Construction d'applications réparties ouvertes

ODAC : une méthodologie de construction de systèmes à base d'agents fondée sur ODP

Marie-Pierre Gervais

LIP6

Marie-Pierre.Gervais@lip6.fr

Résumé : La méthodologie ODAC (Open Distributed Applications Construction) a pour objectif de fournir un ensemble de méthodes et outils à un concepteur de systèmes à base d'agents pour lui permettre de maîtriser la complexité du processus de construction de tels systèmes. Elle vise à supporter la spécification et l'implantation d'une application répartie au sein d'environnements de type plate-forme à agents mobiles. Nos travaux sont fondés sur la norme de traitement réparti ouvert (Open Distributed Processing - ODP) et le paradigme agent.

Mots-Clés : traitement réparti ouvert (ODP), conception d'applications réparties, agent logiciel

Table des matières

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.

Nos recherches portent sur la définition de modèles, de méthodes et d'outils pour la construction d'applications réparties sous forme de composants interagissants. Au vue des propriétés que de tels composants doivent présenter (interaction de haut niveau, autonomie, flexibilité, interopérabilité), nous choisissons d'utiliser le paradigme agent pour les caractériser. En effet, ces derniers exhibent des capacités d'interactions et d'adaptation qui offrent la flexibilité nécessaire pour répondre aux exigences d'interopérabilité et de réutilisation. Ainsi les applications que nous construisons sont des systèmes à base d'agents. Il s'agit alors de définir une méthodologie orientée agent en tirant partie de normes et techniques existantes et provenant de différents domaines tels que le génie logiciel orienté objet et les systèmes répartis. Nos travaux se focalisent sur l'étape de spécification d'un système à base d'agents. Ils reposent sur un cadre normatif par l'utilisation de la norme de traitement réparti ouvert (Open Distributed Processing — ODP).

Ce document présente la méthodologie ODAC dans sa version actuelle. Les paragraphes 2 et 3 introduisent les aspects principaux respectivement du paradigme agent et de la norme ODP, pré-requis pour aborder la méthodologie qui fait l'objet du paragraphe 4. La conclusion énonce les développements futurs de la méthodologie.

2. La technologie agent

L'apparition d'Internet a largement favorisé le développement d'applications utilisant la technologie agent [BOU97]. Ce concept provient de l'intelligence artificielle distribuée et est aujourd'hui employé dans des contextes très différents [FER95]. Il existe donc de très nombreuses définitions et taxinomies [GRE97, JEN96, NWA96, WOO95]. Nous retiendrons ici non pas une définition du terme "agent" qui nous intéresse finalement peu mais plutôt une liste de propriétés qui caractérisent un agent. Ainsi nous commencerons par caractériser un agent logiciel comme une entité logicielle autonome, réactive, pro-active et sociale. Il est de plus parfois mobile. Un agent est souvent défini comme "un objet qui …", aussi nous expliciterons la notion d'agent par rapport à celle d'objet. Nous présenterons les efforts de standardisation dans ce domaine, ce qui nous amènera à considérer un aspect spécifique de la technologie agent qui focalise beaucoup d'attention : les plates-formes à agents mobiles. Enfin nous dresserons le panorama des différentes méthodologies orientées agent.

2.1. Caractérisation d'un agent logiciel

Un agent est **autonome** au sens où il possède le contrôle de son comportement, c'est-à-dire qu'il décide de ses actions. Il exerce un contrôle sur son état interne et ses actions, indépendamment de toute intervention externe.

Un agent est **réactif** car il perçoit son environnement et répond de manière opportune aux changements qui s'y produisent. Ainsi il ne réagit pas uniquement à la demande mais aussi aux événements observables de l'environnement [ATG99].

Un agent est **pro-actif** car il agit en fonction d'un but et non simplement en réaction à l'environnement. Il est capable de comportements (dirigés par l'objectif à accomplir) prenant l'initiative plutôt que d'attendre des ordres lui indiquant ce qu'il doit faire.

Un agent est **social** parce qu'il est capable d'agir avec ou pour d'autres agents. II est donc aussi capable de communiquer avec eux. Cette communication repose généralement sur l'échange de messages, une suite d'échanges de messages formant une conversation. Il faut noter qu'un agent peut engager des conversations avec d'autres agents, c'est-à-dire être engagé dans de multiples transactions concurrentes.

Un agent est **mobile** s'il possède la capacité de se déplacer dans un réseau sur des sites distants pour y exécuter des tâches spécifiques [CHE95, HAR95]. Cette caractéristique se fonde sur un principe de communication appelé "programmation distante" (Remote Programming — RP) [MAG96, WHI96], jugé plus économique que l'appel de procédure à distance (Remote Procedure Call — RPC) car l'agent interagit localement plutôt qu'à distance avec les ressources nécessaires à l'accomplissement de sa tâche.

2.2. Agent vs objet

De même que la définition d'agent, la différence entre agent et objet est un débat qui fait couler beaucoup d'encre [BUR96, IGL98, JEN00].

La propriété d'autonomie d'un agent est une première différence entre les deux paradigmes. Un objet est passif par nature et est activé sur réception d'un message (typiquement la réception d'une invocation d'une de ses méthodes). Il n'encapsule aucune capacité de choix d'actions. L'agent quant à lui a un rôle à jouer au sein du système, qui justifie qu'il peut être amené à exécuter des actions pour lui, même s'il est déjà en relation directe avec d'autres agents. C'est un objet toujours actif. Ceci se résume par le désormais célèbre : “ Les objets le font gratuitement ; les agents le font pour l'argent ” [JEN98].

Cependant, cette idée de passivité de l'objet est remise en cause avec l'introduction de la notion d'objet actif [JEN98, BOO98]. Aussi la question se pose : "un objet actif est-il un agent ?". Il est vrai que les notions se rapprochent, si on fait abstraction de la pro-activité et selon l'interprétation que l'on a des caractéristiques d'autonomie, de capacité sociale et de réactivité. Ainsi un objet distribué CORBA peut être considéré comme étant un agent rudimentaire, c'est-à-dire autonome, social et réactif mais manquant d'intelligence que nécessite la pro-activité [SCH99]. Reste le degré de mise en œuvre de chacune des caractéristiques énumérées. Sur l'aspect social, par exemple, nous avons mentionné qu'un agent peut participer à plusieurs conversations simultanément. Cette caractéristique, aisément concevable dans un contexte d'échanges de messages, l'est beaucoup moins dans le contexte d'invocations de méthodes qui est celui des objets [ATG99]. La réactivité d'un objet, quant à elle, est de type événement-action au sens où un objet réagit à des invocations de méthodes.

Le débat reste encore largement ouvert, mais il semble cependant que la classification objet actif - agent s'estompe au profit d'une graduation dans les caractéristiques exhibées par l'"entité" à qualifier. La question n'est plus tant de savoir si on a à faire à un agent ou à un objet, mais plutôt d'évaluer le degré d'autonomie, de réactivité, ou de sociabilité de ladite entité. C'est ainsi que l'on voit fleurir des agents intelligents, des agents mobiles mais aussi des objets mobiles, des objets actifs, etc. Ce foisonnement de termes est à imputer au fait que la technologie agent présente de nombreuses facettes et par conséquent mobilise plusieurs communautés scientifiques (intelligence artificielle, informatique répartie à objet, génie logiciel) de sensibilité différente. Même si un agent diffère d’un objet, cela ne signifie pas que les agents ne sont pas composés à partir d’objets. Cette approche n’est pas seulement défendue pas des organismes de standardisation comme l'OMG, elle est également utilisée dans l’implémentation des plates-formes agent (l’infrastructure agent construite au-dessus d’un système complexe orienté objet). Nous abordons ces aspects dans les paragraphes suivants.

2.3. La standardisation

La technologie agent faisant l'objet de nombreux travaux, une intégration des concepts clés est apparue essentielle avec comme finalité la production de spécifications destinée à l'industrie. C'est l'objectif que s'est fixé la Foundation for Intelligent Physical Agents (FIPA) que de promouvoir le développement et la spécification de la technologie agent [FIP]. De même l'Object Management Group (OMG) a étendu le champ de ses travaux initialement fondé sur le paradigme objet pour intégrer la technologie agent mobile en vue de produire des spécifications.

2.3.1. FIPA

FIPA est une association internationale de compagnies[1] qui consentent à partager l'effort pour produire des spécifications génériques de la technologie agent, acceptées au plan international, avec un haut niveau d'interopérabilité entre un grand nombre d'applications.

Les standards FIPA fournissent :

[1] 50 membres: Europe - Alcatel, BT, Nortel, Simens, France Télécom, etc., USA - HP, IBM, Lucent, Motorola, SOLEIL, etc, Asie

- Des moyens communs par lesquels les agents peuvent communiquer. Ainsi peuvent-ils échanger des informations, négocier des services, ou déléguer des tâches via un langage de communication agent (Agent Communication Language — ACL) ;
- Des facilités de répertoire grâce auxquelles les agents peuvent se localiser l'un l'autre (Directory Facilities — DF) ;
- Un environnement fiable dans lequel les agents peuvent opérer et échanger des messages confidentiels ;
- Une modalité unique d'identifier d'autres agents (ex. les noms globalement uniques) ;
- Un moyen d'accéder aux systèmes non-agents et à d'autres applications ;
- Un moyen d'interaction avec les utilisateurs ;
- Un moyen de migrer d'une plate-forme à une autre, si nécessaire.

FIPA produit deux types de spécifications : des spécifications **normatives** sur le comportement externe d'un agent, qui assurent l'interopérabilité avec d'autres sous-ensembles spécifiés par FIPA et des spécifications **informatives** d'applications servant de conseils d'utilisation industrielle des technologies FIPA.

Le troisième ensemble de spécifications normatives FIPA a été publié en 1999 [FIP1-FIP4]. Des implantations de systèmes agent conformes à FIPA commencent à apparaître. Ainsi une version publique appelée FIPA-OS est disponible sur le site de Nortel Networks. C'est une plate-forme qui est utilisée dans les projets de collaboration européens : FACTS, Caméléon, MAPPA. Elle met en œuvre des composants logiques conformes à ceux décrits dans le modèle de référence de la gestion d'agent défini dans la partie 1 des spécifications [FIP1]. Le modèle de référence définit les composants suivants :

- Un **Agent** (conforme FIPA) est l'acteur fondamental dans un domaine. Il offre une ou plusieurs capacités de service dans un modèle unifié et intégré d'exécution qui peut inclure l'accès au logiciel externe, aux utilisateurs humains et aux équipements de transmission.
- La plate-forme agent (**Agent Platform — AP**) fournit une infrastructure dans laquelle des agents peuvent être déployés. Un agent doit s'enregistrer sur une plate-forme afin d'interagir avec d'autres agents situés sur cette plate-forme ou d'autres plates-formes. Une plate-forme se compose de trois ensembles de capacités : le système de gestion d'agent, le canal de communication agents et le répertoire auxiliaire.
- Le système de gestion d'agent (**Agent Management System — AMS**) est un agent qui assure la création, la suppression, la suspension, la reprise, l'authentification et la migration d'un agent sur la plate-forme agent et fournit un répertoire " pages blanches " pour tous les agents résidants sur une plate-forme agent. Il assure la mise en correspondance des noms globalement uniques d'agent (Globally Unique IDentifier - GUID) avec les adresses de transport locales employées par la plate-forme.
- Le canal de communication agents (**Agent Communication Chanel — ACC**) est un agent qui utilise l'information fournie par le système de gestion d'agent pour router les messages vers les agents de la plate-forme ou vers ceux résidants sur d'autres plates-formes.
- Le répertoire auxiliaire (**Directory Facilitator — DF**) est un agent qui fournit un service de répertoire " pages jaunes " pour les agents. Il enregistre les descriptions des agents et les services qu'ils offrent.

