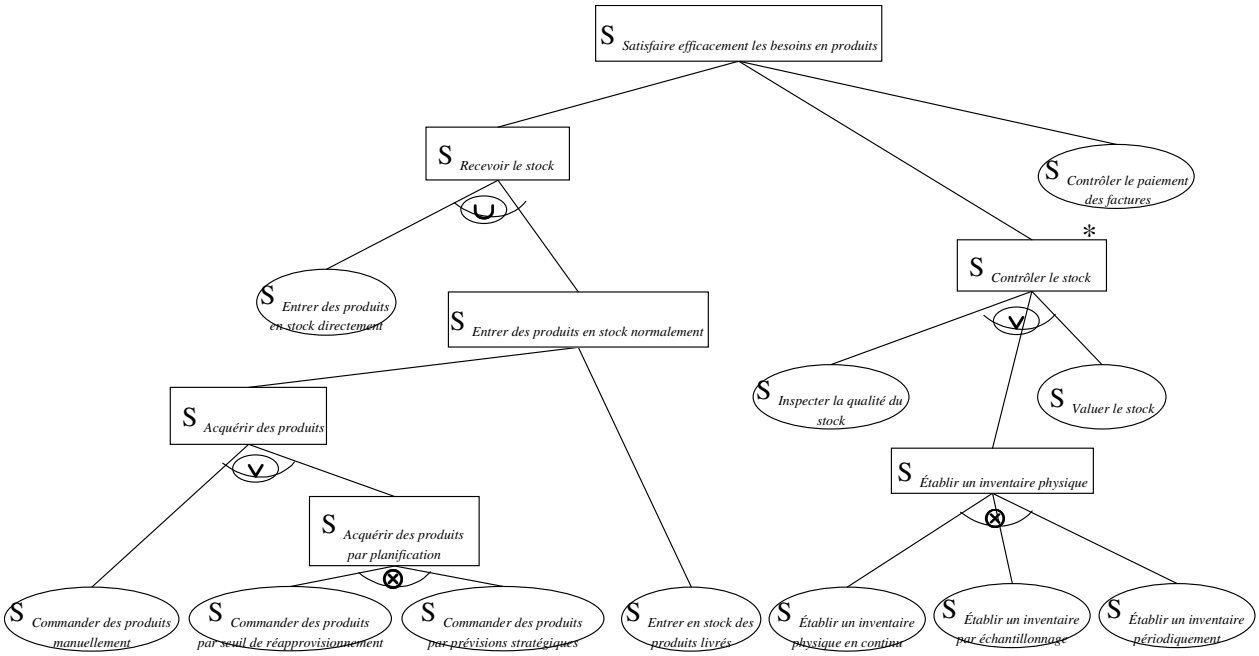


Figure 91. Exemple d'une hiérarchie d'agents

Tableau 27. Identification des agents à partir du modèle $\mathcal{M}_{IS}$ de l'exemple *Satisfaire efficacement les besoins en produits*

Type du service	Service	Type agent	Règle appliquée
Service atomique	ab_1,ab_2, ab_3, ac_1, cc_1, cc_2, cc_3, cc_4, cc_5, cc_6, cd_1	Exécutant	R1
Service à choix alternatif	$S_{Acquérir\ des\ produits\ par\ planification} = \otimes (S_{Commander\ des\ produits\ par\ seuil\ de\ réapprovisionnement}, S_{Commander\ des\ produits\ par\ prévisions\ stratégiques})$ $S_{Etablir\ un\ inventaire\ physique} = \otimes (S_{Etablir\ un\ inventaire\ physique\ en\ continu}, S_{Etablir\ un\ inventaire\ physique\ par\ échantillonnage}, S_{Etablir\ un\ inventaire\ physique\ périodiquement})$	Contrôleur de choix alternatif	R3
Service à choix multiple	$S_{Acquérir\ des\ produits} = \vee (S_{Commander\ des\ produits\ manuellement}, S_{Acquérir\ des\ produits\ par\ planification})$ $S_{Contrôler\ le\ stock} = \vee (S_{Etablir\ un\ inventaire\ physique}, S_{Inspecter\ la\ qualité\ du\ stock}, S_{Transférer\ des\ produits}, S_{Valuer\ le\ stock})$	Contrôleur de choix multiple	R2
Service à variation de chemin	$S_{Recevoir\ le\ stock} = \cup (S_{Entrer\ les\ produits\ en\ stock\ directement}, S_{Entrer\ les\ produits\ en\ stock\ normalement})$	Contrôleur de chemin	R4
Service composite séquentiel	$S_{Satisfaire\ efficacement\ les\ besoins\ en\ produits} = \bullet(S_{Recevoir\ le\ stock}, S_{Contrôler\ le\ stock}^{*}, S_{Contrôler\ le\ paiement\ des\ factures})$	Contrôleur séquentiel	R5

Pour la construction de la hiérarchie d'agents, on procède comme suit :

(1) Identifier l'agent racine de la hiérarchie. Cet agent contrôle le service représentant tout le modèle $\mathcal{MiS}$. Dans notre exemple au Tableau 27, le service relatif à tout le modèle $\mathcal{MiS}$ est $S_{Satisfaire\ efficacement\ les\ besoins\ en\ produits}$. Il est géré par un contrôleur séquentiel par application de la règle R5.

(2) Identifier les agents contrôlant les services du modèle $\mathcal{MiS}$ par analyse de leurs formules respectives en commençant par le service représentant tout le modèle $\mathcal{MiS}$. Les agents identifiés sont, par la suite, reliés à leur agent père (l'agent correspondant au service se trouvant à la partie gauche de la formule) par des liens de délégation. Si une formule donnée contient un service atomique alors associer un agent exécutant par application de la règle R1 et placer l'agent dans les feuilles de la hiérarchie. Si une formule contient des services composites ou à variation alors identifier les contrôleurs correspondant par application des règles R2, R3, R4, R5 ou R6 et poursuivre l'analyse de leurs formules respectives.

Par exemple, le service $S_{Satisfaire\ efficacement\ les\ besoins\ en\ produits}$ est :

$S_{Satisfaire\ efficacement\ les\ besoins\ en\ produits} = \bullet(S_{Recevoir\ le\ stock}, S_{Contrôler\ le\ stock}^{*}, S_{Contrôler\ le\ paiement\ des\ factures})$

Nous avons identifié dans (1) que le service $S_{Satisfaire\ efficacement\ les\ besoins\ en\ produits}$ est géré par un contrôleur de séquence qui est en même temps la racine de la hiérarchie. Chaque service composant le service racine identifié à son tour un type de contrôleur ou exécutant. Le service à variation de chemin $S_{Recevoir\ le\ stock}$ identifie un contrôleur de chemin selon la règle R4. Le service à choix multiple $S_{Contrôler\ le\ stock}$ par application de la règle R2. Enfin, le service atomique $S_{Contrôler\ le\ paiement\ des\ factures}$ identifié un exécutant selon la règle R1. Les contrôleurs et exécutants identifiés représentent les fils de l'agent racine comme le montre la Figure 91. Les services $S_{Recevoir\ le\ stock}$ et $S_{Contrôler\ le\ stock}$ sont à leur tour analysés pour identifier les contrôleurs et exécutants respectifs.

6.1.2. Etape 1.2 : Spécification des agents

Les agents identifiés sont spécifiés par configuration de certains paramètres des types d'agents qu'ils instancient.

Chaque agent est configuré grâce à sa facette Formule. Pour chaque agent, l'interprétation du Modèle $\mathcal{MiS}$ permet de renseigner les différents éléments de la facette Formule :

- Le but de l'agent
- La liste des agents fils
- La liste des fils itératifs pour les contrôleurs séquentiels
- La situation initiale type spécifiée sous forme d'un type complexe XML
- La situation finale type spécifiée sous la forme d'un type complexe XML

- La pré-condition décrit l'ensemble des conditions que la situation initiale doit vérifier pour que la situation soit considérée comme candidate.
- La post-condition décrit l'ensemble des conditions que la situation finale doit vérifier pour constater la satisfaction du but.
- Le pré-traitement et le post-traitement sont décrits pour expliquer la manière de construire l'état de l'agent par rapoort à l'agent père et la manière d'intégrer la situation finale de l'agent dans celle de l'agent père.

6.2. Etape 2 : Identification et spécification des services logiciels

Dans le cadre de l'orchestration intentionnelle de services, chaque service intentionnel atomique est opérationnalisé par un agent exécutant.

Le modèle $\mathcal{M}oS$ développe l'opérationnalisation d'un service intentionnel atomique à l'aide d'un service logiciel. L'implémentation du service logiciel consiste à développer chacune de ses trois éléments logiciels à savoir :

- Le service d'interface utilisateur est implémenté à l'aide d'une technique de développement web,
- Le service de coordination est implémenté par la génération de la spécification selon le langage BPEL et,
- Le service métier est implémenté en deux étapes : le WSDL est généré à partir de la spécification $\mathcal{M}oS$ et ensuite une implémentation doit être développée selon une technique de développement de composants métier.

L'orchestration intentionnelle dirigée par le modèle $\mathcal{M}iS$ délègue au modèle $\mathcal{M}oS$ l'opérationnalisation des services. Comme nous l'avons, dans le Chapitre 4, l'interconnexion des deux modèles se fait de manière bidirectionnelle.

L'outil d'orchestration intentionnelle prenant en charge le modèle $\mathcal{M}iS$ propose de normaliser la délégation de $\mathcal{M}iS$ vers un modèle opérationnel de service à l'aide d'un agent wrapper du service logiciel.

L'agent wrapper définie une enveloppe générique pour opérationnaliser la relation bidirectionnelle qui doit exister entre $\mathcal{M}iS$ et le modèle opérationnel. Cette communication peut être de type asynchrone dans le cas où le service logiciel exécuterait une transaction métier longue. Par conséquent, l'agent wrapper du service logiciel est décrit par deux messages :

- Le message en entrée : est la structure de message que l'agent exécutant envoie à l'agent wrapper c'est à dire sa situation initiale.
- Le message en sortie : est la structure de message que le service logiciel produit en sortie et qui est envoyé à l'agent exécutant c'est à dire sa situation finale.

Deux méthodes sont définies au niveau de l'agent wrapper du service logiciel :

- TransmettreResultat (SituationFinale) et,
- DéclencherService (SituationInitiale).

Le comportement de l'agent de type wrapper est le suivant :

- A la réception d'un message en entrée de l'agent exécutant, la méthode *DéclencherService()* est exécutée,
- A la réception d'un message de sortie du service logiciel, la méthode *TransmettreResultat()* est exécutée pour la transmettre à l'agent exécutant.

7. MECANISME D'ORCHESTRATION GUIDEE PAR LES BUTS DES SERVICES DURANT L'EXECUTION

7.1. Description du principe d'orchestration choisi

Durant l'exécution, l'orchestration des services intentionnels se fait comme suit :

1- Les agents constituant l'application proposent au client des services à exécuter compte tenu de la situation courante dans laquelle il se trouve.

2- Le client sélectionne un service.

3- Le choix d'un service peut, selon les cas, amener l'application à demander au client d'affiner encore plus son choix en lui proposant des sous services ou bien invoquer directement le service logiciel implémentant le service choisi.

4- L'exécution d'un service logiciel produit une nouvelle situation que l'application prend en compte pour identifier les services à suggérer au client (retour à l'étape 1).

L'orchestration intentionnelle possède les caractéristiques suivantes :

- Une orchestration à l'exécution par adaptation de l'application : l'application propose au client des services et c'est à ce dernier de choisir ceux qui lui conviennent.
- Une orchestration guidée par les besoins des clients et adaptée à leur langage : le client est amené à faire des choix sur les buts qu'il désire satisfaire. Les propositions de services faites par l'application sont exprimées en termes de buts. Par ailleurs, l'application guide le client dans la satisfaction de ses besoins en lui proposant, à

chaque étape, les services qu'il peut exécuter compte tenu de la situation courante d'exécution.

L'orchestration est mise en œuvre par une hiérarchie constituée de deux sortes d'agents : les contrôleurs qui gèrent le choix des services et les exécutants qui assurent l'invocation des services logiciels. La section 4 et 5 de ce chapitre, présentent la structure des agents ainsi que leur comportement général. Dans cette section, nous mettons l'accent sur la manière dont chacun des contrôleurs définit les services adéquats compte tenu de la situation courante d'exécution.

Nous détaillons dans ce qui suit le comportement de chacun des agents.

7.2. Description du comportement des agents contrôleurs

Comme nous l'avons mentionné à la section 5 de ce chapitre, tous les contrôleurs possèdent un comportement générique.

Les spécificités de chacun des contrôleurs se situent au niveau de la gestion de la satisfaction du but qui leur est rattaché. En effet, nous rappelons que le rôle d'un contrôleur est de guider l'utilisateur à satisfaire un but final. A un moment donné, l'utilisateur se trouve dans une situation donnée caractérisée par un ensemble de produits en cours obtenus suite à la réalisation de buts. Il choisit le prochain but à satisfaire. Le contrôleur responsable du but choisi se charge de réaliser ce dernier en proposant à l'utilisateur :

- soit des manières possibles d'atteindre directement le but,
- soit les buts intermédiaires à satisfaire au préalable pour atteindre finalement le but final (comme c'est le cas des contrôleurs de chemin, de séquence et parallèle).

La proposition des manières de satisfaction du but du contrôleur est déterminée par l'opération *calculerCandidats()* qui est différente pour chaque type de contrôleur.

Pour le cas d'un contrôleur de choix alternatif, cette opération retourne les fils du contrôleur (des exécutants) représentant les stratégies possibles pour atteindre le but du contrôleur.

Pour le cas d'un contrôleur séquentiel, parallèle et de choix multiple, cette opération donne comme résultat le ou les prochains agents à invoquer correspondant aux prochains buts intermédiaires à satisfaire.

Pour le cas d'un contrôleur de chemin, cette opération détermine les différents chemins possibles pour satisfaire le but final.

Par ailleurs, l'opération *MAJCandidats()* peut présenter des différences d'un type de contrôleur à un autre.

Nous expliquons dans les prochaines sous sections les différences de comportement de chacun des contrôleurs.

7.3. Comportement d'un contrôleur de choix alternatif

Un contrôleur de choix alternatif propose à l'utilisateur de choisir un fils exécutant à invoquer parmi tous ses fils. Lorsque l'exécutant termine son exécution, il renvoie son résultat à l'agent contrôleur de choix alternatif qui termine son exécution ou renvoie la main à un autre agent de la hiérarchie (son agent père).

L'algorithme de calcul des candidats est décrit comme suit :

`calculerCandidats(s :Situation) :Couple [ ]`

- Soit s la situation fournie en entrée à l'agent contrôleur de choix alternatif, s est de la forme <nomSituationInitialeType, liste<situationInstanciée>>>
- Soit ListeFils l'ensemble des agents fils (exécutants) contrôlés par le contrôleur de choix alternatif
- Soit a un agent appartenant à listeFilsCandidats
- Soit p une instance de situation appartenant à s.listeProduits
- Soit Type la description XML de la situation type initiale (s.type)
- Soit c un couple candidat de la forme <p, a>
- Soit C la liste des couples candidats (vide au départ)

L'ensemble des couples candidats C est obtenu comme suit :

C<-construireCouples(s, ListeFils)

L'algorithme de la méthode est le suivant :

`construireCouples(Situation s, listeFilsCandidats) :Couple [ ]` est le suivant

```
Si listeProduits est non vide
   Pour chaque a ∈ listeFilsCandidats
       Pour chaque p ∈ s.listeProduits
         Si evaluerCondition (a.preCondition, s.type, p) Alors
           nouvelleSituation<-créerSituation(s.type, p)
           c<-créerCouple(nouvelleSituation,a)
           ajouterCouple(C,c)
        fin si
       fin pour
   fin pour
Sinon
   Pour chaque a ∈ listeFilsCandidats
         c<-créerCouple(s,a)
         ajouterCouple(C,c)
   fin pour
fin si
retourne C
```

Explication de quelques méthodes utilisées dans les deux algorithmes

- `créerSituation(nomSituationTypeInitiale, situationInstanciée) :Situation` prend en entrée le nom du type complexe XML représentant la description de la situation type et une instance de cette situation type. La fonction retourne une situationCandidate. L'objectif de la fonction est de vérifier

que l'instance de situation fournie sous forme XML est conforme à son type. Le résultat est nommé situation candidate et est représenté par un couple de la forme <nomSituationTypeInitiale, situationInstanciée >.

- `créerCouple (situationCandidate, agent) : Couple` prend en entrée une situation candidate ainsi qu'un agent et retourne un couple de la forme <situation candidate, agent candidat>.
- `ajouterCouple(listeCouplesCandidats, couple)` ajoute un couple <situation candidate, agent candidat> à la liste listeCouplesCandidats.
- `evaluerCondition(préCondition,NomSituationTypeInitiale,SituationInstanciée)` evalue la pré-condition sur l'instance de situation selon la description XML de la situationType et retourne une valeur booélenne. Elle retourne vraie lorsque les pré-conditions sont vérifiées et faux sinon.

