Correspondances sémantiques entre des modèles de services, de composants et d'agents

Nour Alhouda Aboud
LIUPPA / Université de Pau
B.P. 1155
64013 PAU CEDEX, France
Nour-alhouda.Aboud@univ-pau.fr

Eric Cariou
LIUPPA / Université de Pau
B.P. 1155
64013 PAU CEDEX, France
Eric.Cariou@univ-pau.fr

Eric Gouardères
LIUPPA / Université de Pau
B.P. 1155
64013 PAU CEDEX, France
Eric.Gouarderes@univ-pau.fr

Philippe Aniorté
LIUPPA / Université de Pau
2 allée du parc Montaury
64460 ANGLET, France
aniorte@iutbayonne.univ-pau.fr

ABSTRACT

Pour le développement d'applications distribuées, les approches orientées services, composants ou agents offrent chacune des intérêts et des caractéristiques propres qu'il serait intéressant de pouvoir utiliser conjointement. Notre but est d'intégrer ces trois approches en se focalisant sur les concepts de service et d'interaction, points clé notamment de la coopération entre des agents et des composants : le service représente l'abstraction métier d'un composant ou d'un agent et permet de les faire interagir. Précédemment, nous avions présenté nos modèles de services, d'agents et de composants dans ce but. Dans cet article, nous établissons les correspondances sémantiques entre les concepts de chacun de ces domaines afin de pouvoir passer d'un modèle à un autre.

Keywords

Services, composants, agents, correspondances sémantiques

1. INTRODUCTION

Les approches orientées services, composants ou agents offrent chacune des intérêts et des caractéristiques propres dans le développement d'applications distribuées. Les services offrent une abstraction et une interopérabilité à large échelle. Abstraction dans le sens où un service spécifie un élément fonctionnel sans préciser comment cet élément est implémenté (par des composants ou des agents par exemple). Les composants sont une approche robuste basée sur la composition et la réutilisation d'éléments clairement définis par leurs interfaces. Les agents[1] sont eux des éléments présentant un comportement dynamique dirigé par un but et des interactions de haut niveau avec les autres agents formant l'application. La figure 1 représente les aspects principaux et complémentaires de ces approches. Malheureusement, ces avantages propres ne sont pas partagés par toutes les approches. Par exemple, [6, 15, 17] montrent le manque de réutilisabilité ou de composition de la part des agents et [4] pointe le manque de capacité de raisonnement et de dynamicité des composants.

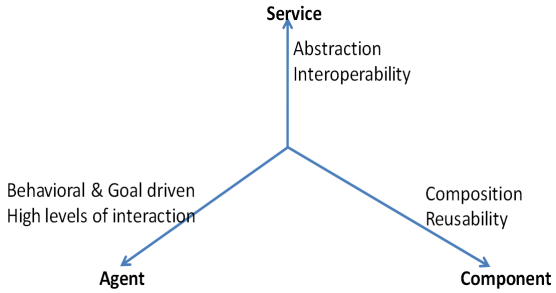

Figure 1: Relations services / agents / composants

Il serait pourtant intéressant de pouvoir disposer à la fois dans une application des intérêts de chaque approche. Le but de notre travail est donc d'intégrer ces trois approches. Nous nous focalisons sur les concepts de service et d'interaction, points clé notamment de la coopération entre des agents et des composants : le service représente l'abstraction métier d'un composant ou d'un agent et sert d'interopérabilité entre les deux vu qu'un service peut être indifféremment implémenté par un agent ou un composant. Précédemment, nous avions présenté nos modèles de services, d'agents et de composants dans ce but [3]. Dans cet article, nous établissons les

[1]Dans cet article, quand nous parlerons des agents, cela est à comprendre systématiquement dans le cadre des systèmes multi-agents organisationnels (OMAS).

correspondances sémantiques entre les concepts de chacun de ces domaines. Dans une démarche d'Ingénierie Dirigée par les Modèles (IDM), ces modèles sont des DSMLs (*Domain Specific Modeling Languages*) et ces correspondances sémantiques seront implémentées par des transformations de modèles pour passer d'un DSML à un autre.

Le reste de cet article est organisé comme suit. La section 2 rappelle notre processus de conception et ses 3 modèles de services, de composants et d'agents. La section 3 spécifie à titre d'exemple une même application via chacun de ces modèles. La section 4 définit les correspondances sémantiques entre nos 3 modèles. Enfin, nous présentons les travaux connexes avant de conclure.

2. RAPPEL DES MODÈLES DE SERVICES, COMPOSANTS ET AGENTS

Afin de bien situer et présenter la contribution de cet article, nous rappelons d'abord dans cette section des éléments que nous avions déjà présentés dans [2, 3].

2.1 Processus de conception

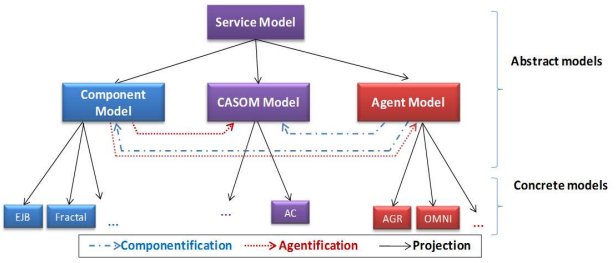

Figure 2: Processus d'intégration des approches services, composants et agents

Notre proposition générale d'intégration des approches services, composants et agents se base sur un processus de conception décrit par la figure 2. Le processus contient 4 modèles principaux (c'est-à-dire 4 DSMLs dans une approche IDM). Au premier niveau, abstrait, se trouve le modèle de services permettant de décrire une application uniquement sous forme de services interagissant entre eux. Il s'agit donc de se concentrer sur la spécification purement métier de l'application. À un niveau plus bas, cette même application est spécifiée en précisant quels éléments concrets implémentent ces services. On utilisera alors soit uniquement des composants, soit uniquement des agents, soit un mélange des deux en passant pour cela respectivement par l'utilisation du modèle de composants, d'agents ou CASOM (pour *Component Agent Service Oriented Model*) qui intègre à la fois les concepts d'agents et de composants. Nos modèles de composants et d'agents ainsi que le modèle mixte CASOM sont des modèles généraux, et il est possible de projeter une spécification composants, agents ou mixte vers un modèle technologique concret comme EJB[2] ou Fractal [7] pour les composants, AGR [10] ou OMNI [19] pour les agents ou bien encore AC [14] pour une approche mixte agents/composants.

Le processus peut être utilisé de plusieurs manières : avec une approche descendante comme précisé ci-dessus, où on définira à un niveau abstrait les services d'une application, puis on les projettera sur des composants, des agents ou un mélange des deux, avant de projeter cette nouvelle spécification vers une plateforme technologique concrète. On peut inversement procéder à une approche ascendante de *reverse-enginneering*. Enfin, on pourra se placer dans une approche horizontale où il s'agira de transformer dans une spécification existante des agents en composants et vice-versa via respectivement des actions de *componentification* et d'*agentification*.

[2]http ://www.oracle.com/technetwork/java/javaee/ejb/

En l'état actuel, les 3 modèles de services, composants et agents ont été définis et présentés dans [3]. Dans cet article, nous proposons les règles sémantiques de passage entre les concepts de ces 3 modèles, c'est-à-dire correspondant aux flèches entre ces 3 modèles dans notre processus.

Il existe déjà de très nombreux modèles généraux ou abstraits de composants, agents ou services. Nous avons néanmoins décidé de définir nos propres modèles d'agents et de composants car aucun n'intégrait explicitement de manière satisfaisante à la fois les aspects d'interaction et de service qui sont dans notre approche les clés de l'intégration des approches agents et composants comme expliqué en introduction. De même, nous avons défini notre modèle de services car aucun ne satisfaisait totalement notre besoin d'abstraction, à savoir de ne pas faire apparaître les éléments réalisant les services. [1, 3] décrivent les modèles de composants, agents et services existants que nous avons étudiés pour définir nos 3 modèles.

