INGENIAS : Développement Dirigé par Modèles des Systèmes Multi-Agents

Dossier d'Habilitation à Diriger des Recherches

de l'Université Pierre et Marie Curie

Spécialité : Informatique

Juan PAVÓN

Universidad Complutense Madrid

Présentée le 7 décembre 2006 devant le Jury composé de :

Président du Jury :	Yves Demazeau
Rapporteurs :	Marie-Pierre Gleizes Christian Lemaitre Francisco Garijo
Examinateurs :	Amal El Fallah-Seghrouchni Massimo Cossentino

Préambule

Ce mémoire décrit ma principale activité de recherche depuis mon incorporation à la Universidad Complutense Madrid (UCM) en tant que *Profesor Titular* (Maître de Conférences) depuis la fin 1997. Pendant tout ce temps j'ai créé et dirigé une ligne de recherche qui centre son activité sur le développement de la technologie d'agents et ses applications. Le groupe *grasia!* (*GRupo de Agentes Software : Ingeniería y Aplicaciones*) se compose actuellement de deux maîtres de conférences et de six assistants (donc trois docteurs et trois thésards doctorants), tous affectés au Département *Ingeniería del Software e Inteligencia Artificial* de l'UCM, ainsi que trois doctorants et un nombre variable d'étudiants de master (en moyenne trois). Les activités de notre groupe se sont développées dans le cadre de plusieurs projets de recherche, aussi bien sur le plan national qu'européen, subventionnés par des fonds publics et privés. La collaboration avec le secteur industriel montre son intérêt pour ce type d'activités et la capacité de transfert technologique de notre groupe de recherche.

Même si la plupart des projets ont eu une forte part expérimentale, il y a aussi une ligne de travail plus formelle, centrée sur les aspects méthodologiques pour le développement de Systèmes Multi-Agents (SMA). Le résultat est INGENIAS, un cadre de développement dirigé par modèles des SMA. INGENIAS part du fait que le paradigme agents apporte des éléments de modélisation pour la conception des systèmes complexes qui vont plus loin que d'autres paradigmes, comme les objets. La modélisation avec des agents permet d'approcher les concepts du domaine d'application plus facilement. Par exemple, avec un SMA il est possible de simuler une organisation humaine et les processus qui s'exécutent régulièrement. Des agents peuvent aussi modéliser des entités autonomes avec un comportement rationnel, basé sur des buts, des croyances, et ont une capacité pour établir des plans d'action. Mais à part la modélisation, il faut considérer la réalisation finale des systèmes, qui peut être sur différentes plates-formes, non seulement orientées agents, mais aussi objets ou composants (J2EE ou .NET, par exemple). Pour cela nous avons développé un ensemble d'outils qui permettent la transformation des spécifications (modèles) des SMA en code pour les plates-formes finales.

Ainsi, INGENIAS promeut un développement dirigé par modèles, dont l'implémentation est dérivé automatiquement par des transformations.

Ce dossier présente le contexte et les principaux résultats de la méthodologie INGENIAS, actuellement développée dans le cadre d'un projet national du même nom. Le chapitre 1 fait une révision de différentes approches pour le développement des SMA, pour identifier la problématique et montrer nos contributions dans le domaine. Les éléments de base de INGENIAS pour le développement dirigé par modèles sont décrits dans les chapitres suivants. La base de INGENIAS est la définition de méta-modèles, structurés en plusieurs aspects, qui décrivent les éléments qui permettent de spécifier des SMA (chapitre 2). Les outils, le INGENIAS Development Kit (IDK), sont générés à partir de ces méta-modèles. Le chapitre 3 décrit comment les outils ont été construits pour permettre l'édition des spécifications des SMA et la génération de code sur plusieurs plates-formes. Ces outils ont permis d'élaborer un processus de développement qui considère la définition des transformations et la génération de code, INGENIAS-MDD (chapitre 4). Ce cadre nous a permis de réaliser d'autres contributions dans le domaine des SMA, qui sont décrits dans différentes publications. Le chapitre 5 décrit quelques uns de ces travaux, en cours de réalisation, comme l'application de INGENIAS pour la simulation des systèmes sociaux et la vérification des propriétés sociales dans les SMA, ou la définition de métriques pour le développement de logiciel orienté agents. Finalement, le chapitre **¡Error! No se encuentra el origen de la referencia.** présente le résumé de mon parcours scientifique et professionnel.

Glossaire des abréviations

AAII	Australian Artificial Intelligence Institute
ACL	Agent Communication Language
ACM	Association for Computing Machinery
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
AUML	Agent Unified Modelling Language
BDI	Believes, Desires, Intentions
CASE	Computer Aided Systems Engineering
CBR	Case Based Reasoning
CORBA	Common Object Request Broker Architecture
FIPA	Foundation for Intelligent Physical Agents
IDK	INGENIAS Development Kit
ISO	International Standards Organization
IST	Information Society Technologies Programme
J2EE	Java 2 Enterprise Edition
KQML	Knowledge Query Manipulation Language
LIP6	Laboratoire d'Informatique de Paris 6
MDA	Model Driven Architecture
MDD	Model Driven Development (Développement dirigé par modèles)
MDE	Model Driven Engineering
MOF	Meta-Object Facility

ODP	Object Distributed Processing
OMG	Object Management Group
PIM	Platform Independent Model
PSI3	Personalised Systems Integration Using Software Agents
PSM	Platform Specific Model
QVT	Query/View/Transformation
SMA	Système Multi-Agents
TA	Théorie de l'Activité
TINA-C	Telecommunications Information Networking Architecture Consortium
UCM	Universidad Complutense Madrid
UML	Unified Modelling Language
XML	Extensible Mark-up Language

Table des matières

1. INTRODUCTION GENERALE

Pendant la dernière décennie s'est développée l'hypothèse que si les programmes qui composent les systèmes distribués étaient intelligents, s'ils pouvaient réorganiser leurs fonctions pour corriger ou tolérer les erreurs et si leur coordination pouvait se structurer en termes intuitifs, alors ces systèmes seraient plus faciles à élaborer et maintenir, plus adaptables et plus fiables.

Dans la construction de ces programmes s'applique une technologie intimement liée à l'intelligence artificielle. Il s'agit des *agents* qui, pour le moment, s'entendent comme des programmes autonomes avec la capacité d'interagir entre eux. Du point de vue de cette technologie, les systèmes distribués deviennent des Systèmes Multi-Agents (SMA). La conception de SMA, généralement, est abordée en pensant aux agents comme *entités avec motivation*. Au lieu de modéliser un système avec composants qui exécutent des méthodes, le développeur doit penser à ses buts. Les buts de système guident la structuration de celui ci en composants qui vont collaborer pour les satisfaire. De cette façon, on espère que le processus de développement soit plus intuitif puisque cette forme de modéliser et de raisonner est plus proche de la pensée humaine que les paradigmes de programmation traditionnelle.

La construction de SMA intègre des méthodes et techniques de différentes aires de connaissance : du génie logiciel pour structurer le processus de développement, d'intelligence artificielle pour doter les programmes de capacité pour traiter des situations imprévues et prendre des décisions, et de la programmation concurrente et distribuée pour traiter la coordination de tâches exécutées dans différentes machines. Étant donnée cette combinaison de technologies, le développement de SMA se complique. Il existe des plates-formes de développement qui donnent des solutions partielles à la modélisation de comportement et à la coordination des agents. La panoplie de ces solutions va de fournir des services basiques (gestion d'agents, librairies d'algorithmes, localisation d'agents, communications ou mobilité), comme JADE [8] ou ABLE [98], des primitives organisationnelles en MadKit [93], jusqu'à des

outils de développement où se paramétrisent des structures software (*frameworks*), comme ZEUS [130] ou agentTool [180].

Bien qu'elles facilitent le processus, les plates-formes de développement restent incomplètes sans un processus de développement qui permettrait la création des SMA dans un système de production de software industriel. Les techniques usuelles d'ingénierie, comme le Processus Unifié [104], ne tiennent pas compte des aspects particuliers des SMA, comme la spécification de planification de tâches, l'organisation du système, l'échange d'information avec des langages de communication orientés agents, la mobilité du code ou la motivation des composants du système. Pour cela, sont envisagées de nouvelles méthodologies basées sur des agents [81]. Ces méthodologies partent d'un modèle, généralement informel, de comment doit être un SMA et donnent des guides pour sa construction. Lors des premières méthodologies, les guides consistaient en une liste brève des pas à suivre. Les plus modernes, bien qu'elles aient progressés grâce à l'intégration de techniques du génie logiciel classique, doivent s'améliorer sur plusieurs aspects, comme, par exemple, la gestion de tout le cycle de vie du développement, l'utilisabilité et la robustesse des outils de support, faciliter le travail en équipe, ou l'adéquation du langage de spécification de SMA dans divers domaines d'application. Récemment se sont produites beaucoup d'avancées dans tous ces aspects, cependant il manque l'intégration des résultats des différentes réponses méthodologiques. C'est ainsi qu'il est fondamental de considérer les efforts qui sont faits dans FIPA ou dans AgentLink pour standardiser les concepts, les notations et les méthodes [11].

Pour mieux comprendre le problème nous allons dans un premier temps caractériser les SMA (section 1.1), comme nous les entendons. Il existe beaucoup d'acceptions du terme selon les groupes de recherche et nous verrons que notre réponse se situe dans une ligne d'acception largement admise. Puis dans la section 1.2 nous verrons dans quel type d'applications il est approprié d'utiliser la technologie d'agents. C'est ainsi que, connaissant les divers aspects de ce que l'on veut développer, nous pourrons réviser les techniques et méthodologies qui existent actuellement (section 1.3), pour déterminer le contexte et les hypothèses qui nous ont amenés à développer la méthodologie INGENIAS (section 1.4). Plus concrètement, nous considérerons les principales contributions scientifiques d'INGENIAS dans le domaine de génie logiciel orienté agents (section 1.5).

1.1 Agents et Systèmes Multi-Agents

Le terme *agent* est largement utilisé et on peut trouver une multitude de définitions. Dans le premier *European Agent Systems Summer School* (EASSS'99), organisé par *AgentLink*, où se réunirent plusieurs experts sur le sujet, apparut la disparité des critères. C'est ainsi que cela se reflète dans les actes [179]:

- Michael Wooldridge: *encapsulated computer system, situated in some environment, and capable of flexible autonomous action in that environment in order to meet its design objectives.*
- Michael Huhns: *an active computational entity with: (a) persistent identity, (b) that can perceive reason about, and initiate activities in its environment, (c) that can communicate (with other agents).*
- Stephan Covaci: *a computer program that acts autonomously on behalf of a person or organisation and has its own thread of execution*.
- Henry Lieberman: (un *agent d'interface* est) *software that is autonomous, learns, is context-sensitive, presents human-like appearance, has specialised knowledge ... Agent does not have to satisfy all of them ...*

De ces définitions, et d'autres qui apparaissent dans la littérature, comme par exemple [30][95] [61] [173] [181], nous pouvons identifier plusieurs propriétés qui caractérisent les agents :

- Autonomie : Les agents peuvent travailler sans l'intervention directe de l'usager et ont un certain contrôle de leurs actions et état interne.
- Réactivité : Les agents peuvent percevoir leur environnement, qui peut inclure le monde physique, un usager derrière une interface graphique, des applications sur le réseau, ou d'autres agents, et répondre à des changements qui se produisent dans celui-ci.
- Initiative : Le comportement des agents est déterminé par les buts qu'ils poursuivent et par conséquent ils peuvent produire des actions qui ne sont pas seulement des réponses à leur environnement.
- Habilité sociale : Pour satisfaire ses buts un agent peut demander la collaboration d'autres agents à qui il délègue ou avec lesquels il partage la réalisation de tâches. Pour cela il a besoin de se coordonner, négocier, en définitive, interagir.

- Raisonnement : Un agent peut décider quel but poursuivre ou à quel événement réagir, comment agir pour accomplir un but, ou suspendre ou abandonner un but pour se dédier à un autre.
- Apprentissage : L'agent peut s'adapter progressivement à des changements dans des environnements dynamiques grâce à des techniques d'apprentissage.
- Mobilité: Dans des applications déterminées il peut être intéressant de permettre aux agents de migrer d'un noeud à un autre dans un réseau tout en préservant leur état lors de sauts entre noeuds.

Il est intéressant de noter, comme cela est montré dans la Figure 1 [181], qu'il y a trois lignes de recherche dans l'aire des agents, selon que l'on donne plus ou moins d'importance aux aspects cognitifs (raisonnement, apprentissage), sociaux (communication et coopération), ou de mobilité (agents mobiles).

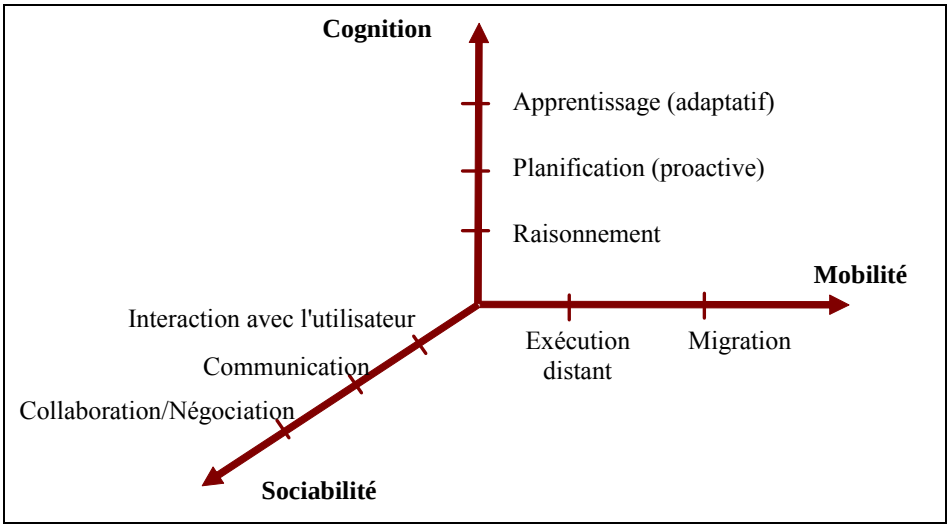


Figure 1. Caractérisation des agents

En général, les travaux sur les agents mobiles laissent de côté les caractéristiques sociales et cognitives et se centrent sur le fondement de modèles de migration et de navigation des agents dans un réseau, et d'autres problèmes associés comme celui de la sécurité. Bien qu'il y ait beaucoup d'initiatives sur cet aspect (voir par exemple sur le site web http://www.cetus-links.org/oo_mobile_agents.html) dans nos travaux il nous a paru suffisant d'utiliser un modèle basé sur la communication par messages. La disponibilité de systèmes d'agents mobiles globalement est difficile, à cause, dans la pratique, du besoin de disposer de plates-formes homogènes pour l'exécution des agents (qui exigent, par exemple, un même modèle de cycle de vie pour les agents) dans tous les noeuds du réseau où veulent se déplacer les agents, en plus des problèmes qu'entraînent la sécurité et la gestion des systèmes d'agents mobiles (suivi et

contrôle des agents, entre autres). En général, une solution basée sur la coopération des objets distribués sur un *middleware* standard avec la possibilité de charge dynamique de code (par exemple, du style des *applets*, qui ne sont pas des agents mobiles puisqu'ils n'ont pas besoin de transfert d'état) nous a semblé suffisante pour résoudre de manière efficace les problèmes qui se posent usuellement en matière de paradigme agent mobile. Pour cette raison, dans notre proposition nous n'allons pas considérer la mobilité.

Les aspects cognitifs déterminent à quel point le comportement de l'agent peut être complexe. Un des aspects les plus valorisés dans la provision de services est celui de la personnalisation. Pour doter un système de cette caractéristique, il est nécessaire, d'une part, la persistance de l'information qui caractérise les événements qui se produisent dans le système et, d'autre part, la capacité de traiter la dite information afin de déterminer des actions spécifiques en fonction du nouveau contexte. La combinaison de mécanismes de raisonnement, de planification et d'apprentissage confère différents niveaux d'intelligence au système (c'est-à-dire la capacité d'adaptation à un environnement dynamique). De cette manière, la *programmation orientée agents* (terme déjà utilisé par Shoham [168]) diffère de la programmation orientée objets, étant donné qu'un objet n'a pas le contrôle sur son comportement puisque la décision sur la méthode à exécuter est prise par le client qui l'invoque. La programmation des agents doit définir quels sont les buts que poursuit l'agent et comment il va déterminer les tâches qui doivent être réalisées pour les satisfaire.

Par ailleurs, les aspects sociaux sont l'élément différenciateur des systèmes basés sur les agents par rapport à d'autres paradigmes du domaine de l'intelligence artificielle, comme les systèmes experts. Ceci est, probablement, une des principales raisons qui a motivé l'intérêt dans ce domaine ces dernières années : la décomposition du problème de gestion de connaissance en traitements plus simples, mais coordonnés. C'est ainsi que, certains auteurs comme Demazeau [45] et Ferber [56] considèrent cet élément comme essentiel et qu'une entité n'est pas un agent si elle n'est pas une partie d'un système multi-agents (SMA). Un SMA donne la capacité de résoudre des problèmes conjointement entre plusieurs agents de manière différente de la planification classique. D'après cette perspective, un agent s'entend comme une entité autonome qui communique avec d'autres entités, possiblement hétérogènes, en un SMA. Dans ce sens, Huhns et Singh [95] proposent une preuve qui s'appuie sur les relations entre agents pour déterminer si un système est ou non un SMA :

> *A system containing one or more reputed agents should change substantively if another of the reputed agents is added to the system*

Le changement de la *preuve d'agents de Huhns-Singh* est une amélioration du rendement du SMA, ce qui implique que les agents doivent être conscients de l'apparition de nouveaux agents. Bien qu'il soit possible de concevoir des systèmes avec un seul agent, comme les assistants personnels, dans lesquels un agent agit comme un expert qui aide l'usager à réaliser certaine tâche avec l'ordinateur, le cas multi-agents est plus général et plus intéressant du point de vue du génie logiciel. Les SMA sont adéquats pour représenter des problèmes qui ont plusieurs méthodes de solution, de multiples perspectives où interviennent de multiples entités. En plus des avantages comme la distribution et la concurrence dans la solution, ils permettent d'utiliser des patrons d'interaction sophistiqués de coopération (travail en commun pour atteindre un but commun), de coordination (organisation du processus de solution du problème de manière à éviter des interactions nocives et à exploiter celles bénéfiques) et de négociation (formulation d'un accord qui soit acceptable par toutes les parties impliquées). C'est précisément la flexibilité et la nature de haut niveau de ces interactions qui distinguent et donnent de l'importance à un SMA par rapport à d'autres paradigmes du génie logiciel.

Pour notre travail, les agents ne sont pas seulement des composants distribués et réactifs (ceux-ci seraient alors seulement des processus distribués). Ils nous intéressent davantage pour leurs caractéristiques *sociales* (les agents peuvent connaître dynamiquement d'autres agents, arriver à interagir avec eux, et même modifier avec le temps les formes de communication) et *cognitives* (le comportement des agents peut être assez complexe et peut requérir la prise de décisions de forme autonome). C'est ainsi que, quand nous parlons d'un système à base d'agents nous nous référons à un Système Multi-Agents (SMA), avec plusieurs types d'agents, où peuvent apparaître de nouveaux types d'agents dynamiquement, et avec un nombre aussi variable d'agents de chaque type.

Certains agents peuvent être assez simples, de nature réactive, mais il est normal que quand un système manifeste un certain comportement intelligent (par exemple, la capacité de planifier, la prise de décisions, la délégation de tâches, la décomposition des buts, la recherche d'alternatives aux échecs) il existe un agent avec un modèle cognitif puissant. Pour programmer ces comportements il est nécessaire d'utiliser quelque chose de plus que les langages impératifs. Bien qu'à la base pour construire nos systèmes d'agents nous utilisons le langage Java, nous intégrons d'autres paradigmes comme les systèmes basés en règles (par exemple, JESS ou ILOG JRules) ou de raisonnement basés sur des cas (CBR, *Case Base Reasoning*).

De manière schématique, le type de SMA que nous considérons est formé par :

- Des *agents*, de plusieurs types, qui coopèrent pour se procurer des services adaptables et personnalisés. Chaque agent est un système encapsulé, avec des limites bien définies,

capable d'une action autonome et flexible dans son environnement pour atteindre ses buts de conception.

- Des *ressources* qui peuvent être gérées par des agents ou par un certain système propriétaire. Ces ressources constituent une partie de l'environnement des agents. Une ressource peut être un serveur web, une base de données, un serveur de courrier, un système de gestion électronique des processus métier (*workflow management system*), selon le domaine d'application.
- D'un *middleware* pour supporter la communication entre les agents, mais aussi avec des systèmes propriétaires. En nous référant au *middleware* nous incluons les services associés, comme la localisation, les notifications et la sécurité. Le *middleware* peut supporter des standards de communication des agents comme FIPA ACL ou KQML, ou définir des interfaces spécifiques pour chaque agent avec un langage comme OMG IDL.

Dans la conception des SMA, à tous ces aspects déjà mentionnés il faut ajouter ceux qui sont inhérents à la distribution des agents dans un réseau, importants du point de vue du génie logiciel :

- la concurrence, avec les propriétés de sécurité et vivacité ;
- le passage à l'échelle, pour maintenir à un prix raisonnable la qualité du service à des changements de la demande ;
- la disponibilité, ce qui exige la robustesse et la tolérance aux pannes ;
- la maintenance, pour l'adaptation aux nouveaux besoins et la gestion de versions ;
- la globalité d'accès, à travers de différents dispositifs ;
- la configuration, pour adapter le système aux nécessités du client et aux ressources disponibles ;
- la mesurabilité, par exemple pour facturer l'utilisation du système ou pour étudier le degré d'acceptation des services ;
- l'efficacité et la rationalité dans l'usage des ressources.

1.2 Applications des SMA

Depuis les premières expériences à la fin des années 80 (par exemple, un système de gestion de fabrication [142] basé sur le protocole *contract net* [171]), une grande diversité de systèmes ont été construits en utilisant des agents. Celles-ci vont des applications simples, comme les

assistants de courrier électronique, aux systèmes complexes, comme le contrôle du trafic aérien [117]. Un aspect intéressant pour l'application de la technologie d'agents est précisément que le concept d'agent permet d'une façon naturelle de modéliser une gamme de systèmes aussi ample et avec des besoins aussi divers.

Dans [105] s'est discuté précisément quels types d'applications étaient appropriés pour être construits avec des agents. En tenant compte du fait que s'agissant d'une nouvelle technologie, on devait solutionner des problèmes qui avant n'étaient pas automatisables (parce qu'il n'existait pas la technologie capable de donner une solution au problème ou bien parce que la solution existante était trop chère à appliquer), ou améliorer significativement (en coût, rapidité ou facilité) le développement de systèmes qui pourraient s'implémenter avec les technologies existantes.

Le type de systèmes les plus simples à réaliser sont les *fonctionnels*, qui produisent une sortie à partir d'une entrée en appliquant une fonction ou un algorithme plus ou moins complexe. Ils n'assument pas des changements dans l'environnement pendant son exécution. Les systèmes *réactifs,* cependant, maintiennent une interaction continue avec l'environnement, dont les changements peuvent faire que les préconditions valables au début de l'exécution d'un processus ne le soient plus durant son exécution et même que le but pour lequel a commencé un processus cesse d'avoir une justification. C'est dans ce dernier type de systèmes où se trouvent des cas de réalisation difficile: des systèmes ouverts (où la structure du système peut changer de manière dynamique, comme c'est le cas du web où continuellement apparaissent de nouveaux outils et services), des systèmes complexes (avec divers aspects qui nécessitent une grande spécialisation pour résoudre chacun d'entre eux) et d'accessibilité (*the future of computing will be 100% driven by delegating to, rather than manipulating computers* [128]).

Par rapport à l'amélioration de l'efficacité que peut laisser supposer l'adoption d'un système basé sur les agents, Bond et Gasser [22] envisagent trois aspects qui seraient adéquats :

- Distribution inhérente de données, contrôle, expérience ou ressources. Il est typique des systèmes de travail collaboratif [113], par exemple pour la gestion d'un hôpital [94] où les données que manipule le médecin ne sont pas les mêmes que celles de l'infirmière, chaque individu est responsable d'un ensemble de tâches, la connaissance spécialisée est différente selon qu'il s'agisse d'une infirmière ou d'un médecin, et la responsabilité de gestion des ressources est répartie en divers postes. A chaque rôle on peut associer un agent avec son expérience, ses responsabilités et ses moyens, et les relations entre les différents rôles permettent de définir les processus de gestion de l'hôpital.

- La notion d'agent autonome permet de modéliser de façon naturelle une fonctionnalité logicielle spécifique. Par exemple, l'assistant personnel de courrier électronique comme un agent spécialisé pour aider l'usager dans une tâche, ou les jeux en réseau où chaque personnage peut se représenter avec un agent (*bot*).
- Le système comporte des composantes héritées qui doivent interagir entre elles et avec de nouvelles composantes logicielles. Pour cela on propose de recouvrir et de gérer l'accès aux applications avec agents (par exemple, dans [72]).

D'autres exemples d'applications basées sur les agents peuvent se trouver dans [32] [135] [153] [154]. Une révision plus récente de l'état de l'art et des applications de la technologie des agents, depuis la perspective d'AgentLink apparaît dans [119].

1.3 Méthodologies de développement de SMA

Malgré l'existence de beaucoup d'applications basées sur les agents, il manque encore l'expérience pour la conception et la construction de SMA à un niveau industriel. La grande majorité des applications basées sur les agents est construite sans utiliser des composants agents réutilisables et ne sont pas généralisables. C'est pour cela que la recherche sur les méthodes, les outils et les plates-formes d'agents a beaucoup d'importance pour l'implantation de la technologie d'agents en dehors du domaine purement académique. Dans la situation actuelle, peu de méthodologies ont amené des cas réels significatifs à la pratique et sont assistées par des outils. En plus, les méthodologies existantes ne considèrent que certains aspects du cycle de vie, généralement l'analyse et la conception. Pour l'implémentation, la plupart des méthodologies sont conditionnées par l'utilisation d'une architecture d'agents déterminée. Disposer d'une méthodologie consistante, utile et prouvée permettra de profiter des bénéfices qu'apporte la technologie d'agents, de la même manière que les méthodologies, tel que le Processus Unifié, et les patrons de conception ont permis d'exploiter de manière efficace les caractéristiques du paradigme objets. Le propos de disposer d'une méthodologie est de changer la pratique de construction du logiciel « *comme ça vient* » par un processus d'ingénierie bien structuré qui produise du logiciel de qualité, tenant en compte des restrictions d'un ensemble limité de ressources et adhérant à une planification prévisible.

Selon l'expérience de ses créateurs, les différentes propositions méthodologiques pour développer des SMA partent des résultats du domaine du génie logiciel orienté objets (ADELFE [150], AAII/BDI [156], Kendall [110], MaSE [42], Gaia [183], MASB [127], ODAC [74], MESSAGE [31]) ou appliquent des techniques de construction de systèmes experts (CoMoMAS

[77], MAS-CommonKADS [100]). Certains mettent plus d'emphase sur les aspects organisationnels, comme AALAADIN [55] ou sur les relations entre les divers aspects du SMA, comme dans VOYELLES [44], INGENIAS [144] ou MASSIVE [116]. Il existe aussi des raisonnements plus formels, basés sur le langage Z [118] ou sur la logique temporelle (Concurrent METATEM [60] et DESIRE [28]). Bien que ces derniers ne soient pas facilement applicables, ils sont intéressants pour les solutions qu'ils apportent à des problèmes spécifiques, (par exemple, les communications et l'organisation d'agents, la modélisation de la définition d'agents) et pour vérifier les applications critiques ou les modèles de coopération complexes.

Dans pratiquement tous les cas, y compris ceux qui ont des fondements dans les techniques d'intelligence artificielle ou dans les modèles formels, les méthodologies de développement des SMA élargissent le modèle d'orientation objets. Cela ne doit pas surprendre puisqu'ils existent assez de points communs entre le modèle d'agents et celui d'objets (à un tel point pour que certains professionnels la différence entre ces concepts n'est pas encore claire), et pour la plus grande facilité d'acceptation que peuvent avoir pour son utilisation les ingénieurs en logiciel, maintenant familiarisés avec le paradigme d'objets.

Les extensions du concept d'agent par rapport à celui d'objet qu'il faut prendre en compte sont variées :

- Autonomie des agents. Contrairement aux objets, les agents peuvent décider exécuter ou non une action sollicitée par un autre agent, ou même négocier de quelle manière va se réaliser une tâche. Un objet, par contre, est déterminé pour exécuter la méthode qui a été choisie : *Objects do it for free; agents do it because they want to* [182]. Un agent est une entité dont le comportement est dirigé par des buts et cela peut déterminer dans quelles situations il décide d'agir ou non selon les événements de son environnement. Les agents étendent l'encapsulation des objets (état et implémentation du comportement) en cachant aussi l'activation du comportement et l'élection des actions. Cet aspect est assez important dans les environnements ouverts comme le Web où coexistent des applications très diverses et où on doit contrôler et négocier l'accès aux services fournis par des entités différentes, ce qui exige l'implantation de politiques et de stratégies variées.

- Chaque agent a son propre flux de contrôle, tandis qu'un objet, normalement, agit selon le flux de contrôle de celui qui l'invoque : Un objet est passif et s'active quand il reçoit une requête pour exécuter une de ses méthodes. Mise à part l'utilisation d'objets spéciaux, comme par exemple ceux de la classe *Thread* de Java, une application orientée objets est généralement conçue comme un flux de contrôle séquentiel où vont

se réaliser des opérations sur des objets passifs. Dans un système d'agents, par contre, on considère de multiples flux d'exécution indépendants avec des interactions concurrentes. La modélisation de cet aspect est une des extensions qu'il serait nécessaire d'aborder en UML pour spécifier un SMA [134].

- Le comportement des agents a une nature réactive, sociale et dirigée par des buts. Selon le type d'agent chacun de ces aspects aura plus ou moins d'importance, mais tous sont considérés de forme intégrée, quelque chose que l'on ne prend pas en compte en général dans le modèle d'objets.

  De même que les objets, les agents ont une identité, un état et un comportement, mais se décrivent par des termes plus sophistiqués. L'*état mental* de l'agent est formé par la connaissance, les croyances et les buts qu'il doit satisfaire. Le comportement est défini par les rôles qu'il peut jouer, les actions qu'il peut réaliser et les réactions aux événements. Ceci veut dire que le comportement de l'agent ne se spécifie pas par ce qu'il doit faire pour atteindre une sortie à partir d'une entrée mais par *comment décider* de ce qu'il doit faire.

- Les interactions entre agents sont plus sophistiquées qu'une simple communication par messages. Entre objets l'invocation des méthodes se traite à un niveau symbolique (nom de la méthode, valeurs des paramètres), mais entre agents les interactions ont lieu au *niveau de connaissance* [129]. Ce qui veut dire qu'elles se conçoivent en fonction des buts que les agents poursuivent, au moment ou elles se produisent. L'intentionnalité détermine les différents types de messages qui se modélisent comme les actes de langage (*speach acts*) [4] et se cadrent dans de multiples protocoles de négociation entre agents.

- Un SMA considère de façon explicite les aspects d'*organisation du système.* Les agents interagissent dans un contexte qui définit la nature des relations entre eux. L'interaction sociale signifie que les relations peuvent changer, avoir une durée temporelle limitée et se créer de manière dynamique.

Tous ces aspects requièrent, par conséquent, une modélisation des systèmes basés sur les agents d'un point de vue différent de celui qui est considéré avec les objets. Il existe des propositions qui abordent l'extension d'UML pour représenter certaines des caractéristiques des agents, comme la concurrence des interactions ou la notion de rôle, c'est le cas d'Agent-UML [134]. Mais il se contente simplement de traiter certains aspects de notation et non de méthodologie. Les méthodologies proposées jusqu'à maintenant, bien qu'elles identifient plusieurs de ces aspects, sont théoriques (n'ont pas été expérimentées avec des cas réels) ou bien abordent

seulement une partie de la problématique (par exemple, elles ne considèrent pas tout le cycle de vie, ou l'organisation, ou l'intentionnalité des agents).

Quant au conception du comportement des agents, un des modèles qui a eu le plus d'influence, est celui du BDI (*Beliefs, Desires, Intentions*), initialement abordé dans le domaine de la philosophie [27]. Bratman proposait que les interactions jouent un rôle significatif dans le raisonnement pratique, comme les plans d'actions partiels qu'un agent est prêt à exécuter pour atteindre ses buts (désirs), tenant compte de ses croyances (la vision du monde). Cette approche a été utilisée pour définir des architectures d'agents délibératifs [73][156]. Leur application a donné lieu à la méthodologie de développement de SMA de l'Australian Artificial Intelligence Institute (AAII) qui étend le paradigme des objets avec le modèle BDI [111]. La méthodologie AAII considère que la spécification d'un SMA doit se réaliser aussi bien d'un point de vue externe, qui montre le système composé par des agents (caractérisés par leur propos, leurs responsabilités, les services qu'ils réalisent, l'information qu'ils requièrent et maintiennent, et leurs interactions externes), que d'un point de vue interne, où se modélisent les éléments pour une architecture d'agents particuliers, dans ce cas les croyances, les buts et les plans de l'agent. Du point de vue externe, un *modèle d'agents* définit une hiérarchie de classes d'agents, les responsabilités des agents, et les services qu'ils offrent. Les interactions associées et les relations de contrôle entre agents sont définies dans *un modèle d'interactions*. Le processus de construction de ces modèles est fondé sur l'analyse des rôles du domaine de l'application, à partir duquel se détermineront les agents du système et leur caractérisation, bien que la relation rôle-agent ne soit pas univoque, mais que les agents comprennent un ensemble *de services* initialement associés aux responsabilités des rôles. L'architecture BDI, sur laquelle est basé le point de vue interne des agents, est reflétée dans trois modèles: le *modèle des croyances,* qui décrit l'information sur l'environnement, l'état interne que peut garder l'agent et les actions qu'il peut réaliser, le *modèle des buts*, qui décrit les buts que peut adopter l'agent et les événements auxquels il peut répondre, et le *modèle de plans*, qui décrit les plans (séquences d'actions) que peut employer l'agent pour atteindre ses buts. La construction de ces modèles est fondée sur l'analyse du propos (but) des services, qui se décompose en sous-buts jusqu'à atteindre un niveau où s'identifient les plans avec lesquels il peut atteindre son but. Pour déterminer si un plan est approprié on utilisera les croyances de l'agent sur l'état de son environnement et le contexte de l'agent.

A la différence des méthodologies orientées objets, l'emphase de cette méthodologie est basée sur les rôles, les responsabilités, les services et les buts. Il s'agit de considérer le domaine d'application en fonction de ce que l'on doit atteindre et dans quel contexte, au lieu de déterminer les types de comportement (classes d'objets). Ce raisonnement est fondé sur la

considération que les buts, comparés aux comportements et aux plans, sont plus persistants dans le temps. Par conséquent, une identification correcte des buts peut amener à un système plus robuste, stable et modulaire, où peuvent être conçus différents comportements pour atteindre un même but. Par exemple, face à des changements dans l'environnement, on peut définir de nouveaux comportements pour atteindre un même but. Du point de vue du génie logiciel cette façon de voir facilite une conception extensible qui peut traiter de multiples changements et de cas particuliers, et permet également un développement et des tests incrémentaux.

Apparemment, cette méthodologie a été expérimentée dans des cas réels [117] et son plus grand mérite consiste à déterminer les aspects qu'introduit le concept d'agent sur le modèle des objets. La méthodologie est centrée sur les aspects qui détermineront à la fin l'architecture interne de chaque agent en accord avec une architecture déterminée, et laisse ouvert le traitement de l'organisation du SMA ou l'introduction dans le système d'agents basés sur d'autres modèles de comportement, délibératif ou réactif.

D'autres méthodologies essayent d'aborder la conception d'un SMA sans un modèle d'architecture d'agent préétabli. Tel est le cas de DESIRE et Gaia.

DESIRE (DEsign and Specification of Interacting REasoning components) [28] aborde la spécification formelle de SMA en mettant l'emphase sur la décomposition des tâches. En accord avec la définition d'agent que nous avons présentée antérieurement, et comme nous l'avons commenté lors de la discussion sur l'usage de BDI, un des aspects différenciateurs du raisonnement des agents est l'emphase sur les buts des agents. Cependant, DESIRE considère davantage la distribution des tâches, ce qui rend assez discutable son application sur des problèmes comme ceux envisagés dans la section 1.2. Par ailleurs, DESIRE se centre sur le développement d'un langage basé sur la logique temporelle plus que sur la proposition d'une méthodologie de développement proprement dite.

Gaia [183] tente d'étendre la méthodologie Fusion [33] avec des concepts et terminologie d'agents. Elle aborde les étapes d'analyse et de conception de SMA du point de vue de SMA comme organisation des entités qui interagissent. Dans l'analyse on conçoit le système comme une organisation (ou société) de rôles en interaction, et on définit deux modèles, l'un de rôles et l'autre d'interactions. Pour chaque rôle sont assignés des responsabilités (décrites comme propriétés de vivacité, états auquel devrait arriver l'agent dans des conditions déterminées de l'environnement, et de sécurité, invariantes qui doivent se satisfaire à travers tous les états d'exécution), des permissions (ressources, comme l'information, disponibles pour réaliser leurs responsabilités, par exemple la capacité pour lire, modifier ou générer un document), des activités (actions qu'il peut réaliser sans interagir avec d'autres agents) et des protocoles (le

mode d'interaction avec d'autres agents). Ces protocoles se définissent dans le modèle d'interaction. Dans la conception s'identifient trois modèles: le *modèle d'agent* pour définir les types d'agents (comme un ensemble de rôles) et leurs instances, le *modèle de services* pour identifier les fonctions que réalise chaque rôle d'agent (déterminés à partir de la liste de protocoles, d'activités et de responsabilités), et le *modèle des voisins*, qui liste l'ensemble des agents connus avec lesquels chaque type d'agent peut interagir (le propos de ce modèle est d'identifier de possibles goulots d'étranglements dans les communications).