- Au sein de chaque plate-forme doit résider au moins un agent fournissant le service de courtage de ressource agent (**Agent Resource Broker — ARB**) afin de permettre le partage des services non-agent.

2.3.2. OMG

L'OMG, organisme de standardisation spécialisé dans les problèmes d'interopérabilité avec le standard CORBA, a étendu son effort originairement centré sur la technologie objet vers la technologie agent. En effet, les différences entre les systèmes d'agents mobiles actuellement développés empêchent l'interopérabilité entre eux. La spécification MASIF (Mobile Agent System Interoperability Facilities), adoptée par l'OMG en mars 1998, permet l'interopérabilité entre des plates-formes agent fournies par différents vendeurs [MAS97]. MASIF intègre le paradigme RPC dans la technologie agent mobile et utilise les avantages de CORBA. Elle standardise la gestion d'agent (création/destruction, suspension/reprise d'exécution d'un agent), la localisation des agents dans un environnement réparti, la communication entre les agents, le transfert d'un agent (état, code et données) et le service de nommage agent.

MASIF définit un modèle d'environnement réparti agent. Chaque composant du modèle est interfacé par des interfaces IDL.

Un Agent conforme MASIF est une entité qui agit pour le compte d'une personne ou d'une organisation de manière autonome. L'exécution des agents est assurée par un système agent (interface IDL - MAFAgentSystem) qui est une **plate-forme** agent. Une place (**Place**) est un contexte dans le système agent au sein duquel l'agent est exécuté. Une région (**Region**) regroupe plusieurs systèmes agent avec un détecteur (**Finder**). Un détecteur **Finder** (interface IDL - MAFFinder) est un registre pour localiser les agents, les places et des systèmes agent.

Il existe actuellement une implantation MASIF d'un environnement agent réalisée par GMD FOKUS appelée Grasshopper [GRA].

2.3.3. Conclusion

Les deux courants de standardisation de la technologie agent FIPA et OMG-MASIF laisse apparaître de nombreuses similarités :

- Une entité responsable de la création, la suppression, l'authentification et la migration d'agent (c'est l'AMS de FIPA et le MAFAgentSystem dans MASIF) ;
- Une entité maintenant des registres sur les agents (c'est le DF de FIPA et le MAFFinder de MASIF) ;
- Une entité assurant le transfert des messages dans un environnement réparti (c'est l'ACC de FIPA et un ORB dans MASIF).

Par contre, FIPA utilise un langage de communication agent qui permet la spécification des opérations et un protocole de communication de haut niveau. MASIF assure seulement un ensemble minimal de fonctionnalités, accessibles via des interfaces IDL.

Depuis peu, les deux organismes font des efforts conjugués pour fusionner les deux approches. Dans ce cadre, l'OMG a émis une demande d'information (Request For Information — RFI). Le résultat sera utilisé pour développer un livre vert de la technologie agent (Agent Technology Green Paper), déjà disponible dans une première version [ATG99], un plan d'adoption de la technologie agent et une série de demandes de proposition (Requests for Proposal — RFPs).

2.4. Les plates-formes à agent mobile

Pour supporter la mise en œuvre de la propriété de mobilité des agents, des plates-formes agent ont vu le jour, la plupart d'entre elles se fondant sur Java [KOT97, MER97, STR96]. Aglet ou Voyager sont des exemples significatifs de telles plates-formes [LAN97, GLA99]. Plus récemment, des plates-formes conformes aux standards FIPA et OMG ont été développées, respectivement FIPA-OS et Grasshopper [FIPOS, GRA].

2.4.1. Grasshopper

Grasshopper est produit par IKV++ (GMD FOKUS). C'est la seule plate-forme conforme à la spécification MASIF de l'OMG dans sa conception. L'environnent Grasshopper consiste en un nombre d'agences (hôtes) et un Region Registry (une base de données réseau qui contient des informations sur les hôtes et sur les agents) connectés à distance par un ORB propriétaire (Grasshopper ORB). Bien sûr, la conformité MASIF de la plate-forme induit une capacité d'interconnexion avec des ORB CORBA. Les agences sont les environnements d'exécution pour les agents mobiles. Plusieurs agences peuvent êtres groupées dans une région représentée par un Region Registry.

2.4.2. FIPA-OS

FIPA-OS est un logiciel libre implantant des éléments obligatoires des spécifications FIPA pour l'interopérabilité d'agent. FIPA-OS est un cadre expérimental d'agent, provenant de la recherche des laboratoires Harlow de Nortel Networks. Les futures versions de FIPA-OS seront développées par Nortel dans le cadre des programmes de recherche européennes FACTS et Cameleon.

2.4.3. Aglets

Le terme Aglet vient de applet agent (**AG**ent app**LET**). Le modèle de programmation applet en Java a eu une influence sur la conception de la plate-forme elle-même. Aglet est une plate-forme d'agent mobile qui peut opérer indépendamment, sans l'utilisation d'un navigateur. Elle a aussi des facilités qui permettent la coopération avec des navigateurs ou avec des applets Java. La bibliothèque des classes d'Aglets fournit une interface de programmation (**Application Interface-API**) riche qui facilite le codage d'un comportement complexe d'agent.

2.4.4. Voyager

Voyager est une plate-forme basée sur Java et produite par ObjectSpace. La plus importante caractéristique de Voyager (et qui la différencie par rapport à d'autres plates-formes) est l'intégration entre les agents et l'exécution répartie. Le point central de cette intégration est l'implémentation d'un ORB Java propriétaire orienté agent (Voyager ORB).

2.4.5. Conclusion

L'apparition de ces plates-formes a permis le développement de nombreuses applications orientées agent (mobile ou non), dont les exemples types sont le commerce électronique [MER96, WHI94], la recherche d'informations [CAM97, PER96, VAN96], la gestion de réseau [ESF96, OLI96, ROS96] ou le multimédia [FAL97, KIS96, KWA96, NYG96]. Dans le domaine des télécommunications par exemple où la technologie agent connaît un grand essor, les premiers travaux se sont focalisés sur la gestion de réseau par délégation [YEM91, ZNA97] et sur la fourniture d'assistants personnels [KOZ96, MON96]. En février 1998, le programme européen ACTS a constitué la banque de données CLIMATE (Cluster for Intelligent Mobile Agents for Telecommunication Environments)[CLI98]. Son but est de promouvoir l'échange d'informations et de faciliter la coopération entre les projets européens utilisant la technologie agent. Cette initiative est d'une importance clé pour la FIPA et l'OMG car elle se pose en candidate naturelle d'utilisation de ces standards émergents. De ce fait, une liaison entre le groupe Agent de l'OMG et CLIMATE est en cours d'établissement.

2.5. Les méthodologies de développements de systèmes basés agent

Les premières applications aussi bien que les plates-formes agent furent tout d'abord développées de façon ad-hoc, sans considération réelle des procédés utilisés en génie logiciel lors du développement d'applications. Ainsi il est souvent illusoire de chercher à obtenir les spécifications de telle application, ou telle plate-forme : il n'y en a tout simplement pas ! Les développements ont été réalisés directement en Java (ou un autre langage de programmation, toutefois l'usage de Java reste prédominant). L'idée de génie logiciel orienté agent est somme toute assez récente [WOO94]. L'intérêt d'introduire des méthodologies orientées agent est illustré dans [JEN00]. Une conséquence immédiate de la mise à disposition de méthodes et outils fondés sur le paradigme agent serait bien évidemment la promotion de celui-ci et son évolution vers une utilisation à grande échelle.

Nous présentons dans ce paragraphe les différentes méthodologies recensées à l'heure actuelle dans la littérature. Nous fondons cette présentation sur le panorama dressé dans [IGL98]. La tendance majeure des différents travaux sur les méthodologies agent est de fonder celles-ci sur des méthodologies existantes et de les faire évoluer pour prendre en compte les spécificités agent plutôt que d'inventer une nouvelle méthodologie "from scratch". Deux catégories principales de méthodologies se dégagent ainsi : celles fondées sur les méthodologies orientées objet et celles fondées sur les méthodologies orientées ingénierie de la connaissance. Par ailleurs, d'autres méthodologies sont proposées, non recensées dans les deux catégories précédemment citées.

2.5.1. Extensions de méthodologies orientées objet

Les méthodologies OO, actuellement largement utilisées en contexte industriel, semblent naturellement candidates pour le développement de systèmes basés agent vu les similarités entre agent et objet (cf. paragraphe 3.2). Il reste cependant des différences qui ne permettent pas une application directe de ces méthodologies et par conséquent, qui nécessitent de les faire évoluer. Rappelons que l'agent est autonome et gouverne ses actions, qu'il interagit de façon complexe ne correspondant pas uniquement à des invocations de méthodes et enfin, qu'il est caractérisé par un état mental lui permettant d'agir en fonction d'un but et de planifier ses actions en conséquence. Il s'agit alors d'inclure ces aspects dans la démarche méthodologique. C'est dans cette optique que les différentes méthodologies suivantes ont été proposées.

Analyse et conception agent de Burmeister

Burmeister définit trois modèles pour l'analyse d'un système [BUR96]. Le premier, le *modèle agent*, propose d'identifier la structure interne de l'agent selon le modèle BDI (Belief, Desire, Intention). Une extension des cartes CRC[2] (Classes-Responsabilites-Collaborations) est utilisée

[2] Les cartes CRC sont utilisées dans la méthode orientée objet RDD (Responsability Driving Design) [WIR90]

pour décrire ce modèle [WIR90]. Le deuxième, le *modèle organisationnel*, utilise la notation OMT (Object Modelling Technique) pour exprimer les rôles de chacun au sein du système ainsi que les différentes relations d'héritage [RUM91]. Enfin, le dernier, le *modèle de coopération*, décrit les différentes interactions entre les composants du système.

Technique de modélisation agent pour les systèmes à agent BDI (méthode AOM)

Cette technique propose deux niveaux de conception pour un système composé d'agents BDI [KIN96a, WOO99]. Le premier niveau, externe, permet de décomposer le système en agents et d'identifier les différentes interactions entre eux. À ce niveau, deux modèles sont définis : le *modèle agent* qui décrit les classes d'agent et les instances et le *modèle d'interaction* décrivant les communications et les relations de contrôle entre les agents. Le second niveau, appelé niveau interne, permet d'exprimer la structure des agents. Les modèles internes décrivent pour chaque classe d'agents leurs croyances, leurs buts et leurs intentions.

Une méthode de scenarii pour les systèmes multiagents (méthode MASB)

Cette méthode propose dans la phase d'analyse de décrire les scénarii entre les différents composants du système, les rôles de chacun, les données qu'ils manipulent et les interfaces entre le système et le monde extérieur [MOU96]. La phase de conception sélectionne les scenarii et les rôles qui seront réellement implantés, décrit une décomposition fonctionnelle des agents et spécifie les structures de données manipulées dans le système.

GAIA : Une méthodologie pour la conception et l'analyse orientées agent

Cette méthodologie traite les aspects de la conception d'un système multi-agents au niveau social (macro) et le niveau agent (micro) [WOO99, WOO00]. Durant la phase d'analyse, le système est considéré au niveau macro et est vu comme un ensemble de rôles qui interagissent. Deux modèles sont identifiés pendant cette phase : le *modèle de rôles* et le *modèle d'interactions*. À un rôle sont attachés les permissions/droits du rôle et les responsabilités du rôle. Le modèle d'interaction consiste en un ensemble de définitions de protocoles, un pour chaque type d'interaction inter-rôles. La phase de conception permet la génération de trois modèles : le *modèle agent* qui identifie les types d'agent et les instances, le *modèle de services* qui identifie les services associés à chaque type d'agent et le *modèle d'accointance* qui documente les accointances de chaque type d'agent.