2.2 Trois modèles

Nos 3 modèles de services, composants et agents sont présentés sur la figure 3. Il s'agit de 3 modèles différents (représentés sous forme de diagrammes de classes UML) mais que nous avons décidé de superposer afin de ne représenter qu'une seule fois les concepts communs à plusieurs modèles. L'intérêt de cette vision des choses, outre le gain de place par rapport à 3 diagrammes distincts, est de montrer que nos trois modèles partagent des concepts communs, comme ceux de service ou d'interaction. Cela permet de voir, d'un point de vue visuel et de manière générale, les liens plus ou moins proches entre ces modèles. Néanmoins, il s'agit bien concrètement de 3 diagrammes différents. Des contraintes OCL (non présentées ici faute de place) supplémentaires spécifient complètement ces diagrammes.

Dans les sous-sections suivantes, nous résumons le contenu de nos 3 modèles (pour plus de détails, voir [3]). Les termes écrits en italique correspondent à des éléments d'un modèle (via une traduction de l'anglais au français).

2.2.1 Modèle de services

Un *service* est formé d'*opérations* via des *points de service*. Un point de service regroupe des opérations en mode *offert* ou *requis*. Un service est donc le regroupement logique d'ensembles d'opérations, certains ensembles étant offerts et d'autres requis (ce qui correspond au même type de définition de services qu'en WSDL par exemple). Un service peut être *composite*, c'est-à-dire structurellement composé d'autres services, sinon, il est *primitif*.

Des services différents interagissent entre eux via des *inter-*

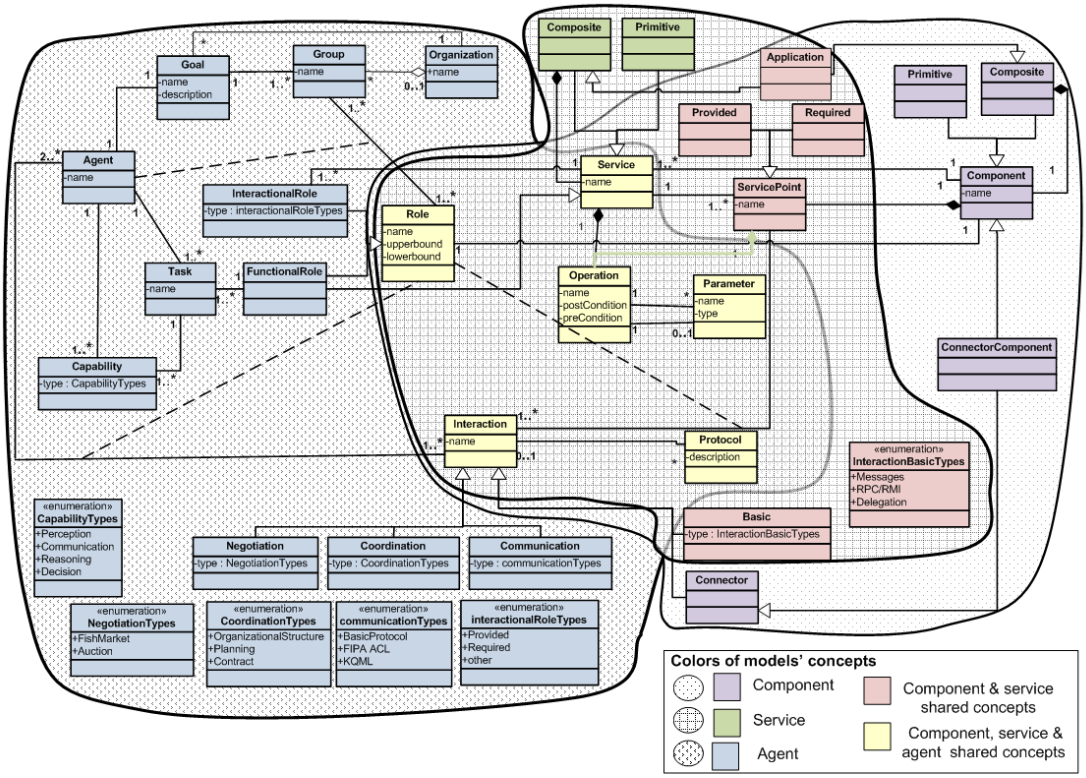

Figure 3: Vue virtuellement unifiée des trois modèles (services, agents, composants)

actions associées à leurs points de service. Un service joue un certain *rôle* dans une interaction. Il existe deux types principaux d'*interaction basique* : de type *appel d'opération* (RPC / RMI) entre un point de service offert et un point de service requis de services différents, ou de type *délégation* entre un point de service d'un composite et un point de service du même type (offert ou requis) d'un de ses services internes. Si une interaction plus complexe est requise entre les points de service de différents services, elle est alors définie par un *protocole*.

Enfin, le concept d'*application* correspond à une application dans sa globalité. Comme une application est formée d'un ensemble de services interagissant entre eux, une application est vue comme un service composite particulier.

2.2.2 Modèle de composants

Un composant est une entité réutilisable avec des points d'interaction bien spécifiés : ces points d'interaction sont des *points de service*. Chaque point de service est associé à un *service*[3], en mode *offert* ou *requis*. Un service est composé d'un ensemble d'*opérations*. Un composant peut être *primitif* ou *composite*, c'est-à-dire structurellement composé d'autres composants.

Des composants différents interagissent entre eux via des *interactions* associées à leurs points de service. Un composant joue un certain *rôle* dans une interaction. Il existe deux types principaux d'*interaction basique* : de type *appel d'opération* (RPC / RMI) entre un point de service offert et un point de service requis de composants différents, ou de type *délégation* entre un point de service d'un composite et un point de service du même type (offert ou requis) d'un de ses composants internes. Si une interaction plus complexe est requise entre les points de service de différents services, elle est implémentée par un *connecteur* qui réalise un *protocole*. Un *composant connecteur* est un composant particulier, dédié à la communication entres composants [9] : il s'agit donc également d'un connecteur et il réalise un protocole.

Enfin, le concept d'*application* correspond à une application dans sa globalité. Comme une application est formée d'un ensemble de composants interagissant entre eux, une application est vue comme un composant composite particulier.

2.2.3 Modèle d'agents

La notion d'agent du modèle s'inscrit ici dans une vision centrée organisation des systèmes multi-agents. Dans le cadre du *service-oriented computing*, qui se focalise sur les notions de services et d'interaction, une organisation est considérée comme un moyen pour construire des systèmes à partir de services collaboratifs [18]. Une *organisation* est associée à un *but* fonctionnel qui peut être décomposé en différents sous-buts. Ces sous-buts sont associés à des *groupes* qui représentent les entités structurelles de l'organisation et définissent les acteurs et leurs rôles au sein de celle-ci. Dans ce

[3]Avec un vocabulaire plus habituel dans le monde des composants, un point de service est un port et un service est une interface. Nous avons utilisé ces termes afin de nommer les concepts communs entre nos 3 modèles de la même façon.

contexte, un *agent* est une entité autonome rationnelle (associée à un but) qui joue des *rôles* au sein de groupes. Un *rôle fonctionnel* permet de spécifier les tâches et le comportement de l'agent par rapport à un *service* ; un service étant un ensemble d'*opérations*. Un *rôle interactionnel* permet de définir les responsabilités que doit assumer un agent lorsqu'il interagit avec d'autres agents. En particulier, il permet d'offrir ou de requérir des services. Pour jouer un rôle, un agent doit avoir certaines *capacités*, notamment en termes de perception, de communication et de raisonnement.

Généralement les agents interagissent entre eux via leurs rôles interactionnels en utilisant des interactions de haut niveau (langage de communication d'agents (ex : FIPA-ACL), négociation, coordination) basées sur des *protocoles*.

