HAL
open science

Méthodologie ODAC : Le guide de spécification comportementale

Marie-Pierre Gervais

To cite this version:

Marie-Pierre Gervais. Méthodologie ODAC : Le guide de spécification comportementale. [Rapport de recherche] lip6.2001.024, LIP6. 2001. hal-02545596

HAL Id: hal-02545596

https://hal.science/hal-02545596v1

Submitted on 17 Apr 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Méthodologie ODAC :
Le guide de spécification comportementale

Marie-Pierre Gervais

LIP6

Marie-Pierre.Gervais@lip6.fr

Résumé : ODAC (Open Distributed Applications Construction) fournit un guide méthodologique de développement de systèmes répartis permettant à un constructeur de décrire le système qu'il développe, c'est-à-dire d'écrire la spécification de son système, avec la notation UML selon une sémantique ODP. La méthodologie fait la distinction entre la spécification du comportement du système indépendamment de tout contexte d'exécution, appelée spécification comportementale, et le paramétrage de cette spécification selon un environnement d'exécution cible, qui la transforme en spécification opérationnelle. Un guide de spécification comportementale est donc fourni aux constructeurs de systèmes pour les assister dans leur rédaction de spécifications comportementales. Il est complété par des guides de description d'environnements cibles, tel que le guide MASIF-DESIGN pour les systèmes à base d'agents et le guide ODAC*for*ANTS pour les réseaux actifs. Ce rapport se concentre sur le guide de spécification comportementale ODAC.

Mots-Clés : traitement réparti ouvert (ODP), profil UML, analyse et conception d'applications réparties, guide de spécification comportementale

Table des matières

1 Introduction 1
2 Présentation générale de la méthodologie ODAC 2
2.1 Étude de cas : l'agence de voyages électronique 4
3 Le guide de spécification comportementale 4
3.1 Profil pour la spécification du point de vue Entreprise et ses règles d'application 5
3.2 Profil pour la spécification du point de vue Information et ses règles d'application 10
3.3 Profil pour la spécification du point de vue Traitement et ses règles d'application 13
3.4 Les règles de correspondance entre les spécifications 23
4 La spécification opérationnelle 25
5 Conclusion 26
6 Bibliographie 26
7 Annexe : ODP : la norme de traitement réparti ouvert 27
7.1 Le modèle objet 27
7.2 Le modèle architectural 28

1 Introduction

L'évolution conjuguée de la technologie des télécommunications et de la structure des organisations conduit à l'émergence d'applications réparties de grande complexité. Ces applications sont ouvertes pour supporter l'intégration permanente de services sophistiqués et interagir dans de multiples environnements eux-mêmes évolutifs en permanence. Leur construction, leur déploiement, leur fonctionnement, leur maintenance posent des problèmes difficiles pour lesquels les solutions actuelles sont jugées insuffisantes par les utilisateurs et coûteuses par les opérateurs de services.

L'environnement d'exécution d'une application répartie est constitué de systèmes interconnectés généralement hétérogènes. L'hétérogénéité technique se caractérise par des machines d'architectures différentes, des systèmes d'exploitation distincts ou encore des réseaux de nature diverse. L'hétérogénéité organisationnelle apparaît lorsque l'environnement d'exécution relève de domaines d'organisation multiples, avec des objectifs, des politiques d'usage et de gestion et des contraintes différentes. L'interconnexion de systèmes soulève de plus les problèmes plus classiques de la concurrence, de l'asynchronisme, ou de l'absence d'état global.

Maîtriser cette complexité est nécessaire pour supporter des applications constituées de composants interagissants et localisés sur des sites distants. Le terme "composant" désigne ici une unité fonctionnelle indépendante douée d'une forte autonomie. Construire une application par assemblage de composants, dont les interactions réalisent les fonctionnalités attendues, pose des problèmes d'intégration et par conséquent d'interopérabilité. L'intégration provient de la nécessité de conserver et faire évoluer les applications existantes pour des raisons économiques évidentes et également pour offrir une certaine continuité vis-à-vis des utilisateurs. Cependant, les composants de ces applications ne sont pas toujours homogènes ni propriétaires (par exemple, ils sont écrits au moyen de langages différents ou développés par des organisations distinctes). Leur réutilisation ou leur coopération pour offrir de nouvelles fonctionnalités reposent sur la capacité d'interactions cohérentes de tous leurs composants. La complexité du processus de construction de telles applications nécessite donc des environnements de développement incluant des méthodes et outils permettant aux équipes de développement de maîtriser l'interopérabilité dès les étapes de spécification et de la maintenir pendant tout le cycle de vie de ces applications, incluant l'analyse, la conception et l'implantation de l'application. Comme la terminologie n'est pas normalisée, pour éviter toute confusion, nous précisons ce que nous appelons analyse et conception en empruntant les définitions de [OCE 01]. L'*analyse* concerne l'élaboration d'une solution détaillée mais indépendante des moyens de réalisation. Elle comprend la description de tous les processus composant le fonctionnement du système, la définition des informations utilisées, la spécification des tâches à effectuer. La *conception* a pour but de conduire à l'implantation du système et à effectuer des choix d'opérationalisation des modèles pour la réalisation. Elle concerne les spécifications opérationnelles nécessaires pour assurer la réalisation du futur système. Elle comprend la prise en compte des moyens choisis pour la solution retenue. Nous complétons ces définitions par celle de l'implantation qui consiste à générer le code en fonction de l'environnement cible et de l'installer dans cet environnement pour procéder aux tests.

Le LIP6 a mis en œuvre un projet appelé ODAC (Open Distributed Applications Construction) visant à fournir une méthodologie pour le développement d'applications réparties [GER 00]. La méthodologie est suffisamment générale pour être utilisée pour tout type d'applications, néanmoins nous privilégions quelques domaines cibles comme les systèmes à base d'agents et les réseaux actifs [MUS 01][BOU 01]. Une méthodologie doit fournir des concepts, les règles d'utilisation de

ces concepts en les organisant en différentes étapes, le processus associé à chacune de ces étapes et une notation. ODAC utilise les concepts du modèle de référence du traitement réparti ouvert (Reference Model of Open Distributed Processing - RM-ODP). RM-ODP est une norme issue conjointement de l'ISO et l'ITU-T [ISO 95]. Elle définit un ensemble de concepts rigoureux pour la modélisation des systèmes répartis. Ce n'est pourtant pas une méthodologie aussi ne définit-elle pas de notation ni de processus. ODAC définit alors les règles d'utilisation des concepts ODP en identifiant différentes étapes et en associant à chacune d'elles un processus. Par ailleurs, ODAC utilise la notation UML standardisée par l'OMG [BOO 99].

Ainsi ODAC fournit un guide méthodologique de développement de système permettant à un constructeur de décrire le système qu'il développe, c'est-à-dire d'écrire la spécification de son système, avec la notation UML selon une sémantique ODP. La méthodologie fait la distinction entre la spécification du comportement du système indépendamment de tout contexte d'exécution, appelée spécification comportementale, et le paramétrage de cette spécification selon un environnement d'exécution cible, qui la transforme en spécification opérationnelle. Un guide de spécification comportementale est donc fourni aux constructeurs de systèmes pour les assister dans leur rédaction de spécifications comportementales. Il est complété par des guides de description d'environnements cibles, tel que le guide MASIF-DESIGN pour les systèmes à base d'agents et le guide ODAC*for*ANTS pour les réseaux actifs.

Ce rapport se concentre sur le guide de spécification comportementale ODAC. Il est organisé comme suit. Dans le paragraphe 2, nous faisons une présentation générale de la méthodologie ODAC en présentant l'étude de cas utilisée tout au long de ce document pour illustrer les principes de la méthodologie. Le paragraphe 3 est consacré au guide de spécification comportementale. Le paragraphe 4 traite de la spécification opérationnelle et de son obtention à partir de la description d'un environnement d'exécution. Le paragraphe 5 dresse le bilan de ces travaux et en donne les perspectives.

2 Présentation générale de la méthodologie ODAC

Le projet ODAC (Open Distributed Applications Construction) a pour objectif de fournir des méthodes et outils pour maîtriser la complexité du processus de construction d'applications réparties. Cette méthodologie vise à supporter l'analyse, la conception et le déploiement d'une application au sein d'environnements cibles telles les plates-formes à agents mobiles ou les réseaux actifs [GER 00]. Elle est fondée sur la norme de traitement réparti ouvert ODP [ISO 95][PUT 01].

ODP[1], élaborée conjointement par l'ISO et l'ITU-T, définit un ensemble de concepts rigoureux pour modéliser les systèmes répartis dont le principal est celui de « point de vue », concept permettant de structurer l'activité de modélisation. Le modèle de référence ODP (RM-ODP) définit cinq points de vue : Entreprise, Information, Traitement, Ingénierie et Technologie. Chacun d'eux se focalise sur un aspect spécifique et permet de spécifier le système selon un ensemble de concepts appropriés aux préoccupations liées à ce point de vue. Ainsi, le concept de point de vue est issu de l'approche de séparation des préoccupations. Le point de vue Entreprise permet de spécifier les besoins du système, donc de répondre au « pour quoi ? » du système. Le point de vue Information permet de spécifier l'information gérée et manipulée ainsi que les traitements de cette information par le système, donc répondre au « quoi ? ». Le point de vue Traitement permet d'exprimer une décomposition fonctionnelle du système en objets interagissant via leurs interfaces,

[1] Une introduction à ODP est fournie en annexe de ce document

donc de décrire le « comment faire ? » dans un sens fonctionnel. Le point de vue Ingénierie permet de décrire l'infrastructure répartie requise pour mettre en œuvre une spécification de traitement, donc de décrire le « où ? » et le point de vue Technologie est l'implantation du système. La spécification complète du système est alors constituée de l'ensemble des spécifications établies dans chaque point de vue avec leurs règles de correspondance. Cependant, RM-ODP fournit un cadre architectural et non une méthodologie. En effet, une méthodologie doit définir un ensemble de concepts, des règles d'utilisation de ces concepts selon les différentes étapes, un processus associé à ces étapes et une notation. Or ODP est peu prescriptive et ne fournit par exemple aucune notation ou aucun processus pour concrétiser l'élaboration d'une spécification de système. Il n'y a pas d'ordre ou de hiérarchie entre les points de vue.

ODAC utilise les concepts de RM-ODP. Le concept de point de vue, en particulier, est utilisé pour définir les étapes de la méthodologie et le processus associé à ces étapes. Par ailleurs, ODAC utilise la notation UML standardisée par l'OMG. Ainsi ODAC prescrit un ordre d'utilisation des points de vue pour l'élaboration de la spécification d'un système, à savoir Entreprise, Information, Traitement, Ingénierie et Technologie. Pour établir une comparaison avec les étapes classiques du développement logiciel, on peut considérer qu'ODAC associe de façon informelle les activités relevant des points de vue Entreprise, Information et Traitement à celles d'analyse, celles du point de vue Ingénierie à celle de la conception et celles du point de vue Technologie à celle d'implantation. Elle distingue alors dans ces étapes destinées à définir le système celles qui le décrivent indépendamment de tout environnement cible de celles qui le décrivent en fonction de l'environnement dans lequel il sera exécuté. Deux catégories de spécifications sont ainsi identifiées : la *spécification comportementale* du système et sa *spécification opérationnelle* (Figure 1).

La *spécification comportementale* résulte des spécifications établies dans les points de vue Entreprise, Information et Traitement. Elle décrit le système en fonction de son objectif, de sa place dans l'organisation dans laquelle il est développé, des informations qu'il traite et des tâches qu'il accomplit. Étant indépendante de tout environnement cible d'exécution, elle constitue un modèle indépendant de toute plate-forme (Platform-Independant Model) tel que décrit dans le MDA (Model Driven Architecture) de l'OMG [OMG 01]. Pour assister le modélisateur dans sa rédaction de spécification comportementale, nous fournissons un guide de spécification comportementale.