2.5.2. Extensions de méthodologies orientées ingénierie de la connaissance

Les méthodologies d'ingénierie de la connaissance fournissent une bonne base pour modéliser des systèmes orientés agent puisqu'elles s'appliquent aux systèmes de connaissance et par conséquent elles procurent des techniques permettant la modélisation de la connaissance d'un agent. Bien que ces méthodologies ne soient pas aussi extensibles que celles orientées objet, elles ont cependant été appliquées avec succès dans différents projets. Les méthodologies exposées ci-après sont toutes fondées sur CommonKADS, standard européen pour la modélisation de la connaissance [SCH94].

La méthodologie CoMoMAS

CoMoMAS propose une extension de la méthodologie CommonKADS en fournissant un ensemble de modèles [GLA96]. Le *modèle agent* est le principal modèle, qui définit l'architecture et la connaissance d'un agent. Le *modèle d'expertise* décrit les compétences réactives et cognitives d'un agent. La distinction est faite entre tâches, résolution de problèmes et connaissances réactives. Le *modèle de tâche* décrit la décomposition des tâches. Le *modèle de coopération* décrit la coopération entre les agents, en utilisant les méthodes de résolution de conflit et la connaissance de coopération. Le *modèle système* définit les aspects organisationnels de la société d'agents ainsi que les aspects architecturaux des agents. Enfin le *modèle de conception* collecte tous les modèles précédents dans le but de les rendre opérationnels.

La méthodologie MAS-CommonKADS

Cette méthodologie étend les modèles définis dans CommonKADS en ajoutant des techniques provenant des méthodologies OO (OOSE et OMT) et de l'ingénierie des protocoles (SDL et MSC) pour décrire les protocoles d'agent [IGL97].

La première phase de la méthodologie est appelée phase de conceptualisation. C'est une phase informelle consistant à collecter les besoins de l'utilisateur pour obtenir une description du système du point de vue utilisateur. Elle utilise les "use case" de OOSE et les interactions entre use cases sont modélisées en MSC. Les phases d'analyse et de conception utilisent les modèles suivants. Le *modèle agent* décrit les caractéristiques principale d'un agent (capacité de raisonnement, services, buts, etc). Le *modèle d'expertise* décrit la connaissance nécessaire des agents pour accomplir leurs tâches. Le *modèle de tâche* décrit les tâches exécutées par les agents ainsi que leur décomposition. Le *modèle de coopération* décrit les conversations entre les agents, en modélisant les interactions en MSC et SDL. Le *modèle d'organisation* définit les aspects organisationnels de la société d'agents. Le *modèle de communication* détaille les interactions agent-utilisateur. Enfin le *modèle de conception* collecte tous les modèles précédents et s'intéresse aux aspects de leur mise en œuvre.

2.5.3. Autres méthodologies

La méthodologie Cassiopeia

Cette méthodologie distingue trois étapes dans la conception d'un système multi-agents [DRO98]. Tout d'abord, les comportements élémentaires de l'agent sont listés en utilisant des techniques fonctionnelles ou orientées objet. Puis les comportements relationnels sont analysés, i.e., les dépendances entre agents sont étudiées en utilisant un graphe de couplage. Enfin la dynamique de l'organisation est décrite par analyse du graphe de couplage.

Une méthodologie organisationnelle de conception de systèmes multi-agents

Cette méthodologie se focalise sur l'aspect organisationnel d'un SMA et de ce fait restreint volontairement le modèle afin qu'il s'intègre le plus aisément possible avec d'autres méthodologies [GUT99]. Le modèle organisationnel proposé fait partie d'un ensemble plus général, appelé Aalaadin [FER98]. Le modèle Aalaadin définit les concepts d'agents, de groupe et de rôle. La méthodologie se compose d'un ensemble de modèles et processus articulés autour de ce modèle agent/groupe/rôle. *Le modèle organisationnel générique* représente les notions de structure de groupes et les définitions des rôles. *Le modèle organisationnel spécifique* raffine le modèle générique pour un domaine particulier. La *structure organisationnelle* est la spécification complète d'un modèle d'organisation particulière, décrite dans un modèle organisationnel spécifique.

Une méthode intégrée pour la conception de systèmes multi-agents

Cette méthode est le résultat de la fusion de travaux fondamentaux sur les modèles génériques utilisés dans les SMA et de la volonté de réutiliser une expérience acquise dans le développement d'applications opérationnelles utilisant ces concepts [OCC00]. Elle intègre une démarche unifiée de conception qui vise à assembler des composants conceptuels. Elle est supportée par un outil de développement MASK (Multi-Agent System Kernel) chargé de la réalisation du système final par l'assemblage des composants opérationnels correspondants.

La démarche considère deux étapes de génie logiciel qui sont appelées analyse et conception orientées systèmes multi-agents. L'analyse permet de modéliser un SMA en quatre composantes fondamentales : les Agents, l'Environnement, les Interactions et l'Organisation, conformément à l'approche "Voyelles" [DEM97]. Un ou plusieurs modèles sont proposés pour décrire chacune de ces composantes. La conception permet de construire le SMA une fois connu ce que les agents ont à faire, par intégration des agents, environnements, interactions et organisations dans un SMA. Différentes approches de conception peuvent être considérées (conception par l'agent, par l'interaction, par l'organisation ou par l'environnement). Cette méthode permet d'intégrer dans un même outil ces différentes approches.

Une méthodologie de conception de systèmes multi-agents de résolution de problèmes par émergence

L'approche suivie se fonde sur l'analyse des systèmes existants, naturels ou artificiels, exhibant des propriétés émergentes afin de déterminer leurs points communs et d'en dériver une méthodologie heuristique [MUL98]. Le domaine d'application de la méthodologie vise les approches émergentistes de la résolution de problèmes. La méthodologie consiste à décomposer la structure de l'espace de recherche en composantes, à déterminer les interactions qui produiront ces composantes par effet de bord, d'en déduire alors les agents et leur dynamique qui permet d'engendrer ces interactions et à travers elles de parcourir l'espace de recherche, enfin de compléter la spécification de l'environnement qui inscrit l'état courant de la recherche avec une représentation des contraintes exogènes et d'une dynamique propre de propagation des conséquences.

Approches formelles

Plusieurs approches formelles ont été utilisées pour combler le vide entre les théories formelles et les implantations. Dans [WOO98], la logique modale temporelle permet de représenter les aspects dynamiques d'un agent et sert de base pour spécifier, implanter et vérifier des systèmes basés agent. L'implantation de la spécification peut être faire directement par exécution de la spécification avec un langage comme Concurrent Metatem [FIS97] ou par compilation de la spécification. Les langages formels de spécification de systèmes multi-agents comme DESIRE sont une alternative intéressante en tant que langages de spécification détaillée pouvant être utilisés dans toute méthodologie. DESIRE (DEsign and Specification of Interacting Reasoning components) fournit une cadre pour la conception fondée sur les composants qui s'appuie sur la décomposition des tâches [BRA97].

2.5.4. Conclusion

Ce panorama sur les méthodologies orientées agent illustre l'importance du sujet. De ces différentes propositions, il apparaît malgré leurs spécificités des convergences qui permettent d'énoncer que lors de la construction d'un SMA, il convient de prendre en compte trois composantes majeures, qui sont l'agent, l'organisation et la sociabilité. Une seule proposition fait intervenir une quatrième composante qui est l'environnement.

Les aspects relatifs à l'agent traitent de sa structure interne, et de façon plus ou moins détaillée de ses capacités, son expertise, les tâches qu'il réalise etc. L'aspect "organisation" est centré sur la notion de rôle, avec plus ou moins de précisions sur les responsabilités, permissions et droits qui s'y attachent. Enfin, la sociabilité recouvre les aspects de communication, d'interaction, de collaboration des agents.

Il n'y a pas dans les différentes méthodes de forte prescription sur l'ordre dans lequel ces trois composantes doivent être considérées. Toutefois, certaines rattachent telle composante à la phase d'analyse et telle autre à la phase de conception. Il est d'ailleurs important de souligner que la plupart de ces méthodes ne traitent que ces deux phases. Or une méthodologie se doit de prendre en compte l'ensemble du cycle de vie tel qu'il est considéré dans les approches de génie logiciel "classique", i.e., orienté objet. C'est à cette condition que le génie logiciel orienté agent pourra réellement émerger.

2.6. Conclusion

Cette présentation sur la technologie agent permet de mettre en évidence que celle-ci dépasse largement le cadre de l'intelligence artificielle et recouvre des aspects multiples. Nous avons insisté sur deux d'entre eux principalement qui sont l'infrastructure de support d'un système multi-agents en nous focalisant sur les agents mobiles et la méthodologie de développement de SMA. Ceci nous amène à constater qu'il n'existe à l'heure actuelle que peu de tentatives visant à concilier ces deux aspects, c'est-à-dire cherchant à définir une méthodologie permettant la définition de l'architecture d'un SMA et son déploiement au sein d'un environnement d'exécution donné, ce qui correspond à la prise en compte de l'ensemble des phases du cycle de développement des systèmes. Pour ce faire, il est nécessaire d'intégrer des approches et concepts issus de domaines distincts tels le génie logiciel pour les aspects méthodologiques et l'informatique répartie à objets pour les aspects architecturaux.

Nous nous attachons à la définition d'une telle méthodologie, qui repose pour les aspects architecturaux sur la norme ODP que nous présentons ci-après.

3. 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 [ISO10746-x].

3.1. 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[3] (*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.

3.2. 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 ;

---

[3] 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.

- **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 "avec quoi le faire ?" ou au "comment ?", cette fois-ci "comment" opérationnel ;
- **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.

3.2.1. Le point de vue "Entreprise"

Considérer un système ODP selon le point de vue entreprise permet d'écrire une spécification d'entreprise dudit système [ISO15414]. 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.

3.2.2. 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.

3.2.3. 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 [ISO14750] [IDL95].

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 [ISO14753].

3.2.4. 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** 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** 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** correspondant à une configuration d'objets d'ingénierie de base constituée à des fins de persistance, de sauvegarde et de migration.

Une transposition de cet agencement peut être faite avec les systèmes Unix : un nœud s'assimile à un site dont le noyau est le système d'exploitation, une capsule à un processus, l'objet d'ingénierie de base à un thread, une grappe contenant un ou plusieurs objets d'ingénierie de base et étant manipulée de façon atomique.

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.

3.2.5. 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.

3.3. Conclusion

L'expansion du traitement réparti a fait apparaître la difficulté de faire interagir des composants créés dans des environnements différents. Le besoin d'une activité de normalisation est né de cette constatation et de la nécessité de créer un modèle conceptuel permettant la définition d'une architecture de systèmes répartis en environnement hétérogène.

Le modèle de référence ODP fournit ce cadre architectural. Il détermine les concepts de l'architecture et les structures et règles de construction de systèmes qui s'y conforment. Il offre ainsi un ensemble de concepts rigoureux permettant de spécifier un système construit à partir d'objets répartis.

Aujourd'hui le modèle de référence a le statut de norme internationale. Il constitue un point d'entrée pour les travaux de l'OMG qui en sont très largement inspirés. Par ce biais, en constatant la large adhésion industrielle aux standards de l'OMG, il est légitime de penser qu'ODP a pleinement atteint son but : servir de référence à toute une communauté, celle de l'informatique répartie à objets. Il serait toutefois étroit de cantonner RM-ODP à cette unique communauté. En effet, le modèle objet de RM-ODP est suffisamment général pour être utilisé à différents niveaux d'abstraction. En particulier, il ne contraint pas la nature des objets ou leur granularité. Les objets peuvent être de taille arbitraire, de la valeur d'une donnée à un système informatique complet, au choix du modélisateur. Ainsi, la définition d'objet utilisée dans le modèle de référence est suffisamment ouverte pour s'adapter au paradigme agent. C'est sur cette considération que nous fondons notre approche pour définir une méthodologie de construction de systèmes à base d'agents, décrite dans le paragraphe suivant.