3. EXEMPLES DE SPÉCIFICATION D'UNE APPLICATION

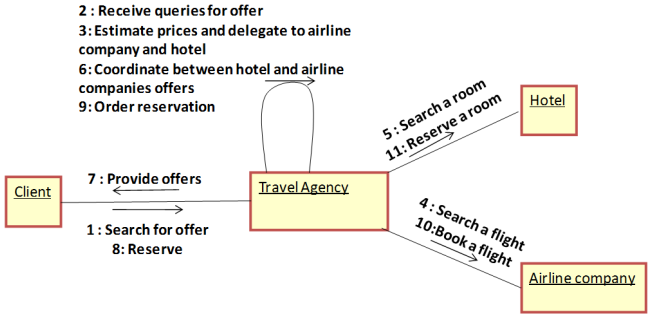

Figure 4: Vision générale de l'exemple d'application

Afin d'illustrer notre approche, nous utilisons un exemple de spécification d'une application de réservation de vacances. Cette application contient quatre types d'éléments principaux :

- Un client qui fait des requêtes pour réserver des vacances (réservation d'un billet d'avion et d'une chambre d'hôtel).
- Une agence de voyage qui reçoit et traite les requêtes des clients.
- Une compagnie aérienne qui répond aux requêtes d'une agence de voyage pour réserver des places sur ses vols.
- Une chaîne d'hôtels qui répond également aux requêtes d'une agence de voyage pour réserver des chambres dans ses hôtels.

La figure 4 représente ces 4 éléments et une des principales interactions globales consistant pour un client à effectuer une requête pour s'informer sur des vols et des chambres d'hôtel puis valider une réservation. Nous allons spécifier cette application en utilisant chacun des trois modèles présentés ci-dessus : par des services (figure 5), par des composants (figure 6) et par des agents (figure 7). Ces trois spécifications serviront ensuite d'illustration aux règles de correspondances sémantiques définies dans la section suivante. Nous utilisons pour chaque modèle une syntaxe graphique propre que nous avons définie (voir la légende sur chaque figure).

Certaines informations ne sont pas présentées pour ne pas surcharger les figures. Il s'agit notamment des noms des rôles pour les interactions, les ensembles d'opérations associés aux points de service ou équivalents (pour des exemples d'associations d'opérations aux points de service, voir les figures 8 et 11) ou bien encore les cardinalités sur les interactions[4].

Dans les trois spécifications, les différents éléments[5] ont des buts et fonctionnalités identiques ou du moins le plus similaire possible. Chacun des 4 éléments peut-être décomposé en deux parties internes, chacune s'occupant de gérer une des actions principales du client : d'une part s'informer sur les offres de vols et d'hôtels et d'autre part effectuer le paiement d'une réservation sur un vol et dans un hôtel. Ainsi le client contient deux éléments ou deux services associés à ces deux actions dans les trois spécifications. Dans la même logique, l'agence de voyage (*TravelAgency*) contient un premier élément pour les demandes d'information (*SearchVacationOffer*) et un deuxième élément pour valider et payer les réservations (*ReserveVacation*). La compagnie aérienne (*AirlineCompany*) fonctionne sur le même schéma. Le dernier élément, la chaîne d'hôtel (*HotelChain*) est différent : en effet, en interne, la chaîne d'hôtels gère plusieurs hôtels (*InternalHotel*) et un élément particulier *Management* a pour but de diffuser les interactions et les demandes des clients (via l'agence de voyage) aux différents hôtels. Cela se fait via une interaction de haut niveau (représenté côté service et composant par un protocole et/ou un connecteur). Chaque hôtel possède lui aussi deux fonctionnalités internes : une pour les demandes d'information et l'autre pour les réservations.

4. CORRESPONDANCES SÉMANTIQUES

Dans cette section, nous établissons, en partant de nos trois modèles présentés plus haut, les correspondances sémantiques entre les concepts de chaque domaine. Concrètement, ces correspondances permettent de "traduire" une spécification faite pour un modèle vers un autre modèle.

4.1 Composants et services

4.1.1 Correspondances générales

Les modèles de composants et de services sont très proches. Conceptuellement, on peut même considérer que le modèle de composants est une projection directe du modèle de services en rajoutant simplement l'élément qui implémente les services, à savoir le composant. À partir de là, toutes les règles de correspondance seront directes et bidirectionnelles : un service primitif correspond à un composant primitif, un service composite à un composant composite, une interaction à une interaction similaire (RPC/RMI, délégation ou avec protocole), etc.

La table 1 détaille les correspondances bidirectionnelles entre les concepts du modèle de services et ceux du modèle de composants. Telles que présentées, les correspondances sont

[4]À l'exception des interactions internes de l'élément *Hotel* en " 1 / * " afin de montrer l'interaction complexe avec plusieurs hôtels internes comme détaillé ci-après. Les autres interactions sont à considérer par défaut comme de cardinalités " 1 / 1 ".

[5]Comme les trois spécifications correspondent à la même application et pour ne pas donner les explications en triple, nous expliquons le fonctionnement de l'application de manière commune. Un élément est alors soit un service, soit un composant ou bien encore un agent ou un groupe selon que l'on considère la spécification via le modèle de services, de composants ou d'agents.