L'utilisation de Gaia conduit à un ensemble de spécifications qui peuvent se considérer au niveau d'analyse. Mais la conception est très faible, puisque aucun modèle computationnel n'est envisagé. Il n'y a pas non plus de guides méthodologiques, par exemple, pour assigner les rôles aux agents. Et bien que dans sa version la plus récente sont envisagés des aspects organisationnels [186], aucun patron d'organisation n'est proposé, ni comment déterminer le type de rôles qui devrait exister dans une organisation. Par conséquent l'utilité de Gaia pour la réalisation de SMA est assez limitée, mis à part les apports qui peuvent être obtenus quant à l'étude qu'il fait sur la définition de rôles et d'interactions, bien que ces dernières se spécifient de manière assez simple et sans considérer les aspects de concurrence.

Plus centrées sur les aspects pratiques de construction de SMA et assistées par des outils graphiques sont les méthodologies ADELFE, MASE et Zeus.

ADELFE [13][150] se centre sur les aspects adaptatifs et auto-organisateurs des SMA. Il propose une notation qui étend UML et une extension du Processus Unifié pour prendre en compte la modélisation et la réalisation de systèmes SMA adaptatifs. Ce processus permet de prendre en compte l'ensemble des activités du cycle de vie en V du logiciel (recueil des besoins, analyse, conception, implémentation et tests) sous la forme de modèles UML. Le travail est facilité avec l'outil OpenTool [78].

En ADELFE, les agents sont considérés comme des composants logiciels qui s'auto-organisent pour configurer les applications. Un agent en ADELFE est composé de sept modules : communication avec les autres agents, communication avec l'environnement, croyances sur lui-même, croyances sur les autres agents, croyances sur son environnement, compétences, attitude sociale coopérative. L'organisation de SMA n'est pas spécifiée d'une façon explicite, mais elle est conséquence des interactions entre les agents. La méthodologie spécifie comment les phases d'identification, de découpage du système en classes et composants peuvent être adaptées à une décomposition en agents auto-organisateurs.

MaSE [42] considère les agents comme une extension des objets avec une capacité de se coordonner au moyen de conversation et non pas de simples appels à méthodes. MaSE propose

un cycle de développement interactif et incrémental avec des activités d'analyse et de conception, qui sont assistées par l'outil *agentTool* [41]. Pendant l'analyse s'identifient les buts du système à partir d'un ensemble de besoins fonctionnels puis se définissent les rôles nécessaires pour assumer ces buts. On utilise des diagrammes de cas d'utilisation pour valider les buts et dériver l'ensemble de rôles initiaux. Tout cela comprend trois étapes : capture des buts (à partir d'une spécification de système initial, ce qui est, une spécification de besoins fonctionnels), application de cas d'utilisation et raffinement de rôles.

L'idée de commencer avec une analyse des buts du système est similaire à celle de AAII [111] puisque MaSE considère aussi que les buts sont plus stables dans l'analyse et la conception que les fonctions, les processus et les structures d'information. La conception comprend quatre étapes : la création de classes d'agents, la construction de conversations, l'assemblage de classes d'agents (conception de l'architecture interne des agents et les processus de raisonnement) et la conception du système (diagrammes de déploiement).

Cette analyse des buts et leur distinction claire par rapport à une analyse fonctionnelle classique sont les apports les plus intéressants de MaSE. Il propose aussi un mécanisme pour l'identification des rôles et leur assignation aux agents. Pour arriver finalement à un modèle computationnel des agents, il réalise, cependant, une certaine simplification quand on considère que chaque type d'agent correspond à une classe d'objets. D'autre part, certaines considérations, comme celle d'assumer que l'on dispose d'un ensemble de besoins fonctionnels initial ou la détermination d'un ensemble établi de conversations, font que leur applicabilité se limite à résoudre des problèmes similaires à ceux traités avec une méthodologie orientée objets classique. C'est ainsi que le résultat est un ensemble de classes dont le comportement est défini par des automates mais il n'est pas clair comment on pourrait aborder la construction des agents délibératifs, comme c'est le cas des agents BDI, puisque on ne définit plus, par exemple, comment gérer et contrôler la satisfaction ou l'échec des buts.

Zeus [130] est un environnement visuel de développement de SMA qui prétend faciliter à l'ingénieur de se concentrer sur le problème à résoudre. Il fournit les éléments nécessaires d'une plate-forme d'exécution d'agents, incluant les agents d'utilité comme les services de nommage, notification, ou facilitateurs, et les composants basiques pour la réalisation des agents. Fondamentalement il tente de diriger le développeur dans la configuration d'un agent générique au moyen de la définition d'ontologies, d'agents, de tâches, d'organisation et de coordination. Pour l'usage des outils une méthodologie est proposée [34]. Elle est basée sur quatre étapes : (1) analyse du domaine basé sur la modélisation de rôles, (2) conception des agents (identification d'ontologies, services, tâches et relations entre agents), (3) réalisation des agents (définition des

éléments identifiés dans la conception et la programmation des agents) et (4) assistance pendant l'exécution (debug et optimisation du code). Pour faciliter son utilisation, des prototypes sont offerts pour divers domaines d'application, qui peuvent servir de point de départ pour le développement de SMA similaires.

Conceptuellement Zeus peut être considéré comme supérieur à MASE puisqu'il applique des éléments de technologie d'agents comme ACL, ontologies, planification, et coordination. Cependant, Zeus est sujet à l'application d'une architecture d'agent très concrète et, dans les étapes d'analyse et de conception (non supportées par les outils) il se limite à suggérer comment regrouper la fonctionnalité du système à l'intérieur de chaque rôle, laissant de côté les considérations sur l'organisation des tâches, la définition des ontologies et les dépendances sociales.

Une méthodologie plus représentative d'adaptation des techniques d'intelligence artificielle à la problématique de distribution des agents est MAS-CommonKADS [100], qui ajoute à la méthodologie CommonKADS, conçue pour développer des systèmes experts qui interagissent avec un usager [38], des techniques d'orientation objets pour faciliter leur application, et d'ingénierie de protocoles (SDL et diagrammes de séquences de messages) pour décrire les interactions entre les agents. La principale différence est dans l'orientation agents, ce qui implique de définir un modèle d'agent et de donner plus d'importance aux modèles d'organisation et de coordination (interactions entre les agents).

Le modèle de cycle de vie est un modèle en spirale dirigé par des risques [18] suivant la gestion des projets de CommonKADS, bien que pour des petits projets il recommande d'utiliser un modèle en cascade avec réutilisation. Il commence par une phase de *conceptualisation*, qui essentiellement consiste en la définition des cas d'utilisation, avec le propos de déterminer les rôles du système (une idée qu'emploie MaSE aussi). A partir de là, pendant l'analyse, s'élabore la spécification du système depuis plusieurs points de vue: modèle d'agent (caractéristiques de l'agent comme capacités de raisonnement, senseurs, effecteurs, services), modèle de tâches (incluant des buts et leur décomposition), modèle d'expérience (connaissance que nécessitent les agents pour atteindre leurs buts), modèle d'organisation de la société d'agents, modèle de coordination (définit les interactions et les protocoles entre agents), et le modèle de communication avec l'usager (tel que le définit CommonKADS). Finalement ces modèles sont détaillés en un modèle de conception. C'est ici où se détermine l'architecture la plus adéquate pour chaque agent et les besoins du réseau des agents. De cette manière, il est possible d'arriver à la réalisation du système sans être trop conditionné au niveau de l'étape d'analyse par une solution particulière. Les autres étapes du cycle de vie d'un SMA ne sont pas développées dans

MAS-CommonKADS, quoique identifiées comme travail futur, tout comme la réalisation d'outils de support à la méthodologie. Dans le cadre d'un tel support, la définition des états des modèles, qui vient de CommonKADS est utile, et permet de leur associer buts et risques pour pouvoir gérer le processus de développement.

Les aspects organisationnels, que MAS-CommonKADS considère comme une caractéristique essentielle du paradigme agent, sont aussi considérés par d'autres approches. Les rôles et les services aident à structurer les fonctionnalités associées aux agents ou groupes d'agents, et permettent aussi de mieux comprendre et de gérer des systèmes complexes. Le projet AALAADIN [55] est un des premiers à établir les principes de développement des SMA avec une perspective organisationnelle. Il propose un modèle générique de SMA basé sur les concepts organisationnels des agents, groupes et rôles, le modèle AGR (Agent/Groupe/Rôle), qui est supporté par l'outil MadKit (http://www.madkit.org/).

Devant toutes ces alternatives il paraît logique de chercher une solution intégratrice et ceci est le but du projet MESSAGE [54]. Suivant le principal courant d'idées qui pose l'extension du modèle des objets, il considère le Processus Unifié de Rational [104] pour définir le cycle de développement d'un SMA et UML comme base pour la notation. Son apport le plus intéressant est l'utilisation de langages de méta-modélisation pour spécifier les éléments qui sont nécessaires pour la conception d'un SMA. Ces éléments se considèrent de plusieurs points de vue, chacun d'eux se décrit comme un méta-modèle où se définissent les relations entre les dits éléments :

- *Organisation*, qui capture la structure globale du système.
- *Tâches-buts*, qui détermine ce que fait le SMA et ses agents constituants en termes des buts qu'ils poursuivent et les tâches impliquées dans le processus.
- *Agent,* qui contient une description détaillée et extensive de chaque agent et rôle dans le SMA.
- *Domaine,* qui agit comme répertoire d'information (pour entités et relations) du domaine d'application
- *Interaction*, qui traite la communication entre agents sur plusieurs niveaux d'abstraction.

Comme on le décrira de manière plus détaillée dans la section suivante, la méthodologie MESSAGE n'est pas complète puisqu'elle ne décrit que la phase d'analyse et ne donne que certaines idées sur la conception. De plus il lui manque l'amélioration de la consistance des méta-modèles.

Les points de vue en MESSAGE doivent une grande influence au paradigme VOYELLES [44], qui considère que les SMA sont composés d'Agents, d'Environnements, d'Interactions, et d'Organisations, et que ces quatre entités de base sont d'égale importance dans la conception du SMA. Les agents peuvent être conçus comme de simples automates ou comme systèmes cognitifs complexes. Les interactions peuvent être étudiées comme des modèles physiques (par exemple, propagation des signaux) ou comme actes de parole. Les organisations peuvent être inspirées des modèles biologiques ou gérées par des modèles sociologiques. Le but de l'approche VOYELLES est de considérer des librairies de composants qui fournissent des solutions pour chaque aspect, de telle façon que le développeur puisse instancier un modèle d'agent, un modèle d'organisation, etc. La méthodologie propose de considérer les voyelles (aspects) dans un certain ordre, dépendant du type de système à développer. Par exemple, si les relations sociales sont importantes, le processus de développement doit partir de l'organisation. Si le processus commence avec les agents, le système aura une organisation qui probablement émergera comme résultat des interactions des agents individuels. Cette approche a inspiré la plateforme de développement MASK [132], organisée en quatre outils, chacun correspondant à un de ces aspects.

D'autres méthodologies qui proposent la modélisation des SMA en plusieurs points de vue sont MASSIVE [116], qui en considère sept (environnement, tâches, rôles, interactions, société, architecture et système), et ODAC [74] qui utilise les cinq points de vue du modèle Object Distributed Processing, ODP (entreprise, information, computationnel, technologie et ingénierie) [Rec. ITU-T X.900].

1.4 Discussion

Bien qu'il existe de multiples applications basées sur les SMA, il n'existe pas encore une méthodologie standard de développement de ce type de systèmes suffisamment mature qui aborde tous les aspects nécessaires pour définir un SMA, ni tout le cycle de développement de ce type de logiciel. De la même façon que ce qui est arrivé pour le paradigme objet, il existe de multiples propositions méthodologiques et nous pouvons considérer qu'il y aura aussi un processus d'unification, intégrant les aspects les plus importants de celles-ci. INGENIAS part de cette supposition en considérant aussi que :

- Le paradigme d'agent s'étend à celui d'objet. En conséquence, une méthodologie de développement de SMA devrait profiter de l'expérience de l'approche objet. Les méthodologies orientées objets ont acquis un haut niveau de maturité et leurs avantages sont largement reconnus. Une grande partie des développeurs de logiciel sont

familiarisés avec elles et utilisent la large gamme d'outils disponibles. Un cycle de développement itératif et incrémental, basé sur des cas d'utilisation, est assez approprié pour des systèmes complexes et dynamiques comme sont les SMA. Les extensions méthodologiques doivent tenir compte de manière particulière les aspects sociaux (organisation, interactions, négociation), de comportement (autonomie, état mental, buts, tâches), de concurrence et de distribution.

- La construction des modèles est l'élément clé du processus de développement. Les modèles de SMA doivent être structurés en plusieurs perspectives complémentaires pour pouvoir gérer la complexité du SMA.
- L'analyse basée sur les rôles et services aide à regrouper les différentes fonctionnalités associées à un agent ou à un groupe d'agents en facilitant la compréhension de systèmes complexes.
- Il existe une ample variété de concepts spécifiques de la technologie d'agents qui devraient être clairement définis pour leur utilisation dans le processus de développement de SMA [11]. Nous verrons dans ce sens l'utilité de la spécification des méta-modèles.
- Par rapport à la phase de conception, il y a des méthodologies qui assument une architecture d'agent concrète (par exemple, AAII, MaSE, Zeus) et d'autres qui considèrent que celle-ci devrait être plus générique (comme Gaia). Les premières sont normalement celles qui sont supportées par des outils tandis que les secondes finissent par être simplement des exposés théoriques. Notre position à ce sujet se situe dans la ligne de MAS-CommonKADS : considérer que pendant la conception se définit un modèle computationnel d'agent et par conséquent une architecture de celui-ci. C'est lors de cette étape que l'on peut considérer si un agent nécessite une architecture réactive ou délibérative, par exemple. Ultérieurement, lors de l'étape d'implémentation on déterminera une réalisation concrète de l'architecture sur une plate-forme spécifique d'agents, avec un processus de transformation.
- Il est très important de disposer d'outils qui contrôlent le processus de développement dans toutes ses étapes et aident le développeur à produire et mesurer la qualité des résultats en accord avec la méthodologie.

En plus de ces conclusions que nous pouvons tirer de l'étude de l'état de la question, nos expériences dans le développement de plusieurs SMA dans le cadre de différents projets ont beaucoup influencés nos travaux. Tout cela nous a fait évoluer, d'une façon très naturelle, dans

l'approche de développement dirigé par modèles (en anglais, *Model Driven Development*, MDD). Cela veut dire que la modélisation au niveau spécification est le centre de notre activité, pendant que l'implémentation est dérivée avec le support d'un ensemble d'outils. De cette façon, nous avons eu besoin de donner de plus en plus d'importance à la définition des transformations. Et, d'autre part, nous avons commencé à considérer la définition de plusieurs langages, de plus haut niveau, plus près du domaine d'application, comme extensions du modèle fondamental de SMA. Ce saveur MDD est, probablement, l'aspect le plus original d'INGENIAS dans le domaine du génie logiciel orienté agents.

1.5 Contributions scientifiques et positionnement

Dans le but de continuer le travail que nous avons réalisé dans le projet MESSAGE (Eurescom P907), et intégrant l'expérience dans le développement des applications de SMA des autres projets (par exemple, IST PSI3, IST DEMOS, Eurescom P815, SIMBA), nous avons initié le projet INGENIAS. Il s'agit en effet d'un cadre dans lequel notre équipe de recherche intègre plusieurs travaux dans le domaine du génie logiciel pour SMA. Ainsi, il y a des résultats divers, pour la modélisation des SMA, leur vérification, la définition de métriques logicielles, l'intégration des théories des sciences sociales, ou la simulation des systèmes complexes, par exemple. Formellement, INGENIAS est un projet de recherche subventionné par le Conseil Espagnol des Sciences et Technologie entre 2003 et 2005 (TIC2002-04516-C03), actuellement en exécution et dont la continuation de 2006 à 2008 a été approuvée (INGENIAS 2, TIN2005-08501-C03).

Le but global de INGENIAS est de fournir un ensemble de méthodes et d'outils pour le développement de SMA. Pour atteindre ce but, le parcours proposé a été d'étendre la proposition originelle de MESSAGE à de nouvelles méthodes et outils. On a considéré aussi l'adéquation d'INGENIAS aux standards pour que les ingénieurs software puissent appliquer des concepts et techniques basées sur les agents pour le développement de nouvelles applications. Aussi, tenant compte qu'actuellement les propositions dans ce domaine sont variées, INGENIAS devra être capable de s'adapter à l'évolution des possibles standards de spécification des SMA.

Pour l'implémentation des SMA, INGENIAS considère qu'on ne peut pas assumer qu'il y aura une plate-forme unique d'exécution d'agents. Dans certains cas il y aura des systèmes FIPA, mais probablement il y aura des contextes très divers selon les domaines d'application. Par conséquent, il faudra rendre possible la transformation de SMA modelés avec INGENIAS en code pour divers contextes d'exécution.

Ainsi, INGENIAS assume la nécessité d'évoluer dans le langage de modélisation comme dans les contextes d'application. La base pour s'adapter à cette évolution est l'utilisation de méta-modèles. Les méta-modèles permettent de décrire les concepts qui peuvent s'utiliser pour spécifier un SMA. Si les méthodes et outils d'INGENIAS sont fondés sur ces méta-modèles, les changements dans la spécification des méta-modèles se verront reflétés immédiatement dans les outils. C'est pour cela que la base des méthodes et outils d'INGENIAS sont les méta-modèles qui décrivent les éléments d'un SMA. Les méta-modèles sont également le facteur unifiant les différents travaux de notre groupe : pour la modélisation des SMA, mais aussi comme base pour réaliser les outils de génération de code, de vérification ou de simulation.

Par rapport aux transformations de modèles de SMA à modèles d'implémentation, et tenant compte que l'on va utiliser des techniques de méta-modèles, dans INGENIAS on propose de suivre d'une façon particulière l'approche de Développement Dirigée par Modèles (*Model Driven Development*, MDD) [166]. En INGENIAS on considère d'une part, le modèle du SMA, en utilisant le langage de modélisation de SMA d'INGENIAS, et d'autre part le modèle d'implémentation. Le modèle d'implémentation décrit comment réaliser les concepts d'agent en une plate-forme spécifique, qu'elle soit ou non orientée agent. Pour chaque modèle d'implémentation possible on définit une transformation depuis le modèle de SMA qui est utilisée par l'outil de génération de code. De cette manière il est possible d'automatiser une grande partie du processus de développement.

Nos principales contributions scientifiques sont :

1. La définition d'une méthode de spécification des SMA.

   INGENIAS cherche à intégrer des résultats de différents domaines de recherche sur les agents. Pour y parvenir, INGENIAS définit un ensemble de méta-modèles. Ces méta-modèles décrivent différents aspects du SMA, concrètement : l'environnement, les interactions, les organisations, les agents et leurs buts et tâches. Le processus d'intégration consiste à faire évoluer ces méta-modèles, en les détaillant ou en incluant de nouveaux concepts. Ceci implique, dans les étapes initiales, de modifier substantiellement les méta-modèles de MESSAGE, mais en l'état actuel les modifications consistent à détailler ou étendre les concepts existants au moyen des primitives du langage de méta-modélisation. La première révision des méta-modèles est publié en [87] et illustré avec exemples en [79] et [80].

   Ces méta-modèles sont appliqués dans plusieurs cas d'étude et évoluent pour incorporer de nouveaux concepts qui n'existaient pas avant. Cependant, il est inévitable que la modification de ces méta-modèles soit biaisée par la formation de ceux qui participent à la

méthodologie INGENIAS. Pour cela, actuellement nous avons la participation d'autres groupes de recherche qui contribuent à l'extension et à la critique de ces méta-modèles. Ceci implique de grands défis, puisque apparaissent différentes lignes de développement de recherche des méta-modèles qui ne doivent pas être nécessairement compatibles. Cependant, c'est un pas nécessaire face au but intégrateur d'INGENIAS. En ce sens, nous considérons important notre participation aux réseaux thématiques comme AgentLink ou AgentCities.ES. Plusieurs exemples d'applications avec d'autres groupes ont été publiés [75][86][145].

2. Définition d'un processus de développement des SMA.

   INGENIAS élargit la proposition de MESSAGE, en définissant avec plus de détails les activités d'analyse, conception et implémentation pour élaborer un guide de méthodologie aux développeurs de systèmes d'agents.

   Cet aspect est important, car se manipulent des concepts très divers et parce que le degré de complexité d'un système avec des agents multiples exige une définition soigneuse de l'organisation et des interactions de SMA. Ainsi, il est important de déterminer les relations qu'il y a entre les éléments qui apparaissent dans plusieurs vues. Par exemple, les buts qui se définissent dans la vue de tâches/buts, comme décomposition des buts de l'organisation, comme cause des interactions et comme entités gérées pour le contrôle d'agent.

   Ce processus a été défini en détail dans la thèse de doctorat de Jorge Gómez-Sanz [79], et publié aussi avec des exemples en [146].

   Actuellement, il évolue vers une approche dirigée par des modèles (MDD) [147]. Celle ci a comme avantage de faciliter, en utilisant des notions d'agent, la connexion entre la modélisation plus abstraite et les modèles d'implémentation, qui dans certaines occasions peuvent être limités par une technologie concrète, n'étant pas forcément orientée agent.

3. Outils de développement et validation de SMA.

   Pour que la méthodologie soit applicable, le support d'outils est nécessaire. Les outils devraient aider à l'édition des spécifications des SMA par l'usager, permettre de traduire les spécifications en composants logiciels, valider les modèles et générer la documentation. La spécification de SMA se construirait en utilisant des éléments définis dans les méta-modèles. Celle-ci est aussi un sujet de recherche pour la conception des nouvelles techniques. De maneiere concrete, nous travaillons actuellement sur les méthodes de transformation des spécifications en utilisant des templates. Ceci demande une étude plus

approfondie de la relation entre modèle, template et code final, et leur impact sur le processus de développement [147].

L'expérience dans le développement de SMA nous a aussi permis d'identifier les pas répétitifs dans la transition de la conception à l'implémentation. Cela nous a amené à définir le processus de génération partielle de code à partir des spécifications de modèles de SMA. Pour cela nous proposons la définition des schémas ou *frameworks* de SMA qui peuvent être spécialisés et configurés à partir de ces spécifications. Ces schémas peuvent être désignés pour la réalisation du SMA sur une plate-forme d'agents, comme Jade [8], qui offrent les services basiques pour l'implantation de ce type de systèmes, bien qu'aussi nous considérons d'autres plates-formes non orientées aux agents, comme J2EE ou Robocode.

Pour la vérification et la validation nous avons tenu compte des caractéristiques intentionnelles et sociales des SMA [63]. Pour cela, nous avons considéré l'application des théories provenant du domaine de Sciences Sociales. Plus concrètement, la Théorie de l'Activité. Celle-ci a été adoptée dans le cadre d'INGENIAS [65][66] et utilisée pour l'analyse des besoins et la vérification de contradictions dans la conception de SMA et les propriétés sociales des SMA [67][68].

4. Établir des principes architecturaux pour le développement de SMA.

   Malgré le fait de disposer d'une méthodologie, des outils de développement et de notation, on ne dispose toujours pas de concepts d'ingénierie solides qui facilitent la construction de SMA à grande échelle. Les concepts architecturaux conventionnaux ne paraissent pas être les plus adéquats pour décrire et gérer les sociétés de composants qui interagissent de façon non déterministe. Nous pensons que la ligne fondamentale sur laquelle il faut travailler est le concept d'organisation comme équivalent à une architecture software adaptée au SMA. A la marge de l'intégration d'autres résultats de recherche, nous pensons que l'abstraction d'organisation est fondamentale pour approcher une méthode de développement qui ajoute une valeur par rapport à ce qui existe aujourd'hui dans les méthodes traditionnelles de génie logiciel. INGENIAS intègre ce concept d'organisation mais dans un état primitif [71]. Il serait nécessaire de pouvoir exprimer que, de la même manière qu'une organisation humaine peut utiliser les services d'autres organisations pour obtenir une série de services d'infrastructure pour son fonctionnement, on peut voir un SMA qui offre des services chaque fois plus avancés et personnalisés pour d'autres SMA. Une idée similaire est celle qui existe dans la conception de systèmes d'exploitation basés sur la notion de micro-kernel, qui offre des services très basiques d'intercommunication de processus, le reste des fonctions du système d'exploitation étant fourni par un ensemble de serveurs (gestion de

fichiers, gestion de mémoire, gestion de communication de réseau, etc.). Ce but peut être une solution pour permettre que les agents puissent s'implanter dans des dispositifs avec des capacités computationnelles diverses.

5. Méthodes et outils pour la conception et simulation des systèmes complexes, en utilisant le paradigme agent, avec un niveau plus haut d'abstraction.

   L'idée est de masquer la terminologie SMA avec des concepts d'un domaine spécifique d'application. De cette façon nous espérons rendre plus accessible la technologie à différents utilisateurs, non spécialisés en programmation, pour modéliser et étudier des systèmes complexes, comme les systèmes sociaux ou les processus économiques. En utilisant les outils de transformation d'INGENIAS il serait possible de dispenser l'utilisateur des détails liés à la infrastructure d'implémentation. Nous avons déjà validé l'expressivité du langage de modélisation INGENIAS pour la simulation [164] et traité l'implémentation avec plusieurs outils de simulation basé agents [163].

Ces différentes contributions ont été notamment validées sur différents domaines d'application [35][86][88][120][145][164]. Chaque expérience de validation (le prototypage d'un service concret basé sur des agents) montre l'application de INGENIAS tout au long du cycle de développement, qui comprend les activités d'analyse, de conception, d'implémentation et de tests. C'est ainsi que pendant l'exposition des contributions à continuation, nous avons inclus des références à certains projets significatifs que nous avons réalisés.

2. SPECIFICATION DES SMA

INGENIAS est fondé sur la définition de méta-modèles qui décrivent les éléments d'un SMA et leurs relations. Cette tâche peut se voir comme l'élaboration d'une ontologie de concepts nécessaires pour définir un SMA. Cette ontologie a évolué avec le temps, s'enrichissant de nouveaux concepts selon de nouveaux domaines d'application. Pour supporter cette évolution il est nécessaire d'élaborer des méthodes de modélisation qui permettent de spécialiser les concepts en fonction du problème considéré. Concrètement, la méthode de spécification de cette méthodologie est celle des méta-modèles. Cette méthode permet d'étendre les concepts, en ajoutant des détails, composer des concepts et de les associer dans des diagrammes.

L'application des techniques de méta-modèles dans le domaine des agents software a eu un précédent significatif sur le projet MESSAGE [31]. INGENIAS a continué et amélioré les méta-modèles initiaux de MESSAGE. Ces méta-modèles s'incorporent dans l'outil INGENIAS Development Kit (IDK), ce qui a permis de valider les résultats de son application de manière précise et fiable. Ils sont disponibles pour le public avec la distribution du IDK en http://grasia.fdi.ucm.es/ingenias/metamodel/ et ont été validés avec un ensemble de cas d'études de SMA appliqués à différents domaines, comme le commerce électronique, le filtrage collaboratif d'information, des jeux de stratégie d'équipe, ou des services sur systèmes mobiles.

Ce chapitre présente le langage de modélisation de SMA INGENIAS. La section 2.1 montre les éléments communs qui permettent, par exemple, de structurer les modèles, et les principes qui ont été suivis pour la définition des méta-modèles en INGENIAS. Les sections suivantes montrent les cinq perspectives qu'INGENIAS considère pour la conception des SMA : agent, organisation, buts/tâches, interactions et environnement. Chacune présente d'abord le méta-modèle et illustre avec un projet de SMA la perspective.

2.1 Les méta-modèles INGENIAS

De la même façon que pour exprimer un modèle orienté objets il faut considérer les classes, les interfaces, les objets, l'héritage, etc., pour développer un SMA on peut utiliser ces concepts et d'autres, comme *organisation, agent, perception, interaction, but, tâche*, etc. Le langage de modélisation détermine les concepts disponibles pour le développeur. Le langage de modélisation de SMA en INGENIAS est spécifié avec les méta-modèles.

Les méta-modèles INGENIAS décrivent les entités qui devraient faire partie d'un SMA et leurs relations. Les modèles de SMA sont construits avec des instances des entités de méta-modèles. Ainsi, pour un SMA il faut identifier des types d'organisation, groupes, workflows, agents, perceptions, interactions, états mentaux, buts, tâches, etc., qui sont des instances des entités des méta-modèles de SMA. En ce sens, les méta-modèles de SMA définissent un langage de haut niveau pour le développement de SMA à partir de concepts de la technologie agent, même si à la fin ils seront transformés en termes computationnels (comme classe, règle, méthode, message) de la plate-forme d'implémentation.

Pour structurer la spécification de SMA, on peut considérer plusieurs perspectives. Cette séparation a été déjà appliquée dans la plupart de méthodologies de développement de SMA, comme cela a été discuté en la section 1.3. INGENIAS part des spécifications de MESSAGE [31], qui ont été influencées par les perspectives proposées par VOYELLES [45]. On considère cinq perspectives (Figure 2) : agent, organisation, buts/tâches, interactions, et environnement.

- La perspective Agent décrit les responsabilités d'un agent avec tâches et rôles associés. La définition du contrôle de l'agent présente les buts et l'état mental requis pendant son exécution. Avec le modèle d'agent on peut définir les limitations à la liberté d'action de l'agent sans être restrictives à un paradigme de contrôle spécifique.
- La perspective Organisation établit l'architecture du système. D'après Ferber et Gutknecht [55], il y a des relations structurelles qui peuvent pas être restreintes à des hiérarchies entre rôles. Ces structures sont déléguées à des entités spécialisées, les groupes. Dans un modèle organisationnel il y a aussi des relations de pouvoir entre agents, groupes et organisations. La fonctionnalité dans l'organisation est exprimée par des workflows qui montrent les associations producteur-consommateur entre tâches ainsi que les responsabilités pour leur exécution, et les ressources associées à chacun.
- La perspective Environnement montre les senseurs et les effecteurs des agents. Elle identifie aussi les ressources et les applications existantes.

- La perspective Buts-Tâches décrit la décomposition des buts généraux en plus concrets, jusqu'à un certain niveau qui permet d'identifier des tâches pour les satisfaire. Aussi, il faut identifier les ressources nécessaires pour l'exécution des tâches, les composants software nécessaires, les entrées et sorties en termes d'entités du modèle.
- La perspective Interactions décrit comment se réalise la coordination entre les agents. INGENIAS va plus loin que les diagrammes d'interaction UML car elle montre la motivation des participants à l'interaction. Dans INGENIAS, il est également possible d'ajouter l'information sur l'état mental des agents pendant l'interaction et les tâches associées. Cela permet de justifier pourquoi un agent s'implique dans une interaction et pourquoi il doit continuer.

La génération de modèles n'est pas triviale puisqu'il y a des dépendances entre les différentes perspectives. Par exemple, les tâches dans le workflows du modèle organisationnel devraient être décrites aussi dans la perspective buts-tâches.

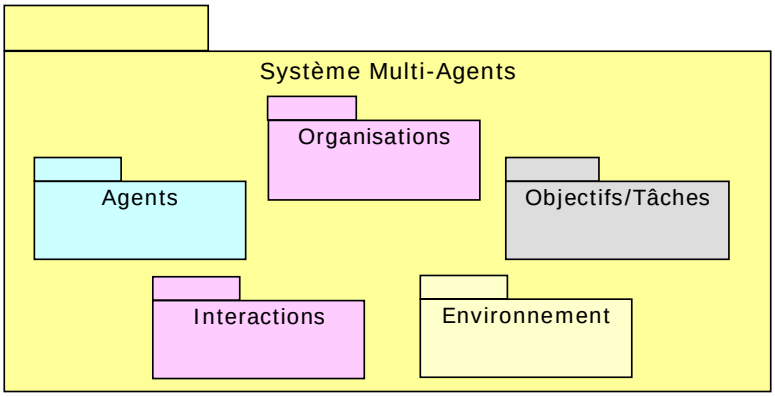


Figure 2. Les perspectives de SMA en INGENIAS

Ces perspectives peuvent être complémentées par des extensions de notations connues, comme par exemple les diagrammes de cas d'utilisation UML ou les diagrammes de collaboration UML.

La Figure 3 montre des fragments du méta-modèle pour les diagrammes de cas d'utilisation. Le premier fragment montre tous les éléments qui peuvent participer dans un diagramme de cas d'utilisation. INGENIAS ajoute, par rapport à UML, de nouvelles entités, telles que les agents, les cas d'utilisation INGENIAS et les interactions. Les autres fragments montrent certaines relations entre entités du méta-modèle. Par exemple, les associations « extends » entre cas d'utilisation, « ParticipatesInUseCase » qui inclut les agents en plus de rôles (UML), et « UMLDescribesUseCase ».

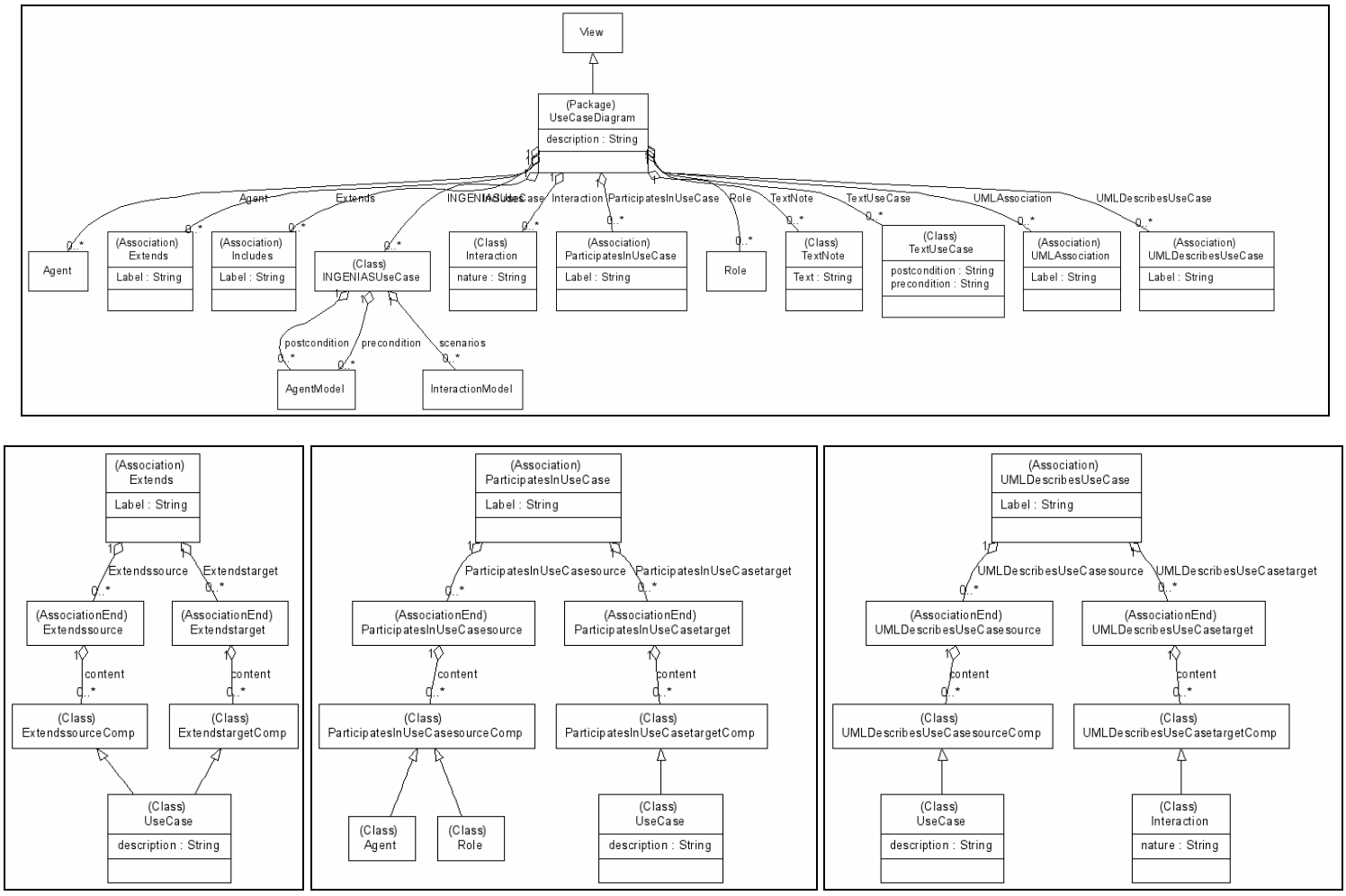


Figure 3. Fragments du méta-modèle pour décrire les diagrammes de cas d'utilisation en INGENIAS

Dans les sections suivantes, plusieurs diagrammes de ce type seront présentés pour décrire les différentes perspectives. Ces diagrammes ont été générés automatiquement à partir de fichiers des descriptions de méta-modèles INGENIAS, en XML, qui servent aussi pour la génération des outils IDK. Pour générer ces images on a utilisé le logiciel UMLGraph, disponible en http://www.spinellis.gr/sw/umlgraph/. Les diagrammes montrent les méta-classes et les méta-associations mais pas les extremités des associations (*AssociationEnd*), pour simplifier les figures.

Table 1. Statistiques de primitives MOF appliquées dans le méta-modèle INGENIAS

Class	87
Association	85
AssociationEnd	137

Le méta-modèle INGENIAS est disponible en http://grasia.fdi.ucm.es/ingenias/metamodel. La Table 1 montre les nombres d'entités qui sont spécifiées dans le méta-modèle. A première vue il pourrait paraître assez complexe, mais les entités spécifiées sont celles nécessaires pour construire des modèles de SMA assez détaillés, pour permettre une génération automatique de code complète. Néanmoins, le développeur peut déterminer le niveau d'abstraction approprié à ses besoins et simplifier les concepts à utiliser. Dans la première version d'INGENIAS [146] nous proposons de suivre le processus unifié, et plusieurs diagrammes d'activité sont proposés pour diriger l'analyse et la conception de SMA.

2.2 La perspective Agent

INGENIAS considère les agents comme des entités intentionnelles, qui suivent le principe de Rationalité, d'après Newell [129]. Ce principe établit qu'un agent va exécuter des actions qui peuvent le conduire à satisfaire ses buts.