4. La méthodologie ODAC

Le développement de systèmes à base d'agent est une activité complexe qui ne peut être menée de façon empirique et doit être supportée par une démarche rigoureuse. Ces systèmes sont répartis au sens ODP du terme, c'est-à-dire qu'ils s'exécutent dans un contexte hétérogène technique et organisationnel. Ils sont constitués d'entités interagissantes qui peuvent être des agents, i.e., des objets actifs, mais aussi des objets au sens traditionnel du terme, i.e., objets passifs (cf. paragraphe 2.2). Le concept objet ODP inclut ces deux notions, puisqu'il définit qu'un objet exhibe un comportement composé d'actions, sans présumer de la façon dont ces actions peuvent être initiées. Dans notre optique, un agent est alors un objet ODP. La réciproque est fausse, i.e., tous les objets ODP ne sont pas des agents.

L'architecture d'un système à base d'agent peut alors être définie conformément aux concepts et règles de structuration des points de vue ODP. Cependant, le modèle de référence ODP fournit un cadre architectural et non une méthodologie. De plus, la norme est peu prescriptive et ne fournit aucun support pour le développement de systèmes. Par exemple, un langage est défini pour chaque point de vue, sous forme de concepts et règles, mais aucune notation ou formalisme n'est proposé pour concrétiser l'élaboration d'une spécification de système selon un point de vue.

Nous proposons alors une méthodologie orientée agent qui s'inscrit dans le cadre normatif ODP par l'utilisation de l'approche par points de vue et des concepts et règles qui s'y rattachent. Ceci permet de spécifier des systèmes à base d'agents selon la sémantique normative ODP. Nous y adjoignons l'utilisation de la notation standardisée UML qui permet la rédaction des spécifications de points de vue. Pour assister le développeur de systèmes dans sa démarche, nous fournissons un outil logiciel de spécification de systèmes.

Ce paragraphe décrit la méthodologie ODAC dans sa version actuelle. Elle s'appuie sur les trois points de vue ODP entreprise, information et traitement. La prise en compte du point de vue ingénierie fait l'objet du développement ultérieur de la méthodologie présentée en conclusion de ce paragraphe.

4.1. Les étapes de la méthodologie

La construction d'un système à base d'agents s'inscrit dans une démarche de génie logiciel incluant différentes étapes correspondant au cycle de vie du système. D'une façon générale, les modèles de cycle de vie d'un système sont composés de quatre phases qui sont la construction du système, son déploiement dans l'environnement réel d'exécution, son utilisation et son retrait. Chacune de ces phases définit un ensemble complet d'activités qui peuvent être appliquées au système. Nous focalisons notre méthodologie sur la phase de construction détaillée ci-dessous.

La phase de construction inclut les activités d'analyse des besoins, d'analyse puis de conception du système et enfin d'implantation dans un environnement cible reflétant au mieux l'environnement réel afin de procéder aux tests. Ces activités ne sont pas strictement séquentielles et des recouvrements ou des allers-retours peuvent se produire. De plus, la frontière entre ces activités est parfois un peu floue selon les méthodes, particulièrement en ce qui concerne l'analyse et la conception. Nous retenons par la suite les définitions mentionnées dans [OCC00], à savoir :

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'implémentation 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.

Un parallèle peut alors être établi entre ces activités et les points de vue ODP. Ainsi la méthodologie ODAC associe de façon informelle le point de vue entreprise à l'analyse des besoins, les points de vue information et traitement à l'analyse, le point de vue ingénierie à la conception et le point de vue technologie à l'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 identifiées : la *spécification comportementale* du système et sa *spécification opérationnelle*.

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'entreprise dans laquelle il est développé, des informations qu'il traite et des tâches qu'il accomplit.

La *spécification opérationnelle* résulte de la projection de la spécification comportementale sur 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.

Dans sa version actuelle, la méthodologie se focalise sur l'élaboration de la spécification comportementale par l'application des concepts et règles de structuration des points de vue entreprise, information et traitement. Les points de vue ne sont pas strictement ordonnés, cependant la méthodologie préconise de considérer en premier lieu le point de vue entreprise, puis les points de vue information et traitement. La description du système selon chacun de ces points de vue donne lieu à une spécification dans ce point de vue. L'ensemble des spécifications obtenues constitue la spécification comportementale du système. Nous détaillons ci-après le processus d'élaboration de cette spécification comportementale selon les trois points de vue ODP mentionnés.

4.2. Spécification d'entreprise

La spécification d'entreprise d'un système à base d'agents 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.

4.2.1. Concepts utilisés

Pour établir la spécification d'entreprise d'un système à base d'agents, nous préconisons l'utilisation des concepts suivants, qui forment un ensemble minimal des concepts ODP du point de vue entreprise. Nous jugeons que cet ensemble est pertinent pour exprimer la plupart des spécifications d'entreprise.

Objet d'entreprise

C'est un concept de base du point de vue entreprise. Un objet d'entreprise est le modèle d'une entité, celle-ci faisant partie de l'univers de discours (par exemple, des humains, des systèmes de traitement de l'information, des ressources de toutes sortes etc). Le système à spécifier est constitué d'agents et/ou d'objets. Dans le point de vue entreprise, ces agents et/ou objets sont globalement désignés sous le terme d'objets d'entreprise. Un objet d'entreprise a une identité, un état et un comportement. Cette définition semble alors imposer de décrire dans une spécification d'entreprise les objets identifiés qui composent le système, autrement dit les instances. En fait, les objets d'entreprise sont utilisés pour représenter différents degrés d'abstraction, selon le choix du modélisateur. Ainsi celui-ci peut très bien choisir de spécifier un objet d'entreprise par son type, ou par son gabarit.

Action

Une action est quelque chose qui se produit. Ce peut être une action interne ou une interaction. En général, on spécifie le type de l'action plutôt que l'action elle-même, i.e. son occurrence. Le terme "action" est d'ailleurs fréquemment employé à tort, en lieu et place de "type d'actions".

Comportement

Le comportement est une collection d'actions avec les contraintes qui sont associées à leur occurrence.

Communauté

Ce concept est fondamental dans une spécification d'entreprise. Quand des objets d'entreprise se groupent pour réaliser un objectif commun, ils forment une communauté. Un objet d'entreprise appartient au moins à une communauté et peut appartenir à plusieurs. Une communauté C1 peut être représentée comme un objet d'entreprise dans une autre communauté C2. Cet objet est appelé le c-objet. C'est une abstraction de la communauté C1 au sein de la communauté C2.

Objectif

Un objet d'entreprise appartient à une communauté uniquement s'il participe à la réalisation de l'objectif de cette communauté. Chaque communauté a son propre objectif. Les communautés ne partagent pas d'objectif, mais deux communautés peuvent avoir le même objectif. De plus, l'objectif d'une communauté peut constituer une part de l'objectif d'une autre communauté.

Rôle

Comme nous l'avons mentionné précédemment, un objet d'entreprise peut appartenir à différentes communautés. Dans chacune d'elles, il remplit un rôle. En effet, un ensemble de rôles est attaché à chaque communauté, chacun d'eux identifiant un comportement, i.e., un ensemble d'actions nécessaires pour l'accomplissement de l'objectif. Ainsi les objets qui remplissent ces rôles peuplent la communauté. Le cas particulier du c-objet conduit à la définition d'un rôle spécifique appelé "rôle d'interface". Ce rôle représente les services proposés par une communauté (sous la forme d'un c-objet) au sein d'une communauté plus large.

Politiques

Les politiques sont des règles qui s'appliquent aux objets d'entreprise, aux rôles et aux communautés.

Conclusion

Pour résumer :
- Une communauté a un objectif ;
- Pour réaliser cet objectif, un comportement doit être exhibé par la communauté. Celui-ci est identifié par des rôles ;
- Les objets d'entreprise qui remplissent un rôle dans une communauté peuplent la communauté ;
- Les politiques s'appliquent aux objets d'entreprise, aux rôles et aux communautés.

4.2.2. Rédaction d'une spécification d'entreprise

Élaborer une spécification d'entreprise consiste à :

1) Fixer l'objectif ;
2) Énumérer tous les rôles permettant de remplir cet objectif ;
3) De l'ensemble des rôles, séparer ceux qui sont des rôles de l'environnement du système de ceux qui sont des rôles du système. Ceci revient à identifier la S-communauté ;
4) Parmi les rôles du système, identifier les éventuels rôles d'interface, donc d'autres communautés. Attribuer alors à ces communautés les rôles qui s'y attachent ;

Pour chaque communauté :

5) Identifier les objets d'entreprise remplissant les rôles de la communauté ;
6) Décrire le comportement de la communauté ;
7) Décrire les politiques.

Pour réaliser ces différentes descriptions, nous proposons d'utiliser la notation standard UML de la façon suivante [BLA99].

Objectif -> Use Case

L'objectif d'une communauté est représenté par un Use Case (donc une ellipse [UML00]). En effet, un Use Case est défini comme une séquence d'actions qu'un système exécute et qui conduit à un résultat observable par l'environnement du système [BOO98]. Définir ce résultat en tant qu'énoncé d'un état futur que le système atteint suite à l'exécution d'un certain comportement est équivalent à fixer l'objectif de la communauté qui induira son comportement.

Rôle -> Stéréotype de classe "rôle"

Un rôle est un identifiant de comportement, celui-ci représentant un ensemble d'actions et de contraintes sur l'apparition de ces actions. Au vue de la définition d'opération dans UML, à savoir "une opération est une abstraction de quelque chose qu'un objet peut faire", une action ODP est comparable à une opération UML [BOO98]. À noter que cette définition d'opération n'implique pas du tout qu'opération UML et méthode soient synonymes, bien que ces deux termes soient fréquemment confondus. Nous représentons alors un rôle par un nouveau stéréotype de classe qui exprime une collection d'opérations. Une opération (i.e. action) peut être une interaction ou une action interne. Pour les distinguer, les noms des actions internes sont soulignés. Les contraintes sur les opérations sont représentées par des notes UML.

S-Communauté -> diagramme de Use Case

La S-communauté est représentée par un diagramme de Use Case. En effet, un tel diagramme est un graphe d'acteurs UML, un ensemble de Use Case et des relations entre ces éléments. Les Use Case peuvent être optionnellement encadrés par un rectangle qui représente les frontières du système. Un acteur UML définit un ensemble de rôles que les utilisateurs d'une entité jouent lorsqu'ils interagissent avec cette entité [UML00].

Or la S-communauté représente le système comme un unique objet d'entreprise interagissant avec son environnement. Dans cette notation de diagramme de Use Case, les rôles de l'environnement sont les acteurs UML et les Use Case représentent l'objectif du système. Il peut y avoir plusieurs Use Case puisqu'un objectif peut être décomposé en sous-objectifs. L'union des Use case forme alors l'objectif.

Objet d'entreprise -> objet UML

Un objet UML est défini de la même façon qu'un objet ODP, i.e., le modèle d'une entité avec une identité, un état et un comportement [BOO98]. Un objet d'entreprise sera donc représenté par un objet UML. Notons cependant qu'en général, une spécification d'entreprise ne manipule pas des instances, 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 qui réalise toutes les actions 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.

Nous avons également introduit qu'un objet d'entreprise peut être un agent. Le système à spécifier est donc composé d'agents et d'objets. Le modélisateur doit distinguer parmi les objets d'entreprise ceux qui sont des agents de ceux qui sont des objets en fonction de la nature de leur participation aux actions dans lesquelles ils sont impliqués. Cette distinction à faire selon les définitions données au paragraphe 2.2 relève donc d'un choix de modélisation.

Communauté -> collaboration et interaction

Une collaboration définit un ensemble de participants et les relations qu'ils ont pour un but donné. Elle identifie les rôles qui interagissent et leur relation. Au sein d'une collaboration, les objets interagissent pour accomplir un objectif. Une interaction UML reflète le comportement de la collaboration dédié à cet aspect de communication.

