Modélisation et Supervision d'Institutions Multi-Agents

Benjamin Gâteau

Ecole Nationale Supérieure des Mines
SAINT-ETIENNE

N° d'ordre : 442 I

THESE
présentée par

Benjamin Gâteau

Pour obtenir le grade de Docteur
de l'École Nationale Supérieure des Mines de Saint Étienne

Spécialité : Informatique

Modélisation et Supervision d'Institutions Multi-Agents

Soutenue à Luxembourg le 26 Juin 2007

Membres du jury :

Président :	
Eric Dubois	Professeur, Université de Namur
Rapporteurs :	
Christophe Sibertin-Blanc	Professeur, IRIT, Toulouse
René Mandiau	Professeur, LAMIH, Valenciennes
Directeurs de thèse :	
Olivier Boissier	Professeur, ENSM.SE, Saint-Etienne
Djamel Khadraoui	Docteur Ingénieur, CRP Henri Tudor
Examinateurs :	
Pascal Bouvry	Professeur, Université du Luxembourg
Zahia Guessoum	Maître de conférence, HDR, Université de Reims

● **Spécialités doctorales :** **Responsables :**

SCIENCES ET GENIE DES MATERIAUX	**J. DRIVER** Directeur de recherche – Centre SMS
MECANIQUE ET INGENIERIE	**A. VAUTRIN** Professeur – Centre SMS
GENIE DES PROCEDES	**G. THOMAS** Professeur – Centre SPIN
SCIENCES DE LA TERRE	**B. GUY** Maitre de recherche – Centre SPIN
SCIENCES ET GENIE DE L'ENVIRONNEMENT	**J. BOURGOIS** Professeur **–** Centre SITE
MATHEMATIQUES APPLIQUEES	**E. TOUBOUL** Ingénieur – Centre G2I
INFORMATIQUE	**O. BOISSIER** Professeur **–** Centre G2I
IMAGE, VISION, SIGNAL	**JC. PINOLI** Professeur – Centre CIS
GENIE INDUSTRIEL	**P. BURLAT** Professeur – Centre G2I
MICROELECTRONIQUE	**Ph. COLLOT** Professeur – Centre CMP

● **Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat** (titulaires d'un doctorat d'Etat ou d'une HDR)

BATTON-HUBERT	Mireille	MA	Sciences & Génie de l'Environnement	SITE
BENABEN	Patrick	PR 2	Sciences & Génie des Matériaux	SMS
BERNACHE-ASSOLANT	Didier	PR 1	Génie des Procédés	CIS
BIGOT	Jean-Pierre	MR	Génie des Procédés	SPIN
BILAL	Essaïd	DR	Sciences de la Terre	SPIN
BOISSIER	Olivier	PR 2	Informatique	G2I
BOUDAREL	Marie-Reine	MA	Sciences de l'inform. & com.	DF
BOURGOIS	Jacques	PR 1	Sciences & Génie de l'Environnement	SITE
BRODHAG	Christian	MR	Sciences & Génie de l'Environnement	SITE
BURLAT	Patrick	PR 2	Génie industriel	G2I
CARRARO	Laurent	PR 1	Mathématiques Appliquées	G2I
COLLOT	Philippe	PR 1	Microélectronique	CMP
COURNIL	Michel	PR 1	Génie des Procédés	SPIN
DAUZERE-PERES	Stéphane	PR 1	Génie industriel	CMP
DARRIEULAT	Michel	ICM	Sciences & Génie des Matériaux	SMS
DECHOMETS	Roland	PR 2	Sciences & Génie de l'Environnement	SITE
DESRAYAUD	Christophe	MA	Mécanique & Ingénierie	SMS
DELAFOSSE	David	PR 2	Sciences & Génie des Matériaux	SMS
DOLGUI	Alexandre	PR 1	Génie Industriel	G2I
DRAPIER	Sylvain	PR 2	Mécanique & Ingénierie	CIS
DRIVER	Julian	DR	Sciences & Génie des Matériaux	SMS
FOREST	Bernard	PR 1	Sciences & Génie des Matériaux	CIS
FORMISYN	Pascal	PR 1	Sciences & Génie de l'Environnement	SITE
FORTUNIER	Roland	PR 1	Sciences & Génie des Matériaux	CMP
FRACZKIEWICZ	Anna	MR	Sciences & Génie des Matériaux	SMS
GARCIA	Daniel	CR	Génie des Procédés	SPIN
GIRARDOT	Jean-Jacques	MR	Informatique	G2I
GOEURIOT	Dominique	MR	Sciences & Génie des Matériaux	SMS
GOEURIOT	Patrice	MR	Sciences & Génie des Matériaux	SMS
GRAILLOT	Didier	DR	Sciences & Génie de l'Environnement	SITE
GROSSEAU	Philippe	MR	Génie des Procédés	SPIN
GRUY	Frédéric	MR	Génie des Procédés	SPIN
GUILHOT	Bernard	DR	Génie des Procédés	CIS
GUY	Bernard	MR	Sciences de la Terre	SPIN
GUYONNET	René	DR	Génie des Procédés	SPIN
HERRI	Jean-Michel	PR 2	Génie des Procédés	SPIN
KLÖCKER	Helmut	MR	Sciences & Génie des Matériaux	SMS
LAFOREST	Valérie	CR	Sciences & Génie de l'Environnement	SITE
LE COZE	Jean	PR 1	Sciences & Génie des Matériaux	SMS
LI	Jean-Michel	EC (CCI MP)	Microélectronique	CMP
LONDICHE	Henry	MR	Sciences & Génie de l'Environnement	SITE
MOLIMARD	Jérôme	MA	Sciences & Génie des Matériaux	SMS
MONTHEILLET	Frank	DR 1 CNRS	Sciences & Génie des Matériaux	SMS
PERIER-CAMBY	Laurent	PR1	Génie des Procédés	SPIN
PIJOLAT	Christophe	PR 1	Génie des Procédés	SPIN
PIJOLAT	Michèle	PR 1	Génie des Procédés	SPIN
PINOLI	Jean-Charles	PR 1	Image, Vision, Signal	CIS
STOLARZ	Jacques	CR	Sciences & Génie des Matériaux	SMS
SZAFNICKI	Konrad	CR	Sciences de la Terre	SITE
THOMAS	Gérard	PR 1	Génie des Procédés	SPIN
VALDIVIESO	Françoise	CR	Génie des Procédés	SPIN
VALDIVIESO	François	MA	Sciences & Génie des Matériaux	SMS
VAUTRIN	Alain	PR 1	Mécanique & Ingénierie	SMS
VIRICELLE	Jean-Paul	MR	Génie des procédés	SPIN
WOLSKI	Krzysztof	CR	Sciences & Génie des Matériaux	SMS
XIE	Xiaolan	PR 1	Génie industriel	CIS

Glossaire :

PR 1	Professeur 1ère catégorie
PR 2	Professeur 2ème catégorie
MA(MDC)	Maître assistant
DR (DR1)	Directeur de recherche
Ing.	Ingénieur
MR(DR2)	Maître de recherche
CR	Chargé de recherche
EC	Enseignant-chercheur
ICM	Ingénieur en chef des mines

Centres :

SMS	Sciences des Matériaux et des Structures
SPIN	Sciences des Processus Industriels et Naturels
SITE	Sciences Information et Technologies pour l'Environnement
G2I	Génie Industriel et Informatique
CMP	Centre de Microélectronique de Provence
CIS	Centre Ingénierie et Santé

Remerciements

Je tiens à remercier en premier lieu les membres du jury ; René Mandiau et Christophe Sibertin-Blanc d'avoir accepté d'être les rapporteurs de cette thèse, Pascal Bouvry, Éric Dubois et Zahia Guessoum de faire partie des examinateurs de la soutenance, et tout particulièrement Olivier Boissier et Djamel Khadraoui d'avoir été mes co-directeurs.

∞

Je remercie également les membres du projet LIASIT et de l'Université du Luxembourg, notamment Thomas Engel, Björn Ottersten, Christoph Meinel, Gérard Hoffman et Pascal Bouvry pour m'avoir fait confiance et m'avoir permis d'être un des doctorants à effectuer sa thèse dans le cadre de ce projet. C'est grâce à cette structure que le Ministère de la Culture, de l'Enseignement Supérieur et de la Recherche du Luxembourg et le Fond National de la Recherche Luxembourgeois ont mis à ma disposition les ressources nécessaires à l'accomplissement de cette thèse.

∞

Je voudrais aussi remercier le Centre de Recherche Public Henri Tudor et son Administrateur Délégué Claude Wehenkel, le département CITI et ses directeurs Jean-Pol Michel et Éric Dubois ainsi que l'équipe Génie Logiciel et ses chefs Pierre Brimont et feu Olivier Marchand de m'avoir accueilli, épaulé et finalement intégré dans leur équipe. Je tiens à saluer l'École Nationale Supérieure des Mines de Saint-Étienne, notamment l'équipe G2I et le laboratoire Système Multi-Agents pour m'avoir fourni un environnement complémentaire de recherche et d'étude.

Enfin je voudrais saluer Jomi Hübner, Jaime Sichman, Yves Demazeau, Eugenio Oliveira et Leon van der Torre pour leurs conseils avisés durant le déroulement de mes travaux de recherche.

∞

Enfin je tiens à exprimer toute ma reconnaissance à mes amis de France et du Luxembourg pour leur soutien. Il en est de même pour ma famille qui a su être compréhensive quant à la rareté de mes visites. Et enfin mes pensées vont à celle qui m'a supporté, encouragé et aidé, ma compagne et future femme, Sandrine.

Résumé

Le domaine scientifique dans lequel nos recherches s'insèrent est celui des Systèmes Multi-Agents. Dans ce domaine, nous nous intéressons à l'élaboration de modèles globaux définissant et structurant l'activité commune d'agents autonomes. En effet l'autonomie d'un agent se traduit par sa capacité à déterminer ses propres buts face à la configuration de l'environnement ainsi qu'à ses motivations. Du fait de cette autonomie, des conflits liés à l'existence de buts propres à chacun des agents autonomes d'une société d'agents peuvent être incompatibles, complémentaires ou en compétition avec les objectifs de la société. Des modèles de gestion de conflits existent, et dans le cadre de ce travail, nous nous sommes intéressés aux modèles organisationnels.

Nous défendons la thèse selon laquelle la définition et la mise en place de contraintes globales influençant le fonctionnement d'agents autonomes doit être faite de manière à prendre en compte l'éventualité que les agents puissent ne pas respecter les contraintes selon les contextes et les objectifs individuels de chacun. Afin d'atteindre ces objectifs, nous proposons un modèle d'Institution Électronique comportant une description explicite et déclarative des contraintes à appliquer aux agents ainsi qu'un système de supervision et de renforcement de ces contraintes.

Ce modèle permet de définir et de structurer grâce à $\mathcal{M}$OISE^{Inst} l'activité d'un ensemble d'agents autonomes au sein d'une organisation normée et de superviser son respect grâce au système d'arbitrage $\mathcal{S}$YNAI. Ce dernier est composé d'un ensemble d'agents de supervision capables de mettre en place des mécanismes de régulation et de renforcement de ces contraintes sur les agents autonomes.

L'application du modèle se fait dans deux domaines totalement différents puisqu'ils concernent la création et l'exécution de contenu pour la télévision interactive d'un côté et la gestion de contrats électroniques d'une plate-forme de e-commerce de l'autre. Ils exhibent cependant la même problématique d'arbitrage d'entités autonomes devant respecter des contraintes. De plus, ces deux applications sont complémentaires par leur différence de portée de l'Institution.

Abstract

The scientific domain of our research work is Multi-Agent System. In this domain, we are interested in the development of global models defining and structuring the common activity of autonomous agents. The autonomy of an agent results in its capacity to determine its own goals vis-a-vis the configuration of the environment and its motivations. Because of this autonomy, conflicts related to the existence of goals suitable for each autonomous agent of a society can be incompatible, complementary or in competition with the society objectives. Models of conflicts management exist and within the framework of this work, we focused on the organisational models.

We assert that the definition and implementation of global constraints influencing autonomous agents functioning must be done so as to take into account the possibility that the agents can not respect constraints according to individual contexts and objectives' of each one. In order to achieve these goals, we propose an Electronic Institution model composed of a constraints' description and specification model as well as a system of supervision and reinforcement of these constraints.

This model is used to define and structure the activity of a set of autonomous agents within a normed organisation thanks to $\mathcal{M}$OISE^{Inst} and to supervise the respect of the organisation specification with $\mathcal{S}$YNAI. This arbitration system is composed of a set of supervisor agents able to implement constraints' regulation and reinforcement mechanisms on the autonomous agents.

The model experimentation is done in two completely different domains. They concern the contents creation and execution for interactive television on a side and the management of electronic contracts in a e-commerce platform on the other side. They have the same arbitration of autonomous entities having to respect constraints issue. Moreover, both applications are complementary by their Institution scope.

Table des matières

Table des figures

Liste des tableaux

Liste des acronymes

AAI	**Autonomous Actors within Institution** : Acteurs normés et arbitrés au sein d'une Institution
AdOrBAC	**Administration of Organisation Based Access Control** : Système d'administration du modèle OrBAC
AGR	**Agent-Groupe-Rôle** : Modèle organisationnel basé sur les concepts d'Agents jouant un Rôle au sein d'un Groupe
AGRE	**Agent-Groupe-Rôle-Environnement** : Modèle organisationnel basé AGR et prenant en considération l'environnement
AGREEN	**Agent-Groupe-Rôle-Environnement-Norme** : Modèle organisationnel basé sur AGRE et prenant en compte un ensemble de normes
AMELI	**Agent-based Middleware for ELectronic Institution** : Intergiciel exécutant des Institutions Électroniques définies avec ISLANDER
CAS	**Contractual Agent Societies** : Sociétés d'agents basées sur les contrats définis par Chrysanthos Dellarocas
CNET	**Contract NET protocol** : Protocole d'interactions entre agents
CS	**Contextual Specification** : Point de vue contextuel de la définition d'une organisation $\mathcal{M}$OISE^{Inst}
DAC	**Discretionary Access Control** : Type de contrôle restreignant l'accès aux objets sur base de l'identité du sujet auquel ils appartiennent
EBSME	**Electronic Business for the Small and Medium Enterprises** : Plate-forme de gestion et d'exécution de contrats électroniques relatifs aux services de traduction
ECF	**Electronic Contract Framework** : Cadre d'exécution de contrats électroniques, défini par Matthias Sallé
FS	**Functional Specification** : Point de vue fonctionnel de la définition d'une organisation $\mathcal{M}$OISE^{Inst}
IAS	**Institution Arbitration System** : Système d'arbitrage d'une Institution
IDE-eli	**Integrated Development Environment for Electronic Institutions** : Environnement de développement intégré d'Institutions Électroniques regroupant entre autres ISLANDER et AMELI
IeMA	**Institution e-contract Management Agent** : Agent institutionnel de gestion des contrats électroniques supervisant et contrôlant l'action des PeMA sur les contrats électroniques
IDL	**Institution Definition Language** : Langage de définition d'Institutions Électroniques
iDTV	**Interactive Digital Television** : Télévision numérique interactive
ITEA	**Information Technology for European Advancement** : Programme européen de recherche et développement dans le domaine de la technologie de l'information
LCR	**Logic for Contract Representation** : Modèle de contrats électroniques défini par Virginia Dignum

MaBeli **Multi-Agent Based ELectronic Institution** : Modèle d'Institution Électronique composé du langage de spécification $\mathcal{M}$OISE^{Inst}et du système d'arbitrage $\mathcal{S}$YNAI

MAC **Mandatory Access Contro** : Type de contrôle restreignant l'accès aux objets sur base de la sensibilité de l'objet et d'un ensemble d'autorisations des sujets sur les degrés de sensibilité

Moise+ **Model of Organisation for multI-agent SystEm** : Modèle Organisationnel pour les Systèmes Multi-Agents

Moise^{Inst} **Model of Organisation for multI-agent SystEm in Institution** : Modèle Organisationnel pour la définition et l'exécution d'Institution Électronique

NS **Normative Specification** : Point de vue normatif de la définition d'une organisation $\mathcal{M}$OISE^{Inst}

OASIS **Organization for the Advancement of Structured Information Standards** : Consortium international à but non-lucratif qui conduit le développement, la convergence et l'adoption de standards e-business

OE **Organisation Entity** : Instanciation d'une Organisation $\mathcal{M}$OISE^{Inst}

OMNI **Organizational Model for Normative Institutions** : Modèle Organisationnel pour les Institutions Normatives, représenté par une matrice prenant en compte les dimensions et les niveaux du modèle

OrBAC **Organisation Based Access Control** : Type de contrôle restreignant l'accès aux objets suivant l'organisation des sujets

OS **Organisation Specification** : Spécification d'une Organisation $\mathcal{M}$OISE^{Inst}

PeMA **Personnal e-contract Management Agent** : Agent personnel de gestion des contrats électroniques consultant les contrats et agissant sur les délivrables au nom de leur utilisateur

RBAC **Role Based Access Control** : Type de contrôle restreignant l'accès aux objets sur base des rôles joués par les sujets

ROADMAP **Role Oriented Analysis and Design for Multi-Agent Programming** : Méthodologie pour les SMA étendant GAIA

SIM **Specification Institution Model** : Spécification des contraintes d'une Institution Électronique

SMA **Système Multi-Agents** : Système regroupant un ensemble d'entités autonomes dont le comportement est guidé par un objectif local ou global. Domaine de recherche de l'Intelligence Artificielle Distribuée

SS **Structural Specification** : Point de vue structurel de la définition d'une organisation $\mathcal{M}$OISE^{Inst}

STEAM **Shell for TEAM work** : Modèle et cadre d'exécution d'équipes de travail pour les agents

Synai **SYstem of Normative Agent for Institution** : Système d'Arbitrage du modèle MA$\mathcal{B}_{\text{ELI}}$construit sur un ensemble d'agents de supervision organisés selon un modèle $\mathcal{M}$OISE^{Inst}

TAEMS **Task Analysis, Environment Modeling and Simulation** : Langage de modélisation permettant de décrire la structure des tâches

TOP **Team Oriented Pogramming** : Moyen de définir le comportement d'une équipe à l'aide d'une hiérarchie de rôles, d'une hiérarchie de tâches et d'un ensemble de contraintes de coordination

XACML **eXtensible Access Control Markup Language** : Langage de contrôle d'accès basé sur XML et standardisé par OASIS

SIM **Specification Institution Model** : Modèle de spécification d'Institution

Chapitre 1

Introduction

Nous présentons ici la problématique générale de la thèse. Nous allons tout d'abord exposer les motivations au développement d'un système informatique pour contraindre le fonctionnement d'un ensemble d'agents autonomes au sein de Systèmes Multi-Agents (SMA). Nous situerons également ce travail de recherche dans le domaine des SMA. Nous terminerons cette introduction par l'exposé de nos objectifs et de la démarche que nous avons suivie et détaillerons la structure du manuscrit.