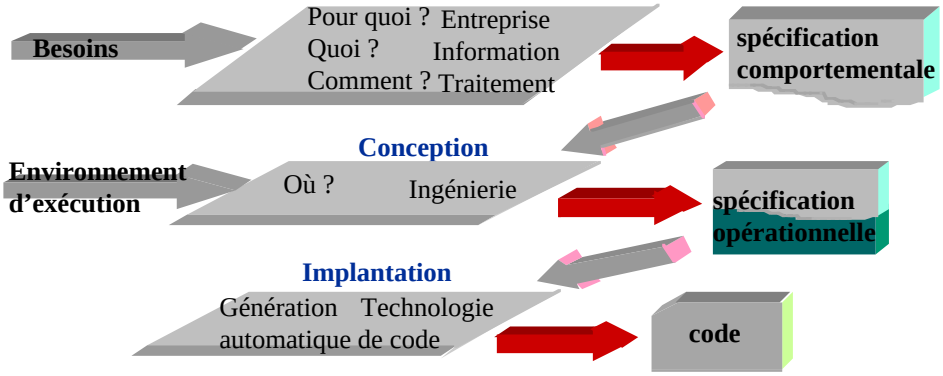

Figure 1. *Principes de la méthodologie ODAC*

La *spécification opérationnelle* résulte de la transformation de la spécification comportementale selon un environnement cible reflétant l'environnement réel d'exécution. Elle dépend de la spécification établie dans le point de vue Ingénierie qui décrit l'environnement d'exécution. Elle constitue la description à partir de laquelle du code est généré et l'implantation est réalisée. Elle

correspond donc à une transformation de la spécification comportementale paramétrée par la spécification d'ingénierie. C'est le modèle spécifique à une plate-forme (Platform-Specific Model) du MDA. Ainsi pour chaque catégorie d'environnements, la méthodologie ODAC fournit au concepteur un guide de spécification d'ingénierie sous forme d'un profil UML lui permettant de décrire l'environnement considéré. Actuellement, deux guides de spécification d'ingénierie sont proposés : le guide MASIF-DESIGN qui concerne les plates-formes à agents mobiles [MUS 01] et le guide ODAC*for*ANTS développé dans le cadre du projet RNRT Amarrage et qui concerne les réseaux actifs [BOU 01][GER 01c].

Nous nous focalisons dans ce rapport sur le guide de spécification comportementale. Pour expliciter les principes de la méthodologie, nous utilisons une étude de cas présentée dans le paragraphe suivant.

2.1 Étude de cas : l'agence de voyages électronique

Cette application est une de celles retenues par l'organisme de standardisation des systèmes à agents, FIPA [FIP 97]. Dans cet exemple, les clients représentés par leur assistant personnel achètent des voyages auprès d'une agence appelée « travel broker ». Celle-ci est en contact avec des compagnies de voyage, des hôtels, des agences de location de voiture etc. Il agit donc en tant qu'intermédiaire entre l'assistant et les différents prestataires de service.

3 Le guide de spécification comportementale

La spécification comportementale est le modèle de l'application indépendant de toute plate-forme, c'est-à-dire qu'elle permet de décrire le comportement d'un système indépendamment de tout environnement cible d'exécution. Dans une première version de la méthodologie ODAC, nous avons prescrit les règles d'écriture d'une spécification comportementale et à titre d'expérimentation, nous les avons appliquées à la spécification d'un système à base d'agents [GER 01a]. Dans un deuxième temps, l'utilisation de la méthodologie ODAC pour le développement de services dans le contexte des réseaux actifs nous a permis de raffiner les règles d'écriture d'une spécification comportementale [BOU 01].

Ainsi, pour qu'un modélisateur puisse décrire le comportement d'un service (ou application) en cours de développement, nous lui fournissons un **Guide de spécification comportementale** sur lequel il va s'appuyer pour réaliser sa spécification comportementale [CHI 01]. Ce guide est construit comme un profil UML. Un **profil** est une modification ou une extension du langage UML qui soit utilisable dans un domaine applicatif, qui n'est pas prévu par le standard. C'est un ensemble de stéréotypes, de tagged-value, de contraintes et de nouveaux symboles de notations [OMG 00].

Le profil que nous définissons est en fait, composé de trois Profils, chacun correspondant à l'un des points de vue ODP : Entreprise, Information, Traitement. Chaque profil est présenté de la façon suivante :

a) Résumé du Profil : Il contient un résumé de tous les stéréotypes et les *tagged values* qui expriment les concepts utilisés dans la spécification comportementale.

b) Règles de bon format (*Well Formedness Rules*): Elles contiennent les contraintes qui vérifient la consistance de la spécification réalisée à partir de ce profil. Parmi ces règles, nous trouvons les règles de correspondances ODP qui seront discutées à part dans le paragraphe 3.4.

Les règles d'application du profil :

c) Etapes de spécification : Cette partie centrale contient un ensemble de règles, qui appliquent les prescriptions du profil. Le concepteur doit suivre ces étapes pour effectuer la spécification comportementale.

Nous présentons ci-après chacun des trois profils qui ensemble forment le guide de spécification comportementale.

Il est bien évident que chacun des profils doit être cohérent avec les autres : c'est le même système qui est décrit, selon des perceptions distinctes. Il est donc nécessaire que la description du système selon un point de vue soit cohérente avec celle d'un autre point de vue. Par conséquent, le guide de spécification inclut également des règles de correspondance entre les points de vue que nous détaillons à la suite de la description des trois profils.

3.1 Profil pour la spécification du point de vue Entreprise et ses règles d'application

Une spécification d'entreprise d'un système décrit le comportement du système dans l'environnement avec lequel il interagit. C'est une abstraction du système et de l'environnement dans lequel ce système existe. Elle décrit les aspects pertinents de ce que le système est censé réaliser dans le contexte de l'objectif, de la portée et des politiques de son environnement. Elle est axée sur les rôles, les objectifs, de domaine d'application et les politiques du système [ISO 00].

3.1.1 Résumé du profil

Pour la spécification du point de vue Entreprise, nous proposons un profil qui permet de représenter les concepts du point de vue Entreprise avec les concepts d'extension UML. Pour cela nous avons défini un seul stéréotype de classe dans le tableau suivant :

Nom du stéréotype	Concept de Base UML	Description
«role type»	Classe	Décrit le type de rôle (et le comportement associé) à partir duquel est instancié le rôle de l'objet d'entreprise

3.1.2 Règles de bon format

Dans ce profil, les seules règles du bon format de la spécification sont les règles de correspondances du point de vue Entreprise avec les autres points de vue. Ces règles seront décrites dans le paragraphe 3.4.

3.1.3 Étapes de spécification du point de vue Entreprise

ODAC définit le processus de rédaction d'une spécification d'Entreprise comme étant composé de différentes étapes. Celles-ci constituent un ensemble de règles décrivant la façon d'appliquer le profil pour rédiger une spécification entreprise [GER 01b] :

Étape 1) Fixer l'objectif du système

Étape 2) Énumérer les types de rôles et leurs comportements associés permettant d'atteindre cet objectif. Pour cela, raffiner l'objectif de façon itérative jusqu'à identifier les objectifs élémentaires, c'est-à-dire que la décomposition n'est plus pertinente. Assigner alors un type de rôle à chaque objectif élémentaire [DAU 00]. Un rôle étant

l'identifiant d'un comportement, i.e. d'une collection d'actions avec leurs contraintes d'occurrence, lister ces actions en distinguant les actions internes des interactions. Identifier les contraintes d'occurrence, s'il y en a.

Étape 3) Parmi les rôles, distinguer les rôles du système des rôles de l'environnement du système. Ceci revient à identifier la <S>-Communauté.

Étape 4) Parmi les rôles du système, identifier les éventuels rôles d'interfaces, c'est-à-dire d'autres communautés. Assigner à ces communautés les rôles qui s'y rattachent.

Pour chaque communauté :

Étape 5) Identifier les objets d'entreprise qui remplissent les rôles de la communauté. Cette étape est en fait souvent omise car à ce stade de l'analyse, le modélisateur traite plutôt des rôles que des instances.

Étape 6) Décrire le comportement de la communauté qui est l'ensemble des actions de chaque rôle de la communauté avec leurs contraintes d'occurrence.

Étape 7) Décrire les politiques.

Chaque étape sera décrite à travers la spécification d'entreprise du service « Travel Agency ». Les différentes figures représentées sont réalisées avec l'outil Rational Rose©.

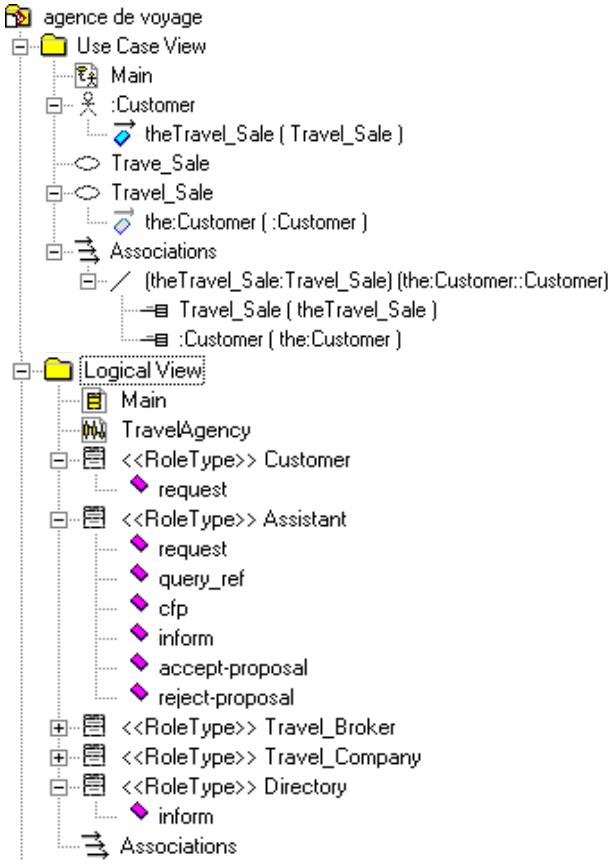

Figure 2 *Utilisation du profil Entreprise pour le service « Travel Agency » avec Rational Rose©*

3.1.4 Exemple : spécification d'entreprise du service « Travel Agency »

Nous décrivons les différentes étapes suivies par un modélisateur utilisant le guide de spécification comportementale pour spécifier une application de « Travel Agency ».

Étape 1 : Fixer l'objectif du service

L'objectif de ce service est de construire un service « Travel Agency », qui vend des voyages. Il est représenté par un **Use Case** (une ellipse UML) (Figure 3).

Figure 3. *Objectif du service « Travel Agency »*

Étape 2 : Énumérer les types de rôle

Les types de rôle permettent d'instancier les rôles qui seront joués par les objets d'entreprise. Un rôle identifie un comportement exhibé par un objet d'entreprise. Le comportement est une collection d'actions internes et d'interactions et les contraintes associées sur leur apparition.

Le modélisateur utilise pour représenter le type de rôle et son comportement associé le stéréotype de classe « Role Type » fourni par le guide de spécification comportementale. Pour distinguer les actions internes des interactions, les noms des actions internes sont soulignés. Les contraintes sur les actions/interactions sont représentées par des notes.

Les types de rôles que le modélisateur identifie sont (Figure 4) :

- Customer : le client qui demande un voyage ;
- Assistant : l'assistant de voyage qui traite la requête du client et achète le voyage pour lui. Pour cela, il cherche une liste de courtiers dans un annuaire, diffuse la requête et négocie les propositions conduisant à une réservation ;
- Travel_Broker : le courtier qui agit comme intermédiaire entre l'assistant et les prestataires de services. Il diffuse la demande de l'assistant à ces prestataires et redirige la meilleure offre à l'assistant ;
- Travel_Company : les prestataires de services qui proposent des voyages au courtier qui répondent à sa requête ;
- Directory : l'annuaire, service de pages jaunes, qui contient des listes de courtiers.