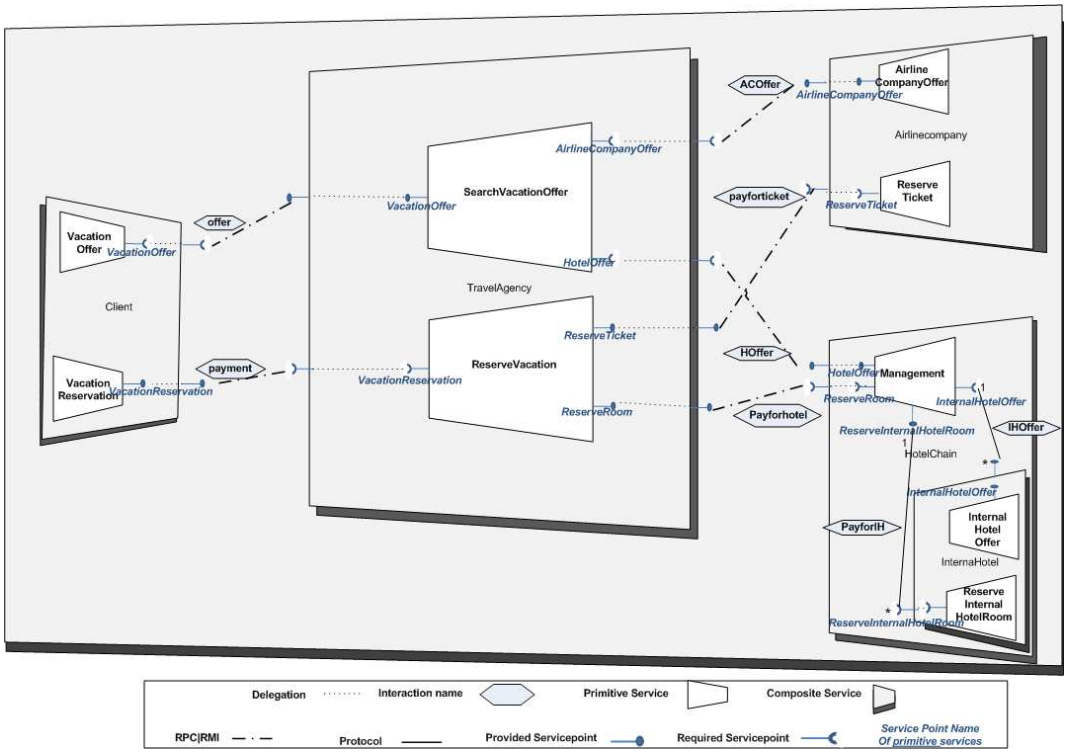

Figure 5: Spécification par des services de l'application de réservation

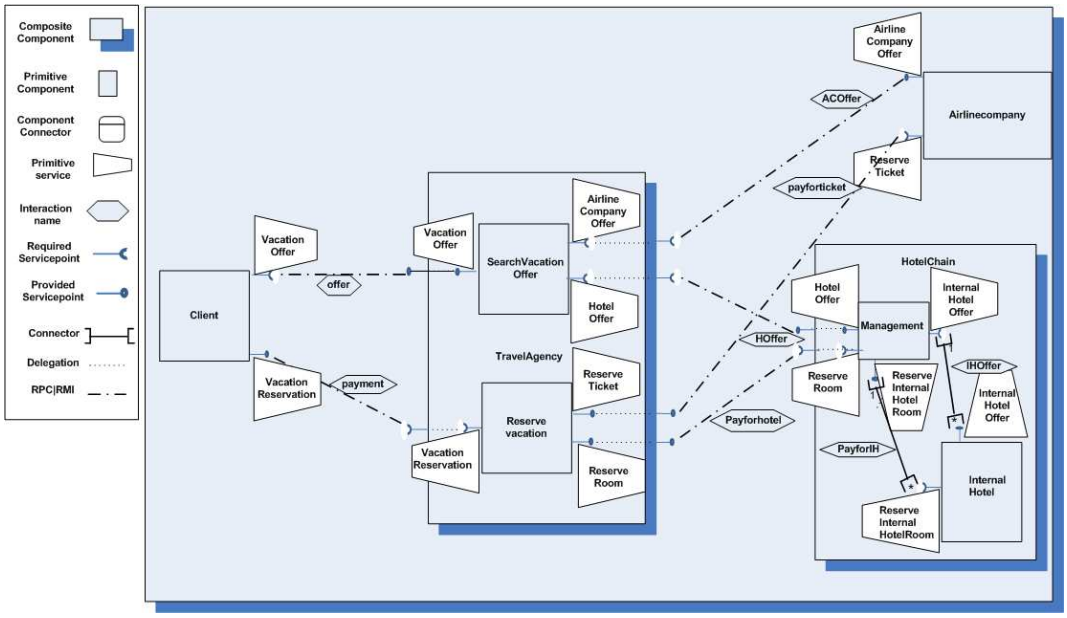

Figure 6: Spécification par des composants de l'application de réservation

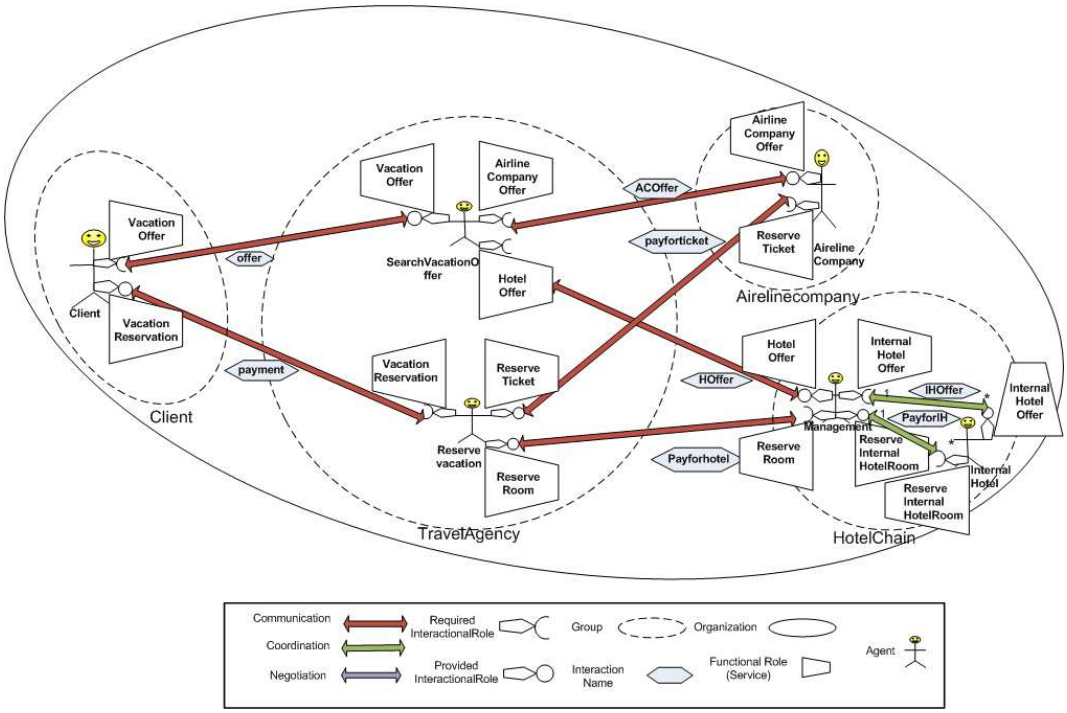

Figure 7: Spécification par des agents de l'application de réservation

Coté services	Coté composants
Service primitif	**Composant primitif** du même nom
Service composite (resp. **application**) avec services internes	**Composant composite** (resp. **application**) du même nom avec composants internes équivalents aux services internes
Point de service offert (resp. requis) par un service avec son ensemble d'opérations	**Point de service** offert (resp. requis) par le composant équivalent au service avec un **service** (du nom du point de service coté service) associé contenant les opérations de l'ensemble
Interaction basique d'un certain type (resp. associée à un **protocole**) et ses **rôles** associés à des points de service	**Interaction basique** du même type (resp. associée à un même **protocole** et à un **connecteur**) avec les mêmes **rôles** associés aux points de service équivalents sur les composants

Table 1: Correspondances services / composants

écrites dans le sens services vers composants. En pratique, les mêmes correspondances fonctionnent dans l'autre sens de manière symétrique. Il s'agit donc bien de correspondances bidirectionnelles. Sauf indication contraire, cela sera également le cas pour les correspondances des sections suivantes pour services / agents et composants / agents.

Notons tout de même une différence notable : la particularité du modèle de services est de ne pas définir les éléments support qui réaliseront les services (des composants ou des agents dans les autres modèles). Dans le modèle de services, un service contient plusieurs ensembles d'opérations ; chaque ensemble étant associé à un point de service offert ou requis. Ici, c'est le service qui sert de lien logique entre les ensembles d'opérations. Coté composant, cette notion de service global disparaît puisque le composant devient l'implémentation concrète de ce service. À partir de là, chaque ensemble d'opérations du service associé à un point de service correspond au même ensemble d'opérations associé au point de service équivalent du composant. Le point à noter ici est que nous avons nommé service cet ensemble d'opérations associé à un point de service coté composant. Un composant est donc associé à plusieurs services – chaque service étant un ensemble d'opérations – qui correspondent aux ensembles d'opérations des points de service du coté du modèle de services. La notion de service, bien que systématiquement associée à des opérations, n'est donc pas exactement la même dans les deux modèles. Pour illustrer cela, la figure 8 montre l'exemple d'un service primitif avec 3 points de service associés (et donc 3 ensembles d'opérations associés) et son composant primitif correspondant avec les 3 services équivalents (reprenant les 3 ensembles d'opérations).

Concernant le connecteur, il n'existe que du côté composant

mais pas du côté service. Le connecteur est la réalisation d'un protocole, tout comme le composant est la réalisation d'un service. Dans la même logique que pour le modèle de services où nous avons un service abstrait sans son "support d'implémentation" (le composant), le protocole n'a donc pas non plus son support d'implémentation (le connecteur) dans ce modèle et seul le protocole est présent.

Le composant connecteur est un élément particulier au niveau des correspondances à deux points de vue. Le premier est que dans le sens services vers composants, à l'exception de la variante présentée ci-dessous, il n'existe pas de correspondance entre un élément du modèle de services et un composant connecteur. Le second est que dans l'autre sens, composants vers services, le composant connecteur est géré comme un composant ordinaire en appliquant les règles de la table 1 pour les composants primitifs ou composites selon la nature du composant connecteur. La seule différence est que son protocole associé est alors ignoré.

En plus de la figure 8, les figures 5 et 6, spécifiant respectivement notre application via le modèle de services et de composants, montrent d'autres exemples d'applications de correspondances : à l'exception des éléments client et hôtels internes traités par une variante de correspondance comme précisé ci-dessous, tous les autres éléments se correspondent mutuellement via les règles que nous venons de définir.