1.1 Motivations

Dans le monde qui nous entoure tout individu, animal, objet ou organisation voit son comportement individuel et social régi par un ensemble de contraintes. Selon les domaines et les contextes, ces contraintes sont appelées règles, lois, normes, etc. Les lois de la gravité par exemple, nous empêchent de flotter dans les airs tandis que le code de la route nous interdit de conduire un véhicule à plus de 130 kilomètres par heure sur autoroute en France. Ces deux exemples ne définissent cependant pas le même genre de contrainte. Pour le premier, quel que soit l'individu et quelle que soit sa volonté, il ne pourra faire autrement que de respecter cette contrainte. Dans le second, un individu ayant des objectifs personnels prioritaires au code de la route sera en capacité de rouler plus vite afin, par exemple, de ne pas arriver en retard à un rendez-vous. Ainsi l'individu est autonome vis-à-vis de cette contrainte du code de la route et peut la violer en encourant les différentes sanctions qui peuvent en découler.

Dans le cadre de ce travail nous nous intéressons exclusivement à ce deuxième type de contrainte, à savoir aux contraintes pouvant faire l'objet d'un raisonnement explicite sur leur respect.

L'application des contraintes est complexe. Elles s'appliquent à des entités certes, mais parfois dans des contextes organisationnels particuliers. Si nous reprenons l'exemple du code de la route, la limitation à 130 kilomètres par heure est valable sur les autoroutes françaises mais ne l'est plus sur les autoroutes belges. Cette limitation n'existe même plus dans le contexte d'un circuit automobile. Les contextes font que les contraintes influencent différemment les entités.

L'application des contraintes peut également être différente selon l'individu. Ainsi, toujours dans le cas du code de la route, un véhicule prioritaire ne sera pas influencé de la même manière qu'un véhicule normal. Il sera autorisé à dépasser les limitations de vitesse. Par contre, même si un véhicule n'y est pas autorisé, il peut rouler plus vite et ainsi violer une contrainte. Cela est possible dans le cas où, sur une autoroute, un véhicule roule à une vitesse de 140 kilomètres par heure afin de dépasser un véhicule roulant à une vitesse de 120 kilomètres par heure. Il n'est pas autorisé à dépasser la limitation de vitesse mais il le fait pour atteindre un objectif individuel. Un véhicule peut également être amené à violer certaines règles du code de la route pour des raisons de sécurité et donc contribuer à un objectif global que le strict respect du code n'aurait pas permis de satisfaire.

1.2 Contexte

Le domaine scientifique dans lequel nos recherches s'insèrent est celui des Systèmes Multi-Agents. Un SMA est un système informatique constitué d'un ensemble d'agents interagissant et pouvant montrer une activité organisée. Un agent est une entité logicielle ou matérielle qui interagit, via des modalités de perception et d'action, avec un environnement *partagé* avec d'autres agents ou directement par l'échange d'actes langagiers avec ces *autres agents*. Son comportement peut être qualifié d'*autonome*. L'autonomie se traduit par la capacité de chaque agent à déterminer ses propres buts, plans ou actions face à la configuration de l'environnement, face à ses motivations mais également par rapport aux autres agents du système ou par rapport aux engagements qu'il a contractés auprès d'autres agents du système.

Du fait de cette autonomie, la gestion de conflits ou d'interférences entre agents est un problème central de ce domaine de recherche. Ceux-ci sont liés au partage d'un environnement et de ressources communes entre les agents, à l'existence de buts propres à chacun des agents qui peuvent être incompatibles, complémentaires ou en compétition. Des modèles sophistiqués de gestion de ces conflits ont été et sont encore élaborés dans le domaine. Ils se traduisent par exemple par des protocoles de négociation [111], par des modèles organisationnels [29]. Dans le cadre de ce travail, nous nous intéresserons essentiellement à ces dernières approches.

Ainsi, en imaginant que, dans l'exemple précédent, les conducteurs soient des agents autonomes, ceux-ci évoluent dans différents contextes organisationnels, type d'infrastructure routière ou législation du pays dans lequel se trouve la route. Ainsi, leur objectif principal et individuel étant d'aller d'un point A à un point B à une certaine vitesse pourrait entrer en conflit avec l'objectif, voire l'obligation commune au sein de l'organisation de ne pas dépasser la limite de vitesse autorisée.

La problématique de recherche dans laquelle nos travaux s'insèrent, concerne l'élaboration de modèles globaux (communs à l'ensemble des agents) définissant et structurant l'activité d'un ensemble d'agents autonomes. Ces formalismes rejoignent l'ensemble des travaux issus de la communauté d'Intelligence Artificielle et les lois qui s'attachent, dans le domaine des SMA, à la modélisation et la formalisation d'institutions [10] ou de normes [26].

1.3 Objectifs et Démarche

Dans ce manuscrit, nous défendons la thèse que la définition et la mise en place de contraintes globales sur le fonctionnement d'agents autonomes doivent être faites de manière à permettre aux agents de ne pas respecter ces contraintes selon les contextes et les objectifs individuels des agents. Afin que le SMA puisse cependant fonctionner d'une manière cohérente, il est nécessaire de mettre en place un ensemble de mécanismes de régulation et de renforcement de ces contraintes sur les agents autonomes.

Ainsi, nous proposons un modèle d'Institution Multi-Agents. Afin de répondre à notre objectif, cette institution doit comporter (i) une description explicite et déclarative des contraintes à appliquer aux agents afin que ceux-ci puissent raisonner et délibérer sur leur respect ou non et (ii) un système de supervision et de renforcement de ces contraintes. Afin de répondre aux exigences d'ouverture et de flexibilité qui sont essentielles dans les applications informatiques actuelles, ce modèle d'institution doit permettre à n'importe quel nouvel agent de rejoindre/quitter l'institution et de reconfigurer dynamiquement la spécification de contraintes pour, par exemple, mettre en place de nouvelles règles exceptionnelles (par exemple obliger tout véhicule à rouler moins vite en raison de fortes chaleurs et de pic d'ozone dans les zones urbaines). Cet objectif principal se décompose donc en deux sous-objectifs :

1. Élaborer un modèle pour contraindre le comportement d'entités autonomes tout en leur permettant de raisonner et de conserver leur autonomie quant à l'interprétation de ces contraintes. Ce modèle doit permettre d'organiser un ensemble d'entités autonomes afin de spécifier la structure et le fonctionnement de cet ensemble. Sur cette base, le modèle définit un ensemble de normes construites sur la structure et le fonctionnement des entités afin de définir leurs droits et devoirs.

2. Élaborer un système de gestion de l'institution pour contrôler le comportement d'un ensemble d'entités autonomes contraintes avec le modèle précédent. Ce système, composé d'agents autonomes, doit vérifier que toutes les contraintes sont respectées par les entités autonomes.

Le modèle d'institution que nous voulons définir doit satisfaire nos exigences en terme de supervision, de flexibilité et d'ouverture. Sa réalisation est validée dans deux domaines d'application :

- le domaine du multimédia et plus particulièrement dans celui de la télévision numérique interactive (iDTV) afin de définir des spécifications normatives et déclaratives de comportements d'objets autonomes dans des environnements dynamiques. Pour ceci nous voulons que ces objets suivent des scénarios (afin de réglementer leurs comportements) exprimés sous forme déclarative et qu'ils soient capables de les interpréter.
- le domaine du commerce électronique et plus particulièrement dans celui du *e-contracting* afin de définir des spécifications normatives (obligation, permission, interdiction) de comportements. Pour cela nous utilisons et enrichissons la notion de contrats électroniques utilisée au sein de l'application EBSME (**E**lectronic **B**usiness for the **S**mall and **M**edium **E**nterprises) [65].

Ces deux domaines fournissent deux perspectives intéressantes et complémentaires d'étude de cette problématique. En effet le passage de l'expérimentation d'une Institution Électronique dans le domaine iDTV vers celui du *e-contracting* illustre une évolution d'une société organisée d'agents dans une institution à une multitude d'institutions organisées contenant seulement deux agents. Ces deux domaines d'application bien qu'éloignés, fournissent des nouveautés intéressantes autant pour les entreprises que pour les utilisateurs. Enfin, l'utilisation de ces deux domaines valide la généricité de notre système d'arbitrage.

1.4 Organisation du manuscrit

Ce manuscrit est organisé en trois parties (cf. Fig. 1.1). La première partie dresse un panorama des modèles d'institution en termes de description de leurs systèmes de contrainte ainsi que de leur système de gestion et de supervision. La seconde partie décrit notre modèle ainsi que sa mise en œuvre. La troisième partie aborde deux cas d'utilisation de notre modèle dans deux domaines différents.

Partie I : État de l'art Dans cette première partie, nous faisons un état des lieux des concepts et des différents travaux liés à notre problématique des institutions au sein des SMA.

Chapitre 2 ▷ Le monde qui nous entoure est régi par des règles, lois, normes dont nous faisons un rapide inventaire dans les domaines de l'économie, du droit, de la sociologie et des politiques de sécurité informatique. Ce chapitre a pour objectif de mettre en évidence les points communs entre les différents modèles afin de fixer un cadre d'analyse des modèles d'institutions au sein des SMA.

Chapitre 3 ▷ A partir du cadre d'analyse du chapitre précédent, nous étudions plus particulièrement les modèles d'organisations et de descriptions au sein des SMA.

Chapitre 4 ▷ Poursuivant notre analyse des travaux au sein des SMA, ce chapitre s'attache à étudier les systèmes de gestion d'institution proposés dans la littérature Multi-Agents. Ces systèmes viennent en support aux descriptions d'institutions que nous avons décrites dans le chapitre précédent.

Chapitre 5 ▷ Cette première partie se termine avec l'énoncé des caractéristiques et de nos exigences pour un modèle d'Institution Multi-Agents. Par rapport à cela, nous synthétisons les manques des modèles et systèmes étudiés dans l'état de l'art.

Partie II : Ma$\mathcal{B}_{\text{eli}}$ Dans cette partie, nous détaillons les composantes de notre modèle d'institution MA$\mathcal{B}_{\text{ELI}}$.

Chapitre 6 ▷ Nous présentons dans ce chapitre le modèle organisationnel de description d'institution : $\mathcal{M}$OISE^{Inst}. Ce modèle spécifie la structure, le fonctionnement, les contextes et les normes des organisations que les agents rejoindront et qui auront une influence sur leur comportement. Ce modèle est basé sur une extension du modèle $\mathcal{M}$OISE^{+} par l'ajout de contextes et de normes.

Chapitre 7 ▷ Ce chapitre présente $\mathcal{S}$YNAI, le système de supervision d'institution de MA$\mathcal{B}_{\text{ELI}}$ composé d'un ensemble d'agents institutionnels. Ces agents ont leur comportement contraint par une organisation $\mathcal{M}$OISE^{Inst} dont nous présentons les différentes spécifications. En plus de leur architecture, nous présentons les différentes actions qu'ils peuvent exécuter ainsi que les protocoles qu'ils doivent suivre et respecter afin de détecter et sanctionner toute violation d'une spécification organisationnelle par les agents du domaine.

Chapitre 8 ▷ Ce chapitre présente la mise en œuvre de $\mathcal{M}$OISE^{Inst} et de $\mathcal{S}$YNAI au travers de leur API JAVA respective.

Partie III : Réalisation et Expérimentation Dans cette dernière partie, nous illustrons l'utilisation de MA$\mathcal{B}_{\text{ELI}}$ dans deux cas concrets d'étude. Ces deux domaines complémentaires visent à prouver la généricité de MA$\mathcal{B}_{\text{ELI}}$ par le biais du fonctionnement identique du système de supervision et par la mise en œuvre, d'une part d'un petit nombre d'institutions composées d'un grand nombre d'agents et d'autre part d'un grand nombre d'institutions composées d'un petit nombre d'agents.

Chapitre 9 ▷ Ce chapitre illustre l'utilisation de MA$\mathcal{B}_{\text{ELI}}$ au sein d'une application de télévision interactive utilisée comme démonstrateur du projet européen ITEA Jules Verne. Grâce à $\mathcal{M}$OISE^{Inst}, nous organisons un ensemble d'Avatars ainsi que leurs droits et leurs devoirs concernant aussi bien la partie agent que la partie multimédia des Avatars. Une fois le modèle $\mathcal{M}$OISE^{Inst} et l'implémentation présentés, nous détaillons un cas d'utilisation et les résultats obtenus.

Chapitre 10 ▷ Ce chapitre vient donc compléter le précédent en illustrant cette fois l'utilisation de MA$\mathcal{B}_{\text{ELI}}$ au sein d'une application de gestion de contrats électroniques. De la même manière nous présentons le modèle $\mathcal{M}$OISE^{Inst} utilisé pour spécifier le contrat conclu entre les parties via leur assistant virtuel et définissant les droits et devoirs de chacun au sein de cet accord. Les résultats obtenus sont comparés à ceux du domaine précédent.

Cette partie se termine sur une validation générale et complémentaire de l'utilisation de MA$\mathcal{B}_{\text{ELI}}$ avant de laisser place au dernier chapitre présentant les conclusions et les perspectives de ces travaux sur la supervision d'un ensemble d'entités autonomes à l'aide d'une Institution Électronique.