7.4. Comportement d'un contrôleur de choix multiple

Un agent contrôleur de choix multiple gère des agents exécutants complémentaires. Plusieurs agents exécutants peuvent être candidats pour une même situation de départ. Initialement, tous les agents exécutants sont candidats. A la suite de l'exécution d'un agent exécutant particulier, l'agent contrôleur met à jour sa trace d'exécution et recalcule les agents exécutants candidats qui peuvent être de nouveau invoqués par l'opération *MAJCandidats()*. Cette opération appelle l'opération *calculerCandidats()* pour déterminer les candidats selon la situation courante. Les nouveaux agents candidats sont l'ensemble des agents candidats sans ceux déjà exécutés. L'agent contrôleur arrête son exécution s'il n'y plus d'agents candidats ou lorsque l'utilisateur a choisi de réaliser un autre but.

L'algorithme ci-dessous décrit la méthode de calcul des candidats d'un contrôleur de choix multiple :

```
calculerCandidats(s :Situation) :Couple [ ]
```

```
    ▭ Soit AE l'ensemble des agents déjà choisis pour une situation s
    ▭ Soit ListeFilsCandidats l'ensemble des fils non encore
      sélectionnés pour une situation s
    ▭ Soit a un agent
L'ensemble des couples résultats C est obtenu comme suit :

//chercher les agents déjà choisis à partir de la trace
AE<-chercherAgentExecute(s)

Si (AE vide) alors //cas première exécution pour la situation s
 //tous les agents sont candidats pour une situation s en entrée
     C<-construireCouples(s, ListeFils)
Sinon
//AE non vide
//définir les nouveaux couples candidats pour la situation en entrée
//en supprimant les couples pour lesquels un agent a déjà été choisi
Pour chaque a ∈ ListeFils
     Si a ∉ AE alors
       ajouterFilsCandidats(ListeFilsCandidats, a)
     Fin si
Fin pour
C<-construireCouples(s, ListeFilsCandidats)
Fin si
retourne C
```

Explication de quelques méthodes utilisées dans l'algorithme :

- `ajouterFilsCandidats (listeAgents, agent)` permet d'ajouter un agent à la liste d'agents listeAgents
- `chercherAgentExecute(situation) : listeAgentsExecutes` identifie à partir de la trace la liste des agents qui ont déjà été choisis et exécutés par l'utilisateur compte tenu de la situation fournie en entrée.

7.5. Comportement d'un contrôleur de chemin

Un contrôleur de chemin possède un comportement similaire à celui d'un contrôleur de choix multiple à la différence qu'il gère des agents de différentes sortes (exécutants ou contrôleurs) alors qu'un contrôleur de choix multiple gère uniquement des exécutants. Le contrôleur de chemin propose à l'utilisateur plusieurs chemins pour satisfaire ses besoins. Lorsque ce dernier termine l'exécution du chemin, le contrôleur propose de nouveau des chemins candidats en écartant ceux qui ont été déjà explorés par l'utilisateur.

7.6. Comportement d'un contrôleur de séquence

Un contrôleur de séquence guide l'utilisateur dans la satisfaction de son but, en lui proposant au fur et à mesure les étapes suivantes à réaliser (c'est à dire les buts suivants à réaliser) jusqu'à atteindre le but final souhaité. Chaque étape est gérée par un contrôleur ou un

exécutant. La détermination de l'agent candidat à chaque étape dépend de sa nature c'est à dire s'il est itératif ou non.

S'il s'agit d'une première exécution, l'agent candidat est le premier fils du contrôleur de séquence. S'il s'agit d'une étape intermédiaire, l'agent candidat est le successeur de l'agent courant. Dans le cas où l'agent successeur est itératif, il est proposé de nouveau comme candidat.

L'algorithme de l'opération de calcul des candidats responsables de la détermination des agents candidats possède le comportement suivant :

`CalculerCandidats(Situation s) : Couple [ ]`

```
   ▭ Soit ListeFilsCandidats la liste des agents candidats pour une
     situation s donnée en entrée
   ▭ Soit SituationEnCours le couple <situation, agent> exécuté durant
     l'étape précédente
L'ensemble C des couples Candidats à l'étape suivante est obtenu comme
suit :

Si (première exécution)Alors
      //proposer le premier fils
      premierFils<-identifierPremierFils(ListeFils)
      AjouterFilsCandidats(ListeFilsCandidats, premierFils)
Sinon
//cas agent itératif
Si (s.description==agentEnCours.situation.description) alors
//proposer la situation en entrée avec comme agent candidat l'agent
//courant
AjouterFilsCandidats(ListeFilsCandidats,agentEnCours.agent)
Fin si
//proposer l'agent successeur à l'agent courant
prochainFils<-identifierProchainFils(ListeFils,agentEnCours.agent)

AjouterFilsCandidats(ListeFilsCandidats, prochainFils)
Fin si
C<-construireCouple(s, ListeFilsCandidats)
retourne C
```

Explication de quelques méthodes utilisées dans l'algorithme

- `identifierPremierFils(listeFils) : Agent` retourne le premier agent de la liste listeFils.
- `identifierProchainFils(listeFils, agent) : Agent` retourne l'agent successeur à l'agent fourni en entrée dans la liste listeFils.

Le contrôleur de séquence arrête son exécution lorsqu'il a atteint son but ou bien lorsque l'utilisateur arrête l'exécution. L'opération *MAJCandidat()* teste, après chaque fin d'exécution d'un agent fils, si le but du contrôleur est atteint. Si le but n'est pas encore atteint, cette opération invoque de nouveau l'opération *calculerCandidats()* en lui passant en paramètre la nouvelle situation obtenue en résultat à partir du fils exécuté.

L'algorithme de cette opération est comme suit :

```
MAJCandidats(nouvelleSituation :Situation)
```

```
   ⌨ Soient C1, C des listes de couples candidats
   ⌨ Soit ListeCandidats l'ensemble des couples candidats du contrôleur


//enlever les couples ayant comme agent, l'agent déjà exécute

ListeCandidats.supprimerCouples(agentEnCours.agent)

//si le but du contrôleur n'est pas atteint redéfinir les candidats

si (non estAtteint()) alors

  //si l'agent courant exécuté est itératif, le proposer comme candidat
   si (agentEnCours.agent.estIteratif()) alors
    C1<- calculerCandidats(nouvelleSituation)
    //ajouter les couples candidats à ListeCandidats
     ListeCandidats.ajouterCouples(C1)
   Fin si

//définir les couples candidats pour la nouvelle situation
C<-calculerCandidats(nouvelleSituation)
//ajouter les couples candidats à ListeCandidats
     ListeCandidats.ajouterCouples(C1)

Fin si
```

7.7. Comportement d'un contrôleur parallèle

Le principe d'un contrôleur parallèle est qu'il n'y a pas d'ordre pré-établi d'exécution des agents fils. Initialement et compte tenu d'une situation donnée, tous les agents fils sont candidats. A la suite de l'exécution d'un agent, la nouvelle situation est utilisée comme situation d'entrée pour déterminer les agents candidats qui sont tous les agents fils moins les agents déjà exécutés. Nous proposons l'algorithme suivant de la méthode *CalculerCandidats()* :

```
CalculerCandidats(Situation s) : Couple [ ]
```

```
   ▭ Soit ListeFilsCandidats la liste des agents candidats pour une
     situation s donnée en entrée
   ▭ Soit AE l'ensemble des agents déjà choisis

L'ensemble C des couples candidats est obtenu comme suit :

Si (première exécution) Alors
        //proposer tous les fils
       C<-construireCouple(s, ListeFils)
Sinon
//identifier les fils candidats = liste des fils moins les fils déjà
//exécutés
   Pour chaque a ∈ ListeFils
        Si (a ∉ AE  ou estIteratif(a)) alors
          ajouterFilsCandidats(ListeFilsCandidats, a)
        Fin si
   Fin pour
   C<-construireCouples(s, ListeFilsCandidats)

Fin si
```

8. CONCLUSION

Nous avons proposé dans ce chapitre, une solution qui offre un guidage au développeur pour implémenter les services du modèle $\mathcal{MIS}$.

La solution propose de réaliser les services intentionnels sous forme de composants architecturaux appelés agents. Chaque agent est responsable de la gestion d'un service particulier. Le passage des services du modèle $\mathcal{MIS}$ vers les agents est défini par un processus méthodologique spécifique. La solution proposée dans ce chapitre consiste à définir les principaux composants de l'application, comment ils doivent être structurés et comment ils doivent interagir. Le développeur affine par la suite la solution en intégrant la spécification du service intentionnel atomique pour obtenir une description complète.

Cette démarche de développement permet d'architecturer un système selon les termes d'un modèle intentionnel centré sur les buts de ses utilisateurs.

L'architecture proposée permet l'orchestration intentionnelle basée sur la navigation intentionnelle à travers les services du modèle $\mathcal{MIS}$. L'orchestration décrit les différentes combinaisons possibles de services et en quoi ils peuvent participer à la réalisation des besoins des utilisateurs. En raisonnant en quoi un service peut satisfaire son besoin, l'utilisateur peut décider s'il le choisit ou non.

CHAPITRE 7

Cas d'application

1. INTRODUCTION

Ce chapitre présente d'une manière détaillée l'application de la démarche sur un cas d'étude permettant d'aider les personnes handicapées à obtenir une pension. Les dirigeants des organisations de traitement des pensions 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 (entre administrations) et B2C (entre administrations et citoyens).

Ce chapitre est organisé comme suit : la section 2 présente le cas d'étude e-Pension. Ensuite, la section 3 présente une vue d'ensemble de la mise en œuvre du processus méthodologique pour générer les services intentionnels. La section 4 est consacrée à l'application du processus méthodologique sur le cas e-Pension. La section 5 se focalise sur la construction de l'application e-Pension à base de services logiciels. La section 6 illustre le processus d'orchestration intentionnelle à l'exécution du cas e-Pension. Enfin les apports et conclusions sont présentés à la section 7.

2. INTRODUCTION DU CAS D'ETUDE

Cette section présente l'étude de cas dans les grandes lignes et introduit les axes d'évolution que les utilisateurs souhaitent prendre.

2.1. Cas d'étude

Le cas d'étude utilisé dans ce chapitre, est extrait de [Batini01] et présente une version simplifiée du processus qui a comme but « *Aider les personnes handicapées à obtenir une pension* ».

Dans l'état actuel, un citoyen présentant un handicap peut demander une pension du gouvernement. 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 et 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

La description précédente montre que le processus est long et compliqué surtout pour des personnes handicapées. On observe qu'un temps considérable est gaspillé à transmettre des documents d'une administration à une autre. 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. La diagnostique de ces dysfonctionnements montre qu'il serait utile d'automatiser le processus pour aboutir à un état de fonctionnement satisfaisant pour les administrations, 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 souhaitent implanter une solution mettant en œuvre une application qui permet la coopération du LHA, de la Préfecture et de la Mairie afin de réaliser le but « *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 : 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 : mettre en place une application coopérative qui ouvre la voie des interactions B2B et B2C afin de réaliser le but 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 des 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 les autres. Les services web permettent justement de se prémunir de ce type de problèmes tant que les interfaces restent stables. La capacité des services à se décrire eux-mêmes permet d'envisager l'automatisation de l'intégration de services.

Enfin, 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 va passer automatiquement une demande de pension auprès de ses partenaires en vue d'une étude. Pour cela, il va employer les services web proposés par ses 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 en notifier le service de traitement des pensions de 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 METHODOLOGIQUE

Les dirigeants ont décidé de mettre en œuvre la démarche méthodologique proposée dans cette thèse afin de réaliser une application à base de services en partant des besoins et attentes des citoyens.

La démarche appliquée à l'exemple met en œuvre trois étapes à savoir :

- Construire le modèle $\mathcal{MiS}$
- Construire le modèle $\mathcal{MoS}$
- Orchestrer les services du modèle $\mathcal{MiS}$

Le modèle $\mathcal{MiS}$ représente les services qui exhibent une intentionnalité formulée par le *but* qu'ils permettent à ses clients d'atteindre dans une perspective intentionnelle.

Pour construire le modèle $\mathcal{MiS}$, on se base sur le modèle de la carte pour la capture des besoins des utilisateurs d'e-Pension ainsi que d'un ensemble de règles.

Le modèle $\mathcal{MoS}$ décrit à un niveau opérationnel, les services logiciels qui sont un moyen d'opérationnalisation du service intentionnel atomique du modèle $\mathcal{MiS}$. Dans cette optique, un service logiciel constitue une entité opérationnelle ayant une responsabilité spécifique et décrite indépendamment d'une plate-forme spécifique.

L'orchestration des services du modèle $\mathcal{MiS}$ se concrétise par l'identification de services réalisant le but et parmi lesquelles l'utilisateur choisit le service adéquat. Une fois un but satisfait, l'utilisateur peut progresser vers la réalisation d'autres buts. Au niveau de l'architecture, l'identification et l'invocation des services se fait suite à une série d'actions de délégation entre les différents agents concernés et peut nécessiter des interactions avec l'utilisateur.

La section suivante montre de manière détaillée l'application du processus pour construire l'application e-Pension à base de services.

4. CONSTRUIRE LE MODELE $\mathcal{MIS}$

La construction du modèle $\mathcal{MiS}$ met en œuvre un ensemble d'étapes méthodologiques. Ces étapes ont été prises en compte dans l'élaboration de l'application e-Pension.

La première étape consiste à construire le modèle de la carte pour la capture des besoins des utilisateurs de e-Pension. Le modèle de la carte exprime les besoins des utilisateurs par des buts et des stratégies pour les atteindre. La deuxième étape a pour objectif d'identifier les services intentionnels à partir du modèle de la carte e-Pension.

Le déroulement de chacune de ces étapes est détaillé dans les sous sections ci-après de la manière suivante : la sous section 4.1 illustre la construction de la carte e-Pension et la sous section 4.2 illustre la génération du modèle de représentation de services $\mathcal{MIS}$ correspondant.

4.1. Etape 1 : Construire le modèle de capture des besoins : la carte *e-Pension*

Cette section a pour but de décrire le processus pour aider les personnes handicapées à obtenir une pension. Ce processus est décrit en termes intentionnels sous forme de 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 objectifs principaux assignés au processus sont donc : *Formuler la demande* et *Statuer sur la demande*. La carte de la Figure 92 décrit la manière de procéder pour réaliser les objectifs identifiés. Comme le montre la carte *e-Pension*, un ensemble de buts sont identifiés qui doivent ou peuvent être accomplis afin d'aider les personnes handicapées à obtenir une pension ainsiq qu'un ensemble de stratégies qui définissent la manière d'accomplir ces buts.

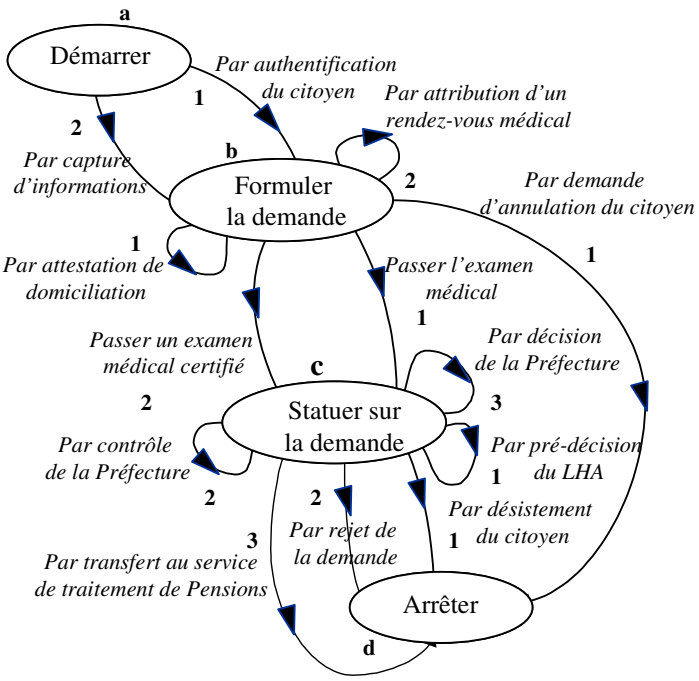


Figure 92. La Carte *e-Pension*

Les quatre buts et treize stratégies de la carte e-Pension sont détaillés ci-dessous.

Les buts de la carte e-Pension

Les buts de la carte e-Pension sont *Démarrer*, *Formuler la demande*, *Statuer sur la demande* et *Arrêter* :

- *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.

Les stratégies de la carte e-Pension

Quatre stratégies sont mises en œuvre pour *Formuler une demande* :

- *Par authentification du citoyen* : cette stratégie a été introduite par e-Pension dans la perspective de sécuriser l'accès aux données personnelles des citoyens.