4.1.2 *Variantes de correspondances*

Il est possible d'envisager des variantes de correspondances à la place de certaines correspondances présentées ci-dessus. La première variante consiste côté composants à utiliser un composant connecteur primitif à la place d'un connecteur associé à un protocole. Dans ce cas, on ajoutera les points de service symétriques (mêmes services associés mais en passant d'un offert à un requis ou inversement et avec une interaction de type RPC/RMI) sur le composant connecteur par rapport aux composants connectés. La figure 9 montre les deux choix possibles entre un connecteur et un composant connecteur. Coté service (partie (a)), deux points de service sont liés par un protocole. Coté composant, les points de service sont soit liés par un connecteur (partie (b)) soit par un composant connecteur primitif possédant les points de service symétriques (partie (c)).

Une autre variante consiste à ne pas transformer systématiquement un composite (resp. primitif) en un composite (resp. primitif) de l'autre modèle. La règle ici est que si dans un composite (service ou composant), les éléments internes ne sont pas liés par des interactions internes, alors on peut transformer ce composite en un primitif. Si les éléments internes sont reliés, cela signifie que le concepteur a explicitement spécifié des interactions internes entre ces éléments et que "fusionner" ces éléments en un seul ferait perdre ces interactions et cette structuration interne explicite. Il y aurait donc perte d'information. La figure 10 montre par exemple un service composite (à gauche) avec deux services primitifs internes transformé en un composant primitif (à droite). Cet exemple est intéressant dans le cas où on ne voudrait pas modéliser un service comme implicitement formé d'ensembles d'opérations offerts ou requis (via les points de service associés) mais dédier explicitement chaque ensemble d'opérations à un service particulier (c'est-à-dire un service primitif

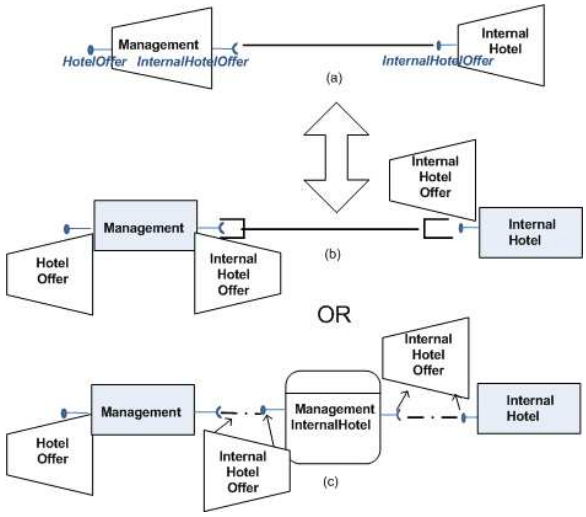

Figure 9: Variante : protocole (a) vers connecteur (b) ou composant connecteur (c)

ne contenant alors qu'un seul point de service). Coté composant, il reste par contre plus pertinent d'avoir un composant primitif vu que l'on retrouvera nativement par les correspondances chaque ensemble d'opérations associé à un point de service et donc à un service.

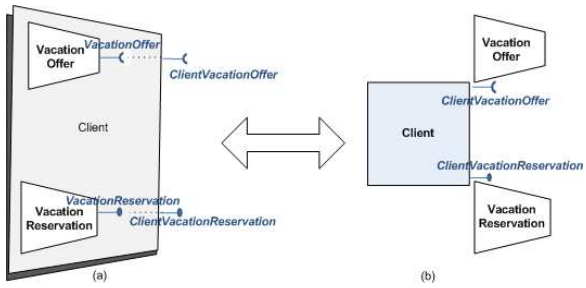

Figure 10: Variante : service composite (a) et composant primitif (b)

4.2 Agents et services

4.2.1 *Correspondances générales*

De manière similaire à un composant, un agent est un élément qui réalise l'implémentation des services. Mais on peut constater que les modèles de services et d'agents sont relativement éloignés avec de nombreux concepts propres à chaque domaine. En pratique, il existe tout de même des correspondances entre les concepts principaux de chaque domaine.

La notion de service est associée à des opérations dans les deux modèles. Cependant cette association n'est pas vue de la même manière, notamment parce que le concept de point de service n'est pas présent dans le modèle d'agents. En effet, dans une approche multi-agents organisationnelle, on va considérer que toute interaction se fait à travers les rôles et que c'est la notion de rôle d'interaction qui permet d'offrir ou requérir le service d'un agent défini par un rôle fonctionnel. Un rôle fonctionnel côté agent correspond donc à un point de service avec ses opérations associées côté service.

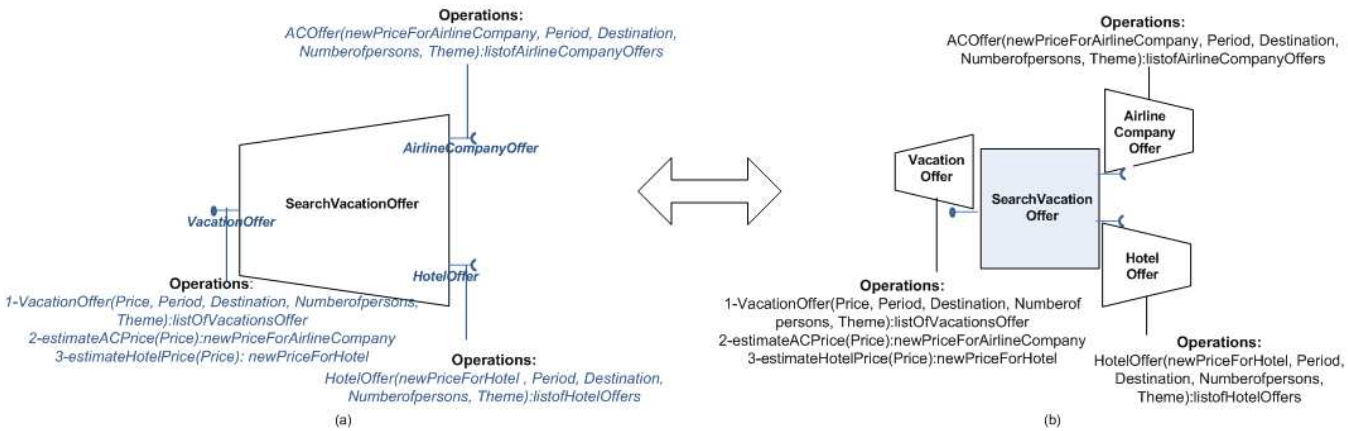

Figure 8: Service primitif (a) et composant primitif (b) avec plusieurs points de services

La table 2 présente de façon synthétique les correspondances bidirectionnelles autour des concepts centraux de service et d'interaction du modèle de services et de ceux du modèle d'agents. Concernant les buts "par défaut", cela est à comprendre que dans le sens services vers agents, on crée un but associé à chaque agent ou groupe avec son champ description non renseigné (le concepteur aura pour charge de le modifier). Dans l'autre sens, agents vers services, le but associé à un agent ou à un groupe est ignoré.