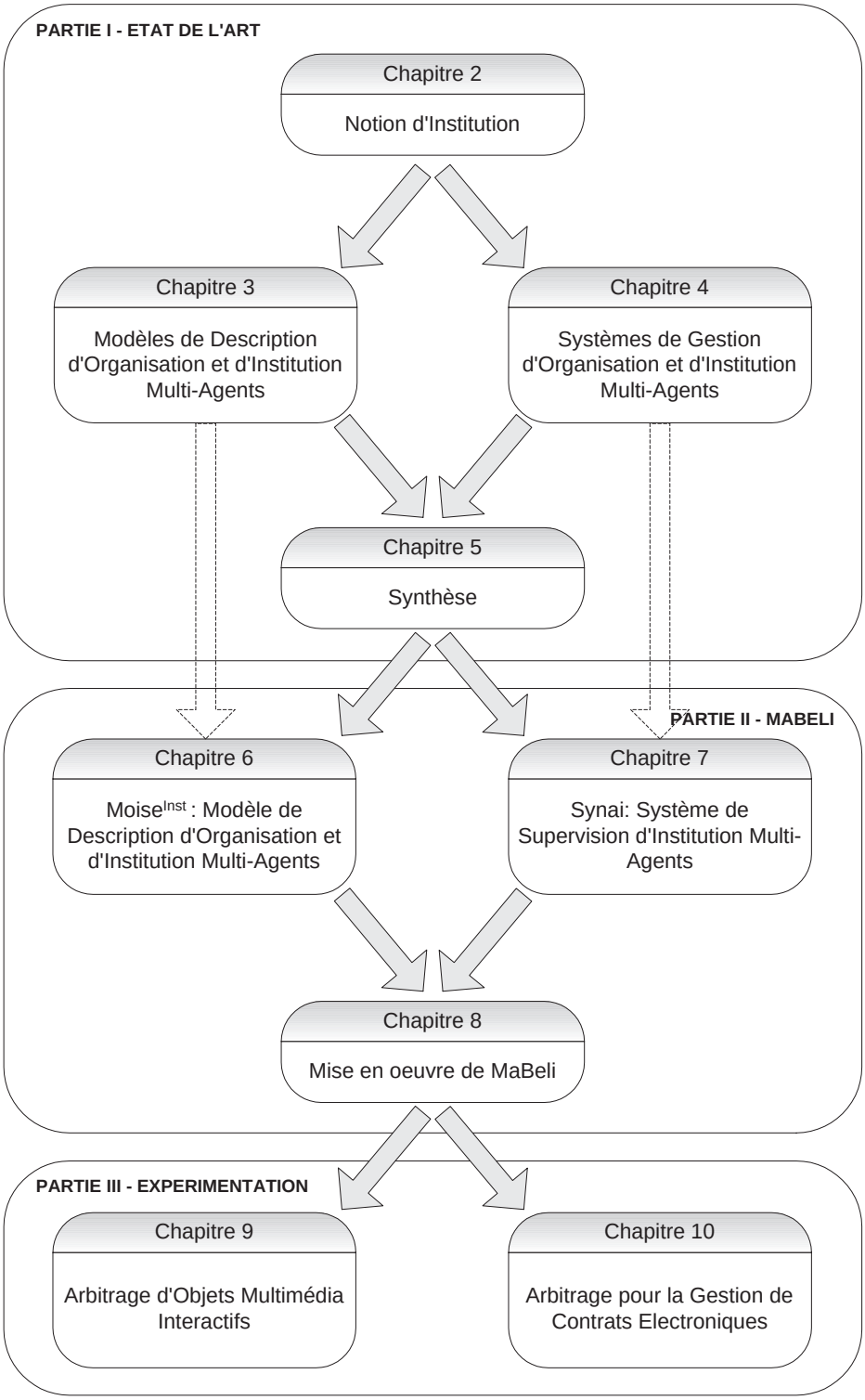


FIG. 1.1 – Organisation du mémoire de thèse

Première partie

Etat de l'art

Notre problématique a fait naître le besoin de contraindre des entités autonomes et de contrôler le respect des règles qui en découlent à l'aide d'un système d'arbitrage construit sur la base d'un Système Multi-Agents. Notre solution devra être générique et pouvoir s'adapter dans tous les domaines d'application. Dans cette première partie, nous faisons donc un état des lieux des concepts et des différents travaux en lien avec les institutions.

Nous commençons par définir notre vocabulaire en regardant dans différents domaines les éléments composant une spécification de contrainte institutionnelle. Puis à l'aide de ce vocabulaire, nous regardons dans le domaine des SMA s'il existe des modèles de spécification d'institutions et de système d'arbitrage d'institutions instanciant les définitions obtenues.

Chapitre 2

Notions d'Institution

Nous venons de voir que le but de nos travaux est de modéliser un système capable de contraindre un ensemble d'entités tout en préservant leur autonomie. Nous considérons dans ce chapitre les types de contraintes pouvant faire l'objet d'un raisonnement sur leur respect et pouvant de ce fait être violées. Nous souhaitons pouvoir définir un ensemble de contraintes ainsi que leur cadre d'exécution afin qu'un non respect puisse apparaître et soit également traité. Pour contraindre le comportement d'entités autonomes que nous nommerons 'acteurs', différentes techniques sont utilisables. Celle que nous considérons ici est l'***institution***.

La définition du Petit Larousse [6] définit une 'institution' comme une "structure régissant la vie des individus, des groupes sociaux ou des États. L'institution peut aussi bien définir l'organe légalement constitué que l'ensemble des règles et des coutumes nécessaires à son fonctionnement". Une institution est également considérée comme "un ensemble de normes qui s'appliquent dans un système social et définissent ce qui, dans ce système, est légitime et ce qui ne l'est pas" [5]. Les termes de structure et de système social font penser à une organisation tandis que les règles[1] sont à la base des normes voire même équivalentes. Ainsi, une institution serait un dispositif ***organisé*** comportant des ***normes***. Voyons également leurs définitions.

La définition globale du terme 'organisation' donnée par le Petit Larousse [6] est "Action d'organiser, de structurer, d'arranger". Le dictionnaire encyclopédique Axis [5] quant à lui décrit une organisation comme étant une "association créée en vue d'un but précis". Enfin, le grand dictionnaire terminologique [1] définit l'organisation comme un "groupement, régi ou non par des institutions, qui se propose des buts déterminés". Il est précisé en plus que le terme 'institution' désigne une structure socialement sanctionnée, qui a valeur officielle, telles les institutions politiques, sociales et religieuses. Nous proposons, en nous basant sur ces trois définitions, la nôtre qui se résume au fait qu'une organisation est "un ensemble d'acteurs jouant des rôles, regroupés au sein d'une structure régulée et tendant à atteindre des objectifs tout en étant à la poursuite d'intérêts personnels".

Le terme de 'norme' quant à lui désigne de manière générale un ensemble de "règles collectives ou communes qui servent de guides ou de standards dans l'orientation de l'action"[1]. Les autres ouvrages donnent globalement la même définition. Dans le domaine légal, le sens accordé permet de considérer qu'une norme représente une expression des droits et des devoirs (obligations et permissions) d'un rôle qu'un individu joue au sein d'une société [99]. Le fait de jouer un rôle au sein d'une société est une façon de structurer la société. Nous rejoignons ainsi la définition des organisations. Les normes et les organisations peuvent donc se compléter afin d'agir sur le comportement d'entités autonomes. Une organisation régie par un ensemble de normes devient alors une institution si les acteurs peuvent être sanctionnés (cf. définition du paragraphe précédent).

[1] Règle : prescription qui s'impose à quelqu'un dans un cas donné ; principe de conduite, loi [6]

Les définitions que nous venons de voir sont applicables dans un cadre général et de ce fait ne sont pas satisfaisantes. Toutes ces définitions ont été étudiées et instanciées dans des domaines aussi différents que variés. Dans ce chapitre, nous étudions donc les différentes formes que les institutions, les organisations et les normes peuvent avoir selon leur domaine d'application. Nous commençons tout d'abord par le domaine de l'économie et des politiques économiques qui y sont définies par des organisations et pour des organisations. Ensuite, nous aborderons les domaines du droit constitutionnel et de la sociologie contraignant des groupes d'individus à l'aide de normes légales ou sociales. Enfin nous terminerons par le domaine informatique formalisant et appliquant les notions des domaines précédents pour la définition et le management des droits d'accès. Nous déduirons dans la synthèse de ce chapitre notre définition d'une Institution Électronique adaptée à notre étude.

2.1 Économie

La définition de [1] de l'économie politique et sociale est l'"ensemble des règles, de principes, d'éléments destinés à uniformiser des méthodes ou des moyens d'action et à guider dans l'application d'une politique quelconque". Nous retrouvons la notion de règle. Dans ce domaine, le but est de définir des règles pour guider l'action des acteurs du monde économique.

2.1.1 Analyse économique des institutions

La définition de la politique économique étant trop restrictive, elle est élargie et donne naissance au début des années 60 à l'Analyse Économique des Institutions (AEI)[2]. Elle étudie entre autres les institutions, les comportements des hommes et les marchés politiques. Plusieurs voies de recherche relatives à l'analyse économique des institutions se sont alors développées : la nouvelle histoire économique, l'analyse économique des structures sociales et l'analyse économique de la politique. Douglass North faisait alors partie de la première voie. Depuis, il a fondé la société internationale pour la nouvelle économie institutionnelle.

Douglass C. North, Prix Nobel d'économie en 1993, donne en 1990 dans [76] sa définition d'une institution. Pour lui une institution est la définition des règles du jeu d'une société par n'importe quel moyen de contrainte. En 1991 dans [77], il précise que les institutions sont les contraintes mises en place par les humains pour structurer les interactions politiques, économiques et sociales. Elles consistent en un ensemble de contraintes informelles (sanctions, tabous, traditions, coutumes, codes de conduite) et de règles formelles (constitutions, lois, droits de propriété).

Appliquées au domaine économique, les contraintes d'une institution peuvent être considérées et regroupées par exemple en politiques budgétaire, monétaire, de change, de croissance ou fiscale. Une politique économique a également des objectifs qui sont l'allocation, la stabilisation et la redistribution. Ainsi, afin d'atteindre un ensemble d'objectifs (les buts), un ensemble de décideurs (une organisation structurée) va contraindre un ensemble d'acteurs à respecter les politiques budgétaires, monétaires, etc. (les règles) qu'il aura mises en place sous peine de sanctions (l'institution).

2.1.2 Acteurs des institutions économiques

Les acteurs de la politique économique sont multiples. Il y a ceux qui subissent et qui doivent se conformer à ces politiques, les personnes physiques ou morales, et il y a les acteurs qui mettent en place ces politiques, ce sont les gouvernements, les banques centrales, les organisations internationales, etc. Enfin la gouvernance mondiale peut être elle aussi considérée comme une institution sauf que les acteurs qui devront se conformer aux contraintes de manière globale sont également ceux qui les mettent en place de manière locale. Les exemples sont le FMI (Fonds Monétaire International), l'OMC (l'Organisation Mondiale du Commerce), l'ONU (l'Organisation des Nations Unies), les ONG (Organisation Non Gouvernementale), etc.

[2] http ://fr.wikipedia.org/wiki/Analyse_%C3%A9conomique_des_institutions

Prenons le cas de l'OMC. C'est une organisation internationale s'occupant des règles régissant le commerce entre pays. Au cœur de l'organisation se trouvent les Accords de l'OMC, négociés et signés par la majeure partie des puissances commerciales du monde et ratifiés par leurs parlements. Le but est d'aider, par la réduction d'obstacles au libre-échange, les producteurs de marchandises et de services, les exportateurs et les importateurs à mener leurs activités. Ce sont les membres eux-mêmes qui font respecter les règles conformément aux procédures convenues qu'ils ont négociées, y compris en recourant à des sanctions commerciales. Ces sanctions sont imposées par les pays et sont autorisées par l'ensemble des membres [3].

Dans ce domaine, la définition d'institution que nous retenons est celle-ci : Une institution est capable de définir des règles de comportement d'un ensemble d'acteurs grâce à n'importe quel moyen de contrainte tout en supervisant leur respect via également un ensemble d'acteurs. Ainsi, par exemple les organisations telles le FMI ou l'OMC sont des institutions mettant en place des politiques économiques. Leurs membres sont des pays négociant et acceptant les accords spécifiant un ensemble de règles à respecter. Ils se supervisent eux-mêmes et s'appliquent des sanctions le cas échéant. Maintenant que nous avons une définition générale d'une institution, entrons dans les détails des organisations et des règles en étudiant le droit constitutionnel appliqué à chaque pays.

2.2 Droit

Tout citoyen est contraint de respecter les lois d'un État sous peine de se faire sanctionner par les acteurs du respect des lois. Dans ce contexte, l'institution est ce que nous pourrions appeler la justice. Elle est composée des lois, correspondant aux normes et des personnes subissant et/ou faisant respecter ces lois, c'est-à-dire les acteurs.

2.2.1 Justice

L'institution désigne ici l'ensemble des structures résultant du régime politique. Ces structures sont spécifiées entre autres par la Constitution, les lois et les règlements.

Les lois sont des règlements édictés par des corps spécialisés (parlement, juge, ...). Les normes juridiques constituent le droit c'est-à-dire l'ensemble des règles officielles qui fixent les façons de se conduire jugées légitimes à un moment donné. Le droit indique ce qui est légal. Il faut distinguer ce qui est légal de ce qui est normal, faisant plutôt référence aux normes sociales que nous verrons dans le chapitre suivant. Ces normes ont un caractère impératif dans le sens où le non respect de la loi entraîne une sanction appliquée par des instances spécialisées (police, justice, ...). Nous pouvons considérer que l'État est l'institution qui a le monopole de la contrainte organisée.

Le système juridique est organisé, pour tous les États, selon la théorie dite de la pyramide des normes de Hans Kelsen [64]. Ces normes désignent l'ensemble des règles obligatoires édictées par les autorités publiques. Dans l'ordre décroissant nous trouvons : la Constitution, les traités internationaux, les lois, les ordonnances, les décrets, les règlements et les arrêtés (ministériels, préfectoraux, ou communaux). Les normes inférieures doivent ainsi être en accord avec ce qui est défini par la norme supérieure. La forme définitive d'une loi est un texte législatif publié, dans le cas de la France, au "Journal Officiel de la République Française" (édition Lois et décrets).

2.2.2 Acteurs des institutions judiciaires

Afin qu'une loi soit authentifiée et exécutoire, il faut que le Chef de l'État la promulgue. Au préalable, une loi doit être rédigée et proposée. En France, une loi est d'initiative gouvernementale alors qu'une proposition de loi est d'initiative parlementaire. La loi est ensuite discutée entre les deux assemblées pour parvenir à un accord sur le fond et la forme du texte législatif. En l'absence de consensus, c'est l'Assemblée Nationale qui tranche.

Une fois que la loi est adoptée et promulguée par le Chef de l'État, elle entre en vigueur. Toute personne physique ou morale doit alors s'y conformer et agir en respect de la loi. Chaque personne interprète elle-même la loi ou fait appel à un juriste. En cas de non-respect d'une loi le fautif doit en répondre à la justice. L'organisation juridictionnelle nationale française est l'organisation des tribunaux nationaux français, dans l'ordre juridique interne. Les juridictions internationales, européennes ou communautaires, qui résident dans un ordre juridique externe, émanent d'autres organisations internationales (Union Européenne, Conseil de l'Europe, Nations unies, ...).

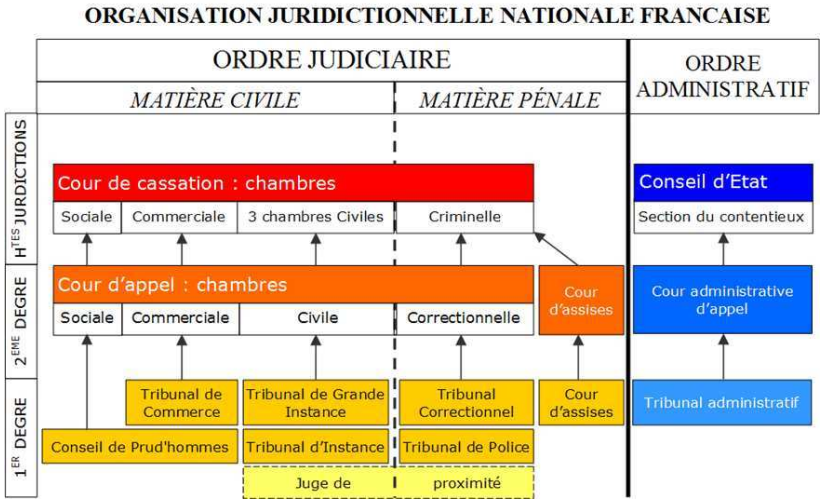


FIG. 2.1 – Organisation Juridictionnelle Nationale en France (provenant de [4])

L'organisation juridictionnelle est représentée sur la FIG. 2.1. Nous y retrouvons les catégorisations des affaires juridiques que cette organisation peut avoir à traiter ainsi que les niveaux de gravité ou d'importance. Cette organisation fait intervenir différents acteurs qui auront à jouer un rôle. Ce sont les métiers de l'organisation judiciaire qui sont, entre autres, les Avocats, les Juges, les Greffiers, le Procureur de la République, les Magistrats, etc. Les individus jouant ces rôles doivent également en tant que citoyens respecter ces règles. Nul n'est au dessus des lois. En tant qu'acteurs du respect des lois et suivant leur métier, un contrat stipule les règles qu'ils doivent respecter. Un contrat est une autre forme de contrainte légale faisant intervenir des contractants qui doivent respecter les contraintes et faisant intervenir des acteurs de la justice devant émettre un jugement en cas de conflit sur un contrat. Un conflit étant une différence de point de vue sur le respect du contrat ou d'une partie.

Dans le domaine du droit constitutionnel, les acteurs du respect des lois sont eux-mêmes soumis à ces mêmes lois. Nous illustrons donc ici la notion de rôle que nous avons utilisée dans notre définition d'une organisation. En effet, suivant le rôle que l'individu jouera, son comportement vis-à-vis des lois sera différent. En tant que personne juridique il aura pour but de faire respecter la loi, ceci en suivant certaines règles liées à son métier. Mais en tant que citoyen, son but sera personnel, et il devra l'atteindre en respectant ces mêmes lois qu'il doit contrôler.

Maintenant que nous avons vu comment les acteurs économiques (allant du contribuable à l'entreprise) étaient contraints de respecter certaines politiques économiques, puis comment plus précisément les citoyens étaient contraints de respecter les lois de leur pays, diminuons encore plus la portée des normes pour voir comment un ensemble de règles sociales peuvent influencer le comportement d'un être humain.