- *Par capture d'informations :* les informations personnelles constituant la requête du citoyen sont capturées en utilisant cette stratégie.
- *Par attestation de domiciliation :* afin de s'assurer que l'ensemble des citoyens demandeurs de pension sont bien des habitants de la ville, la stratégie *Par attestation de domiciliation* est mise en place avec l'idée de faciliter l'obtention de l'attestation par le citoyen.
- *Par attribution d'un rendez-vous médical :* l'examen assuré par le LHA reste nécessaire et la stratégie *Par attribution d'un rendez-vous médical* a été introduite pour permettre au citoyen d'obtenir un rendez-vous aisément.

La carte *e-Pension* montre aussi qu'il existe cinq stratégies permettant de satisfaire le second but *Statuer sur la demande* à partir du but *Formuler la demande* :

- *Passer un examen médical* : le citoyen peut choisir de passer l'examen médical au LHA.
- *Passer un examen médical certifié* : le citoyen peut choisir de passer l'examen médical chez un médecin assermenté par l'organisation *e-Pension.*
- *Par contrôle de la Préfecture* : le contrôle du processus ainsi que la prise de la décision finale relative à une demande spécifique sont des tâches déléguées à la Préfecture ; le contrôle du processus comprend l'envoi des messages de relance si nécessaire dans le cas où le dossier d'un citoyen serait incomplet. Ces tâches sont réalisées via la stratégie *Par contrôle de la préfecture* qui garantit que le monitoring du processus n'est plus assuré par le citoyen mais bien par l'organisation virtuelle.
- *Par pré-décision du LHA* : cette stratégie permet au LHA de prendre la pré-décision relative à une demande de pension.
- *Par décision de la Préfecture* : la décision finale incombe à la préfecture par application de la cette stratégie.

Enfin, d'après la carte e-Pension il existe un multi-segment entre le but source *Statuer sur la demande* et le but cible *Arrêter* qui repose sur trois stratégies :

- *Par transfert au service de traitement des pensions :* l'application de cette stratégie arrête le processus dans le cas où une décision positive serait prise par la préfecture et dans ce cas le dossier du citoyen est transféré au service payeur.
- *Par rejet de la demande* : cette stratégie est appliquée lorsque la décision est négative.
- *Par désistement du citoyen* : à chaque instant, le citoyen peut arrêter le processus en retirant sa demande de pension en appelant cette stratégie.

Le citoyen peut aussi arrêter le processus en annulant sa demande de pension une fois formulée en appelant la stratégie *Par demande d'annulation du citoyen*.

La carte de *e-Pension* montre diverses stratégies proposées pour atteindre les buts. Ce qui permet d'identifier plusieurs services. La section suivante présente les services du modèle $\mathcal{MiS}$ construits à partir de cette carte

4.2. Etape 2 : Générer le modèle $\mathcal{MiS}$ associé à la carte *e-Pension*

La génération des services du modèle $\mathcal{MiS}$ à partir de la carte e-Pension se fait en appliquant le processus, présenté au chapitre 5. Celui-ci est structuré en deux sous étapes :

- Etape 2.1 : Définir les services atomiques.
- Etape 2.2 : Définir les services agrégats.

Le déroulement de chacune de ces sous étapes est détaillé dans les sous sections ci-après de la manière suivante : la sous section 0 illustre la manière d'identification des services atomiques et la sous section 4.2.2 illustre la manière d'identification des services agrégats.

La carte e-Pension ne présente pas de sections non opérationnalisables. Dans ce cas, l'étape de définition des services relatifs à une section non opérationnalisable de la carte n'est pas utilisée.

4.2.1. Etape 2.1 : Définir les services atomiques

Selon le processus de génération du modèle $\mathcal{MiS}$, les services atomiques sont identifiés en associant à chaque section opérationnalisable de la carte un service atomique.

La carte e-Pension compte treize sections opérationnalisables. Par conséquent, nous identifions treize services atomiques récapitulés au Tableau 20.

Tableau 28. Identification des services atomiques à partir de la carte *e-Pension*

Section de la carte	Service atomique
ab_1	S *Authentifier le citoyen*
ab_2	S *Formuler la demande par capture d'informations*
bb_1	S *Formuler la demande par attestation de domiciliation*
bb_2	S *Attribuer un rendez-vous médical*
bd_1	S *Annuler la demande de pension*
bc_1	S *Passer un examen physique*

bc_2	S *Passer un examen physique approuvé*
cc_1	S *Statuer sur la demande par pré-décision du LHA*
cc_2	S *Contrôler la demande de pension*
cc_3	S *Statuer sur la demande par décision de la préfecture*
cd_1	S *Arrêter par désistement du citoyen*
cd_2	S *Arrêter par rejet de la demande de pension*
cd_3	S *Transférer la demande au service de traitement des pensions*

4.2.2. Etape 2.2 : Définir les services agrégats

Selon le processus de génération du modèle $\mathcal{MiS}$, la définition des services agrégats se fait en appliquant deux sous étapes à savoir (i) générer tous les chemins de la carte et (ii) identifier les services agrégats.

Les deux sous sections suivantes décrivent l'application de chacune de ces sous étapes.

4.2.2.1. Générer tous les chemins de la carte

Les enchaînements des sections dans une carte identifient diverses sortes de services. Cependant, il est difficile d'identifier toutes les possibilités de manière exhaustive. Nous avons choisi d'utiliser l'algorithme de MacNaughton et Yamada [MacNaughton60] et de l'appliquer au modèle de la carte afin d'identifier tous les chemins possibles entre un but source *Démarrer* et un but cible *Arrêter*.

Durant cette étape, les chemins de la carte e-Pension sont décrits en premier lieu par des formules mathématiques propres au formalisme de l'algorithme. Par la suite, des mises en correspondance sont appliquées afin de définir les relations entre les sections de la carte.

La formule décrivant tous les chemins entre le but *a* et le but *d* de la carte e-Pension est :

$$\mathbf{Y_{a,\{a,b,c,d\},d} = \bullet\ (X_{ab},\ X^*_{bb},\ \cup(\bullet(X_{bc},\ X^*_{cc},\ X_{cd}),\ X_{bd}))}$$

Cette formule nous indique que pour aller de *a* à *d*, il y a une séquence à suivre. En effet, la formule X_{ab} est utilisée pour satisfaire le but *b* à partir du but *a*. Ensuite, X_{bb} est utilisée pour aller de *b* vers *b*. Pour aller de *b* à *d*, il faut choisir l'un des deux chemins suivants : soit aller de *b* à *d* directement (cette possibilité est décrite par la formule X_{bd}), soit aller de *b* à *c* puis de *c* à *c* et enfin de *c* à *d* (ce second chemin est exprimé par la formule $\bullet(X_{bc}, X^*_{cc}, X_{cd})$).

Spécialisation des X_{st} en sections de la carte e-Pension et relations entre elles

La spécialisation de chacun des X_{st} est présentée au Tableau 23.

Tableau 29. Liste des relations entre les sections de la carte *e-Pension*

Type de la relation	Relation identifiée
Paquet	$P_{bc}= \otimes (bc_1, bc_2)$ $P_{cd}= \otimes (cd_1, cd_2, cd_3)$
Multi-Segment	$MS_{ab} = \vee (ab_1, ab_2)$ $MS_{bb}= \vee (bb_1, bb_2)$ $MS_{cc}= \vee (cc_1, cc_2, cc_3)$
Chemin	$C_{b,\{c\},d}= \bullet (P_{bc}, MS^*_{cc}, P_{cd})$ $C_{a,\{b,c\},d}= \bullet (MS_{ab}, MS^*_{bb}, MC_{b,\{c\},d})$
Multi-Chemin	$MC_{b,\{c\},d}= \cup (bd_1, C_{b,\{c\},d})$

Nous pouvons vérifier les résultats obtenus dans le Tableau 23 à partir de la carte *e-Pension*. Le chemin $C_{a,\{b,c\},d}$ définit toutes les compositions possibles de services entre le but source *Démarrer* (ayant le code *a*) et le but cible *Arrêter* (ayant le code *b*). Au début, il y un choix d'une ou plusieurs possibilités du multi-segment MS_{ab} pour atteindre le but *b*. L'étape suivante consiste à choisir une ou plusieurs possibilités du multi-segment MS_{bb} pour atteindre le but b. Ensuite, il y a deux chemins pour satisfaire le but *d* à partir du but *b* qui sont décrits par le multi-chemin $MC_{b,\{c\},d}$. Le premier chemin consiste à choisir bd_1. Le deuxième chemin est $C_{b,\{c\},d}$ qui suggère d'atteindre le but *c* en choisissant une alternative du paquet P_{bc}, puis atteindre le but *c* en choisissant une ou plusieurs possibilités du multi-segment MS_{cc} puis réaliser le but *d* en choisissant une alternative du paquet P_{cd}.

4.2.2.2. Identifier les services agrégats

Nous proposons dans cette section, d'identifier les services agrégats à l'aide d'un ensemble de règles présentées en détail au chapitre 5 (cf. section 3.2.2). Ces règles permettent la mise en correspondance entre les relations entre les sections de la carte e-Pension et les services correspondants.

Les règles à appliquer consistent à (i) affecter à chaque chemin de la carte un service composite séquentiel, (ii) affecter à chaque multi-chemin de la carte un service à variation de chemin, (iii) affecter à chaque paquet de la carte un service à choix alternatif, et (iv) affecter à chaque multi-segment de la carte un service à choix multiple.

L'application de cette étape sur la carte e-Pension donne les services agrégats résumés au Tableau 24.

Tableau 30. Liste des services agrégats générés à partir de la carte *e-Pension*

Type de la relation	Service identifié

Paquet	$P_{bc}= \otimes (bc_1, bc_2)$ $P_{cd}= \otimes (cd_1, cd_2, cd_3)$	Service à choix alternatif	$S_{\textit{Passer un examen physique}} = \otimes (S_{\textit{Passer un examen physique}}, S_{\textit{Passer un examen physique certifié}})$ $S_{\textit{Prise de décision}}= \otimes (S_{\textit{Arrêter par désistement du citoyen}}, S_{\textit{Arrêter par rejet de la demande de pension}}, S_{\textit{Transférer la demande au service de traitement des pensions}})$
Multi-Segment	$MS_{ab} = \vee (ab_1, ab_2)$ $MS_{bb}= \vee (bb_1, bb_2)$ $MS_{cc}= \vee (cc_1, cc_2, cc_3)$	Service à choix multiple	$S_{\textit{Formuler la demande de pension}}= \vee (S_{\textit{Authentifier le citoyen}}, S_{\textit{Formuler la demande par capture d'informations}})$ $S_{\textit{Compléter la formulation de la demande de pension}}= \vee (S_{\textit{Formuler la demande par attestation de domiciliation}}, S_{\textit{Attribuer un rendez-vous médical}})$ $S_{\textit{Statuer sur la demande}} = \vee (S_{\textit{Statuer sur la demande par pré-décision du LHA}}, S_{\textit{Contrôler la demande de pension,}} S_{\textit{Statuer sur la demande par décision de la préfecture}})$
Chemin	$C_{b,\{c\},d}= \bullet (P_{bc}, MS^*_{cc}, P_{cd})$ $C_{a,\{b,c\},d}= \bullet (MS_{ab}, MS^*_{bb}, MC_{b,\{c\},d})$	Service composite séquentiel	$S_{\textit{Traiter la demande de pension}} = \bullet (S_{\textit{Passer un examen physique}}, S^*_{\textit{Statuer sur la demande}}, S_{\textit{Prendre la décision relative à une demande}})$ $S_{\textit{Aider les personnes à obtenir une pension}}= \bullet (S_{\textit{Formuler la demande de pension}}, S^*_{\textit{Compléter la formulation de la demande de pension}}, S_{\textit{Gérer la demande de pension.}})$
Multi-Chemin	$MC_{b,\{c\},d}= \cup (bd_1, C_{b,\{c\},d})$	Service à variation de chemin	$S_{\textit{Gérer la demande de pension}} = \cup (S_{\textit{Annuler la demande de pension}}, S_{\textit{Traiter la demande de pension}})$

Le service agrégat qui permet d'aider les personnes handicapées à obtenir une pension est $S_{\textit{Aider les personnes à obtenir une pension}}$. Ce service est une composition de trois services : $S_{\textit{Formuler la demande de pension}}$, $S_{\textit{Compléter la formulation de la demande}}$ et $S_{\textit{Gérer la demande de pension}}$ Le premier service est un service à variation multiple qui permet de formuler la demande du citoyen à travers la réalisation du but *Formuler la demande*. Ceci consiste à

- Sécuriser l'accès aux données personnelles des citoyens $S_{\textit{Authentifier le citoyen}}$ et
- Capturer les informations personnelles constituant la requête du citoyen $S_{\textit{Formuler la demande par capture d'informations}}$.

Le deuxième service $S_{\textit{Compléter la formulation de la demande}}$ est un service à variation multiple qui permet au citoyen de compléter la formulation de sa demande de pension à travers la réalisation du but *Formuler la demande*. Ceci consiste à :

- Faciliter l'obtention de l'attestation de domiciliation S *Formuler la demande par attestation de domiciliation* et
- Permettre au citoyen d'obtenir un rendez-vous médical aisément S *Attribuer un rendez-vous médical*.

On note que le service S *Compléter la formulation de la demande* peut s'itérer lorsque cela est nécessaire.

Enfin, le troisième service S *Gérer la demande de pension* est un multi-chemin. Le premier chemin consiste à exécuter le service atomique S *Annuler la demande de pension*. Ceci consiste à arrêter le processus par le citoyen en annulant la demande de pension. Le deuxième chemin consiste à traiter la demande de pension une fois formulée à l'aide du service composite séquentiel S *Traiter la demande de pension*. S *Traiter la demande de pension* est une composition de trois services : S *Passer un examen physique*, S *Statuer sur la demande* et S *Prise de décision*.

Le service S *Passer un examen physique* est un service à choix alternatif qui permet de réaliser partiellement le but *Statuer sur la demande*. Ceci consiste à choisir l'une des alternatives suivantes :

- Passer l'examen médical au LHA S *Passer un examen physique* et,
- Passer l'examen médical chez un médecin assermenté par l'organisation e-Pension S *Passer un examen physique approuvé*.

Le service S *Statuer sur la demande* est un service à choix multiple qui consiste à :

- Contrôler le processus ainsi que la prise de la décision finale relative à une demande spécifique S *Statuer sur la demande par contrôle de la Préfecture*,
- Prendre la pré-décision relative à une demande de pension S *Statuer sur la demande par pré-décision du LHA* et
- Prendre la décision finale S *Statuer sur la demande par décision de la Préfecture*.

Le service S *Prise de décision* est un service à choix alternatif qui permet de traiter la demande suite à la prise de décision. Ceci consiste à choisir l'une des alternatives suivantes :

- Transférer la demande de pension au service de traitement des pensions dans le cas d'une réponse positive S *Transférer la demande*,
- Rejeter la demande de pension dans le cas d'une décision négative S *Arrêter par rejet de la demande* et
- Arrêter le processus en retirant la demande S *Arrêter par désistement du citoyen*.

4.2.3. Description des services du modèle MiS

Nous présentons, dans cette section, une structure de description textuelle de trois services intentionnels basée sur le standard XML (e**X**tensible **M**arkup **L**anguage). La DTD (**D**efinition **T**ype de **D**onnées) correspondant à la structure descriptive du service intentionnel est détaillée au Chapitre 4 (cf. section 4).

La Figure 93 montre un exemple de description formulée en XML 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 93. Description du service atomique $S_{Authentifier\ le\ citoyen}$

La Figure 94 montre un exemple de description formulée en XML du service atomique qui permet de capturer les informations personnelles constituant la requête du citoyen $S_{Formuler\ la\ demande\ par\ capture\ d'informations}$.

```
<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'∧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 94. Description du service atomique S *Formuler la demande par capture d'informations*

La Figure 95 montre un exemple de description formulée en XML du service à choix multiple qui 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 S *Formuler la demande de pension*.

```
<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'∧demande.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 95. Description du service à choix multiple S *Formuler la demande de pension*

5. CONSTRUCTION DE L'APPLICATION *E-PENSION* A BASE DE SERVICES DU MODELE $\mathcal{M}os$

La construction de l'application e-Pension à base de services du modèle $\mathcal{M}os$ est mise en œuvre en introduisant une démarche méthodologique qui a été expliquée en détail au chapitre

4. Elle consiste à guider l'identification, au niveau opérationnel, de services du modèle $\mathcal{M}_{oS}$, qui sont un moyen d'opérationnalisation de chaque service intentionnel atomique du modèle $\mathcal{M}_{iS}$.

La démarche de construction du modèle $\mathcal{M}_{oS}$ comporte trois étapes :

- Etape 1 : Ecrire le scénario de base et découvrir les exceptions,
- Etape 2 : Construire le modèle $\mathcal{M}_{oS}$ par analyse des scénarios et,
- Etape 3 : Identifier le modèle d'implémentation.

Les étapes de la démarche ont été prises en compte dans l'élaboration de l'application e-Pension.