Coté services	**Coté agents**
Service primitif	**Agent** du même nom avec un **but** par défaut
Service composite avec services internes primitifs	**Groupe** du même nom contenant un agent par service interne et un **but** par défaut (figure 12)
Application	**Organisation** du même nom avec un **but** par défaut
Point de service offert (resp. requis) par un service avec son **rôle** et son ensemble d'opérations	**Rôle fonctionnel** de même nom avec le même ensemble d'opérations et associé à un **rôle interactionnel** qui prend le nom du rôle associé au point de service et son type (offert ou requis). Ces deux rôles sont affectés à l'agent correspondant au service (figure 11)
Interaction avec **protocole** entre points de service	**Interaction** équivalente entre les **rôles interactionnels** associés aux rôles fonctionnels correspondant aux points de service (figure 13)
Interaction basique d'un certain type entre points de service	**Communication** de type protocole de base entre les **rôles interactionnels** associés aux rôles fonctionnels correspondant aux points de service

Table 2: Correspondances services / agents

On ne trouve pas l'équivalent de la notion de composite au niveau d'un agent. Cette notion doit être prise en compte au niveau de la structure organisationnelle. Cela soulève différents problèmes au niveau de la structure elle même mais également au niveau de la notion de délégation. La structure organisationnelle du modèle d'agents n'offre que deux niveaux : le groupe et l'agent. Cela pose problème pour une hiérarchie de composition à plus de 2 niveaux comme par exemple un service interne d'un composite qui est également un composite (c'est pour cela que la table 2 présente une correspondance seulement pour un service composite contenant des services primitifs : la correspondance est alors directe et simple). Une façon de contourner ce problème est d'utiliser le fait qu'un agent puisse appartenir à différents groupes pour représenter les niveaux structurels d'une organisation. C'est l'approche utilisée par exemple dans le modèle AGR [10]. Un agent membre d'un groupe de niveau n et également membre d'un groupe de niveau $n+1$ peut alors être considéré comme le représentant du groupe de niveau inférieur et toutes les interactions entre les deux niveaux doivent passer par lui. Ensuite, le principe de délégation d'un point de service externe vers un point de service interne au sein d'un service composite n'existe pas côté agent. Lors de la transformation d'un service composite en un groupe d'agents, on ne considère que les points de service internes pour créer les rôles fonctionnels des agents (un point de services interne est en effet associé aux mêmes opérations que son point de service par délégation au niveau du composite). Pour la transformation inverse, il faudra créer les points de services externes sur le composite ainsi que les rôles et les interactions basiques de type délégation nécessaires.

L'absence de la notion de composite au niveau du modèle d'agents rend également plus complexe la transformation de la notion d'application du modèle de services. De manière intuitive, on peut trouver une correspondance entre le concept d'application du modèle de services et celui d'organisation du modèle d'agents. Mis à part le problème de la notion de but, déjà abordé et qui devra être traité de manière similaire, si tous les services de l'application sont composites, la correspondance entre les deux concepts est directe : une application composée de services composites devient une organisation composée de groupes et vice versa. Comme une organisation est toujours composée de groupes, il n'y a pas de problème pour passer du modèle d'agents au modèle de services. Par contre une application côté service peut aussi contenir directement des services primitifs. Pour transfor-

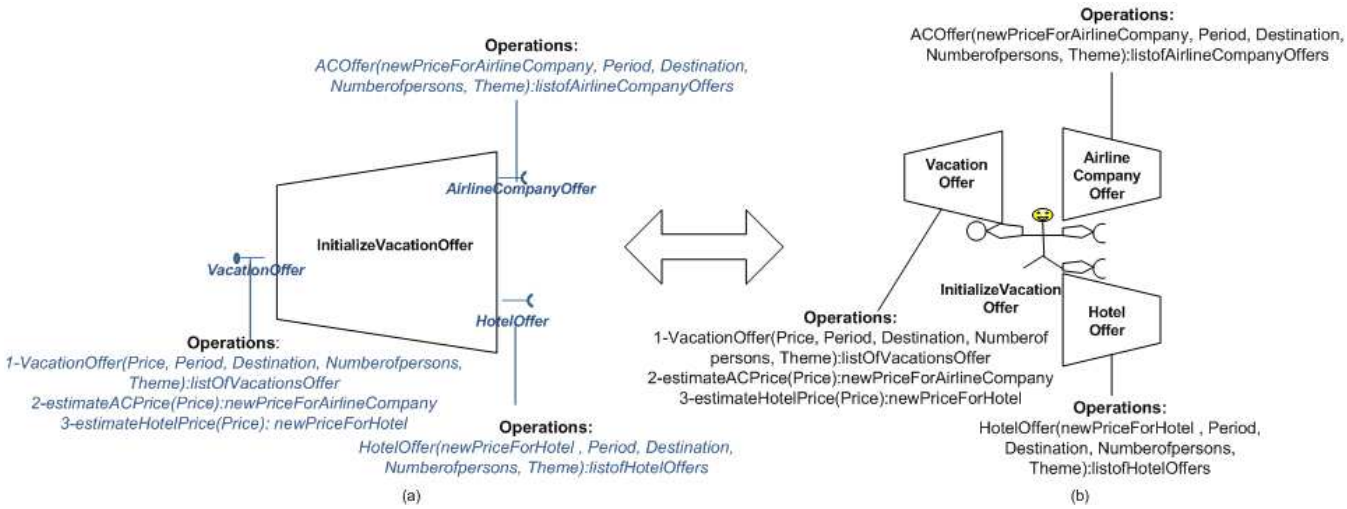

Figure 11: Service primitif avec plusieurs points de service (a), un agent avec les rôles fonctionnels et les rôles interactionnels associés (b)

mer une application côté service en organisation côté agent, il faudra alors en plus créer un groupe pour chaque agent correspondant à un service primitif ; ce groupe ayant le même nom que l'agent qu'il contient.

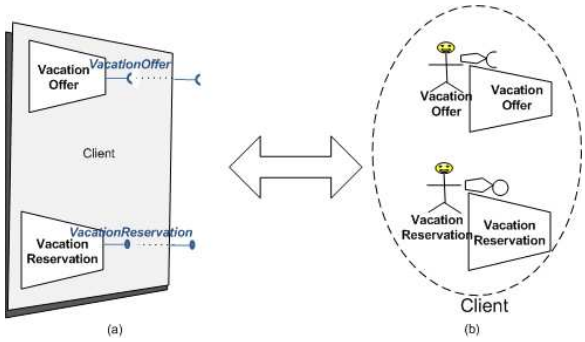

Figure 12: Service composite (a), groupe d'agents (b)

Il est à noter que lorsqu'un service primitif interagit avec d'autres services via un protocole (ex : diffusion, facilitateur, médiateur...) l'agent correspondant doit posséder les capacités requises en terme de communication (langage et protocoles élémentaires comme FIPA ACL, FIPA Query, FIPA Request ...) et éventuellement de raisonnement et de décision (coordination, négociation) pour pouvoir jouer les rôles interactionnels associés au protocole. Par exemple, le service *Management* d'une chaîne d'hôtels propose une liste d'hôtels disponibles après avoir diffusé la demande en provenance de l'agence de voyage et collecté les offres de différents hôtels de la chaîne (ces capacités ne sont pas représentées sur les différentes figures pour ne pas les surcharger). Malheureusement, il n'y a pas d'équivalent au niveau du modèle de services pour ce concept de capacité, ni pour celui de tâche ou de but dont nous avons parlé ci-dessus. Quand on passera du modèle d'agents au modèle de services, tous ces éléments seront "supprimés" et dans l'autre sens, le concepteur devra rajouter manuellement ces éléments sur la spécification obtenue.

Au niveau des interactions non basiques, on dispose côté service du concept unique de protocole alors que du côté agent, il existe une classification bien plus riche avec trois principaux types d'interaction (communication, coordination et négociation). Il n'y a donc pas le même niveau de détail des deux côtés. Quand on passera du côté agent à service, une interaction non basique sera systématiquement associée à un protocole, mais dans l'autre sens, il faudra demander au concepteur de choisir parmi un des trois types d'interaction ou choisir une interaction par défaut.

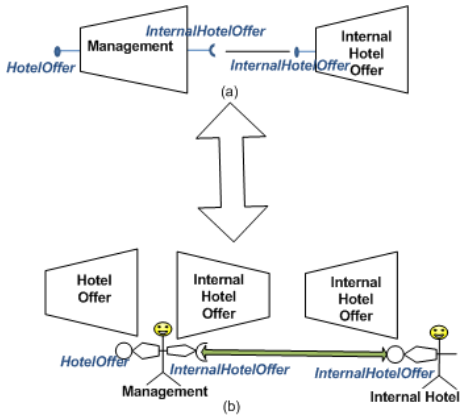

Figure 13: Services et protocole d'interaction (a), agents et protocole d'interaction (b)

4.2.2 Variantes de correspondances

De manière similaire à ce qui a été présenté au niveau des services et composants, une variante de correspondance consiste à ne pas associer systématiquement un service primitif du modèle de services à un agent du modèle d'agents. Si l'on reprend l'exemple de service composite dont les deux services internes ne sont pas liés par des interactions internes (figure 10), il n'est pas nécessaire d'avoir deux agents diffé-

rents. On peut faire le choix de regrouper les services dans un même agent en lui affectant les deux rôles fonctionnels et les deux rôles interactionnels correspondant aux deux points de service, comme le montre la figure 14.