2.3 Sociologie

Les contraintes légales que nous venons de voir sont à mettre en opposition aux contraintes sociales que nous allons aborder dans cette section. Dans son sens sociologique le plus général, une institution désigne une structure sociale (ou un système de relations sociales) dotée d'une certaine stabilité dans le temps. Une définition plus élégante consiste à dire qu'une institution est une règle du jeu acceptée socialement [4]. Nous nous rapprochons de la définition de North donnée dans [76]. Le Grand Dictionnaire Terminologique [1] définit quant à lui le concept d'institution dans le domaine sociologique comme une "norme, coutume ou pratique socialement sanctionnée, établie dans une société donnée, qui revêt habituellement une valeur officielle ou légale". Il décrit le concept de norme comme un ensemble de "règles collectives ou communes qui servent de guides ou de standards dans l'orientation de l'action". Nous pouvons comparer de manière caricaturale la norme sociale à une coutume par rapport à la norme légale qui est une loi sanctionnable en cas de non respect. Nous ne pouvons à proprement parler d'institution dans le cadre social. La violation des normes n'entraîne pas de sanction.

2.3.1 Institutions et Normes sociales

Dans le domaine de la théorie sociale, Raimo Tuomela distingue dans [97] les règles sociales et les normes sociales. Il fait une classification en quatre groupes :

- *r-norms* : règles qui sont respectées à partir du moment où elles sont acceptées ;
- *s-norms* : normes sociales qui sont suivies à partir du moment où les "autres" attendent qu'elles soient respectées ;
- *m-norms* : normes morales qui sont respectées par bonne "conscience" ;
- *p-norms* : normes de prudence qui sont suivies parce qu'elles correspondent à un comportement rationnel.

A la différence des normes juridiques, il s'agit ici de comportements prescrits par la société. Le rôle de la socialisation est ici essentiel. En faisant partie d'un groupe ou d'une société, l'individu est soumis à un ensemble d'influences. Ces comportements sont intériorisés par les individus, c'est-à-dire que les normes influencent plus la personne de manière individuelle qu'un groupe ou une société. Elles font parties de la personne (en pouvant toutefois être mises à jour) au même titre qu'un ensemble de désirs ou d'intentions. Ces règles s'appuient en effet sur un ensemble de croyances et un mécanisme de raisonnement afin d'agir par tradition ou par coutume, par bonne conscience ou par prudence. Ces règles sont également transférées entre individus afin de procéder à une socialisation.

2.3.2 Acteurs sociaux

Ces normes, traditions et autres coutumes sont liées aux valeurs d'un groupe social ou d'une société qui les rendent compréhensibles. Chaque individu appartenant à un groupe ou à une société sera influencé par ces normes. Pourtant, nous trouvant dans le domaine de la sociologie, ces normes n'ont pas de cadre légal. Donc même si les normes peuvent être violées, elles ne peuvent pas être sanctionnées. Alors pourquoi un individu va accepter et respecter une norme ? La différence ne réside pas dans la définition des normes mais dans leur dimension mentale. Une norme sociale est acceptée et réprimandée collectivement.

Ainsi les normes sociales relèvent surtout d'une pression collective. Il n'est pas question de sanction ou de punition mais de concept moins formels tels que la réputation ou la confiance. Si une personne ne respecte pas les contraintes qui ont cours dans son environnement social, alors elle s'expose à une baisse de sa réputation ou une confiance amoindrie. Chaque individu, ayant accepté personnellement les normes du groupe auquel il appartient, définira la réputation et la confiance qu'il a envers les autres individus sur base des normes en cours. Mais lorsque les systèmes de valeurs de référence changent, des conflits sur les normes sociales apparaissent. La socialisation (ou intégration) permet

d'adapter les contraintes internes afin d'être en cohérence avec l'environnement social (phénomène de conformisme ou de déviance).

En résumé, les normes sociales sont des contraintes individuelles comparables à des croyances mises en application dans un certain contexte. Les normes sociales influençant individuellement les personnes, c'est individuellement que chaque membre de la société effectuera son propre contrôle. Le résultat de leur contrôle leur fera mettre à jour la confiance et la réputation qu'ils ont envers les autres membres.

A travers les trois domaines que nous venons d'étudier, nous avons observé qu'une institution constituait un ensemble de règles sur un ensemble d'acteurs dont le respect est également supervisé par un ensemble d'acteurs. Pour cela, des rôles permettent aux acteurs de préciser ce qui les influence en fonction du contexte dans lequel ils se trouvent. Nous plaçant maintenant dans le domaine informatique et plus particulièrement dans celui du contrôle d'accès, nous allons voir comment ces concepts sont formalisés.

2.4 Politique de sécurité des systèmes informatiques

Certains des systèmes informatiques avec lesquels nous interagissons tous les jours engendrent des droits d'accès. Pour cela, notre identité est spécifiée et utilisée pour définir nos droits face à l'outil informatique. Par rapport aux expressions des contraintes vues précédemment, celles-ci sont interprétées et contrôlées par un logiciel. De ce fait, elles ne peuvent pas être exprimées sous forme de texte de lois comme pour la politique économique ou le droit constitutionnel ou sous forme de normes sociales individuelles et internes puisque ce n'est pas l'ordinateur qui est contraint mais l'utilisateur.

L'utilisateur est contraint via une représentation que l'outil informatique se fait de son identité. Les permissions faisant intervenir ces identités sont explicites et exprimées formellement afin de pouvoir être prises en compte par l'ordinateur. Ainsi, cette expression de la sécurité d'un système informatique permet de gérer les accès aux informations. Les contrôles d'accès peuvent se faire sur les réseaux, les systèmes d'exploitation ou encore les applications.

2.4.1 Modèles de contrôle d'accès

Respectant la définition que nous avons donnée d'une institution, les modèles de contrôle d'accès permettent de gérer les accès des utilisateurs (les acteurs) à différentes ressources informatiques en fonction du groupe auquel ils appartiennent (organisation) grâce à des politiques de sécurité (les normes). Le modèle I-BAC (*Identity Based Access Control*) [68] est le premier modèle de contrôle d'accès proposé dans la littérature. Ce modèle introduit les concepts fondamentaux de sujet, d'action et d'objet. Ce modèle introduit également le concept de politique d'autorisation. Dans I-BAC une politique d'autorisation correspond à l'expression d'un ensemble d'autorisations positives (ou permissions) spécifiant qu'un *Sujet* a la *Permission* de réaliser une *Action* sur un *Objet* et ayant le format suivant :

$$Permission(Sujet, Action, Objet)$$

- Le *Sujet* est l'utilisateur agissant directement dans un système ou via des processus travaillant pour son compte.
- L'*Objet* représente une entité passive du système informatique et contient les informations à protéger ; il est accessible par le *Sujet* pour réaliser une *Action*.
- L'*Action* permet au *Sujet* de manipuler un *Objet*.

Le modèle RBAC (*Role Based Access Control* [89, 47, 43]) ou Contrôle d'accès à base de rôles propose de structurer l'expression de la politique d'autorisation autour du concept de rôle. La Fig. 2.2 représente en UML le méta-modèle RBAC. Un rôle est un concept organisationnel : des rôles sont

affectés aux utilisateurs conformément à la fonction que ces utilisateurs jouent dans l'organisation. Les autorisations sont directement associées aux rôles. Un rôle peut concerner plusieurs utilisateurs et peut hériter d'un autre rôle.

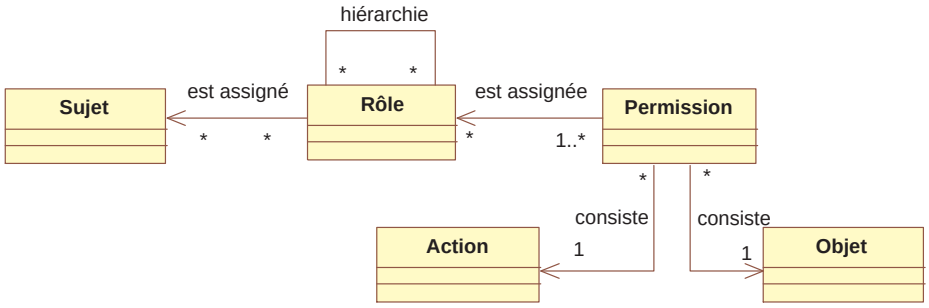


FIG. 2.2 – Méta-modèle des permissions d'une politique RBAC

Il existe également les modèles de contrôle de flux (ou MAC pour *Mandatory Access Control*) [11], et les modèles de contrôle d'accès à base de règles (*Rule Based Access Control*) [61, 12, 51]. Ces modèles n'offrent cependant pas la possibilité d'exprimer des règles contextuelles (applicables seulement dans un certain contexte) relatives aux permissions, aux interdictions et aux obligations. Ce type de règle est particulièrement utile pour exprimer des politiques de sécurité dans certains domaines. Le modèle appelé Organisation Based Access Control (Or-BAC) [63, 73] et s'appuyant sur un langage formel basé sur la logique du premier ordre, permet de spécifier de telles politiques de sécurité contextuelles.

2.4.2 Acteurs et système de contrôle d'accès

Les modèles de contrôle d'accès peuvent être administrés. L'administration consiste en la création et la maintenance des éléments constituant la politique de sécurité tels que les utilisateurs, les actions, les objets, les rôles, les droits, etc. Comme l'ordinateur ne peut pas de lui-même créer de nouvelles règles, l'utilisateur peut en plus d'être contraint par le système, l'administrer. Cela dépend bien entendu de ses droits. Les éléments de la politique étant des ressources informatiques comme les autres, seuls les utilisateurs autorisés peuvent y avoir accès.

L'administration du modèle Id-BAC est souvent de type discrétionnaire (modèle DAC pour *Discretionary Access Control*). Elle repose sur la notion de propriétaire : chaque objet a un propriétaire qui décide quels sont les sujets qui ont accès à cet objet. Il est souvent le créateur de l'objet et dispose de tous les droits sur cet objet. Ce plein contrôle dont le propriétaire dispose sur ses ressources, lui donne aussi la possibilité de déléguer à un autre sujet le droit d'accorder des droits sur certaines d'entre-elles. Le modèle RBAC est quant à lui contrôlable et administrable grâce au modèle ARBAC (*Administrative Role-Based Access Control*) tandis que le modèle Or-BAC l'est par le modèle AdOr-BAC [21, 73].

L'exécution de la politique consistant à donner l'accès ou non à la ressource est mise en œuvre automatiquement par le système de contrôle. Si nous reprenons le méta-modèle du modèle R-BAC, pour qu'un utilisateur (`Sujet`) puisse accéder en lecture (`Action`) à une ressource informatique (`Objet`), il faut qu'une permission (`Permission`) liée à un rôle (`Rôle`) que l'utilisateur jouerait soit également liée à l'action et à la ressource. Dans le cas contraire, le système de contrôle d'accès empêchera l'utilisateur de faire l'action sur l'objet. Il s'agit ici de contrôle, de ce fait, aucune notion de sanction n'existe.

Afin d'illustrer le fonctionnement du contrôle d'accès de type Id-BAC, nous prenons l'exemple du modèle XACML (*eXtensible Access Control Markup Language*) [2]. XACML est un langage de contrôle d'accès basé sur XML et standardisé par OASIS [72]. XACML est à la fois un langage pour

écrire des polices de droit d'accès et un langage de demande/réponse. La FIG. 2.3 représente le fonctionnement du contrôle automatique par un système informatique de l'accès d'un utilisateur à une ressource. En plus de synthétiser sur un cas d'utilisation tous les concepts que nous avons vus jusqu'à maintenant, il introduit la façon dont le contrôle du respect des règles est fait.

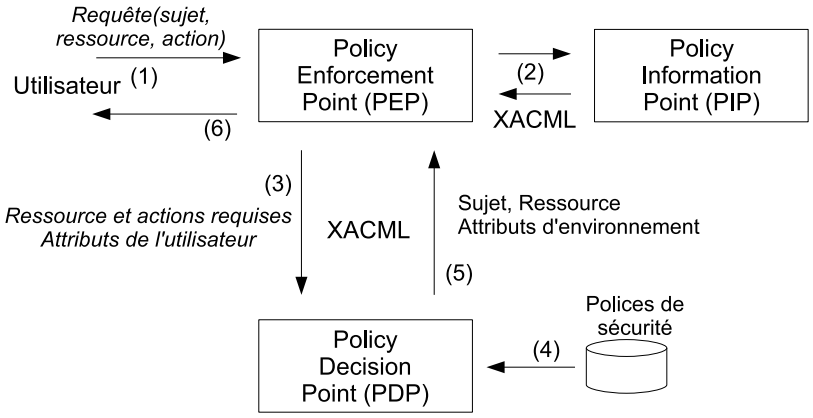


FIG. 2.3 – Principe de contrôle simplifié du modèle XACML

Ainsi, dans un scénario typique d'utilisation de XACML et afin de faire le parallèle avec les concepts vus jusqu'à maintenant, un ***acteur*** (l'utilisateur) veut mettre en place une action (composant le ***but*** pour lequel le système informatique et l'***organisation*** d'utilisateurs ont été créés) sur une ressource. L'utilisateur fait donc sa requête au *Policy Enforcement Point* ou PEP (1) qui en fonction du PIP (2) va la transformer en une requête XACML qu'il envoie au *Policy Decision Point* ou PDP (3). L'association PEP+PDP est considérée comme le ***système de contrôle***. Le PDP, en fonction des polices de sécurité existantes (4) va créer ou non une permission autorisant le sujet à agir sur la ressource (5) permettant au PEP de donner l'accès ou non à la ressource (6).

Dans le domaine de la sécurité des systèmes informatiques, de nombreux modèles existent. Ils se basent sur une représentation de l'utilisateur (identité, rôle, membre d'une organisation) afin de lui permettre ou non d'agir sur une ressource informatique. Le système de contrôle se place entre l'utilisateur et les ressources et prend en compte les permissions pour accepter ou rejeter les requêtes des utilisateurs. Nous situant dans un domaine informatique, les permissions sont formalisées pour être interprétées par le système de contrôle. Cette modélisation est intéressante par le côté structurel des permissions s'approchant de la spécification déclarative des normes que nous souhaitons. Cependant, ce domaine touchant celui de la sécurité informatique, il s'en trouve très fermé et réellement contrôlé ne laissant ainsi aucune place à la supervision et l'arbitrage.

2.5 Synthèse

Nous avons abordé dans ce chapitre plusieurs domaines différents dans lesquels la notion de règles pouvant influencer le comportement d'un ensemble d'individus était présente. Chaque domaine nous a apporté ses définitions et ses concepts :

- Ainsi la définition de [5] prétendant qu'une ***Organisation*** est une association créée dans un but précis a été confirmé par l'utilisation des organisations dans les domaines de l'économie et du droit.
- Nous avons vu que dans le domaine du droit, l'***Organisation*** est constituée d'un ensemble d'***Acteurs*** jouant des ***Rôles***.
- Le domaine de la sécurité informatique nous a fait faire le parallèle entre les normes et les permissions nous permettant de définir une ***Norme*** comme une ***permission*** accordée à un

Rôle d'exécuter une ***Action***.
- Nous avons également vu dans le domaine de la sociologie que les ***Acteurs*** peuvent jouer des ***Rôles*** différents et de ce fait être influencés selon les ***Normes*** les concernant. Nous établissons alors des ***Contextes*** distincts qui prennent place au sein des ***Organisations***.

Nous retenons de ceci la définition d'une ***Institution*** faite par North [76] :

> Une ***Institution*** définit l'ensemble des règles du jeu d'une société à l'aide de n'importe quel moyen de contrainte.

Le moyen de contrainte que nous considérons ici est une ***Organisation*** et la définition que nous en faisons est la suivante :

> Une ***Organisation*** est un ensemble d'acteurs jouant des rôles, regroupés au sein d'une structure régulée et tendant à atteindre des objectifs tout en étant à la poursuite d'intérêts personnels.

Notre domaine d'application étant l'informatique, nous considérons l'***Institution Électronique*** comme moyen d'influencer le comportement d'un ensemble d'entités logicielles. Nous la définissons comme suit :

> Une ***Institution Électronique*** est une Organisation d'acteurs autonomes au sein de laquelle leur comportement est influencé par des Normes supervisées par un système d'Arbitrage.

Il est à noter ici que la définition que nous retenons d'une Institution Électronique diffère de ce qui existe dans la littérature (cf. Section 3.2) et notamment de la définition de ISLANDER considérant une Institution Électronique comme un environnement virtuel régulé au sein duquel un ensemble d'interactions entre les participants a lieu [7]. ISLANDER est orienté 'interactions' alors que nous sommes orientés 'organisations'.

Nous considérons donc une ***Institution Électronique*** comme étant composée de trois éléments principaux :
- un Modèle de Spécification d'Institution (***SIM*** pour *Specification Institution Model*) regroupant les ***Normes*** liées à l'***Organisation*** ;
- un Système d'Arbitrage (***IAS*** pour *Institution Arbitration System*) vérifiant le respect du ***SIM*** et mettant en place les sanctions ;
- les Acteurs (***AAI*** pour *Autonomous Actors within Institution*) évoluant au sein de l'organisation et dont le comportement est influencé par un ensemble de normes (donc soumis au ***SIM***) et arbitrés et sanctionnés par le ***IAS***.