Les interactions de chaque type de rôle correspondent aux actes de communication FIPA [FIP 98].

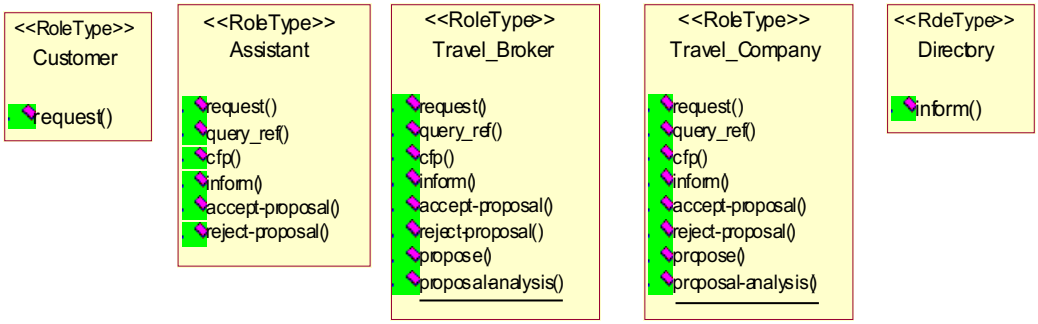

Figure 4. *Les types de rôle du service « Travel Agency »*

Étape 3 : Identifier la S-Communauté

La S-communauté représente le système comme un unique objet d'entreprise interagissant avec son environnement. Elle sépare les types de rôles qui appartiennent à l'environnement du système de ceux qui sont intrinsèques au système. Elle est représentée par un diagramme de Use Case. Dans cette notation, les types de rôles de l'environnement sont les acteurs UML et les Use Case représentent l'objectif du système.

Pour le service « Travel Agency », parmi les types de rôles énumérés ci-dessus, le modélisateur identifie que Customer est un type de rôle de l'environnement alors que les autres sont dans le système « Travel Agency » (Figure 5).

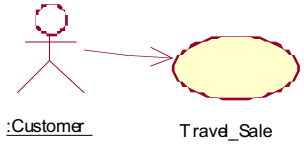

Figure 5. *La <S>-Communauté du service « Travel Agency »*

Étape 4 : Identifier les rôles d'interface des communautés

Il n'existe qu'une seule communauté dans le système « Travel Agency ». Il n'y a donc pas de rôles d'interface à identifier entre plusieurs communautés.

Étape 5 : Identifier les objets d'entreprise

Un objet d'entreprise est représenté par un objet UML. En général, une spécification d'entreprise ne manipule pas des instances d'objet, mais plutôt des types ou des gabarits. La notation d'objet anonyme offre cette facilité et peut donc être employée. Ainsi toutes les notations suivantes peuvent figurer dans une spécification d'entreprise :

- Un objet d'identité O1 instancié à partir du gabarit B sera noté O1 :B
- Un objet d'identité O2 qui joue le rôle A, c'est-à-dire qu'il réalise toutes les actions du comportement du rôle, sera noté O2 :A (puisqu'un rôle peut être vu comme un type d'objet).
- Un objet anonyme qui remplit le rôle A sera noté :A.

Supposons pour notre exemple que le modélisateur de l'application choisit d'utiliser des objets anonymes de façon à ne pas figer le système dans une configuration donnée (Figure 6). Ceci permet de ne pas imposer a priori le nombre d'objets d'Entreprise jouant le rôle de Travel_Broker ou de Travel_Company par exemple. De cette façon, l'assistant peut s'adresser à plusieurs objets d'entreprise jouant le rôle de courtier et choisira une réservation en fonction des réponses qu'il aura reçues. Le même schéma s'applique entre les courtiers et les prestataires de service.

D'autre part, nous supposons également que le modélisateur choisit que chaque objet d'Entreprise joue un (et un seul) rôle. Ce choix de modélisation permet de simplifier la présentation de l'exemple et n'est en aucun cas une restriction du guide de spécification comportementale. Rien ne s'oppose à ce que le même objet d'Entreprise remplisse plusieurs rôles.

Étape 6 : Décrire le comportement de la communauté

La description d'une communauté peut s'attacher à son aspect statique, i.e., la description structurelle de sa population ou son aspect dynamique, i.e., les schémas d'interaction entre les objets d'entreprise. L'aspect dynamique est privilégié dans la spécification d'entreprise, qui se focalise sur la description du comportement de la communauté, sous la forme d'un diagramme de séquences. Ce dernier exprime le déroulement temporel des interactions ODP entre les objets d'entreprise. Dans ce cas, une interaction n'est pas le synonyme d'une invocation de méthode, mais elle peut être vue comme une interaction de haut niveau (ou protocole d'interaction), qui donnera lieu à un ensemble d'actions de traitement (opérations, flux, signaux). Les flèches utilisées dans ce diagramme ont une seule signification : identifier la direction de l'interaction.

Dans un souci de lisibilité, nous illustrons Figure 6 le comportement de la communauté en nous restreignant aux cas positifs. Cependant, chaque objet d'Entreprise est capable de réaliser les différents protocoles décrits dans [FIP 98].

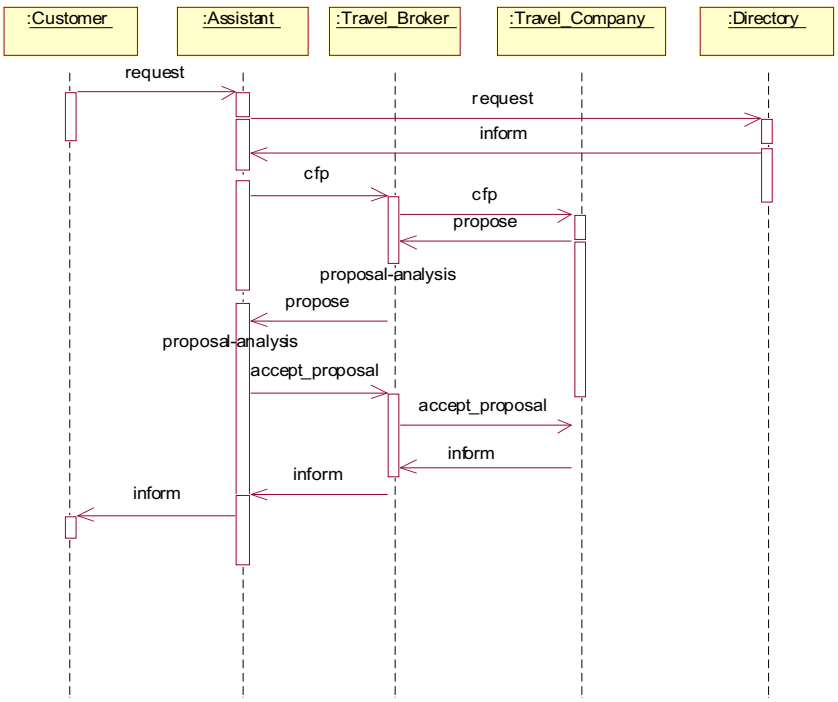

Figure 6. *Comportement de la communauté du service « Travel Agency »*

Étape 7: Les politiques

Les politiques sont des contraintes ou des règles qui s'appliquent aux rôles, objets et communautés. Une politique peut être exprimée comme une obligation, une autorisation, une permission ou une interdiction.

Les politiques sont représentées à l'aide de notes UML affectées aux éléments sujets à ces politiques tels que la communauté, les rôles ou les objets. Les notes UML contiennent des expressions OCL (*Object Constraint Language*) qui expriment la sémantique des politiques utilisées [OMG 97]. Les expressions sont présentées sous formes d'invariants, de pré conditions, de post conditions, ou de *guards*.

Nous représentons Figure 7 un exemple de politique qui est une politique de population, c'est-à-dire relative au nombre d'objets pouvant remplir un rôle. Ainsi le modélisateur exprime le fait que plusieurs objets peuvent remplir le rôle Travel_Broker.

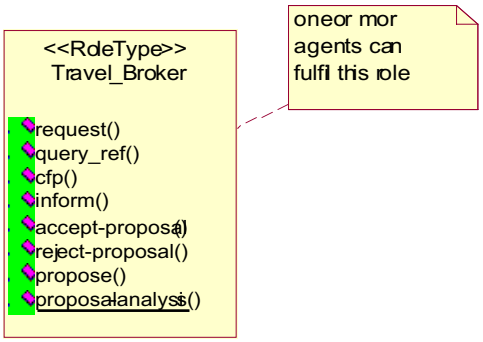

Figure 7. *Exemple d'une politique pour le service « Travel Agency »*

3.2 Profil pour la spécification du point de vue Information et ses règles d'application

Une spécification d'information d'un service définit la sémantique de l'information et la sémantique du traitement de l'information de ce service et inclut les objets d'information et les trois schémas définis dans ce point de vue, à savoir le schéma d'invariant, le schéma statique et le schéma dynamique.

3.2.1 Résumé du profil

Pour la spécification des différents schémas du point de vue Information, nous proposons un profil qui permet de représenter les concepts du point de vue Information ODP avec les concepts d'extension UML, qui sont résumés dans le tableau suivant :

Nom du stéréotype	Concept de Base UML	Description
Invariant schema	Diagramme de classes	Décrit un schéma d'invariant ODP
Dynamic schema	Diagramme d'états transitions	Décrit un schéma dynamique ODP
Static schema	Diagramme de classes	Décrit un schéma statique ODP

3.2.2 Règles de bon format

Dans ce profil, les seules règles de bon format de la spécification concernent les règles de correspondances ODP, décrites dans le paragraphe 3.4.

3.2.3 Étapes de spécification du point de vue Information

Nous proposons au modélisateur d'appliquer le profil pour spécifier le système selon le point de vue Information selon quatre étapes [GER 01b] :

Étape 1) Identifier les objets d'information.

Étape 2) Définir le schéma d'invariants.

Étape 3) Définir le schéma statique.

Étape 4) Définir le schéma dynamique.

Dans la suite, nous décrivons chaque étape en donnant un exemple avec la spécification d'information du service « Travel Agency ». Les différentes figures représentées sont réalisées avec l'outil Rational Rose©.

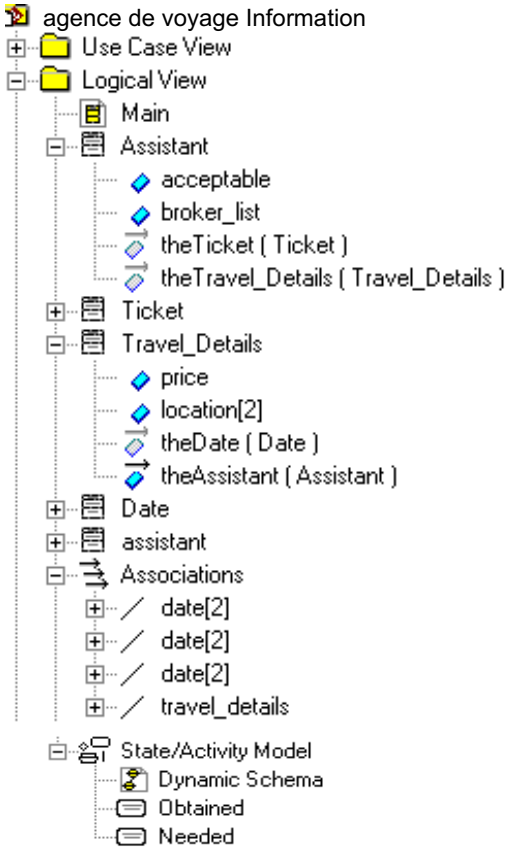

Figure 8. *Utilisation du Profil Information pour le service « Travel Agency » avec Rational Rose*

3.2.4 Exemple : spécification d'information du service « Travel Agency »

Par souci de simplification, nous ne fournissons pas tous les diagrammes qui constituent la spécification d'information, seuls ceux relatifs à l'assistant sont mentionnés.