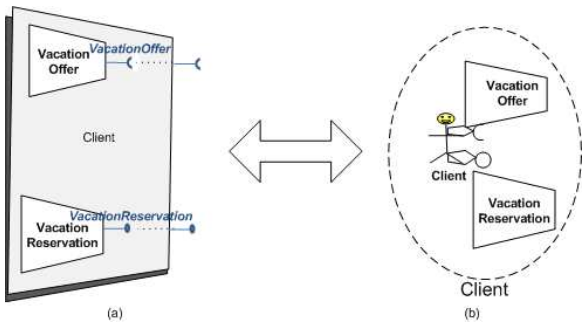

Figure 14: Variante : services internes d'un service composite (a) regroupés dans un seul agent (b)

4.3 Composants et agents

4.3.1 Correspondances générales

Dans la mesure où le modèle de composants est très proche du modèle de services, nous allons retrouver certaines similitudes au niveau des règles de correspondances entre les composants et les agents et entre les services et les agents mais également les mêmes limites en ce qui concerne les concepts propres à chaque domaine, en particulier la notion de composite côté composant et la notion de but côté agent. En revanche l'agent comme le composant étant des éléments d'implémentation de services, le concept de service est perçu dans le modèle d'agents de façon similaire à celui du modèle de composants à ceci près qu'il sera considéré comme une abstraction de la notion de rôle fonctionnel. Un agent (resp. composant primitif) peut être associé à plusieurs rôles fonctionnels (resp. services), chaque rôle fonctionnel (resp. service) est composé d'un ensemble d'opérations. Un rôle interactionnel (resp. point de service) est associé à un rôle fonctionnel (resp. service). Des agents (resp. composants primitifs) différents interagissent entre eux via des interactions associées à leurs rôles interactionnels (resp. points de service). Pour reprendre un exemple précédent, la partie composants (b) de la figure 8 correspond à la partie agents (b) de la figure 11. À noter cependant que la notion de rôle interactionnel côté agent correspond au regroupement des deux notions de point de service et de rôle côté composant. Cela nécessite de faire le choix d'une convention pour nommer le rôle interactionnel afin de ne pas perdre d'information. On peut par exemple utiliser un nom composé de la forme *<nom du rôle>.<nom du point de service>*.

La table 3 présente de façon synthétique les correspondances bidirectionnelles entre les concepts centraux du modèle de composants et du modèle d'agents.

On retrouve les mêmes problèmes pour la transformation d'un composant composite que ceux rencontrés pour la transformation d'un service composite. Il faudra adopter une démarche similaire pour le traitement de hiérarchies de composition, de la notion d'application et du mécanisme de délégation.

Avec la même logique que pour les correspondances entre services et composants, un composant connecteur est géré de la façon suivante. Dans le sens agents vers composants, à l'exception de la variante ci-dessous, il n'existe pas de cas pour aboutir à un composant connecteur côté composants. Dans le sens composants vers agents, un composant connecteur est traité comme un composant classique en ignorant son protocole. Ceci étant, le concepteur pourra manuellement rajouter à l'agent (ou aux agents dans le cas d'un composant connecteur composite) correspondant au composant connecteur les capacités requises pour réaliser son protocole.

Coté composants	Coté agents
Composant primitif	**Agent** du même nom avec un **but** par défaut
Composant composite avec composants internes primitifs	**Groupe** du même nom contenant un agent par composants internes et un **but** par défaut
Application	**Organisation** du même nom avec un **but** par défaut
Service d'un composant avec son ensemble d'opérations	**Rôle fonctionnel** du même nom avec le même ensemble d'opérations et affecté à l'agent correspondant au composant associé au service
Point de service offert (resp. requis) par un composant avec le **rôle** associé	**Rôle interactionnel** ayant un nom composé du nom du point de service et du nom du rôle associé. Prend le type du point de service (offert ou requis). Ce rôle est affecté à l'agent correspondant au composant
Interaction et ses **rôles** associés à des points de service	**Interaction** entre **rôles interactionnels** équivalents
Interaction basique d'un certain type	**Communication** de type protocole de base
Connecteur associé à un **protocole**	**Interaction** associée à un **protocole** équivalent

Table 3: Correspondances composants / agents

Enfin, il n'y a pas d'équivalent au niveau du modèle de composants pour les concepts de capacité, de tâche et de but du modèle d'agents. Quand on passera du modèle d'agents au modèle de composants, tous ces éléments seront "supprimés" et dans l'autre sens, le concepteur devra rajouter manuellement ces éléments sur la spécification obtenue ou modifier la description des buts par défaut.

4.3.2 Variantes de correspondances

De manière similaire à ce qui a été présenté au niveau de la correspondance entre le modèle d'agents et le modèle de services, une variante consiste à ne pas transformer systématiquement un composant composite en un groupe contenant des agents. On peut faire le choix de regrouper dans un même agent les services de composants internes d'un composite s'ils ne sont pas en interaction.

De même, en reprenant l'idée d'une variante de correspon-

dance entre services et composants, on pourra faire correspondre un composant connecteur primitif avec des "points de services symétriques" avec une interaction complexe entre agents réalisant le même protocole que le composant connecteur.

5. TRAVAUX CONNEXES

Pour rappel, les modèles de services, composants et agents étudiés pour définir nos trois modèles sont présentés et analysés dans [1, 3]. Dans cette section, nous allons uniquement nous intéresser aux travaux liés à la contribution de notre article, à savoir la définition de correspondances entre les concepts des approches services, composants et agents. À notre connaissance, il existe très peu de travaux sur la définition précise de correspondances entre ces trois domaines. Définition précise est à comprendre ici comme étant suffisamment détaillée pour être par exemple implémentable via des transformations de modèles concrètes dans le contexte de l'IDM. C'est le cas des correspondances que nous avons présentées dans cet article. Elles sont en effet la spécification informelle des transformations de modèles que nous implémenterons par la suite.

Le travail le plus proche du nôtre est [12] qui définit une transformation de modèles (et donc des correspondances) entre le modèle de services SoaML de l'OMG [11] et un modèle propre d'agents basé sur l'approche BDI (*Believe Desire Intention*) [16]. Nous pouvons noter deux points de divergence qui empêchent la réutilisation du contenu de ce travail. Le premier point est que le modèle d'agents considéré est assez éloigné de nos préoccupations. En effet, par volonté d'intégration des trois domaines (services, composants, agents) comme expliqué en introduction et en section 2, nous nous focalisons sur l'interopérabilité par les services ainsi que les interactions. Le modèle BDI est lui tourné vers les notions de buts, de plans et de raisonnement délibératif. Le deuxième point est que SoaML est lui aussi un modèle ne rentrant pas dans le cadre de notre processus de conception. En effet, au niveau le plus abstrait, celui du modèle de services, nous définissons uniquement les services et leurs interactions, sans préciser quel élément (agent ou composant) les implémente. Or SoaML définit cet élément via le concept de participant. Il va même plus loin en intégrant directement le concept d'agent comme étant une spécialisation d'un participant. [12] définit alors directement des correspondances entre l'agent du modèle de services SoaML et l'agent de leur modèle propre d'agents, ce qui n'est pas possible ni souhaitable dans notre approche où le modèle de services est abstrait.