Nous avons choisi le service intentionnel atomique $S_{\textit{Attribuer un rendez-vous médical}}$ pour illustrer le déroulement de chacune de ces étapes.

Le déroulement de chacune de ces sous étapes, appliqué au service $S_{\textit{Attribuer un rendez-vous médical}}$ de e-Pension, est détaillé dans les sous sections ci-après de la manière suivante : La sous section 5.1 illustre la construction des scénarios permettant de décrire le comportement d'un service atomique dans la satisfaction du but associé ; La sous section 5.2 permet de découvrir les éléments constituant le service logiciel du modèle $\mathcal{M}_{oS}$ à partir des scénarios obtenus (scénario de base et scénarios alternatifs de succès ou d'échecs); La sous section 5.3 montre l'implémentation de ces services à l'aide de la technologie des services web.

5.1. Etape1 : Ecrire le scénario de base et découvrir les exceptions

Considérons le service intentionnel atomique $S_{\textit{Attribuer un rendez-vous médical}}$ ayant comme but *Attribuer un rendez-vous médical*.

On se base sur les directives de style et celles de contenu présentées au chapitre 4 (cf. section 4.1) pour écrire les flux d'actions puis on se sert 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 96. Le scénario relatif au service S *Attribuer un rendez-vous médical*

5.1.1. Recherche d'exception

La découverte de nouveaux scénarios alternatifs à partir d'un scénario de base 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 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 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

On suggère de construire les combinaisons possibles avec les négations des conditions. Chaque imbrication générée à partir des négations de conditions identifie une manière alternative de satisfaction/ non satisfaction du but.

Pour la négation de condition identifiée, on doit déterminer s'il s'agit d'un cas de succès ou d'échec :

1- Il n'existe pas de disponibilités .

La manière correspondante au cas identifié doit permettre de reconnaître s'il s'agit d'un cas de succès ou d'un cas d'échec.

Manière	Type de cas
Il n'existe pas de disponibilités	Echec

Le résultat obtenu comporte un nouveau scénario alternatif d'échec puisqu'il ne permet pas de satisfaire le but *Attribuer un rendez-vous médical*.

Le scénario suivant (cf. Figure 97) est le scénario associé au but *Attribuer un rendez-vous médical* dans le cas où il n'existerait pas de disponibilités de rendez-vous. Les interactions notées en style italique proviennent du scénario de la Figure 96. Les interactions notées en gras sont spécifiques au scénario associé au but *Attribuer un rendez-vous médical* quand il n'existe pas de disponibilités.

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 97. Le scénario alternatif relatif au service S ***Attribuer un rendez-vous médical***

5.2. Etape 2 : Construire le modèle $\mathcal{M}_{oS}$ par analyse des scénarios

L'objectif de cette étape est de découvrir les éléments constituant le service logiciel du modèle $\mathcal{M}_{oS}$ à partir des scénarios obtenus lors de l'étape 1 (Ecrire le scénario de base et découvrir les exceptions).

La génération du modèle $\mathcal{M}_{oS}$, à partir des scénarios, est réalisée par application des deux sous étapes suivantes :

- Etape 2.1 : Construire le modèle $\mathcal{M}_{oS}$ par analyse du scénario de base et,
- Etape 2.3 : Construire le modèle $\mathcal{M}_{oS}$ par analyse des scénarios alternatifs d'échec.

L'étape 2.2, qui permet de construire le modèle $\mathcal{M}_{oS}$ par analyse des scénarios alternatifs de succès, n'est pas appliquée dans notre cas. En effet, compte tenu du résultat obtenu à la section dédiée à la recherche des exceptions (cf. section5.1.1), le scénario S *Attribuer un rendez-vous médical* comporte deux nouveaux scénarios alternatifs d'échec puisqu'ils ne permettent pas de satisfaire le but *Attribuer un rendez-vous médical*.

5.2.1. Etape 2.1 : Construire le modèle $\mathcal{M}_{oS}$ par analyse du scénario de base

La génération du modèle $\mathcal{M}_{oS}$, à partir du scénario de base S *Attribuer un rendez-vous médical*, est réalisée par application des quatre sous étapes suivantes :

- Etape 2.1.1 : Spécialiser les agents du scénario,
- Etape 2.1.2 : Modéliser le service d'interface utilisateur,
- Etape 2.1.3 : Modéliser les services métier et,
- Etape 2.1.4 : Modéliser le service de coordination.

qui sont décrites dans les sections suivantes.

5.2.1.1. Etape 2.1.1 : Spécialiser les agents du scénario

L'application de l'étape 2.1.1 au scénario S *Attribuer un rendez-vous médical* aboutit à identifier les trois agents :

- Le Citoyen : correspond au *Demandeur* du service.
- E-Pension : est découpée en deux rôles du système : *Interface* et *Coordonnateur*.
- LHA : correspond au *Fournisseur* 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.

5.2.1.2. Etape 2.1.2 : Modéliser le service d'interface utilisateur

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 (cf. section 3.2.1.2).

- La DSU1 consiste à extraire les communications utilisateur à partir du scénario de base S *Attribuer un rendez-vous médical*. On distingue les quatre actions suivantes numérotées 1, 7, 9 et 13 :

> *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.*
> *13. 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 13 sont de type *sortie*.

- La DSU2 permet d'identifier, à partir des communications utilisateur, les trois rôles à savoir : Demandeur, Interface et Coordonnateur.

En appliquant la DSU2 aux communications utilisateur du scénario S *Attribuer un rendez-vous médical*, on identifie deux agents : le citoyen qui joue le rôle de Demandeur et e-Pension qui joue à la fois le rôle d'Interface et de Coordonnateur.

- L'étape 3 (DSU3) 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 la DSU3 aux quatre communications utilisateur permet d'identifier les interactions de base du service d'interface utilisateur suivantes (cf. Tableau 31) :

Tableau 31. 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 (Coordonnateur)	métier	demande utilisateur
	ReçoitListeRDV	e-Pension (Coordonnateur)	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 (Interface)	e-Pension (Coordonnateur)	métier	demande utilisateur
	RéçoitAccuséReception	e-Pension (Coordonnateur)	e-Pension (Interface)	métier	réponse utilisateur
13	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.

- La DSU4 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 la DSU4 aux communications utilisateur du scénario S *Attribuer un rendez-vous médical*, on obtient la liste des ressources présentées au Tableau 32.

Tableau 32. Liste des ressources dans un service d'interface utilisateur

Communication utilisateur	Ressource
1. Le citoyen formule une demande d'attribution d'un rendez-vous médical au système e-Pension.	Demande_Rendez-Vous

7. E-Pension notifie la liste de rendez-vous au citoyen	Liste (Rendez_vous)
9. Le citoyen confirme le rendez-vous à e-Pension	Rendez_Vous_Séléctionné
13. E-Pension envoie la confirmation de rendez-vous au citoyen	Confirmer_Rendez_Vous

Le Tableau 33 présente les valeurs requises par le modèle $\mathcal{MoS}$ pour chacune des interactions de base du service d'interface utilisateur.

Tableau 33. Valeurs requises par $\mathcal{MoS}$ pour chacune des interactions

N°	Interaction	Agent source	Agent cible	Type	Sens	Ressource
1	FormuleDemandeRDVMédical	Citoyen	e-Pension (Interface)	utilisateur	entrée	Demande_Rendez-Vous
	EnvoieDemandeRDVMédical	e-Pension (Interface)	e-Pension (Coordonnateur)	métier	demande utilisateur	Demande_Rendez-Vous
	ReçoitListeRDV	e-Pension (Coordonnateur)	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 (Coordonnateur)	métier	demande utilisateur	Rendez_Vous_Séléctionné
	RéçoitAccuséReception	e-Pension (Coordonnateur)	e-Pension (Interface)	métier	réponse utilisateur	Confirmer_Rendez_Vous
13	AfficheAccuséReception	e-Pension (Interface)	Citoyen	utilisateur	sortie	Confirmer_Rendez_Vous

- La règle DSU5 sert à compléter la modélisation du service d'interface utilisateur en identifiant les interactions structurées de type contrainte.

Cette règle commence par l'extraction des cas de contrainte existant dans le scénario.

5. S'il existe des disponibilités (C1)

Cette 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

34 dispose la contrainte par rapport aux interactions de base.

Tableau 34. Disposition des contraintes par rapport aux 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 (Coordonnateur)	métier	demande utilisateur
	Contrainte C1	Contrainte			
	ReçoitListeRDV	e-Pension (Coordonnateur)	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 (Interface)	e-Pension (Coordonnateur)	métier	demande utilisateur
	RéçoitAccuséReception	e-Pension (Coordonnateur)	e-Pension (Interface)	métier	réponse utilisateur
13	AfficheAccuséReception	e-Pension (Interface)	Citoyen	utilisateur	sortie

5.2.1.3. Etape 2.1.3 : Modéliser le service métier

La modélisation du service métier, 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 (cf. section 3.2.1.3).

L'application de ces étapes aide à modéliser progressivement les éléments des services métier de e-Pension selon les termes de $\mathcal{MoS}$.

- La DSM1 consiste à extraire les actions de communication qui représentent une invocation de service métier.

En appliquant la DSM1 au scénario S *Attribuer un rendez-vous médical*, on obtient les deux communications numérotées 3 et 9 :

3. E-Pension demande rendez-vous au LHA.

9. E-Pension confirme le rendez-vous au LHA.

- La DSM2 aide à découvrir les opérations du service métier 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 la DSM2 au scénario S *Attribuer un rendez-vous médical* permet d'obtenir les couples suivants :

a. *3. E-Pension demande rendez-vous au LHA : 6. Le LHA confirme la liste de rendez-vous à e-Pension.*
b. *9. E-Pension confirme le rendez-vous au LHA: 11. Le LHA envoie une confirmation du rendez-vous à e-Pension.*

Le Tableau 35 présente les opérations obtenues à partir des couples de communications déjà identifiés :

Tableau 35. Liste des opérations de type demande/réponse

Liste des C_{Appel}	**Liste des opérations identifiées**
3. E-Pension demande rendez-vous au LHA.	Demander rendez-vous ()
9. E-Pension confirme le rendez-vous au LHA.	Confirmer rendez-vous ()

- Il est à noter que la DSM3 n'est pas appliquée étant donné que le scénario S *Attribuer un rendez-vous médical* ne présente aucune opération de type unidirectionnel.

- La DSM4 aide à déterminer les messages en entrée et en sortie de chacune des opérations. L'application de l'étape DSM4 au scénario S *Attribuer un rendez-vous médical*, donne la spécification de chaque opération présentée au Tableau 36.

Tableau 36. 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

- La DSM5 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.

L'application de l'étape DSM6 au scénario S *Attribuer un rendez-vous médical*, permet d'obtenir l'interface présentée au Tableau 37.

Tableau 37. Liste des interfaces relatives aux services métier

Liste des agents fournisseur	Liste des opérations	Les interfaces des services métier
LHA	Demander rendez-vous () Confirmer rendez vous ()	ServiceLHA

5.2.1.4. Etape 2.1.4 : Modéliser le service de coordination

La modélisation du service de coordination, 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 (cf. section 3.2.1.4).

Chacune de ces étapes permet d'identifier un des aspects de la description d'un service de coordination dans e-Pension selon les termes du modèle $\mathcal{MoS}$.

- L'étape DSC1 consiste à extraire, à partir du scénario, les actions de communication $C_{Coordination}$ qui ont comme source ou cible un agent qui a le rôle de Coordonnateur.
Les $C_{Coordination}$ correspondent aux actions échangées à partir de et vers le Coordonnateur pour réaliser l'objectif de la coordination. Par conséquent, les $C_{Coordination}$ correspondent aux activités de base du service de coordination.
Le Tableau 38 résume les communications $C_{Coordination}$ relatives au scénario S *Attribuer un rendez-vous médical* et les actions de base correspondantes.

Tableau 38. Liste des $C_{Coordination}$ et leurs activités de base

Liste des $C_{Coordination}$	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	
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
8. Le citoyen choisit une date de rendez-vous à e-Pension.	SélectionRDV
9. E-Pension confirme le rendez-vous au LHA	Confirmer 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	RéçoitAccuséReception

Une fois les $C_{Coordination}$ identifiées, leur type est spécifié comme suit (cf. Tableau 39).

Tableau 39. Liste des $C_{Coordination}$ et leurs types respectifs

Liste des $C_{Coordination}$	Type	Relation avec les services métier ou d'interface utilisateur
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	affectation	
3. E-Pension demande rendez-vous au LHA	appel	Demander rendez-vous (service 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*)
8. Le citoyen choisit une date de rendez-vous à e-Pension.	réception	SélectionRDV (Interface *Attribuer un rendez-vous médical*)
9. E-Pension confirme le rendez-vous au LHA	appel	Confirmer rendez-vous (service ServiceLHA)
11. Le LHA envoie une confirmation du rendez-vous à e-Pension.		
12. E-Pension envoie la confirmation de rendez-vous au citoyen	réponse	RéçoitAccuséReception (Interface *Attribuer un rendez-vous médical*)

- La DSC2 propose de déterminer (i) les agents participant à la coordination, (ii) les services métier invoqués et (iii) les services d'interface utilisateur invoquant le service de coordination.

L'étape DSC2 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.
- Le service métier fournis par le participant tel que le ServiceLHA.
- Le service d'interface utilisateur 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 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.

Tableau 40. Les activités de base et leurs relations avec les autres services

Liste des $C_{Coordination}$	Type	Relation avec les services métier 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	affectation	

de rendez-vous médical		
3. E-Pension demande rendez-vous au LHA	appel	service 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	Interface *Attribuer un rendez-vous médical*
8. Le citoyen choisit une date de rendez-vous à e-Pension.	réception	Interface *Attribuer un rendez-vous médical*
9. E-Pension confirme le rendez-vous au LHA	appel	service ServiceLHA
11. Le LHA envoie une confirmation du rendez-vous à e-Pension.		
12. E-Pension envoie la confirmation de rendez-vous au citoyen	réponse	Interface *Attribuer un rendez-vous médical*

- La DSC3 aide à construire le modèle de données d'un service de coordination.

En appliquant la DSC3 au scénario S *Attribuer un rendez-vous médical*, nous obtenons le modèle de données suivant (cf. Tableau 41) :

Tableau 41. Liste des ressources formant le modèle de données

Liste des $C_{Coordination}$	Ressources identifiées
1. Le citoyen formule une demande d'attribution d'un rendez-vous médical au système e-Pension	Demander_rendez_vous_médical
6. Le LHA confirme la liste de rendez-vous à e-Pension	Liste_rendez_vous
8. Le citoyen choisit une date de rendez-vous à e-Pension.	Rendez_vous
11. Le LHA envoie une confirmation du rendez-vous à e-Pension	Confirmation_rendez_vous

- La DSC4 définit le modèle d'orchestration du service de coordination en considérant l'ordre d'exécution des activités de base. Le résultat obtenu correspond à l'identification de l'activité structurée.

La DSC4 appliquée 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 la DSC5 pour le contexte de l'exemple e-Pension vu la non nécessité de l'adaptation des données.

5.2.2. Etape 2.3 : Construire le modèle $\mathcal{M}_{oS}$ par analyse des scénarios alternatifs d'échec

Cette section permet de construire le modèle $\mathcal{M}_{oS}$ 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 $\mathcal{M}_{oS}$, à partir du scénario alternatif d'échec, ne considère que les actions précédemment présentées à la Figure 97 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 re-appliquer le processus proposé à l'étape 2.1 (cf. section 5.2.1) et à l'étendre avec des règles supplémentaires.

5.2.2.1. Spécialiser les agents du scénario

L'application de la sous étape 2.1.1, précise les agents suivants et leurs rôles respectifs :

- Le citoyen : correspond au rôle de *Demandeur* du service
- E-Pension : est découpé en deux rôles systèmes celui de l'*Interface* et celui de *Coordonnateur*.

5.2.2.2. Modéliser le service d'interface utilisateur

L'application de l'étape 2.1.2, selon la séquence des règles (DSU1 à DSU5), modélise le service d'interface utilisateur.

- La DSU1 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.

- La DSU2 identifie, à partir des communications utilisateur, les rôles Demandeur, Interface et Coordonnateur. Cette étape va servir à par la suite pour déterminer le type de chacune des interactions de base.

Les rôles identifiés, à partir de la communication 6, sont les suivants : le citoyen qui jour le rôle de Demandeur et e-Pension qui joue à la fois le rôle d'Interface et de Coordonnateur.

- La DSU3 détermine le type de l'interaction de base *ReformulationDemande* comme le montre le Tableau 42 :

Tableau 42. Type des interactions de base

N°	Interaction	Agent source	Agent cible	Type	Sens
6	ReformulationDemande	e-Pension (Interface)	Citoyen	utilisateur	sortie

- La DSU4 complète la description de chaque interaction de base par l'identification des ressources échangées. Nous présentons le résultat obtenu dans le Tableau 43.