Nous en déduisons le schéma de la FIG. 2.4. Les liens entre l'***IAS*** et les deux autres éléments sont moins explicites et consensuels dans les différents domaines d'application. En politique économique, ce sont les membres de l'organisation qui définissent les normes qu'ils doivent respecter et ce sont eux qui se supervisent et, le cas échéant, se sanctionnent. En droit constitutionnel, certains acteurs définissent les normes et d'autres les supervisent mais tous doivent les respecter sous peine d'être sanctionnés. En sociologie, les normes sont définies collectivement et supervisées individuellement sans entraîner de sanction. Enfin, en sécurité informatique, les normes sont définies par des acteurs, elles doivent être respectées par l'ensemble des acteurs et elles sont contrôlées par un système indépendant. Nous constatons une différence entre les normes contrôlées et les normes supervisées. Les normes contrôlées nécessitent l'accord du ***IAS*** avant que l'***Action*** concernée puisse être exécutée par le ***AAI***. La supervision donne la possibilité de violer une norme et implique l'existence de mécanismes de détection et de sanction. Nous nous situons à la limite de ces deux principes en définissant notre ***Système***

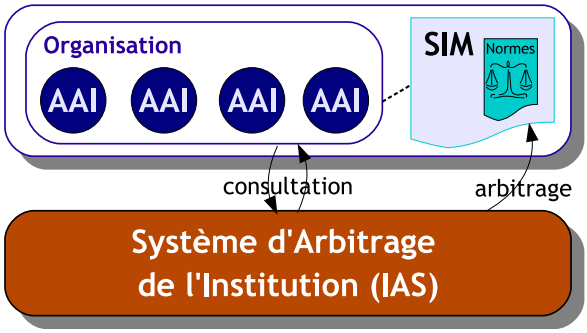


FIG. 2.4 – Composants d'une Institution Électronique

d'Arbitrage d'Institution comme étant un système qu'il faut consulter pour agir mais ne bloquant pas les actions non autorisées. Nous entendons alors par 'arbitrage' le fait de contrôler le respect des normes et d'appliquer des sanctions le cas échéant (sanctions pouvant être positives ou négatives).

Considérons maintenant que les ***AAI*** à normer et à arbitrer sont les agents d'un SMA. Nous allons voir dans le chapitre suivant un aperçu de l'existant dans le domaine des SMA en matière d'organisation et de normes. Puis, dans le Chapitre 4 nous verrons les systèmes permettant de gérer, de mettre en œuvre et de superviser les spécifications d'organisation et de normes.

Chapitre 3

Modèles de Description d'Organisation et d'Institution Multi-Agents

Dans ce chapitre, nous allons voir les moyens existant dans le domaine des Systèmes Multi-Agents permettant de guider et d'influencer le comportement des agents. En effet, au lieu de définir des règles pour les humains sous forme de lois, de normes sociales ou juridiques ou de droit d'accès, nous voulons pouvoir dicter le comportement des agents autonomes capables de mettre en place des mécanismes de décision au sein d'une société. Nous avons vu dans le chapitre précédent que le SIM, la Spécification du Modèle d'Institution, était composée d'une Organisation et d'un ensemble de Normes. De plus, pour qu'une Organisation ait un but, que les Acteurs y jouent un rôle et qu'elle soit liée à la définition des Normes, les Organisations sont composées de différentes dimensions de modélisation.

Ce chapitre se compose d'une étude des Modèles d'Organisation et des Modèles d'Institution. Pour chaque modèle de contrainte, en plus de définir leurs différentes dimensions de modélisation, nous verrons le niveau de description des contraintes (pouvant aller d'une description abstraite à l'implémentation) et leur portée c'est-à-dire le nombre d'agents pouvant être influencés.

3.1 Modèles d'Organisation

Les modèles organisationnels spécifient la structure et les fonctionnalités d'une société d'agents, c'est-à-dire un ensemble d'agents évoluant au sein du même environnement. Nous allons voir différentes façons d'exprimer les contraintes que nous voulons imposer aux agents avec une organisation. Ces modèles sont plus ou moins orientés autour de la spécification de la fonctionnalité du système et/ou de la structure de l'ensemble d'agents. Nous abordons d'abord les systèmes exclusivement orientés organisation structurelle pour terminer par les spécifications fonctionnelles permettant même la coordination entre agents.

3.1.1 Modèles d'analyse au sein de GAIA

GAIA [107, 108, 110] est une méthodologie pour l'analyse et la conception de systèmes orientés-agent. GAIA modélise un SMA en une organisation informatisable composée de rôles interagissant les uns avec les autres. Pour cela, un ensemble de modèles basés sur des expressions d'exigences permet d'obtenir une analyse d'un niveau d'abstraction assez bas. Ainsi une technique de conception plus traditionnelle (orientée objet par exemple) peut être appliquée pour implémenter les agents. Ces modèles sont représentés sur la Fig. 3.1. Ceux qui nous intéressent ici sont celui de rôle, composé d'un ensemble de ***rôles*** et celui d'interaction, composé d'un ensemble de ***protocoles***, tous deux composant

le modèle d'organisation.

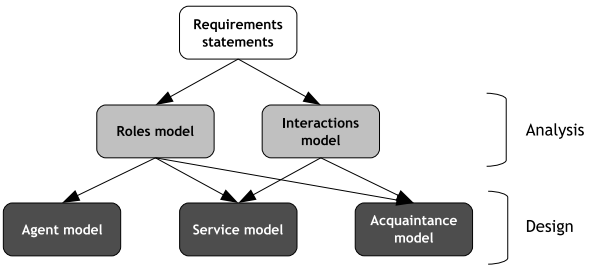


FIG. 3.1 – Liens entre les modèles de la méthodologie GAIA (découlant de [108])

Un ***rôle*** est défini par quatre attributs :
- Les *responsabilités* déterminent la fonctionnalité des agents.
- Pour remplir ses responsabilités, un rôle possède un ensemble de *permissions*. Les *permissions* limitent les ressources auxquelles le rôle peut avoir accès. Généralement ces ressources sont des informations que le rôle peut lire, écrire ou créer. Les *permissions* spécifient donc ce que peut et ce que ne peut pas utiliser le rôle.
- Les *activités* sont des tâches ou des actions qu'un rôle peut exécuter sans qu'il interagisse avec d'autres rôles.
- Les *protocoles* sont des tâches ou des actions qu'un rôle peut exécuter et qui impliquent d'autres rôles. Les *protocoles* définissent la façon dont un rôle peut interagir avec les autres rôles.

Le modèle d'interactions complète le modèle de rôles. En effet, il contient une définition de protocole pour chaque ***protocole*** de chaque rôle dans le système. Les protocoles définissent toutes les interactions inter-rôles. Plus précisément, ils décrivent brièvement la nature de l'interaction (son but) en ignorant les détails d'implémentation ainsi que les séquences de messages échangés. La définition des protocoles représente également le rôle initiateur (celui qui démarre l'interaction), le rôle répondeur (celui avec lequel l'initiateur interagit), des informations d'entrée (utilisées par l'initiateur durant l'exécution du protocole) et de sortie (fournies par le répondeur ou pour le répondeur durant le déroulement de l'interaction) et enfin une brève description du traitement que l'initiateur effectue pendant l'exécution de ce protocole.

Comme illustré sur la FIG. 3.1), ces deux modèles d'analyse sont ensuite raffinés en modèles moins abstraits tels que le modèle agent (les types d'agent qui seront utilisés dans le système), le modèle de services (les fonctionnalités exhibées par les agents) et le modèle d'accointances (les liens de communication entre les agents).

Dans [109], les auteurs de GAIA proposent, en plus des modèles d'origine, un modèle de coordination se situant au même titre que les modèles de rôles et d'interactions au niveau analyse de la méthodologie. Ce modèle de coordination prend en compte la façon d'exprimer la coordination entre rôles au travers d'un medium spécifique. Il définit ainsi des lois sociales afin de diriger toutes les communications à travers le medium modélisé.

Le modèle ROADMAP [62] étend également la méthodologie GAIA avec de nouveaux modèles d'analyse. Nous avons ainsi les modèles de cas d'utilisation, d'environnement, de connaissance, de protocoles et d'interactions. Le modèle de rôles est toujours présent mais permet de définir une hiérarchie de rôles aboutissant sur une structure plutôt que sur une société d'agents.

Nous avons affaire ici à une méthodologie utilisant un modèle organisationnel pour faire l'analyse du système à agents à concevoir. Cependant et contrairement à ce que nous allons voir dans les

sections suivantes, le modèle défini de manière explicite lors de l'analyse n'est plus disponible lors de l'implémentation. En effet, les fonctionnalités ainsi que les protocoles d'interaction étant liés aux rôles dans la partie analyse et donc aux agents, les contraintes de comportement sont ensuite encodées au sein des agents ce qui leur permet de prendre en compte l'organisation lors de l'exécution mais pas de pouvoir raisonner dessus. Cela réduit la flexibilité du modèle, obligeant la re-programmation du système afin de modifier l'organisation.

3.1.2 Modèle Agent-Groupe-Rôle

Le modèle organisationnel AGR pour *Agent*, *Groupe* et *Rôle* [41] est une évolution du modèle Aalaadin) [40] : la modélisation d'un ensemble d'Agents jouant des Rôles dans des Groupes. Un modèle AGR est séparé en une représentation de niveau Agent instanciant et une représentation de niveau organisationnel. Ces deux niveaux sont représentés sur la Fig. 3.2 définissant le méta-modèle de AGR. Nous constatons que le niveau organisationnel permet de spécifier des structures de groupe (`Group Structure`), des Rôles (`Role`), des contraintes entre Rôles (`Constraint`) et des interactions entre rôles (`Interaction`). Le niveau Agent permet d'instancier les Groupes et d'y associer des ensembles d'Agents y jouant des Rôles.

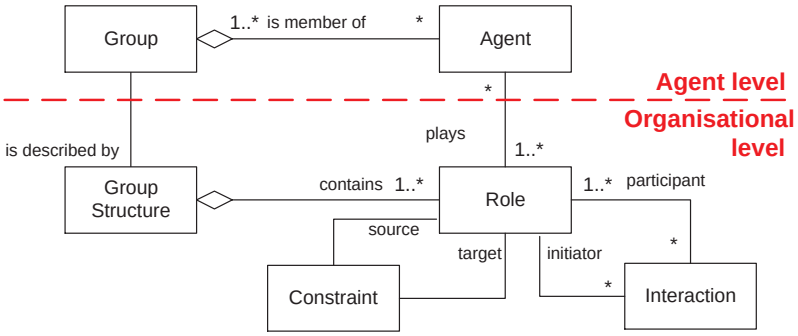
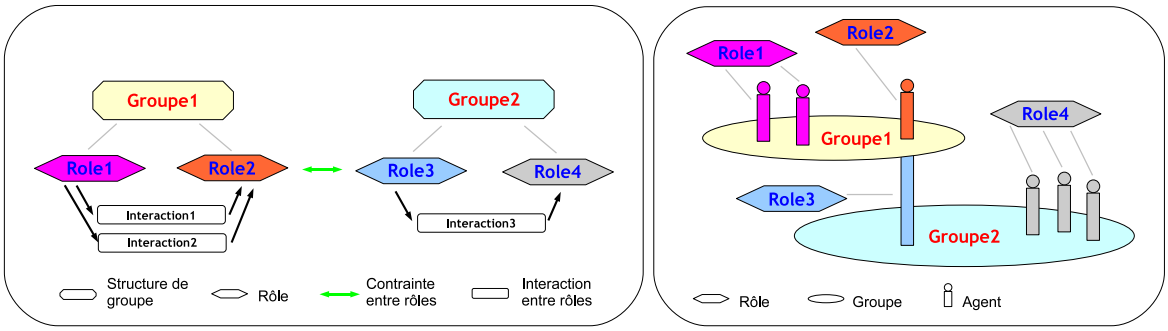


Fig. 3.2 – Méta-modèle de AGR (découlant de [41])

La ***Structure d'un Groupe*** est une description abstraite d'un Groupe composée de Rôles. Les ***Rôles*** sont des descriptions *a priori* du comportement que les Agents auront. Les Rôles décrivent également les contraintes que les Agents jouant ces Rôles doivent satisfaire. Ainsi une Structure de Groupe permet de diviser l'organisation en contextes d'activités communes (activités de défense pour une équipe de football par exemple). Les Rôles sont locaux aux Structures de Groupe. Ils sont associés entre eux au travers de description d'interactions et de relation de dépendance (le même Agent devra jouer ces deux Rôles). La Fig. 3.3(a) donne un exemple de représentation graphique du niveau organisationnel d'un modèle AGR. Nous y retrouvons deux Groupes *Groupe1* et *Groupe2* constitués respectivement des Rôles *Role1*, *Role2* et *Role3*, *Role4*. Deux relations d'interactions permettent aux Rôles *Role1* et *Role2* d'interagir selon un protocole bien particulier de même pour les Rôles *Role3* et *Role4*. Enfin, les Rôles *Role2* et *Role3* sont liés par une contrainte de dépendance obligeant un Agent à jouer ces deux Rôles.

Le modèle agent est la concrétisation du modèle organisationnel. Les ***Agents*** sont des entités actives et communicantes pouvant jouer plusieurs Rôles et appartenir à plusieurs Groupes. AGR ne contraint pas les Agents au niveau de leur architecture ou de leurs capacités mentales. Les Agents ne peuvent communiquer entre eux que s'ils font partie du même Groupe. Un Rôle ne peut appartenir qu'à un seul Groupe mais peut être joué par plusieurs Agents. La Fig. 3.3(b) donne un exemple de représentation graphique du niveau agent d'une modélisation AGR. Nous y retrouvons les mêmes



(a) Structure organisationnelle (b) Modèle agent

FIG. 3.3 – Exemple de modèlisation avec AGR

Rôles que dans l'exemple précédent, joués par un ensemble d'agents regroupés dans des Groupes. Un Agent respecte la contrainte entre les Rôles *Role2* et *Role3* en jouant ces deux rôles dans les deux instances de Groupe *Groupe1* et *Groupe2*.

Des travaux récents ont abouti à des extensions du modèle AGR : AGRE [42] prend en compte l'environnement dans la modélisation et AGREEN [8] tente d'intégrer à la fois la dimension environnementale et normative des organisations multi-agents. Les normes de AGREEN sont construites autour d'opérateurs déontiques et définissent des sanctions applicables en cas de non-respect.

Pour conclure sur ce modèle, AGR spécifie la structure d'une organisation en terme de groupes, de rôles et d'interactions entre rôles, mais rien ne définit la fonctionnalité de cette structure et des rôles en particulier. De ce fait, les modélisations que nous pouvons faire avec AGR donnent une vue partielle d'une organisation telle que nous avons pu la définir dans le Chapitre 2. De plus, les liens entre les rôles structurent de façon minimaliste les organisations imposées aux agents (pas de notion de hiérarchie ou d'héritage entre rôles). La description de l'organisation est abstraite et n'est plus accessible aux agents une fois l'organisation instanciée. Ce modèle permet toutefois d'organiser un ensemble d'agents de n'importe quelle taille.

3.1.3 Modèle TÆMS

TÆMS (Task Analysis, Environment Modeling, and Simulation) [24, 23, 22, 54] permet de décrire les structures de tâches des agents. La représentation d'une structure de tâches peut se faire selon trois points de vue différents. Le premier est un modèle génératif des étapes de résolution d'un problème dans un environnement. Le deuxième est une vue objective des vraies structures de tâches instanciées présentes dans une occurrence de résolution de problème. Enfin le dernier point correspond à la vue subjective que les agents ont de la vue objective dans la réalité. Le niveau objectif est le cœur de TÆMS et celui que nous allons décrire.

Une ***structure de tâche*** objective de TÆMS est essentiellement un arbre de décomposition de tâches annotées. Le noeud de plus haut niveau dans l'arbre, appelé "task group" représente un des buts qu'un agent doit essayer d'atteindre. Sous cette "task group" se trouve une séquence de ***tâches*** et de ***méthodes*** décrivant comment la "task group" doit être effectuée. Les tâches représentent des sous-buts eux-mêmes décomposés en sous-buts, etc. Par contre les méthodes sont terminales et représentent les actions primitives qu'un agent peut accomplir. En plus des éléments de bases que sont les tâches et les méthodes, nous avons également des relations qui modélisent les ***interactions*** entre différents nœuds et des ***ressources*** qui représentent les états des ressources du système.