Ces notions recouvrent parfaitement le concept de communauté, ensemble d'objets d'entreprise qui interagissent pour remplir un objectif. Les objets d'entreprise qui sont des agents possèdent un ensemble de capacités et ont leur propre objectif. Cependant, dans le contexte d'une communauté à laquelle ils participent, ils doivent être capable de modifier leurs plans conduisant à la réalisation de leur objectif en fonction de l'objectif commun auquel ils ont accepté d'apporter leur contribution en entrant dans la communauté. Ainsi dans une communauté, chaque agent s'appuie sur ses propres capacités et sur celles que les autres agents mettent à sa disposition par le biais des interactions pour que l'objectif global soit atteint. Ceci fait apparaître une notion de dépendance entre agents, i.e., un agent remplit son rôle dans la communauté à condition que son environnement, donc les autres agents, lui fournissent les services nécessaires.

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. Dans le point de vue Entreprise, nous privilégions cet aspect en nous focalisant sur la description du comportement de communauté sous la forme d'un diagramme de séquences. L'aspect structurel sera quant à lui privilégié dans le point de vue Information.

Comportement de communauté -> diagramme de séquences

Le diagramme de séquence permet d'illustrer l'aspect comportemental d'une collaboration, donc d'une communauté en exprimant le déroulement temporel des interactions entre les objets d'entreprise. Le terme "interaction" est employé ici avec la sémantique ODP, c'est-à-dire une action à laquelle au moins deux objets d'entreprise participent. Il n'est donc pas synonyme d'invocation de méthode. En particulier, une interaction décrite dans un comportement de communauté peut être complexe. Elle peut être vue comme une interaction de haut niveau (ou protocole d'interaction) qui donnera lieu à un ensemble d'opérations, flux et signaux dans le point de vue Traitement. De plus, le point de vue Entreprise ne se soucie pas de la sémantique de communication d'une interaction en terme de messages bloquants ou non bloquants. Nous n'attachons donc aucune signification de ce type dans le graphisme des flèches que nous utilisons. Enfin, le modélisateur est libre de définir les protocoles d'interaction qui lui conviennent sous la forme d'un ensemble de services fournis entre agents avec une qualité de service associée et des règles de séquence permettant la réalisation de cette qualité [GER98]. Cependant, nous préconisons l'utilisation des protocoles d'interaction définis par FIPA avec les actes de communication associés [FIP2].

De plus, une interaction ODP n'est pas limitée à deux participants, ainsi plusieurs rôles peuvent être impliqués dans la même interaction, joués par le même agent ou non. L'extension à la notation des diagrammes de séquences UML proposée par [ODE00] pour prendre en compte des interactions concurrentes avec un agent jouant plusieurs rôles est alors utilisée en considérant que les rôles sont remplis par différents agents.

Politique -> Notes

Les politiques sont des contraintes ou des règles qui s'appliquent aux rôles, objets et communautés. Elles sont similaires à des annotations accolées aux rôles, objets et communautés, c'est pourquoi au premier stade de ces travaux, nous les représentons par les notes UML. L'expression des politiques par des langages de type OCL (Object Constraint Language) ou autres est envisagée dans la prochaine version de la méthodologie [OCL97].

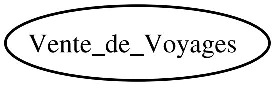
4.2.3. Spécification d'entreprise du système "Agence de voyages" : version 1

Nous illustrons dans ce paragraphe comment rédiger une spécification d'entreprise sur l'exemple de l'agence de voyages électronique proposé par FIPA [FIP4]. Cet exemple fait intervenir des clients qui, représentés par leur assistant personnel de voyages, achètent des voyages auprès d'une agence désignée par la suite par le terme "courtier de voyage". Le courtier est en liaison avec des prestataires de services de voyage (compagnies de transport, des hôtels, etc). Il sert donc d'intermédiaire entre l'assistant et les prestataires de services de voyage.

La spécification d'entreprise du système "Agence de voyages" est rédigée selon les étapes mentionnées au paragraphe 4.2.2. Elle est focalisée sur la vente d'un voyage.

Objectif

Le système à modéliser est une Agence de Voyages. Son objectif est la vente de voyages.



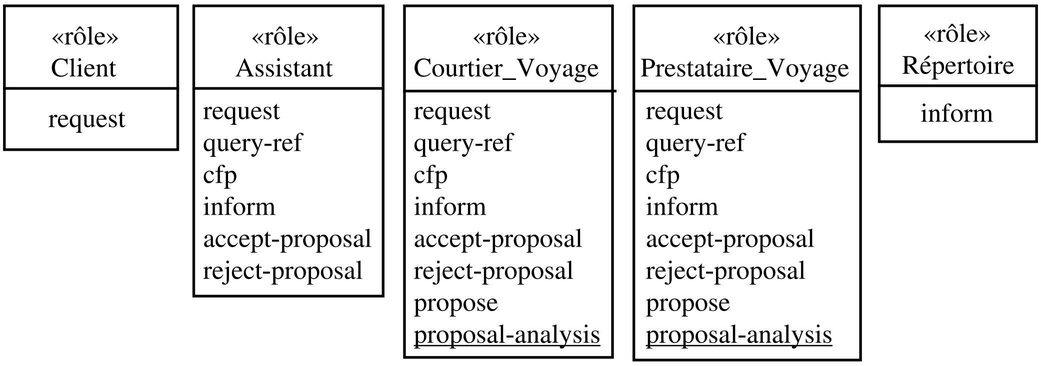
Rôles

Les rôles qui interviennent dans le cadre du système que l'on modélise sont :

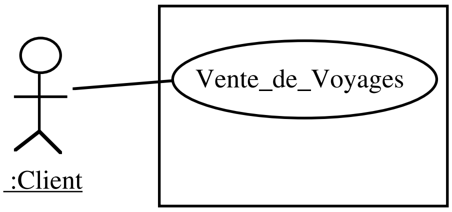
- Le client, qui énonce une requête pour un voyage ;
- L'assistant de voyage qui prend en charge la demande du client et qui réalise pour lui l'achat d'un voyage. Pour cela, il recherche dans un répertoire une liste de courtiers auxquels il va adresser la demande du client et avec lesquels il négociera les propositions pour aboutir à une réservation ;
- Le courtier de voyages qui sert d'intermédiaire entre l'assistant du client et les prestataires. Il transmet les demandes de l'assistant aux prestataires et lui redirige les meilleures réponses après analyse ;
- Le prestataire de voyages offre aux courtiers des propositions de voyage en fonction des demandes que celui-ci lui transmet ;

- Le répertoire est un annuaire contenant des listes de courtiers de voyages, i.e., des pages jaunes.

Les interactions de chaque rôle correspondent à des actes de communication FIPA.



S-Communauté



Parmi les rôles énumérés ci-dessus, nous identifions que le client est un rôle de l'environnement du système tandis que les autres rôles sont intrinsèques au système "Agence de Voyages".

Objets d'entreprise

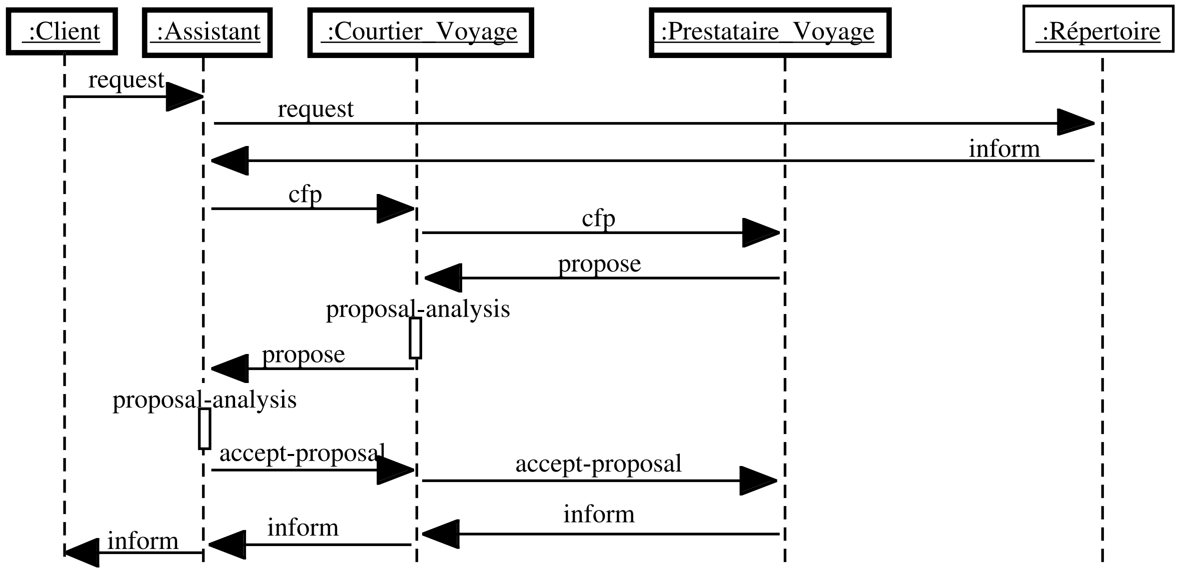
Nous choisissons d'utiliser des objets anonymes, ce qui permet de ne pas figer le système dans une configuration donnée. En particulier, cela n'impose aucune restriction sur le nombre d'objets d'entreprise jouant le rôle de courtiers, ou de prestataires par exemple. Ainsi, on peut imaginer que l'assistant s'adresse de façon concurrente à plusieurs objets d'entreprise remplissant le rôle de courtier et qu'il choisira sa réservation en fonction des différents résultats qu'il obtiendra. De même pour les courtiers vis-à-vis des prestataires de voyages.

D'autre part, nous choisissons que chaque objet d'entreprise ne joue qu'un seul rôle. Ceci est un choix de modélisation, rien n'empêcherait d'opter pour que le même objet d'entreprise remplisse le rôle d'assistant et de courtier, par exemple. Dans ce cas, la notation proposée dans [ODE00] pour le diagramme de séquences peut être adoptée.

Enfin, seul l'objet d'entreprise jouant le rôle du répertoire est choisi comme étant un objet. Tous les autres sont des agents. La distinction est marquée par l'épaisseur du trait du rectangle.

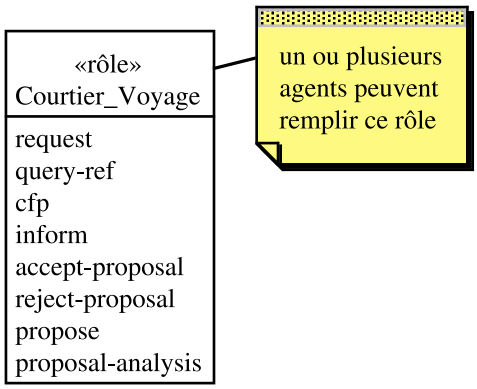
Comportement de communauté

Par souci de simplification, nous ne faisons figurer sur ce schéma que les cas positifs, en particulier en ce qui concerne le protocole Contract-Net dont nous utilisons la version non itérative. Cependant, chaque agent doit être en mesure de réaliser les différents protocoles utilisés, FIPA-Request, FIPA-Query et FIPA-Contract-net (itérative ou non) tels qu'ils sont décrits dans [FIP2]. Rappelons que dans notre approche, une flèche entre deux types d'agent ne fait qu'indiquer qu'il y a interaction au sens ODP du terme, sans plus. En aucun cas il ne faut y voir une notion d'invocation de méthode ou d'envoi de message synchrone.



Politiques

Les politiques s'appliquent sous forme de notes. Un exemple de politiques est celui de politique de population : comme nous l'avons indiqué précédemment, il est tout à fait possible que l'assistant s'adresse à plusieurs courtiers pour obtenir la meilleure offre. Ceci se traduit par la règle qu'il peut y avoir plusieurs agents jouant le rôle de courtier dans le système. Soit la modification du rôle courtier de la façon suivante :



D'autres exemples de politiques peuvent être définis, comme :

- Obligation pour le Courtier_Voyage de transmettre à l'Assistant la meilleure offre reçue des Prestataire_Voyage ,
- Interdiction pour un Prestataire_Voyage de proposer une offre au Courtier_Voyage supérieure au prix demandé
- Etc.