Étape 1 : Les objets d'information :

L'identification des objets d'information est spécifique au système modélisé. Nous supposons dans le service « Travel Agency » que l'objet d'entreprise Assistant correspond aux objets d'information suivants :

- L'assistant
- Le ticket
- Le détail du voyage
- Les dates

Étape 2 : Le schéma d'invariants

Le schéma d'invariants est un ensemble de prédicats qui doivent toujours être vrais sur un ensemble d'objets d'information. Il est représenté par un diagramme de classe dans lequel ne figurent pas les opérations sur ces objets. L'objet d'information est représenté par une classe UML. Le compartiment des attributs servira pour spécifier les attributs de cet objet.

La figure 9 illustre un extrait du schéma d'invariants relatif à l'assistant du service « Travel Agency ».

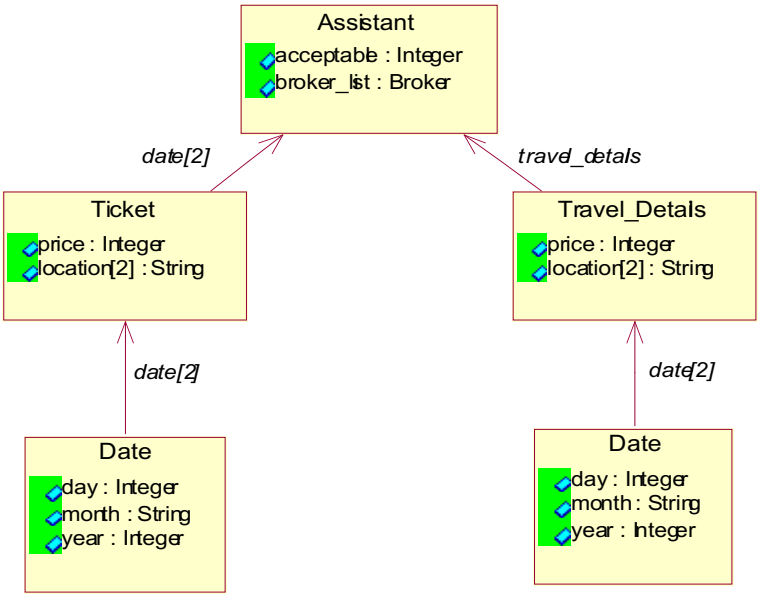

Figure 9. *Un exemple de schéma d'invariants : extrait du service « Travel Agency »*

Étape 3 : Le schéma statique

Le schéma statique spécifie l'état d'un ou plusieurs objets d'information à un instant donné dans le temps. Il permet de spécifier pour ces objets des états dans lesquels il faut forcer l'objet, distincts des états usuels apparaissant en fonctionnement normal du système. Ce schéma peut représenter une photographie instantanée d'un état particulier du système. Il est représenté par un diagramme de classe qui montre le détail de l'état de ces objets d'information sans représenter le changement sur l'état. Pour cela nous utilisons une classe qui représente l'objet d'information et dans son

compartiment « attributs », nous introduisons un attribut état qui prend l'état de cet objet d'information.

Les informations du schéma statique seront utiles dans les traitements correspondants à ces états particuliers, par exemple un état d'exception.

Dans le service « Travel Agency », nous distinguons un seul état particulier qui est l'état d'initialisation du service, par exemple de l'assistant (Figure 10).

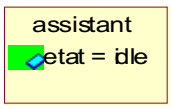

Figure 10. *Un exemple de schéma statique : extrait du service « Travel Agency »*

Étape 4 : Le schéma dynamique

Le schéma dynamique spécifie les changements d'états autorisés en fonctionnement normal pour un ou plusieurs objets d'information. Contrairement au schéma statique il n'y a pas besoin de forcer l'objet à passer dans ces états et à subir ces changements. Le schéma dynamique est représenté par un diagramme UML d'état transition (*Statechart diagram*), qui reflète le changement sur les états des objets d'informations, sans montrer le détail de ces états.

Dans le service « Travel Agency », le schéma dynamique se compose de plusieurs diagrammes. La figure 11 illustre le diagramme relatif à aux changements d'états du ticket suite à l'action « request » de l'assistant.

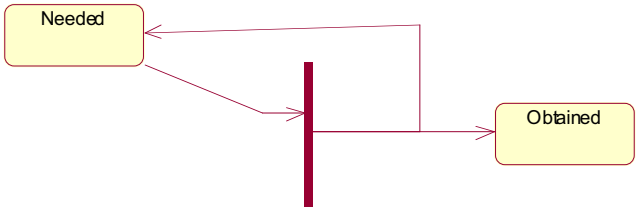

Figure 11. *Un exemple de schéma dynamique : extrait du service « Travel Agency »*

3.3 Profil pour la spécification du point de vue Traitement et ses règles d'application

La spécification de traitement permet d'expliciter comment le système décrit dans les points de vue entreprise et information réalise la fonctionnalité attendue, i.e., remplit son objectif. Elle adresse donc les aspects fonctionnels du système en se focalisant sur ses interactions.

3.3.1 Résumé du profil

Ce Profil donne une représentation des concepts ODP du point de vue Traitement. Les concepts d'extensions utilisés (Stéréotypes et *Tagged Value*) sont résumés ci-après :

Nom du stéréotype	Concept de Base UML	Description
ComputationalObject	Class	Décrit un objet de traitement
BindingObject	Class	Décrit un objet de liaison
SignalInterface	Class	Décrit une interface Signal ODP et le détail de classe représente sa signature Il diffère d'une interface UML
StreamInterface	Class	Idem
OperationInterface	Class	Idem
Announcement	Opération	Précède la signature d'une interaction de type annonce dans une interface opération
Interrogation	Opération	Précède signature d'une interaction de type interrogation dans une interface opération
Initiator	Opération	Précède la signature d'une interaction de type signal envoyé dans une interface signal
Responder	Opération	Précède la signature d'une interaction de type signal reçu dans une interface signal
Producer	Opération	Précède la signature d'une interaction de type flux envoyé dans une interface flux
Consumer	Opération	Précède la signature d'une interaction de type flux reçu dans une interface flux
Offer	Directed Association	La classe objet de traitement ou objet de liaison offre l'interface, la direction de l'association est de la classe source objet vers la classe cible interface X.
PrimitiveBinding	Non Directed Association	Associe deux classes interface qui ont le même type et des signatures complémentaires.
InternalActionState	ActionState	Etat d'action Interne dans le diagramme d'activité
InteractionState	ActionState	Etat d'interaction dans le diagramme d'activité

T*agged values* : dans ce profil nous avons défini une tagged-value «Causality » dans la classe interface opération (compartiment nom de la classe) pour exprimer la causalité.

Les signatures des opérations dans l'interface représentées par les méthodes de la classe peuvent être associées avec des tagged-values, pour exprimer les paramètres des opérations comme les contraintes de qualité de service.

3.3.2 Règles de bon format

Pour la cohérence de la spécification Traitement, nous avons imposé les contraintes suivantes :

L'association stéréotypé *PrimitiveBinding* doit relier deux classes qui ont le même stéréotype parmi les stéréotypes (*Operation Interface*, *Signal Interface*, *Stream Interface*) avec des signatures complémentaires (voir l'exemple du service « Travel Agency »).

Les autres contraintes concernent les règles de correspondances entre le point de vue Traitement et les points de vue Information et Entreprise et seront discutées dans le paragraphe 3.4.

3.3.3 Étapes de spécification du point de vue Traitement

Pour l'élaboration de cette spécification, le concepteur va se baser sur le profil défini ci-dessus et sur les étapes suivantes :

Étape 1) Identification des objets de traitement qui interagissent durant la vie du système.

Étape 2) Identification des interactions de Traitement.

Étape 3) Spécification des liaisons et des signatures des interfaces.

Étape 4) Spécification du comportement des objets de traitement.

Figure 12. *Utilisation du Profil Traitement pour le service « Travel Agency » avec Rational Rose*

3.3.4 Exemple : spécification de traitement du service « Travel Agency »

Étape 1 : Identification des objets de traitement qui interagissent durant la vie du système

Dans cette étape, le modélisateur identifie les différents objets de traitement qui composent le système. Ces objets sont en relation étroite avec les objets d'entreprise, identifiés dans le point de vue Entreprise. Cette correspondance est réalisée à l'aide de la règle ODP suivante : «Un objet d'entreprise peut correspondre à un ou plusieurs objets de traitement » (voir règles de correspondances paragraphe 3.4).

Dans le service « Travel Agency », nous supposons que le modélisateur a choisi que chaque objet d'entreprise corresponde à un objet de Traitement. Nous limitons ici l'exemple à l'objet Assistant.

Étape 2 : Identification des interactions de Traitement

Pour chaque objet de traitement identifié dans l'étape précédente, le modélisateur identifie dans cette étape les différentes interactions. Il existe trois catégories d'interactions, ayant une sémantique distincte.

- L'opération est une interaction de sémantique client-serveur ;
- Le flux est une interaction de sémantique producteur-consommateur ;
- Le signal est une interaction de sémantique initiateur-répondeur.

Selon la sémantique des interactions du point de vue Entreprise et des transitions du point de vue Information, le modélisateur choisit l'une ou l'autre de ces catégories d'interactions du point de vue Traitement pour faire la mise en correspondance.

Dans la spécification du service « Travel Agency » restreinte à l'assistant, nous supposons que le modélisateur a fait les choix suivants. Selon la spécification d'entreprise, l'assistant interagit avec le courtier pour répondre au customer en lui fournissant le voyage demandé. Cette interaction est de type client-serveur : l'assistant utilise le service de proposition de voyage offert par le courtier (« cfp » puis « propose ») ainsi que le service de réservation de voyage (« accept_proposal » et « inform »). Le modélisateur choisit alors de représenter ces deux services par deux opérations qui sont des interrogations, nommées respectivement Broker_Proposal et Broker_Reservation.

Étape 3 : Spécification des liaisons et des signatures des interfaces

Les interactions entre les objets se produisent sur des interfaces reliées entre elles avec des liaisons. À chaque catégorie d'interaction correspond une catégorie d'interfaces. Il existe donc trois catégories d'interfaces, interface opération, interface flux et interface signal. Chaque interface est identifiée par un nom unique, et sa signature décrit les interactions de l'objet qui l'offre (avec lequel elle est associée).

Les liaisons mettent en relation des interfaces. Une liaison peut être simple (directe entre deux objets de traitement) réalisée avec des actions primitives de liaison, comme elle peut être composée (entre plusieurs objets de traitement), réalisée avec des objets de liaison. Les objets de liaison sont à la base des objets de traitement ordinaires. Le contrôle de la qualité de service compte parmi les utilisations des objets de liaisons.

Pour illustrer les interactions entre les différents objets de traitement et décrire les interfaces et leurs signatures associées, le modélisateur utilise un diagramme de classe avec les stéréotypes définis

dans le profil. Dans ce diagramme figure également la liaison entre les interfaces des objets de traitement, ainsi que la causalité des interfaces (par exemple, causalité client, ou causalité serveur, etc).

Selon que la liaison est simple ou composite, l'application du profil pour chaque type d'interface diffère.

Liaison simple – interface opération

L'objet de traitement qui offre une interface opération avec la causalité *Client*, invoque une opération nommée dans la signature de son interface. Un objet qui offre une interface opération avec la causalité serveur attend une des opérations nommées dans sa signature d'interface.