Comme nous le constatons sur la FIG. 3.4, les relations décrivent comment l'exécution d'une méthode ou l'atteinte d'un but peut affecter les autre nœuds de la structure. Par exemple, l'exécution

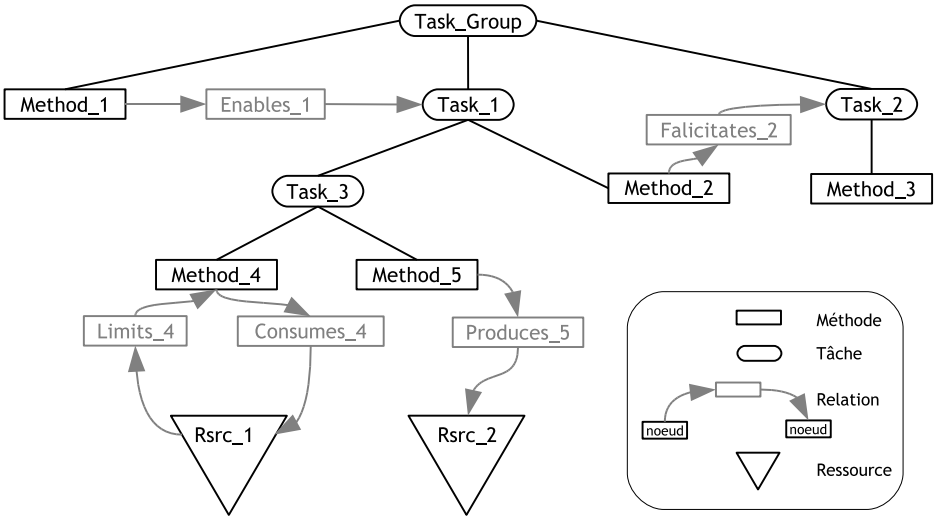


FIG. 3.4 – Exemple de structure de tâche TÆMS

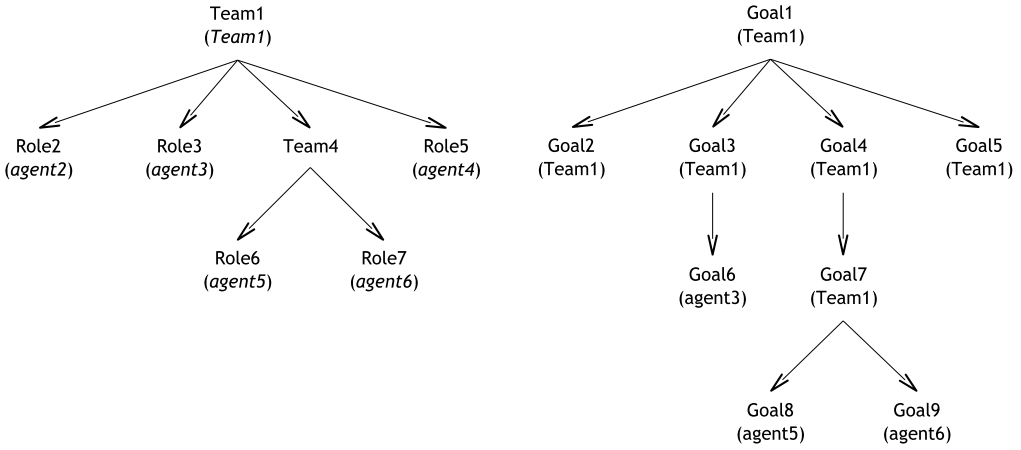
de la méthode "Method_1" peut permettre d'accomplir la tâche "Task_1". En d'autres termes, la tâche "Task_1" ne peut pas être atteinte avant que la méthode "Method_1" ne se soit achevée avec succès. Nous constatons également que plusieurs types de relations existent décrivant ainsi différents types de situation. Les différents labels des relations sont *enables*, *disables*, *facilitates*, *hinders*, *produces*, *consumes* et *limits*. Les relations peuvent partir de n'importe quel nœud et arriver sur n'importe quel autre nœud (en plus des ressources qui ne sont pas considérées comme des nœuds). Les relations sont également classées en trois catégories, *hard*, *soft* et *resource based*. Ceci contraint les relations qu'il peut y avoir entre certains nœuds. Des relations peuvent également lier deux structures de tâches différentes. Ainsi si ces deux structures appartiennent à deux agents différents, ces relations non locales décrivent les points où la négociation (ou la coordination) entre ces deux agents devrait avoir lieu. Ce modèle a donc une portée allant jusqu'à la société d'agents coordonnés.

Les ***QAF*** (*Quality Accumulation Function*) sont des étiquettes associées aux tâches et aux "task group". Elles définissent comment la qualité d'une sous-tâche peut être utilisée pour évaluer la qualité de la tâche globale. Il en existe de plusieurs types (*min*, *max*, *sum*, *exactly_one* par exemple). Suivant la QAF, le calcul de la qualité de la tâche ne se fera pas de la même manière. Ainsi, une QAF *min* est équivalente à un ET logique tandis qu'un QAF *max* est équivalente à un OU logique. Ainsi, les agents ne doivent pas seulement exécuter les tâches en respectant leur structure, mais ils doivent exécuter les tâches avec une qualité maximale. La vue structurelle donne aux agents une notion claire de la façon dont ces tâches sont organisées tandis que l'aspect qualitatif (avec l'utilisation des QAFs) permet à l'agent de comparer et de contraster ses différentes actions possibles, de prévoir leurs effets et de raisonner à propos de coordination avec d'autres agents. Cela est possible grâce à différents algorithmes mettant en place des formules mathématiques afin de maximiser les qualités des tâches que les agents exécuteront.

Ainsi ce modèle repose surtout sur une spécification fonctionnelle du SMA. Les agents ne sont pas contraints par une structure organisationnelle ou par un ensemble d'expressions normatives. Ils sont simplement guidés par leur objectif d'exécuter leurs tâches. Cela permet cependant une séparation entre la notion d'agent et le modèle de tâches d'environnement. Les agents ne sont ainsi pas obligés d'avoir une architecture particulière. Les relations entre les tâches permettent de faire le lien éventuellement entre deux tâches d'agents différents rendant possible la coordination entre eux. Cependant le modèle d'interaction l'autorisant n'est pas fourni.

3.1.4 Modèle STEAM

Le but de STEAM [93, 95, 96] est de permettre aux agents de former une équipe de travail de façon cohérente (*Shell for TEAMwork*). STEAM est à considérer sous deux aspects différents : un cadre d'exécution pour le développement des équipes de travail et un modèle d'équipe de travail, explicite et général, sur lequel les agents peuvent raisonner durant l'exécution de l'équipe. Le cadre d'exécution définit l'organisation et le plan de l'équipe avec TOP (Team-Oriented Program) [84]. TOP spécifie à un niveau abstrait l'organisation d'une équipe d'agents sous forme d'une ***hiérarchie de rôles*** et d'une ***hiérarchie de plan***. La hiérarchie de plan est composée des buts et des procédures de l'équipe ; les buts complexes sont décomposés en buts plus simples. Un programme TOP définit aussi un ensemble de contraintes de coordination entre les agents exécutant les activités de l'équipe. La Fig. 3.5 illustre un programme TOP avec d'un côté sa hiérarchie d'organisation et de l'autre sa hiérarchie de plan. Sur la partie 3.5(a), nous voyons que chaque feuille de l'arbre correspond à un rôle individuel tandis que le nœud interne et le nœud racine correspondent à des équipes d'agents. Pour chaque nœud, le label en italique indique l'agent du domaine remplissant les caractéristiques requises pour jouer le rôle à l'intérieur de l'organisation. Sur la partie 3.5(b), chaque nœud de l'arbre correspond à un but et à un agent ou une équipe (entre parenthèses) responsable de son accomplissement. Le programmeur de l'équipe d'agents spécifie les rôles et les équipes, provenant de la hiérarchie de rôles, devant atteindre les buts. L'infrastructure TOP assigne ensuite les agents et équipes aux rôles appropriés [84]. Les liens entre les buts correspondent à la décomposition des buts en sous-buts.



(a) Hiérarchie organisationnelle d'agents

(b) Hiérarchie de plan fonctionnelle

Fig. 3.5 – Exemple d'organisation suivant la méthode TOP

Le modèle d'équipe, sur lequel les agents peuvent raisonner pendant leur exécution, spécifie des directives décrivant les responsabilités et les engagements des membres d'une équipe de travail. Ces directives et actions sont indépendantes des domaines dans lesquels elles se situent. Le but est de fournir une connaissance commune avec laquelle les agents raisonnent de manière autonome sur la coordination et la communication dans l'équipe. Les comportements sont codés à l'extérieur des agents, améliorant ainsi la flexibilité de l'équipe. STEAM se base sur la théorie des intentions communes [70] permettant un raisonnement flexible sur les activités de coordination et sur la théorie des plans partagés [49] (structure hiérarchique d'intentions communes et individuelles). La connaissance commune de l'équipe de travail consiste en un ensemble de ***règles*** regroupées en trois sous-ensembles :

- les règles de "préservation de la cohérence", requérant une communication entre les membres de l'équipe afin d'assurer une initiation et une fin cohérente des plans de l'équipe,
- les règles de "gestion des rôles et de réparation", assurant la substitution des rôles entre les

membres de l'équipe en cas de disparition ou de défaillance de l'un d'entre eux,
– les règles de "sélectivité dans les communications", c'est-à-dire l'évaluation des coûts et des bénéfices des communications possibles afin de choisir les plus performantes et d'éviter ainsi des communications excessives dans l'équipe.

Environ 300 règles indépendantes des domaines d'application de STEAM sont réparties dans ces trois catégories. Ces règles sont exécutées dans une architecture Soar[1] [75] sur laquelle STEAM est basée.

Les hiérarchies de rôles structurent les rôles que les agents joueront et les hiérarchies de plans structurent les buts que ces rôles doivent atteindre. Ainsi un agent sera contraint d'atteindre les buts correspondant au rôle qu'il joue. Afin de faire en sorte que les agents se coordonnent, un ensemble de règles explicites, générales, indépendantes du domaine d'application et issues du modèle d'équipe permet aux agents de gérer leurs tâches, leurs rôles et leurs communications. Si nous reprenons les définitions des concepts sur lesquelles nous avons abouti dans le Chapitre 2, nous considérons qu'une Organisation est composée de différentes dimensions. Dans le cas de STEAM, nous pouvons faire l'amalgame entre une équipe et une organisation. Ainsi, nous distinguons trois dimensions : une dimension structurelle avec une définition des rôles, une dimension fonctionnelle avec une définition des tâches et une dimension réglementaire avec une définition des règles à suivre au sein de l'équipe. Ces règles sont de la forme "if-then-else" et concernent donc les dimensions structurelles et fonctionnelles. Elles traitent également des interactions alors que ces dernières ne font pas l'objet d'une dimension. Enfin, à part la notion de domaine faisant le lien entre la hiérarchie de rôles et la hiérarchie de buts, il n'existe pas de contraintes imposant ou interdisant explicitement le fait de devoir atteindre un but pour un rôle.

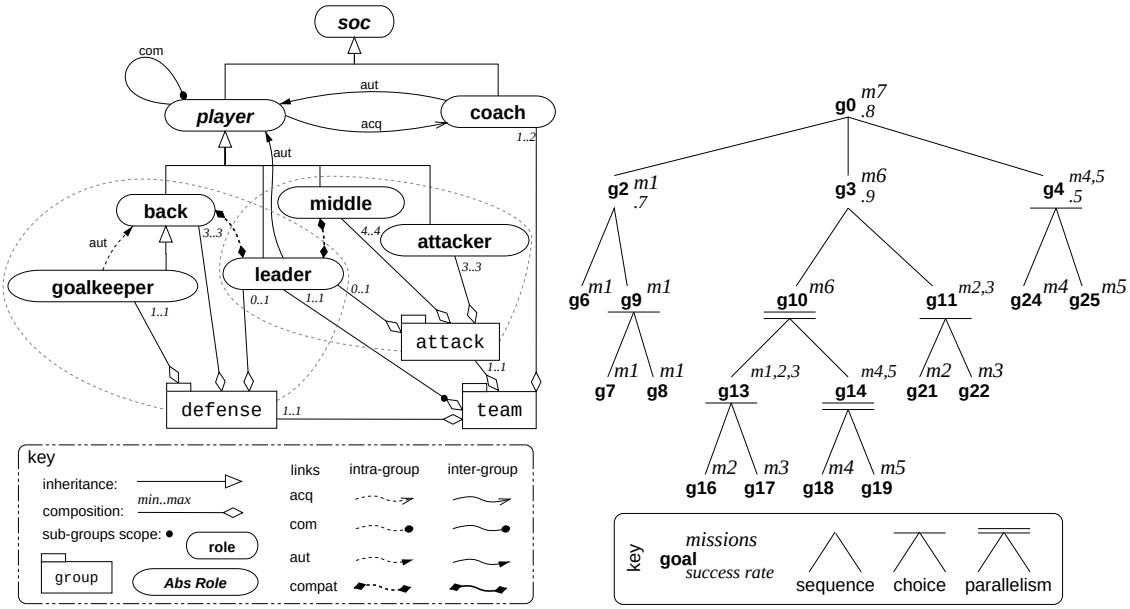
Après avoir étudié tous ces modèles organisationnels, nous voyons qu'ils contraignent le comportement des agents d'une façon structurelle en définissant des rôles comme dans AGR, GAIA et STEAM, ou d'une façon fonctionnelle en créant des inter-dépendances entre les agents via les buts qu'ils doivent atteindre comme dans TAEMS et STEAM. GAIA et AGR introduisent également une contrainte interactionnelle définissant les protocoles que les rôles sont en mesure de suivre. Les interactions existent également dans STEAM puisqu'elles font l'objet de règles mais ne sont pas explicitement spécifiées. Les règles abordées par STEAM sont un moyen complémentaire de contraindre les agents. Elles peuvent faire intervenir des expressions déontiques et définir le comportement d'agents envers d'autres agents au sein d'un contrat social par exemple (cf. un aperçu de différents modèles de contrats sociaux dans l'Annexe A.1 ou être encore plus complexes et prendre le nom de normes. Nous abordons maintenant les modèles organisationnels faisant intervenir un ensemble de normes dans leur définition.

3.1.5 Modèle $\mathcal{M}$oise$^+$

$\mathcal{M}$OISE$^+$ (Model of Organization for multI-agent SystEms) [57, 55] est un modèle organisationnel spécifiant un ensemble de contraintes pour les agents selon trois dimensions : une ***spécification structurelle***, une ***spécification fonctionnelle*** et une ***spécification déontique***. La spécification structurelle est construite sur les concepts de ***rôle***, de ***liens*** entre rôle et de ***groupes***. Comme illustré sur la Fig. 3.6(a), une spécification structurelle se représente graphiquement et nous permet de constater que les groupes sont composés de rôles, que ces rôles sont hiérarchisés et qu'ils peuvent être liés entre eux avec quatre types de liens. Les rôles sont des labels utilisés pour assigner un ensemble de contraintes sur le comportement des agents les jouant. Les liens sont des relations entre deux rôles qui contraignent directement les agents dans leurs interactions avec les autres agents jouant les rôles correspondants. Les groupes sont des ensembles de liens, de rôles et de relations de compatibilité entre rôles.

En ce qui concerne la spécification fonctionnelle, les concepts de bases sont le ***schéma***, le ***plan***, le ***but*** et la ***mission***. Comme illustré sur la Fig. 3.6(b), un schéma social est composé d'un ensemble de buts, de plans et de missions et représente une fonction d'un ou de plusieurs agents au

[1] Soar est une architecture générale de résolution de problème avec une mémoire basée sur les règles



(a) Spécification Structurelle (b) Spécification Fonctionnelle

FIG. 3.6 – Exemple d'organisation respectant le modèle $\mathcal{M}$OISE$^+$(provenant de [58])

sein de l'organisation. Le schéma a un but racine qui sera le but à atteindre pour exécuter le schéma social. Les buts sont découpés en plans, c'est-à-dire en ensembles de sous-buts qu'il faut atteindre séquentiellement, en parallèle, ou au choix (seulement l'un d'entre eux). Enfin les missions regroupent les buts en ensembles cohérents au sein d'un même schéma pour être assignées aux rôles.

La spécification déontique est composée d'un ensemble d'expressions mettant en relation les deux autres spécifications (un rôle et une mission) paramétrées par un opérateur déontique (Obligation, Permission ou Interdiction) et une contrainte temporelle. Ces relations rendent explicites la dimension normative des rôles que joueront les agents en définissant par exemple l'Obligation pour le rôle $R1$ d'accomplir la mission $m1$ avant la date t. Ces expressions contraignent les agents sur la possibilité de s'engager sur une mission ou d'atteindre un but suivant le rôle qu'ils jouent.

La spécification de ces trois dimensions forme une spécification d'organisation (OS) tandis que l'instanciation des trois dimensions à l'aide d'agents jouant les rôles et donc contraints par l'organisation représente une entité organisationnelle (OE). A un instant t de l'évolution de l'OE, chaque agent se voit contraint d'accomplir ou de ne pas accomplir une mission.