2.2.1 Spécification des agents

La vision agent considère la fonctionnalité de chaque agent : le propos (les buts que l'agent poursuit), les responsabilités (les tâches qu'il est censé réaliser), et les capacités (les rôles qu'il va jouer). Le comportement de l'agent est défini par trois composants :

- L'état mental, comme agrégation des entités mentales comme les buts, les croyances, les faits, et les compromis. Chaque agent a un état mental initial, représenté par une association de l'agent à une entité d'état mental. Le concept d'*agent concret* permet d'exprimer l'évolution de l'état mental de l'agent (voire l'exemple de la Figure 13 en la page 39).
- Le manager d'état mental, responsable des opérations pour créer, supprimer et modifier des entités mentales.
- Le processeur d'état mental, qui détermine comment l'état mental va évoluer, au moyen de règles, planification, etc.

L'état mental peut être vu comme toute l'information qui permet à l'agent de prendre ses décisions. Cette information est gérée et traitée pour produire les décisions et les actions de l'agent par le Manager (M) et le Processeur (P) d'état mental. Le Processeur d'état mental prend la décision de quelle action exécuter, pendant que le Manager d'état mental procure les actions pour créer, supprimer ou modifier les entités de l'état mental. Au niveau de la spécification d'un SMA, M et P sont définis avec un haut niveau d'abstraction. C'est le processus de génération de

code qui va définir la réalisation concrète de M et P. Par ailleurs, il est intéressant de découpler M et P pour mettre d'une part les mécanismes de manipulation des entités mentales (M) et d'autre part les politiques de décision (P).

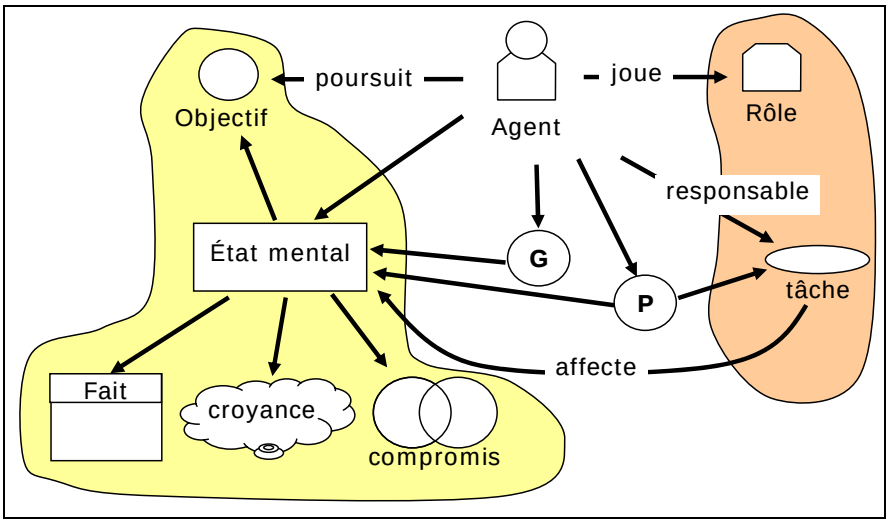


Figure 4. L'agent INGENIAS

La Figure 4 montre la représentation graphique des éléments qui sont utilisés pour décrire un agent. Ces éléments sont spécifiés dans le méta-modèle, dont une version simplifiée est présentée dans la Figure 5.

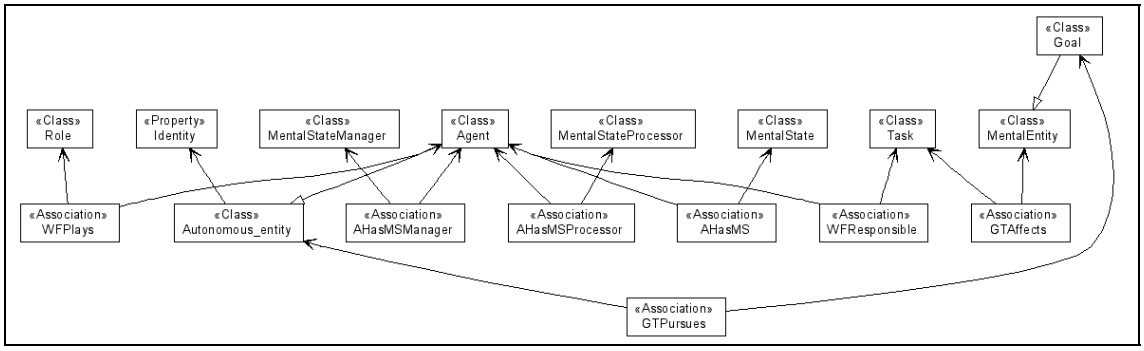


Figure 5. Fragment de méta-modèle de la perspective Agent

Pour la notation graphique, INGENIAS propose l'utilisation des stéréotypes UML qui font référence aux entités de méta-modèle, ou bien un ensemble d'icônes spécifiques, comme la Figure 6 le montre. Pour la perspective Agent, les icônes spécifiques sont montrées dans la Figure 4.

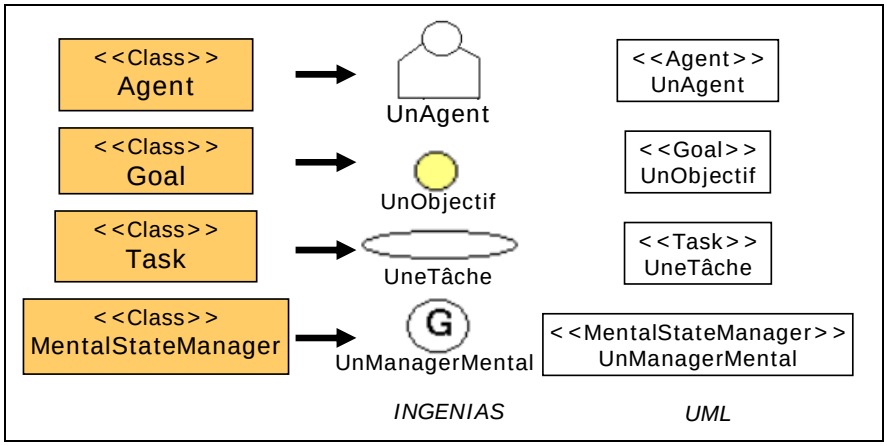


Figure 6. Notation des modèles en INGENIAS

2.2.2 Une architecture d'agents cognitifs pour agents d'interface

Une des premières expériences d'application des SMA dans le domaine des télécommunications a été réalisée dans le contexte de projet Eurescom P815 (*Communications Management Process Integration Using Software Agents*) [52]. Les projets Eurescom promeuvent la coopération des centres R&D des principaux opérateurs européens de télécommunication, dans ce projet, Telefónica I+D, BT, KPN Research, Deutsche Telekom, GMD Focus, Computas AS, Telenor AS, Telia Research AB et Broadcom Eireann Research. Par conséquent, ce projet se centrait sur la problématique de ce secteur : comment les opérateurs peuvent améliorer la provision de services dans un marché de télécommunications ouvert à de multiples fournisseurs et avec de nombreux domaines de gestion. Le contrôle et la prise de décision centralisée sur les systèmes de gestion traditionnels n'étaient pas viables devant les nécessités de l'infrastructure disponible. De nouvelles architectures, telles que TINA [143], ont été envisagées pour obtenir des solutions distribuées. Dans ce contexte, le projet Eurescom P815 envisagea d'explorer l'application de la technologie d'agents pour la gestion de processus (*workflows*) et de systèmes à travers plusieurs domaines de gestion de communications.

Tel que cela se définit dans [52], le but était de valoriser la maturité et les implications de la technologie des agents intelligents et mobiles et leur applicabilité au développement et à l'amélioration de la gestion électronique des processus métier (*workflows*) dans le domaine des réseaux et services de télécommunications. Pour cela a été proposée la réalisation de plusieurs prototypes expérimentaux à partir desquels se définit une architecture de logiciel pour la construction de systèmes d'agents. D'autres buts additionnels consistaient en la validation de standards émergents dans la technologie d'agents (en concret, FIPA et OMG) et en la production de recommandations sur la maturité et la stabilité de cette technologie pour son applicabilité dans le domaine des télécommunications.

Le projet était divisé en deux phases, la première dédiée à l'étude de la situation existante de la technologie d'agents, à la sélection de scénarios de gestion de processus et à la description des expériences à réaliser. Notre groupe de recherche collabora à la deuxième phase, qui consistait à développer les prototypes, à les expérimenter et à évaluer les résultats.

Des deux prototypes réalisés nous fûmes responsables de l'un d'entre eux qui considérait des agents d'aide aux ingénieurs en ce qui concerne la gestion du processus de développement de logiciel pour des réseaux intelligents. Les deux buts de ce prototype étaient :

1. Ajouter des fonctionnalités à un système existant (l'outil *Service Creation Environment,* propriétaire de Telefónica I+D) au moyen d'agents qui soient capables de capturer des événements de lui-même et fournir des services d'amélioration de la productivité aux ingénieurs engagés dans un processus de création de services de Réseaux Intelligents. De cette façon, nous pouvions démontrer qu'au moyen d'agents il était possible d'ajouter des fonctionnalités à un système hérité. Et pas seulement cela, puisque avec différents types d'agents nous pouvions apporter de la fonctionnalité spécifique pour chaque type d'usager (par exemple, pour le chef de projet et pour le développeur).
2. Structurer le développement des agents de manière que l'on dispose d'une architecture et de composants réutilisables, en évitant de devoir recommencer de futurs développements à partir de rien. Dans ce sens, on considère également important d'organiser la manière de spécifier le comportement d'agents (puisque les systèmes basés sur des règles de production n'offrent pratiquement pas de mécanismes de structuration) et de pouvoir contrôler en tout moment ce que fait l'agent.

Pour le développement du prototype, nous avons adopté un modèle itératif et incrémental dirigé pour des cas d'utilisation dans la lignée du Processus Unifié de Développement de Software [104]. Ceci a permis de caractériser l'architecture des *agents d'interface*, appelés ainsi parce qu'ils enrichissent la fonctionnalité dont dispose un usager lors de son interaction avec le système. Cette architecture d'agent est structurée en quatre niveaux :

1. Niveau de *contrôle*, responsable de la coordination et de la synchronisation des activités des autres composants. Les composants de ce niveau sont le Task Manager, avec l'architecture cognitive qui est décrite dans la section suivant, et l'Agent Management, que procure les opérations de gestion basique pour l'agent (démarrage, finalisation, instanciation des autres composants).

2. Niveau de *session*, qui gère les interactions avec d'autres agents, avec l'usager, ou avec d'autres applications. Chaque interaction évolue sur un contexte qui est géré par un composant de type Session Management, dans le même sens que TINA [143]. Pour l'interaction avec l'utilisateur ce composant peut être complexe parce qu'il peut être adaptable et personnalisable. Pour interagir avec des composants logiciels classiques, une implémentation avec une machine d'états finis est suffisante.
3. Niveau de *domaine*, qui donne l'accès à des ressources de l'agent dans un domaine d'application spécifique : pour cette application, un agenda, un gestionnaire de rapports et un gestionnaire de projets.
4. Niveau de *ressources* génériques de l'agent, comme la persistance, les proxies de communication, les éléments d'interface d'usager, traces, etc.

Les composants qui requièrent un modèle de comportement plus complexe (par exemple, la gestion de tâches au niveau de contrôle) se réalisèrent en suivant un modèle BDI [156]. Ces composants ont été structurés comme le montre la Figure 7 [88]. Le système est structuré en un moteur d'inférence (dans ce cas JESS [108]), une base de connaissances et une mémoire de travail. Les événements générés par l'interface d'usager, par une application ou par d'autres agents, sont assertés comme des faits.

La base de connaissances se compose d'un ensemble de règles d'inférence (*connaissance déclarative*), qui gère le raffinement, la résolution, la validation et la gestion des erreurs dans la recherche des buts, et d'un ensemble d'objets qui représentent le domaine et les buts (*connaissance du domaine et de la résolution*).

La mémoire du travail contient des informations sur le monde à chaque moment, ceci est un ensemble de faits (par exemple, des évidences des événements qui ont succédé), et l'espace des buts qui déterminent la décomposition et les relations des buts que poursuit l'agent. Cette structuration des buts (en arbres Y/O) est un des apports principaux quant à la structuration de la spécification du comportement d'agent. De plus, elle permet de contrôler, au moyen de règles de focalisation des buts, ce que fait l'agent à chaque instant.

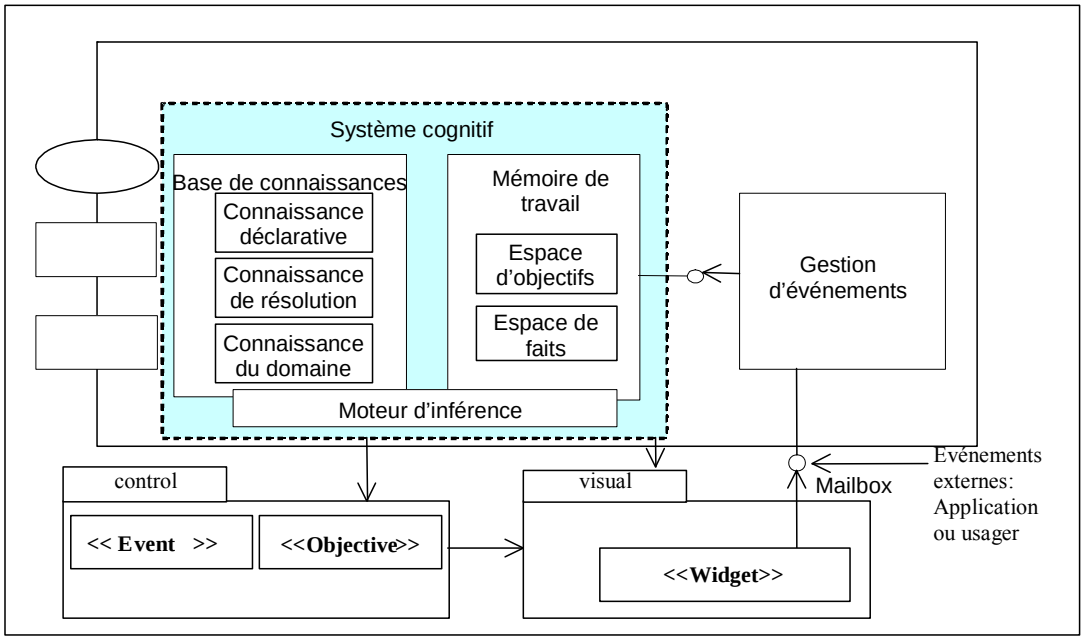


Figure 7. Contrôle d'un agent délibératif

La construction du contrôle de tâches de l'agent est fondée sur un processus en trois étapes :

1. Identification des événements, intentions et ressources.
2. Organisation des actions associées aux buts et définition de l'espace des buts de l'agent.
3. Détermination des croyances (événements) qui permettent d'exécuter chaque partie du plan (espace des buts).

La réalisation du prototype en plusieurs phases (développement itératif et incrémental) a permis de préciser la définition des éléments qui constituent l'agent et, plus particulièrement quant au comportement, la construction d'un paquet de classes pour réaliser des espaces des buts qui permettent de structurer de manière adéquate, de rendre plus compréhensible et d'améliorer la maintenance du système de règles associées. Cette architecture a été utilisée dans nos projets suivants et incorporée dans les développements basés sur les agents délibératifs de Telefónica I+D, ce qui montre son utilité. Elle a été également implantée à d'autres systèmes de production, comme ILOG JRules [101], dans un prototype de SMA pour le projet MESSAGE. Elle a aussi été la base expérimentale pour l'identification des éléments concernant le méta-modèle agent d'INGENIAS. Une description plus détaillée a été publiée lors de la conférence MICAI 2000 [88].

2.3 La perspective Buts et Tâches

La perspective de tâches et buts considère leur décomposition, et décrit les conséquences de l'exécution d'une tâche et pourquoi elle devrait être réalisée : elle justifie l'exécution des tâches comme un moyen de satisfaire les buts. D'après le Principe de Rationalité [129], les actions d'un agent sont justifiées par ses buts. Dans ce sens, les relations de satisfaction et echec permettent d'identifier quels buts sont influencés par l'exécution d'une tâche. Finalement, la perspective de tâches et buts explique comment la résolution d'un but peut affecter la résolution d'autres buts par des relations de décomposition et dépendance.

La raison pour laquelle cette perspective est présentée séparément de la perspective organisationnelle ou des agents est que le propos de ces perspectives n'est pas la relation des tâches et buts. La perspective organisationnelle détermine le contexte d'exécution des tâches et la perspective agent montre uniquement les capacités d'un agent concret.

2.3.1 Spécification des tâches

Plusieurs concepts de tâche sont révisés par Ferber [56]. INGENIAS considère essentiellement deux acceptions complémentaires pour le concept tâche. D'abord, une tâche peut transformer l'état mental d'un agent. De ce point de vue, une tâche est spécifiée avec des préconditions et des post-conditions. Cela permet l'incorporation des tâches dans les mécanismes de planification et de raisonnement. La deuxième acception, une tâche comme implémentation d'un processus, est plus pragmatique et est en accord avec la réalité finale : une tâche est un ensemble d'instructions qui doivent s'exécuter.

Le méta-modèle pour la définition des tâches prend des idées de TAEMS [39] pour la division des tâches, les influences entre tâches et l'intégration des ressources comme des biens nécessaires pour la réalisation des tâches. De Zeus [130] on reprend le modèle d'exécution des tâches. Les tâches peuvent commencer quand certaines préconditions, exprimées en termes d'entités de l'état mental de l'agent, sont satisfaites. Une action possible est l'actualisation de l'état mental de l'agent, par l'addition, la suppression ou la modification des entités d'état mental. Chaque tâche est finalement associée à un agent, qui est le responsable de son démarrage, contrôle et exécution, d'après les termes établis par le développeur.

D'autres travaux qui pourraient être pertinnents n'ont pas été pris en compte. Par exemple, MaSE [42] propose une façon de détailler le comportement basé sur des automates, qui n'est pas très approprié pour la construction d'agents cognitifs. Gaia [183] propose l'utilisation de préconditions et postconditions exprimées par des formules logiques. Même si d'une perspective formelle celle-ci est intéressante, cela n'est pas pratique ni pour modéliser le

domaine, ni pour la génération de code. Pour cette raison nous avons préféré définir les préconditions d'une façon similaire à Zeus. Les préconditions identifient les éléments de la partie gauche des règles, et leurs termes sont des entités de connaissance déterminées dans une ontologie.

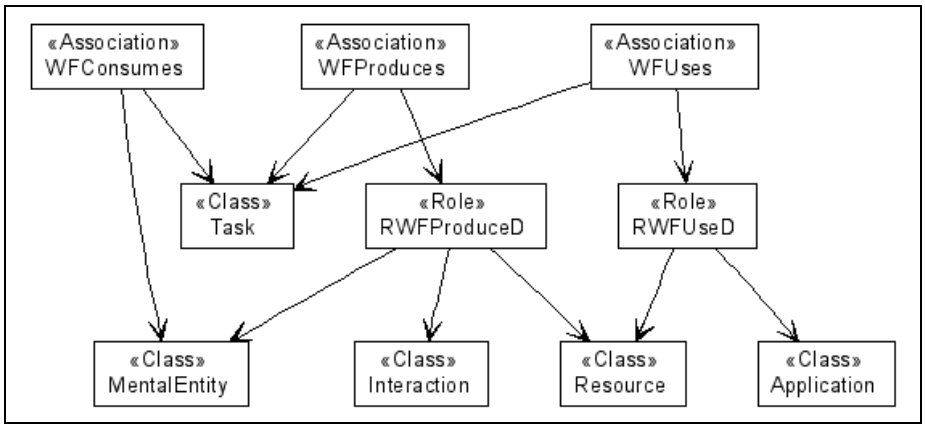


Figure 8. Description des tâches

La Figure 8 montre les relations qui permettent de définir les préconditions (*WFConsume, WFUses, GTAffects*) et les postconditions (*WFProduces* et *GTAffects*) pour les tâches. *WFConsume* montre que pour l'exécution d'une tâche il faut certaines entités mentales. *WFUses* dénote les ressources de l'environnement qui sont nécessaires pour l'exécution de la tâche. *GTAffects* indique que la tâche agit sur une entité mentale. Comme postcondition, *WFProduces* dénote la création des entités mentales, interactions où restitution des ressources.

2.3.2 Spécification des buts

Les buts sont utilisés pour raisonner sur les alternatives d'un agent à un instant donné. En planification classique (par exemple, STRIPS [57]), un but représente l'état du monde à atteindre. Cet état est décrit par un ensemble de prédicats, alors le but est une agrégation de prédicats. Dans le modèle BDI, un but est représenté comme une entité, un désir à satisfaire. Les buts sont le guide et la justification pour les actions de l'agent. Il y a une troisième interprétation du terme but comme besoins. Par exemple, KAOS [26], Tropos [29] ou MaSE [42] utilisent les buts dans la conception pour représenter les besoins que doit satisfaire le système. En INGENIAS, ces trois perspectives sont intégrées en considérant les buts comme entités auto-représentatives qui guident le comportement de l'agent. Pour la planification, les buts sont associés à l'ensemble des éléments (par exemple, prédicats) qu'ils représentent. Finalement, l'assimilation de besoins en buts est purement interprétatif : c'est à l'ingénieur de considérer le but comme un besoin ou pas. Cette interprétation est très convenable dans le processus de développement, comme on le verra dans la section 4.3.

2.3.3 La relation entre buts et tâches

Le méta-modèle des buts-tâches permet de décrire la motivation pour les tâches et les options qu'un agent peut avoir à un moment donné. La Figure 9 montre des relations entre buts et tâches. Les buts sont poursuivis par des entités autonomes, agents ou organisations. Les rôles aussi peuvent être associés à des buts, dans le cadre d'un workflow.

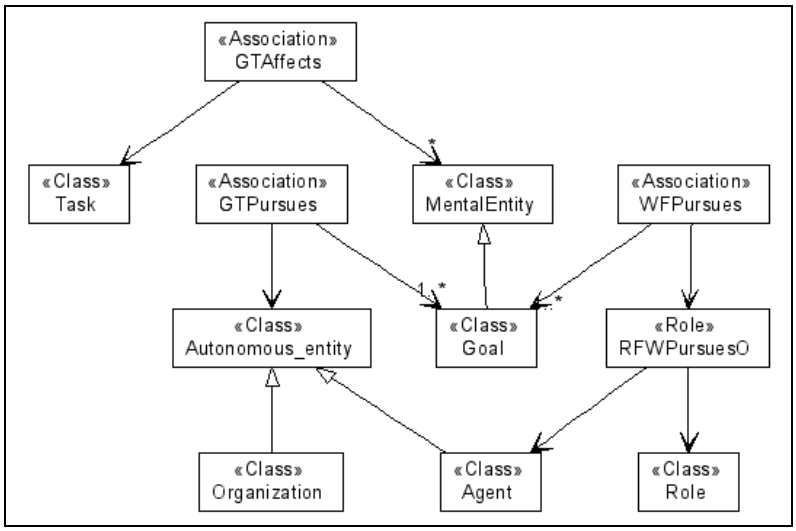


Figure 9. Relations entre buts et tâches

L'exécution des tâches peut affecter (relation *GTAffects*) les entités de l'état mental. Une entité mentale principale est le but (*Goal*). Il y a plusieurs façons d'affecter une entité mentale : destruction (*GTDestroys*), création (*GTCreates*) et modification d'attributs (*GTModifies*). Il est possible de etiqueter chaque relation avec un *patron d'état mental*, pour indiquer dans quelles conditions il peut avoir lieu. Dans le cas des buts, la relation *GTModifies* est spécialisée en *GTSatisfies*, pour indiquer le succès de la réalisation du but, et en *GTFails*, pour indiquer que le but n'a pas plus être satisfait. Pour ces deux relations il faut l'existence de certaines propriétés d'évidence.

Pour gérer la complexité des buts et des tâches, il y a deux relations : *GTDecomposes* pour décomposer un but en sous-buts, et *WFDecomposes*, pour décomposer une tâche en un workflow. La décomposition des buts donne lieu à des arbres AND/OR [158], qui sont supportés par des relations *GTDependsAND* et *GTDependsOR*.

2.3.4 Exemple : Agent planificateur

La Figure 10 illustre une modélisation des buts et des tâches pour un agent planificateur. On considère un agent qui peut exécuter deux types de tâches : TaskA et TaskB. De l'exécution de celles ci dépend la satisfaction du but O.

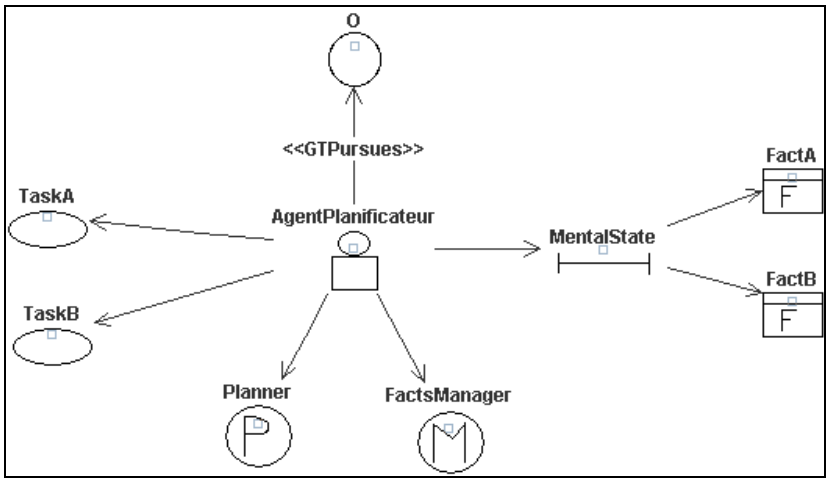


Figure 10. Un modèle d'agent planificateur

En principe, l'agent part de deux faits, A et B. Avec cette information, le processeur d'état mental, qui est vraiment un planificateur des tâches, se demande s'il est possible d'atteindre le but O. Pour cela il faut plus d'information sur les conditions dans lesquelles le but O est satisfait et les effets prévus de l'exécution des tâches A et B.

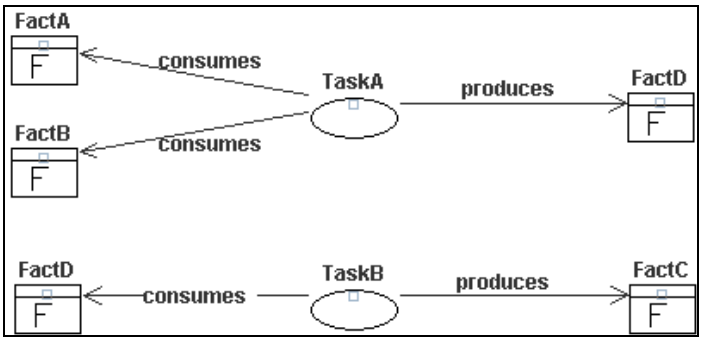


Figure 11. Modèle des buts et tâches pour spécifier les tâches TaskA et TaskB

La Figure 11 montre les conséquences de l'exécution des tâches TaskA et TaskB. Dans ce cas, toutes les deux produisent de nouvelles entités mentales. La Figure 12 montre l'association de la satisfaction du but O par la tâche TaskB. Cette association est etiqueté par un patron d'état mental qui dénote l'état mental requis pour considérer satisfait le but O. Pour cela, la Figure 13 montre l'état mental requis dans l'agent pour atteindre le but O. Dans ce cas, l'agent qui poursuit le but O doit connaître le FactC. À partir de ce moment, le processeur d'état mental de l'agent a l'information suffisante pour déduire que la tâche TaskA doit être exécutée avant TaskB pour atteindre le but O.

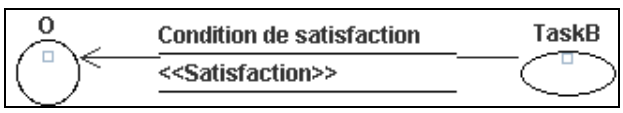


Figure 12. Modèle qui montre la satisfaction du but O par l'exécution de TaskB

Figure 13. Modèle d'agent pour représenter l'état mental requis pour satisfaire le but O

Le problème pourrait être spécifié d'une façon differente par la décomposition du but O, comme l'illustre la Figure 14. La décomposition est de la forme AND (partie droite de la figure). Ces sous-buts sont associés aux tâches comme le montre aussi la Figure 14. La description de l'état mental requis pour O1 et O2 est omise ici. Il faut simplement spécifier que pour satisfaire O1 et O2 l'existence des faits C et D est nécessaire, respectivement.

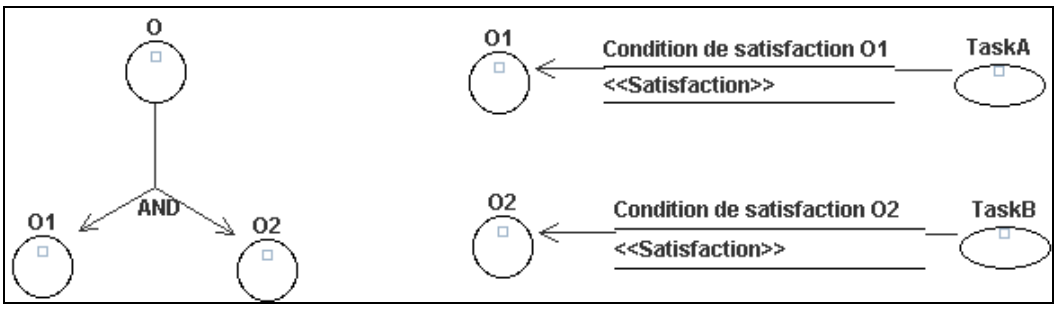


Figure 14. Décomposition AND du but O en sous-buts O1 et O2.
Et association des tâches TaskA et TaskB avec les buts O1 et O2

2.4 La perspective Organisation

L'organisation joue un rôle important dans le SMA puisqu'elle détermine le propos général du système, sa structure, le contexte dans lequel les agents jouent leurs rôles spécifiques, les politiques et les interactions pour la collaboration entre les agents. Dans la section 1.3 nous avons vu comment le concept de rôle est déjà présent dans la plupart des méthodologies. AALAADIN [55] est fondé sur ce concept et met en relation agent-groupe-rôle (AGR), permettant la conception de différents types d'organisation, comme les marchés ou les hiérarchies. Ces concepts ont été adoptés dans MESSAGE, et puis dans INGENIAS, en ajoutant aux concepts structurels certains qui permettent une vision dynamique de l'organisation en termes de workflows, et la définition des relations et normes à plusieurs niveaux, entre agents, groupes ou organisations [71].

2.4.1 Spécification de l'organisation

L'organisation décrit le cadre dans lequel les agents, les ressources, les tâches et les buts coexistent. Dans INGENIAS, elle est définie par une structure, une fonctionnalité et des relations sociales.

D'un point de vue structurel, l'organisation est un ensemble d'entités associées par des relations d'agrégation et d'héritage (Figure 15). Cette structure détermine le cadre pour l'existence d'agents, de ressources, de tâches et des buts. Sur cette structure se définissent une série de relations qui induisent la formation de workflows (dynamique) et les restrictions sociales.

L'organisation est une entité autonome, comme les agents, qui poursuit des buts. Les propos de l'organisation, ses buts, est partagé par tous les agents qui y participent. Il ne faut pas confondre organisation et agent, qui sont des concepts très différents. La différence principale est que l'organisation n'a ni la capacité d'exécuter des tâches ni de prendre des décisions : ce sont les agents qui la composent qui en ont la responsabilité.

L'organisation définit une structuration du SMA en groupes et workflows. Les groupes peuvent être formés d'agents, de rôles, de ressources, et d'applications. Ils sont utiles quand le nombre des entités dans le SMA devient considérable. L'assignation des entités à un groupe obéit à un propos organisationnel : ou bien parce que cela va faciliter la définition des workflows, ou bien parce que leurs membres ont certaines caractéristiques communes, par exemple, déployés dans une certain plate-forme ou un domaine administratif ou une distribution fonctionnelle des services de système, etc.

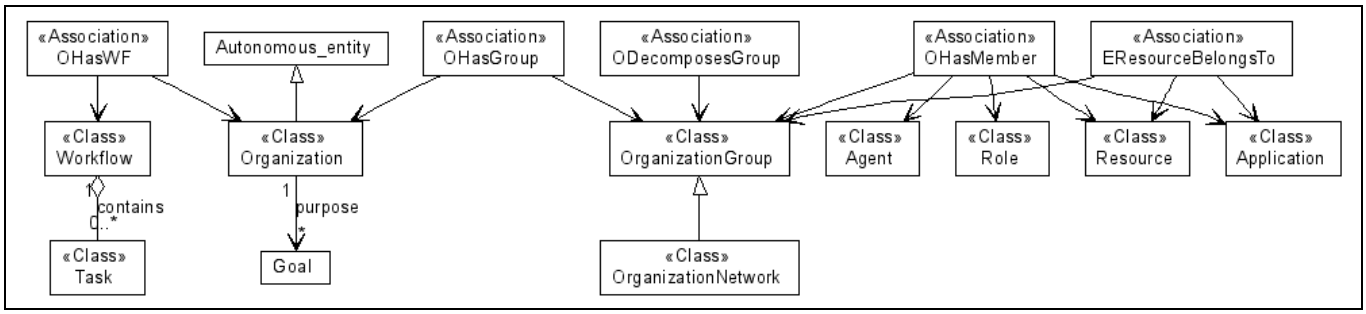


Figure 15. Éléments nécessaires pour définir une organisation

La Figure 15 montre comment une organisation peut se structurer par rapport aux groupes (*OHasGroup*) et workflows (*OHasWorkflow*). La décomposition récursive de groupes et workflows est réalisée par les relations *ODecomposeGroup* et *ODecomposeWorkflow*, respectivement. L'assignation des agents, des rôles et des ressources aux groupes est déterminée par des besoins organisationnels. En principe, INGENIAS n'impose pas de restrictions sur la façon d'établir les groupes. L'ingénieur peut ainsi définir des groupes avec plusieurs rôles, et assigner un même agent à plusieurs rôles dans différents groupes. Il est normal aussi de définir certains rôles du gestionnaire du groupe qui peuvent être joués par des agents. Ce type de situations est similaire à celles que l'on peut trouver dans des organisations humaines.

Pour augmenter la capacité d'abstraction et incrémenter la réutilisation des agents et rôles, il est possible de définir des relations d'héritage simple pour les agents et multiple pour les rôles. Ainsi, si un agent A « extends » un agent B, l'agent A peut remplacer B dans toutes les situations. Et si un rôle A « extends » les rôles B1 et B2, A peut remplacer B1 ou B2 dans toutes les situations.

La fonctionnalité de l'organisation est définie par son propos (buts) et ses tâches. Une organisation a un ou plusieurs buts et est dépendante de ses agents pour réaliser les tâches nécessaires pour les satisfaire. Les workflows déterminent les relations entre les tâches et ses responsables. Pour chaque tâche, un workflow détermine ses résultats, le rôle qui est responsable de l'exécuter, et les ressources qu'il demande (voir Figure 16).

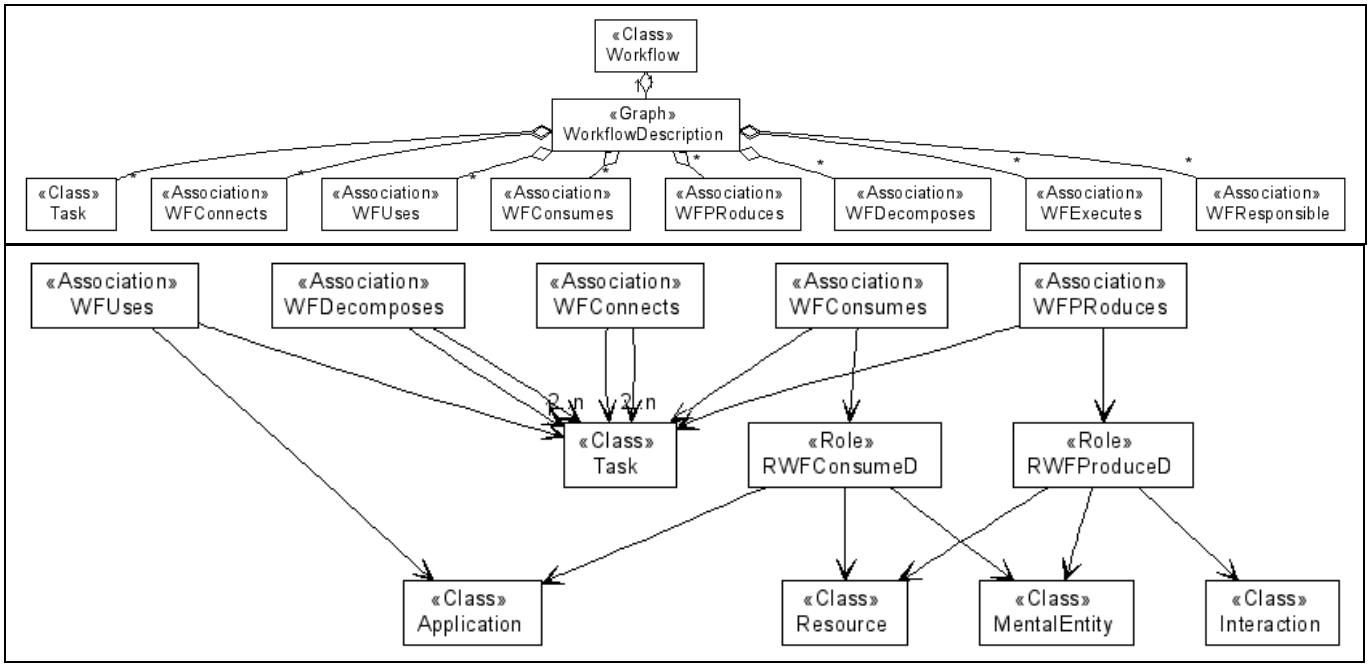


Figure 16. Méta-modèle pour les workflows

Finalement, la description sociale détermine les normes dans le SMA, les relations de service (par exemple, client-serveur, *peer to peer*), la subordination conditionnelle ou inconditionnelle, etc. (voir Figure 17). Les règles sociales déterminent les restrictions sur les interactions entre les entités de l'organisation. Ces relations sont définies entre les agents, les rôles, les organisations et les groupes. À noter qu'il est possible aussi de définir les relations sociales entre les organisations.