Une opération peut être une annonce (une invocation) ou une interrogation (une invocation suivie de terminaisons qui représentent la réponse du serveur au client). Dans le cas d'une interrogation, le serveur répond à l'invocation en initialisant les terminaisons associées à cette invocation et qui sont nommées dans la signature de son interface. Pour cette interrogation, le client attend ces terminaisons nommées dans son interface.

Le modélisateur utilise un diagramme de classes pour décrire les interactions opération et leurs interfaces opération associées. La signature des interfaces est décrite à l'intérieur de l'interface opération.

La classe stéréotypée **«OperationInterface»** représente l'interface opération. Elle est liée avec la classe objet de traitement par une association stéréotypée **«Offer »**, qui signifie que l'objet de traitement offre une interface opération à travers laquelle se produisent des opérations.

Dans le compartiment « méthodes » de la classe interface le modélisateur décrit chaque interaction opération, où le nom de l'opération ODP est le nom de la méthode. Les paramètres de la méthode représentent les paramètres de l'opération ODP. Chaque opération est précédée par un stéréotype qui précise l'opération (annonce ou interrogation). Dans le cas d'une annonce, elle est précédée par le stéréotype **« Announcement »** suivi du mot clé **in.** Dans le cas d'une interrogation, elle est précédée par le stéréotype **« Interrogation »** suivi par les mots clés **inout** ou **out** .

Dans une interface opération, la causalité ne porte pas sur chaque opération mais sur la globalité de l'interface. Pour décrire cette causalité, le modélisateur utilise la tagged-value **{Causality}** dans le compartiment nom de la classe Interface opération et qui prend la valeur « Client » ou « Server ».

L'association stéréotypée **« PrimitiveBinding »** entre deux classes interface opération représente la liaison simple. La figure 13 montre un exemple d'interface opération.

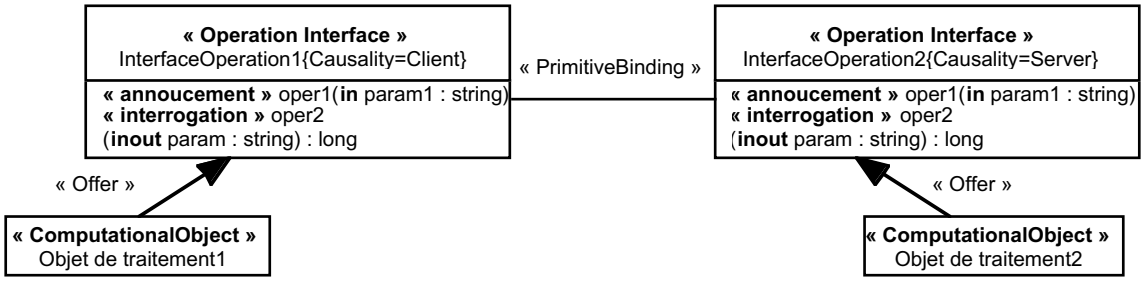

Figure 13. *Exemple d'une interface opération*

Dans le service « Travel Agency » restreint à l'Assistant, les deux interrogations identifiées à l'étape 2 sont placées par le modélisateur dans une interface opération appelée Interface Negotiation. La figure 14 illustre le diagramme correspondant.

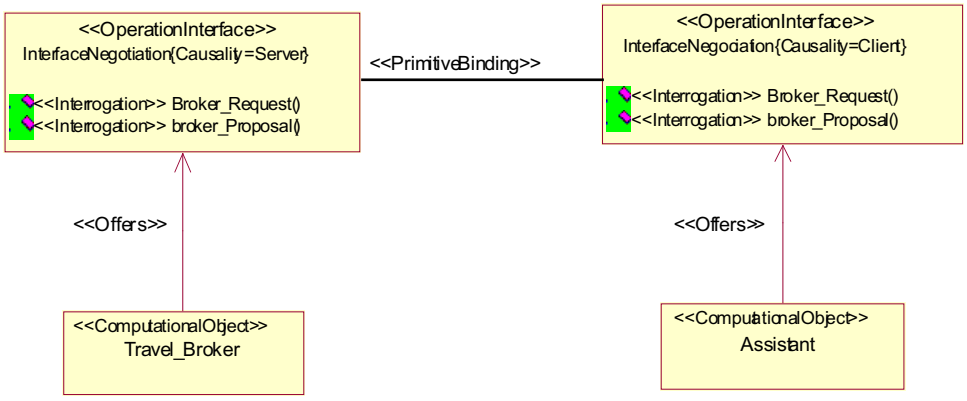

Figure 14. *Exemple des interfaces offertes dans le service « Travel Agency »*

Depuis un tel diagramme, la génération de la signature de l'interface opération dans d'autres langages est relativement aisée. Par exemple en OMG-IDL, pour une opération de type interrogation qui comporte deux parties, l'invocation se compose de mots clé ***in*** (précède un paramètre de l'invocation) et la partie terminaison se compose de mots clé ***out*** (précède un paramètre de la terminaison), ***raise*** (pour une exception), et du résultat de retour de l'invocation. Quand il s'agit d'une annonce, son écriture en IDL comporte seulement des mots clé ***in***. Les outils commerciaux d'UML comme Rational Rose peuvent être facilement étendus pour générer ces signatures IDL. La figure 15 donne l'exemple de l'écriture IDL des interfaces de la figure 14.

```
interface negotiation {
void Broker_Proposal {
       in int Travel_Details.price,
       in string Travel_Details.location,
       out int Ticket.price,
       out string Ticket.location
};

void Broker_Reservation {
       in int Ticket.price,
       in string Ticket.location,
       out boolean ok
};
};
```

Figure 15. *Exemple de l'écriture IDL des interfaces de la Figure 14*

Liaison simple – interface signal

L'objet de traitement produit des signaux qui ont la causalité Initiateur dans son interface signal et reçoit les signaux qui ont la causalité Répondeur dans son interface signal.

Les interfaces signal offert par les objets de traitement, ainsi que les actions primitives de liaison qui relient deux interfaces signal (respectivement les objets qui les offrent), sont représentées par des classes stéréotypées.

La classe stéréotypée **« SignalInterface »**, qui représente l'interface signal, est liée avec la classe objet de traitement par une association stéréotypée **« Offer ».** Celle-ci signifie que l'objet de traitement offre une interface signal, à travers laquelle se produisent un ensemble de signaux (interactions).

La signature des interfaces est décrite à l'intérieur de la classe stéréotypée. Dans le compartiment opération de la classe, chaque interaction signal est décrite de la façon suivante : le nom de la méthode est le nom du signal, les paramètres de la méthode représentent les paramètres du signal, sachant qu'ici les paramètres sont toujours précédés par le mot clé **in** pour éviter que son absence soit interprétée comme un **inout**. En effet, un signal ne peut pas avoir de paramètres résultat, donc le mot clé **out** ne peut jamais être accolé à un paramètre de signal. La signature d'un signal sera toujours précédée par l'un des stéréotypes **«Initiator»** ou **«Responder»** pour exprimer la causalité du signal dans l'interface.

La liaison simple entre deux classes Interfaces est représenté avec l'association stéréotypée **« PrimitiveBinding »**.

La figure 16 illustre un exemple de diagramme d'interface signal.

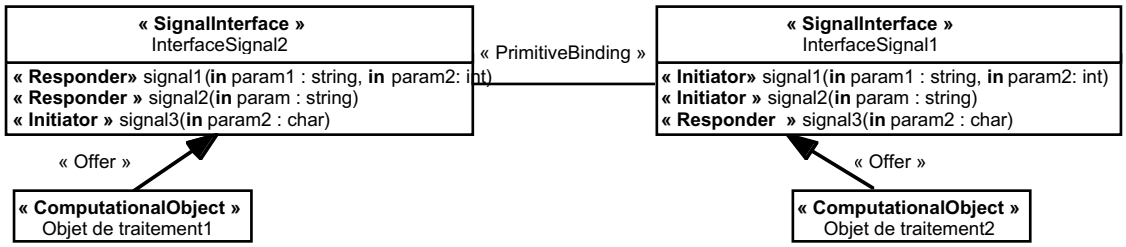

Figure 16. *Exemple d'une interface signal*

Il n'y a pas d'interface signal dans le service « Travel Agency ».

À partir de la signature de l'interface signal représentée dans le diagramme précédent, la génération de l'interface dans un langage approprié peut se faire sans difficulté. En IDL, ce n'est pas possible car les versions récentes de OMG-IDL ne permettent pas l'écriture des signatures de signaux. Par contre, dans des langages de type ODL, de TINA, qui est une extension d'IDL pour prendre en compte des interactions autres que les opérations, il est possible de décrire les différents signaux dans une interface [ODL 96]. Dans ODL, les signaux sont des actions unidirectionnelles qui ne retournent pas de résultat (void), les paramètres sont **in** (pas de retour de paramètres) et le nom de l'action est précédé par le mot clé **oneway void**. Nous présentons ci-dessous l'écriture de l'interface flux en ODL correspondant à la figure 16.

```
interface InterfaceSignal {
// déclarations de types :optionnel

...
//déclaration des signaux qui ont la causalité initiateur
void oneway  signal1 (in string param1, in int param2, …);
void oneway signal2 (in string param);
//déclaration des signaux qui ont la causalité répondeur
void oneway  signal3 (in char param1,…);
}; // Fin de InterfaceSignal
```

Liaison simple – interfacæ flux

L'objet de traitement qui envoie un flux de donnée offre une interface flux dans laquelle l'action est décrite avec la causalité *Producteur*. Celui qui reçoit un flux aura dans son interface flux l'action décrite avec la causalité *Consommateur*.

La classe stéréotypée **«StreamInterface»** représente l'interface flux. Elle est liée avec la classe objet de traitement par une association stéréotypée **« Offer »** qui signifie que l'objet de traitement offre une interface Flux à travers laquelle se produisent les flux.

La signature des interfaces est décrite à l'intérieur de la classe stéréotypée. Dans le compartiment opération, chaque flux est décrit par une méthode sans paramètres et qui a comme type de retour le type de flux. Le nom de la méthode est le nom du flux, qui est unique dans l'interface. La signature de chaque flux sera toujours précédée par l'un des stéréotypes « Producer » ou « Consumer » pour exprimer la causalité du flux dans l'interface.

L'association stéréotypée **« PrimitiveBinding »** entre deux classes Interfaces représente la liaison simple entre deux interfaces.

La figure 17 illustre un exemple de diagramme d'interface flux.

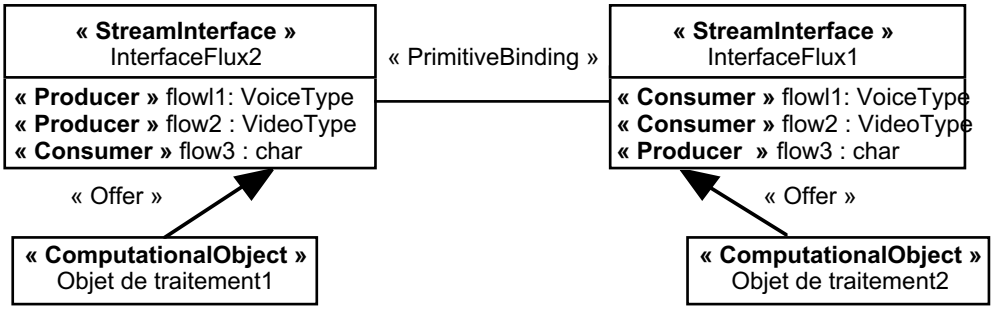

Figure 17. *Exemple d'une interface flux*

Il n'y a pas d'interface flux dans le service « Travel Agency ».