[13] et [5] sont des études comparant les approches services et composants. Mais ces comparaisons restent à un niveau général d'équivalences de concepts, sans rentrer dans un détail plus précis de correspondances implémentables comme nous le faisons. Au delà de certaines problématiques qui ne nous concernent pas, comme le cycle de vie d'un élément par exemple, les correspondances générales de ces travaux se retrouvent dans nos correspondances détaillées.

La limite d'ordre général de ces trois approches ([12, 13, 5]) est également de ne prendre en compte que deux domaines à la fois : soit composants et services, soit composants et agents.

Enfin, nous n'avons trouvé aucune approche définissant comme la nôtre simultanément des correspondances entre les concepts des trois domaines de services, agents et composants.

6. CONCLUSION

En nous basant sur des modèles de services, d'agents et de composants préalablement définis, nous avons établi des règles de correspondances sémantiques entre ces modèles (incluant des variantes de correspondances en fonction des besoins du concepteur). Ces correspondances entrent dans le cadre d'un processus de conception permettant d'intégrer les approches services, composants et agents afin de tirer partie des avantages de chaque approche dans la spécification d'une application distribuée.

Le modèle de composants peut être vu comme une mise en œuvre du modèle abstrait de services où un composant réalise un service et un connecteur une interaction complexe. Ainsi, il y a des correspondances bidirectionnelles permettant de passer de manière systématique et totale d'une spécification services à composants et inversement. Du côté des agents, les choses sont un peu plus compliquées par rapport aux modèles de services ou de composants. Les concepts principaux d'interaction et de service trouvent une correspondance bidirectionnelle systématique mais certains concepts secondaires n'existent que côté agent sans équivalent dans les deux autres modèles. Du point de vue des interactions, il y a une richesse plus grande côté agents que l'on ne retrouve pas côté services ou composants. En contrepartie, la mise en œuvre de la composition est moins naturelle côté agent. Ces différences au niveau interaction et composition sont finalement assez logiques : on retrouve là en effet les points forts et faibles des agents par rapport aux deux autres approches, comme décrit en introduction.

Les correspondances que nous avons définies s'appliquent sur nos modèles de services, composants et agents. Ceci dit, ces modèles sont relativement généraux et reprennent les principes conceptuels de chaque domaine. Notre étude des correspondances sémantiques entre ces trois domaines peut donc être considérée comme une étude théorique et générale, et pas seulement dédiée à nos modèles. Il sera possible de réutiliser assez facilement, en les adaptant, les principes de correspondances que nous avons établis pour des modèles particuliers de composants, agents ou services.

La suite de notre travail consiste principalement en deux étapes. La première va être de définir CASOM, c'est-à-dire le modèle mixte qui contient à la fois des aspects composants et agents. Ce modèle sera défini en nous basant sur les correspondances sémantiques, en déterminant les éléments pivots que l'on retrouve dans les deux modèles de composants et d'agents. En complément de CASOM, nous définirons, en adaptant les correspondances déjà définies, les liens entre CASOM et les trois autres modèles (services, composants, agents). Nous fournirons également un guide méthodologique précisant dans quels contextes l'usage d'un composant ou d'un agent est préférable dans la spécification d'une application. Ce guide pourra être utilisé pour définir une nouvelle application ou pour modifier la spécification d'une application existante en transformant, via nos correspondances, des agents en composants ou inversement.

La deuxième étape va être d'implémenter ces 4 modèles et les correspondances sémantiques dans un environnement d'ingénierie des modèles afin de pouvoir en pratique spécifier des applications via nos modèles. Nous utiliserons pour cela la plateforme Eclipse / EMF. Chaque modèle sera implémenté sous la forme d'un méta-modèle Ecore et des transformations de modèles permettront de passer d'un modèle à un autre. Comme nous avons vu que certaines correspondances requièrent un choix de la part du concepteur ou qu'il peut retravailler une spécification d'application à la main, il faudra contrôler que les modifications manuelles faites par le concepteur sont valides. Cela pourra être réalisé par des contrats de transformations de modèles comme expliqué dans [8].

7. REFERENCES

[1] N. A. Aboud, P. Aniorté, E. Cariou, and E. Gouardères. Towards a Component Agent Service Oriented Model (CASOM). Technical report, LIUPPA / Université de Pau, July 2010.

[2] N. A. Aboud, E. Cariou, and E. Gouardères. Towards a Component Agent Service Oriented Model. In *5ème Conférence Francophone sur les Architectures Logicielles (CAL 2011)*, 2011.

[3] N. A. Aboud, E. Cariou, E. Gouardères, and P. Aniorté. Service-oriented Integration of Component and Agent Models. In *Special session on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC) of the 6th International Conference on Software and Data Technologies (ICSOFT 2011)*, pages 327–336. SciTePress Digital Library, 2011.

[4] F. Bergenti and M. N. Huhns. *Methodologies and Software Engineering for Agent Systems : The Agent-Oriented Software Engineering handbook*, chapter On the Use of Agents as Components of Software Systems, pages 19–32. Kluwer Academic Publishing, 2004.

[5] H. P. Breivold and M. Larsson. Component-Based and Service-Oriented Software Engineering : Key Concepts and Principles. In *EUROMICRO'07 : Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 13–20. IEEE Computer Society, 2007.

[6] J.-P. Briot, T. Meurisse, and F. Peschanski. Architectural design of component-based agents : A behavior-based approach. In *4th International Workshop on Programming Multi-Agent Systems (ProMAS 2006)*, volume 4411 of *LNCS*, pages 71–90. Springer, 2007.

[7] E. Bruneton, T. Coupaye, and J.-B. Stefani. The Fractal Component Model, 2004. http ://fractal.ow2.org/specification/index.html.

[8] E. Cariou, N. Belloir, F. Barbier, and N. Djemam. OCL Contracts for the Verification of Model Transformations. In *Proceedings of the Workshop The Pragmatics of OCL and Other Textual Specification Languages at MoDELS 2009*, volume 24. Electronic Communications of the EASST, 2009.

[9] E. Cariou, A. Beugnard, and J.-M. Jézéquel. An Architecture and a Process for Implementing Distributed Collaborations. In *The 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2002)*. IEEE Computer Society, 2002.

[10] J. Ferber, O. Gutknecht, and F. Michel. From Agents to Organizations : an Organizational View of MultiAgent Systems. In *Agent-Oriented Software Engineering (AOSE) IV*, volume 2935 of *LNCS*. Springer, 2004.

[11] O. M. Group. Service oriented architecture Modeling Language (SoaML). http ://www.omg.org/spec/SoaML/, 2012.

[12] C. Hahn, S. Jacobi, and D. Raber. Enhancing the Interoperability between Multiagent Systems and Service-Oriented Architectures through a Model-Driven Approach. In *the 8th German conference on Multiagent system technologies (MATES' 10)*, volume 6251 of *LNCS*, pages 88–99. Springer, 2010.

[13] A. A. Hock-koon and M. Oussalah. The Product-Process-Quality Framework. In *37th EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA 2011)*, pages 20–27. IEEE, 2011.

[14] R. Krutisch, P. Meier, and M. Wirsing. The Agent Component Approach, Combining Agents, and Components. In *Multiagent System Technologies, First German Conference (MATES' 03)*, volume 2831 of *LNCS*, pages 1–12. Springer, 2003.

[15] J. Lind. Relating Agent Technology and Component Models, 2001. http ://www.agentlab.de/documents/Lind2001e.pdf.

[16] A. S. Rao and M. P. Georgeff. BDI Agents : From Theory to Practice. In *the First International Conference on Multiagent Systems (ICMAS 95)*, pages 312–319. The MIT Press, 1995.

[17] S. N. Schiaffino and A. Amandi. User - interface agent interaction : personalization issues. *Int. J. Hum.-Comput. Stud., Elsevier*, 60(1) :129–148, 2004.

[18] M. P. Singh and M. N. Huhns. *Service-Oriented Computing - Semantics, Processes, Agents*. Wiley, 2005.

[19] J. Vázquez-Salceda, V. Dignum, and F. Dignum. Organizing Multiagent Systems. *Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers*, 11 :307–360, November 2005.