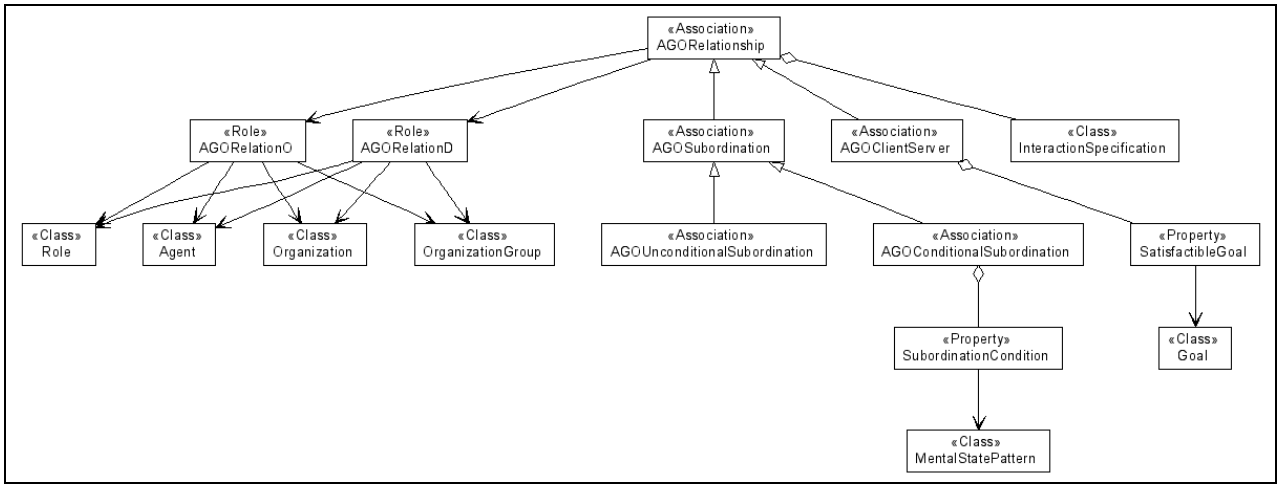


Figure 17. Méta-modèle pour les relations sociales

2.4.2 Exemple d'organisation : Communautés virtuelles en PSI3

Le projet IST PSI3 (*Personalized Service Integration Using Software Agents*) [86] permet d'illustrer comment les normes sociales et la définition des workflows peuvent faciliter la conception d'un SMA qui réalise un système de recommandations hybride basé sur des techniques de filtrage par contenu et collaboratif. De plus, comme il s'agissait d'une application sur le web, avec une grande quantité d'usagers, il était nécessaire de résoudre les problèmes de mise en échelle, qui requièrent une étude approfondie des aspects organisationnels et de distribution des agents.

Le but général du projet était de développer un ensemble de composants pour la récupération d'information, en considérant des aspects de personnalisation et la formation des communautés virtuelles avec les usagers. On a utilisé des agents pour gérer les profils de chaque usager et pour identifier et gérer des communautés d'usagers avec des intérêts similaires. Appliqué aux sites web, on a développé un ensemble de services personnalisés comme la réalisation de recherches plus qualifiées ou la notification de l'apparition de l'information relevant à chaque usager.

Plus concrètement, le SMA a permis de définir des communautés virtuelles qui offrent un service de recommandations des informations. Le SMA intègre des techniques de filtrage par contenu et collaboratif. Le filtrage basé sur les contenus analyse le contenu des objets pour obtenir une catégorisation des intérêts de l'usager. Il part de la définition d'un ensemble de catégories et détermine l'adéquation de chaque objet à chaque catégorie en analysant le contenu. On ne fait pas une analyse sémantique des documents car le traitement du langage naturel est assez complexe. Par contre, on analyse l'occurrence des termes les plus significatifs du

document et leurs relations de proximité. A partir de cela on peut les comparer à d'autres documents pour vérifier s'ils ont des caractéristiques similaires en fonction des relations extraites de leurs contenus. En général, ces techniques sont adéquates quand les objets à analyser sont facilement traitables, comme c'est le cas des documents HTLM, et quand n'intervient pas beaucoup la subjectivité de l'usager (par exemple, pour voir si deux documents parlent du même thème). Une application claire dans ce sens est le filtrage de documents à partir des documents qu'a lu l'usager. Il existe plusieurs outils, certains de nature commerciale comme Autonomy (http://www.autonomy.com) et d'autres avec code ouvert et de libre distribution comme Rainbow (http://www.cs.cmu.edu/~mccallum/bow/rainbow/) que nous avons utilisés dans nos projets.

Par ailleurs, les techniques de filtrage collaboratif se basent sur la prise de préférences et des opinions des usagers, sur le fait de les classifier et les combiner en utilisant certains algorithmes, résultant de plusieurs groupes d'usagers de goûts similaires. A partir des groupes auxquels peut être associé un usager particulier, on peut déduire son intérêt pour un élément particulier. A la différence du filtrage par contenu, la réponse est déterminée par le comportement du groupe au lieu d'une simple correspondance pour un profil d'usager. Ces techniques sont adéquates surtout pour recommander des produits simples et homogènes comme les livres, la musique, les films, pour lesquels il y a un degré important de subjectivité de l'usager. Parmi les outils les plus connus basés sur un filtrage collaboratif se trouvent *Firefly* (MIT Media Lab, www.firefly.com) qui permet d'obtenir des recommandations musicales en se basant sur la formation de groupes d'usagers avec des goûts similaires, et *GroupLens* (commercialisé par NetPerceptions, www.netperceptions.com), une machine de recommandations à laquelle on accède par interfaces programmables à travers desquelles les applications peuvent envoyer des qualifications réalisées par les usagers et recevoir des prédictions.

Dans le système PSI3, chaque usager a un agent personnel (*Personal Agent*, PA). Les usagers avec des intérêts similaires sont regroupés en communauté. Chaque communauté est caractérisée et représentée par un agent de communauté (*Community Agent*, CA). Ces agents de communauté, d'une part, permettent d'implanter des politiques de gestion des usagers dans la communauté et, d'autre part de réduire le trafic de messages dans la diffusion de documents d'intérêt aux divers usagers et l'évaluation des documents suggérés par les agents personnels.

Ici, aussi bien les agents personnels (un pour chaque usager) que les agents de communauté gardent un profil basé sur les documents qui ont été considérés comme bons par l'usager ou la communauté. De cette façon les agents peuvent filtrer de nouveaux documents par rapport à l'expérience passée. Mais, en même temps, tous coopèrent pour réaliser un filtrage collaboratif

dans chaque communauté virtuelle. La combinaison de deux techniques de filtrage a plusieurs avantages. D'abord, elle permet de réduire la participation directe des usagers dans le processus d'évaluation. Ceci est intéressant parce que la pratique démontre qu'il est difficile que ceux-ci collaborent de manière explicite à l'évaluation de l'information qui leur est présentée. Il améliore aussi l'indice de recommandation des documents pour tous les usagers, autant ceux qui ont un profil bien développé que ceux qui sont récents dans le système. Il améliore enfin la performance globale du système parce que il n'est pas nécessaire pour tous les agents de filtrer tous les documents, et que les agents de communauté permettent de réduire le trafic d'information.

En prenant comme exemple le processus d'évaluation de documents, qui implique l'agent personnel qui suggère un document de possible intérêt pour la communauté (*Initiator*), l'agent de la communauté (*Community*), et plusieurs agents personnels à ceux à qui ont sollicité l'évaluation du document (qui peuvent décider par eux mêmes ou demander à l'usager correspondant), ce dit processus s'organise autour de flux de travail ou *workflow* (Figure 18).

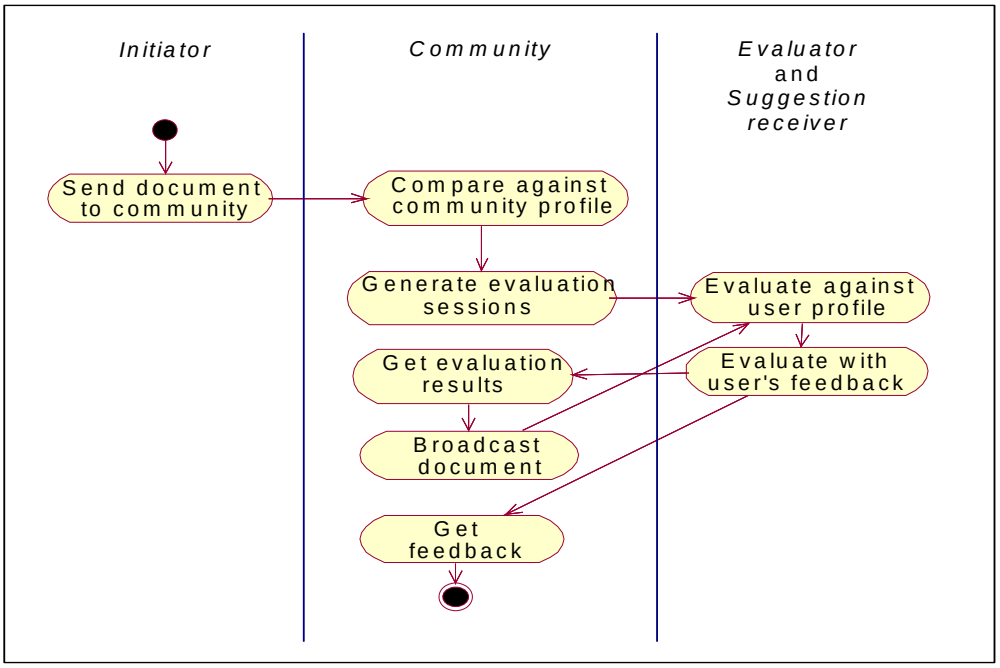


Figure 18. Workflow réalisé par les agents dans un processus de filtrage collaboratif de documents

L'idée de flux de travail vient du travail réalisé dans MESSAGE dans la modélisation des organisations [71], aspect qui, comme nous l'avons commenté, est fondamental quant il s'agit de systèmes avec un grand nombre d'agents. Ce qui est intéressant de l'application de cette idée dans PSI3 est que les flux de travail peuvent se configurer avec plusieurs paramètres, de manière à ce que leur application soit assez flexible. Par exemple, au niveau des paramètres (1), algorithmes (2), et règles (3 et 4) :

1. Numéro des membres de la communauté à qui on va solliciter une évaluation.
2. Les critères de décision pour l'évaluation autonome des documents. Ils déterminent si un document est approprié pour un profil (d'usager ou de communauté). Ici, un profil est un ensemble de documents évalués positif, ce qui détermine une catégorie, et le degré de pertinence d'un document pour un profil est déterminé par l'algorithme *k-nearest neighbour* avec extraction de termes [124]. Ce degré doit être plus important qu'un seuil configurable (voici un autre paramètre pour le workflow).
3. Les critères pour considérer les résultats de l'évaluation. Une fois que les usagers ont évalué les documents, ces critères doivent être utilisés pour décider de continuer ou non le processus de dissémination du document. Le critère le plus simple est de considérer que le nombre des évaluations positives est plus important que celui des évaluations négatives.
4. La politique d'actions à prendre par rapport aux usagers qui ignorent les documents. Par exemple, en augmentant ou diminuant une valeur d'appartenance à la communauté.

Spécialement avec les règles sociales, il est possible de déterminer des politiques de gestion de membres dans la communauté. De cette façon, une communauté peut éviter les usagers nuisants, ceux qui par exemple ne contribuent pas avec des évaluations positives (peut être parce qu'ils ne sont pas vraiment intéressés au sujet de la communauté) ou ceux qui font spam (parce qu'ils proposent trop de documents pas intéressants, ou que de la publicité). Quelques résultats d'expérimentation avec ce type de règles son décrits en [86] et plus récemment en [84].

2.5 La perspective Interaction

Les interactions déterminent le comportement des agents en montrant quelle est la réaction attendue quand on agit sur eux (par exemple, en envoyant un signal ou un message, ou en modifiant un espace partagé). Alors, les interactions sont très liées aux buts des agents et à l'exécution des tâches.

Le niveau d'abstraction avec lequel les interactions sont définies change de l'analyse à la conception. Dans le Processus Unifié, par exemple, l'analyse commence avec la définition de cas d'utilisation qui déterminent les interactions clé, et aussi quelques diagrammes de séquence ou collaboration, ou bien des diagrammes d'activité, pour montrer l'évolution de système. Pendant la conception, ces interactions sont détaillées avec des diagrammes complets de séquence ou collaboration, et des diagrammes de classes qui montrent les interfaces pertinentes.

Dans le domaine des agents ce n'est pas si simple. Même si l'analyse peut se réaliser d'une façon similaire, dans la conception il faut considérer :

1. Les acteurs qui participent à l'interaction. Si l'on considère un agent ou un rôle, un acteur devrait montrer quel est son intérêt dans l'interaction. Le fait de considérer la motivation pour participer à une interaction est consistant avec le modèle d'agent en accord au principe de rationalité, comme c'est indiqué dans la section 2.2.
2. La définition des unités d'interaction. La nature de chaque unité d'interaction détermine comment le récepteur devrait la traiter. Une unité d'interaction peut être simple comme un message ou l'addition d'un item dans un espace de tuples [139].
3. L'ordre imposé sur les unités d'interaction. Les unités d'interaction sont arrangées d'après un protocole standard ou particulier. Il y a plusieurs moyens de représentation de l'ordre des unités d'interaction : le plus courant sont les diagrammes de séquences de messages, mais il y a d'autres formalismes plus élaborés telle que le réseaux de Petri [48][123], ou bien les diagrammes de protocoles AUML [6].
4. Les actions réalisées avec l'interaction. Cela inclut :
    a. Le critère pour décider si on doit exécuter un non une tâche. Ce n'est pas suffisant que quelqu'un ait demandé l'interaction parce que les agents ont liberté de refuser son exécution (s'il n'y a pas de compromis).
    b. Les conséquences de l'exécution de la tâche : les résultats obtenus, les changements dans l'état mental de l'agent ou dans le monde.
5. La définition du contexte de l'interaction. Le contexte se compose des acteurs participants et leurs raisons, le propos de l'interaction, et les besoins pour chaque unité d'interaction, de façon que chaque acteur décide d'initier ou de collaborer à l'interaction.
6. Un modèle de contrôle. Il assure que l'interaction progresse comme elle est définie. Ce contrôle doit tenir en compte le fait qu'il est souvent possible d'avoir plusieurs interactions en parallèle.

Même si la quantité d'information pour décrire une interaction peut paraître impressionnante, la plupart de ces aspects peuvent se décrire avec un simple diagramme de collaboration, sauf pour les points 4 et 6, ce qui a motivé le méta-modèle d'INGENIAS.

2.5.1 Spécification des interactions

Le méta-modèle pour les interactions (Figure 19) couvre les aspects mentionnés avant. Ils sont classés en quatre aspects : le contexte, la nature, l'exécution et la représentation.

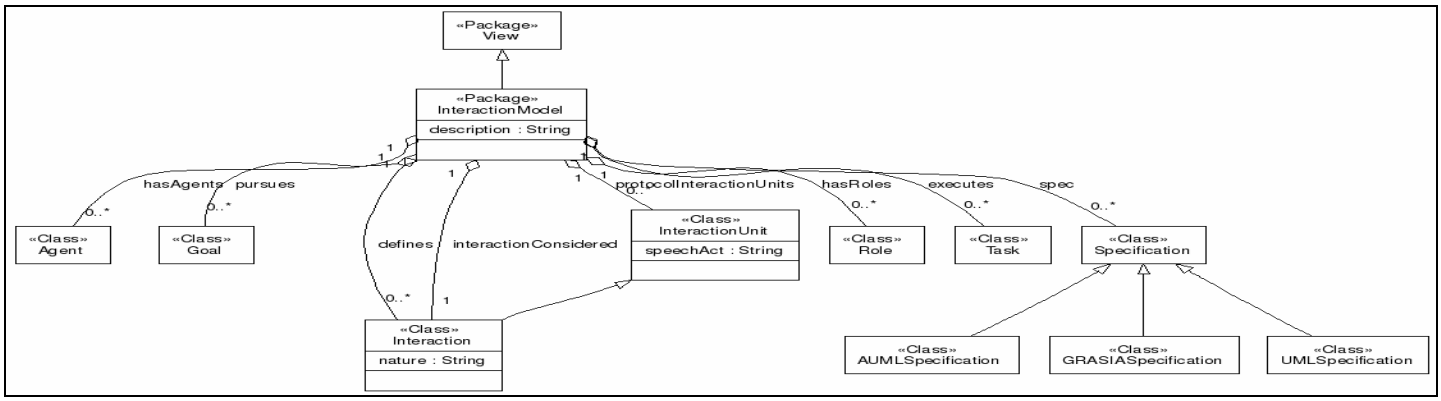


Figure 19. Éléments nécessaires pour décrire une interaction

Le *contexte* aide l'ingénieur à placer l'interaction dans le cadre d'un SMA. Il établit les conditions dans lesquelles l'interaction aura lieu. Cette information peut être utilisée pour implémenter le comportement de l'agent sous forme de règles de production ou pour générer les méthodes de validation de l'interaction. Dans le méta-modèle, le contexte est représenté par les méta-relations *IPursues, ICollaborates* et *IInitiates* (voir Figure 20) : *IPursues* pour signaler les buts (goals), *IInitiates* pour indiquer qui démarre l'interaction, et *ICollaborates* pour associer le reste des participants (il est possible d'engager plusieurs agents dans une interaction).

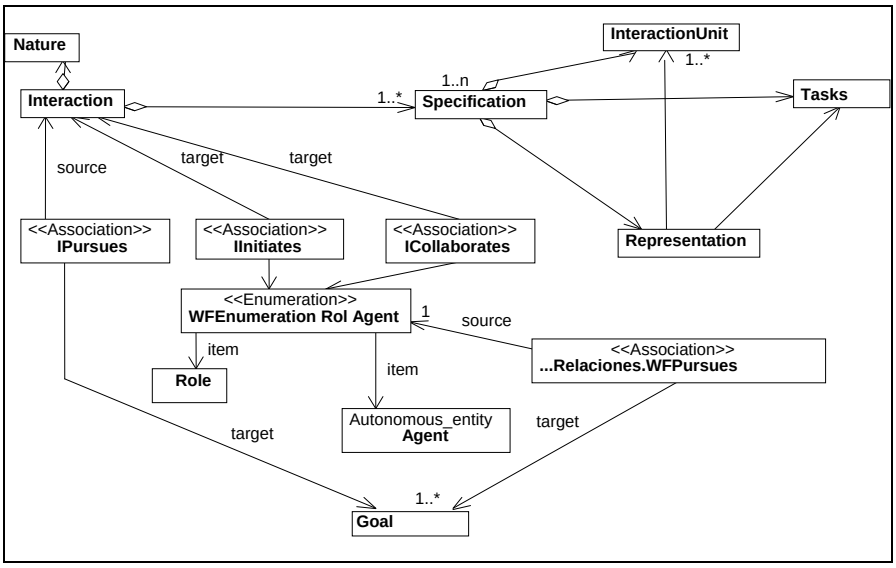


Figure 20. Contexte de l'interaction

La *nature* de l'interaction indique le type d'interaction par rapport à son contrôle, i.e., quelle coordination. La coordination est la gestion des dépendances entre activités, et le sujet d'étude de la théorie de la coordination [122]. Comme taxonomie de modèles de coordination d'agents, INGENIAS adopte celle de Huhns (1999), dans la Figure 21. La nature de l'interaction détermine quels algorithmes et outils peuvent guider le développeur. Si il s'agit d'une planification, l'ingénieur peut regarder les algorithmes de coordination comme GPGP [40] ou d'autres solutions [47]. La nature de l'interaction montre aussi quels éléments doivent être considérés. Par exemple, s'il s'agit de une compétition, il devrait exister dans l'interaction une référence à l'objet de discussion.

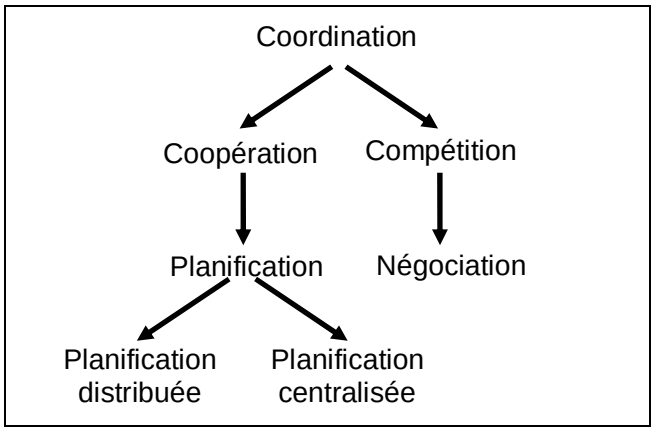


Figure 21. Une taxonomie de la coordination du comportement des agents

L'*exécution* de l'interaction fait référence aux activités nécessaires pour réaliser l'interaction. Dans l'exécution il y a un ordre qui établit le protocole de l'interaction.

Finalement, la *représentation* prend l'information sur l'exécution et la nature pour créer une formalisation graphique ou textuelle, utilisable par les ingénieurs. Par exemple, une négociation qui suit le protocole *contract-net* peut se représenter avec un diagramme de protocole AUML ou plusieurs diagrammes de séquence UML.

2.5.2 Animation des interactions avec l'IDK

Le module *JADE Organization*, qui est distribué avec l'IDK (voir section 3.2), facilite l'animation des spécifications de SMA. Ce module permet de voir l'évolution du comportement des agents pendant l'exécution des interactions. La dynamique de comportement détermine l'évolution des entités comprises dans l'état mental de l'agent et les tâches qui peuvent exécuter les agents. Cela illustre bien la relation entre la perspective Interaction avec celles d'Agent, Organisation et Buts-Tâches.

Ce module réalise une transformation des modèles SMA INGENIAS en un modèle computationnel, dans ce cas pour la plate-forme JADE. Pour définir cette transformation il faut d'abord identifier la représentation computationnelle appropriée, particulièrement pour chaque entité du méta-modèle Interaction. Il y a plusieurs aspects à tenir en compte pour obtenir un modèle computationnel. D'abord, il faut sélectionner une architecture d'agent. Il y a plusieurs types d'agents, comme se montre dans les révisions sur SMA, par exemple, dans le chapitre 2 [21] en [30]. Après, il faut déterminer quels aspects de l'interaction vont apparaître dans la conception et quelle représentation computationnelle tiendront. Les aspects considérés et sa représentation computationnelle sont très influencés par l'architecture d'agent choisie. La table de la Figure 22 montre une représentation computationnelle pour quelques entités du méta-modèle de la Figure 20. On peut facilement reconnaître ici que l'interprétation est destinée à une architecture d'agent délibératif, dans laquelle le contrôle est défini avec des règles.

Entité de méta-modèle	**Représentation computationnelle**
Ordre des activités de communication	Une ou plusieurs machines d'états finis pour capturer l'ordre d'exécution des unités d'interaction.
Interaction Unit	La livraison d'un message ACL, un RPC, une invocation de protocole.
IExecutes	Une règle associée avec les conditions de collaboration et initiation. Quand ces conditions sont satisfaites, la tâche s'exécute.
ICollaborates	Une condition qui représente le changement espéré dans l'état mental pour permettre la collaboration. Voir *IExecutes*
IInitiates	Une condition qui représente le changement espéré dans l'état mental pour permettre l'initiation d'une unité d'interaction. Voir *IExecutes*
Rôle	Une interface. Les représentations computationnelles des entités associés, comme les buts, sont inclues dans la représentation computationnelle de chaque agent qui joue le rôle.

Figure 22. Représentation computationnelle de quelques entités du méta-modèle d'interaction

En parallèle avec la définition de la représentation computationnelle, il faut déterminer la procédure de transformation des entités du méta-modèle en entités computationnelles. Celle-ci est complexe parce qu'il faut considérer aussi les relations avec d'autres perspectives du système, la complexité associée aux représentations computationnelles sélectionnées, et l'adaptation de tout le processus en une architecture d'agent. Tout cela demande, normalement, une conception plus détaillée. Par exemple, la gestion de plusieurs conversations en même temps ou les dépendances entre l'ordre d'exécution et la gestion des interactions. Dans ce sens, chaque activité doit être détaillée pour arriver au niveau de détail nécessaire à la transformation. Le résultat final du processus de transformation serait un ensemble d'entités computationnelles qui puissent être intégrées dans une architecture d'agent concrète. Ce processus de transformation se complète avec les transformations d'éléments d'autres modèles que couvrent

d'autres aspects du système. Tout ensemble, ils constituent un flux d'activités pour le développement d'un SMA complet, comme s'explique dans le chapitre 4.

Le module *JADE Organization* dans la distribution de l'IDK montre une implémentation d'une spécification INGENIAS. Le résultat est un système qui permet l'animation de la spécification en fonction des interactions et tâches que les agents vont exécuter. Comme indique le manuel de l'IDK [82], pour réaliser les transformations de ce module il faut avoir la spécification des éléments suivants :

- Un diagramme d'agent pour chaque agent, représentant son état mental initiale et ses buts.
- Un diagramme d'organisation, qui montre le workflow des tâches. Le workflow est représenté comme des tâches connectées par des relations *WFConnects*. Cette relation déclare que la sortie d'une tâche est utilisée comme l'entrée d'autre. Il faut déclarer aussi les relations *WFResponsible* pour indiquer quels rôles sont responsables de réaliser les tâches.
- Diagrammes buts-tâches pour identifier les tâches qui peuvent conduire à la satisfaction des buts.
- Un diagramme d'interaction pour chaque interaction. Une interaction est représentée par une entité d'interaction dans laquelle plusieurs rôles participent. Actuellement l'association directe des agents aux interactions n'est pas supportée par l'IDK. À la place, il faut utiliser des rôles et les associer avec des agents dans un diagramme d'organisation.
- Un diagramme d'interaction qui montre le protocole. Actuellement le module utilise la représentation de diagrammes de collaboration INGENIAS, qui donne plus d'information que les diagrammes AUML, comme par exemple, les tâches impliquées dans l'interaction. Les diagrammes AUML ne donnent pas assez d'information pour intégrer les interactions avec le reste des éléments du système.

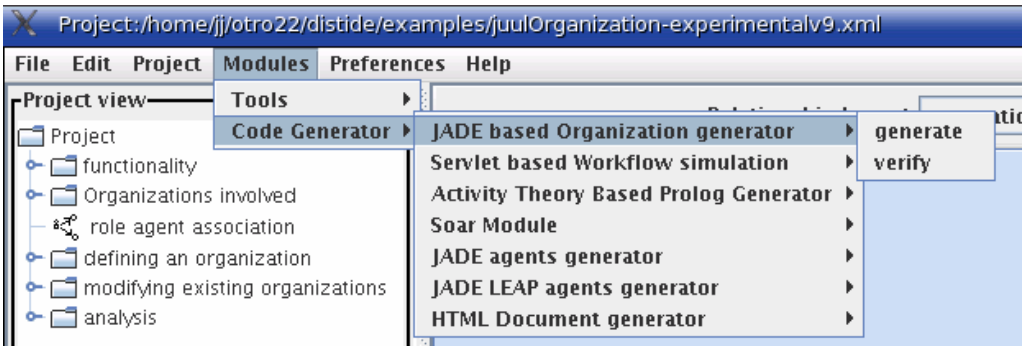
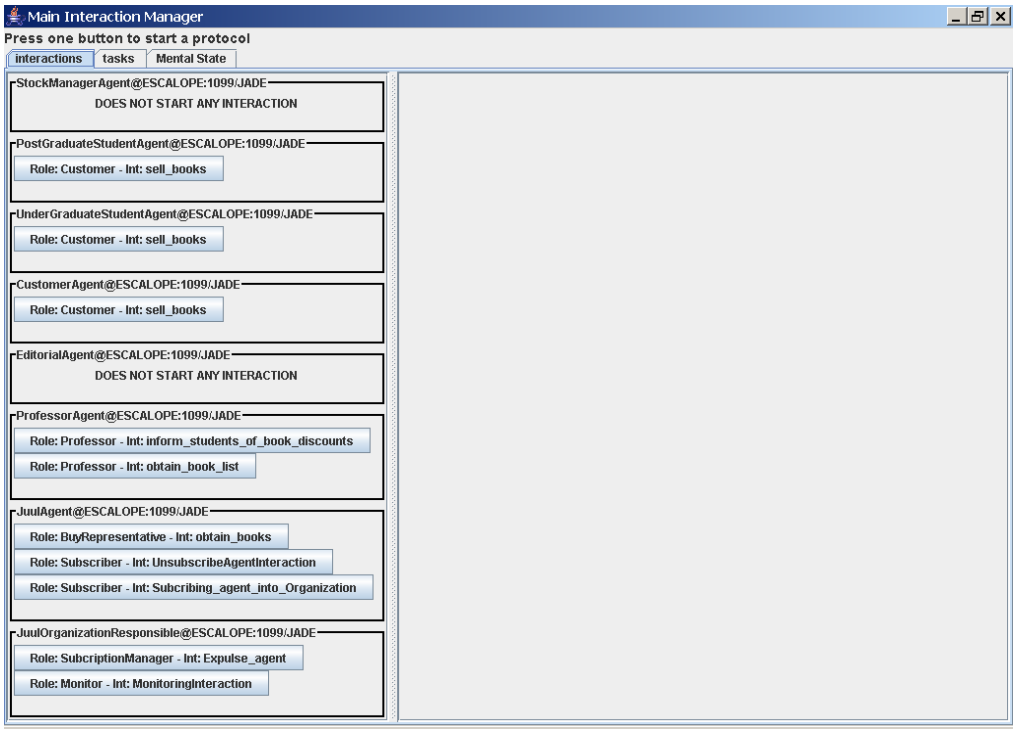


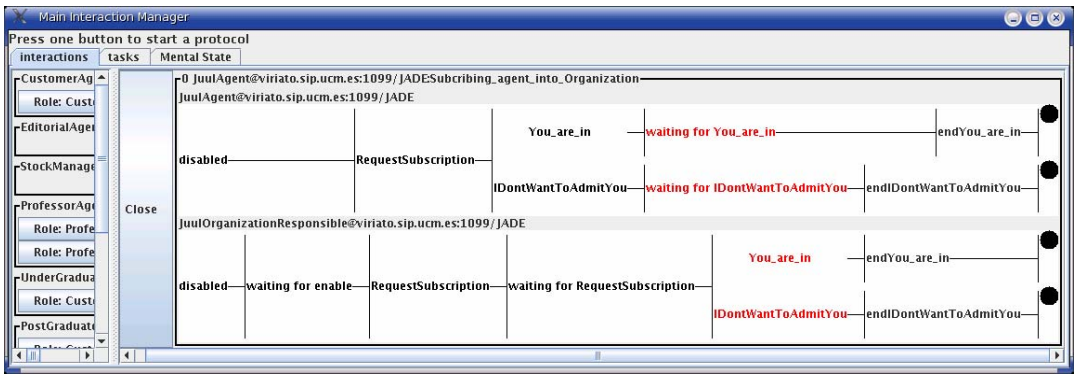
Figure 23. Invocation du module de génération de code *JADE Organization*

Avec tous ces diagrammes on peut générer le code, en invoquant le module *JADE Organization*, comme le montre la Figure 23.

Le résultat de l'invocation de ce module est un ensemble de classes Java qu'il faut compiler et exécuter. Pour cela, la distribution de l'IDK offre plusieurs scripts *ant* : *compjademasorg*, pour tout compiler, *runjade*, pour démarrer la plate-forme JADE, et *runjademasorg*, pour démarrer le SMA avec une interface graphique pour gérer l'animation du SMA.



(A) Situation initiale



(B) Evolution d'une interaction

Figure 24. Interface pour la gestion des interactions

L'interface graphique est très intéressante pour apprendre comment évoluent les agents et comment se déroulent leurs interactions. Elle compte trois sections :

- Gestion des interactions. Il n'y a que quelques agents qui peuvent démarrer une interaction et uniquement certaines interactions peuvent démarrer immédiatement. La Figure 24 montre à gauche les agents du système avec des boutons pour les interactions qui peuvent démarrer. Quand l'utilisateur clique sur un bouton, l'agent cherche des participants appropriés, en regardant les rôles spécifiés pour l'interaction, et démarrera l'interaction si les conditions de son état mental le permettent. Une fois démarrée, l'interaction va continuer jusqu'au point où certaines conditions de l'état mental ne sont

plus remplies. En ce moment l'interaction s'arrête mais elle peut continuer une fois que les conditions d'état mental changent. L'évolution de l'interaction est représentée par des machines d'état. Si l'interaction n'évolue pas il peut s'agir du fait que l'agent soit en attente d'une tâche qui doive produire certaines entités mentales.

- Gestion des tâches. La Figure 25 montre l'interface pour la gestion des tâches. En tout moment, elle montre uniquement les tâches qui contribuent à l'exécution d'un but. Ce modèle est lié au principe de rationalité qui est sous-jacent en INGENIAS. En cliquant sur le bouton radio à droite d'une tâche, on peut examiner les types d'entrées que la tâche attend et les sorties qu'elle peut produire. La représentation des entrées et sorties se réalise comme diagrammes d'agent, qui montrent les entités mentales concernés. Il est possible pour l'utilisateur d'ajouter, modifier ou supprimer ces entités mentales. De cette façon on peut modifier le comportement d'une tâche pendant l'animation et déduire ce qu'il faut ajouter ou modifier dans la spécification du SMA.

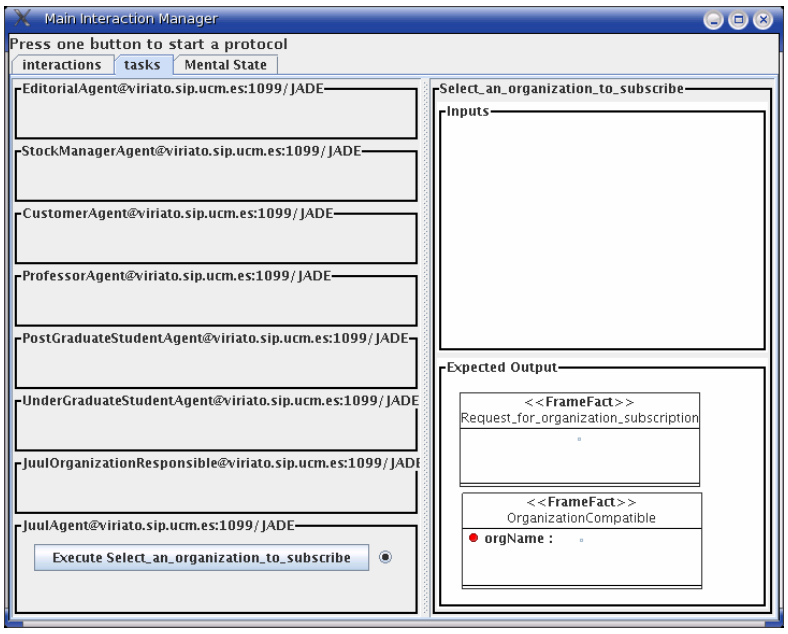


Figure 25. Interface pour la gestion de tâches

- Gestion de l'Etat Mental. La Figure 26 montre comment on peut examiner les entités qui constituent l'état mental de chaque agent du SMA. L'utilisateur peut manipuler ces entités de la même façon que pour les tâches.

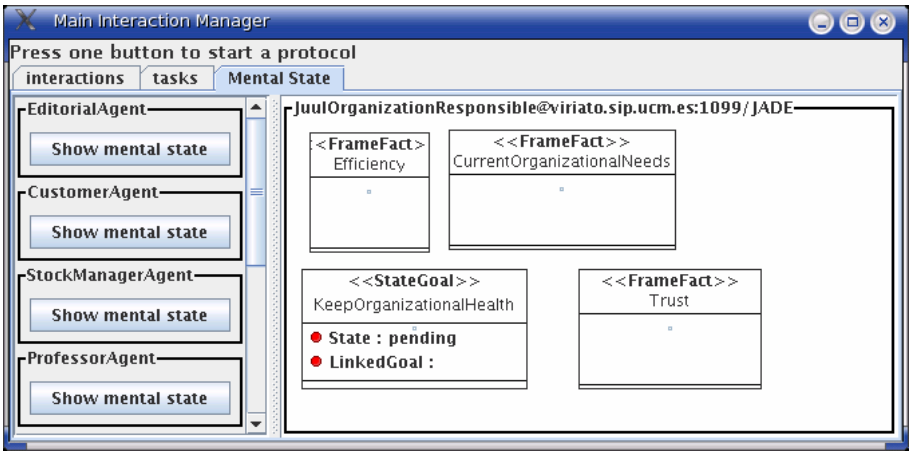


Figure 26. Interface pour la gestion de l'état mental des agents

2.6 La perspective Environnement

La perspective environnementale détermine les entités qui interagissent avec le SMA. Cela ne veut pas dire une description de ces entités, comme les cas de Ferber [56], mais uniquement des aspects qui sont intéressants pour l'interaction des agents avec son environnement. Pour simplifier cette tâche, nous avons discrétisé l'environnement en deux sens : une catégorisation des entités qui sont pertinentes dans l'environnement, et les restrictions des interactions avec ces entités. Ainsi, l'environnement se compose d'applications, d'agents, et de ressources, et la perception et l'actuation des agents sont limitées.

2.6.1 Spécification de l'environnement

Les applications et composants logiciels qui offrent un API (local ou distant) sont des éléments courants de l'environnement. Ils sont normalement utilisés pour modéliser les capacités de perception et agir sur l'environnement. Les applications produisent des événements qui peuvent être observés. Les agents définissent leur perception en identifiant les événements qu'ils peuvent percevoir. Les agents agissent aussi sur l'environnement en invoquant des procédures ou des méthodes offertes par les applications. Une application peut être etiqueté comme environnement (E) si elle enveloppe un logiciel externe (*wrapper*), ou bien comme interne (I) s'il s'agit des composants de SMA (comme librairies).