À partir de la signature de l'interface flux représentée dans le diagramme précédent, de la même façon que pour les interfaces signal, la génération de la signature de l'interface dans un langage de type ODL peut se faire sans difficulté. Nous présentons ci-dessous l'écriture de l'interface flux en ODL correspondant à la figure 17 (où le mot clé **sink** précède la signature d'un flux qui a la causalité *consommateur*, et le mot clé **source** précède la signature d'un flux qui a la causalité *producteur*).

```
interface InterfaceFlux1 {
// déclarations de types :optionnel

//déclaration des signaux qui ont la causalité consommateur
sink VoiceType  flow1 ;
sink VideoType flow2

//déclaration des signaux qui ont la causalité producteur
source Char flow3;

}; // Fin de InterfaceFlux1
```

Liaison composite

Si des interactions nécessitent de lier plusieurs interfaces ensemble ou de contrôler l'interaction entre plusieurs objets de traitement à la fois, alors il faut utiliser un objet de liaison pour supporter cette liaison complexe.

Cet objet est, à la base, un objet de traitement qui doit comporter toutes les interfaces correspondant aux interfaces des objets de traitement participant à l'interaction. Les signatures des interfaces de cet objet de liaison doivent être complémentaires aux signatures des interfaces associées aux objets de traitement correspondants. Cela n'exclut pas l'idée qu'un objet de liaison peut comporter d'autres interfaces de types différents, comme les interfaces de contrôle.

L'objet de liaison utilise des actions de liaisons primitives pour lier (faire un mapping) ses interfaces avec les interfaces des objets de traitement correspondants.

Cette liaison entre objets de traitement, qualifiée de liaison composite comporte, avec le mapping d'interfaces pour les objets de traitement, des interfaces de contrôle qui ont des fonctions tels que le contrôle de l'utilisation de la liaison et de changement sur la liaison (autoriser les changements sur la liaison, le changement des membres de cette liaison, changement de la qualité de service de la liaison, supprimer la liaison, et changer le type de communication dans la liaison).

L'application du profil permet de représenter un objet de liaison par une classe stéréotypée **« BindingObject »**. Cette classe est reliée à au moins deux classes interfaces ODP avec l'association stéréotypée **« Offer »**, qui sont eux-mêmes reliés à d'autres classes interfaces ODP avec des associations stéréotypées **« PrimitiveBinding »**. Dans le cas ou l'objet de liaison est associé à des contraintes, celles-ci sont représentées par des notes.

Le diagramme de classes suivant (Figure 18) illustre l'application du profil pour ce type de liaison. Il faut noter que l'objet de liaison peut supporter plusieurs types d'interfaces.

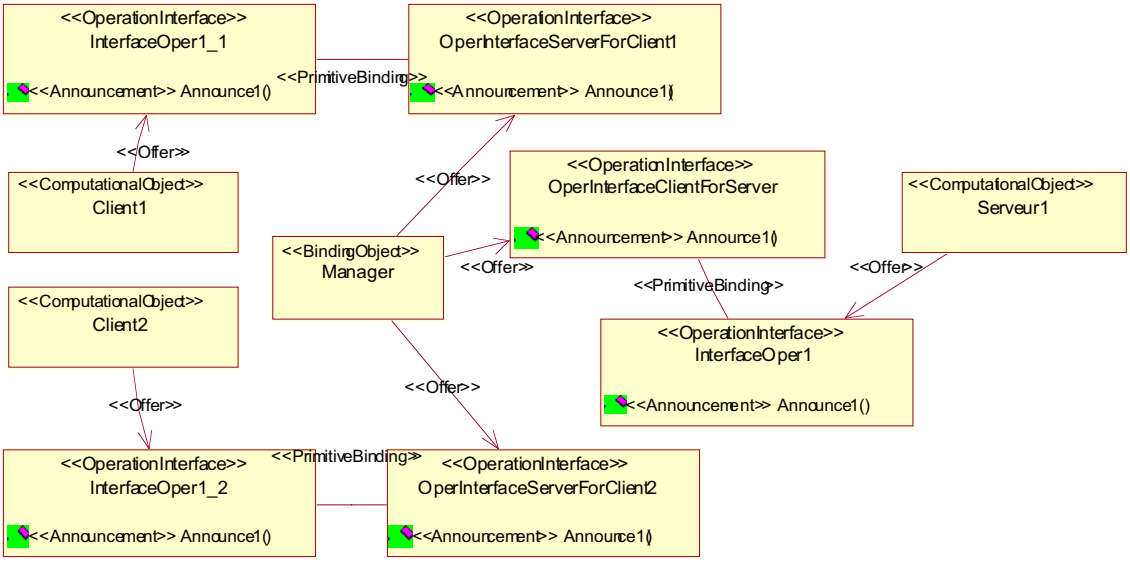

Figure 18. *Exemple d'une liaison composite*

Étape 4 : Spécification du comportement des objets de Traitement

Après avoir identifié les interactions entre les objets de traitement et les interfaces associées, ainsi que leurs liaisons, le modélisateur décrit le comportement des objets de traitement.

L'ensemble des comportements des objets de traitement forme le comportement global du système pour réaliser son objectif. Un comportement est une séquence d'états d'actions séparées par des transitions. Pour représenter ce comportement, le modélisateur utilise un diagramme d'activité UML. Dans ce diagramme, les actions internes des objets de traitement sont représentées par le stéréotype **« InternalActionState »** sur un ActivityState UML (carré oval). Les interactions sont représentées par le stéréotype **« InteractionState »** sur un *ActivityState* UML. Il faut noter que ce diagramme est complémentaire au diagramme précédent des différentes interactions et leurs signatures d'interfaces associées.

Nous illustrons dans la figure 19 à titre d'exemple une partie de la spécification de comportement des objets de traitement du service « Travel Agency », à savoir celle de l'objet Assistant.

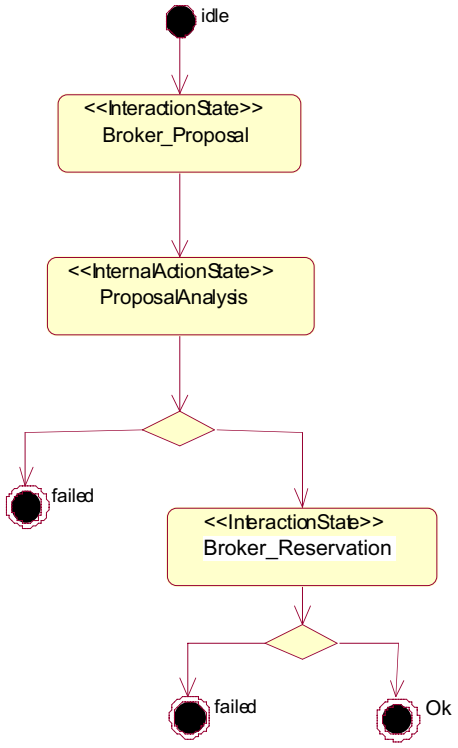

Figure 19. *Diagramme d'activité illustrant le comportement de l'objet Assistant du service « Travel Agency »*

3.4 Les règles de correspondance entre les spécifications

Le guide de spécification comportementale offre donc trois profils permettant à un modélisateur de décrire le système selon trois préoccupations différentes : celles qui relèvent du point de vue Entreprise, celles qui relèvent du point de vue Information et celles qui relèvent du point de vue Traitement.

Ces trois descriptions, ou spécifications, ne doivent pas être contradictoires. Elles sont complémentaires et doivent nécessairement être cohérentes. Le respect de cette cohérence est réalisé à travers l'application des règles de correspondance définies dans la norme ODP. Nous les énonçons ci-après en illustrant sur l'exemple du service « Travel Agency » leur usage.

3.4.1 Correspondance entre les points de vue Entreprise et Information

Règles

- **EI1 :** Un objet d'entreprise et le rôle qu'il assume correspondent à un ou plusieurs objets d'information.
- **EI2 :** Les politiques d'entreprise peuvent correspondre aux invariants du schéma d'invariants. Ces invariants gouvernent les états des objets d'information.
- **EI3 :** Une interaction d'entreprise correspond à une ou plusieurs transitions dans le schéma dynamique.

Exemple du service « Travel Agency »

Dans l'exemple suivant nous montrons l'application de la règle **EI1** sur un l'objet d'entreprise **Source** et son rôle. La correspondance ici s'interprète comme suit : l'objet Assistant utilise pour réaliser son rôle les objets d'information décrit dans la figure 20.

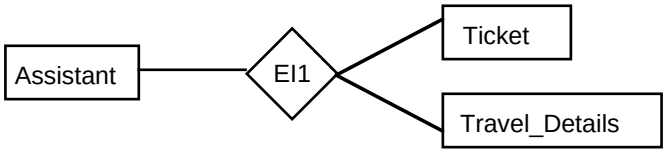

Figure 20. *Exemple de correspondance entre spécifications Entreprise et Information du service « Travel Agency »*

3.4.2 Correspondance entre les points de vue Entreprise et Traitement

Règles

EC1 : Un rôle d'entreprise correspond à un ou plusieurs objets de traitement qui participent à la réalisation de ce rôle.

EC2 : Un objet d'entreprise et le rôle qu'il assume correspondent à un ou plusieurs objets de traitement.

EC3 : Le comportement identifié par un rôle d'entreprise correspond à un ou plusieurs objets de traitement avec leurs interfaces respectives.

EC4 : Une action d'entreprise correspond à un ou plusieurs objets de traitements qui réalisent cette action.

EC5 : Une interaction d'entreprise correspond à une ou plusieurs actions de traitement.

EC6 : Une politique entreprise associé à un rôle retreint les actions des objets de traitement correspondant à ce rôle.

Exemple du service « Travel Agency »

Dans l'exemple suivant nous montrons l'application de la règle **EC5**. Elle s'interprète comme expliqué dans l'étape 2 : Les interactions d'Entreprise « cfp » et « propose » correspondent à l'interrogation Broker_Proposal et les interactions d'Entreprise « accept_proposal » et « inform » correspondent à l'interrogation Broker_Reservation.

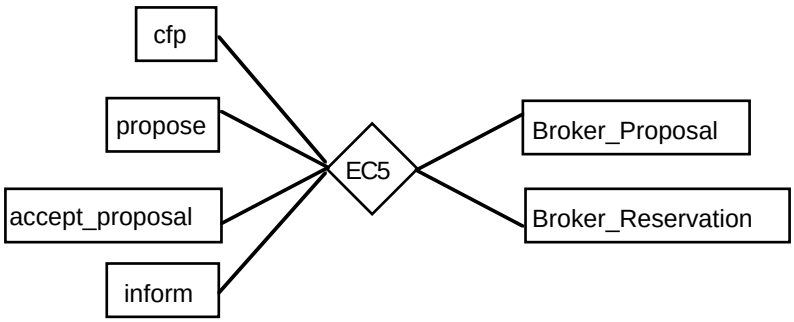

Figure 21. *Exemple de correspondance entre spécifications Entreprise et Traitement du service « Travel Agency »*

3.4.3 Correspondance entre les points de vue Information et Traitement

Règles

- **IC1 :** Le schéma dynamique et le schéma d'invariants correspondent aux comportements des objets de traitements.
- **IC2 :** Un ou plusieurs objets d'information correspondent à un ou plusieurs objets de traitement. **N.B :** quand un objet d'information correspond à un objet de traitement le schéma statique et le schéma d'invariant correspondent aux états d'actions de cet objet de traitement.
- **IC3 :** Une transition d'états dans le schéma dynamique correspond à une ou plusieurs actions des objets de traitement.
- **IC4 :** L'état d'un objet d'information peut correspondre à une ou plusieurs états d'actions de traitement.

Exemple du service « Travel Agency »

Dans l'exemple suivant nous montrons l'application de la règle **IC4**. Les états possibles du ticket, Needed et Obtained, correspondent respectivement à l'interaction de traitement Broker_Proposal de l'objet de traitement Assistant et à l'interaction de traitement Broker_Reservation de ce même objet de traitement Assistant.