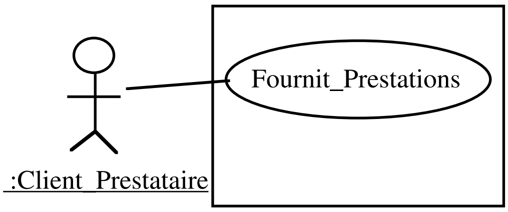
Conclusion

Dans cet exemple, nous avons considéré que le système et son environnement était composé d'objets d'entreprise appartenant tous à la même communauté. La spécification d'entreprise correspond donc à la spécification de cette communauté.

Or une spécification d'entreprise peut inclure plusieurs communautés dont les relations s'expriment à travers la notion de rôle interface. Nous illustrons un tel cas dans le paragraphe suivant.

4.2.4. Spécification d'entreprise du système "Agence de voyages" : version 2

Nous considérons dans ce même exemple du système "Agence de voyages" que les prestataires de service appartiennent à une corporation et qu'il existe une spécification d'entreprise du système "Prestataires_Services" dont nous ne retenons ici qu'un extrait, à savoir la description de la S-communauté.



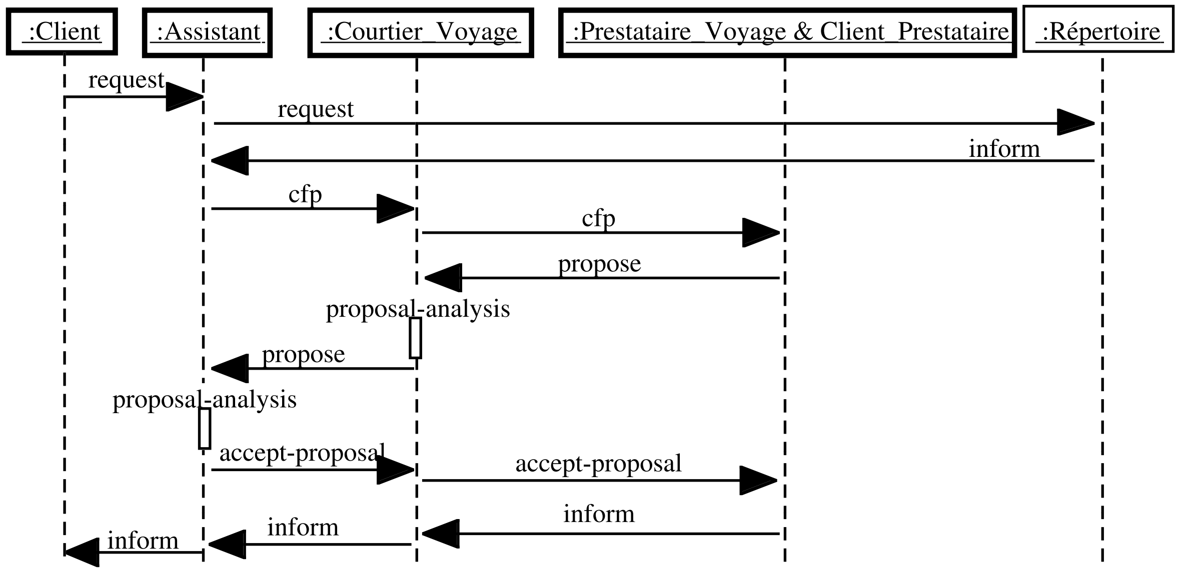
Le modélisateur du système "Agence de voyages" peut alors utiliser cette spécification d'entreprise pour rédiger sa propre spécification d'entreprise de la façon suivante :

1) La détermination de l'objectif est identique à celle du paragraphe précédent.
2) La détermination des rôles est également identique. Toutefois il n'est pas nécessaire de représenter le rôle "Prestataire_Voyage".
3) La S-communauté du système "Agence de voyages" reste la même que précédemment. En effet, la S-communauté représente le système comme un seul objet d'entreprise, donc une vue générale du système qui ne permet pas de faire apparaître les rôles d'interface.
4) Cette étape n'a pas été nécessaire dans l'exemple précédent. Ici, par contre, elle est indispensable. Elle permet d'indiquer que le rôle "Prestataire_Voyage" identifié en 2) est un rôle d'interface, attaché à la communauté "Prestataires_Service".

Pour cette communauté, les étapes 5 à 7 sont déjà réalisées puisque la spécification existe. La rédaction de la spécification du système "Agence de voyages" se termine alors de la façon suivante :

5) L'identification des objets d'entreprise remplissant les rôles de la communauté se fait de la même façon que précédemment excepté pour le rôle d'interface "Prestataire_Voyage". Ce rôle représente les services que la communauté "Prestataires_Service" fournit dans la communauté "Agence de Voyages". L'objet d'entreprise remplissant ce rôle est appelé c-objet. Il appartient aux deux communautés, ce qui se traduit par le fait qu'il joue le rôle "Prestataire_Voyage" pour la communauté "Agence de Voyage" et le rôle "Client_Prestataire" dans la communauté "Prestataires_Service". C'est de cette façon que le lien est signifié entre les deux communautés. Elles interagissent via cet objet d'entreprise remplissant un rôle distinct dans chacune d'elles.
6) Le comportement de la communauté "Agence de Voyage" est identique, exception faite du c-objet. Ainsi les interactions avec cet objet d'entreprise sont en fait traitées par la communauté correspondante sous-jacente.
7) Les politiques sont représentées comme précédemment.

La spécification complète du système inclut donc tous les diagrammes relatifs à la communauté "Agence de Voyage" et le diagramme de Use Case représentant la S-Communauté "Prestataires_Service". La spécification complète de "Prestataires_Service" peut être inclue de façon optionnelle.



4.3. Spécification d'information

Une spécification d'information inclut les trois schémas définis dans ce point de vue, à savoir le schéma d'invariant, le schéma statique et le schéma dynamique. Ceux-ci peuvent être mis en correspondance relativement aisément avec la notation UML.

Ainsi, le schéma d'invariant est représenté par un diagramme de classe dans lequel on ne fait pas figurer les opérations. Le schéma statique et le schéma dynamique sont tous deux représentés à partir d'un diagramme d'états. Le schéma statique ne fait figurer que les détails des états sans représenter les changements d'états alors que le schéma dynamique reflète les changements d'états mais pas le détail des états.

Le paragraphe suivant illustre la spécification d'information du système "Agence de voyages".

4.3.1. Spécification d'information du système "Agence de voyages"

Nous ne fournissons pas tous les diagrammes qui constitueraient la spécification complète. Seuls ceux relatifs à l'assistant de voyage sont mentionnés.

Schéma d'invariants

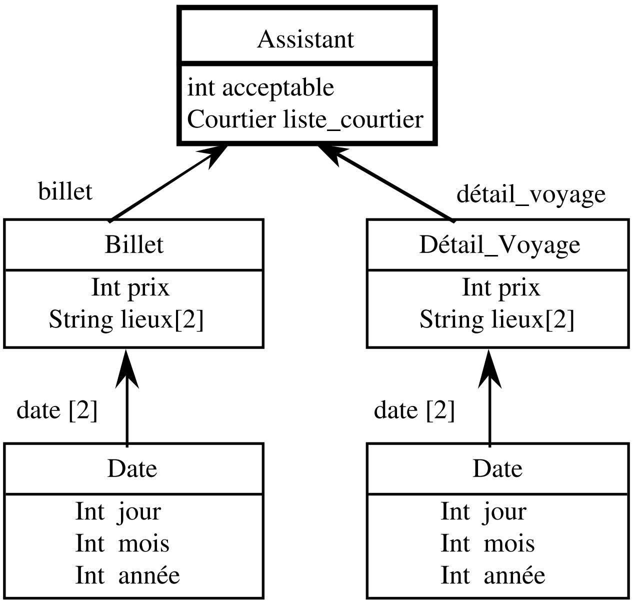


Schéma statique

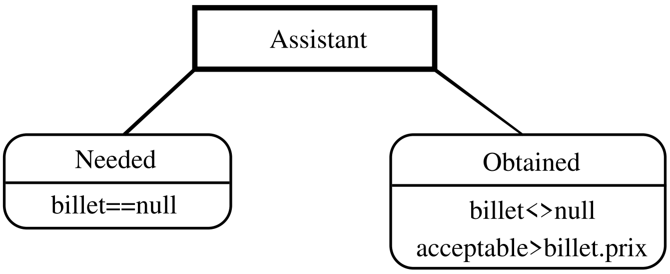
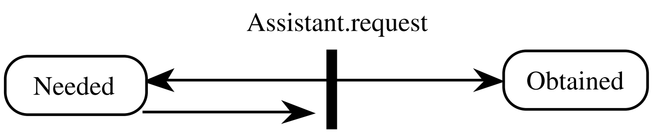


Schéma dynamique



4.4. Spécification de traitement

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 : le système est composé d'agents et d'objets qui par leurs interactions atteignent l'objectif. Ainsi ce point de vue s'intéresse à cette décomposition fonctionnelle induisant des interactions, mais fait abstraction de la répartition des agents/objets et par conséquent, des mécanismes supportant les interactions. Ceux-ci sont pris en compte dans le point de vue ingénierie.

Il est bien évident que les spécifications d'entreprise, d'information et de traitement doivent être cohérentes puisqu'elles décrivent le même système. Cependant, cela n'implique pas que les objets identifiés dans un point de vue soient mis en correspondance de façon un-un avec ceux d'un autre point de vue. Par exemple, tout objet de traitement ne correspond pas forcément à un (et un seul) objet d'entreprise. Tout dépend jusqu'à quel niveau de détail le modélisateur a décrit dans la spécification d'entreprise le "pour quoi" du système. Si cette description est très précise, elle peut alors correspondre à la description du "comment faire", donc à la décomposition fonctionnelle du point de vue Traitement. Dans ce cas, un objet de traitement peut être directement mis en correspondance avec un objet d'entreprise. Cette même considération s'applique pour la mise en correspondance des objets d'information et ceux de traitement.

Une fois les objets de traitement identifiés, la spécification de traitement s'attache alors à l'identification des interactions, à l'écriture des interfaces correspondantes en IDL ainsi qu'à la description des comportements qui y sont associés à l'aide de diagrammes d'activité UML. Actuellement, seules les interactions décrites en IDL sont considérées, i.e., les interrogations bloquantes et les annonces. Cependant, la prochaine version inclura non seulement les interrogations non bloquantes, mais également les flux et les signaux et pour prendre en compte ceux-ci, l'écriture des interfaces sera en IDL-like[4]. La description du comportement nécessitant alors de faire apparaître différentes formes d'interactions, il est envisagé d'utiliser une notation autre qu'UML. Celle-ci devrait également supporter la validation formelle. SDL (Specification and Description Language) est un exemple de notation qui pourrait être utilisée.

4.4.1. Spécification de traitement du système "Agence de voyages"

Pour cet exemple, nous supposons qu'à chaque objet d'entreprise correspond un et un seul objet de traitement. Par souci de simplification et de clarté, nous nous limitons ici à la présentation de l'objet de traitement " :Assistant". Rappelons que celui-ci est décrit dans le point de vue Entreprise comme un agent qui traite les demandes de voyages émanant d'un utilisateur et négocie avec un courtier les offres de voyage. Nous nous intéressons à ce traitement selon le protocole FIPA-Contract-Net tel que mentionné dans la spécification d'entreprise. Nous considérons que cette interaction du point de vue Entreprise correspond à une relation client-serveur dans le point de vue Traitement. L'assistant est client du courtier de voyages qui lui fournit d'une part le service de proposition de voyages ("cfp" suivi de "propose") et d'autre part le service de réservation de voyages ("accept_proposal" suivi de "inform"). Aussi nous assimilons ces deux services à deux opérations qui sont des interrogations (invocation suivie d'une réponse) que nous appelons respectivement Courtier_Proposal et Courtier_Reservation.