Les agents d'autres organisations sont considérés aussi comme partie de l'environnement. Les agents d'une organisation peuvent avoir besoin d'interagir avec des agents d'autres organisations pour satisfaire leurs buts.

Finalement, les ressources sont les éléments nécessaires pour l'exécution de tâches, qui n'ont pas d'API. Des exemples de ressources sont les descripteurs de fichiers, la mémoire et le processeur. Les ressources sont assignées aux agents ou aux groupes dans l'organisation.

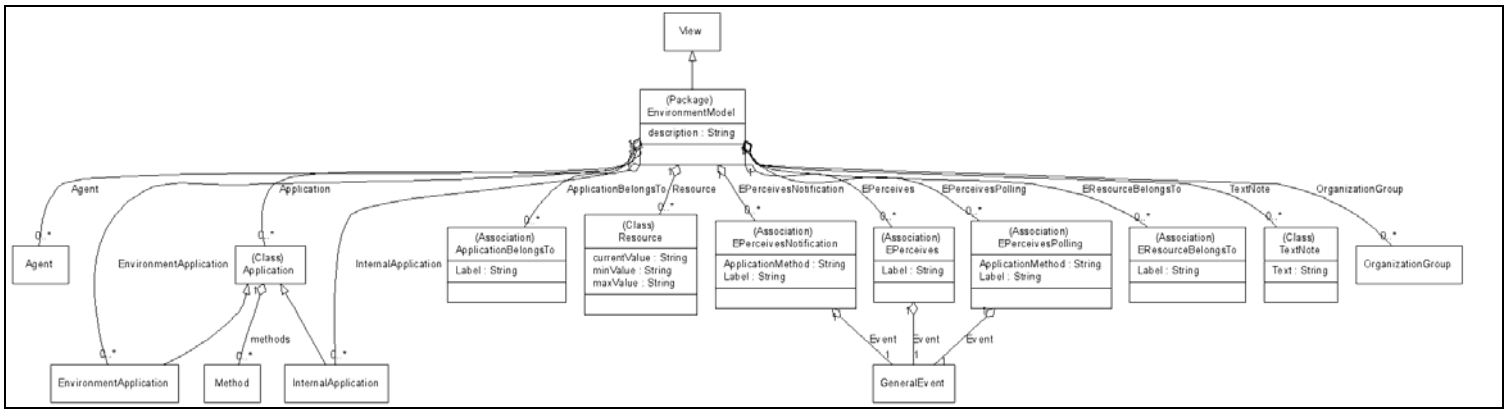


Figure 27. Le méta-modèle de l'environnement

La Figure 27 montre qu'un agent définit sa perception à travers des applications. La perception peut se concevoir comme la notification des événements qui arrivent à l'application ou comme *polling* de l'état de l'application à travers ses méthodes. Certains systèmes ont besoin d'autres types de perception mais pour l'instant notre expérience montre que ce modèle est suffisant.

Les applications et ressources peuvent être gérées par les groupes, les rôles ou les agents. Quand une ressource est assignée à un groupe, un rôle ou un agent, il devient responsable du contrôle de son utilisation et accès. Dans le cas d'un groupe, l'accès est garanti pour tous les agents du même groupe. Les agents externes peuvent interagir avec un agent du groupe pour essayer de gagner l'accès.

2.6.2 Intégration avec de composants logiciels et systèmes hérités

Normalement un système n'est pas conçu comme une entité isolée et à partir de zéro. Il est commun de se trouver avec le besoin d'étendre ou d'interagir avec des systèmes existants. En plus, il est très courant de réutiliser des composants existants comme partie du nouveau logiciel. Ces aspects ont été pris en compte en INGENIAS car dans un projet habituel ils sont toujours présents.

Le code source peut être considéré comme partie d'une application, de point de vue INGENIAS. Par exemple, dans la Figure 28, deux composants de code Java, *FlightManagerInitialization* et *FlightManagerShutdown*, font partie d'un composant d'application, *FlightManager*. Ce dernier est vu par les autres éléments d'une spécification INGENIAS comme une classe avec des opérations qu'on peut invoquer.

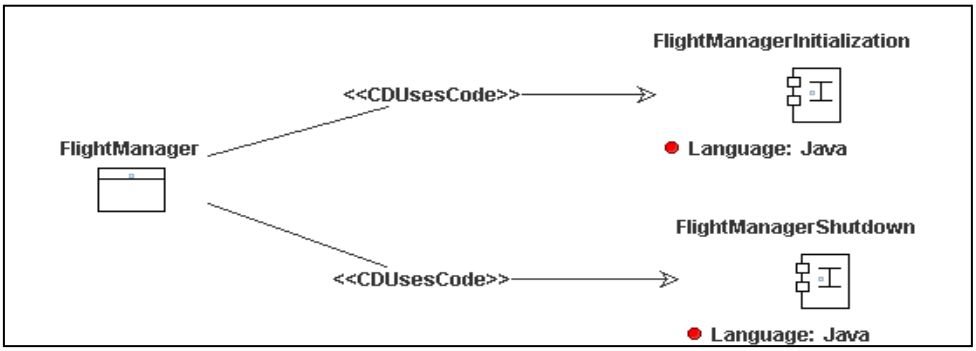


Figure 28. Exemple d'usage des composants de code source

De cette façon il est possible de définir comment plusieurs éléments de code externe sont intégrés dans un SMA, comme le montre la Figure 29. L'idée est de définir le composant application comme réalisé par différents composants logiciels externes. Ces composants d'application, telles que *UserGui* ou *EventsDatabase*, sont accessibles par les autres composants d'une spécification SMA en INGENIAS. Par exemple, une tâche dans un workflow peut être implémentée par l'un de ces applications ou bien l'invoquer.

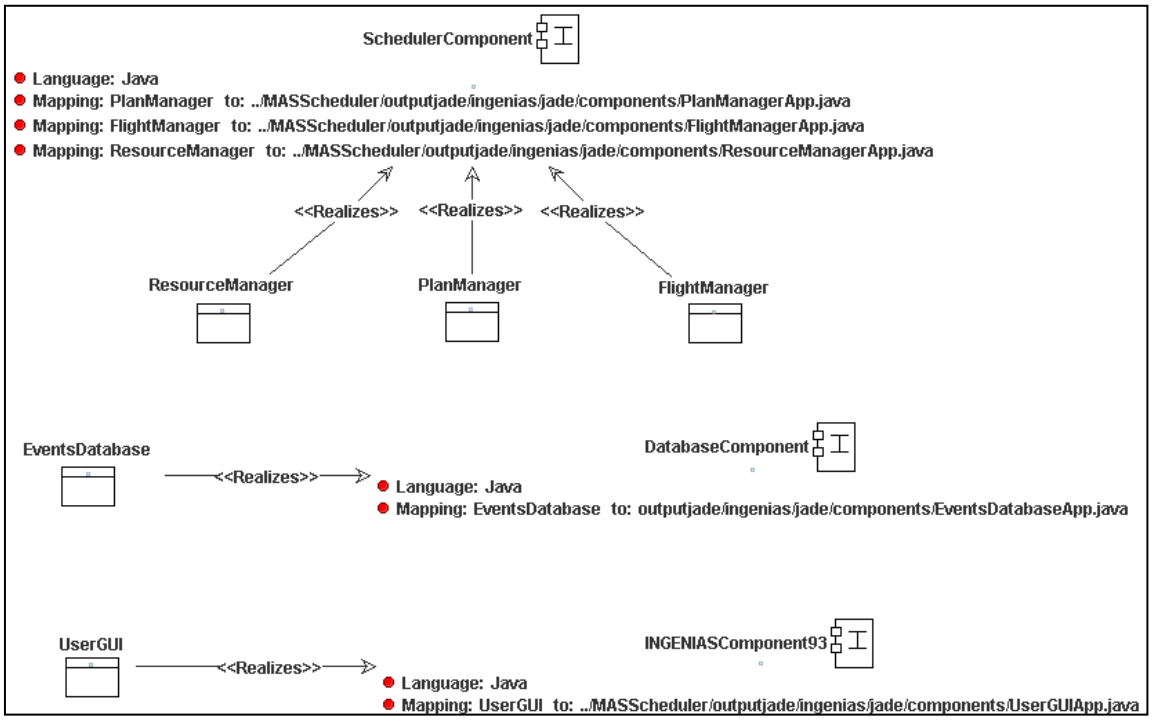


Figure 29. Intégration de code source non généré par l'IDK

2.7 Dépendances entre perspectives

Les ingénieurs devraient avoir conscience que certains éléments peuvent apparaître dans plusieurs diagrammes. Cette répétition des entités en différentes perspectives induit des dépendances entre les perspectives. Cette caractéristique est la conséquence de la définition de plusieurs perspectives du système et peut être source d'inconsistances dans la spécification d'un SMA. Par exemple, une tâche peut apparaître dans une perspective agent, une perspective tâche/but, une perspective organisation et une perspective interaction. Donc, la définition complète d'une tâche peut demander plusieurs diagrammes. Si l'ingénieur ne crée pas tous ces diagrammes, la spécification du SMA peut être incomplète. D'un autre côté, si l'ingénieur crée tous les diagrammes nécessaires et si une tâche est assignée à un rôle dans un diagramme de la perspective organisation et à un autre rôle dans un diagramme de la perspective agent, cela pourrait être interprété comme une inconsistance de la spécification.

Pour traiter ces dépendances, la thèse de Gómez-Sanz [79] propose quelques tests à appliquer à chaque perspective. Ces tests sont formulés comme des règles qui montrent pour chaque élément d'une perspective les autres perspectives qui peuvent être utilisées pour complémenter sa définition. Une partie de ces tests est incorporée dans l'IDK comme une forme de validation des spécifications : les modèles d'un SMA doivent être conformes aux méta-modèles.

2.8 Conclusion

Le développement des méta-modèles d'INGENIAS s'est réduit à notre groupe de recherche, *grasia!*, essentiellement pour obtenir un noyau initial de concepts stables validés par des cas d'étude. En ce moment nous envisageons que d'autres groupes de recherche contribuent à son évolution. Ceci est un défi car nous ne possédons pas l'expérience dans le développement collaboratif de langages de spécification. Pour le moment, la méthode de travail est de personnaliser l'IDK avec différents méta-modèles pour pouvoir valider son intérêt. Jusqu'à présent nous maintenons trois lignes d'évolution :

- Systèmes holoniques, développés par l' Université Polytechnique de Valencia dans la thèse d'Adriana Giret (2005), dirigée par le professeur Vicent Botti [75].
- La modélisation d'agents avec la Théorie d'Activité. Ceci est en train de se développer dans le cadre d'une collaboration entre notre groupe de recherche et le groupe de recherche LTCS de la Universidad Nacional de Educación a Distancia (UNED), du professeur Beatriz Barros. Un résultat de ce travail est la thèse de doctorat de Rubén Fuentes [62] dirigée par Jorge J. Gómez-Sanz et Juan Pavón.

- La définition de langages de simulation à base d'agents. Il s'agit d'utiliser les concepts d'INGENIAS pour construire les outils de modélisation graphique de systèmes de simulation, adaptés aux domaines d'application spécifiques [164].

L'intégration des concepts résultants n'est pas difficile grâce à l'utilisation des méta-modèles. Cependant, la valeur pragmatique des nouveaux concepts et des éléments dans le langage de modélisation demande du temps pour la validation dans des cas expérimentaux réels.

Par ailleurs, les capacités d'extensibilité du langage de modélisation nous amène finalement à la définition de langages spécifiques du domaine d'application. Comme on le verra dans le chapitre 4, l'approche MDD focalise sur les modèles comme éléments essentiels pour le développement. Ainsi, la disponibilité des langages de modélisation qui permettent d'exprimer le problème et la solution dans des termes familiers aux usagers finaux va raccourcir le gap entre l'ingénieur de logiciel et les experts du domaine d'application. Nous considérons que les concepts offerts par le paradigme agent, dans ce sens, sont plus proches de ce but que d'autres approches, comme c'est le cas, plus spécifiquement, d'UML. Voici un fort avantage de la modélisation orienté agent.

3. OUTILS : INGENIAS DEVELOPMENT KIT (IDK)

Pour une application effective de la méthodologie de développement de SMA il est nécessaire de disposer d'un ensemble intégré d'outils qui la supporte. Ces outils devraient prendre en compte plusieurs faits. Dans le cas d'INGENIAS, en tenant compte de l'approche MDD (chapitre 4), les outils doivent supporter les activités d'édition, de vérification et de transformation des modèles. La méthodologie et les outils de développement doivent aussi évoluer dans le temps en incorporant de nouveaux services. De plus, les types de projets qui vont se développer seront à différente échelle, quant à leur complexité et leurs ressources. Enfin, étant donné que cette méthodologie cherche à intégrer différents résultats de recherche, ces outils doivent pouvoir s'adapter aux changements qui se produisent dans les méta-modèles, que ce soit pour l'extension ou pour la modification de concepts existants. Ce dernier point est le besoin le plus important de tous, puisqu'il oblige que toute l'infrastructure d'appui au développeur se base sur des méta-modèles qui peuvent évoluer dans le temps.

L'*INGENIAS Development Kit* ou *IDK* est un ensemble ouvert d'outils basés sur les méta-modèles d'INGENIAS qui ont été présentés dans le chapitre 2. L'IDK se compose d'un *éditeur* qui génère les spécifications et d'un framework de *modules* qui traitent ces spécifications, par exemple, pour la génération de code ou pour la validation et la vérification des propriétés. Ces éléments sont intégrés dans l'éditeur qui permet d'appeler les modules qui traitent les spécifications. La structure générale de l'IDK est présentée dans la Figure 30. L'éditeur permet de créer les spécifications (modèles) du SMA. Il applique les restrictions imposées par le méta-modèle. La section 3.1 explique comment il est construit à partir de la spécification des méta-modèles. Une fois générées avec l'éditeur, les spécifications des SMA peuvent être traitées par des modules. Ces modules utilisent le méta-modèle pour connaître la structure des spécifications et les parcourir pour extraire l'information nécessaire, comme il est décrit en la section 3.2. La section 3.3 montre les pas nécessaires pour construire ces modules et plus concrètement pour la génération de code en utilisant les templates. Et la section 3.4 discute sur l'état actuel de l'évolution de ces outils.

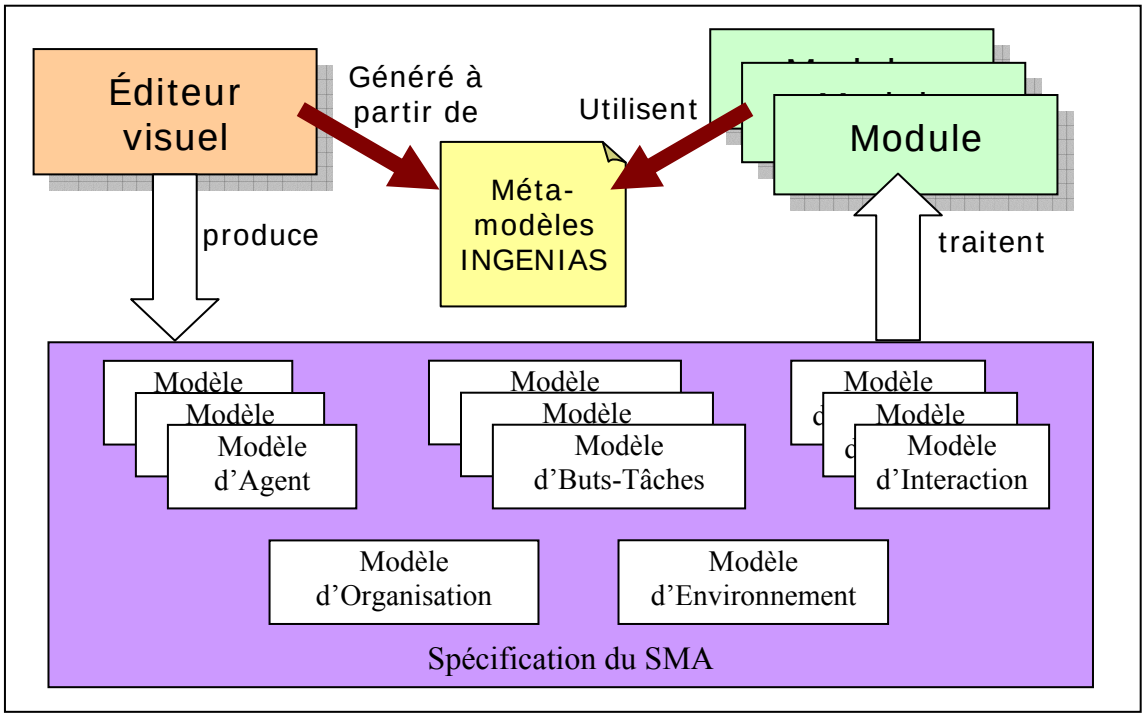


Figure 30. IDK : Éditeur + Modules

3.1 L'éditeur visuel d'INGENIAS

L'éditeur visuel (Figure 31) est un outil de modélisation qui permet de créer et de modifier des diagrammes pour travailler avec plusieurs perspectives d'un SMA, telles qu'elles ont été décrites dans le chapitre 2. Comme normalement il y a un nombre des diagrammes assez important, il est recommandable de les structurer en une hiérarchie de paquets, comme pour les spécifications UML.

L'IDK est associé à la définition de méta-modèles à travers une représentation en XML des concepts, des relations, des diagrammes de méta-modèle, ainsi comme associations à éléments de représentation graphique de chaque concept. De cette façon, il est possible de changer les méta-modèles sans affecter significativement la fonctionnalité de l'IDK. Cette caractéristique permet de découpler la méthodologie des outils et, en même temps, facilite l'expérimentation avec les concepts d'agent. La Figure 32 montre comment, à partir de la spécification des méta-modèles (en fichiers XML), des représentations graphiques des concepts (fichiers d'icônes) et d'une description des attributs de l'éditeur (avec par exemple les associations des entités de méta-modèle aux fichiers d'icônes), il est possible de générer une nouvelle version d'éditeur pour l'IDK. Pour ça il y a un interprète des descripteurs des méta-modèles et un template de l'éditeur (qui détermine l'aspect général et la distribution de ses éléments).

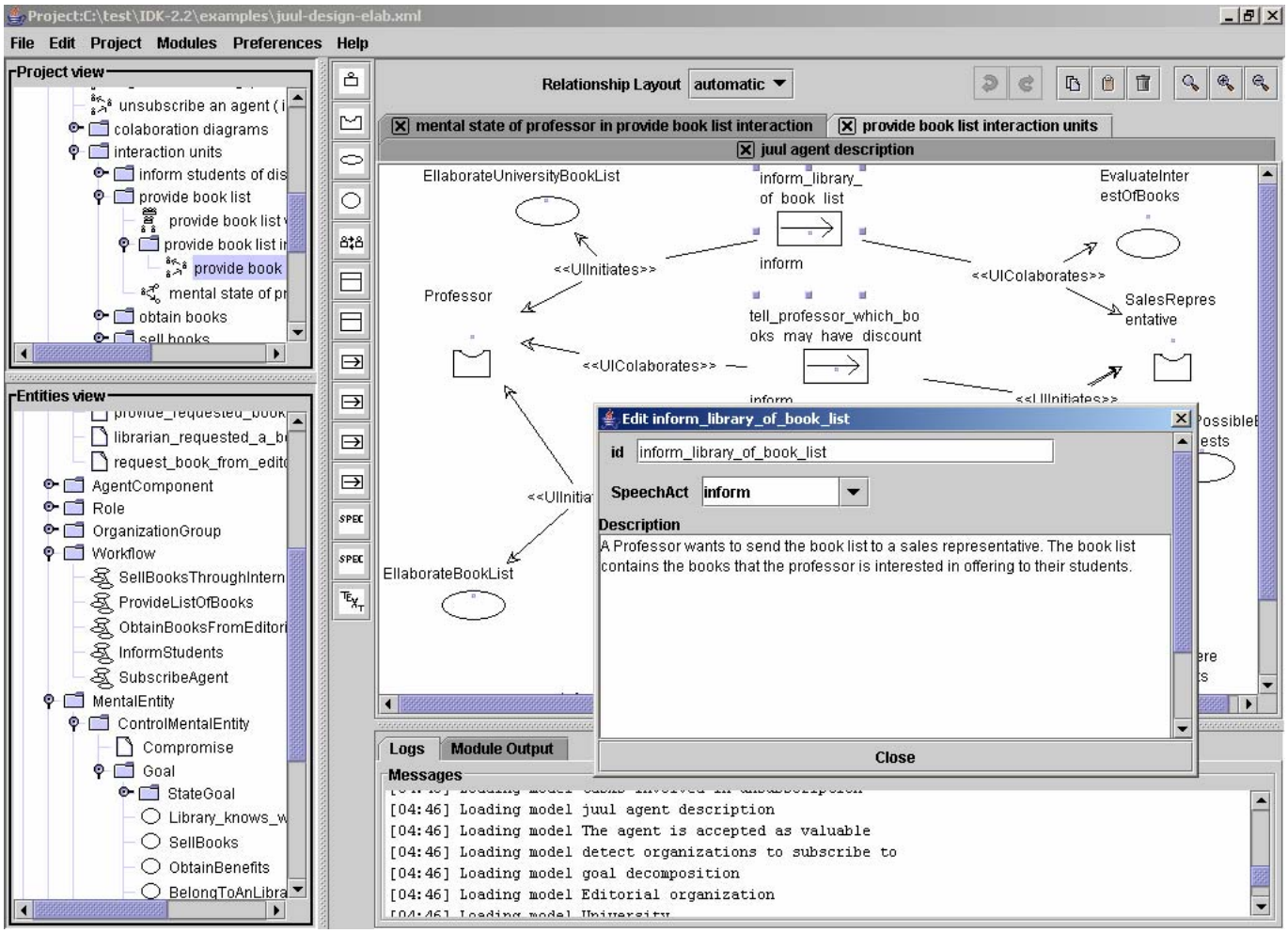


Figure 31. Éditeur visuel INGENIAS

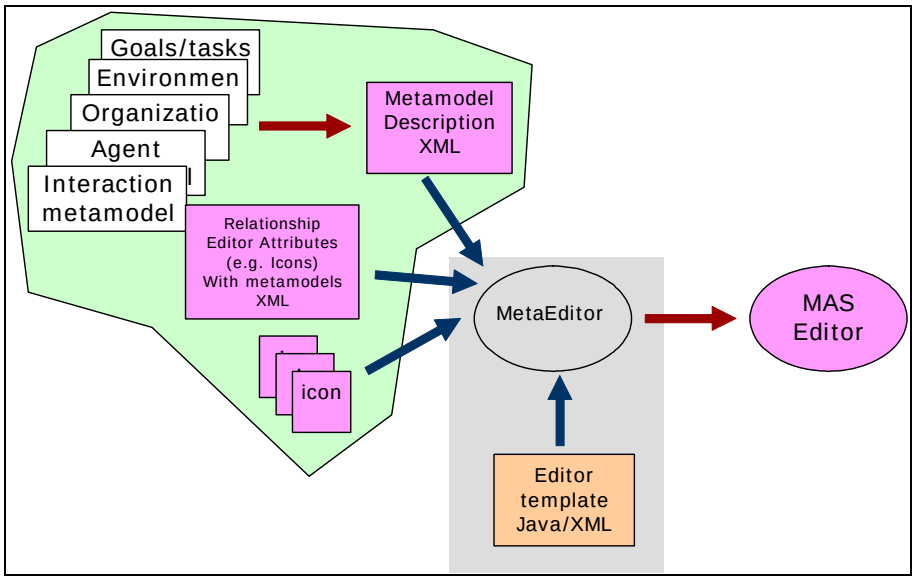


Figure 32. Construction de l'éditeur INGENIAS

Un des avantages de cette approche, c'est qu'à partir des changements dans la définition des méta-modèles, il est très facile de générer des nouveaux éditeurs, qui peuvent être personnalisés pour les besoins concrets d'un domaine d'application particulier. Un exemple de cette adaptation a été réalisé par Giret [76] pour le développement des systèmes holoniques.

3.2 Les modules IDK

L'éditeur est l'outil principal pour le développeur du SMA. Mais, dans une approche MDD il faut compter aussi avec des outils pour la vérification et la validation des modèles, et avec des moteurs de transformation pour dériver le code pour la plate-forme finale. Tout cela est supporté par les *modules* du IDK. Les modules, ou plugins, dans l'IDK sont des programmes qui traitent des spécifications pour produire une sortie :

- Code source. Il y a une infrastructure qui facilite la transformation des spécifications en code source. Celle-ci est basée sur la définition des templates pour chaque plate-forme cible et des procédures d'extraction d'information des modèles.
- Rapports. Les spécifications peuvent être analysées pour vérifier, par exemple, qu'elles ont certaines propriétés ou qu'une sémantique spéciale, définie par le développeur, est respectée, ou pour cueillir une information statistique sur l'utilisation des différents éléments.
- Transformations en autres modèles. De la même façon qu'il est possible de produire un code source, il n'y a pas de différence pour produire aussi d'autres modèles basés sur des méta-modèles différents.

Les modules sont construits sur un framework qui offre des facilités pour parcourir les spécifications, extraire l'information des spécifications, et mettre cette information en templates. Un module est normalement écrit en langage de programmation Java en suivant quelques instructions strictes pour permettre son intégration avec l'IDK. Par exemple, il est obligatoire d'implémenter des interfaces spécifiques, certaines procédures d'extraction d'information, l'utilisation des templates avec une certaine structure, l'empaquetage de tous les fichiers dans fichiers *jar*, et le déploiement du résultat en un dossier concret.

3.2.1 L'interface pour parcourir les spécifications

Pour réaliser les transformations, il faut d'abord parcourir les spécifications des modèles. Pour cela, l'IDK offre un API qui montre une spécification comme un ensemble de graphes. Les types d'éléments qui forment la spécification d'un SMA avec l'IDK sont (voir Figure 33) :

- *Project*, pour représenter la spécification de SMA (dans l'IDK on travaille avec projets, un pour chaque SMA à développer). Chaque projet de SMA est composé de plusieurs diagrammes.
- *Graph*, pour représenter chaque diagramme. Un graphe tient plusieurs entités liées par relations. Un graphe peut avoir aussi plusieurs attributs.
- *Entity,* pour représenter les éléments principaux d'une spécification. Chaque entité a plusieurs attributs. Certaines entités peuvent faire référence à des diagrammes.
- *Relationship,* pour connecter les entités. Chaque relation peut avoir deux ou plusieurs extrêmes, *GraphRole*. Chaque rôle peut avoir aussi des attributs. À noter que les relations sont n-aires.
- *Attribute,* peuvent être des entités ou faire référence à d'autres graphes.

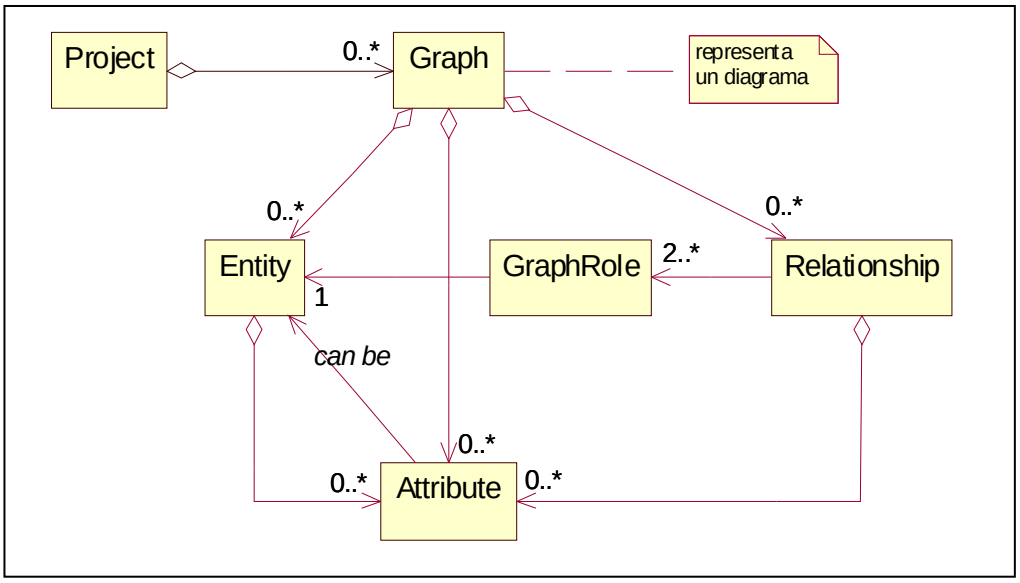


Figure 33. Structure d'information pour parcourir les diagrammes d'un SMA avec l'IDK

L'API pour parcourir les graphes offre un ensemble d'interfaces, comme montre la Figure 34. La classe *ingenias.generator.browser.BrowserImp* implémente un patron Singleton et sert comme point d'accès initial à la spécification. L'interface Browser offre alors les méthodes pour parcourir les diagrammes. Par exemple, pour obtenir toutes les entités de tous les diagrammes d'une spécification le code suivant fait le travail :

```
Browser browser=BrowserImp.getInstance();

Graph[] gs = browser.getGraphs();
StringBuffer result = new StringBuffer();

for (int k = 0; k < gs.length; k++) {
      Graph g = gs[k];
```

```
        result = result.append( "Diagramme " + g.getName()+ "\n");
        result.append(this.generateDiagramReport(g)+"\n");
}
System.out.println(result);
```

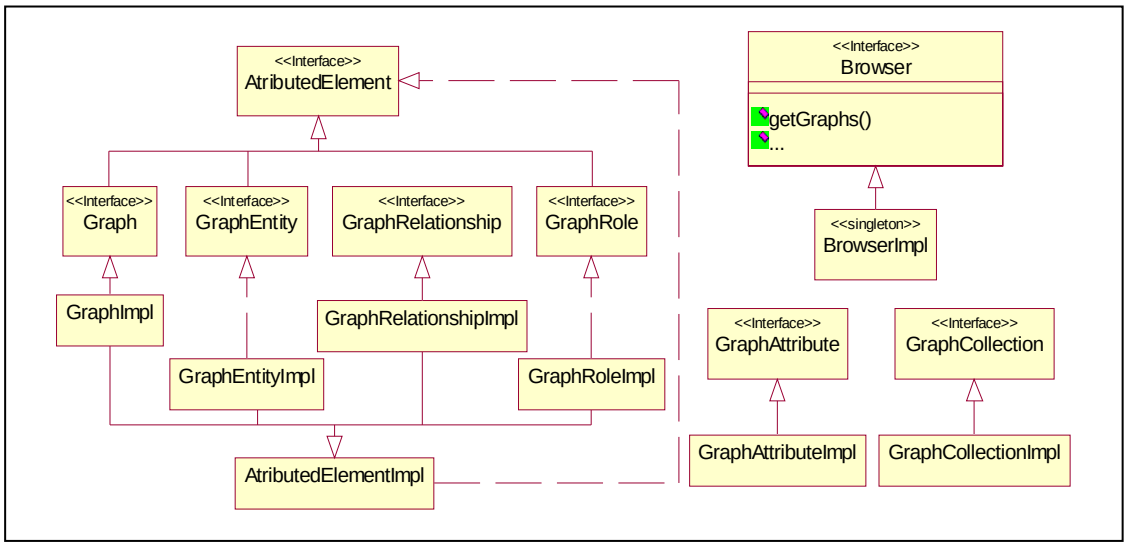


Figure 34. API pour parcourir modèles

Pour parcourir la spécification il faut définir un algorithme de parcours de graphe qui passe par les éléments de la spécification, assurer que tous les éléments nécessaires soient présents et extraire l'information des éléments requis. En effet, les interfaces *GraphEntity*, *Graph*, et *GraphRelationship* offrent les méthodes nécessaires.

3.2.2 Structure de templates

Les templates pour la génération de code ont d'une part le code source dans le langage de programmation de la plate-forme cible, et d'autre part les étiquettes qui signalent les endroits où il faut ajouter l'information du modèle. Ainsi, les templates peuvent se voir comme des documents XML qui satisfont le DTD de la Figure 35.

Chaque étiquette de cette DTD est conçue pour être remplie avec l'information du modèle :