Par rapport aux modèles vus dans la section précédente, nous remarquons que $\mathcal{M}$OISE$^+$ utilise à la fois une spécification de la structure et de la fonction du SMA ainsi qu'une spécification déontique. De plus, même si la fonctionnalité du système n'est pas aussi poussée que dans TÆMS, la structure est une des plus complètes vue jusqu'à maintenant et les expressions déontiques se basent sur ces deux spécifications pour définir précisément les droits et devoirs non pas des agents mais des rôles que ces agents joueront. Suivant si l'agent joue un ou plusieurs rôles, l'agent sera plus ou moins contraint à accomplir ou non des missions définies dans les fonctions du système. Cependant, AGR comme GAIA spécifie les interactions entre les agents (ou les rôles selon le point de vue), ce que $\mathcal{M}$OISE$^+$ ne fait pas directement puisqu'il ne définit aucun protocole mais des contraintes sur les communications entre rôles grâce aux liens.

La séparation en trois spécifications distinctes apporte une plus grande flexibilité au modèle et

permet d'assigner plus explicitement des buts à atteindre aux rôles contrairement à la description des activités de GAIA associées directement à chaque rôle. STEAM est plus proche de $\mathcal{M}$OISE$^+$ dans le sens où ce modèle définit une dimension structurelle et une fonctionnelle. Le lien entre ces deux dimensions est cependant fait à l'aide des domaines regroupant rôles et buts. Nulle part il est fait mention d'obligation pour un rôle d'atteindre un ou plusieurs buts de son domaine. Les règles présentes permettent seulement la gestion des rôles, des communications et des buts. L'aspect normatif du modèle n'est donc pas explicite. Concernant $\mathcal{M}$OISE$^+$, il manque quant à ce modèle une spécification des protocoles permettant de définir les interactions entre rôles et la possibilité de normer ces interactions.

Les normes que nous venons de voir sont adéquates pour des agents censés respecter les règles auxquelles ils sont soumis. Toutefois, dans le cas de notre étude et de notre volonté de faire évoluer des agents autonomes, capables de décider si oui ou non ils respectent leurs contraintes, ces définitions ne sont pas complètes par rapport à celles de normes vues précédemment. Les normes ici mettent en relation un rôle, une action et une expression déontique. Nous pouvons donc exprimer pour un agent l'obligation, l'interdiction ou la permission sur une action. Cependant, un agent délibératif se trouvant face à un choix entre deux normes à respecter ne pourra pas résoudre le conflit autrement qu'aléatoirement. Il faudrait par exemple une expression des normes plus précises faisant intervenir les sanctions encourues.

3.2 Modèles d'Institution

Nous avons vu dans le Chapitre 2 que les Organisations et les Normes se complétaient. En effet, les normes définissent les droits et devoirs des acteurs au travers du rôle qu'il joue et sur les actions qu'ils sont capables d'exécuter. Avant d'étudier les normes au sein de trois modèles organisationnels, nous allons voir deux formalismes de normes dans le domaine des SMA.

3.2.1 Notion de norme

Dans le domaine des SMA les normes sont considérées tour à tour comme des contraintes du comportement des agents, les buts de l'agent, ou des obligations de l'agent. Leurs définitions s'en trouvent pour cette raison légèrement différentes que celles abordées dans le Chapitre 2, notamment dans la section traitant des normes sociales.

Guido Boella et Leonardo Lesmo définissent dans [13] un modèle d'agents normatifs et délibératifs. Ils se basent pour cela en partie sur [20] de Rosaria Conte, Cristiano Castelfranchi et Frank Dignum. Les agents sont des ensembles composés de cinq éléments dont l'un d'eux L est un ensemble de tuples représentant les obligations connues par l'agent et dont il est le porteur. Ces obligations sont une sorte de norme. Dans le composant L d'un agent, une obligation Ω est représentée par un ensemble de quatre éléments O, B, N, R où :

- O est le contenu de l'obligation, c'est-à-dire l'état ou l'action que N veut que B atteigne ou exécute.
- B est l'agent porteur de l'obligation.
- N est l'agent normatif.
- R est une action (appelée sanction) que N va faire en cas de détection de la violation de l'obligation.

Nous avons donc deux individus composant l'obligation, à savoir le porteur de la norme c'est-à-dire celui qui devra la respecter et l'autorité qui a posé l'obligation et qui peut sanctionner en cas de violation. Les normes concernent directement les agents à propos d'une action ou d'un but. La notion d'Obligation est renforcée par une sanction applicable en cas de non respect de la norme.

Mark d'Inverno, Michael Luck et Fabiola López y López ajoutent à cette définition une notion de condition d'application de la norme. En effet ils définissent dans [34, 71] une norme comprenant

la référence de deux agents autonomes, un contexte et des exceptions de validité, des buts étant les buts à atteindre pour satisfaire la norme ainsi que les buts qui seront atteints en cas de récompense ou de punition. A partir de cela, une classification peut être faite. Une obligation est une norme avec une punition, une interdiction est une obligation, un *social commitment* est une norme avec une récompense, et un *social code* est une norme sans punition ni récompense (une permission en quelque sorte). Les normes ne sont pas isolées, le respect ou non d'une norme peut en déclencher une autre. La description d'une norme est donnée ci-dessous en langage de spécification Z.

```
Norm {
   addressees, beneficiairies : P AutonomousAgent
   context, exceptions : EnvState
   ngoals, rewards, punishments : P Goal
}
```

Maintenant que nous avons vu qu'une norme définissait une obligation d'un agent sur une action ou un but à atteindre, nous allons voir la relation que les normes peuvent avoir avec les différentes dimensions d'un modèle organisationnel.

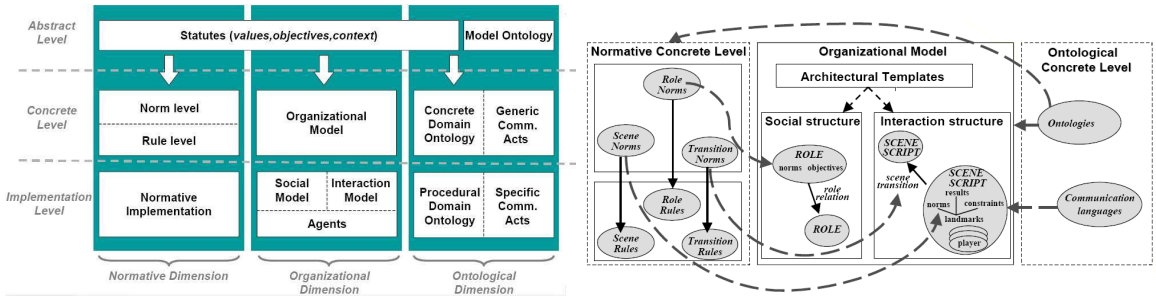
3.2.2 Modèle OMNI

La plate-forme OMNI (Organizational Model for Normative Institutions) [32, 33, 104] couvre d'après Virginia Dignum et Javier Vázquez-Salceda tous les aspects qu'un SMA ouvert ou non doit pouvoir fournir. Ces aspects sont :

- Pouvoir modéliser des systèmes complexes en définissant l'organisation dans laquelle l'agent évolue.
- Pouvoir définir et détecter les bons et mauvais comportements des agents dans le but d'instaurer un climat de confiance entre les agents.
- L'organisation doit pouvoir représenter tout ce qui permettrait de modéliser le contexte dans lequel un agent peut interagir.

.

Le modèle OMNI est basé sur les modèles OperA de Virginia Dignum [30] apportant la ***dimension organisationnelle*** de OMNI et Harmon*IA* de Javier Vázquez-Salceda [102] apportant ***la dimension normative***. Comme illustré sur la Fig. 3.7(a), le modèle OMNI peut se représenter comme une matrice avec d'un côté un découpage en dimensions et de l'autre un découpage en niveaux. Nous avons ainsi les ***dimensions Normative***, ***Organisationnelle*** et ***Ontologique*** ainsi que les niveaux Abstraits, Concrets et d'Implémentation. Nous allons nous intéresser plus particulièrement ici au niveau Concret.



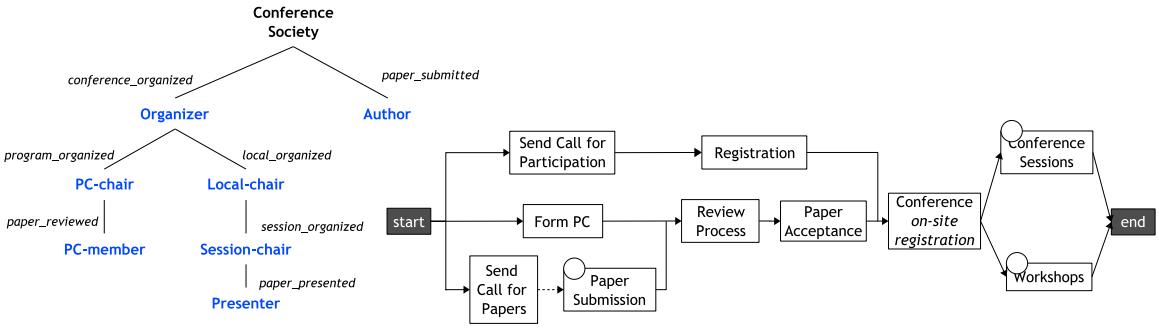
(a) Niveaux et dimensions (b) Détails du niveau concret

Fig. 3.7 – Aperçu du modèle OMNI (provenant de [104])

Le ***modèle organisationnel concret*** se compose d'une structure de rôles et d'une structure d'interactions qui seront implémentées en deux modèles. Le modèle social spécifie les rôles que jouent

les agents [2] et le modèle d'interaction décrit les interactions effectives des agents. La structure de rôles est en fait un arbre de dépendance de rôles comme nous pouvons le constater sur la FIG. 3.8(a)[3]. Chaque rôle est associé de façon non-flexible à un objectif. En ce sens, la spécification fonctionnelle est intégrée à la spécification structurelle. A noter également que la définition des rôles inclut les normes les concernant (en plus d'un identifiant et d'un type) et dont nous verrons la construction plus tard.

La ***structure d'interaction*** permet quant à elle de définir un ensemble de scènes liées les unes aux autres par des transitions décrivant un ordre partiel des scènes plus d'éventuelles contraintes de synchronisation. Une scène suit la spécification d'un script de scène. Un script de scène décrit une scène par ses acteurs (les rôles), ses résultats attendus et les normes régulant les interactions. Les résultats d'une scène sont atteints par la réalisation des objectifs des rôles. Un script de scène permet aussi de définir un modèle d'interaction entre les rôles. Les scènes peuvent avoir lieu en même temps et un même agent peut participer simultanément à différentes scènes. Nous avons sur la FIG. 3.8(b) un exemple de structure d'interactions. Les scènes d'interactions sont elles-même décrites par des scripts de scènes établissant également un plan d'interactions entre les rôles et qui est une combinaison des objectifs et sous-objectifs des rôles. La description d'une scène d'interaction se fait donc avec la spécification d'un ensemble de rôles, de *patterns* d'interactions et de normes.



(a) Structure sociale (b) Structure d'interactions

FIG. 3.8 – Exemple d'Organisation OMNI pour le scénario *Conference*

Les ***normes*** utilisées pour spécifier les rôles ou les scripts de scène sont définies au sein de la dimension normative de OMNI. Elles sont liées aux éléments de la dimension organisationnelle comme illustré sur la FIG. 3.7(b). Les normes sont définies au niveau concret avec le langage pour les normes concrètes, *CNorms* [103] utilisant une logique déontique, temporelle, relative et conditionnelle. Ces normes sont ensuite transformées en règles. La transformation des normes en règles dans OMNI est une transition d'une perspective normative vers une plus descriptive et permet donc de passer de la logique déontique à la Logique Dynamique Propositionnelle (PDL) [44]. Chaque norme peut être traduite en une expression de violation. Il faut donc également définir une sanction (l'action à exécuter contre le violeur de la règle), une correction (les actions à mettre en place pour résoudre les problèmes survenant de la violation) ainsi qu'un ou des rôles ayant la responsabilité de détecter ce genre de violation.

Le but de OMNI est finalement de définir de manière précise les contextes au sein desquels les agents devront interagir. Ainsi, les rôles, les scènes et les interactions sont vus comme des normes,

[2]Le composant de base d'un modèle social est un contrat social défini en LCR et décrit en Annexe A.1.3. En effet, étant donné une structure de rôle SR et un ensemble d'agents A, le modèle social est défini comme étant un ensemble de contrats sociaux mettant en relation les agents de A avec les rôles dans SR.

[3]Les structures de la FIG. 3.8 reprennent la modélisation des rôles et des interactions pour l'exemple de l'organisation d'une conférence telle qu'elle est définie dans la présentation "Ins and Outs of Agent Organizations" de Virginia Dignum à l'occasion de son passage dans différentes Universités australiennes en 2006.

c'est à dire que pour chaque concept, des normes sont incluses dans leur définition. Le concept de rôle permet de structurer une organisation de 1 à n agents. Les éléments de modélisation sont pris en compte sur plusieurs niveaux allant de l'abstraction à l'implémentation. Les ontologies font partie intégrante du modèle et lui permettent de fonctionner sur différents domaines d'application tels que l'organisation d'une conférence scientifique ou l'allocation d'organes [100, 101].

Cependant, aucune fonction de système, de plan ou de schéma d'exécution n'est défini avec OMNI. Seuls quelques objectifs et sous-objectifs peuvent éventuellement servir de structure fonctionnelle. Ces objectifs se retrouvent dans les structures de rôles et d'interaction et ne sont donc pas indépendants de la spécification des autres dimensions.

3.2.3 Modèle ISLANDER

ISLANDER [38, 105, 37, 36] se définit comme un IDL (*Institution Definition Language*). Ce langage basé sur XML est une syntaxe pour la définition d'Institutions Électroniques représentées comme un système dialogique permettant l'échange de messages. Ces interactions sont structurées au travers de regroupement d'agents appelés scènes et qui suivent des protocoles explicitement définis. Une Institution Électronique est vue par Marc Esteva [35] comme étant composée d'un cadre d'exécution dialogique (dialogic frameworw, DF), d'une structure performative (PS) et de normes. Les normes gouvernent les interactions entre les agents et sont exprimées par une représentation en langage informatisable.

Le ***cadre d'exécution dialogique*** contient les éléments pour la construction des expressions du langage de communication. Le but est de permettre à des agents partageant le même cadre d'exécution dialogique de pouvoir s'échanger des connaissances. Il est ainsi composé d'une ontologie, d'un ensemble de définitions de types et de fonctions, d'une liste de *particules illocutoires* [4] valides et d'un ensemble des rôles. Tout ceci construit un ensemble de ce que les auteurs appellent *illocutions*. Les illocutions spécifient la structure des messages qui peuvent être échangés entre les agents. Les rôles seront joués par les agents qui participeront à l'organisation. Chaque rôle est sensé définir un ensemble de comportements types à l'intérieur de l'institution. Il est possible de spécifier des relations entre les rôles grâce aux liens hiérarchiques (si un agent peut jouer un rôle alors il est aussi capable de jouer son super-rôle) et aux liens de séparation statique des devoirs (*static separation of duties* ou ssd) définissant qu'un même agent peut jouer les deux rôles. Nous avons un exemple de représentation de la hiérarchie de rôles avec l'outil d'édition graphique de ISLANDER sur la partie gauche de la Fig. 3.9.

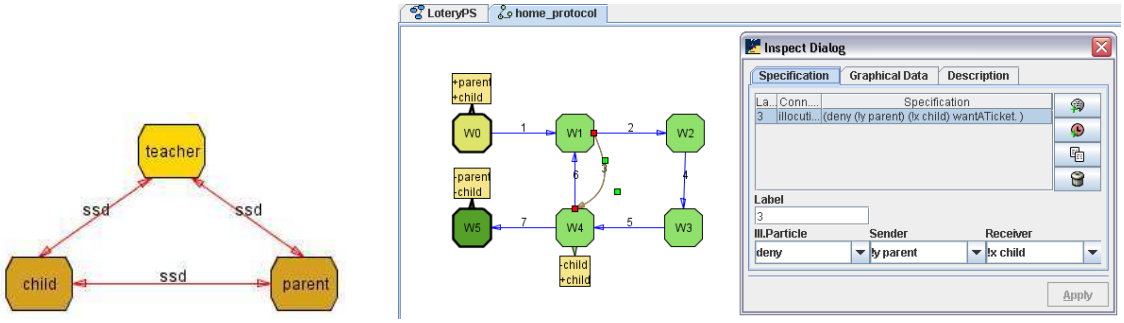


Fig. 3.9 – Exemple d'une définition de hiérarchie de rôles et d'un protocole avec ISLANDER

La ***structure performative*** définit un ensemble de scènes liées entre elles par des transitions. Les scènes sont des regroupements d'agents devant suivre des protocoles de communication spécifiques. Rien ne peut se produire en dehors du contexte d'une scène. Un agent peut participer à plusieurs

[4] une particule illocutoire ou *illocutionary particle* peut être considérée comme un *communicative act* standardisé par la FIPA

scènes en même temps. Les protocoles sont représentés par un graphe orienté où les nœuds correspondent aux différents états de la conversation et les arcs à des schémas d'illocution faisant évoluer l'état de la conversation. Ainsi, à chaque point de la conversation est défini un ensemble d'illocutions spécifiant qui peut dire quoi et à qui. Sur la partie droite de la FIG. 3.9, nous avons la représentation de la spécification d'un protocole avec l'outil d'édition de ISLANDER. La partie droite illustre l'édition du schéma d'illocution pour l'arc ayant le label "3".