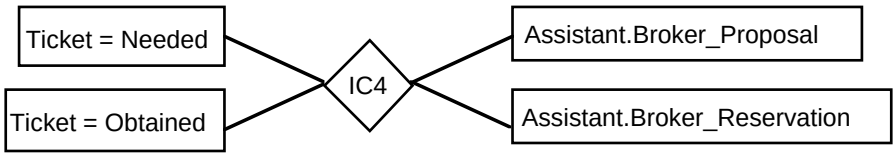

***Figure 22**. Exemple de correspondance entre spécifications Information et Traitement du service « Travel Agency »*

4 La spécification opérationnelle

La spécification opérationnelle résulte de la transformation de la spécification comportementale selon un environnement cible reflétant l'environnement réel d'exécution. Elle dépend de la spécification établie dans le point de vue Ingénierie qui décrit l'environnement d'exécution. Ainsi pour chaque catégorie d'environnements, la méthodologie ODAC fournit au concepteur un guide de spécification d'ingénierie lui permettant de décrire l'environnement considéré. Cette description s'appelle spécification d'ingénierie. Celle-ci inclut des éléments qui nécessitent lors de sa rédaction de la mettre en correspondance avec la spécification comportementale. C'est cette mise en correspondance qui constitue le paramétrage de la spécification comportementale par la spécification d'ingénierie. Elle est réalisée par l'application de règles définies à cet égard et est illustrée par un diagramme de déploiement.

La spécification opérationnelle correspond donc à la spécification comportementale et la spécification d'ingénierie qui inclut le diagramme de déploiement établissant la correspondance entre les deux. C'est donc un modèle de l'application spécifique à une plate-forme [OMG 01].

La création d'un guide de spécification d'ingénierie consiste à établir la correspondance entre les concepts du point de vue ingénierie ODP et ceux de l'environnement considéré, puis à définir un profil UML, c'est-à-dire fournir une notation pour chaque concept. Actuellement, deux guides de spécification d'ingénierie sont proposés : le guide MASIF-DESIGN pour les environnements de

type plates-formes à agents mobiles conformes à la spécification MASIF et le guide ODAC*for*ANTS pour les environnements de type réseaux actifs ANTS [MUS 01][BOU 01].

5 Conclusion

Nous avons présenté dans ce rapport la méthodologie ODAC qui permet à un constructeur de spécifier des applications réparties. Pour cela, la méthodologie offre un ensemble d'outils qui sont les concepts ODP, la notation UML, des étapes et un processus associé à chaque étape. Nous avons focalisé notre description sur le guide de spécification comportementale qui fournit au constructeur un ensemble de recommandations sur l'élaboration d'une spécification comportementale, modèle de l'application indépendant de tout environnement cible d'exécution.

Nous fournissons par ailleurs des guides de spécification d'ingénierie permettant au constructeur de décrire l'environnement considéré pour implanter l'application. Cette description permet alors de paramétrer la spécification comportementale pour la transformer en spécification opérationnelle, modèle de l'application spécifique à un environnement d'exécution, à partir duquel du code peut être généré et déployé dans l'environnement.

La suite de ce travail est d'étudier la transformation de la spécification comportementale en spécification opérationnelle en fonction du paramétrage de la spécification d'ingénierie afin de déterminer le degré d'automatisation de cette transformation.

6 Bibliographie

[BOO 99] G. Booch, J. Rumbaugh and I. Jaccobson, *The Unified Modeling Language User Guide*, Addison Wesley Longman, 1999.

[BOU 01] S. Bouzitouna et al, *Création de services actifs*, A paraître dans les actes du 4ème Colloque francophone sur la gestion de réseau et de service (GRES'01), Marrakech, Décembre 2001

[CHI 01] A. Chibani, *Instrumentation de la méthodologie ODAC pour une spécification comportementale*, Rapport de stage de DEA SIR, Université Paris 6, 2001

[DAU 00] J.F. Dauchez and M.P. Gervais,, *Specifying and Verifying the Behavior of Telecommunications Services*, in Proceedings of the 6th International Conference on Intelligence in Networks (ICIN'2000), Arcachon, France, January 2000

[FIP 97] FIPA, *Specification Part 4 - Application Design Test: Personal Travel Agent*, FIPA97 ver 1.0, June 1997

[FIP 98] FIPA, *Specification Part 2 - Agent Communication Language*, FIPA97 ver 2.0, October 1998

[GER 00] M.P. Gervais, *ODAC : une méthodologie de construction de systèmes à base d'agents fondée sur ODP*, Rapport de recherche, LIP6 2000 / 028, novembre 2000

[GER 01a] M.P. Gervais, *Vers une méthodologie de construction de systèmes à base d'agents*, Actes de l'atelier SMA de la plate-forme AFIA, Grenoble, juin 2001, pp19-27

[GER 01b] M.P. Gervais, *ODAC : An Agent-Oriented Methodology Based on ODP*, soumis in Journal of Autonomous Agents and Multi-Agent Systems, August 2001

[GER 01c] M.P. Gervais, A. Chibani, S. Bouzitouna et M. Elias, *Une méthodologie de spécification de services actifs*, livrable Amarrage-SP4-D4.3.1 du projet RNRT AMARRAGE, Octobre 2001

[ISO 95] ISO/IEC 10746-x. | ITU-T Recommendation X.90x, *Open Distributed Processing – Reference Model – Part x*, 1995

[ISO 00] ISO/IEC CD 15414, *ODP Reference Model : Enterprise Viewpoint*, January 2000

[MUS 01] F. Muscutariu and M.P. Gervais, *On the Modeling of Mobile Agent-Based Systems*, in Proceedings of the 3rd IEEE/ACM International Workshop on Mobile Agents for Telecommunication

Applications (MATA'01), Lecture Notes in Computer Science n°2164, Springer Verlag (Ed), Montreal, August 2001, pp219-234

[OCC 01] M. Occello, J.-L. Koning et C. Baeijs, *Conception de systèmes multi-agents : quelques éléments de réflexion méthodologique*, à paraître dans Technique et science informatique, 2001

[ODL 96] TINA Consortium, *Object Definition Language Manual*, Version 2.3, 22/07/1996

[OMG 97] OMG, OCL version 1.1, September 1997, IBM Press. ad 97/08/08.

[OMG 00] OMG, *Unified Modeling Language Specification 1.3*, TC. Document ad/00-03-01, OMG. 2000. http://www.omg.org

[OMG 01] OMG, *Model Driven Architecture, A Technical Perspective*, Document ab/2001-02-05, February 2001, http://www.omg.org

[PUT 01] J-R. Putman, *Architecting with RM-ODP*, Prentice-Hal, 2001

[ZNA 97] S. Znaty et M.P. Gervais, *Les réseaux intelligents : ingénierie des services de télécommunication*, Editions Hermès, 279 pages, 1997

7 Annexe : ODP : la norme de traitement réparti ouvert

La norme de traitement réparti ouvert (ODP), développée conjointement par l'ISO et l'ITU-T, fournit un modèle de référence qui définit un cadre architectural pour la construction de systèmes et applications réparties. Le modèle de référence ODP (RM-ODP) définit deux modèles : un modèle objet et un modèle architectural que nous présentons ci-après [ISO 95][ZNA 97].

7.1 Le modèle objet

Le modèle objet ODP est très général et introduit un ensemble minimal et générique de concepts de modélisation basée objet. Un objet modélise une entité de l'univers de discours. Il est caractérisé par son identité, son état, son comportement, et ses interfaces. L'identité d'un objet permet de le distinguer de tout autre objet. L'état d'un objet caractérise l'information qu'il détient à un instant donné dans le temps. Son comportement caractérise l'ensemble des changements d'états possibles qui peuvent l'affecter et définit l'ensemble des actions potentielles auxquelles l'objet peut prendre part. Cet ensemble d'actions associées à un objet est divisé en actions internes et externes (interactions). Une action interne se produit sans la participation de l'environnement de l'objet. Une interaction se produit avec la participation de l'environnement de l'objet. Le terme environnement d'un objet désigne tous les objets autres que celui considéré. Un objet étant encapsulé, ses changements d'états ne peuvent résulter que d'une action interne ou d'une interaction avec son environnement. Le modèle objet ne prescrit ni la nature des actions internes de l'objet ni celles de ses interactions avec d'autres objets. Une interaction appartient à une unique interface d'un objet, qui correspond à un point d'accès à cet objet. Une interface définit un ensemble d'interactions possibles d'un objet, qui correspond à un comportement observable. En effet, seules les interactions sont observables, et non les actions internes, du fait de l'encapsulation. Un objet peut avoir plusieurs interfaces et leur nombre peut varier dans le temps. Celles-ci constituent donc des vues abstraites des fonctions fournies par un objet, en masquant la façon dont celui-ci manipule l'information ou effectue les changements d'état.

Le concept d'objet permet de construire des spécifications de système ODP. Un système est alors composé d'objets interagissants. La composition permet de décrire une relation hiérarchique entre objets. Ainsi composer deux objets permet d'en générer un nouveau appelé objet composite. Réciproquement, la décomposition est un procédé de raffinement permettant de passer d'une application répartie complexe à un ensemble d'objets plus simples qui pourront à leur tour être

décomposés. Composition et décomposition sont des termes et des activités de spécification duales, qui s'appliquent non seulement aux objets mais également aux comportements.

Les objets sont décrits par des gabarits[2] (*template*), qui spécifient leurs caractéristiques communes à un niveau de détail suffisant pour permettre leur instanciation. Un gabarit caractérise donc l'ensemble des objets qu'il instancie. Ainsi des objets exhibant le même comportement peuvent être décrits par le même gabarit.

Le modèle objet définit le concept de type comme étant un prédicat caractérisant une collection d'objets. Un objet est alors du type, ou satisfait au type si le prédicat s'applique à cet objet. On l'appelle instance du type T. Notons qu'il n'existe pas forcément de rapport logique entre les objets d'un même type, excepté le fait qu'ils vérifient la même propriété. L'ensemble des objets d'un type donné forme une classe.

7.2 Le modèle architectural

Le modèle architectural ODP est construit sur la notion de **point de vue**.

Un point de vue est une subdivision d'une spécification d'un système complexe. Il permet pendant la phase de conception de considérer le système sous un angle particulier en ne s'intéressant qu'à la partie de spécification relative à celui-ci. Chaque point de vue représente une abstraction différente du même système. Les points de vue ne forment pas une séquence fixe et ne doivent donc pas être assimilés à une structuration en couches. De même, ils ne sont pas créés dans un ordre fixe selon une méthodologie de conception. Notons d'ailleurs que le modèle de référence ODP ne constitue pas en soi une méthodologie de spécification de systèmes répartis, même si beaucoup l'utilisent comme tel. Il définit une architecture qui utilise pour les besoins de sa spécification une séparation en cinq points de vue, qui sont les suivants :

Le point de vue entreprise décrit un système ODP en spécifiant les besoins d'un domaine d'application donné. Ainsi, il est axé sur les rôles, les objectifs, le domaine d'application et les politiques du système. Il permet de répondre au "pour quoi ?" du système ;

Le point de vue information spécifie l'information gérée et manipulée ainsi que les traitements de cette information effectués par un système ODP. Il est axé sur la sémantique et le traitement de l'information. Il permet de répondre au "quoi ?" ;

Le point de vue traitement exprime une décomposition fonctionnelle d'un système en objets interagissants via leurs interfaces et ce, en faisant abstraction de la répartition. Il permet de répondre au "comment faire ?", c'est le "comment" fonctionnel ;

Le point de vue ingénierie décrit l'infrastructure qui est requise pour mettre en œuvre une spécification de traitement. Il est axé sur les mécanismes et les fonctions qui prennent en charge l'interaction répartie des objets du système. Il permet de répondre au "où ?" ;

Le point de vue technologie est axé sur les choix technologiques de réalisation et d'implantation du système.