- *program*, déclare le début de programme.
- *saveto*, déclare qu'il faut enregistrer un fichier. Cette étiquette est la dernière à être traitée et nettoie toutes les étiquettes qui n'ont pas été remplacées. *file* déclare le fichier et *text* le texte à enregistrer.
- *repeat*, déclare que le texte entre les deux étiquettes *repeat* doit se répéter.
- *v*, déclare l'insertion d'une donnée.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT file (#PCDATA | v)*>
<!ATTLIST file  overwrite (yes|no) #REQUIRED>
<!ELEMENT program (#PCDATA|repeat|saveto|v)*>
<!ELEMENT repeat (#PCDATA | saveto | v | repeat)*>
<!ATTLIST repeat id CDATA #REQUIRED>
<!ELEMENT saveto (file, text)>
<!ELEMENT text (#PCDATA | repeat | v | saveto)*>
<!ELEMENT v (#PCDATA)>
```

Figure 35. DTD pour l'information à remplir dans les templates

La Figure 37 montre un exemple de template avec ses étiquettes. Des avantages de ce langage par rapport à d'autres solutions, comme par exemple l'utilisation des transformations XSL, sont d'une part une simplicité majeure et d'autre part de pouvoir assurer que l'algorithme de génération de code se termine, ce qui n'est pas le cas avec XSLT (voir démonstration formelle en [85]).

Les données pour remplir les templates sont aussi structurées comme dans la Figure 36. Chaque *Var* et *Repeat* ont un identificateur (*id*) qui permet de différentier chaque instance dans le template.

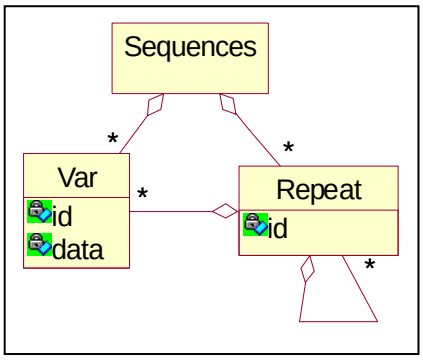


Figure 36. Structure de données pour remplir les templates

3.3 Génération de code

Le processus de développement d'un module pour la génération de code est normalement un processus itératif de plusieurs étapes. Ce processus va en parallèle avec le développement principal de l'application quand la plate-forme cible est nouvelle (i.e., il n'y a pas un module de transformation pour cette plate-forme dans l'IDK). Le processus de développement principal, en conséquence, compte sur la disponibilité du processus de construction du module pour élaborer, d'une façon appropriée dans le temps, un processus de génération de code qui permet la transformation de modèles en code exécutable.

Dans notre cas, la base pour la génération de code est la disponibilité de templates du code pour la plate-forme cible. Normalement, celle-ci est la plus difficile à obtenir, puisque cela demande une bonne connaissance sur comment implémenter des agents sur la plate-forme cible. Notre expérience montre que cela peut s'accomplir à travers un processus itératif, dans lequel l'ingénieur définit progressivement l'architecture du code pour la plate-forme cible et les transformations des modèles aux templates du code. Ce processus se compose de plusieurs pas :

1. Prototype initial. L'ingénieur produit un prototype simple de l'application. Au départ, l'ingénieur tient en compte d'une ou de plusieurs caractéristiques de la spécification, faciles à implémenter si possible. Par exemple, comment utiliser les facilités spécifiques de la plate-forme cible pour faire interagir deux agents. Comme résultat, l'ingénieur obtiendra la connaissance sur la plate-forme cible et aura un prototype d'une application sur la plate-forme cible. Ce prototype implémente déjà une partie de la spécification avec un ensemble de caractéristiques sélectionnées.

2. Marquage du code du prototype. En regardant en même temps le prototype et la spécification, il est possible d'identifier les parties du prototype qui correspondent aux parties de la spécification. Comme résultat, l'ingénieur peut identifier des correspondances possibles entre la spécification et le code du prototype. Celles-ci seront reflétées dans le code du prototype marqué avec des étiquettes XML. Les *templates* sont les morceaux marqués du code source.

3. Génération et modification du module. Un module doit parcourir la spécification pour obtenir l'information nécessaire pour instancier et remplir les templates du prototype. L'IDK offre un API pour parcourir les spécifications (voir section 3.2.1) et les librairies de paquets Java pour construire les modules. Concrètement, pour implémenter un module de vérification ou de validation il faut créer une classe qui hérite de la classe *BasicToolImp*, et pour la génération de code il faut hériter de la classe *BasicCodeGeneratorImp*. Plus d'information sur ces classes et l'API se trouvent dans le manuel de l'IDK [82].

4. Déploiement du module. Les classes et les templates du module résultants doivent être placés ensemble dans un fichier *jar*. Ce fichier *jar* est déployé dans un dossier spécifique où l'IDK peut le charger dynamiquement.

5. Test du module. Le test peut se réaliser depuis l'éditeur de l'IDK. Une fois chargé dans l'IDK, le module peut s'exécuter sur la spécification qui est chargée dans l'éditeur. L'ingénieur doit examiner si le diagramme est parcouru correctement et si les templates ont été bien remplis. Comme les templates demandent une information spécifique, il

peut arriver que cette information ne soit pas présente ou bien qu'elle ne soit pas formulée de la façon appropriée. Si la spécification n'est pas correcte ou incomplète, un module peut être utile aussi pour vérifier sa complétude d'après certains critères. Trois types de problèmes peuvent arriver : avec le code généré par le module, avec le parcours de la spécification, et avec la spécification elle-même.

6. Déboguer. Si quelque chose ne marche pas bien, il faut déboguer le prototype et aller au :
    a. Pas 2. Si il y a un nouveau code qui n'a pas été marqué avant.
    b. Pas 3. Si la faille était dans le module et le parcours des données.
    c. Pas 4. Si la faille était dans le prototype et peut se résoudre sans marquer le code à nouveau.
7. Raffinement et extension. Une fois le module fini, il peut transformer les diagrammes des spécifications en code source ou vérifier la satisfaction de certaines propriétés. Mais le module peut considérer uniquement un ensemble réduit de diagrammes. Le prochain pas est de prendre le code généré par le module et l'étendre pour satisfaire d'autres parties de la spécification. Cela signifie retourner au premier pas.

De cette façon, les modules produisent le code en utilisant des templates. Comme exemple, la Figure 37 montre comment générer le code pour un système de règles, JESS (Java Expert System Shell, http://herzberg.ca.sandia.gov/jess). L'ingénieur peut définir un template de la règle JESS (pas 1 et 2) et extraire les données de la spécification du SMA (pas 3) pour générer le reste des règles. Les règles ont besoin d'une condition, d'une action, et d'un nom. Ces données sont représentées par une structure concrète qui est définie dans l'API de l'IDK. Le résultat pour cet exemple consiste en deux règles différentes, chacune instantiée à partir du même template.

Ces éléments sont configurés dans un module et déployés dans l'IDK (pas 4). Le test du module avec la spécification du SMA (pas 5) doit bien produire le code espéré comme dans le cas de la Figure 37.

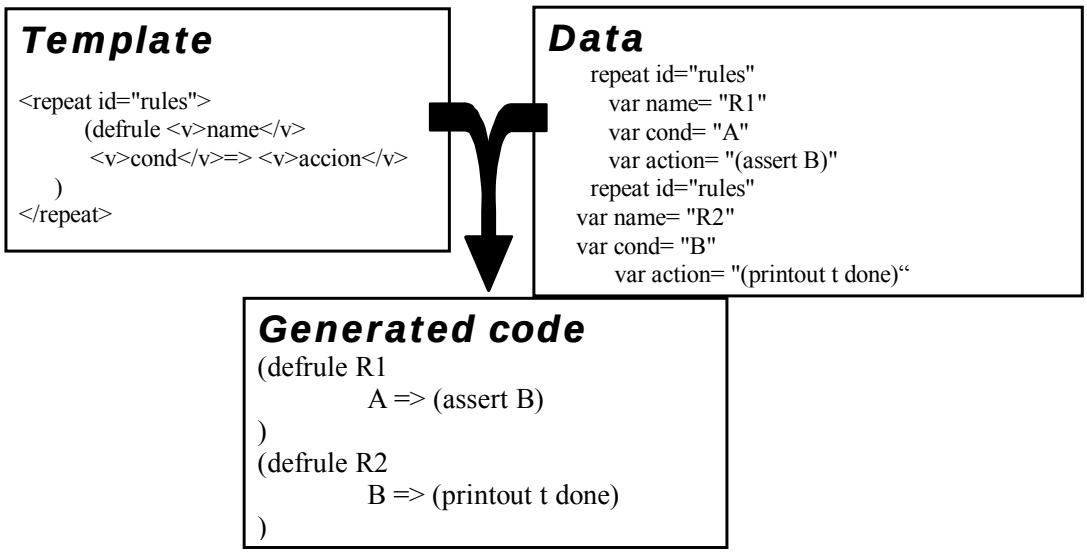


Figure 37. Exemple de génération de code pour un ensemble de règles JESS

Même en utilisant d'autres techniques de transformations, par exemple basées sur XML, la génération du code source doit considérer des éléments et procédures similaires. Il faut toujours déterminer quel code source produire, quelle information du modèle est appropriée, et suivre un processus de test exhaustif pour assurer que la chaîne de transformation travaille comme prévu. Dans tous les cas, la production du code source à partir de modèles demande du temps et de l'expertise de la sémantique des modèles et de la plate-forme cible. D'ailleurs, un module pour traiter la spécification complète est rarement construit en une seule itération. Il demande une analyse progressive des besoins détaillés des modèles. L'intégration du développement des templates et des transformations dans un contexte de développement dirigé par modèles de SMA se discute dans le chapitre 4.

3.4 Conclusion

Actuellement, l'IDK est complètement implémenté en Java et distribué sous licence GNU General Public Licence (GPL) et peut être téléchargé depuis *http://ingenias.sourceforge.net*. Il est distribué avec un manuel qui donne l'information détaillée sur l'utilisation de l'éditeur, pour créer et modifier les diagrammes, et comment développer et utiliser les modules. Sur le même site il y a toujours la dernière version supportée de méta-modèles INGENIAS.

La distribution de l'IDK est accompagnée d'un ensemble de modules qui ont été construits comme le montre en la section précédant. Par exemple :

- Génération de code pour la plate-forme JADE (http://jade.tilab.com). Il y a trois modules qui ont été développés progressivement, mais par différentes personnes. Le

premier traduit les diagrammes d'interaction INGENIAS en implémentations des machines d'état pour chaque participant. Chaque partie tient en compte la connaissance partiale de chaque participant en une conversation telle qu'elle est modélisé avec l'IDK. Le deuxième module est un générateur de code pour JADE Leap qui ajoute à l'antérieur la capacité de mobilité de code et l'exécution de tâches. La mobilité et l'exécution de tâches sont aussi telles que spécifiées avec l'IDK. Plus évolué est le module JADE Organization, décrit en la section 2.5.2. Ce module améliore la représentation computationnelle de tâches (préconditions, postconditions, contrôle de ressources, et intégration en workflows) et l'implémentation de protocoles d'interaction. Avec ce module les SMA résultantes sont presque fonctionnels. Il est laissé au développeur de définir l'implémentation souhaitable pour la gestion et le processeur d'état mental des agents. Il y a aussi une interface graphique qui permet au développeur d'animer manuellement l'évolution de SMA en prenant les décisions sur quelles tâches exécuter et gérer les entités mentales.

- Simulateur de workflows basé sur servlets. Ce module génère des servlets qui permettent de simuler le comportement des agents en un workflow. Il est utile pour étudier le comportement dynamique d'une organisation d'agents en exécutant des workflows. Le simulateur peut détecter, en parcourant la spécification de SMA, les tâches initiales et la progression de workflow, en regardant après l'exécution d'une tâche concrète lesquelles sont celles permises.
- Génération de documentation. Pour faciliter le développement, il est important de pouvoir documenter la spécification de SMA. Ce module utilise des templates de rapports pour produire plusieurs documents HTML qui facilitent la lecture de la spécification.

Finalement, il y a un aspect d'INGENIAS qui n'est pas encore developpé. Il s'agit d'inclure dans l'IDK des outils pour la gestion du processus de développement en équipe. Ici on prétend profiter de l'expérience des travaux de génie logiciel et rechercher en quoi se différencie un développement de SMA quand interviennent de multiples usagers d'un processus conventionnel. Pour le moment, on prévoit deux lignes de travail :

- Un système de contrôle de versions pour les spécifications générées par l'IDK. Actuellement, une spécification se garde comme un fichier XML et il n'est pas possible que plusieurs personnes puissent y travailler en même temps. Il reste à étudier comment on peut atteindre un fonctionnement similaire à celui d'autres outils commerciaux. Ceci passe par établir des modes de travail collaboratif.

- Un contexte pour gérer les tâches. Il existe déjà des contextes capables d'implanter des cycles de développement de tâches, comme *phpCollab* (www.php-collab.org) ou *dotProject* (www.dotproject.net). On dispose aussi d'une liste d'activités structurées selon ce dont dispose le Processus Unifié. Cependant, on a besoin de cas complets qui aident à valider ce processus de développement, des cas dans lesquels participent de multiples développeurs.

4. INGENIAS-MDD

La méthodologie INGENIAS adopte, en suivant MESSAGE [31], un cycle de développement standard : le Processus Unifié (*Unified Process*) [104]. Les activités de développement sont organisées en un cycle de vie comme le Processus Unifié. Ce qu'INGENIAS rajoute, c'est une définition des activités qui peuvent être développées dans les étapes d'analyse et de conception, tout spécialement, pour identifier, définir et associer les éléments qui construisent le SMA. De cette manière, il s'agit de réaliser des instanciations des méta-modèles pour construire les modèles qui définissent un SMA concret. Quant à l'analyse, la base est l'utilisation de la modélisation de rôles et des cas d'usage, assez développée dans plusieurs travaux. Quant à la conception et à l'implémentation INGENIAS introduit les notions de composants d'agents et propose la correspondance des modèles avec des schémas préalablement développés pour des plates-formes d'agents. Un besoin principal de cette première version d'INGENIAS est de permettre l'évolution de la méthodologie. C'est ainsi que l'extensibilité des éléments utilisés pour la modélisation est nécessaire. L'utilisation de méta-modèles procure cette flexibilité et des nouveaux éléments de modélisation ont été introduits.

Le développement des outils IDK, décrits dans le chapitre 3, a établi des éléments pour définir un processus plus agile pour l'élaboration de SMA. Ce processus donne encore plus de pertinence aux modèles du SMA et aux procédures de transformation des spécifications vers l'implémentation. La définition des transformations devient aussi un élément central du processus de développement. Dans cette approche, il faut considérer l'évolution de la spécification de l'application (i.e., des modèles du SMA) en parallèle avec la construction de modules de génération de code pour les plates-formes de l'implémentation finale. Tout passe dans un processus itératif où on ajoute et raffine des nouvelles caractéristiques du système progressivement. Ce sont les principes pour un processus de développement dirigé par modèles des SMA : INGENIAS-MDD.

4.1 Développement Dirigé par Modèles pour les SMA

Le développement dirigé par modèles (en anglais, *Model Driven Development*, MDD) est en train de gagner l'intérêt de l'industrie du logiciel, avec l'adoption par l'OMG des standards *Model Driven Architecture* (MDA) [137] et la disponibilité des outils CASE basés sur ses principes. MDD en essence promotionne l'idée de travailler avec les modèles des systèmes, et que les implémentations peuvent en être dérivées par transformations.

L'utilisation des modèles pour concevoir des systèmes complexes est courante dans la plupart des disciplines. Mais en génie logiciel les modèles ont joué un rôle secondaire dans le processus de développement. Traditionnellement, les modèles ont été considérés uniquement comme documentation, comme un ensemble de diagrammes pour décrire une architecture de haut niveau du système. De plus, ces diagrammes n'évoluent pas avec l'implémentation du système et deviennent obsolètes. Récemment, avec UML, cette situation commence à changer et les langages de modélisation commencent à être pris en compte plus sérieusement pour diriger le processus de développement de logiciel. En particulier, MDD se focalise sur les modèles, plutôt que sur le code, comme le produit principal du processus de développement et conçoit le processus de développement comme un processus de transformation de modèles abstraits, indépendants de la plate-forme, vers le code [167].

La disponibilité de langages de modélisation qui ne sont pas liés aux langages d'implémentation permet une représentation plus proche du domaine de problème. Cela facilite la communication entre les différents acteurs du cycle de développement, des clients aux développeurs de logiciel. Celle ci permet d'encourager la participation dans le développement des experts du domaine. Un autre avantage c'est que les modèles à ce niveau d'abstraction sont moins sensibles aux changements dans la plate-forme de computation sous-jacente.

Ce qui caractérise MDD c'est que les modèles ne sont pas uniquement de la documentation, mais des artefacts à partir desquels les programmes sont automatiquement générés. Ceci est possible avec la disponibilité des techniques de génération de programmes complètes à partir des modèles. Concrètement, trois types d'outils sont nécessaires pour réaliser MDD :

- Les éditeurs de modèles, qui aident le développeur à créer et modifier les modèles avec les concepts et les notations du langage de modélisation, en forçant aussi les règles qui gouvernent leur utilisation.
- Les outils de validation et de vérification, pour prouver que le modèle satisfait des propriétés spécifiques (vérification) et qu'il correspond avec la réalité espérée

(validation). Celle ci peut se réaliser avec une analyse formelle et l'exécution ou la simulation des spécifications des modèles.

- Les outils de transformation, pour générer le code ou les modèles d'un niveau d'abstraction mineur. Même si MDA ne prescrit pas la nécessité de la génération de code automatique, quelques auteurs, tels que [167] et [178], considèrent qu'il est nécessaire parce que sinon les développeurs peuvent se poser des questions sur les bénéfices de la modélisation.

Tous ces outils sont basés sur la manipulation des langages, des langages de modélisation abstraite aux langages de programmation. Ils demandent des formalismes pour la définition des langages à plusieurs niveaux. Avec ce propos, MDD utilise les langages de méta-modélisation, comme MOF [136].

Les méta-modèles ont été utilisés dans le génie logiciel orienté agents pour présenter des concepts et récemment comme base pour les outils de développement de SMA. Probablement, AALAADIN [55] a été le premier méta-modèle pour SMA. Il prétend représenter la structure de SMA, pas le comportement, en termes de trois concepts principaux : agent, groupe et rôle. Ces concepts ont été implémentés avec la plate-forme MadKit. Une proposition plus formelle pour la définition d'organisations d'agents a été proposée récemment par FIPA [59] et publiée en [133]. Celle ci étend la superstructure UML avec MOF, mais il n'y a pas d'outil pour l'implémenter. Aussi, il y a une étude pour utiliser la nouvelle version d'UML, 2.0, pour spécifier des SMA [7]. Ce qui est aussi intéressant dans cette proposition c'est la discussion sur les aspects correspondant aux modèles définis en MDA (CIM, PIM et PSM). Ces méta-modèles sont assez simples, mais ils représentent un travail de base dans le domaine de recherche d'agents.

Récemment, l'utilisation des méta-modèles s'étend à la plupart des méthodologies. Ils ont été proposés comme outil pour aider à comparer des méthodologies, comme en [10] pour Adelfe, Gaia et Passi. Il y a également des efforts pour la définition d'un méta-modèle générique de SMA, par exemple en AgentLink III, comme cela est reporté en [11], ou comme base pour appliquer une *ingénierie des méthodes* [14].

La transformation des modèles consiste à prendre un modèle avec une information additionnelle et appliquer quelques règles de transformation pour dériver un autre modèle. Par exemple, le guide MDA d'OMG [137] décrit un patron de transformation pour transformer un modèle indépendant de la plate-forme (PIM, *Platform Independent Model*) en un modèle spécifique d'une plate-forme (PSM, *Platform Specific Model*), comme le montre la Figure 38.

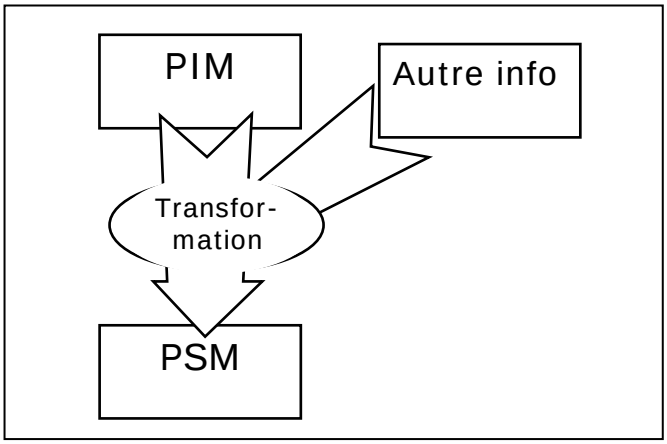


Figure 38. Le patron de transformation de modèle en MDA

Les transformations peuvent être essentiellement de deux types [176] :

- Traduction des langages : quand un modèle est transformé en un modèle d'un autre langage. Le modèle peut se transformer en un autre modèle au même niveau d'abstraction (migration de la spécification d'un langage à autre), à un niveau inférieur (synthèse, génération de code) ou à un niveau supérieur (ingénierie inverse).
- Redéfinition : quand un modèle est transformé en un autre modèle mais dans le même langage. Par exemple, la refactorisation (restructuration d'un modèle pour l'améliorer), l'adaptation (pour considérer de nouveaux besoins ou changements) ou la correction (pour l'élimination des erreurs d'un modèle).

Pour les transformations, OMG est en train d'adopter un langage, *Query/View/Transformation* (QVT) [138], qui devrait permettre de faire des requêtes (*queries*) sur les modèles MOF, de créer différentes perspectives d'un modèle, et de définir des transformations. Des outils existent déjà dans ce contexte, par exemple, ATL [15].

L'utilisation de transformations pour les modèles de SMA a été appliquée dans le cas de Tropos. Par exemple pour la redéfinition des modèles Tropos, en utilisant les techniques de transformation de graphes [130] ou la proposition pour transformer d'un modèle d'architecture de conception en Tropos en un modèle de conception détaillé spécifié en UML [151]. Celle-ci tient en compte le méta-modèle Tropos (voire Figure 39) et le méta-modèle d'UML 2.0. Pour l'instant, un prototype a été appliqué pour transformer un diagramme de décomposition de plan en Tropos à un diagramme d'activité UML. L'édition de modèles se réalise avec un plugin spécifique d'Eclipse qui utilise EMF et GEF [172]. Pour l'implémentation, une correspondance préliminaire a été proposée, avec un cas d'étude, des idées ont été proposés pour implémenter des agents BDI sur la plate-forme Jack [29].

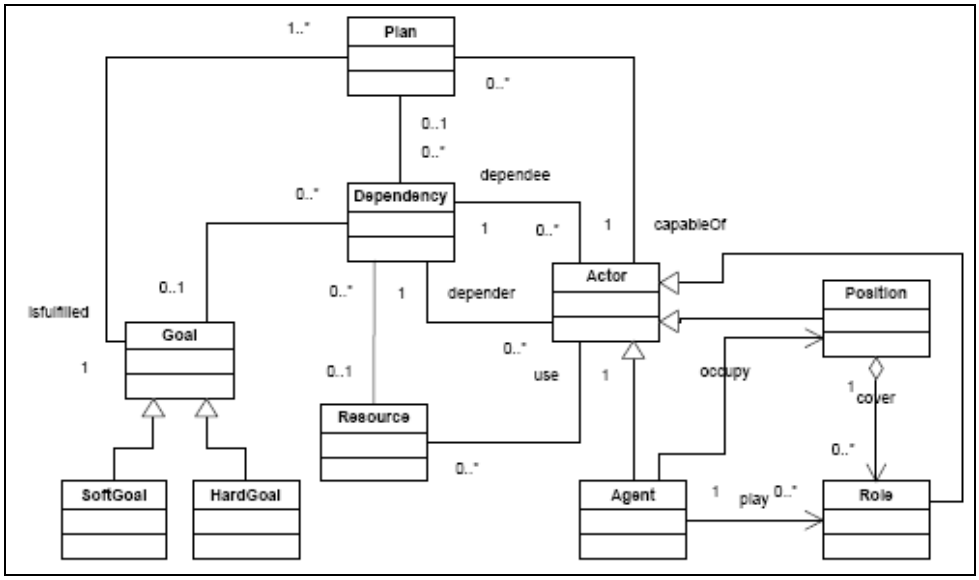


Figure 39. Le méta-modèle Tropos

Une proposition plus prés des propositions MDA pour le développement des SMA est MétaDIMA [174]. Cette approche considère les méta-modèles comme les briques de base nécessaires à la construction d'un SMA opérationnel par une série de transformations automatiques ou semi-automatiques. Un des avantages de l'approche MétaDIMA c'est qu'avec la disponibilité actuelle des nouveaux outils de méta-modélisation et des transformations QVT elle pourra être mise en pratique et évoluer avec les standards.

PASSI [36] propose un processus itératif et incrémental pour passer des besoins au code, à travers cinq modèles, montrés dans la Figure 40 : modèle de besoins de système, modèle de société d'agents, modèle d'agents (diagrammes de classes pour définir la structure des agents et diagrammes d'activité pour décrire leur comportement), modèle de code, et modèle de déploiement. La notation pour décrire tous les modèles est UML mais avec de nouveaux éléments des concepts d'agent qui sont spécifiés dans un méta-modèle (Figure 40) structuré en trois niveaux d'abstraction différents. Même si ce n'est pas défini comme cela en PASSI, on pourrait considérer que le domaine de solution est associé au CIM, le domaine d'agentification pourrait ressembler au PIM et le domaine de problème apporter les éléments pour le PSM. Il manque, néanmoins, une définition de transformations.

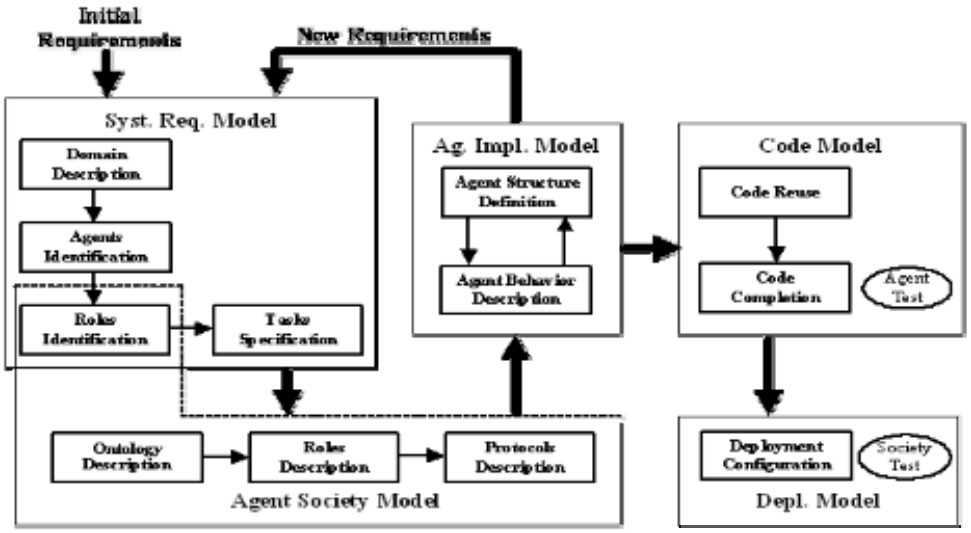


Figure 40. Les modèles de la méthodologie PASSI (d'après [36])

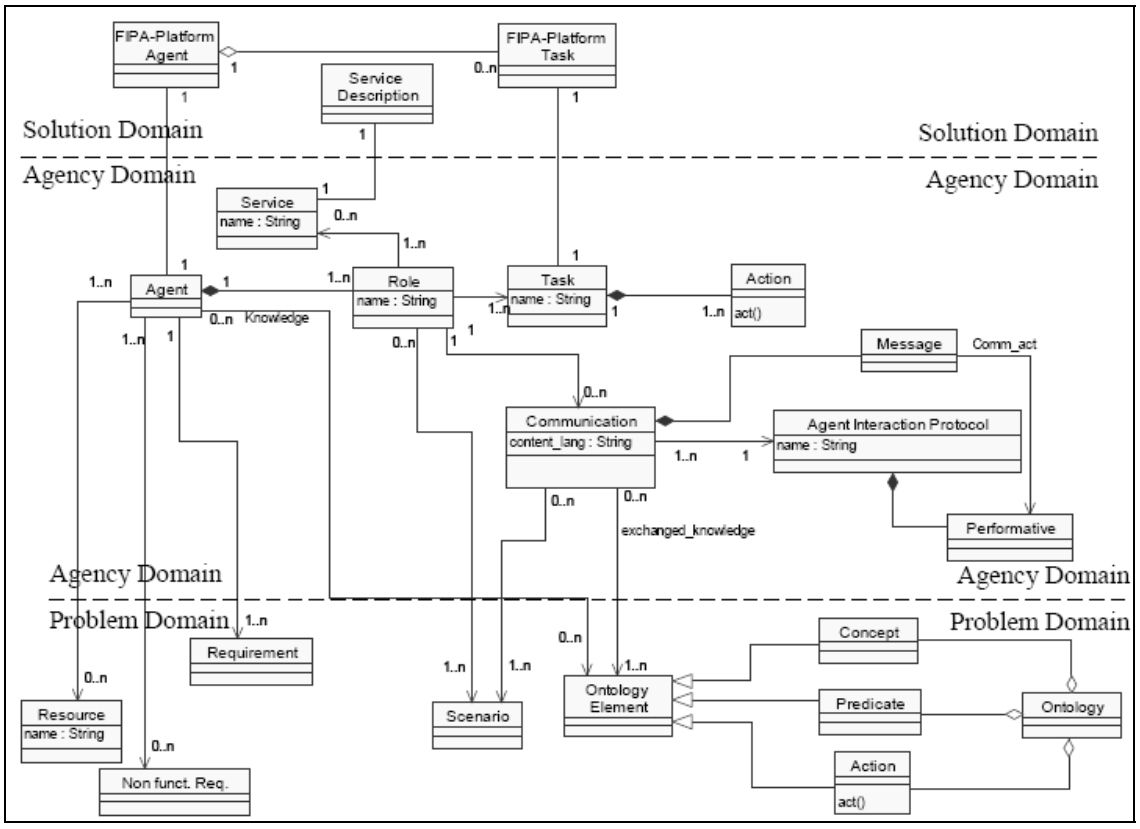


Figure 41. Méta-modèle de SMA en PASSI

Plus récemment, [5] essaie d'intégrer dans un cadre formel différents fragments de méthodologies de SMA. En considérant les différents niveaux d'abstraction impliqués, il utilise un langage de description d'architecture [125], ArchWare [140], qui permet de travailler aux niveau de méta méta-modèle pour définir des transformations entre méta-modèles.

En INGENIAS, les outils pour l'approche MDD couvrent les différents aspects mentionnés auparavant :

- Pour l'édition des modèles, un éditeur généré à partir des spécifications de méta-modèles. Il est généré automatiquement pour s'adapter aux changements dans ces spécifications, pour étendre le méta-modèle INGENIAS ou bien pour le personnaliser pour un domaine d'application particulier (voir section 3.1).
- Pour les transformations, on dispose des facilités pour parcourir les modèles (*query*), extraire l'information spécifique des modèles (*view*) et générer de nouveaux modèles (*transformation*). La différence ici, par rapport au standard MDA est que ces facilités sont offertes comme interface de programmation et non comme un langage déclaratif, tel que QVT. Celle ci est due principalement à l'approche basée sur les templates pour la génération de code, que nous considérons plus accessible aux développeurs. La construction d'outils pour la validation et la vérification est un cas particulier de transformations, puisque elle a besoin des mêmes facilités que pour la génération de code, sauf qu'elle peut se réaliser sans templates.

Dans les sections suivantes on discute comment ces outils affectent le processus de développement INGENIAS.

4.2 Rôles dans un développement dirigé par modèles en INGENIAS

Pour résoudre les dépendances entre l'élaboration des modèles et la construction des transformations, on a raffiné le processus INGENIAS initial en tenant compte des principes MDD. D'abord, on considère deux types principaux de rôles dans le processus de développement, qui s'occupent, respectivement, de chaque aspect :

- Le *développeur des SMA*, illustré dans la Figure 42, utilise l'éditeur de l'IDK pour spécifier les modèles qui décrivent le SMA. Ces modèles peuvent être vérifiés et validés avec des modules d'analyse et de simulation. Une fois qu'ils ont été validés, les modules de génération de code facilitent l'implémentation pour la plate-forme cible. Le résultat est un système exécutable qui peut être testé. Si le développeur des SMA détecte des failles, il faut réviser les modèles. Aussi, le développeur des SMA peut revenir sur les modèles pour ajouter de nouvelles caractéristiques.

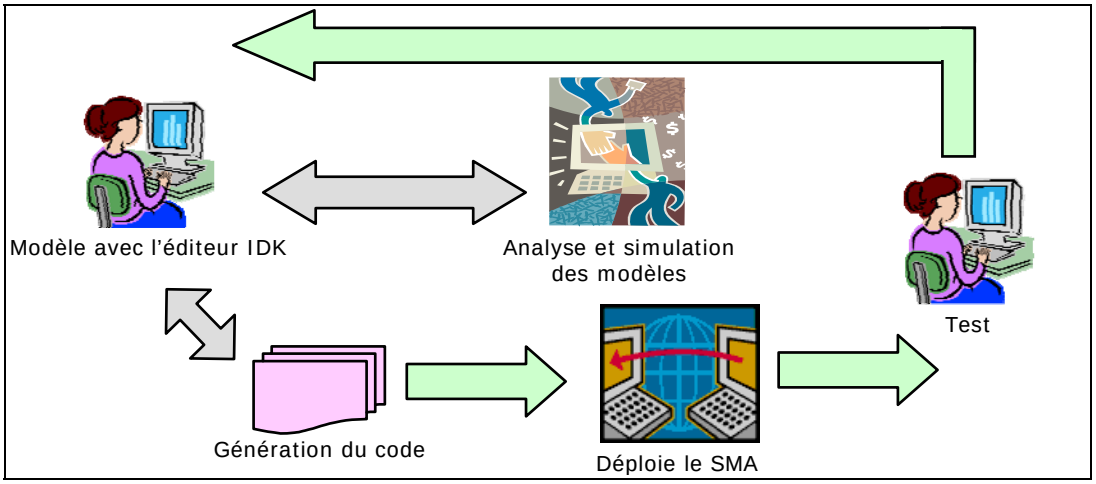


Figure 42. Activités du développeur des SMA

- L'*ingénieur INGENIAS*, illustré en la Figure 43, connaît le méta-modèle INGENIAS et peut travailler sur les outils de l'IDK. Essentiellement, il s'agit de produire des nouveaux modules pour la vérification des propriétés dans les modèles et leur validation ou pour la génération de code en une plate-forme cible. Il est possible également de personnaliser l'éditeur pour un propos spécifique, par exemple, pour l'adapter à un langage de modélisation spécifique du domaine d'application. Ce dernier exige de modifier le méta-modèle et demande une connaissance plus approfondie du méta-modèle INGENIAS.

La plupart des processus de développement identifient davantage des rôles, mais ici notre but est de résoudre les dépendances entre les activités d'élaboration des modèles et la construction des transformations des modèles en code source. Il peut arriver que la même personne joue les deux rôles, ce qui est normal quand l'application à développer doit être déployée dans une nouvelle plate-forme cible, et il est donc nécessaire de construire un nouveau module de génération de code. Mais, en principe, seule une personne de l'équipe de développement doit avoir les compétences pour jouer le rôle d'ingénieur INGENIAS et créer de nouveaux modules. Le reste de l'équipe peut tout simplement faire l'édition des modèles, exécuter les modules de l'IDK pour travailler avec les spécifications et finir la programmation des détails. Dans ce cas, l'ingénieur INGENIAS et le développeur des SMA doivent synchroniser leurs activités.

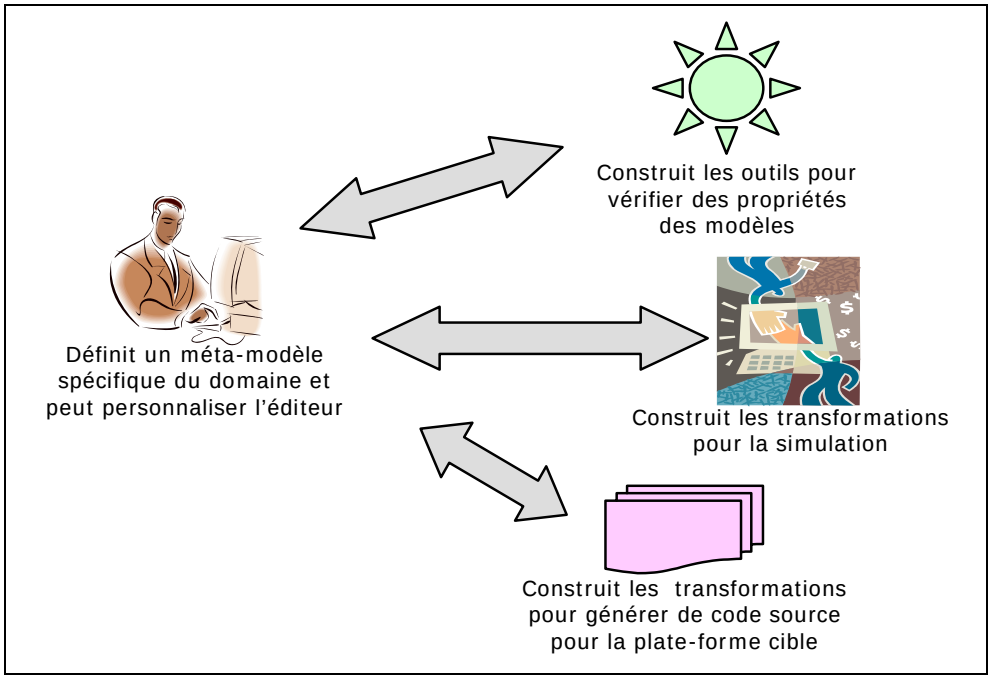


Figure 43. Activités de l'ingénieur INGENIAS

D'autres rôles peuvent s'impliquer dans le développement. Notamment, le *programmateur des SMA*, illustré en la Figure 44, travaille avec le code généré et des libraires additionnelles qui demandent l'application et l'environnement. L'intégration de ces composants avec la spécification de SMA peut se réaliser comme le montre la section 2.6.2.

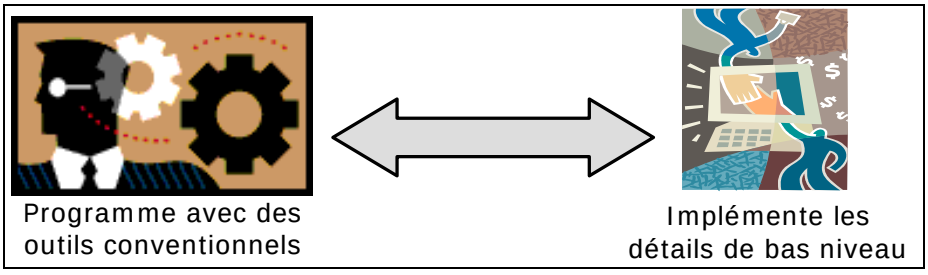


Figure 44. Activités du programmateur des SMA

4.3 Un processus de développement dirigé par les caractéristiques des modèles

En tenant compte de la procédure pour créer de modules de génération de code, présentée dans la section 3.3, il y a plusieurs aspects qu'il faut considérer pour adapter ce flux d'activités au flux de développement principal :

- La spécification de l'application SMA peut s'étendre dans plusieurs directions, et certaines peuvent être incompatibles avec les hypothèses faites pendant la définition de la transformation.
- La procédure de transformation détermine les modèles, et vice-versa. D'une part, pour obtenir le code source il faut assumer une sémantique des modèles non ambiguë. D'autre part, seuls les modèles qui sont valides d'après cette sémantique seront considérés. Mais, est-ce que toute l'information pertinente est considérée ? Si l'on tient compte que le développement est incrémental, la construction d'une transformation implique que chaque pas peut être incompatible avec le précèdent. Une transformation peut considérer de nouveaux éléments de la spécification qui pourraient exiger l'identification de nouvelles dépendances qui n'existaient pas avant.

Pour aborder ces problèmes, la version MDD d'INGENIAS synchronise les activités liées au développement de la spécification à celles liées aux transformations. Ainsi, plutôt que de construire indépendamment les modèles du SMA et les transformations des modèles, les deux flux évoluent en parallèle avec des contre-vérifications fréquentes pour assurer que la transformation actuelle peut s'étendre pour incorporer de nouveaux éléments de la spécification et pour éviter l'addition d'information qui ne pourrait pas s'implémenter dans la livraison finale.

Pour le développement d'une nouvelle application avec INGENIAS MDD il faut spécifier le SMA avec l'éditeur et disposer des transformations pour la plate-forme d'implémentation. Si les transformations existent déjà parce qu'elles ont été développées auparavant pour un autre application sur la même plate-forme et qu'elles sont disponibles avec l'IDK, le processus est simple : le développeur des SMA spécifie le SMA, utilise des modules de validation et de vérification pour assurer les propriétés envisagées pour la spécification, et génère le code automatiquement. Néanmoins, il peut arriver qu'il faille adapter les transformations au nouveau domaine d'application ou pour une nouvelle plate-forme de déploiement. Dans ce cas, l'adaptation ou la création des modules demandent la planification de l'effort requis avec la modélisation et le test.

Les modules de génération de code sont développés de façon itérative et en parallèle à la spécification du SMA. Il y a alors deux aspects à découvrir pendant le processus de développement : les besoins de l'application et la programmation sur la plate-forme cible. Cette deuxième est normalement négligée dans la plupart des méthodologies, mais elle est très importante pour bien définir les transformations. Alors, le processus de développement doit considérer que, dans la pratique, l'ingénieur va apprendre à travailler sur la plate-forme cible, et cela arrive dans un processus itératif. Ce processus est en connexion avec la définition de la

spécification du SMA. Notre expérience nous a montré que les points de connexion entre les deux sont les caractéristiques des modèles qui doivent être implémentés. Par *caractéristiques* on désigne des parties de la spécification du SMA qui ont une signification par elles mêmes, par exemple, une interaction entre deux agents. Cette interaction peut être spécifiée tout simplement par un diagramme d'interaction, qui contient les agents ou les rôles qui y participent, la déclaration des unités d'interaction (telle que messages) échangées, et les tâches associées. Cette interaction peut être implémentée sur la plate-forme cible. Pour ça, il faut programmer deux agents qui utilisent les services de la plate-forme pour envoyer et recevoir des messages, et pour exécuter les tâches. Cette programmation facilite la connaissance de la plate-forme et sert à établir une première structure de code qui pourrait se réutiliser après. En effet, quand on essaie d'implémenter une deuxième interaction, il est possible de généraliser et tester la structure première. Cela peut se décrire alors comme une template de code et une transformation. Cette transformation identifie les éléments du modèle nécessaires pour instancier le template et le remplir avec les données appropriées. Voici que le processus de développement est déjà commencé. En suite, on prend une autre caractéristique du modèle, par exemple, la description de tâches, et on développe le template et la transformation pour les implémenter.

D'une façon idéale, on pourrait arriver ainsi à automatiser le développement et il serait simplement nécessaire de changer les modèles pour modifier l'implémentation automatiquement avec les modules IDK. Néanmoins, l'évolution des templates à travers les différentes itérations est un processus complexe. Il faut considérer la redéfinition des templates, des transformations, l'intégration des composants logiciels, et l'évolution même des spécifications. Tout cela implique les trois acteurs identifiés dans la section précédente : l'ingénieur INGENIAS, le développeur des SMA et le programmeur des SMA. Comme la Figure 45 le montre, ce processus demande aussi le support des nouveaux outils. Le module IDK, développé par l'ingénieur INGENIAS, permet de générer une implémentation à partir de la spécification de SMA, en utilisant les templates et les transformations. Pour faciliter l'intégration de composants logiciels existants, COTS (*Commercial Off The Shell*), ou de code programmé spécifiquement pour l'application, l'AppLinker permet d'associer dans la spécification de SMA ces composants et maintenir les changements dans ces composants. Plus sophistiqué est l'éditeur de templates (actuellement en phase alfa) qui permet de faire des modifications dans le code généré et actualiser de façon automatique les templates et la spécification. Il facilite notamment l'intégration de tous les éléments du processus INGENIAS-MDD : la spécification de SMA, l'implémentation de SMA, les templates et les transformations.

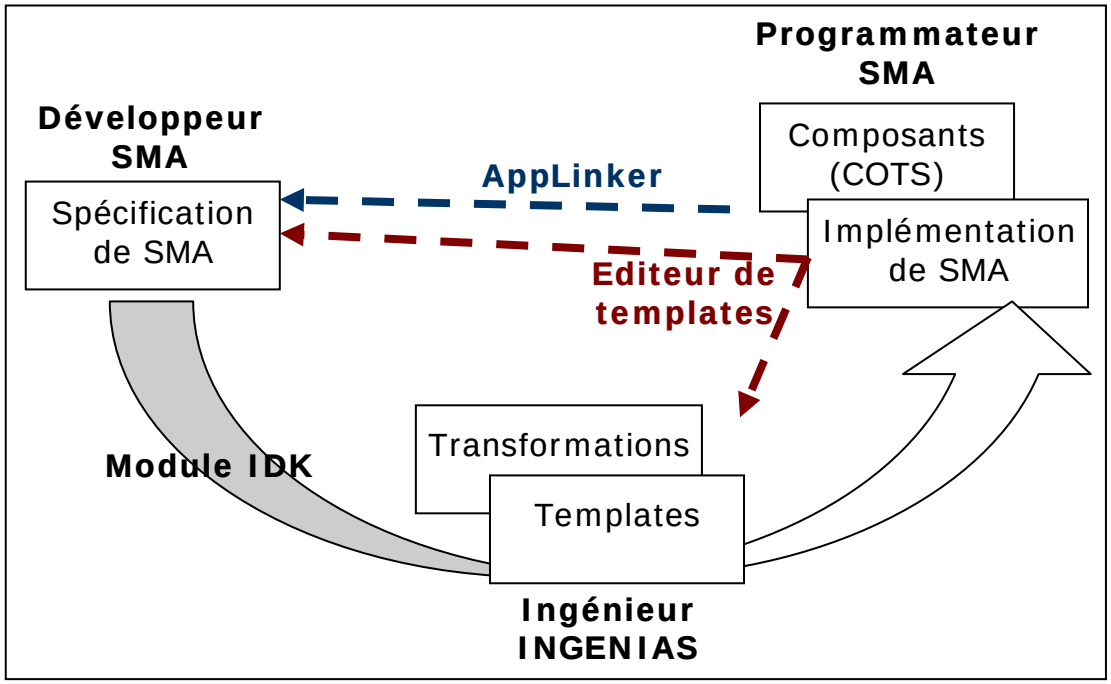


Figure 45. Outils et rôles pour le processus INGENIAS-MDD

Avec ces outils, il est possible de suivre un processus itératif et incrémental dirigé par les besoins ou caractéristiques des modèles. Ce processus est composé des pas suivants, une partie est montrée dans la Figure 46 :

1. Identification des besoins. Les besoins sont représentés par des cas d'utilisation, et pour chaque cas d'utilisation plusieurs scénarios sont possibles.