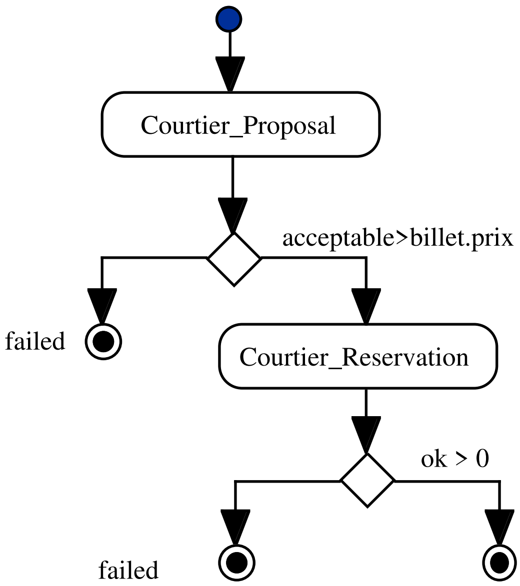
Ainsi un objet de traitement de type Assistant utilise via son interface cliente les interrogations Courtier_Proposal et Courtier_Reservation qui sont fournies par un objet de type Courtier_Voyage. Celui-ci exporte donc ces deux interrogations via son interface serveur. Une déclaration IDL-like d'une telle interface est requise, de la forme :

```
interface negotiation {
 void Courtier_Proposal {
       in int Détail_Voyage.prix,
       in string Détail_Voyage.lieux,
       out int Billet.prix,
       out string Billet.lieux
 };
 void Courtier_Reservation {
```

[4] Nous considérons une extension d'IDL telle que ODL proposée par le consortium TINA pour prendre en compte les flux et les signaux et pour que non seulement les exportations mais également les importations d'interface soient décrites [ODL96].

```
            in int Billet.prix,
            in string Billet.lieux,
            out boolean ok
    };
  };
```

Le comportement de l'assistant est alors décrit par le diagramme suivant sur lequel figurent les deux interactions mentionnées ci-dessus ainsi que l'action interne proposal-analysis identifiée dans la spécification d'entreprise :



4.5. Conclusion

Les trois spécifications obtenues forment la spécification comportementale du système, c'est-à-dire sa description indépendamment de tout environnement cible d'exécution. Elles constituent un ensemble de modèles qui décrivent le système en tant qu'organisation et ses composantes en tant qu'agents et objets munis de leurs interactions. Il est intéressant de noter qu'en adoptant l'approche par points de vue, on sépare les préoccupations lors de la spécification et ainsi, on ne fait pas apparaître un modèle d'agent en tant que tel avec ses états mentaux, ses buts et son comportement comme c'est fréquemment le cas dans les méthodologies agent (cf. paragraphe 2.5). Ces différents éléments se retrouvent dans les trois spécifications et c'est la mise en correspondance des objets de chaque point de vue qui détermine ce qu'est un agent. Ainsi un objet de traitement dont on décrit le comportement joue le rôle de l'objet d'entreprise auquel il correspond et donc réalise les buts de cet objet avec les états mentaux de l'objet d'information auquel il correspond.

Pour compléter la méthodologie, il convient de prendre en considération un environnement cible d'exécution et d'y projeter la spécification comportementale pour obtenir la spécification opérationnelle. Celle-ci décrit le système en fonction de son environnement. Elle s'appuie sur le point de vue ingénierie ODP. Les concepts de ce point de vue sont utilisés pour décrire une plate-forme à agents mobiles [MUS 00]. Ils permettent également d'expliciter la correspondance entre les agents/objets identifiés dans la spécification comportementale et les objets d'ingénierie de base apparaissant dans la spécification opérationnelle. Ceux-ci fournissent une description suffisamment détaillée pour la génération de code et son implantation. La rédaction d'une spécification opérationnelle nécessite une notation qui est en cours de définition.

5. Conclusion

Pour faire face à la complexité grandissante des applications réparties, le développeur utilise des abstractions qui lui permettent de décrire le système le plus naturellement possible, c'est-à-dire proche du raisonnement et de la réflexion humaine, en s'affranchissant des contraintes pratiques induites par la programmation d'ordinateur.

C'est la raison pour laquelle le paradigme agent rencontre une grande adhésion parmi les concepteurs de systèmes complexes et ce, quel que soit le domaine d'applications. Il reste cependant difficile à utiliser dans le développement de grands projets par le manque de méthodologie qui prenne en compte l'ensemble du cycle de vie qui inclut de nombreuses activités alliant aspects de modélisation et architecturaux. Or trop souvent, les méthodologies proposées ne s'intéressent qu'à un aspect en particulier sans tenir compte de l'ensemble.

La méthodologie ODAC a l'ambition de pallier cet inconvénient. Pour cela, elle se fonde sur l'utilisation de la norme ODP qui fournit un cadre architectural pour développer des applications réparties. L'intérêt d'ODP dans la démarche de construction d'un système à base d'agents réside dans le concept de point de vue qui permet de séparer les préoccupations et de décrire un système sous un aspect donné en faisant abstraction de ce qui ne relève pas de cet aspect. L'ensemble des spécifications de chaque point de vue fournit alors une description exhaustive dudit système en prenant en compte tous les aspects nécessaires à son développement.

La méthodologie dans sa version actuelle se focalise sur les étapes de définition des besoins et d'analyse d'un système. Elle s'appuie sur les points de vue Entreprise, Information et Traitement de la norme ODP en prescrivant un mode d'utilisation des concepts et des règles définis dans ces points de vue ainsi que l'usage de la notation UML (complétée par IDL pour le point de vue Traitement). Le résultat de son application permet d'obtenir la spécification comportementale du système. Comme nous l'avons souligné dans le paragraphe 4, la prochaine version de la méthodologie est en cours d'élaboration et devrait inclure des améliorations comme un langage d'expression des politiques et le complément des interactions du point de vue Traitement lié à l'utilisation d'une autre notation. D'autre part, la méthodologie sera étendue pour prendre en compte l'étape de conception, c'est-à-dire les aspects opérationnels qui s'expriment dans le point de vue Ingénierie. Cette étape permettra d'obtenir la spécification opérationnelle du système à partir de laquelle une implantation pourra être réalisée par génération de code. De ce fait, la méthodologie ODAC couvrira la phase de construction dans le cycle de vie d'une application répartie.

Un autre aspect de la méthodologie que ce document n'a pas couvert est celui de la réutilisation. En vue d'optimiser l'élaboration de spécifications de systèmes à base d'agents, il paraît souhaitable de pouvoir intégrer des spécifications existantes. Se pose alors le problème de l'interopérabilité de spécifications en vue de leur réutilisation. Pour le résoudre, nous proposons une approche similaire à celle développée par le standard CORBA et le langage IDL. IDL fournit un mode universel et uniforme de description d'objets hétérogènes. De ce fait, il masque l'hétérogénéité sous-jacente et les objets peuvent alors interopérer via un ORB. Nous appliquons ce principe non plus aux objets (code) mais aux spécifications (modèles). Ainsi un concepteur accédera à la spécification d'un composant pour l'intégrer dans la spécification de sa propre application. De même que l'interopérabilité des objets est assurée par l'IDL, celle des spécifications nécessite un langage universel. ODP, en tant qu'ensemble de concepts et de règles de structuration, nous fournit un tel langage. Ainsi nous fournissons un service d'échange de spécifications RM-ODP [BLA00]. L'étape ultérieure de cette démarche consiste à mettre en correspondance les concepts ODP avec ceux de différents formalismes utilisés en modélisation. Ainsi nous proposerons des règles de transformation d'une spécification source établies dans un formalisme quelconque pour obtenir la spécification ODP correspondante, puis une spécification cible. La mise en relation des concepts ODP avec la notation UML dans les points de vue Entreprise, Information et Traitement a permis d'identifier les questions clés de l'interopérabilité de spécifications, à savoir les risques de pertes lors de la traduction et les limitations de la réutilisation. Cependant, le succès des bus logiciels pour l'interopérabilité des objets est un argument majeur pour étendre cette approche au niveau des spécifications.

6. Bibliographie

[ATG 99] OMG Agent Working Group, *Agent Technology - Green Paper*, Document ec/99-12-02 - version 0.9, December 1999

[ATG 00] OMG Agent Working Group, *Agent Technology - Green Paper*, Document ec/2000-03-01 - version 0.91, March 2000

[BLA 00] X. Blanc, M.P. Gervais and R. Le Delliou, *The Specifications Exchange Service of an RM-ODP Framework*, in Proceedings of the 4th International Enterprise Distributing Object Computing Conference (EDOC'00), IEEE Press (Ed), Makuharin, Japan, September 2000, pp86-90.

[BLA 99] X. Blanc, M.-P. Gervais and R. Le Delliou, *Using the UML Language to Express the ODP Enterprise Concepts*, In Proceedings of the 3rd International Enterprise Distributed Object Computing Conference (EDOC'99), IEEE Press, Mannheim, Germany, September 1999, pp50-59.

[BOO 98] G. Booch, J. Rumbauch and I. Jacobson, *The Unified Modeling Language User Guide,* Addison-Wesley, 1998

[BOU 97] T. Bouron, *Les agents dans l'Internet*, Actes des 5èmes Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'97) - Conférence invitée, Hermès, La Colle-sur-Loup, France, avril 1997, pp9-10.

[BRA 97] F. M. T. Brazier et al., *DESIRE : Modelling multi-agent systmes in a compositional framework*, International Journal of Cooperative Information Systems, Vol. 1, n° 6, pp67-94, January 1997

[BUR 96] B. Burmeister, *Models and Methodology for Agent-Oriented Analysis and Design*, In Proceedings of the Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems, DFKI Document D-96-06, Dresden, Germany, September 1996,

[CAM 97] V. Camps et M. P. Gleizes, *Une technique multi-agent pour rechercher des informations réparties*, Actes des 5èmes Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'97), Hermès, La Colle-sur-Loup, France, avril 1997, pp29-46.

[CHE 95] D. Chess et al., *Itinerant Agents for Mobile Computing*, IEEE Personal Communications, Vol. 2, n° 5, pp34-49, October 1995

[CLI 98] *CLIMATE*, www.fokus.gmd.de/research/cc/ima/climate,

[DEM 97] Y. Demazeau, *Steps Towards Multi-Agent Oriented Programming*, First International Workshop on Multi-Agent Systems IWMAS'97, Boston, 1997

[DRO 98] A. Drogoul and A. Collinot, *Applying an Agent-Oriented Methodology to the Design of Artificial Organizations : A Case Study in Robotic Soccer*, Journal of Autonomous Agents and Multi-Agent Systems, Vol. 1, n° 1, pp113-129, 1998

[ESF 96] B. Esfandiari, G. Deflandre and J. Quinqueton, *An interface agent for network supervision*, 12th European Conference on Artificial Intelligence (ECAI'96), Budapest, Hongrie, August 1996.

[FAL 97] B. Falchuck and A. Karmouch, *AgentSys: A Mobile Agent System for Digital Media Access and Interaction on an Internet*, In Proceedings of the IEEE Globecom'97, Phoenix, USA, November 1997, pp1876-1880.

[FER 95] J. Ferber, *Les systèmes multi-agents,* InterEditions, 1995

[FER 98] J. Ferber and O. Gutknecht, *A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems*, In Proceedings of the the 3rd Int. Conference on Multi-Agent Systems (ICMAS'98), IEEE Computer Society, 1998, pp128-135.