Tableau 43. Liste des ressources

N°	Interaction	Agent source	Agent cible	Type	Sens	Ressource
6	ReformulationDemande	e-Pension (Interface)	Citoyen	utilisateur	sortie	Demande Reformulation

- La DSU5 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 44 montre la disposition de la contrainte identifiée par rapport à l'interaction de base *ReformulationDemande*.

Tableau 44. 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

- La DSC7 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

5.2.2.3. Modéliser le service métier

Habituellement, pour notre processus, l'objectif de cette étape consiste à modéliser les services métier invoqués à partir du scénario alternatif en appliquant la séquence des règles (DSM1 à DSM5). En outre, dans le contexte de cet exemple, nous n'identifions pas de services métier vu que les actions du scénario alternatif d'échec n'ont pas de lien avec un agent ayant le rôle Fournisseur. L'exécution de cette étape sur l'exemple ne produit pas de nouveaux services métier.

5.2.2.4. Modéliser le service de coordination

L'application de l'étape 2.1.4, selon la séquence des règles (DSC1 à DSC6), complète la modélisation du service de coordination par la définition des activités de compensation.

- La DSC1 consiste à extraire les actions de communications qui ont comme source ou cible un agent dont le rôle est Coordonnateur à savoir.

6. E-Pension demande au citoyen de reformuler sa demande

Cette action représente une activité de base, *ReformulationDemande*, de type *réponse* entre le service de coordination et le demandeur (cf. Tableau 45).

Tableau 45. Liste des activités de base et leurs types respectifs

Liste des activités de base		Type
6	ReformulationDemande	réponse

- La DSC2 consiste à déterminer (i) les agents participant à la coordination, (ii) les services métier invoqués et (iii) les services d'interface utilisateur invoquant le service de coordination.

– Le service d'interface utilisateur 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 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.

Tableau 46. Les activités de base et leurs relations avec les autres services

Liste des activités de base		Type	Relation avec les services métier ou d'interface utilisateur
6	ReformulationDemande	Réponse	Interface *Attribuer un rendez-vous médical*

- La DSC3 identifie les ressources échangées dans une coordination. Nous présentons le résultat obtenu dans le Tableau 47 :

Tableau 47. Liste des ressources

Liste des activités de base		Ressources identifiées
6	ReformulationDemande	Demander_Reformulation

- La DSC4 définit le modèle d'orchestration du service de coordination en considérant l'ordre d'exécution des activités de base précédemment identifiées. Dans notre cas, une seule action de base est identifiée et constitue la séquence des actions. Il n'y a pas d'ordre de séquencement des actions à établir vu que la séquence est composée d'une seule action. Cette action est la première et la dernière de la séquence.

- La DSC5 a pour but d'identifier les activités de type affectation nécessaires pour adapter les données entre le service d'interface utilisateur et les services métier

- La DSC8 a pour but d'alimenter le service de coordination par les activités de base déjà identifiées. Ces activités représentent des activités de compensation.

L'application de la règle DSC8 permet d'alimenter le service de coordination par l'activité 6 comme étant une activité de compensation produite.

5.3. Etape 3 : Identifier le modèle d'implémentation

L'exemple de la Figure 98 illustre le document WSDL obtenu par la transformation du service métier du modèle $\mathcal{MOS}$ relatif au service intentionnel S *Attribuer un rendez-vous médical* et en ajoutant la partie concrète.

```
<?xml version="1.0" encoding="UTF-8"?><definitions name="Service_LHA"
targetNamespace="urn://LHA.wsdl">
<types>
 …
</types>
<message name="demandeRDV">
       <part name="demande" type="ns:demande"/>
</message>
<message name="ListeRDV">
       <part name="return" type="ns:listeRDV"/>
</message>
<message name="RendezVous">
       <part name="RendezVous" type="ns :RendezVous"/>
</message>
<message name="Confirmation">
       <part name="return" type="ns :Confirmation"/>
</message>
<message name="NotificationRejet">
       <part name="NotificationRejet" type="ns :NotificationRejet"/>
</message>
<portType name="ServiceLHA">
       <operation name="demanderRDV" parameterOrder="">
              <input message="demandeRDV"/>
              <output message="ListeRDV"/>
       </operation>
       <operation name="confirmerRDV" parameterOrder="">
              <input message="RendezVous"/>
              <output message="Confirmation"/>
       </operation>
       <operation name="rejetRDV" parameterOrder="">
              <input message="NotificationRejet"/>
       </operation>
</portType>
<binding name="LHABinding" type="ServiceLHA">
       <operation name="demanderRDV">
              <input>
                     <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                     encoding/" use="encoded" namespace="urn://LHA/
                     wsdl"/>
              </input>
```

```
                <output>
                        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                        encoding/" use="encoded" namespace="urn://LHA/
                        wsdl"/>
                </output>
                        <soap:operation soapAction=""/>
        </operation>
        <operation name="confirmerRDV">
                <input>
                        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                        encoding/" use="encoded" namespace="urn://LHA/
                        wsdl"/>
                </input>
                <output>
                        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                        encoding/" use="encoded" namespace="urn://LHA/
                        wsdl"/>
                </output>
                <soap:operation soapAction=""/>
        </operation>
        <operation name="rejetRDV">
                <input>
                        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/
                        encoding/" use="encoded" namespace="urn://LHA/
                        wsdl"/>
                </input>
                <soap:operation soapAction=""/>
        </operation>
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
</binding>
<service name="ServiceLHA">
        <port name="LHAPort" binding="LHABinding">
                <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
        </port>
</service>
</definitions>
```

Figure 98. Le document WSDL du LHA (à partir du service métier de $\mathcal{MoS}$)

L'exemple de la Figure 99 montre le document BPEL obtenu par la transformation du service de coordination du modèle $\mathcal{MoS}$ (comme composition d'autres services) relatif au service intentionnel S *Attribuer un rendez-vous médical*.

```
<process name="ServicePension" targetNamespace="http://ServicePension.org/"
suppressJoinFailure="false">
        <partnerLinks>
                <partnerLink name="ServicePension" partnerLinkType="
                ITF_PensionLink" myRole="ITF_PensionProvider"/>
                <partnerLink name="ServiceLHA" partnerLinkType="
                ITF_ServiceLHALink" partnerRole="ITF_ServiceLHAProvider"/>
        </partnerLinks>
        <variables>
                <variable name="demandeRDV" messageType="
                ITF_ServiceLHA_demanderRDV"/>
                <variable name="RDV" messageType="
                ITF_ServiceLHA_confirmerRDV"/>
```

```
        </variables>
        <sequence>
                <receive name="ReceptionDemandeRDV" partnerLink="ServicePension"
                        portType="ITF_ServicePension" operation="demanderRDV" variable="
                        demandeRDV" createInstance="true"/>
                <assign name="AssigndemandeRDV"></assign>
                <invoke name="demandeRDV" partnerLink="
                        ServiceLHA" portType="
                        ServiceLHA" operation="
                        DemanderRDV" inputVariable="
                        demandeRDV"
                        outputVariable="ListeRDV"/>
                <assign name="AssignReturn"></assign>
                <reply name="ReplyFindFlight" partnerLink="ServiceLHA" portType
                        ="ServiceLHA" operation="demanderRDV" variable="
                        ListeRDV"/>
                <terminate name="EndProcess"/>
        </sequence>
</process>
```

Figure 99. Le document BPEL d'e-Pension

6. ORCHESTRATION INTENTIONNELLE DE SERVICES-

Cette section illustre l'orchestration des services du cas e-Pension pendant la phase d'exécution. Comme on l'a montré au Chapitre 6, la sélection des services est guidée par les buts que les utilisateurs désirent atteindre. La satisfaction d'un but donné se concrétise au niveau du modèle $\mathcal{MIS}$ par l'identification de services réalisant le but et parmi lesquelles l'utilisateur choisit le service adéquat. Une fois un but satisfait, l'utilisateur peut progresser vers la réalisation d'autres buts. Au niveau de l'architecture, l'identification et l'invocation des services se fait suite à une série d'actions de délégation entre les différents agents concernés et peut nécessiter des interactions avec l'utilisateur.

Le scénario suivant est un exemple de processus d'orchestration. La carte de la Figure 92 représente les étapes suivies par l'utilisateur pour satisfaire ses buts. Les étapes sont identifiées par des numéros entre parenthèses. Les flèches en gras représentent les manières sélectionnées par l'utilisateur. Le modèle $\mathcal{MIS}$ associé décrit les services choisis.

Nous détaillons dans ce qui suit chacune des étapes.

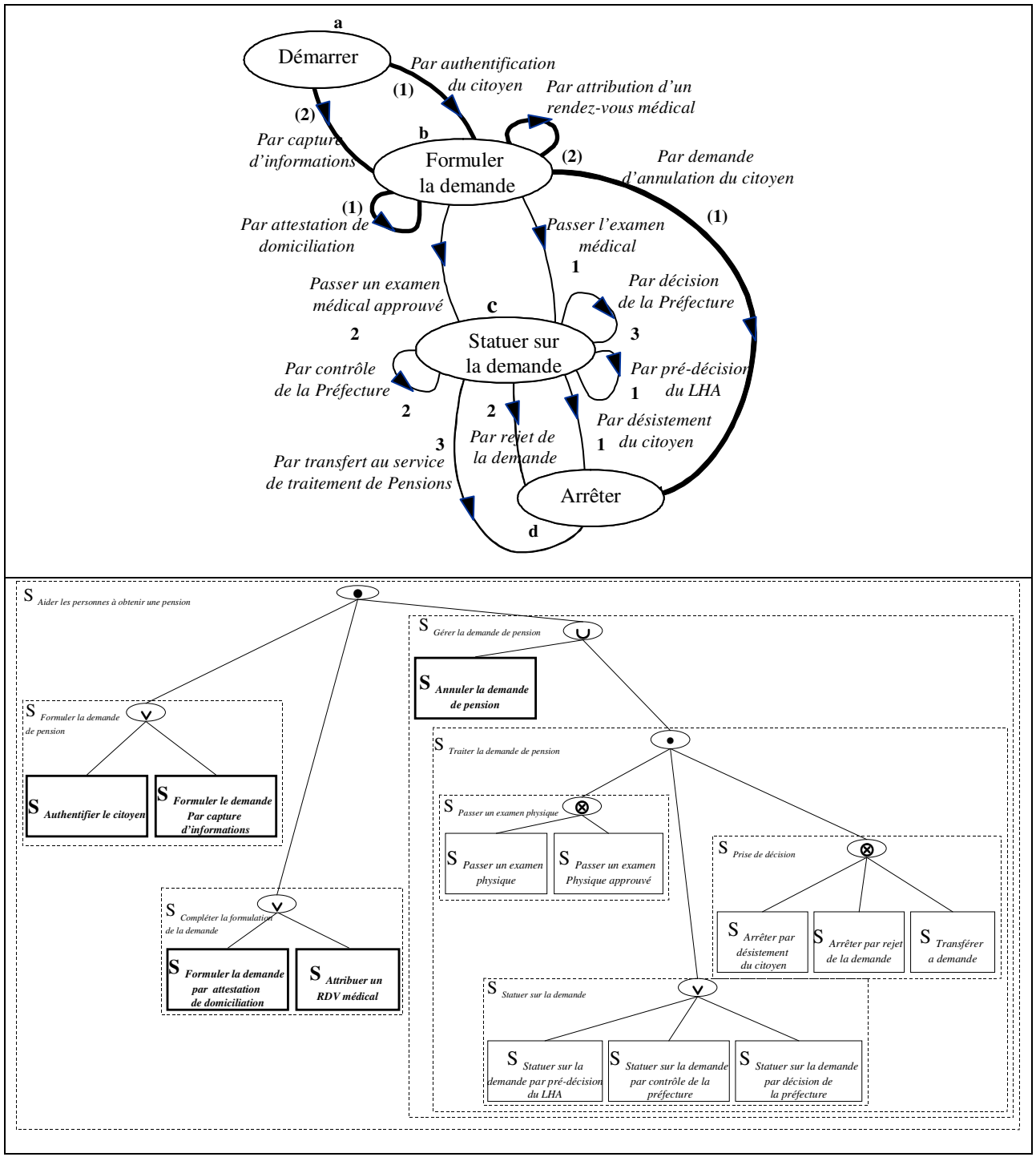


Figure 100. Illustration du parcours d'un utilisateur dans la carte et choix des services à partir du modèle $\mathcal{M}_{IS}$ associé

6.1. Etape 1 : Satisfaction du but *Formuler la demande* à partir de *Démarrer*

D'après la carte de la Figure 92, le seul but que l'utilisateur peut atteindre est *Formuler la demande* par le choix du service S *Authentifier le citoyen* et/ou S *Formuler la demande par capture d'informations*.

La Figure 101 décrit les actions réalisées par les agents pour atteindre le but *Formuler la demande*.

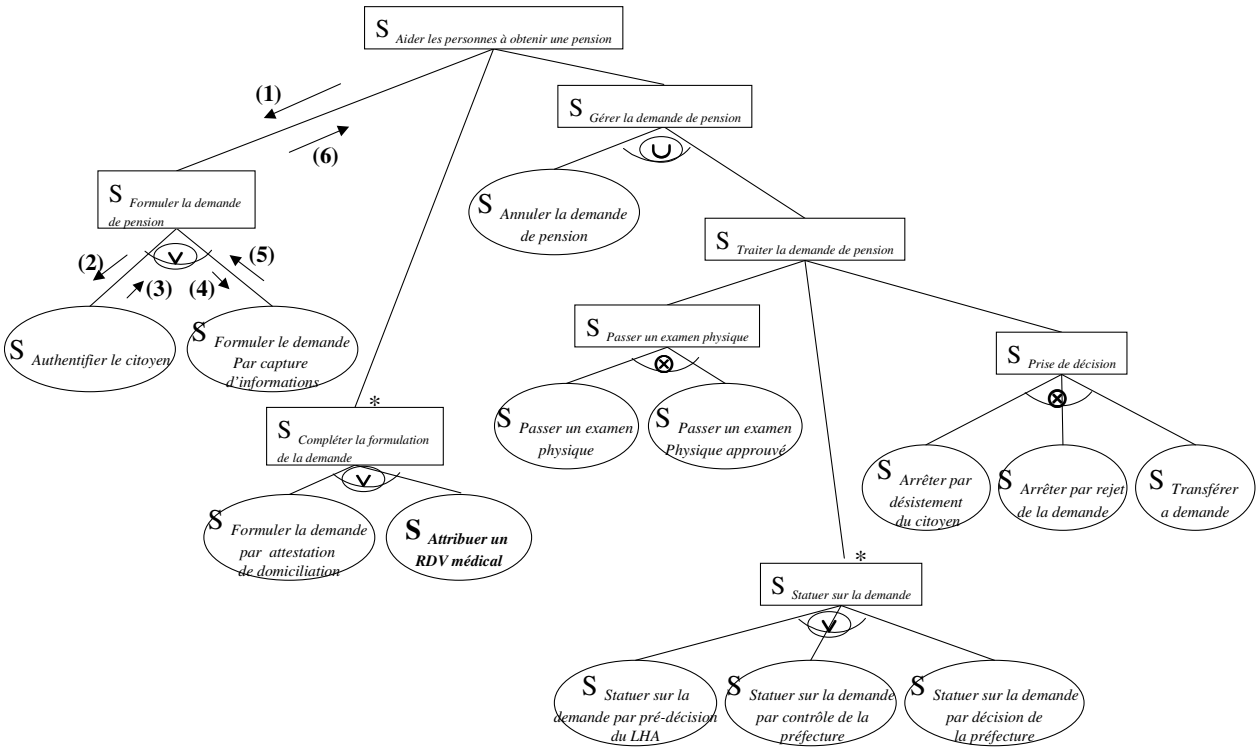


Figure 101. Illustration de l'étape1

L'étape 1 est expliquée en détail comme suit :

(1) l'agent racine S *Aider les personnes à obtenir une pension* ayant le contrôle initialement, détermine l'agent fils candidat compte tenu de la situation fournie en entrée qui est, dans notre cas la situation initiale. L'agent candidat est le premier fils S *Formuler la demande de pension*. L'agent racine invoque le fils candidat qui devient l'agent courant.

(2) L'agent S *Formuler la demande de pension* identifie les agents candidats qui sont tous ses fils à savoir S *Authentifier le citoyen* et S *Formuler la demande par capture d'informations*. Ces derniers sont proposés à l'utilisateur. Comme le montre la Figure 101 l'utilisateur choisit de satisfaire le but *Formuler la demande* par la stratégie *Par authentification du citoyen*. Cette dernière est gérée par l'exécutant S *Authentifier le citoyen.*

(3) L'exécutant S *Authentifier le citoyen* notifie à son père S *Formuler la demande de pension* la fin de son exécution. Ce dernier met à jour sa trace et recalcule les candidats qui sont dans ce cas les agents non encore choisis à savoir S *Formuler la demande par capture d'informations*.