2. Spécification d'un modèle de SMA avec INGENIAS. Il y a plusieurs façons d'aborder la spécification de SMA à partir des cas d'utilisation. La plus directe est de considérer chaque cas d'utilisation comme un but du système et le décomposer en sous-buts et finalement tâches. Il est possible aussi de considérer les scénarios de cas d'utilisation comme des descriptions de workflows, ou bien d'identifier les acteurs (rôles) dans le diagramme de cas d'utilisation et les agents à partir de cela. En tout cas, le but est d'arriver à élaborer une spécification qui considère certaines caractéristiques du système.

3. Cette caractéristique peut s'implémenter dans la plate-forme cible. S'il n'y a pas un générateur de code approprié, la première fois le code serait implémenté manuellement.

4. Ce code peut être généralisé dans un template, en déterminant les éléments qui peuvent changer dans la spécification.

5. Révision et extension du parcours et extraction d'information dans le modèle. Comme il peut avoir de nouveaux éléments de la spécification à tenir en compte pour remplir les templates, il faut reconsidérer les algorithmes de parcours et d'extraction d'information dans le modèle.
6. Re-génération du module IDK. Avec ces changements il faut recompiler les composants du module.
7. Validation et test du module. Il faut vérifier que le module récupère l'information nécessaire pour instancier les templates et que le code résultant est celui que l'on espère.
8. Refactoriser en utilisant des patrons de conception.
9. Validation et test du module refactorisé, pour assurer que les changements ont été faits correctement.
10. Déploiement du module dans l'IDK.
11. Retourner au pas 1 pour reconsidérer de nouveaux besoins, cas d'utilisation ou scénarios, ou bien au pas 3 pour traiter la transformation d'autres parties du modèle.

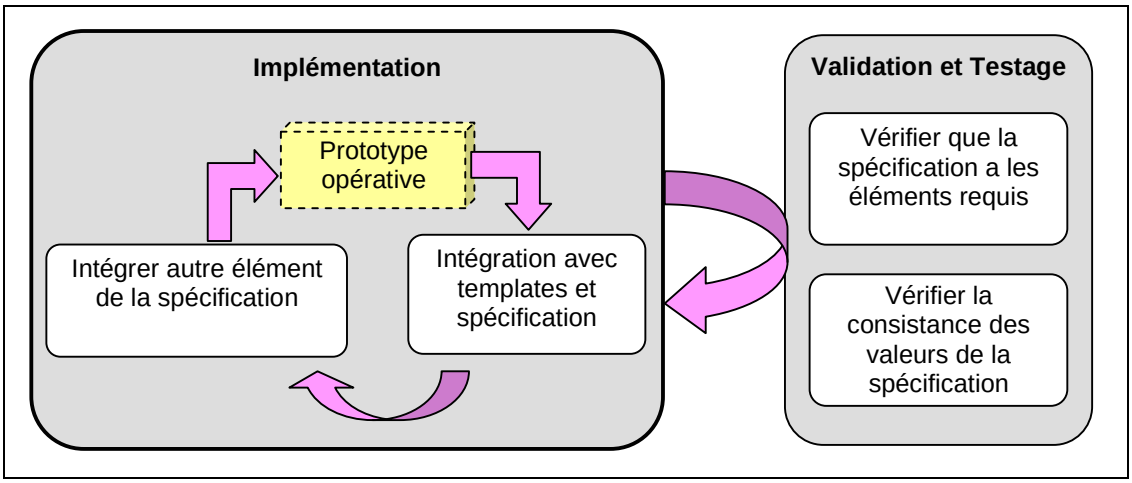


Figure 46. Processus itératif et incrémental pour développer le module de génération de code

4.4 Conclusion

L'IDK facilite les éléments nécessaires pour MDD :

- Un éditeur des modèles. L'éditeur permet de construire modèles de SMA qui sont conformes aux méta-modèles. La notation INGENIAS est indépendante de la plate-forme cible. En plus, elle peut se personnaliser pour des domaines d'applications concrets, comme cela a été montré pour les systèmes holoniques [75] ou pour la simulation sociale [164].
- Des modules pour la transformation des modèles SMA en modèles d'implémentation. Les modules permettent de réutiliser l'expérience et les patrons d'implémentation. Les patrons d'implémentation d'agents sont adaptés à la plate-forme finale avec les templates. L'IDK offre aussi des interfaces (section 3.2) pour aider à la définition des transformations des modèles (déterminés par les algorithmes de parcours à travers les modèles).
- Les modules permettent aussi de vérifier des propriétés et valider les modèles.

L'approche INGENIAS pour la génération du code est en ligne avec Wasserman [177], qui propose un environnement de développement rapide, ou les spécifications graphiques sont traduites en code source avec templates de code. Cette technique a évolué vers des langages de script plus sophistiqués qui permettent de faire la correspondance entre la spécification et le code. Des exemples de cette approche sont METAEDIT+ [109] et WithCLASS [126]. Le premier est un outil méta-case qui permet de construire une correspondance entre les spécifications et n'importe quel type de code. Le second est un outil UML qui permet de traduire les diagrammes en code source, avec un langage de script. La différence de l'IDK par rapport à ces outils, mis à part la forte orientation agent d'INGENIAS, est que l'IDK facilite une infrastructure de programmation pour générer les nouveaux modules, avec des interfaces bien établies pour manipuler les méta-modèles.

L'expérience de cette approche en plusieurs projets montre l'augmentation de la productivité parce que les transformations permettent de réutiliser l'expérience dans l'utilisation des plates-formes en plusieurs projets avec une infrastructure similaire. L'approche fluide vers la plate-forme, avec un processus itératif et incrémental pour le développement des modules, en parallèle avec la modélisation de SMA, facilite aussi l'apprentissage et la gestion des risques. Néanmoins, il y a encore quelques problèmes à prendre en compte, comme les suivants :

- La validation des transformations n'est pas triviale. Spécialement pour détecter des erreurs quand il y a une chaîne des transformations.

- La maintenance des transformations. Une fois qu'une transformation a été créée, elle peut être utilisée en plusieurs développements. Celle-ci implique le besoin de documenter la transformation de façon appropriée et de déterminer des points d'extension pour incorporer de nouvelles fonctionnalités dans la future avec un impact minimum.

La réutilisation des modules en développements futurs permet d'améliorer la productivité, en réutilisant les patrons et l'expérience dans l'implémentation, et le processus facilite l'adaptation à des nouveaux besoins fonctionnels de système et de la plate-forme.

Pour appliquer MDA aux SMA il faut tenir en compte le fait que les méta-modèles ne sont pas faciles à spécifier normalement. Plus concrètement, il n'y a pas de méta-modèle pour les plates-formes FIPA, alors les transformations entre les méta-modèles SMA indépendant de la plate-forme et les méta-modèles spécifiques des plate-forme n'ont pas de solution triviale. Pour cela, l'approche INGENIAS-MDD est plus réaliste. Il facilite l'intégration du langage de modélisation SMA avec l'implémentation des éléments pour le processus de transformation.

5. CONCLUSIONS GENERALES ET TRAVAUX EN COURS

Une des conséquences de l'approche MDD d'INGENIAS c'est que les modèles de SMA sont devenus l'élément clef dans le processus de développement. La valeur du concept agent réside dans la modélisation plutôt que comme un élément d'implémentation. Il réduit la distance entre la conception de système et le domaine d'application. Cette approche s'intègre avec la proposition des langages de modélisation spécifiques du domaine (http://www.dsmforum.org/), mais le point de départ est plus près du but avec les langages de spécification SMA qu'avec les langages orientés objets. L'utilisation des méta-modèles apporte la flexibilité nécessaire pour adapter facilement le langage de modélisation aux différents domaines. L'utilisation des langages de modélisation orientés agents facilite aussi l'application des méthodes de la théorie de SMA, assez divers.

En même temps, la conception est aussi plus indépendante de la plate-forme cible. La disponibilité des outils IDK facilite la réutilisation de l'expérience dans l'implémentation. Néanmoins, le processus de définition des transformations n'est pas évident et demande un test extensif du code résultant. Il s'agit d'un processus incrémental, très itératif. Mais à la fin il permet de réutiliser l'expérience de la plate-forme cible, en patrons de conception adaptés au domaine et à la plate-forme, qui sont réfléchis dans les templates et les algorithmes de transformation. La définition des transformations est aussi un élément important dans le processus de développement. Sa inclusion a demandé une révision du processus INGENIAS, avec des itérations dirigés par les caractéristiques des modèles (section 4.3).

Les méta-modèles, outils et processus de développement INGENIAS-MDD sont le résultat de l'expérience de plusieurs applications de SMA, dans le cadre de quatre projets européens, deux projets nationaux, et le transfert technologique avec plusieurs entreprises (telle que Telefónica, Ibermática, MindFields et Boeing Research & Technology Europe). Ceci nous a permis de passer progressivement d'une proposition d'architecture d'agents délibératifs à la définition d'une méthodologie pour le développement de SMA, dans laquelle on aborde le traitement de la complexité du SMA sur des aspects comme l'organisation et les interactions. Selon notre

expérience, la définition et l'utilisation d'une méthodologie sont la clé pour contrôler et gérer la complexité et le processus de développement d'un SMA et aident à l'identification des éléments qui composent le SMA et leurs relations.

La méthodologie INGENIAS a évolué des méthodes classiques vers les tendances actuelles dans le génie logiciel, plus concrètement dans le contexte de MDE. La flexibilité des mécanismes de méta-modèles et transformations nous a permis d'enrichir le noyau conceptuel d'INGENIAS et d'adapter ses concepts génériques aux nécessités concrètes de différents problèmes. Toute cette base permet de considérer plusieurs travaux futurs dont certains sont en cours :

- L'utilisation de patrons sociaux pour la vérification des propriétés des SMA. Ce travail est basé sur les concepts de la théorie de l'Activité (voir section 5.1).
- La simulation à base d'agents, comme un exemple d'adaptation de la théorie et des outils INGENIAS et les SMA à un domaine d'application concrète (voir section 5.2).
- Des métriques pour l'évaluation de logiciel de SMA (voir section 5.3).

5.1 L'analyse des contradictions individu-societé

La spécification d'un SMA implique l'identification d'un grand nombre d'entités et leurs relations. Comme on l'a déjà vu, cela demande la gestion de différentes perspectives du système. Plusieurs problèmes concernant ces aspects tirent leur origine de la présence des buts et de tâches contradictoires, d'inconsistances et de comportements inattendus. Ces configurations problématiques doivent être détectées et prévenues pendant le processus de développement pour examiner des manières alternatives de les traiter.

Il y a au moins trois types différents de contradictions dans les SMA [173]. D'abord, il y a des contradictions inhérentes au système en cours d'étude, par exemple celle due aux buts en concurrence des agents du SMA. Il y a aussi des tensions liées à la nature dynamique du SMA et son évolution. Ceci peut arriver avec le compromis entre les capacités d'apprentissage des agents et leur temps de réponse dans un environnement changeant. Finalement, il y a des erreurs d'analyse, à cause des malentendus entre clients et développeurs, ou à cause du nombre et de la variété des entités du SMA et des perspectives, que le développeur doit maintenir consistantes et mises à jour.

La plupart des méthodes pour analyser ce type de configurations conflictuelles sont basées sur des techniques de vérification traditionnelle du génie logiciel, et se focalisent sur des aspects sociaux ou intentionnels très concrets. INGENIAS propose l'utilisation des règles syntactiques

pour déterminer que l'information nécessaire pour générer un SMA exécutable est disponible ; par exemple, qu'il y a au moins une tâche associée pour la satisfaction de chaque but. DESIRE [28] considère l'échange d'information correcte entre tâches. Sichman et Demazeau [170] analysent les relations et situations de dépendance des agents en une organisation, ou la dépendance est entendue comme la possibilité qu'un agent puisse contribuer ou empêcher la satisfaction d'un but par autre agent. Toutes ces propositions se focalisent sur des aspects concrets, et nous cherchons à considérer un cadre plus générique qui permet d'intégrer plusieurs aspects de contradiction agents-societé qui puissent arriver pendant la conception de SMA. Cela avec l'hypothèse que la gestion de la complexité de SMA demande la considération des agents comme entités intentionnelles qui participent aux structures d'ordre supérieur (les organisations).

Comme dans le domaine de l'informatique les aspects sociaux ne sont pas assez considérés, nous avons considéré des théories des sciences sociales, et plus concrètement la Théorie de l'Activité (TA). Cette théorie de la psychologie, formulée initialement par Leontiev [113], établit un cadre analytique et conceptuel pour comprendre la motivation des individus et leurs relations avec la société. La TA étudie les contradictions et les conflits entre les individus et leurs communautés en environnements productifs. D'après la TA, les contradictions sont la force que guide le développement. Nous avons considéré l'identification des patrons de contradictions de la TA pour découvrir des situations conflictuelles dans les modèles de SMA. La proposition des solutions alternatives pour améliorer et faire évoluer ces modèles part des réponses à ces contradictions. Ces réponses sont fournies aussi par la TA.

En TA, le comportement individuel ne peut pas être compris en dehors du contexte mais, en même temps, les personnes changent leur contexte. Cette interaction complexe s'appelle *activité*, l'unité fondamentale de l'analyse de la TA. Essentiellement, l'activité reflète un processus, avec les nivaux individuel et social entrelacés. Le contexte de l'activité est le système d'activité, qu'englobent plusieurs éléments et relations, sont représentés dans la Figure 47.

Le niveau individuel de l'activité est caractérisé par le sujet (*subject*), l'élément actif qui réalise l'activité. Il peut être un individu ou un groupe d'individus. Les besoins du sujet sont représentés par le but (*objective*). Le but peut être satisfait par un résultat (*outcome*), qui est produit par transformation des objets (*objects*). Pour réaliser cette transformation, le sujet utilise des outils (*tools*). Les outils arbitrent le processus de sujet sur les objets. Un outil est quelque chose utilisée dans un processus, comme par exemple un ordinateur ou un plan.

Au niveau social, la communauté (*community*) représente les sujets qui partagent le même objet. Une activité peut concerner plusieurs sujets et chaque sujet peut jouer un ou plusieurs rôles et avoir plusieurs motivations. Les règles (*rules*) arbitrent entre le sujet et la communauté. Elles spécifient comment les sujets s'adaptent aux communautés avec des normes explicites et implicites, avec des conventions, et avec des relations sociales dans une communauté. La division de travail (*division of labour*) arbitre entre l'objet et la communauté. Elle regarde l'organisation de la communauté en relation avec le processus de transformation de l'objet pour produire un résultat.

Tous ces concepts sont interconnectés par des relations d'arbitrage (*mediation relationships*). Ces relations reflètent les liens conceptuels d'après la signification des concepts qu'elles associent. Comme exemple, on considère les arbitrages mentionnés auparavant des outils entre sujets et objets, ou avec les règles qui gouvernent les sujets dans les communautés.

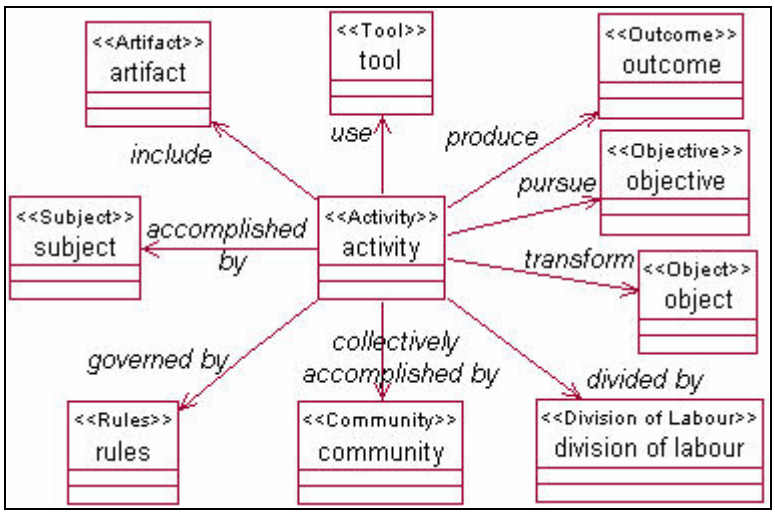


Figure 47. Concepts et relations d'arbitrage de la Théorie de l'Activité

Avec ces concepts, la TA formule les situations sociales, ses changements, ses contradictions et ses solutions. Ces formulations sont d'une nature discursive et demandent une connaissance approfondie de la signification exacte des concepts involucrés et les éléments de base conceptuels sous-jacents. Pour être applicable dans un processus de génie logiciel dans le domaine des SMA il faut formaliser la théorie. Cette formalisation permettra aussi de développer des outils qui soient intégrables avec les outils de développement de SMA, tel que l'IDK. Ainsi, nous avons défini un langage, UML-AT, comme nouveau profil UML. À part les éléments de la TA, il y a de nouveaux éléments nécessaires pour raisonner sur les contradictions et les solutions identifiées par la TA. Par exemple, les relations de contribution, inspirées de i* [184], permettent de montrer comment les artefacts s'influencent les uns les autres. Cela est utile pour montrer la satisfaction des buts, la construction des objets, ou le dégât des outils.

Elles sont représentées par plusieurs types des relations : *contribute* (*positively*, *negatively*, ou *undefined*), *essential*, et *impede*. Il y a aussi d'autres concepts en UML-AT, comme celui d'artefact, qui permet de représenter n'importe quelle classe de concept comme une abstraction générique. La spécification complète d'UML-AT se trouve à *http://grasia.fdi.ucm.es/at/uml-at*.

À partir de ce langage de concepts de TA, on peut définir des patrons de contradictions. Ceux-ci montrent les structures ou les patrons (voir Figure 48) qui décrivent des situations de contradiction et leurs possibles solutions. La description textuelle d'une contradiction explique la signification de la contradiction dans le contexte d'un SMA. Par exemple, l'origine de la contradiction et la classe des effets non désirés qui peuvent en découler. Chaque paire de patrons décrit une correspondance, qui représente la situation de contradiction, et une solution possible. Les deux patrons ont une description textuelle et un diagramme UML-AT. Ces explications sont utiles pour aider les développeurs à comprendre le propos de la contradiction.

La représentation UML-AT permet aux outils de chercher des structures de modèles de SMA qui correspondent aux patrons de correspondance. Les patrons de solution décrivent des dispositions alternatives des éléments originaux de la spécification avec, parfois, des changements dans leur configuration ou l'inclusion de nouveaux éléments.

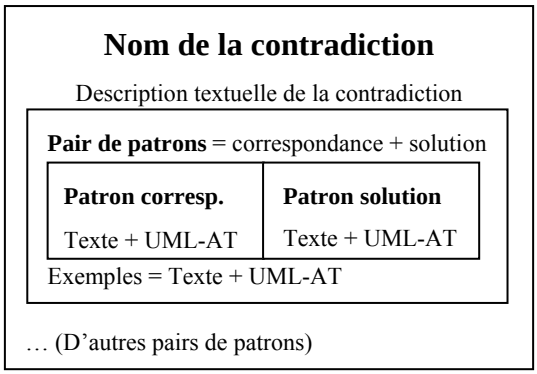


Figure 48. Structure pour la description des contradictions de TA

Cette structure peut servir de base pour construire un répertoire de plusieurs patrons de contradictions. Cela servira pour alimenter les outils. Actuellement nous disposons déjà d'un répertoire de dix contradictions (*http://grasia.fdi.ucm.es/at/contradictions*) dont certaines illustrées dans plusieurs publications [66][67][68]. Un prototype d'outil et un processus d'application de cette théorie dans le cadre d'INGENIAS ont été développés par R. Fuentes dans sa thèse de doctorat [62], co-encadré par Jorge Gómez-Sanz et Juan Pavón.

Le prototype d'outil a été développé en Prolog et n'est pas intégré dans l'éditeur de l'IDK. Nous envisageons de l'intégrer mieux, de façon qu'un assistant puisse aider le développeur pendant

l'édition des modèles avec l'IDK. Aussi, nous pensons améliorer l'algorithme de correspondance des patrons de contradiction. Actuellement le nombre des correspondances qui sont repérées est plus élevé que celui qui est effectivement réel. Cela est dû en partie au fait que pour intégrer UML-AT avec INGENIAS il faut établir une correspondance des concepts entre les deux et parfois il y a des concepts d'UML-AT qui correspondent à plusieurs d'INGENIAS et vice-versa. En tenant compte du contexte d'utilisation il est possible de mieux identifier les correspondances valables. Avec une base de patrons de contradictions plus large que la base actuelle, et les améliorations proposées, on pourrait disposer d'un outil de vérification des propriétés sociales de SMA en INGENIAS.

Ces travaux ont conduit aussi à la définition de bibliothèques de patrons sociaux. Ces patrons permettent de réutiliser des structures sociales en plusieurs projets de développement de SMA, de façon similaire aux patrons de conception dans l'orientation objets. De plus, s'ils sont associés aux templates pour la génération de code sur plateformes d'agents, on peut fournir l'information au processus de transformation des modèles SMA au code, dans notre approche MDD pour SMA. Une première étude de ces idées, appliquée à un cas d'étude d'un SMA de filtrage collaboratif a été présentée récemment dans la conférence ESAW 2006 [69].

5.2 Simulation sociale à base d'agents

Une application de la flexibilité de l'utilisation de méta-modèles et l'approche MDD en INGENIAS est la réalisation d'outils pour faciliter la simulation des systèmes sociaux. Actuellement il y a plusieurs outils de simulation sociale à base d'agents. Mais, pour la plupart des utilisateurs potentiels de ces outils, comme les sociologues et les économistes, leur emploi n'est pas évident. La raison principale est que les modèles doivent être spécifiés comme des programmes, normalement avec un langage orienté objets, Java principalement. Il y a quelques essais pour faciliter la spécification des modèles avec des outils graphiques et des bibliothèques de comportements prédéfinis. Par exemple, Sesam [112] offre une interface graphique pour spécifier les comportement des agents comme des machines à états finis. Ou bien, comme environnement de développement rapide, en utilisant le langage Python, RePast Py (repast.sourceforge.net/repastpy/) fournit un assistant pour construire un modèle. Mais ces outils, pour l'instant, ne sont applicables que pour des modèles assez simples. Un autre aspect à considérer est que les modèles d'agents supportés par les outils de simulation sont assez simples, principalement des agents réactifs, avec un comportement qui est programmé directement par la redéfinition de certaines méthodes de classes de base. Une exception est

SDML (sdml.cfpm.org), qui propose un langage de modélisation déclaratif, mais son apprentissage n'est pas facile.

En examinant les langages de modélisation des méthodologies de développement de SMA, par contre, on trouve une plus grande capacité d'expression avec des concepts agents plus élaborés, pas uniquement pour définir les agents, mais pour les aspects organisationnels aussi. En plus, dans les cas comme INGENIAS, avec l'utilisation de méta-modèles il est possible d'adapter le langage au domaine d'application. Par conséquent, l'expérience dans le domaine de génie logiciel orienté agents peut apporter plusieurs avantages à la modélisation et la simulation des systèmes sociaux que la programmation directe.

Nous partons de cette hypothèse et abordons le problème en plusieurs étapes. D'abord on considère directement le langage de modélisation INGENIAS et on essaie de modéliser plusieurs cas d'étude avec le but de confirmer que les concepts de SMA utilisés sont suffisants pour la modélisation des systèmes sociaux. Cela nous a montré, comme discuté après, qu'il faut introduire quelques éléments, spécialement dans le point de vue environnement, pour satisfaire ce but. Une fois qu'on peut modéliser les systèmes sociaux, on définit des transformations pour générer le code, à partir de modèles, sur plusieurs outils de simulation. Cela permet de profiter de la fonctionnalité et des capacités de ces outils. Aussi, on peut considérer l'exécution du même modèle sur plusieurs plates-formes de simulation. Cela permettra la réplication, pour examiner les possibles variations de résultats en tenant compte des caractéristiques de chaque outil. Un troisième pas consiste à interpréter les résultats fournis par les outils en termes d'éléments des modèles SMA INGENIAS. Finalement, le but est de définir des langages adaptés aux domaines d'études sociales spécifiques, ce qui permettra aux spécialistes de disposer d'outils de travail plus appropriés à leur besoins et expertise. On peut voir dans tout ce parcours le rôle extensif des méta-modèles et transformations qu'on préconise dans la méthodologie INGENIAS.

L'utilisation d'INGENIAS pour spécifier des modèles sociaux a déjà commencé pour résoudre quelques problèmes dans les organisations d'agents, plus concrètement pour maintenir dans un SMA la *santé organisationnelle*. Cela veut dire que dans un SMA ouvert, ou il y a des agents qui entrent et sortent, il faut regarder que les agents obéissent à certains normes de comportement qui contribuent à une bonne évolution de propos de l'organisation. Celle-ci a été montrée dans le cas d'un système de filtrage collaboratif pour plusieurs utilisateurs d'un site web [84], où certaines règles peuvent être introduites pour maintenir cette santé organisationnelle.

D'autres cas ont été développés avec le but plus précis d'expérimenter avec des problèmes de simulation sociale. L'étude des marchés est un problème classique qui a été abordé autant par la communauté de simulation sociale, comme les travaux du groupe d'Organisation Industriel et Etudes de Marches de la Univ. Valladolid [152], ou bien dans le domaine de SMA, par exemple les travaux liés au concept d'institutions électroniques [161]. Ici, le problème principal trouvé est de spécifier la multitude des stratégies et des distributions de participants dans ce type de marchés, et les limites dans les outils pour considérer un nombre d'agents élevé. Ce cas d'étude a été utile pour analyser les problèmes de passage à l'échelle.

Plus intéressante a été l'incorporation dans notre équipe de recherche de sociologues pour aborder cette problématique. Le cas d'étude considéré a été la modélisation du problème proposé dans la thèse doctorale de M. Arroyo [3], qui analyse l'évolution des attitudes religieuses dans la société espagnole à la fin de XX^{ème} siècle. Ce problème a été initialement spécifié en termes de discours sociologique. Il identifie plusieurs patrons d'individus et essaie d'expliquer leur évolution pendant 40 ans, en tenant compte des résultats des enquêtes qui ont été faites périodiquement. Le comportement des individus, et plus précisément leurs attitudes vis-à-vis du fait religieux, est modélisé avec des éléments de l'état mental des agents. L'intérêt du sociologue est de voir comment les interactions entre les agents (représentations des individus de la société) et l'évolution de l'environnement ont influencé l'évolution de l'état mental (attitudes religieuses dans ce cas) des agents. Ceci est actuellement le cas d'étude qu'on considère le plus représentatif des problèmes du domaine social que l'on vise. Les concepts développés ici pourront être appliqués à d'autres attitudes (politiques, écologiques, culturelles, ethnologiques, etc.).

Dans la modélisation de ces expériences, et avec le but de pouvoir simuler les modèles des systèmes sociaux, on a repéré trois extensions nécessaires pour adapter INGENIAS à ces problèmes : le réseau social, le temps et l'espace.

Un élément récurrent dans beaucoup de ces problèmes est la formation dynamique de réseaux sociaux, et pour les représenter on a fait une extension du concept de groupe ou structure organisationnelle d'INGENIAS. Les groupes INGENIAS sont statiques, définis pendant la modélisation de système. Le réseau social est un groupe qui peut s'établir comme résultat des interactions entre agents. Par exemple, dans [164] on présente une étude sur l'altruisme dans une société, ou les réseaux sociaux se déterminent dynamiquement à partir de relations de crédit.

Les autres extensions considèrent l'emplacement des agents dans un espace de l'environnement et l'évolution du temps. Ces deux aspects sont couramment considérés par tous les outils de simulation à base d'agents.

La perspective temporelle considère le progrès du temps dans le modèle pendant l'exécution de la simulation. On considère que les simulations sont plutôt dirigées par le temps que par les événements puisque la plupart des outils de simulation à base d'agents suivent ce schéma (une raison pour cela est qu'un schéma dirigé par événements demanderait un coordinateur central pour les événements ou bien une synchronisation assez complexe entre les agents). Cela veut dire que le modèle ne considère pas de temps constant pour simuler le cycle perception-réaction des agents le long du temps.

La perspective spatiale décrit comment les agents se placent dans l'environnement. En général, les outils de simulation offrent deux ou trois dimensions spatiales avec des configurations diverses.

Pour les transformations, on a étudié la génération de code pour deux plates-formes : RePast et Mason [163]. La méthode est la même que celle qui a été expliquée dans la section 3.3 pour la génération de code avec l'IDK. On a défini des schémas d'implémentation des agents INGENIAS comme des extensions des classes des agents des plates-formes cibles. À partir des modèles spécifiés avec l'éditeur de l'IDK on peut instancier le code Java nécessaire pour exécuter les simulations.

Un avantage de cette approche est la possibilité de répliquer les simulations sur des plates-formes différentes. Comme on a noté en [163], il est intéressant parfois d'étudier les effets de différentes stratégies de planification. Également, pour avoir accès à différentes facilités de présentation de résultats.

Il est possible aussi de faire une simulation directe des modèles avec l'IDK, en utilisant le module de génération de code pour la plateforme JADE, qu'on a présenté dans la section 2.5.2. Ici l'exécution évolue pas à pas, dirigée par l'utilisateur. Celle-ci peut être un travail ennuyeux, mais il est utile pour debogeur et valider le modèle dans le sens que le SMA fait ce qu'il est supposé faire. Nous n'avons pas considéré de développer un moteur de simulation plus puissant pour l'IDK puisque ceux existants sont déjà assez riches en fonctionnalité.

Avec cette approche on espère faciliter la spécification des modèles de systèmes sociaux avec un langage graphique au lieu d'un langage de programmation. Cela permet un niveau d'abstraction plus élevé, plus près du domaine d'application. Même si la construction des modules de transformation de modèles en code demande la participation d'un ingénieur en

informatique, qui devrait être expérimenté avec les méta-modèles INGENIAS et le langage de programmation Java. L'utilisateur final aurait besoin uniquement de savoir utiliser l'éditeur graphique et interpréter les résultats de la simulation. Des modules plus avancés pourraient être développés pour faciliter cette interprétation.

5.3 Métriques pour l'évaluation de méthodologies orientées agent

Une des questions qui arrivent souvent quand on propose l'utilisation de la technologie agent est : Combien coûte la construction d'une application avec une approche multi-agents ?

Le génie logiciel a essayé de répondre à cette question pour d'autres paradigmes. Il y a des modèles de prédiction bien connus comme COCOMO et COCOMO II, ou Putnam [20]. Ils prennent l'expérience de développements précédents pour prédire le coût de nouveaux développements similaires. Avec cette expérience, les gestionnaires des projets peuvent construire des estimations de coût, en tenant compte des variables comme l'expertise moyenne des programmateurs, la complexité du problème, le nombre des lignes de code, les précédents, etc.

Pour adopter cette démarche dans les applications multi-agents on a besoin :

- Du code des applications multi-agents pour faire des mesures. Le code des applications multi-agents connues n'est pas disponible au public ; il est difficile ainsi d'avoir une collection d'applications sur lesquelles se baser.
- De l'adaptation des méthodes de génie logiciel. Les métriques conventionnelles considèrent certains aspects, comme l'encapsulation et la complexité des interfaces, qui sont différentes de celles que l'on considère pour comprendre les développements orienté agents. Par conséquent, les estimations de l'effort sur les paramètres traditionnels ne sont pas applicables à notre domaine.

Deux questions doivent alors être abordées. Pour obtenir les mesures nous considérons les différents développements qu'on a réalisés dans le contexte de plusieurs projets. Certains résultats ont été publiés dans le workshop AOSE 2005 [89], pour les projets PSI3, Eurescom P815 et Eurescom P907. Dans cette contribution on propose déjà une liste de caractéristiques des SMA qui ont été implémentées dans ces projets et qui ont été intégrées dans le modèle d'estimation COCOMO II [19]. Pour évaluer les coûts résultants en fonction du temps de développement et des ressources humaines, on avait utilisé trois outils : Eclipse Metrics Plugin (http://metrics.sourceforge.net), CodeCount (http://sunset.usc.edu/research/CODECOUNT), et USC COCOMO II v2000 (http://sunset.usc.edu/available_tools).

La disponibilité de métriques pourra aider les développeurs pour estimer l'effort nécessaire pour leurs projets et déterminer leurs coûts. Nous avons comparé nos modèles d'estimation avec les données qu'on avait enregistrées pendant le développement des projets et l'analyse de code résultant.

Les SMA qu'on a développés ont les caractéristiques que l'on considère normalement dans les modèles de SMA INGENIAS : contrôle BDI pour les agents (avec buts et tâches), interactions avec un modèle de contrôle de session et communications désignés au niveau abstrait (indépendants du middleware sous-jacent), et capacités de perception de l'environnement. À partir de ces caractéristiques on avait défini des variables de sociabilité (interactions, messages), de comportement (tâches, règles, règles de gestion des buts, machines à état finis) et d'information (entités mentales, événements, buts). On a dû ajuster certains facteurs d'échelle et d'effort qui sont proposés dans le modèle COCOMO II.

Les résultats obtenus, analysés dans [89] sont cohérents avec notre expérience. Mais ils posent un certain nombre de questions à examiner dans le futur. D'abord, les métriques proposées s'adaptent bien au modèle d'architecture de SMA appliqué dans les trois projets. Mais dans la communauté agents il y a une bonne diversité. Nous avons réalisé qu'il faut plus de données et d'expérimentations avec les modèles COCOMO II pour mieux identifier les aspects des agents qui ont le plus d'influence dans le développement des SMA.

Le processus de développement doit être aussi considéré. Dans les cas étudiés, on a suivi une approche type processus unifié mais dans nos projets récents, avec les facilités MDD, nous combinons des techniques de développement rapide avec ingénierie *round-trip*. Dans ces nouveaux cas il y a une influence assez importante de la réutilisation de bibliothèques de composants, schémas des applications, et des transformations, qui contribuent notamment à réduire l'effort de développement. Il faudrait réviser alors ces modèles COCOMO II pour s'adapter à la nouvelle approche INGENIAS-MDD présenté dans ce travail. Pour cela, nous sommes en train de mesurer plus systématiquement les différents développements en cours.

5.4 Autres développements avec INGENIAS

Dans la version actuelle de l'IDK il est possible de parcourir les modèles d'une spécification de SMA pour vérifier les propriétés du système, pour la génération de documentation HTML et pour la génération de code pour différentes plates-formes. Un des buts finaux est de faciliter et de promouvoir l'extension de l'environnement IDK par des tiers. En effet, cette ligne de travail s'est déjà initiée, et il y a plusieurs groupes de recherche qui appliquent les méthodes et outils actuellement disponibles dans l'IDK, et avec lesquels on a établi une collaboration, comme :

- Univ. Castilla la Mancha (A. Fernández Caballero) : création d'un langage spécifique pour le domaine de systèmes de surveillance électronique en adaptant l'environnement INGENIAS (par extension des méta-modèles) [148].