[FIP] *FIPA*, www.fipa.org,

[FIP OS] *FIPA-OS*, http://fipa-os.sourceforge.net/

[FIP 1] FIPA, *Specification Part 1 - Agent Management*, FIPA97 ver 1.0, June 1997

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

[FIP 3] FIPA, *Specification Part 3 - Agent/Software Interaction*, FIPA97 ver 1.0, June 1997

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

[FIS 97] M. Fischer and M. Wooldridge, *On the Formal Specification and Verification of Multi-Agent Systems*, International Journal of Cooperative Information Systems, Vol. 6, n° 1, pp37-65, January 1997

[GER 98] M. P. Gervais and A. Diagne, *Enhancing Telecommunications Service Engineering with Mobile Agent Techology and Formal Methods*, IEEE Communications Magazine, Vol. 36, n° 7, pp38-43, July 1998

[GLA 96] N. Glaser, *Contribution to Knowledge Modelling in a Multi-Agent Framework (the CoMoMAS Approach)*, Thèse de l'Université Henri Poincaré, Nancy I, November 1996

[GLA 99] G. Glass, *Overview of Voyager : ObjectSpace's Product Family for State of the Art Distributed Computing*, CTO Object Sapce, www.objectspace.com, 1999

[GRA] *Grasshopper, A platform for mobile software agents*, IKV++ GmbH, www.ikv.de/products/grasshopper,

[GRE 97] S. Green et al., *Software Agents : A Review*, IAG (Trinity College Dublin and Broadcom Eireann Research Ltd), www.broadcom.ie/~fs/agents.html, May 1997

[GUT 99] O. Gutknecht et J. Ferber, *Vers une méthodologie organisationnelle de conception de systèmes multi-agents*, Actes des Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'99), Hermès, 1999,

[HAR 95] C. G. Harrison, D. M. Chess and A. Kershenbaum, *Mobile Agents: Are they a Good Idea?*, URL: http://www.research.ibm.com/massdist/, IBM T.J. Watson Research Center, March 1995

[IDL95] OMG's CORBA 2.0 Specification, *CORBA Interface Definition Language (IDL)*, 1995

[IGL 97] C. A. Iglesias et al., *Analysis and Design of Multiagent Systems Using MAS-CommonKADS*, In Proceedings of the 4th International Workshop on Agent Theories, Architectures and Languages (ATAL'97), LNAI n°1365 - Springer Verlag, Providence, Rhode Island, USA, July 1997, pp313-327.

[IGL 98] C. A. Iglesias, M. Garijo and J. C. Gonzalez, *A survey of agent-oriented methodologies*, In Proceedings of the 5th International Workshop on Agent Theories, Architectures and Languages (ATAL'98), LNAI n°1555 - Springer Verlag, Paris, France, July 1998, pp317-330.

[ISO10746-1] ISO/IEC IS 10746-1 — ITU-T Rec. X901, *ODP Reference Model Part 1: Overview and Guide to Use*, May 1996

[ISO10746-2] ISO/IEC IS 10746-2 — ITU-T Rec. X902, *ODP Reference Model Part 2: Foundations*, January 1995

[ISO10746-3] ISO/IEC IS 10746-3 — ITU-T Rec. X903, *ODP Reference Model Part 3: Architecture*, January 1995

[ISO10746-4] ISO/IEC IS 10746-4 — ITU-T Rec. X904, *ODP Reference Model Part 4: Architectural Semantics*, 1996

[ISO15414] ISO/IEC CD 15414, *ODP Reference Model : Enterprise Viewpoint*, January 2000

[ISO 14750] ISO/IEC IS 14750, *Open Distributed Processing - Interface Definition Language*, 1999

[ISO 14753] ISO/IEC FDIS 14753, *Open Distributed Processing - Interface references and Binding*, 1999

[ITU G851] ITU-T Study Group 15 - Contribution, *Text of Draft Recommendation G.851-01*, June 1996

[JEN 00] N. Jennings, *On agent-based software engineering*, Artificial Intelligence, n° 117, pp277-296, 2000

[JEN 96] N. Jennings and M. Wooldridge, *Software Agents*, IEE Review, pp17-20, January 1996

[JEN 98] N. Jennings, K. Sycara and M. Wooldridge, *A Roadmap of Agent Reasearch and Development*, Autonomous Agents and Multi-Agent Systems, Vol. 1, n° 1, pp275-3061998

[KIN 96a] D. Kinny, M. Georgeff and A. Rao, *A methodology and modelling technique for systems of BDI agents*, in Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), LNAI n°1038 - Springer Verlag (Ed), 1996

[KIN 96] D. Kinny and M. Georgeff, *Modelling and Design of Multi-Agents Systems*, In Proceedings of the Intelligent agents III : agent theories, architectures, and languages : ECAI'96 Workshop (ATAL'96), LNAI n°1193 - Springer Verlag, Budapest, Hungary, August 1996

[KIS 96] R. Kishimoto, *Agent Communication System for Multimedia Communication Services*, In Proceedings of the IEEE Infocom, San Francisco, USA, March 1996, pp10-17.

[KOT 97] D. Kotz et al., *Agent TCL: Targeting the Needs of Mobile Computers*, IEEE Internet Computing, Vol. 1, n° 4, pp58-67, July-August 1997

[KOZ 96] B. Kozbe et al., *The Requirements for Personal Mobile Assistants in a Mobile Telecommunication Environment*, In Proceedings of the Intelligent Agents for Telecommunication Application Workshop (IATA'96) of the 12th European Conference on Artificial Intelligence (ECAI'96), Budapest, Hongrie, August 1996,

[KWA 96] W. Kwan and A. Karmouch, *Multimedia Agents in a Distributed Broadband Environment*, In Proceedings of the IEEE International Conference on Communications (ICC'96), Dallas, USA, June 1996, pp1123-1127.

[LAN 97] D. B. Lange and M. Oshima, *Java Agent API: Programming and Deploying Aglets Using Java,* Addison-Wesley, 1997

[MAG 96] T. Magedanz, K. Rothermel and S. Krause, *Intelligent Agents : an Emerging Technology for Next Generation Telecommunications ?*, In Proceedings of the IEEE Infocom, San Francisco, USA, March 1996, pp464-472.

[MAS 97] *Mobile Agent System Interoperability Facilities Specification*, OMG TC Document orbos/97-10-05, OMG, November 1997

[MER 96] M. Merz and W. Lamersdorf, *Agents, Services and Electronic Markets: How do they Integrate?*, In Proceedings of the IFIP/IEEE International Conference on Distributed Platforms (ICDP'96), Dresden, Germany, March 1996,

[MER 97] W. Merlat, *JavaNetAgent: une plate-forme d'exécution d'agents mobiles pour le développement de systèmes multi-agents sur Internet*, Actes des 5èmes Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'97) - Poster Session, Hermès, La Colle-sur-Loup, France, avril 1997,

[MON 96] J. Montelius, S. Janson and J. Gabrielsson, *Intentions and Intelligent Screening in an Agent-based Personal Communication System*, 12th European Conference on Artificial Intelligence (ECAI'96), Budapest, Hongrie, August 1996.

[MOU 96] B. Moulin and M. Brassard, *A scenario-based design method and environment for the development of multiagent systems*, In Proceedings of the 1st Australian Workshop on Distributed Artificial Intelligence, LNAI n°1087 - Springer Verlag, 1996, pp216-231.

[MUL 98] J. P. Müller, *Vers une méthodologie de conception de systèmes multi-agents de résolution de problèmes par émergence*, Actes des Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'98), Hermès, 1998, pp355-371.

[MUS 00] F. Muscutariu and M.-P. Gervais, *Modeling an OMG-MASIF Compliant Mobile Agent Platform with the RM-ODP Engineering Language*, in Proceedings of the 2nd International Workshop on Mobile Agents for Telecommunication Applications (MATA'00), Lecture Notes in Computer Science n°1931, Springer Verlag (Ed), Paris, France, September 2000, pp133-141

[NWA 96] H. S. Nwana, *Software Agents : an Overview*, The Knowledge Engineering Review, Vol. 11, n° 3, pp1-40, September 1996

[NYG 96] K. Nygren, I. M. Jonsson and O. Carlvik, *An Agent System for Media on Demand Services*, First International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96), Londres, UK, April 1996.

[OCC 00] 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, 2000

[OCL 97] Rationale Software Corporation, *Object Constraint Language Specification*, Version 1.1., September 1997, www.rational.com/uml/

[ODE 00] J. Odell, H. Van Dyke Parunak and B. Bauer, *Extending UML for Agents*, AOIS Workshop at AAAI'00, http://www.jamesodell.com/publications.html, 2000

[ODL96] TINA Object Definition Language Manual, Version 2.3, July 1996, TR_NM_002_2.2_96

[OLI 96] R. Oliveira and J. Labetoulle, *From Intelligent Agents towards Management by Request*, In Proceedings of the 4th International Conference on Intelligence in Networks (ICIN'96), Bordeaux, France, November 1996, pp30-34.

[PER 96] S. Perret and A. Duda, *MAP: Mobile Assistant Programming for Large Scale Communication Networks*, In Proceedings of the IEEE International Conference on Communications (ICC'96), Dallas, USA, June 1996, pp1128-1132.

[ROS 96] B. Rosenbach and J. Soref, *RMON: The Enterprise Management Standard*, Data Communications Magazine , March 1996

[RUM 91] J. Rumbaugh, *Object Oriented Modeling and Design,* Prentice Hall, 1991

[SCH 94] A. Schreiber et al., *CommonKADS : A comprehensive methodology for KBS development*, Deliverable DM1.2a KADS-II/M1/RR/UvA/70/1.1, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels, 1994

[SCH 99] M. Schroeder, *Are distributed objects agents ?*, International Workshop on Agent-Oriented Information Systems (AOIS'99) - position paper, 1999.

[STR 96] M. Straßer, J. Baumann and F. Hohl, *Mole - A Java Based Mobile Agent System*, 2nd ECOOP'96 Workshop on Mobile Object Systems, Linz, Austria, July 1996.

[UML 00] *OMG Unified Modeling Language Specification,* Version 1.3, Mars 2000

[VAN 96] F. Van Aeken and Y. Demazeau, *When Agents Talk - How to Maintain Integrity*, In Proceedings of the Intelligent Agents for Telecommunication Application Workshop (IATA'96) of the 12th European Conference on Artificial Intelligence (ECAI'96), Budapest, Hongrie, August 1996

[WHI 94] J. E. White, *Telescript Technology: the Foundation for the Electronic Marketplace*, http://www.genmagic.com/Telescript/Whitepapers/, General Magic Inc., 1994

[WHI 96] J. E. White, *Telescript Technology: Mobile Agents*, http://www.genmagic.com/Telescript/Whitepapers/, General Magic Inc., January 1996

[WIR 90] R. Wirfs-Brock, B. Wilkerson and L. Wiener, *Design Object-Oriented Software,* Prentice Hall, 1990

[WOO 94] M. Wooldridge and M. Fisher, *Agents-Based Software Engineering*, Manchester Metropolitan University, 1994

[WOO 95] M. Wooldridge and N. R. Jennings, *Intelligent Agents: Theory and Practice*, The Knowledge Engineering Review, Vol. 10, n° 2, pp115-1521995

[WOO 98] M. Wooldridge. *Agents and software engineering*. In AI*IA Notizie XI(3), pages 31-37, September 1998

[WOO 99] M. Wooldridge, N. Jennings and D. Kinny, *A methodology for agent-oriented analysis and design*, In Proceedings of the 3rd International Conference on Autonomous Agents (Agents'99), ACM Press, Seattle, WA, USA, 1999, pp69-76.

[WOO 00] M. Wooldridge, N. R. Jennings, and D. Kinny, *The Gaia Methodology for Agent-Oriented Analysis and Design*, In Journal of Autonomous Agents and Multi-Agent Systems, 3(3):285-312, 2000

[YEM 91] Y. Yemini, Network Management by Delegation, in *Integrated Network Management II,* Krishnan and Zimmer (Eds.), Editions Elsevier Science Publisher, 1991

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