Afin de représenter un système ODP selon un point de vue donné, il est nécessaire de définir un ensemble structuré de concepts qui permettent d'exprimer la partie de spécification relative à ce point de vue. Cet ensemble forme un langage de point de vue.

---

[2] Précisons que le concept de gabarit, ainsi que les concepts de type et classe introduits dans le modèle ODP, s'appliquent aussi bien aux objets qu'aux actions et aux interfaces.

7.2.1 Le point de vue "Entreprise"

Considérer un système ODP selon le point de vue entreprise permet d'écrire une spécification d'entreprise dudit système [ISO 00]. Un concept de structuration majeur d'une spécification d'entreprise est celui de **communauté**. Une communauté est une configuration d'objets d'entreprise formée pour remplir un objectif. Elle modélise une collection d'entités (êtres humains, système de traitement d'information, ressources variées, etc) sujets à un contrat implicite ou explicite gouvernant leur comportement collectif. Au sein d'une communauté, les objectifs des membres sont contraints pour être conforme à l'objectif de la communauté.

Une spécification d'entreprise inclut au moins la description d'une communauté dans laquelle le système ODP est vu comme un objet unique d'entreprise interagissant avec son environnement. Cette communauté est désignée sous le terme <S>community.

Une spécification d'entreprise peut inclure la description de plus d'une communauté. Elle peut ainsi être structurée en un ensemble de communautés qui interagissent, chacune de ces communautés étant alors vue comme un objet composite (appelé le c-objet).

Une spécification d'entreprise est ainsi composée des spécifications des différents éléments tels que communautés, rôles, objets d'entreprise etc. Selon le choix du modélisateur et le niveau de détail souhaité, chacun de ces éléments peut être spécifié par les caractéristiques de l'élément, ou par le type de l'élément ou par un gabarit de l'élément. La norme ne fait aucune prescription sur le processus de spécification ou sur le niveau d'abstraction utilisés dans une spécification d'entreprise.

L'objectif de la communauté est donc ce qui motive son existence. Au sein d'une communauté, chaque objet d'entreprise joue un rôle. Un objet peut jouer plusieurs rôles, dans la même communauté ou dans des communautés distinctes. Réciproquement, un rôle peut être rempli par différents objets, mais de façon successive dans le temps et non simultanée. Autrement dit, à un instant donné, un rôle est rempli par un et un seul objet. Le rôle d'acteur dans une communauté est distinct de celui d'artefact. Vis-à-vis d'une action, l'acteur est l'objet qui participe à l'action et l'artefact est l'objet qui est référencé dans l'action. Jouer un rôle d'acteur dans une communauté signifie alors que l'objet est acteur pour au moins une action de ce rôle tandis que jouer un rôle d'aterfact dans une communauté signifie que l'objet est artefact pour toutes les actions de ce rôle.

Assigner des objets aux rôles de la communauté permet de peupler la communauté. Le comportement d'un objet auquel on assigne un rôle doit alors être cohérent avec le comportement identifié par ce rôle. Cette assignation peut varier dynamiquement pendant la durée de vie de la communauté. De même, la création et la destruction de rôles sont envisageables. Une spécification d'entreprise doit établir les circonstances de tels changements. L'inclusion de ces changements dans la spécification relève d'un choix de modélisation, i.e., les comportements associés à ces changements peuvent être explicites ou non. La terminaison d'une communauté est également un choix de modélisation. Elle peut se produire car l'objectif a été atteint, ou qu'il y a eu échec dans la recherche de l'objectif. Le comportement de terminaison peut être inclus explicitement ou non dans la spécification.

Les politiques sont utilisées pour exprimer des contraintes sur le comportement des objets jouant un rôle d'acteurs. Une politique s'applique à la communauté, à un ou plusieurs rôles ou à un type d'actions. Elle peut être exprimée comme une obligation, une autorisation, une permission ou une interdiction, les définitions suivantes devant s'appliquer à ces termes :

- obligation : prescription stipulant la nécessité d'un comportement particulier,

- autorisation : prescription stipulant qu'un comportement particulier ne doit pas être empêché,
- permission : prescription autorisant un comportement particulier,
- interdiction : prescription stipulant qu'un comportement particulier ne doit pas se manifester.

Un type particulier de communauté est la fédération, qui est une communauté de domaines établie à des fins de coopération. Le concept de fédération permet d'établir les principes généraux d'inter-fonctionnement entre domaines et de décrire les politiques gouvernant cet inter-fonctionnement.

7.2.2 Le point de vue "Information"

Le point de vue information définit la sémantique de l'information et la sémantique du traitement de l'information dans un système ODP. Pour cela, trois schémas sont définis :

Le **schéma d'invariant** est un ensemble de prédicats qui doit toujours être vrai sur un ensemble d'informations (états d'un objet) ;

Le **schéma statique** spécifie l'état d'un ou plusieurs objets d'information à un instant donné dans le temps. Il permet de spécifier pour des objets des états particuliers dans lesquels il faut forcer l'objet, distincts des états usuels apparaissant en fonctionnement normal (par exemple, un état d'initialisation ou un état d'exception) ;

Le **schéma dynamique** spécifie les changements d'états autorisés en fonctionnement normal pour un ou plusieurs objets d'information. Contrairement au schéma statique, il n'y a pas à forcer l'objet à passer dans ces états et à subir ces changements.

Les schémas statique et dynamique doivent respecter les contraintes de tout schéma d'invariant.

7.2.3 Le point de vue "Traitement"

Le point de vue traitement s'intéresse à la décomposition fonctionnelle d'un système ODP en objets interagissants à travers leurs interfaces. Il fournit les concepts relatifs à un modèle de programmation répartie abstrait, basé objet, pour décrire la structure et l'exécution d'applications réparties dans un environnement ODP. Dans ce point de vue sont définis un modèle d'interaction, des interfaces de traitement et un modèle de liaison.

Le modèle d'interaction

Dans le point de vue traitement, les interactions entre objets sont essentiellement asynchrones, reflétant en cela les hypothèses sur la nature des environnements répartis (absence d'horloge globale, d'état global, délais arbitraires des communications, possibilité de défaillance, etc.). Une interaction peut être de la forme suivante :

- Une **opération** est similaire à une procédure et est invoquée à une interface désignée. L'opération reflète le paradigme client-serveur. C'est une interaction entre un objet client et un objet serveur par laquelle l'objet client demande l'exécution d'une fonction par l'objet serveur. Il y a deux types d'opérations :
- L'**interrogation** est une fonction qui retourne un résultat. Une interrogation peut être synchrone (bloquante) au sens où le client est suspendu en attente du résultat ou asynchrone (non bloquante) ;
- L'**annonce** est une invocation d'opération sans retour de résultat. Elle est non bloquante.
- Un **flux** est une abstraction de séquences d'interactions aboutissant à l'acheminement d'informations d'un objet producteur (source des informations) vers un objet consommateur

(collecteur des informations). Par exemple, un flux peut être utilisé pour modéliser le débit d'information audio ou vidéo d'une application multimédia ou d'un service de télécommunication. Ce peut être également un transfert de fichiers. Sous Unix, on pourrait assimiler un flux à une socket. Un flux est caractérisé par son nom et son type, qui spécifie la nature et le format des données échangées.

- Un **signal** est une interaction atomique élémentaire aboutissant à une communication unilatérale d'un objet initiateur vers un objet répondeur (c'est-à-dire acceptant la communication). Il correspond par exemple à un événement ou une interruption. Sous Unix, on pourrait assimiler le signal à un signal Unix.

Les interfaces de traitement

Un type d'interaction appartient à une unique interface de traitement. Il y a donc trois catégories d'interfaces de traitement : interface opération, interface flot et interface signal. Une interface d'une catégorie donnée ne contient que des interactions de cette catégorie. Par exemple, une interface opération est une interface dont toutes les interactions sont des opérations.

La **signature** d'une interface est définie en fonction de la catégorie de l'interface. La seule notation prescrite dans le modèle de référence ODP est celle utilisée pour les signatures d'interface opération, qui sont décrites par un langage de définition d'interface (Interface Definition Language -- IDL) qui est celui de CORBA.

Un objet peut avoir plusieurs interfaces de différentes catégories. Son nombre d'interfaces peut varier. Une spécification de traitement définit un ensemble initial d'objets de traitement et leur comportement. La configuration change selon que les objets de traitement instancient d'autres objets de traitement ou d'autres interfaces de traitement, exécutent des actions de liaisons, remplissent des fonctions de contrôle sur des objets de liaison et suppriment des interfaces de traitement ou des objets de traitement.

Le modèle de liaison

Les interactions entre interfaces de traitement nécessitent préalablement l'établissement d'une liaison (binding) entre elles, c'est-à-dire un chemin de communication. L'exemple type de liaison statique en programmation est celui de l'éditeur de liens résolvant la référence externe "appel de procédure" dans un programme principal par la procédure en question.

On distingue les liaisons implicites (cf. l'exemple ci-dessus) des liaisons explicites. De plus, pour les liaisons explicites, on distingue les liaisons primitives (entre deux interfaces) des liaisons composites (entre deux ou plus interfaces). Dans ce dernier cas, un objet de liaison est utilisé.

7.2.4 Le point de vue "Ingénierie"

Une spécification d'ingénierie définit les fonctions et les mécanismes requis pour prendre en charge l'exécution et l'interaction répartie entre les objets d'un système ODP. Contrairement au point de vue traitement qui permet de considérer quand et pourquoi les objets interagissent, le point de vue ingénierie s'intéresse à la façon dont ils interagissent (le "comment").

D'un point de vue ingénierie, une application répartie est un ensemble d'objets d'ingénierie de base interagissants. Un objet d'ingénierie de base constitue alors la représentation ingénierie d'un objet de traitement qui nécessite le support d'une infrastructure répartie.

Un système ODP est décrit dans le point de vue ingénierie comme un ensemble de nœuds interconnectés. Un **nœud (node)** est une configuration d'objet d'ingénierie qui constitue une abstraction de ressources informatiques et des logiciels associés. Il est sous le contrôle d'un objet d'ingénierie appelé **noyau (nucleus)** qui fournit la fonction de gestion de nœud. Un nœud contient un ensemble de **capsules**. Une capsule est une configuration d'objets d'ingénierie constituant une abstraction d'un processus local à un nœud et de son espace d'adressage associé. Elle englobe des grappes, une **grappe (cluster)** correspondant à une configuration d'**objets d'ingénierie de base (Basic Engineering Object – BEO)** constituée à des fins de persistance, de sauvegarde et de migration. Un BEO est caractérisé par son identité unique, ses états, son comportement et ses interfaces et le fait qu'il requiert le support d'une infrastructure répartie.

La communication entre objets d'ingénierie de base est prise en compte par les canaux. Les canaux correspondent à la réalisation des objets de liaison de traitement dans le point de vue ingénierie. En fait, un canal n'est pas systématiquement nécessaire pour établir une liaison entre objets d'ingénierie, car dans le cas où des objets appartiennent à la même grappe, la liaison peut être réalisée par des mécanismes spécifiques du système, par exemple, un IPC dans un système Unix. Elle est alors désignée de liaison locale dans le langage d'ingénierie. Le canal prend donc en charge les liaisons dites réparties qui s'établissent entre objets de grappes différentes, voire de la même grappe. Un canal est une configuration d'objets d'ingénierie composée d'objets talons, lieurs, protocoles et intercepteurs. Il peut être physiquement réalisé par exemple par un RPC (Remote Procedure Call) classique, ou un RPC objet.

7.2.5 Le point de vue "Technologie"

Le point de vue "Technologie" est très peu développé dans la norme puisqu'il traite de l'implantation réelle d'un système ODP, c'est-à-dire les choix matériels et logiciels effectués pour implanter les spécifications résultant des autres points de vue. Ces aspects sortent du champ de normalisation.