(4) En supposant que l'utilisateur désire continuer l'exécution, l'agent S *Formuler la demande de pension* invoque l'exécutant S *Formuler la demande par capture d'informations.*

(5) L'exécutant S *Formuler la demande par capture d'informations* notifie à son père S *Formuler la demande de pension* la fin de son exécution. Ce dernier met à jour sa trace et recalcule les candidats s'ils existent. Dans notre cas, aucun agent candidat n'est présent.

(6) L'agent S *Formuler la demande de pension* notifie à l'agent racine la fin d'exécution. Nous passons dans ce cas à l'étape 2.

6.2. Etape 2 : Satisfaction du but *Formuler la demande* à partir d'une demande

La situation courante est à présent une demande créée. L'agent courant ayant le contrôle est l'agent S *Aider les personnes à obtenir une pension*.

La Figure 102 décrit les actions réalisées pour satisfaire le but *Formuler la demande* à partir d'une demande existante.

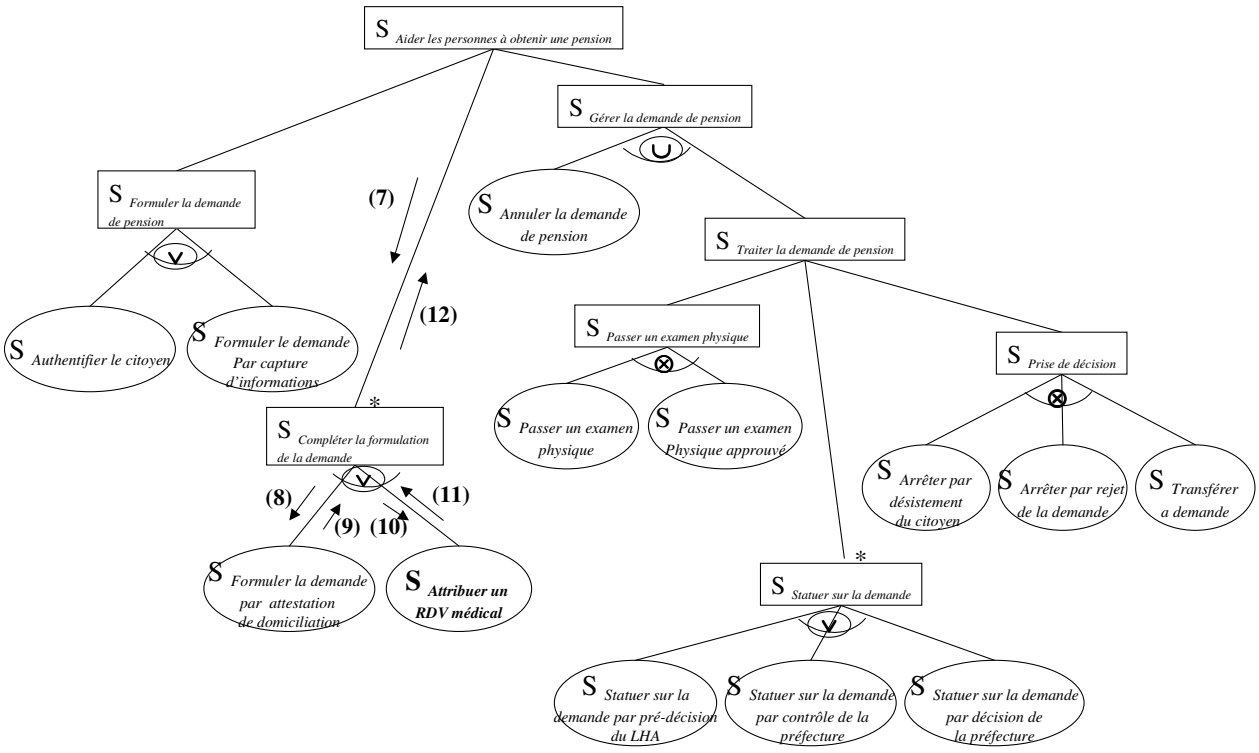


Figure 102. Illustration de l'étape 2

L'étape 2 est décrite comme suit :

(7) l'agent racine S *Aider les personnes à obtenir une pension* ayant le contrôle, détermine l'agent fils candidat compte tenu de la situation fournie en entrée qui est, dans notre cas une demande formulée. L'agent candidat est le premier fils S *Formuler la demande de pension*. L'agent racine invoque le fils candidat qui devient l'agent courant.

(8) L'agent S *Compléter la formulation de la demande de pension* identifie les agents candidats qui sont tous ses fils à savoir S *Formuler la demande par attestation de domiciliation* et S *Attribuer un rendez-vous médical*. Ces derniers sont proposés à l'utilisateur comme le montre la Figure 102, l'utilisateur choisit de satisfaire le but *Formuler la demande* par la stratégie *Par attestation de domiciliation*. Cette dernière est gérée par l'exécutant S *Formuler la demande par attestation de domiciliation.*

(9) L'exécutant S *Formuler la demande par attestation de domiciliation* notifie à son père S *Compléter la formulation de la demande de pension* la fin de son exécution. Ce dernier met à jour sa trace et recalcule les candidats qui sont dans ce cas les agents non encore choisis à savoir S *Attribuer un rendez-vous médical*.

(10) En supposant que l'utilisateur désire continuer l'exécution, l'agent S *Compléter la formulation de la demande de pension* invoque l'exécutant S *Attribuer un rendez-vous médical.*

(11)L'exécutant S *Attribuer un rendez-vous médical* notifie à son père S *Compléter la formulation de la demande de pension* la fin de son exécution. Ce dernier met à jour sa trace et recalcule les candidats s'ils existent. Dans notre cas, aucun agent candidat n'est présent.

L'agent S *Compléter la formulation de la demande de pension* notifie à l'agent racine la fin d'exécution. Nous passons dans ce cas à l'étape 3.

6.3. Etape 3 : Fin d'exécution par la progression vers le but *Arrêter*

D'après la carte à la Figure 92, il existe deux possibilités de progression une fois que le but *Formuler la demande* est atteint :

- Progresser vers le but *Statuer sur la demande* à partir d'une demande formulée par le choix d'un service parmi S *Passer un examen physique* ou S *Passer un examen physique approuvé* ou,
- Progresser vers *Arrêter* par le choix du service S *Annuler la demande de pension*.

Les deux possibilités sont contrôlées par le contrôleur de choix S *Gérer la demande de pension*.

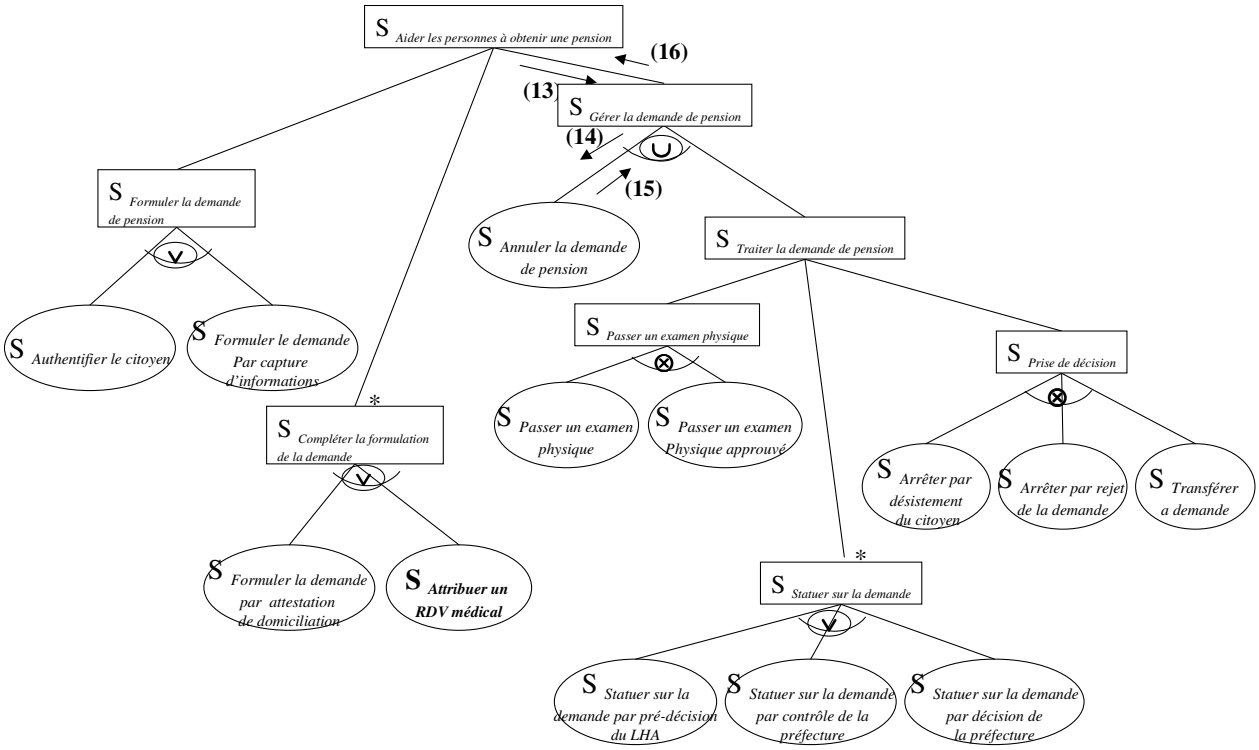


Figure 103. Illustration de l'étape 3

Dans le scénario retenu, l'utilisateur choisit d'arrêter le processus de traitement de sa pension par l'exécution du contrôleur de chemin S *Gérer la demande de pension* qui délègue à son tour l'exécution à l'agent exécutant S *Annuler la demande de pension.*

7. CONCLUSION

Ce chapitre a présenté l'application de la démarche méthodologique 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 qui correspondent aux mieux aux exigences des clients.

Cet exemple montre l'utilité qu'il y aurait à disposer d'un outi logiciel qui guide la démarche. Cet outil est en cours de réalisation.

CHAPITRE 8

Conclusion

1. CONTRIBUTIONS

Cette thèse a exploré un certain nombre de problèmes posés par un changement d'échelle visant à permettre une exploitation de SOA au niveau intentionnel (ISOA), c'est à dire celui où les souhaits, besoins et exigences sont formulés par des buts à atteindre. Nous pensons que ce changement d'échelle est nécessaire pour permettre un raisonnement en termes de services et de composition de services tel que les acteurs du monde du business peuvent/souhaitent le conduire.

Les propositions élaborées dans cette thèse pour répondre aux problèmes de ISOA et que nous avons présenté dans ce mémoire, comportent les éléments suivants :

- Un modèle intentionnel de représentation des services : le modèle $\mathcal{MiS}$,
- Un processus pour identifier les services intentionnels à partir des besoins des utilisateurs,
- Une architecture à agents pour l'orchestration intentionnelle des services du modèle $\mathcal{MiS}$ guidée par les choix des utilisateurs,
- Un modèle opérationnel de services : le modèle $\mathcal{MoS}$,

– Une démarche méthodologique pour l'opérationnalisation des services intentionnels en services du modèle *MoS*.

Le 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 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.

Un service atomique est associé à un but que l'on qualifie 'd'opérationnalisable', c'est-à-dire pour lequel on est capable de définir une séquence d'actions qui permet de réaliser le but. A l'opposé, un service agrégat cherche à satisfaire un but tactique ou stratégique et sa composition est calquée sur l'affinement ET/OU du but. La composition d'un service intentionnel introduit deux types de services pour cette raison : ceux qui sont justifiés par une décomposition OU du but, les variants et ceux qui correspondent à une décomposition ET, les composites.

Le modèle *MiS* introduit donc la variabilité dans la manière d'atteindre le but du service. Chaque variante de services correspond à une manière différente d'atteindre le but.

Nous pensons que l'encapsulation de services alternatifs dans la définition d'un service intentionnel est d'une part, inhérente au mode intentionnel (prise de décision parmi un ensemble de choix) et d'autre part, apporte la flexibilité nécessaire à l'adaptation dynamique du service intentionnel aux conditions particulières de son exécution.

Enfin, le modèle *MiS* introduit la composition de services dirigée par les buts et elle se situe à l'échelle stratégique et intentionnelle des clients. On entend par composition de services dirigée par les buts le fait que le service composite est associé à un but de haut niveau et sa composition suit la décomposition du but père en sous buts.

Nous avons défini un processus pour construire le modèle *MiS* à partir des besoins des utilisateurs. Ce processus propose d'élucider les besoins des utilisateurs à l'aide d'une carte des besoins. Nous considérons que le modèle de la carte est adapté à la découverte des services. En effet, il met en évidence les différentes stratégies pour satisfaire le même but et aussi les différentes combinaisons de stratégies et de buts pour atteindre un but. Il aide donc aussi à la découverte de la variabilité des services. Chaque combinaison possible de buts et de

stratégies identifie une variante de services possible pour atteindre le but du service global. Ce processus est détaillé au Chapitre 5.

L'architecture à agents proposée au Chapitre 6, offre un guidage au développeur pour orchestrer et implémenter les services du modèle $\mathcal{MiS}$. Les services sont implémentés sous la forme de composants architecturaux appelés *Agents* et sont structurés hiérarchiquement. Chaque agent est responsable de la satisfaction du but du service associé. De même, il peut collaborer avec d'autres agents pour la satisfaction de ce but.

Nous distinguons des agents exécutants responsables de l'exécution des services logiciels et des agents contrôleurs responsables de la gestion du choix d'autres agents.

L'orchestration des services du modèle $\mathcal{MiS}$ est réalisée durant l'exécution. Elle décrit les différentes combinaisons possibles de services et surtout en quoi celles-ci peuvent participer à la réalisation des besoins des utilisateurs. En comprenant en quoi un service peut satisfaire son besoin, l'utilisateur est en mesure de le choisir ou non. Par conséquent, le modèle $\mathcal{MiS}$ sert à l'implémentation d'une application orientée services adaptable c'est-à-dire personnalisée pour et par l'utilisateur. Cette personnalisation est dynamique puisqu'elle s'opère au moment de l'exécution du service.

Lorsque l'application adaptable s'exécute, elle offre au client les services possibles à chaque point de variation rencontré. Il y a donc personnalisation guidée et dynamique opérée par l'utilisateur final lui-même.

La vue opérationnelle des services intentionnels est développée au Chapitre 4. Ceci est réalisé, d'une part, par la présentation du modèle opérationnel de services ($\mathcal{MoS}$) 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 exigences des clients tels qu'exprimés, au niveau intentionnel, par des services intentionnels.

Le modèle $\mathcal{MoS}$ propose, à travers le concept de *service logiciel*, un service structuré en trois couches: service d'interface utilisateur, service de coordination et service métier. Chacun de ces éléments logiciels joue un rôle architectural particulier et est implémenté à l'aide de techniques spécifiques de développement.

Les caractéristiques du modèle $\mathcal{MoS}$ sont les suivantes :

- L'interconnexion avec le modèle $\mathcal{MiS}$. Ceci permet de passer de la perspective intentionnelle à la perspective opérationnelle des services.

- L'indépendance à une plate-forme d'implémentation spécifique.

2. PERSPECTIVES

Le travail présenté dans cette thèse peut être poursuivi dans plusieurs directions :

- Développement d'un environnement logiciel pour assister le concepteur lors de la modélisation des services et la génération de la solution logicielle :

Les cartes d'intentions/stratégies sont centrales dans la méthodologie proposée. Un environnement logiciel autour du système de représentation MAP est indispensable pour apporter aux concepteurs d'applications le guidage nécessaire dans la construction, la validation et le raisonnement sur les cartes. Cet environnement est en cours de construction au Centre de Recherche en Informatique (CRI) à l'université Paris 1. Il propose des fonctionnalités concernant la modélisation et la validation des cartes. Il peut être étendu par des fonctionnalités supplémentaires pour la modélisation du modèle $\mathcal{MiS}$, la génération des agents contrôleurs et exécutants des services, ainsi que pour l'automatisation du processus de génération des services logiciels du modèle $\mathcal{MoS}$.

- 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 $\mathcal{MiS}$. 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

[Akram03] A. Akram, O-F. Rana. *Organizing Service-Oriented Peer Collaborations*. In Proc. of the 1st International Conference on Service-Oriented Computing, Trento, Italy, December 2003.

[Alonso04] G. Alonso, F. Casati, H. Kuno et V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 1st edition, 2004.

[Alpdemir03] M-N. Alpdemir, A. Mukherjee, N-W. Paton, P. Watson, A-A-A. Fernandes, A. Gounaris, J. Smith. *Service-Based Querying on the Grid*. In Proc. of the 1st International Conference on Service-Oriented Computing, Trento, Italy, December 2003.

[Andrews03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic et S. Weerawarana. *Business Process Execution Language for Web Services (BPEL4WS) version 1.1*, May 2003.

[Andrews03] T. Andrews, F. Curbera, H. Dholakia. Microsoft, IBM, and SAP. *Business process execution language for web services (BPEL4WS) version 1.1*. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 2003.