- Univ. Castilla la Mancha (M. Piattini) : qui appliquent INGENIAS pour développer des SMA pour la maintenance de software [162].
- Univ. Murcia (J. Botia) : module de test des interactions en SMA massives. Il s'agit d'intégrer son outil ACLAnalyzer comme un module de l'IDK [23].
- Univ. Politécnica de Valencia (V. Botti) : extensions du temps réel (RT-MESSAGE) et des systèmes holoniques (travail de thèse doctorale de A. Giret) [75].
- Univ. Salamanca (M. Corchado) : modélisation de services pour téléphones mobiles avec les agents CBR-BDI [35].
- Univ. Sevilla (R. Corchuelo) : extensions du modèle d'interactions [149].
- Univ. Santiago de Compostela (S. Barro y P.F. Lamas) : INGENIAS s'emploie comme méthodologie de développement dans le projet MEDICI-SICAU (travail de thèse doctorale de S. Fraga).
- UNED (B. Barros) : Utilise INGENIAS comme méthodologie pour créer un outil graphique d'auteur dans un contexte d'apprentissage collaboratif avec des documents actifs (travail de thèse doctorale de T. Sastre).
- Univ. Palermo (M. Cossentino) : spécification SPEM du processus INGENIAS.

BIBLIOGRAPHIE

[1] Amor M., Fuentes L., and Vallecillo A. Bridging the Gap Between Agent-Oriented Design and Implementation Using MDA. In: Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004. Lecture Notes in Computer Science 3382, Springer Verlag, (2005) 93-108.

[2] Ana Mas. *Agentes Software y Sistemas Multiagente: Conceptos, Arquitecturas y Aplicaciones*. Pearson Educación-Prentice Hall. 2005.

[3] Arroyo, M. *Cambio cultural y cambio religioso, tendencias y formas de religiosidad en la España de fin de siglo*. Servicio de Publicaciones de la UCM. Madrid (2004).

[4] Austin, J.L. *How to do things with words.* Oxford University Press. 1962.

[5] Azaiez, S., Huget, M.P., Oquendo, F. *Approach for Multi-Agent Metamodelling*. International Journal on Multi-Agent and Grid Systems, Special Issue on Agent-Oriented Software Development Methodology. (À paraître)

[6] Bauer B, Müller J., & Odell J. Agent UML: A Formalism for Specifying Multiagent Interaction. In: *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000*. LNCS 1957, Springer-Verlag (2001) 91—103.

[7] Bauer B, & Odell J. UML 2.0 and Agents: How to Build Agent-based Systems with the new UML Standard. J*ournal of Engineering Applications of Artificial Intelligence* 18 (2), 2005, 141-157.

[8] Bellifemine F., Poggi A., and Rimassa, G. Developing multi-agent systems with a FIPA-compliant agent framework. *Software Practice and Experience 31*, 2 (2001) 103—128.

[9] Bergenti, F., Gleizes, M.P., Zambonelli, F. (Eds.) *Methodologies and Software Engineering for Agent Systems*. Kluwer. 2004.

[10] Bernon, C. et al. *A Study of Some Multi-agent Meta-models*. 5th International Workshop, AOSE 2004. LNCS 3382, Springer Verlag (2005) 62-77.

[11] Bernon, C., Cossentino, M., Pavón, J. Agent Oriented Software Engineering. *The Knowledge Engineering Review 20, 2* (2005), 99-116.

[12] Bernon, C., Cossentino, M., Pavón, J. An Overview of Current Trends in European AOSE Research. *Informatica 29, 4* (2005), 379-390.

[13] Bernon C., Camps V., Gleizes M.P., Picard, G. *Engineering Adaptive Multi-Agent Systems: the ADELFE Methodology*. In B. Henderson-Sellers and P. Giorgini (Eds.), Agent-Oriented Methodologies. Idea Group Pub (2005) 172-202.

[14] Beydoun, G., González-Pérez, C., Low, G. and Henderson-Sellers, B.: *Synthesis of a Generic MAS Meta-model.* In: Proc. Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'05), LNCS 3914, Springer Verlag (2006) 126-142.

[15] Bézivin, J. Breton, E. Dupé, G., Valduriez, P. *The ATL Transformation-based Model Management Framework*, Research Report No.03.08, Université de Nantes, Sep. 2003.

[16] Blain, G., Guessoum, Z., Perrot, J.-F. Thiefaine, A. *Génération de systèmes multi-agents à partir de modèles.* Technique et Science Informatiques (TSI), 22(4): 107-111 (2003)

[17] Blanco Moreno D. *Metodología para la creación de modelos de evaluación de plataformas para el desarrollo de sistemas multi-agente*. Trabajo de doctorado. Dep. de Sistemas Informáticos y Programación. Universidad Complutense Madrid. 2002.

[18] Boehm, B. W. 1988. *A spiral model of software development and enhancement*. En R. Donald, editor, *Software Management*, 120–131. IEEE Computer Society Press, 1993. Reprinted from Computer, Vol. 21, No. 5, May 1988, 61-72.

[19] Boehm, B.W., Sullivan, K.J.: *Software Cost Estimation with COCOMO II*. Prentice Hall (2000).

[20] Boehm, B.W., Sullivan, K.J.: *Software economics: a roadmap*. In: Proceedings of the Conference on The future of Software Engineering. ACM Press (2000) 319-343.

[21] Boissier, O. 2001. *Modèles et architectures d'agents*. Chapitre 2, 71-108 en [30]

[22] Bond, A.H., Gasser, L. *An Analysis of Problems and Research in DAI*. In: Readings in Distributed Intelligence. Morgan Kaufman Pub. (1998) 3-35.

[23] Botía, J.A., Gómez-Sanz, J., Pavón, J. *Intelligent Data Analysis for the Verification of Multi-Agent Systems Interactions.* In: 7th Int. Conference on Intelligent Data Engineering and Automated Learning (IDEAL 2006). E. Corchado et al. (Eds.), LNAI 4224, Springer-Verlag (2006) 1207-1214.

[24] Botti, V. *Aplicaciones de los Sistemas Multiagente*. In: Ana Mas. *Agentes Software y Sistemas Multi-Agente: Conceptos, Arquitecturas y Aplicaciones*. Pearson-Prentice Hall. 2005. 185-253.

[25] Bouron, T. Application des systèmes multi-agents dans les télécommunications. Briot, J.P. & Demazeau Y. (Eds.) 2001. *Systèmes Multi-Agents. Principes et architectures*. Collection IC2. Hermes Science Publications. 207-233.

[26] Bradshaw, J.M. et al. *KAoS: A Generic Agent Architecture for Aerospace Applications*. Proceedings of the CIKM '95 Workshop on Intelligent Information Agents.

[27] Bratman, M. E. *Intentions, Plans, and Practical Reason*. Harvard University Press. 1987.

[28] Brazier, F. et al. *Desire: Modelling Multi-Agent Systems In A Compositional Formal Framework*. International Journal of Cooperative Information Systems, special issue on Formal Methods in Cooperative Information Systems, 6 (1997) 67-94.

[29] Bresciani, P. et al. Tropos: An Agent-Oriented Software Development Methodology. *Journal of Autonomous Agents and Multi-Agent Systems, 8*(3) (2004), 203-236.

[30] Briot, J.P. & Demazeau Y. (Eds.) *Systèmes Multi-Agents. Principes et architectures*. Collection IC2. Hermes Science Publications. 2001.

[31] Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez-Sanz, J. J., Pavon, J., Kerney, P., Stark, J., & Massonet, P. *Agent Oriented Analysis using MESSAGE/UML*. Second International Workshop, AOSE 2001, Montreal, Canada, May 29, 2001. Revised Papers and Invited Contributions. LNCS 2222. Springer-Verlag (2002) 119-135.

[32] Chaib-draa, B. Industrial applications of distributed AI. *Communications of the ACM*, 38, 11 (1995), 47–53.

[33] Coleman, D. et al. *Object-Oriented Development: The Fusion Method*. Prentice Hall. 1994.

[34] Collis, J. C. & Ndumu, D. T. *Zeus Methodology Documentation*. Applied Research and Technology, BT Labs. 1999.

[35] Corchado, J., Pavón, J., Corchado, S. & Castillo, L. *Development of CBR-BDI Agents. A Tourist Guide Application*. In: Proc. 7th European Conference on Case-Based Reasoning (ECCBR 2004), P. Funk P., P.A. González Calero (Eds.), Advanced in Case-Based Reasoning, LNCS 3155, Springer-Verlag (2004) 547-559.

[36] Cossentino, M. *From Requirements to Code with the PASSI Methodology.* In: Agent Oriented Methodologies. B. Henderson-Sellers and P.Giorgini (Eds.) IDEA Group Publishing (2005) 79-106.

[37] Cossentino, M., Guessoum Z. and Pavón, J.: Roadmap of Agent-Oriented Software Engineering. In: Methodologies and Software Engineering for Agent Systems—The Agent-Oriented Software Engineering Handbook. Kluwer (2004) 431-450

[38] de Hoog, R., Martil, R., Wielinga, B., Taylor, R., Bright, C., & van de Velde, W. *The CommonKADS model set*. ESPRIT Project P5248 KADS-II/M1/DM..1b/UvA/018/5.0, University of Amsterdam, Lloyd's Register, Touche Ross Management Consultants & Free University of Brussels. 1993.

[39] Decker, K. Task Environment Centered Simulation. *Simulating Organizations: Computational Models of Institutions and Groups*, AAAI Press/MIT Press (1996).

[40] Decker, S. K. & Lesser, V. R. Dessiging a family of coordination algorithms. In: *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, AAAI Press (1995) 73-80.

[41] DeLoach, S. & Wood, M. *Developing Multiagent Systems with agentTool*, en Y. Lesperance & C. Castelfranchi, editores, Intelligent Agents VII - Proceedings of the 7th International Workshop on Agent Theories, Architectures, and Languages (ATAL'2000).

[42] DeLoach, S., Wood, M.F. & Sparkman, C. *Multiagent Systems Engineering*. International Journal of Software Engineering and Knowledge Engineering 11, 3 (2001) 231-258.

[43] DeMaria, B.; Silva, V.; Lucena, C.: MDA Based Approach for Developing Multi-Agent System. In: Proceeding of the Forum of the 17th Conference on Advanced Information Systems Engineering, June 13-17, Porto, Portugal, 2005

[44] Demazeau Y. *From Interactions To Collective Behaviour In Agent-Based Systems.* Proceedings of the First European Conference on Cognitive Science, Saint Malo, France, 117-132, Avril 1995.

[45] Demazeau, Y. & Occello, M. *Software Systems Development as Societies of Agents* (abstract), 15th IFIP Conference, IT&KNOWS'98, OGAI, 303-304.

[46] Depke, R., Heckel, R., Küster, J.M. *Agent-Oriented Modeling with Graph Transformation*. In: *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000*. LNCS 1957, Springer-Verlag (2001) 105-120.

[47] Durfee, E. H., Lesser, V. R. & Corkill, D. *Trends in Cooperative Distributed Problem Solving*. IEEE Transactions on Knowledge and Data Engineering. 1989.

[48] El Fallah Seghrouchni, A., Haddad, S. *A Recursive Model for Distributed Planning.* In: M. Tokoro (Ed.), Proceedings Second International Conference on Multi-Agent Systems, ICMAS-96, AAAI Press (1996) 307-314.

[49] El Fallah Seghrouchni, A., Magnin, L. (Eds). *Fondements des Systèmes Multi-Agents : Modèles, spécifications formelles et vérification*. Hermès. 2001.

[50] El Fallah Seghrouchni, A., Suna, A. *CLAIM and SyMPA: A Programming Environment for Intelligent and Mobile Agents*. In: Multi-Agent Programming: Languages, Platforms and Applications, Kluwer Academic Publishers (2005) 95-122.

[51] Esteva, M, De la Cruz, D. & Sierra, C. *Islander: an electronic institutions editor*. In Proceedings of the first international joint conference on Autonomous agents and multiagent systems. ACM Press (2002) 1045–1052.

[52] Eurescom P815. *Communications Management Process Integration Using Software Agents*. http://www.eurescom.de/public/projects/P800-series/P815/P815.htm

[53] Eurescom P907. *MESSAGE: Methodology for Engineering Systems of Software Agents*, http://www.eurescom.de/~public-webspace/P900-series/P907/index.htm

[54] Evans, R. (ed.) et al. *Methodology for Agent-Oriented Software Engineering.* Eurescom Project P907. EDIN 0223-0907. 2001.

[55] Ferber, J. & Gutknecht, O. *A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems*. Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98), IEEE CS Press (1998) 128-135.

[56] Ferber, J., *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*. Harlow: Addison Wesley Longman. 1999.

[57] Fikes, R. y Nilsson, J.: *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving*. Artificial Intelligence 2 (1971), 189-208.

[58] FIPA: *Foundation for Intelligent Physical Agents*. http://www.fipa.org.

[59] FIPA Modeling TC. *Agent Class Superstructure Metamodel*. Document Status: Preliminary. 21 July 2004.

[60] Fisher, M. *A Survey of Concurrent METATEM The Language and its Applications*, Temporal Logic - Proceedings of the First Intemational Conference, LNCS 827, Springer Verlag (1994).

[61] Franklin, S., Graesser, A. *Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents*. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages. Springer-Verlag, 1996.

[62] Fuentes, R. *Teoría de la Actividad para el Desarrollo de Sistemas Multiagente*. Thèse de doctorat de l'Universidad Complutense Madrid. 2004.

[63] Fuentes, R., Gómez-Sanz, J., Pavón, J. *Verification and Validation Techniques for Multi-Agent Systems*, Upgrade V (4): 15-19 (aussi en espagnol: *Técnicas de verificación y validación para Sistemas Multi-agente*, Novática 170 (2004) 11-14)

[64] Fuentes, R., Gómez-Sanz, J., Pavón, J., Uden, L. *Activity Theory applied to Requirements Elicitation of Multi-Agent Systems*, First International Workshop on Activity Theory Based Practical Methods for IT Design, ATIT 2004.

[65] Fuentes, R., Gómez-Sanz, J. & Pavón, J. *Activity Theory for the Analysis and Design of Multi-Agent Systems*. In: Proc. 4th International Workshop on Agent-Oriented Software Engineering (AOSE 2003), P. Giorgini, J.P. Müller, y J. Odell (Eds.), Agent-Oriented Software Engineering IV, LNCS 2935, Springer-Verlag (2004) 110-122.

[66] Fuentes, R., Gómez-Sanz, J. & Pavón, J. *Social Analysis of Multi-agent Systems with Activity Theory*. In: 10th Conference of the Spanish Association for Artificial Intelligence

(CAEPIA 2003), R. Conejo, M. Urretavizcaya, J.L. Pérez-de-la-Cruz (Eds.), Selected Papers from CAEPIA 2003, LNAI 3040, Springer-Verlag (2004) 526 - 535.

[67] Fuentes, R., Gómez-Sanz, J., Pavón, J. *Checking Social Properties of Multi-Agent Systems with Activity Theory*. In: Proc. IX Ibero-American Conference on Artificial Intelligence (Iberamia 2004), C. Lemaitre et al. (Eds.), LNAI 3315, Springer-Verlag (2004), 1-11.

[68] Fuentes, R., Gómez-Sanz, J., Pavón, J. *Managing Conflicts between Individuals and Societies in Multi-Agent Systems*. In: Proc. Fifth International Workshop Engineering Societies in the Agent World (ESAW 2004), M.P. Gleizes, A. Omicini, and F. Zambonelli (Eds.), Engineering Societies in the Agents World V, LNAI 3451, Springer-Verlag (2005), 106-118.

[69] Fuentes, R., Gómez-Sanz, J., Pavón, J. *Model Driven Development of Multi-Agent Systems with Repositories of Social Patterns*. In: Proc. Sixth International Workshop Engineering Societies in the Agent World (ESAW 2006).

[70] Garijo et al. *Development of a Multi-Agent System for cooperative work with network negotiation capabilities*, en Intelligent Agents for Telecommunication Applications. LNAI 1437, Springer Verlag (1988) 204-219.

[71] Garijo, F., Gomez-Sanz, J., Pavon, J, & Massonet, P. *Multi-Agent System Organization. An Engineering Perspective*. Proceedings MAAMAW 2001, 1-15.

[72] Genesereth, M. R. & Ketchpel, S. P. Software agents. *Communications of the ACM*, 37, 7 (1994) 48–53.

[73] Georgeff, M.P. & Lanski, A.L. *Procedural Knowledge*. Proceedings of the IEEE Special Issue on Knowledge Representation, vol. 74. 1986.

[74] Gervais, M. *ODAC: An Agent Oriented Methodology Based on ODP*. Autonomous Agents and Multi-Agent Systems 7(3): 199-228 (2003).

[75] Giret, A., Botti, V., and Valero, S. MAS Methodology for HMS. In: Holonic and Multi-Agent Systems for Manufacturing: Second International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS 2005. Lecture Notes in Artificial Intelligence, Vol. 3593. Springer-Verlag (2005) 39-49.

[76] Giret, A. 2005. *ANEMONA: Una Metodología Multi Agente para Sistemas Holónicos de Fabricación.* Thèse de doctorat de l'Universidad Politécnica Valencia.

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

[78] Gleizes, M.P., Picard, G. *OpenTool, outil pour la réalisation de systèmes multi-agents adaptatifs dans le cadre de la méthode ADELFE*. Technique et Science Informatiques 22(4), Hermes (2003) 249-253.

[79] Gómez Sanz, J. *Modelado de Sistemas Multi-Agente*. Thèse de doctorat de l'Universidad Complutense Madrid. 2002.

[80] Gómez Sanz, J. & Pavón, J. *Meta-modelling in Agent Oriented Software Engineering*. In: Proc. 8th Ibero-American Conference on AI (Iberamia 2002), F.J. Garijo, J.C. Riquelme, M. Toro (Eds.), Advances in Artificial Intelligence, LNAI 2527, Springer-Verlag (2002) 606-615.

[81] Gómez Sanz, J. & Pavón, J. *Methodologies for Developing Multi-Agent Systems*. Journal of Universal Computational Science 10, 4 (2004) 359-374.

[82] Gómez-Sanz, J. & Pavón. *INGENIAS Development Kit (IDK) Manual*, vers. 2.5.2. Universidad Complutense Madrid (2005).

[83] Gómez-Sanz J. J., & Pavón, J.: A Discussion on the MDA Approach for Agent Development. In: Proc. Third European Workshop on Multi-Agent Systems (EUMAS 2005) 160-166.

[84] Gómez-Sanz, J. & Pavón. *Implementing Multi-agent Systems Organizations with INGENIAS*. In: Programming Multi-Agent Systems: Third International Workshop, ProMAS 2005, Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, Amal ElFallah Seghrouchni (Eds.), LNCS 3862, Springer-Verlag (2006) 236 - 251.

[85] Gómez Sanz, J. & Pavón, J. *Template Based Code Generation for Rapid Software Development*. (2006).

[86] Gómez-Sanz, J., Pavón, J. & Díaz Carrasco, A. *The PSI3 Agent Recommender System*. In: Proc. International Conference on Web Engineering (ICWE 2003), J.M. Cuella Lovelle et al., Web Engineering, LNCS 2722, Springer-Verlag (2003) 30-39.

[87] Gómez Sanz, J., Pavón, J. & Garijo, F. *Meta-models for Building Multi-Agent Systems*. In: Proc. The 2002 ACM Symposium on Applied Computing (SAC 2002), G. Lamont et al. (Eds.), ACM Press (2002) 37-41.

[88] Gómez-Sanz, J., Pavón, J. & Garijo, F. *Intelligent Interface Agents Behaviour Modelling*. Proc. MICAI 2000, LNAI 1793, Springer Verlag (2000) 598-609.

[89] Gómez-Sanz, J., Pavón, J. & Garijo, F. *Estimating Costs for Agent Oriented Software*. 6th International Workshop, AOSE 2005, Utrecht, The Netherlands, July 25, 2005. Revised and Invited Papers. LNCS 3950, Springer Verlag (2006) 218-230.

[90] Guessoum Z. and Briot J.-P. (1999), *From active objects to autonomous agents*. In Special Series on Actors and Agents, edited by Dennis Kafura and Jean-Pierre Briot, IEEE Concurrency, 7(3):68-76, July-September 1999.

[91] Guessoum, Z., Faci, N.. DimaX: A Fault Tolerant Multi-Agent Platform. To appear in Proc. of ICSE 2006 Workshop on Software Engineering for Large-Scale Multi-Agent Systems V (SELMAS 2006). ACM Press, May 2006.

[92] Guessoum, Z., Cossentino, M. and Pavón, J. A Roadmap of Agent-Oriented Software Engineering: The European Agentlink Perspective. In: *Methodologies and Software Engineering for Agents Systems*, Federico Bergenti, Marie-Pierre Gleizes and Franco Zambonelli (eds.), Kluwer Publishing (2004) 430-450.

[93] Gutknecht, O. & Ferber, J. *The MADKIT Agent Platform Architecture*. Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems: Int. Workshop on Infrastructure for Scalable Multi-Agent Systems, 2000 Revised Papers. LNCS 1887, Springer Verlag (2001) 48-55

[94] Huang, J. and Jennings, N.R. and Fox, J. *An Agent-based Approach to Health Care Management*. Applied Artificial Intelligence 9, 4 (1995) 401-420.

[95] Hunhns, M. & Singh, M.P. *A Multiagent Treatment of Agenthood*. en Applied Artificial Intelligence: An International Journal, Volume 13, No. 1-2, January-March 1999, 3-10

[96] Hunhns, M. *Multiagent Systems and Societies of Agents.* In: Chapter 2. of G. Weiss (ed.), Multiagent Systems. MIT Press. 1999.

[97] Hyacinth S. Nwana & Divine T. Ndumu. *A Perspective on Software Agents Research*. In: *The Knowledge Engineering Review*, January 1999.

[98] IBM alphaWorks. *ABLE. Agent Building and Learning Environment*. 2000. http://www.alphaworks.ibm.com/tech/able

[99] Iglesias, C. A., Garijo, M. & Gonzalez, J. C.. *A survey of agent-oriented methodologies*. In: J. P. Muller, M. P. Singh, and A. S. Rao (ed.), *Intelligent Agents V*, LNCS 1555, Springer Verlag (1998) 317–330.

[100] Iglesias, C., Garijo, M., Gonzalez J. & Velasco J. *Analysis and design of multiagent systems using MAS-CommonKADS*. In: M. P. Singh, A. Rao, and M. J. Wooldridge (ed.), Intelligent Agents IV, LNAI 1365, Springer Verlag (1998) 313-326.

[101] ILOG Jrules, http://www.ilog.com/products/rules/engines/jrules/

[102] IntelliOne Technologies. AgentBuilder. 2002. http://www.agentbuilder.com

[103] ITU-T. Recommendation Z100. *CCITT Specification and Description Language (SDL*). 1993.

[104] Jacobson, I., Rumbaugh, J., and Booch, G. 1999. *The Unified Software Development Process* Addison-Wesley.

[105] Jennings, N. & Wooldridge, M. 1998. *Applications of Intelligent Agents*. In: Agent Technology: Foundations, Applications, and Markets (eds. N. R. Jennings and M. Wooldridge) 3-28.

[106] Jennings, N. *On agent-based software engineering*, Artificial Intelligence 117 (2000) 277-296.

[107] Jennings, N., Sycara, K. & Wooldridge, M. 1998. *A Roadmap of Agent Research and Development*. Int Journal of Autonomous and Multi-Agent Systems. Vol. 1, 7-38. Kluwer Academics Publishers. 1998.

[108] JESS. Java Expert System Shell, http://herzberg.ca.sandia.gov/jess/

[109] Kelly, S., Lyytinen, K. S. & Rossi, M. *METAEDIT+ A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment*. Proc. 8th Int. Conference Advances Information System Engineering, CaiSE'96. LNCS 1080, Springer Verlag (1996) 1-21.

[110] Kendall E. *A Methodology for Developing Agent Based Systems for Enterprise Integration*, IFIP Working Conference of TC5 SIG on Architectures for Enterprise Integration, Queensland, Australia, November 1995.

[111] Kinny, D., Georgeff, M. & Rao, A. *A methodology and modelling technique for systems of BDI agents*. En W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNAI 1038*, Springer Verlag (1996) 56–71.

[112] Kluegl, F., Herrler, R., Fehler, M. *SeSAm: Implementation of Agent-Based Simulation Using Graphical Specification*, In: Proceedings of the AAMAS 2006, Hakodate, Demonstration Program; pp. 1439f.

[113] Lemaître, C., Excelente, C.B. *Multi-Agent Network for Cooperative Work*. Expert Systems with Applications 14 (1-2), Elsevier (1998) 117-127.

[114] Lemaître, C., Excelente, C.B. *Multiagent Organization Approach*. In: Proceedings 2nd Iberoamerican Workshop on Distributed Artificial Intelligence and Multi-Agent Systems, Toledo (1998) 7-16.

[115] Leontiev, A. *Activity, Consciousness, and Personality*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1978

[116] Lind, J. *Iterative Software Engineering for Multiagent Systems: The MASSIVE Method* Springer 2001.

[117] Ljungberg, M. & Lucas, A. 1992. *The Oasis Air Traffic Management System*. In Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence, PRICAI '92,

[118] Luck, M., Griffiths, N. & d'Inverno M. *From agent theory to agent construction: A case study*. En J. P. Müller, M. Wooldridge, & N. R. Jennings, editores, *Intelligent Agents III, LNAI 1193*. Springer-Verlag (1997).

[119] Luck, M., McBurney, & P., Preist C. *A Manifesto for Agent Technology: Towards Next Generation Computing*. Journal of Autonomous Agents and Multi-Agent Systems 9, 3 (2004) 203-252.

[120] Luhers, R., Pavón, J. & Schneider, M. *DEMOS Tools for On-line Discussion and Decision Making*. In: Proc. Int. Conference on Web Engineering (ICWE 2003), J.M. Cuella Lovelle et al. (Eds.), Web Engineering, LNCS 2722, Springer-Verlag (2003) 525-528.

[121] Maes, P. *Agents that Reduces Work and Information Overload*. Readings in Intelligent User Interfaces. Morgan Kauffman Publishers, Inc. 1998.

[122] Malone, T.W. & Crowston, K. The Interdisciplinary Study of Coordination. *ACM Computing Surveys 26, 1* (1994) 87-119.

[123] Mazouzi, H., El Fallah Seghrouchni, A., Haddad, S. *Open protocol design for complex interactions in multi-agent systems.* In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems (AAMAS'02), ACM Press (2002) 517-526.

[124] McCallum, A. *Bow: A Toolkit for Statistical Language Modeling, Text Retrieval, Classification and Clustering*. 1998. http://www-2.cs.cmu.edu/~mccallum/bow/

[125] Medvidovic N. *A Classification, Comparision Framework for Software Architecture Description Languages*. Technical Report. Departement of Information. Computer Science. University of Irvine. California. 1996.

[126] Microgold. WithCLASS. http://www.microgold.com/, 2003.

[127] Moulin, B. & Brassard, M. *A scenario-based design method and environment for the development of multiagent systems*, En Proceedings of the 1st Australian Workshop on Distributed Artificial Intelligence, LNAI 1087 - Springer Verlag (1996) 216-231.

[128] Negroponte, N. *Being Digital*. Hodder and Stoughton. 1995.

[129] Newell A. The knowledge level. *Artificial Intelligence, 18* (1982) 87-127.

[130] Novikau, A. Perini, A., Pistore, M. *Graph Rewriting for Agent Oriented Visual Modeling.* Proc. of the International Workshop on Graph Transformation and Visual Modeling Techniques, in ETAPS 2004 Conference, Barcelona, Spain, 2004.

[131] Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C. ZEUS: A Toolkit for Building Distributed Multi-Agent Systems. *Applied Artificial Intelligence Journal* 1, 13 (1999) 129-185.

[132] Ocello, M., Baeijs, C., Demazeau, Y., Koning, J.L. *MASK: An AEIO Toolbox to Design and Build Multi-Agent Systems.* Knowledge Engineering and Agent Technology, IOS Press (2004) 114-135.

[133] Odell, J., Nodine, M., and Levy, R.: A Metamodel for Agents, Roles, and Groups. In: Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004.

Lecture Notes in Computer Science, Vol. 3382. Springer-Verlag, Berlin Heidelberg (2005) 78–9

[134] Odell, J., Van Dyke Parunak, H. & Bauer, B. Representing agent interaction protocols in UML. En P. Ciancarini and M. Wooldridge (eds.), Agent-Oriented Software Engineering - Proceedings of the First International Workshop (AOSE-2000). Springer-Verlag (2001).

[135] O'Malley, S.A. DeLoach, S.A. *Determining When to Use an Agent-Oriented Software Engineering Paradigm*, en Proceedings of the 2nd International Workshop on Agent-Oriented Software Engineering (AOSE'01), Montreal, Canada, May 2001, 9-16.

[136] OMG. Meta Object Facility (MOF) Specification. Version 1.4 (2002) formal/02-04-03.

[137] OMG. *Model-Driven Architecture Guide Version 1.0.1* (2003), OMG Document omg/2003-06-01.

[138] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation. OMG Document ptc/05-11-01 (2005).

[139] Omicini, A., Ricci, *A. MAS Organization within a Coordination Infrastructure: Experiments in TuCSoN*. In: Engineering Societies in the Agents World IV, 4th International Workshop, ESAW 2003. LNCS 3071 Springer-Verlag (2004) 200-217.

[140] Oquendo F. *The ArchWare Refinement Language*. Deliverable D6.1b. ArchWare European RTD Project. IST-2001-32360. 2003.

[141] Parunak, H. V. D. Applications of distributed artificial intelligence in industry. En: G. M. P. O'Hare, N. R. Jennings (Eds.) *Foundations of Distributed Artificial Intelligence*, 139−164. Wiley. 1996.

[142] Parunak, H. V. D. *Manufacturing experience with the contract net*. In: M. N. Huhns (Ed.) *Distributed AI*. Morgan Kaufmann. 1987.

[143] Pavón, J. 1996. *Towards Integration of Service and Network Management in TINA*. Journal of Networks and Systems Managemetn, 4, 3 (septiembre), 299-318

[144] Pavón, J. & Gómez-Sanz, J. *Agent Oriented Software Engineering with INGENIAS*. In: Proc. 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003), V. Marik, J. Müller, M. Pechoucek (Eds.), Multi-Agent Systems and Applications II, LNAI 2691, Springer-Verlag (2003) 394-403.

[145] Pavón, J., Corchado, J.M., Gómez-Sanz, J., Castillo, L.F. *Mobile Tourist Guide Services with Software Agents*. In: Proc. First International Workshop on Mobility Aware Technologies and Applications (MATA 2004), A. Karmouch, L. Korba, E. Madeira (Eds.), Mobility Aware Technologies and Applications, LNCS 3284, Springer-Verlag (2004) 322-330.

[146] Pavón, J., Gómez-Sanz, J.J., Fuentes, R. *The INGENIAS Methodology and Tools.* In: Agent Oriented Methodologies. B. Henderson-Sellers and P.Giorgini (Eds.) IDEA Group Publishing (2005) 236-276.

[147] Pavón, J., Gómez-Sanz, J.J. & Fuentes, R.: Model Driven Development of Multi-Agent Systems. In: European Conference on Model Driven Architecture (2006), LNCS 4066, Springer Verlag (2006) 284-298.

[148] Pavón, J., Gómez-Sanz, J., Valencia-Jiménez, J.J., Fernández-Caballero, A. *Desarrollo de un sistema inteligente de vigilancia multi-sensorial con agentes softwareS*. In: Proc. CMPI 2006.

[149] Peña, J., Corchuelo, R. *Towards a Coordination Specification of Complex Multi-Agent Systems*. In: New Trends on Information Technology. 2nd International Workshop on Agents and Multiagent Systems (IWPAAMS 2004) 54-64.

[150] Picard, G., Bernon, C., Gleizes, M., and Peyruqueou, S. ADELFE: a methodology for adaptive multi-agent systems engineering. In Petta, P., Tolksdorf, R., and Zambonelli, F., editors, Engineering Societies in the Agents World III: Third International Workshop (ESAW 2002), LNCS 2577, Springer Verlag (2002) 156–169.

[151] Perini, A., Susi, A.: Automating Model Transformations in Agent-Oriented Modeling. 6th International Workshop, AOSE 2005, Utrecht, The Netherlands, July 25, 2005. Revised and Invited Papers. LNCS 3950, Springer Verlag (2006) 167-178.

[152] Posada, M., Hernández, C., López-Paredes, A. *Learning in ContinuousDouble Auctions.* P. Mathieu et al. (Eds.), , Lecture Notes in in Economics and Mathematical Systems, 564, Springer Verlag (2005) 41-51.

[153] *Proceedings of the First International Conference on Autonomous Agents'97*, Marina del Rey, CA. ACM Press. 1997.

[154] *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-agent Systems*, London. 1997.

[155] *Proceedings of the First International Symposium on Agent Systems and Applications, Third International Symposium on Mobile Agents*. Palm Springs, CA. IEEE-CS. 1999.

[156] Rao, A. & Georgeff, M. *Modeling Rational Agents within a BDI Architecture*. Proceedings of the 2nd international conference on Principles of Knowledge Representation and Reasoning (1991) 473-484.

[157] Revault N., Sahraoui H.A., Blain G., Perrot J.-F., (1995), A Meta-modeling Technique: the MetaGen System, in proc. TOOLS Europe'95 proceedings, TOOLS 16, Prentice Hall

[158] Rich, E. y Knight, K.: *Artificial Intelligence*. McGraw-Hill. 1990.

[159] Ricordel, P. M. & Demazeau, Y. *From Analysis to Deployment: a Multi-Agent Platform Survey*. Proceedings of 1st Int. Workshop on Enginnering Societies in the Agents World (ESAW), ECAI`2000, 93-105.

[160] Ricordel, P. M. *Programmation Orientée Multi-Agents, Développement et Déploiement de Systèmes Multi-Agents Voyelles*. Institut National Polytechnique de Grenoble. 2001.

[161] Rodríguez-Aguilar, J.A., Martín, J., Noriega, P., Garcia, P., Sierra, C. *Towards a Test-Bed for Trading Agents in Electronic Auction Markets*. AI Commun. 11(1): 5-19 (1998)

[162] Rodríguez, O., Vizcaíno, A., Martínez, A. I., Piattini, M., & Favela, J. *Using a Multi-Agent Architecture to Manage Knowledge in the Software Maintenance Process*. Proc. Int. Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES'2004), Wellington, New Zealand, 1181-1188.

[163] Sansores, C. & Pavón, J.. *Agent-Based Simulation Replication: A Model Driven Architecture Approach*. In: Proc. 4th Mexican International Conference on Artificial Intelligence (MICAI 2005), A. Gelbukh, A. Albornoz, H. Terashima-Marín (Eds.), Advances in Artificial Intelligence, LNAI 3789, Springer Verlag (2005), 244-253.

[164] Sansores, C., Pavón, J. y Gómez-Sanz: Visual Modeling for Complex Agent-Based Simulation Systems. In: J.S. Sichman and L. Antunes (Eds.): Int. Workshop on Multi-Agent-Based Simulation 2005, Lecture Notes in Artificial Intelligence, Vol. 3891, Springer-Verlag (2006) 174–189.

[165] Sansores, C., Pavón, J. & López-Paredes, A. 2004. *A Framework for ABSS on the Grid*. In: Proc. Second Conference of the European Social Simulation Association (ESSA'04)

[166] Schmidt, D.C.: Model Driven Engineering. IEEE Computer 39, 2 (2006), 25-31.

[167] Selic, B. *The Pragmatics of Model-Driven Development.* IEEE Software 20, 5 (2003), 19-25.

[168] Shoham, Y. *Agent-oriented programming*. Artificial Intelligence, 60, 1 (1993) 51- 92.

[169] Shoham, Y., Tennenholtz, M. *On the emergence of social conventions: Modeling, analysis, and simulations*. Artificial Intelligence, vol. 94, no. 12 (1997) 139-166.

[170] Sichman, J., Demazeau, Y. *On Social Reasoning in Multi-Agent Systems*. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, Special Issue on Development of Multi-Agent Systems, n°13, AEPIA (2001) 68-84.

[171] Smith, R. The contract net protocol. IEEE Trans. on Computers, C-29, 12 (1980) 1104-1113.

[172] Susi, A. et al. The Tropos Metamodel and its Use. Informatica, An International Journal of Computing and Informatics 29 (2005), 401-408

[173] Sycara, K. *Multi-Agent Systems*. AI Magazine. Summer 1998; 85-92.

[174] Thiefaine, A., Guessoum, Z., Perrot, J.F., Blain, G. *Génération de systèmes multi-agents à partir de modèles*. Technique et Science Informatiques 22(4), Hermes (2003) 107-111.

[175] Van Eijk, R. M., De Boer, F. S., Van Der Hoek, W. & Meyer, J. J. Ch. *A Verification Framework for Agent Communication*, Autonomous Agents and Multi-Agent Systems, vol. 6, pp. 185-219, (2003)

[176] Visser, E. *A survey of strategies in program transformation systems*. Electr. Notes Theoretical Computer Science 57 (2001).

[177] Wasserman, A. and Pircher, P. A graphical, extensible integrated environment for software development. In: *Proceedings of the Second ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, 131–142. ACM Press (1987).

[178] Weis, T., Ulbrich, A., and Geihs, K: Model Metamorphosis. IEEE Software 20, 5 (2003), 46-51

[179] Weiss, G. et al. *Proceedings of the European Agent Systems Summer School*. University of Utrecht, the Netherlands. 1999.

[180] Wood, M. & DeLoach, S. *Developing Multiagent Systems with agentTool*. ATAL 2000. LNAI 1986. Springer-Verlag (2000).

[181] Wooldridge, M. & Jennings, N. *Agent Theories, Architectures, and Languages: a Survey*. Intelligent Agents, Wooldridge and Jennings Eds, 1-22. Springer-Verlag (1995).

[182] Wooldridge, M. & Ciancarini, P. *Agent-Oriented Software Engineering: The State of the Art*. AOSE 2000: 1-28.

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

[184] Yu, E. *Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering*, Proc. 3rd IEEE Intl. Symposium on Requirements Engineering (RE'97), Washington D.C., USA, pp. 226-235, 1997, IEEE Press

[185] Zambonelli, F., Bergenti, F. & Dimarzo, G. *Methodologies and Software Engineering for Agent Systems*. AgentLink News (2002), 23-25.

[186] Zambonelli, F., Jennings, N. & Wooldridge, M. *Developing Multiagent Systems: The Gaia Methodology*. ACM Trans. Software Engineering and Methodology, 12, 3 (2003), 317-370.