[Anton94] A. Anton, W-M. McCracken, C. Potts. *Goal decomposition and scenario analysis in Business Process Reengineering*, 6th International Conference on Advanced Information Systems Engineering (CAiSE '94), Utrecht, Pays-Bas, juin 1994.

[Anton96a] A. Anton. *Goal-based requirements analysis*, 2nd International Conference on Requirements Engineering (ICRE '96), Colorado Springs, Colorado, USA, août 1996.

[Anton96b] A.I. Anton, J. Dempster, D. F.siege. *Goal based requirements analysis*. Proceedings of the 2nd International Conference on Requirements Engineering ICRE'96, pp. 136-144, 1996.

[Arsanjani04] A. Azsanjani. *Service-oriented modelling and architecture*. http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/, November2004.

[Baida04] 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.

[Banerjee87] J. Banerjee, W. Kim, H-J. Kim, H-F Korth. *Semantics and Implementation of Schema Evolution in Object Oriented Databases.* In Proc. of the ACM-SIGMOD Annual Conference, pages 311--322, San Francisco, CA, May 1987.

[Barry03] Douglas K. Barry. *Web Services and Service-Oriented Architectures : The Savvy Manager's Guide*. Morgan Kaufmann, 2003.

[Batini01] C. Batini. Enabling Italian E-Government Through a Cooperative Architecture. IEEE Computer, vol 6, no. 3, 2001.

[Bellwood02] T. Bellwood, L. Clément, and C. von Riegen. *Universal description, discovery and integration.* Technical report, OASIS UDDI Specification Technical Committee, mar 2002. http ://www.oasis-open.org/cover/uddi.html.

[Ben Achour99] C. Ben Achour. *Extraction des Besoins par Analyse des Scénarios Textuels*, Thèse du doctorat à l'Université de Paris6. Janvier 1999.

[Benatallah02] 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.

[Benatallah03] B. Benatallah, Q-Z. Sheng et M. Dumas. *The Self-Serv environment for web services composition.* IEEE Internet Computing, 7(1): 40-48, 2003.

[Berardi03] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella. *Automatic Composition of e-services that export their Behavior*. Proceedings of the 1st International Conference on Service Oriented Computing (ICSOC'03), Trento, Italy, 2003.

[Berardi04] D. Berardi, G. De Giacomo, D. Calvanese. *Synthesis of Underspecified Composite e-Services based on Automated Reasoning*. Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04). New York, USA, 2004.

[Berardi05] D. Berardi. *Automatic Service Composition : Models, Techniques and Tools*. Thèse de doctorat, Université La Sapienza, 2005.

[Bezivin02] J. Bezivin, S. Gerard. *A preliminary identification of MDA components*. In Generative Techniques in the context of Model Driven Architecture, Nov 2002

[Bezivin04] J. Bezivin, M. Blay, M. Bouzhegoub, J. Estubier, JM Favre, S. Gérard, J. Jézéquel, Rapport synthétique de l'AS CNRS sur le MDA (Model Driven Architecture), rapport interne du CNRS, www.adele.imag.fr/mda/as, Septembre 2004.

[Booch98] G. Booch, J. Rumbaugh, I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[Booth03] D. Booth, H. Haas, F. McCabe, and E. Newcomer. *Web services architecture*, aout 2003. http ://www.w3.org/TR/2003/WD-ws-arch-20030808/.

[Brown96] A-W. Brown. *Component-based software engineering*: selected papers from the Software Engineering Institute. Los Alamitos, CA : IEEE Computer Society Press, 1996.

[Caroll95] J. M. Caroll. *The Scenario Perspective on System Development*, in Scenario-Based Design Envisioning Work and Technology in System Development, Ed J.M. Carroll, 1995.

[Casati01] F. Casati and M-C. Shan, *Dynamic and Adaptive Composition of* e-*Services*, Information Systems **6**, no. 3, 143 – 163, 2001.

[Charfi04] A. Charfi, M. Mezini. *Hybrid web service composition: business processes meet business rules.* International Conference on Service Oriented Computing (ICSOC'04). New York, USA, 2004.

[Chauvet02] J-M. Chauvet. *Services WEB avec SOAP, WSDL, UDDI, ebXML*. Eyrolles, Paris, 2002.

[Chen76] P.P.S. Chen. *TheEntity-Relationship Model : Toward a Unified View of Data.* ACM Transactions on Database Systems, Vol.1, N°1, March 1976.

[Chevrin06] 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.

[Christensen01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web services description language (WSDL)* 1.1. Technical report, World Wide Web Consortium, mar 2001. http ://www.w3.org/TR/wsdl.

[Clements02] P. Clements, L. Northrop. *Software Product Lines – Practices and Patterns*, Addition-Wesley, 2002.

[Cockburn00] A. Cockburn. *Writing Effective Use Cases (The Crystal Collection for Software Professionals)*, Addison-Wesley Longman, Incorporated (2000).

[Cockburn95] A. Cockburn, *Structuring use cases with goals*. Technical report. Human and Technology, 7691 Dell Rd, Salt Lake City, UT 84121, HaT.TR.95.1, http: //members.aol.com/acocburn/papers/usecases.htm, 1995.

[Colombo05] C. Colombo, E. Nitto, M. Penta, D. Distante, M. Zuccala. *Speaking a Common Language: A Concpetual Model for Describing Service-Oriented Systems*. Third International Conference on Service Oriented Computing (ICSOC'05), pp. 48-60, 2005.

[Comerio04] M. Comerio, F. De Paoli, S. Grega, C. Batini, C. Di Francesco, A. Di Pasquale. *A service Re-Design Methodology for multi-channel adaptation*. Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04). New York, USA, 2004.

[Czarnecki00] K. Czarnecki, U.W. Eisenecker. *Generative Programming – Methods, Tools and Applications*. Addison-Wesley, 2000.

[Dagstuhl05] Dagstuhl Seminar Proceedings 05462 Service Oriented Computing (SOC) Group report by Barbara Pernici, Politecnico di Milano, November 2005, http://drops.dagstuhl.de/opus/volltexte/2006/525.

[Dardenne91] A. Dardenne, S. Fickas, A. van Lamsweerde. *Goal-directed concept acquisition in requirements elicitation*, Proc. 6th IEEE Workshop System Specification and Design0, Como, Italy, 1991, 14-21.

[Dardenne93] A. Dardenne, A. van Lamsweerde, S. Fickas. *Goal-directed requirements acquisition*, Science of Computer Programming, 20 (1-2), avril 1993.

[Demazeau91] Y. Demazeau, J-P. Müller. *Decentralised AI 2*, Amsterdam, Elsevier North-Holland, 1991.

[Dijkman03] R-M. Dijkman. *A Basic Design Model for Service-Oriented Design*. ArCo/WP1/T1/D2/V1.00, 2003.

[Dik89] S.C. Dik. *The theory of functional grammar*. Foris Publications, Dodrecht, Pays-Bas, 1989.

[Dittrich94] K-R. Dittrich. 1994, *Object-Oriented Data Model Concepts*, Advances in Object-Oriented Database Systems, NATO ASI Series, Series F: Computer and System Science, Vol. 130, 29-45, Springer Verlag, New York.

[Do03] T. Do, M. Kolp, A. Pirotte. *Social Patterns for designing Multi-Agent Systems*, 15th International Conference on Software Engineering and Knowledge Engineering (SEKE'03), San Francisco, USA, 2003.

[Dowson88] M. Dowson. *Iteration in the Software Process*, Proc 9th Int. Conf. on Software Engineering, 1988.

[Drogoul92] A. Drogoul, J. Ferber. *Multi-agent simulation as a tool for modelling societies: Application to social differentiation in ant colonies*. In Workshop On Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '92, Viterbo, 1992.

[Etien06] A. Etien. *Ingénierie de l'alignement : Concepts, Modèles et Processus, La méthode ACEM pour l'alignement d'un système d'information aux processus d'entreprise*, Thèse de doctorat en informatique, Université Paris1, 2006.

[Fauvet02] M-C. Fauvet, M. Dumas et B. Benatallah. *Collecting and querying distributed traces of composite service executions*. Confederates International conferences CoopIS, DOA and ODBASE, pages 373-390, Irvine, California, USA, 2002. Springer.

[Fillmore68] C.J. Fillmore. *The case for case*, in Universals in linguistic theory, Holt, Rinehart and Winston, Inc, E.Bach/R.T.Harms (eds), 1968.

[Franckson91] M. Franckson, C. Peugeot. *Specification of the object and process modelling language*. ESF Report n° D122-OPML-1.0, 1991.

[Gamma95] H. Gamaa, M-E. Shin. *Multiple-view meta-modeling of software product lines*, In 8th International Conference on Engineering of Complex Computer Systems, Decembre 2002.

[Gustavo04] G. Alonso, F. Casati, H. Kuno et V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 1rst édition, 2004.

[Harel00] D. Harel, D. Kozen, J. Tiuryn. *Dynamic Logic*. The MIT Press, 2000.

[Heather01] K. Heather. *Web services conceptual architecture* (wsca 1.0), may 2001. http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

[Henkel04] M. Henkel, J. Zdravkovic, P. Johannesson. *Service-based Processes – Design for Business and Technology*. International Conference on Service Oriented Computing (ICSOC'04). New York, USA, 2004.

[Holbrook90] C. H. Holbrook. *A scenario - based methodology for conducting requirements elicitation*. ACM SIGSOFT, Software Engineering Notes Vol. 15, N° 1, pp. 95-104. January 1990.

[Hull03] R. Hull, M. Benedikt, V. Christophides, J. Su. *E-Services: A Look Behind the Curtain*, Proceedings of the 22nd ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS 2003), ACM, 2003, pp. 1–14.

[Hull87] R. Hull, R. King. *Semantic Database Modelling : Survey*, Application and Research Issues. ACM Computer Survey, Vol.19, N°3, 1987.

[ICSOC04] Second International Conference on Service Oriented Computing, New York City, USA, November 15-19, 2004.

[ICSOC05] Third International Conference on Service Oriented Computing, Amsterdam, The Netherlands, December 12-15, 2005.

[Ingolf04] Ingolf, H. Kruger, R. Mathew. *Systemtic Development and Exploration on Service-Oriented Software Architectures*. Wicsa, p.177, fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04), 2004.

[Jackson95] M. Jackson. *Software Requirements and Specifications – A Lexicon of Practice, Principles and Pejudices*. ACM Press, Addison-Wesley, 1995.

[Kim04] 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.

[Kolp01] M. Kolp, J. Castro, J. Mylopoulous. *A Social Organization Perspective on Software Architectures First international*, Workshop From Software Requirements to Architectures (STRAW'01) at ICSE, Toronto, Canada, 2001.

[Lausen97] S. Lausen and G. Vossen, *Models and languages of Object/Oriented Databases*, Addison-Wesley, Harlow, England, 1997.

[Lejeune01] Y. Lejeune, A. Fermé. *Les Web Services*. Micro Application. 2001.

[Lopes05] D-C-P. Lopes. *Étude et applications de l'approche MDA pour des plates-formes de Services Web*, Thèse du doctorat à l'Université de Nantes. Juillet 2005.

[Lubars93] M. Lubars, C. Potts, Charles Richter. *A Review of the State of Practice in Requirements Modeling*. Proceedings of the IEEE International Symposium on Requirements Engineering, RE'93, San Diego, USA, pp.2-14, January 1993.

[MacNaugthon60] R. MacNaughton, Yamada. *Regular expressions and state graphs for automata.* IEEE transactions on electronic computers, EC-9, P 39-47, 1960.

[Mecella02a] M. Mecella, B. Pernici. *Building Flexible and Cooperative Applications Based on e-Services*. Dipartimento di Informatica e Sistemistica, Universita' di Roma "La Sapienza", Technical Report 21-2002, Roma, Italy, 2002.

[Mecella02b] M. Mecella, F. Parisi-Precisse. *Modeling E -service Orchestration through Petri Nets*. Proceedings of the Third International Workshop on Technologies for E-ServicesPages: 38 – 47, 2002.

[Melliti04] T. Melliti. *Interopérabilité des services Web complexes. Application aux systèmes multi-agents.* Thèse de doctorat, Université Paris IX Dauphine, 2004.

[Meyers02] 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.

[Mylopoulos00] J. Mylopoulos, J. Castro. *TROPOS : A framework for requirements driven sofware development*. Information systems engineering: state of the art and research themes, Lecture Notes in Computer Science, Springer-Verlag, 2000.

[Nilsson71] N-J. Nilsson. *Problem Solving Methods in Artificial Intelligence*. McGraw Hill, 1971.

[O'Hare96] G-M. O'Hare, N-R. Jennings. *Foundations of Distributed Artificial Intelligence*, John Wiley & Sons, 1996.

[ObjectMatter05] Object Matte, www.objectmatter.com

[Ommering02] R.-V. Ommering. *Building product populations with software components*, proceedings of the 24th International Conference on Software Engineering, Orlando, Florida, 2002.

[Orriëns03] B. Orriëns, J. Yang, M-P. Papazoglou. *Model Driven Service Composition*, in Proc. Of the 1st International Conference on Service Oriented Computing (ICSOC'03), Trento, Italy, 2003.

[Pallos01] M. Pallos. *Service Oriented Architecture: A Primer*, EAI Journal, December 2001.

[Papazoglou02] M-P. Papazoglou, P. Grefen. *Service-Oriented Computing "The S.O.C Manifesto"*, 2002.

[Papazoglou03] M-P. Papazoglou and D. Georgakopoulos. *Service Oriented Computing (special issue)*, Communication of the ACM 46 (2003), no. 10, 24–28.

[Papazoglou05] M-P. Papazoglou. *Extended the Service Oriented Architecture*. Business Integration Journal, 2005.

[Papazoglou06a] M-P. Papazoglou, W-J. van den Heuvel. *Service-Oriented Design and Development Methodology*. Int. J. of Web Engineering and Technology (IJWET), 2006.

[Papazoglou06b] M-P. Papazoglou, W-J. van den Heuvel. *Business Process Development Lifecycle Methodology*. To appear in communications of ACM, 2006.

[Papazogou06a] M-P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann. *Service-Oriented Computing, Research Roadmap*, Mars 2006.

[Patrascoiu04] O. Patrascoiu. *Mapping EDOC to Web Services using YATL*. 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004), pages 286–297, September 2004.

[Peltz03] C. Peltz. *Web services orchestration a review of emerging technologies, tools, and standards*, Hewlett Packard, Co. January 2003.

[Penserini05] L. Penserini, J. Mylopoulos. *Design Matters for Semantic Web Services*. ITC-IRST, Technical report: T05-04-03, April 2005.

[Penserini06a] L. Penserini, A. Perini, A. Susi, J. Mylopoulos. *From Stakeholder needs to service requirements specifications*. Technical Report, ITC-IRST, Automated Reasoning Systems, 2006.

[Penserini06b] L. Penserini, A. Perini, A. Susi, J. Mylopoulos. *From Stakeholder intentions to software agent implementations*. CaiSE'06, LNCS 4001, pp. 465-479, 2006.

[Penserini06c] L. Penserini, A. Perini, A. Susi, J. Mylopoulos. *From Stakeholder Needs to Service Designs*. Requirements Engineering Journal 35, 2006.

[Peppers97] D. Peppers, M. Rogers. *The one to one future: Building relationships one customer at a time*. New York: Doubleday. 1997.

[Perini05] A. Perini, A. Susi, J. Mylopoulos. *Tropos Design Process for Web Services*, 1st International Workshop on Service-Oriented Computing: Consequences for Engineering Requirements, Paris, (2005).

[Pfister96] C. Pfister C. Szyperski. *Why objects are not enough*. In *Proceedings, International Component Users Conference*, Munich, Germany, 1996. SIGS.

[Piccinelli03] G. Piccinelli, W. Emmerich, S-L. Williams, M. Stearns. *A Model-Driven Architecture for Electronic Service Management Systems*. ICSOC 2003, LNCS 2910, pp. 241-255, 2003.

[Pistore04] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso, *Planning and Monitoring Web Service Composition*, Proceedings of the 2nd ICAPS International Workshop on Planning and Scheduling for Web and Grid Services, 2004.

[Potts94] C. Potts, K. Takahashi, A.I. Anton. *Inquiry-based requirements analysis*. In IEEE Software 11(2), pp. 21-32, 1994.

[Prat97] N. Prat. *Goal formalisation and classification for requirements engineering*, Proceedings of the Third International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'97, Barcelona, June 1997.

[Prat99] N. Prat. *Réutilisation de la trace par apprentissage dans un environnement pour l'ingénierie des processus*, Thèse de doctorat en informatique, Université Paris1, 1999.

[Quartel04a] 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.

[Quartel04b] D. Quartel. *ISDL home*. Retrieved January 29, 2004 from University of Twente, Center for Telematics and Information Technology Website: http://isdl.ctit.utwente.nl/, 2004.

[Ralyté01] J. Ralyté. *Ingénierie des méthodes à base de composants*, Thèse de doctorat en informatique, Université Paris 1, 2001.

[RDW95] Relational Database Writings 1991-1994, Addison-Wesley, 1995.

[Rolland00] C. Rolland, N. Prakash. *Bridging the gap between Organizational needs and ERP functionality*. Requirements Engineering Journal, 2000.

[Rolland01] C. Rolland, N. Prakash. *Matching ERP System Functionality to Customer Requirements*, Proceedings of the 5th International Symposium on Requirements Engineering (RE'01), Toronto, Canada, pp. 66-75, 2001.

[Rolland04] C. Rolland, C. Salinesi, A. Etien. *Eliciting Gaps in Requirements Change*, Requirements Engineering Journal (REJ), 9:1, pp. 1 - 15, 2004.

[Rolland05] C. Rolland, R-S. Kaabi. *Designing Service Based Cooperative Systems*, Encyclopedia of E-Technologies and Applications (EET&A), IDEA Group (pub), 2005

[Rolland92] C. Rolland, C. Cauvet. *Trends and Perspectives in Conceptual Modelling*. In Conceptual Modelling, Database and CASE : an Integrated View of Information Systems Development, Wiley, 1992.

[Rolland98a] 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.

[Rolland98b] C. Rolland, C. Souveyet, C. Ben Achour. *Guiding Goal Modelling using Scenarios*, IEEE Transactions on Software Engineering, Special Issue on Scenario Management, Vol. 24, No. 12, 1055- 1071, Dec. 1998**.**

[Rolland99] C. Rolland, N. Prakash, A. Benjamen. *A multi-Model View of Process Modelling*, Requir. Eng. Vol. 4 N°4, pp 169-187, 1999.

[Rumbaugh98] J. Rumbaugh, I. Jacobson, G. Booch. *The Unified Modeling Language Reference Manuel*. Addison-Wesley, 1998.

[Rust03] R. Rust, P-K. Kannan. *E-service: a New Paradigm for Business in the Electronic Environment*. Communication of the ACM, June 2003, vol. 46, N°6, pp 37-42.

[Salinesi03] C. Salinesi, C. Rolland. *Fitting Business Models to Software Functionality : Exploring the Fitness Relationship*. Proceedings of the 15th Conference on Advanced Information Systems Engineering, (CAISE'03), Springer Verlag (pub), 2003.

[Sawyer05] P. Sawyer, J. Hutchison, J. Walkerdine, I. Sommerville. *Faceted Service Specification*. Proc. Workshop on *Service-Oriented Computing Requirements (SOCCER)*. Paris, August 2005.

[Schaffner06] 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.

[Schuler03] C. Schuler, R. Weber, H. Schuldt, H-J. Schek. *Peer-to-Peer Process Execution with OSIRIS*. In Proc. of the 1st International Conference on Service-Oriented Computing, Trento, Italy, December 2003.

[Shegalov01] G. Shegalov, M. Gillmann, and G. Weikum, *XML-enabled Workflow Management for* e-*Services across Heterogeneous Platforms*, Very Large Data Base Journal 10, no. 1, 91–103, 2001.

[Sirin04] E. Sirin, B. Parsia, D. Hendler, J. Nau. *HTN Planning for Web Service Composition Using SHOP2*. Journal of Web Semantics 1 (2004) 377-396.

[Smith77] J.M. Smith. *Database Abstractions : Aggregation and Generalization.* ACM Transactions on Database Systems, Vol.2, pp 105-133, 1977.

[SOE] Service Oriented Enterprise, http://www.serviceoriented.org/web_service_orchestration.html.

[Soffer05] P. Soffer, C. Rolland. *Combining Intention-Oriented and State-Based Process Modeling*. Proceedings of ER2005, Klagenfurt, Austria, 2005.

[Solanki04] M. Solanki, A. Cau, H. Zedan. *Augmenting Semantic web service descriptions with compositional specification.* WWW'04, 13th International Conference on World Wide Web, New York, USA, 2004.

[Svahnberg01] Svahnberg et al. *On the notion of variability in Software Product Lines*, Proceedings of the Working IEEE/IFIP Conference on Software architecture, 2001

[Szyperski97] C. Szyperski. *Component software: beyond object-oriented programming*. New York : ACM Press Harlow, England Reading, Mass : Addison- Wesley, 1997.

[Tansey0x] B. Tansey, E. Stroulia. *Towards an Economics Model for Software Development in Support of B2C Services.*

[Tapaloglu04] N-Y. Topaloglu, R. Capilla. *Modeling the Variability of Web Services from a Pattern Point of View*. European Conference on Web Services (ECOWS2004), LNCS Springer-Verlag, 2004, pp. 128-138.

[Tawbi01] M. Tawbi. *Crews-L'Ecritoire : Un guidage outillé du processus d'ingénierie des besoins.* Thèse de Doctorat à l'Univeristé Paris 1, octobre 2001.

[Tomphson01] J. Thomphson, M-P. Heimdalh. *Extending the Product Family Approach to support n-Dimensional and Hierarchical Product Lines*, Proceedings of the 5th IEEE Symposium on Requirements Engineering, Toronto, Canada, 2001.

[Tut02] M-T. Tut, D. Edmond. *The use of patterns in service composition*. Revised papers International Workshop on Web Services, E-Business and Semantic Web, 28-40, 2002.

[Valasco05] C. Lopez-Valesko, M. Villanova-Oliver, J. Gensel, H. Martin. *Adaptabilité à l'utilisateur dans le contexte des services web*. Atelier : Méta-données et adaptabilité pour les systèmes d'information sur le web. 2005.

[van Lamsweerde00] A. van Lamsweerde. *Requirements Engineering in the year 2000 : A research perspective*. 22nd International Conference on Software Engineering, Limerick, Ireland, 2000.

[van Lamsweerds95] A. Van Lamsweerde, R. Dairmont, P. Massonet. *Goal Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt*, in Proc. Of RE'95 – 2nd Int. Symp. On Requirements Engineering, York, IEEE, 1995, pp 194 –204.

[vanGurp01] J. van Gurp, J. Bosch, M. Svahnberg. *On the notion of variability in Software Product Lines*, Proceedings of the working IEEE/IFIP Conference on Software architecture, 2001.

[W3C03] W3C. *Web Services Description Language (WSDL) 2.0 Part1: Core Language*, W3C Working Draft, Novembre 2003.

[W3C04] W3C. *Web Services Architecture (WSA),* February 2004. Disponible sur http://www.w3c.org/TR/2004/NOTE-ws-arch-20040211/.

[W3C04] W3C. *Web Services Glossary*. Disponible sur http://www.w3.org/TR/ws-gloss, 2004.

[Weib99] G. Weib. *Agent Orientation in Software Engineering*, The Knowledge Engineering Review, Vol.16(4), 2001.

[Wodtke97] D. Wodtke and G. Weikum, *A Formal Foundation for Distributed Work-flow Execution Based on State Charts*, Proceedings of the 6th International Conference on Database Theory (ICDT '97), 1997.

[Woodfield97] S-N. Woodfield. *The Impedance Mismatch Between Conceptual Models and Implementation Environments*, ER'97 Workshop on Behavioral Models and Design Transformations: Issues and Opportunities in Conceptual Modeling, 6 - 7 November 1997, UCLA, Los Angeles, California.

[Woolrige99] M. Woolridge. *Intelligent agent*. MultiAgent systems: Amodern Approche to Distributed Artificial Intelligence, pages 27–78, 1999.

[WSCI02] BEA, SAP, Sun, *Web Service Choreography Interface (WSCI) 1.0*: http://wwws.sun.com/software/xml/developers/wsci/wsci-spec-10.pdf, 2002.

[WSCL01] H. Kuno, M. Lemon, A. Karp, D. Beringer. *Conversations + Interfaces = Business Logic*, in Proc of the 2nd VLDB International Workshop on Technologies for e-Services (VLDB-TES 2001), Rome, Italy, 2001.

[WSDL01] *Web Services Description Language (WSDL*) 1.1, W3C Note 15 March 2001 http://www.w3.org/TR/wsdl

[Yang02] J. Yang, M-P. Papazoglou. *Web Component: A Substrate for Web Service Reuse and Composition*. CAISE 2002, LNCS 2348, pp. 21-36, 2002.

[Yang03] J. Yang, M-P. Papazoglou. *Service Components for Managing the Life-Cycle of Service Compositions*. Information Systems Journal, 2003.

[Yang03b] J. Yang, M-P. Papazoglou, B. Orriëns, W-J. van Heuvel. *A rule based approach to service composition life-cycle*. Proceedings of the 4th International Conference on Web Information Systems Engineering (Wise'03), 2003.

[Yu94] E. Yu, J. Mylopoulos. *Using goals, rules and methods to support reasoning in Business Process Reengineering*, 27th Hawaii International Conference on System Sciences, Maui, Hawaii, 1994.

[Yu97] E. Yu. *Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering*, Proceedings of the 3rd IEEE Int. Symp. On Requiorements Engineering (RE'97) Jan. 6-8, 1997, Washington D.C., USA. Pp. 226-235, (1997).

[Zimmermann04] O. Zimmermann, P. Krogdahl, C. Gee. *Elements of Service-Oriented Analysis and Design*. http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/, june 2004.

Annexe A

Les méta-modèles de WSDL et BPEL

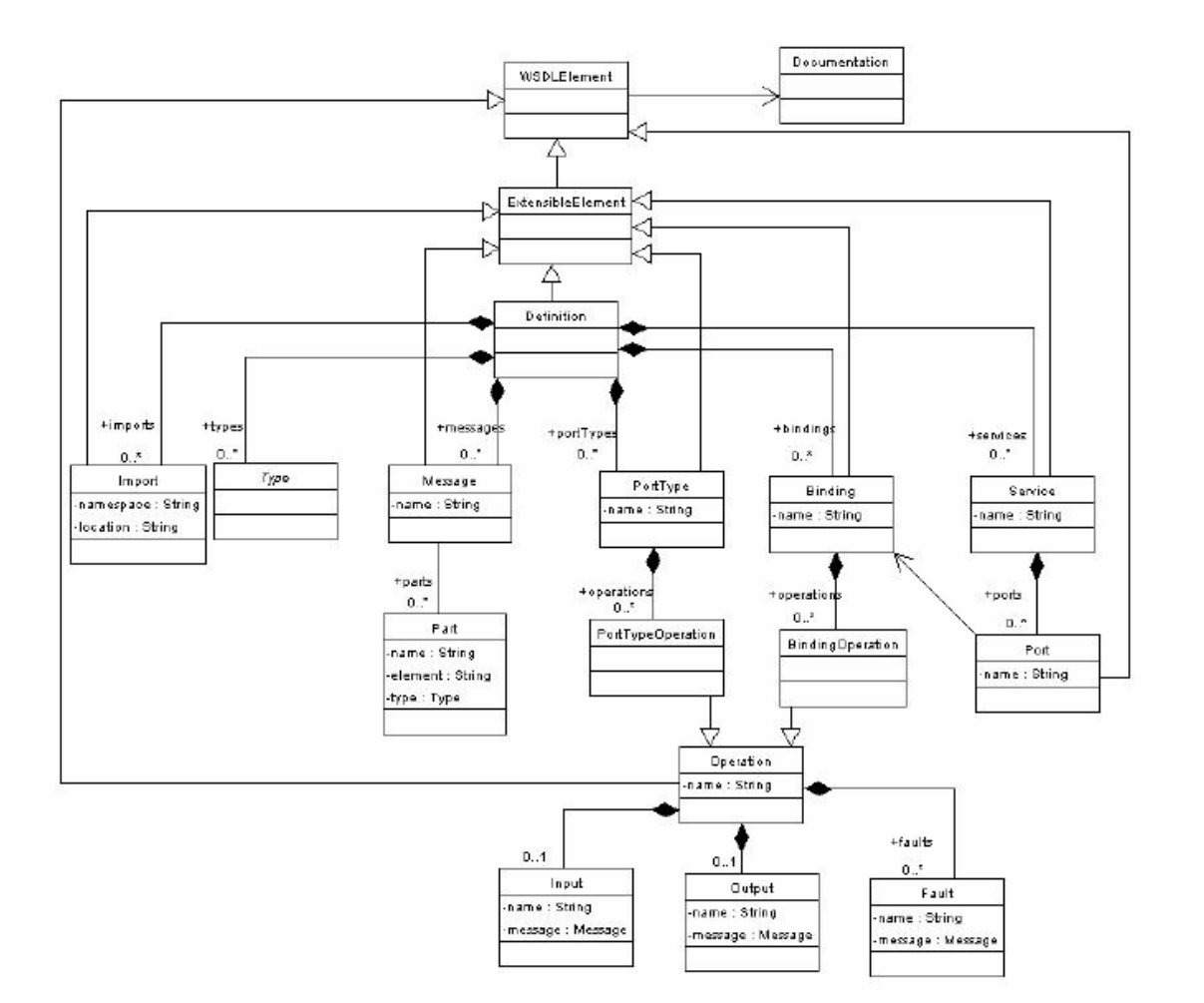


Figure 104. Un méta-modèle de WSDL proposé en [Patrascoiu04]

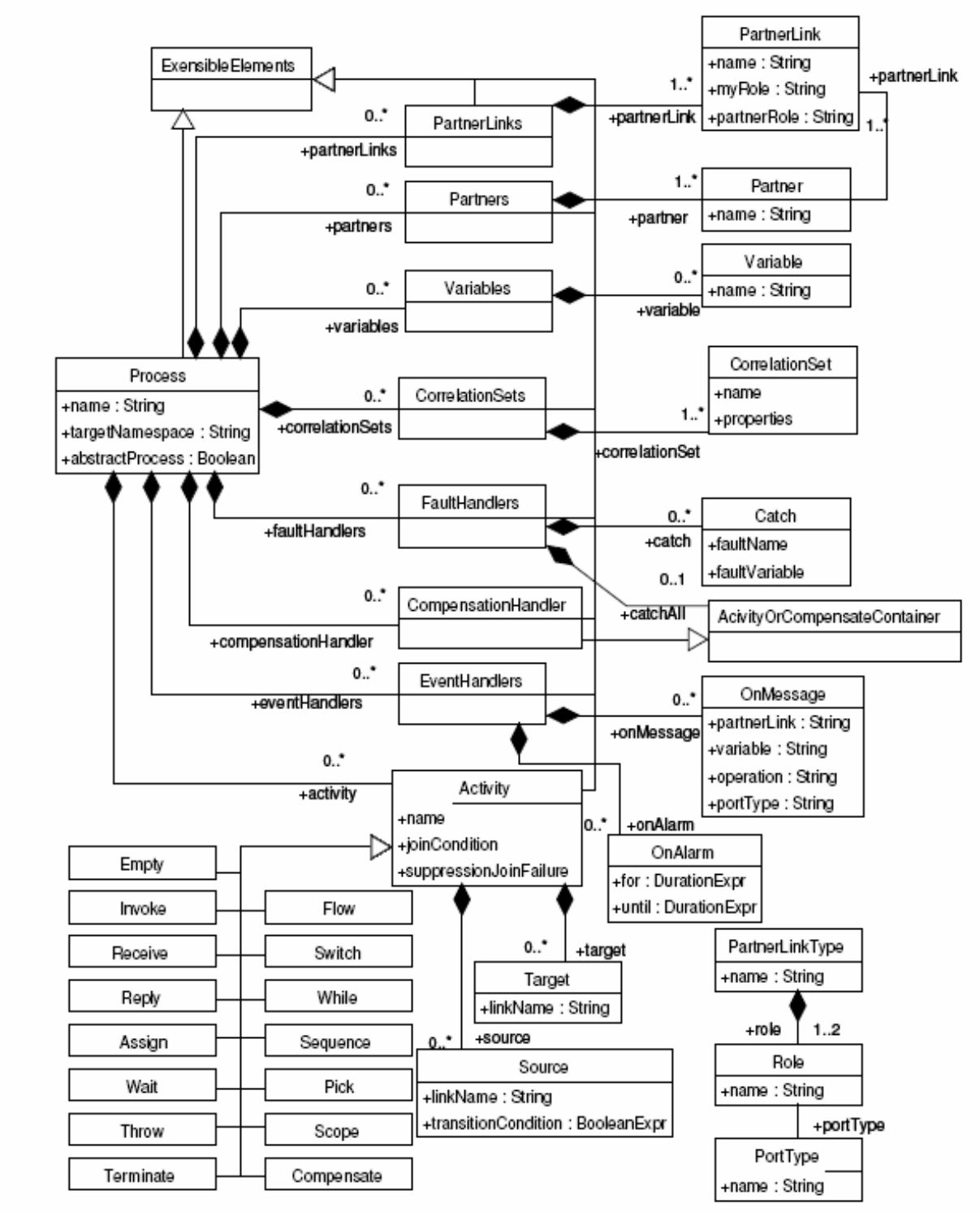


Figure 105. Un Méta-modèle de BPEL4WS (fragment)