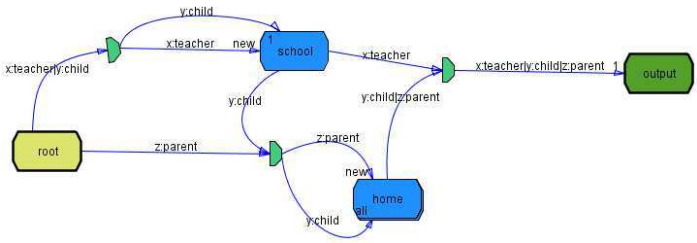


FIG. 3.10 – Exemple d'une structure performative avec ISLANDER

Les transitions liant les scènes au sein d'une structure performative déterminent la "police de flux des rôles" entre les différentes scènes. Cela veut dire *comment* les agents peuvent passer de scènes en scènes selon le rôle qu'ils jouent et *quand* une conversation peut débuter au sein d'une scène. Pour cela les transitions sont représentées par des scènes spéciales pouvant exprimer selon son type (OR ou AND) la synchronisation de rôles, un choix ou un parallélisme entre deux scènes. Parmi l'ensemble des scènes, celles étant définies en tant qu'initiale et finale seront les points d'entrée et de sortie de l'institution. Sur la FIG. 3.10 un exemple de structure performative est représenté. Nous constatons ici que ce sont les rôles qui "naviguent" de scènes en scènes et non pas l'organisation ou la structure de rôles toute entière qui se trouve dans une scène ou un état particulier.

Les ***normes*** servent à capturer les conséquences des actions des agents sous forme d'obligation. Une norme est définie par l'expression des actions qui provoquent son activation, les actions que les agents doivent faire et une liste d'obligations qui deviendront actives. ISLANDER étant basé sur la spécification des interactions entre agents, les actions sont exprimées par la paire schéma d'illocution et scène (où le schéma aura été émis). La notion déontique d'obligation est représentée par le prédicat $Obl(x, \Psi, s)$ exprimant le fait que l'agent x est obligé de faire Ψ dans la scène s, et Ψ étant un schéma d'illocutions. Le schéma d'une norme peut se représenter de la façon suivante $(s_m, \gamma_m) \wedge e_m \wedge \neg(s_{(m+1)}, \gamma_{(m+1)}) \wedge obl_m$; si l'illocution γ_m a été émise dans la scène s_m, que l'expression booléenne e_m est satisfaite et que l'illocution $\gamma_{(m+1)}$ n'a pas été émise dans la scène $s_{(m+1)}$, l'obligation obl_m est valide. Ainsi la règle définissant la validité de la norme est composée de deux parties, la première définit la cause de l'obligation (par exemple gagner une enchère générant l'obligation de payer) et la deuxième est celle annulant l'obligation (par exemple payer le montant dû). Plus de précisions peuvent être trouvées dans les travaux sur ISLANDER et notamment ceux concernant l'implémentation des normes [46].

La division de toutes les interactions possibles entre les agents en scènes permet à ISLANDER d'avoir une conception modulaire concrète du système, apportant ainsi de la flexibilité et de la lisibilité aux modèles. Cependant, en comparaison avec $\mathcal{M}$OISE$^+$, la spécification structurelle en hiérarchie de rôles est minimale dans le sens où on peut seulement définir des rôles et des sous-rôles ainsi que des liens de compatibilité entre rôles. Cette simplicité ressemble à celle de OMNI, ce modèle permettant seulement la définition d'un arbre de dépendance des rôles. Cela permet néanmoins de structurer les agents qui auront à évoluer dans l'Institution Électronique. La définition de fonctions d'agent est impossible avec ISLANDER tandis que $\mathcal{M}$OISE$^+$ ne permet pas la définition d'interactions entre les agents. Les normes sont exprimées plus simplement que les expressions déontiques de $\mathcal{M}$OISE$^+$ et ne peuvent être que des obligations. Les normes concernent seulement les interactions entre agents et les

transitions entre scènes alors qu'elles traitent des missions pour $\mathcal{M}$OISE$^+$ et des interactions et des rôles dans OMNI.

3.3 Étude comparative

Nous avons défini précédemment qu'un SIM pouvait se composer d'une organisation et d'un ensemble de normes et qu'une organisation spécifiait une structure des agents et un ensemble de fonctions que ces agents peuvent exécuter. Nous avons vu durant ce chapitre que l'organisation spécifiait également la façon dont les agents pouvaient interagir les uns avec les autres. Ainsi, nous reprenons ces différents concepts comme points de comparaison dans le Tab. 3.1. Le modèle OMNI que nous avons étudié dans la Section 3.2.2 est structuré en dimensions et également en niveau d'abstraction. Nous semblant un point d'entrée intéressant, nous l'ajoutons au tableau sous le nom de description. Nous y joignons également la portée des modèles étudiés. A l'aide de ce tableau, nous constatons qu'un modèle sur deux ne définit pas de spécification d'interaction. Enfin la plupart des modèles ont une portée allant de la contrainte d'un agent à une société d'agents.

SIM	Structure	Fonction	Interaction	Norme et objet normé	Description	Portée	Adaptabilité
GAIA	rôle	responsabilité; activité	protocole	permission(rôle,fonction)	abstrait	1 agt → société	fixe
AGR	groupe; rôle		protocole		abs. → conc.	1 agt → structure	fixe
TAEMS		tâche; méthode; relation			concret	1 agt → société	fixe
STEAM	hierarchie de rôles	hiérarchie de plans		règle(agent)	abstrait	1 agt → structure	flexible
Moise+	groupe; rôle; lien	but; plan; mission		deon(rôle,mission)	concret	1 agt → structure	évolutive
OMNI	hierarchie de rôles	objectif	scène	norme(rôle); norme(scène)	abs. → impl.	1 agt → structure	flexible et évolutive
Islander	rôle; lien de compat.		scène; protocole	obligation(interaction)	concret	1 agt → structure	fixe

Tab. 3.1 – Tableau comparatif des modèles de spécification de contrainte

Détaillons les informations de ce tableau. Concernant GAIA, comme nous nous restreignons aux modèles d'analyse, il fournit seulement une spécification abstraite des organisations. Par contre, il permet de contraindre aussi bien un agent qu'une société entière. Cependant, les spécifications sont minimales en terme de structure et de fonctionnalités. On a juste un ensemble de rôles et pour chaque rôle une ou plusieurs activités à exécuter. De plus, ces modèles servant de base aux modèles de conception, les agents n'agiront pas en fonction d'un modèle mais celui-ci sera codé au sein de chacun d'entre eux. Le modèle est donc fixe.

AGR fournit une spécification de la structure d'une société d'agent plus complète que les modèles d'analyse de GAIA. On peut en effet définir des structures de groupes et un ensemble de rôles au sein de ces groupes au niveau organisationnel. Un ensemble de contraintes peut compléter la structuration. Cependant, c'est la seule chose qu'AGR permet. Nous avons indiqué dans le tableau qu'il est possible de décrire un ensemble de protocoles lors de la définition des rôles. Ce ne sont que des liens spéciaux entre les rôles. Ce modèle structure également un ensemble d'agents jouant des rôles et appartenant aux groupes décrits par les structures de groupes. Nous avons donc un niveau abstrait et un niveau concret. Les agents, pour peu qu'ils possèdent les méthodes adéquates, pourront prendre en compte le modèle afin de jouer un rôle dans un groupe d'agents. Ce modèle ne leur donnera par contre pas d'information sur les fonctions qu'ils devront exécuter au sein de l'organisation.

Les modèles STEAM et TÆMS se concentrent la spécification des fonctionnalités. Pour STEAM, une hiérarchie de plans (ce qui implique une structuration des buts) est liée à une hiérarchie de rôles, décrivant ainsi la fonctionnalité globale d'une structure d'agent et la façon dont elle sera exécutée. La gestion de la coordination entre les agents se fait grâce à un ensemble de règles influençant les agents. TÆMS ne fournit pas ce genre de règles, que ce soit au niveau des expressions normatives ou de la structure de l'organisation. TÆMS se contente de bien spécifier des hiérarchies de tâches

pouvant être liées les unes aux autres et se terminant par des méthodes interprétables et exécutables par les agents. Cette modélisation concrète permet d'organiser l'activité d'agents dont le nombre va de un à une société. Les règles de STEAM appliquées à la gestion des rôles et de leurs fonctionnalités permet de façon abstraite d'aller jusqu'à contraindre une structure d'agents.

Les trois derniers modèles que nous avons étudiés définissent de façon plus intégrée plusieurs dimensions. OMNI est le seul à couvrir tous les aspects d'une organisation normative de manière presque complète. En effet, la structure de l'organisation fait apparaître un ensemble de rôles liés les uns aux autres dans une hiérarchie de dépendance. Les interactions sont également mieux définies avec la spécification d'un ensemble de scènes liées les unes aux autres et de protocoles pour chacune des scènes. Cependant et tout comme GAIA, les fonctionnalités des agents sont spécifiées comme de simples objectifs que les rôles doivent atteindre.

Enfin, nous pouvons voir les deux derniers modèles comme complémentaires. $\mathcal{M}$OISE$^+$ se concentre sur la structure et la fonctionnalité de l'organisation tandis qu'ISLANDER définit de façon précise le cadre d'interaction à base de scènes, ainsi que les protocoles à y suivre. Une hiérarchie de rôles avec la possibilité de définir des liens de compatibilité entre eux est présente mais est très limitée à côté des groupes, rôles et liens de $\mathcal{M}$OISE$^+$. Alors que $\mathcal{M}$OISE$^+$ définit de façon détaillée (presqu'autant que TÆMS) un ensemble de buts regroupés en missions et définis par des plans que les agents devront atteindre, ISLANDER définit un ensemble de protocoles que les agents devront suivre. Il est donc tout naturel de trouver des expressions normatives portant sur un rôle et une mission pour $\mathcal{M}$OISE$^+$ et sur une interaction pour ISLANDER. Cependant les normes d'ISLANDER sont plus complexes à mettre en place et sont rarement utilisées en pratique.

Concernant les expressions normatives, seul OMNI prend en compte la possibilité que les agents ne respectent pas la norme et subissent une sanction de la part d'une autorité. Nous considérons que cette faculté est essentielle pour que le système d'arbitrage puisse superviser et intervenir si nécessaire. ISLANDER et $\mathcal{M}$OISE$^+$ vus comme les modèles les plus complets et les plus aboutis ne le fournissent pourtant pas.

La dynamique d'une organisation, c'est-à-dire la façon dont elle évolue au cours du temps, est également importante. ISLANDER et OMNI définissent pour cela un enchaînement de scènes. Les modèles définissant une hiérarchie de buts spécifient la dynamique ainsi. Le but global de l'organisation doit être atteint, pour cela, un ensemble de sous-buts doit être satisfait. C'est la satisfaction des buts qui fait évoluer les agents au sein de l'organisation. Il y a donc les modèles définissant comment les agents doivent interagir et les modèles définissant comment les agents doivent agir.

Suivant le domaine d'application, ISLANDER ou $\mathcal{M}$OISE$^+$ sera le plus adapté. Dans le cas où une spécification des interactions ne serait pas indispensable $\mathcal{M}$OISE$^+$ serait la solution. Il faudrait cependant étendre ses expressions déontiques afin de définir l'émetteur de la norme et la sanction encourue en cas de non respect. De même pour ISLANDER en ce qui concerne les normes. Les applications ne se contentent souvent pas de faire interagir des agents. Elles veulent faire en sorte que les agents agissent également. Dans ce cas, ISLANDER devrait ajouter une dimension fonctionnelle à ses spécifications.

3.4 Synthèse

Nous avons passé en revue dans ce chapitre les différentes façon de définir un SIM afin de contraindre un ensemble d'agents. Chacun des modèles nous permet de spécifier une structure d'organisation, un ensemble de fonctionnalités, des protocoles d'interaction, des expressions normatives ou parfois tout ces éléments à la fois. La FIG. 3.11 donne une vision synthétique d'un SIM se composant d'un modèle organisationnel découpé en sous-modèles :

– Un modèle de ***Rôles*** : les agents jouent des rôles et appartiennent à des groupes.

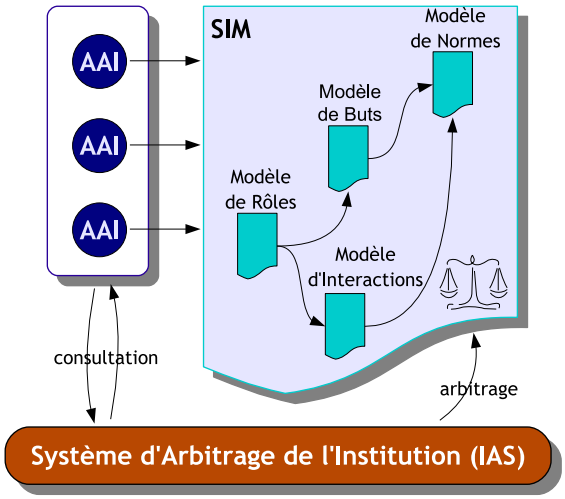


FIG. 3.11 – Modèle générique de spécification d'organisation

- Un modèle de ***Buts*** : les agents doivent atteindre directement ou via les rôles qu'ils jouent des objectifs, des buts communs ou exécuter un plan.
- Un modèle d'***Interactions*** : les agents doivent suivre un ensemble de protocoles définissant comment ils peuvent communiquer les uns avec les autres via les rôles qu'ils jouent.
- Un modèle de ***Normes*** : il exprime un ensemble d'actions ou d'interactions que les agents ou les rôles qu'ils jouent sont autorisés ou obligés à faire.

L'intérêt est maintenant de savoir comment mettre en pratique cette spécification. OMNI semble la concrétiser jusqu'à l'implémenter. Les agents vont donc devoir interagir avec l'implémentation du modèle mettant en œuvre l'instanciation de la spécification des contraintes. Qu'en est-il des autres types de modèles ? Nous allons voir dans ce qui suit un ensemble d'IAS permettant de contrôler qu'un ensemble d'AAI externes à la plate-forme respectent le SIM.

Chapitre 4

Systèmes de Gestion d'Organisation et d'Institution Multi-Agents

Dans le domaine des Systèmes Multi-Agents, une des caractéristiques principales des agents est d'être autonome [52, 15, 79]. L'agent décide seul de mettre en place ou non une action en fonction de l'environnement dans lequel il évolue d'une part et selon ses états mentaux d'autre part. Nous avons vu dans le chapitre précédent que les agents peuvent être soumis à un ensemble de contraintes, qu'elles soient structurelles, fonctionnelles, interactionnelles ou normatives. Cependant, de par leur autonomie, les agents peuvent prendre la décision de ne pas respecter une ou plusieurs de ces contraintes afin d'atteindre un objectif local par exemple. Dans une société humaine comme dans une société d'agents, si tous les individus se mettent à enfreindre les règles pour satisfaire leurs propres buts, les contraintes censées apporter un certain ordre risquent de laisser place à une situation chaotique. Afin de faire en sorte que l'organisation spécifiée soit respectée, les systèmes d'arbitrage (IAS) permettent de superviser l'autonomie des agents.

Comme vu dans le chapitre précédent, les contraintes portent sur les agents (directement ou indirectement via les rôles par exemple) et ce sont ces mêmes agents, les AAI du système, que le système d'arbitrage (le IAS) doit superviser. Notre volonté est d'avoir un système de contraintes basé sur une organisation ouverte et hétérogène (cf. Chapitre 1). De ce fait, n'importe quel agent devrait pouvoir rejoindre une organisation, comprendre ses règles de fonctionnement et s'y conformer. Cependant, les agents doivent avoir dans la plupart des cas une architecture plus ou moins imposée afin qu'ils soient supervisés par l'IAS et soient capables d'interpréter les contraintes.

Nous allons voir dans ce chapitre les IAS des SIM étudiés dans le chapitre précédent (lorsque ceux-ci existent, ce qui n'est pas le cas pour GAIA et TÆMS). Nous aborderons donc les systèmes de gestion d'organisation puis les systèmes de gestion d'institution. Pour chaque IAS, après avoir vu à quel SIM il se rattache, nous étudierons les différents services offerts pour l'arbitrage, le lien qu'il a avec les AAI, l'architecture de ces AAI et enfin la façon dont les AAI vont interpréter les contraintes du SIM.

4.1 Systèmes de gestion d'Organisation

Nous étudions dans cette section l'intergiciel MadKit implémentant des modèles AGR et KARMA permettant de contrôler les organisations de type STEAM. Nous voyons ensuite CAS mettant en place une société d'agents contractuels que nous n'avons pas étudiée dans le chapitre précédent mais qui est disponible en Annexe A.1.1 ainsi qu'un exemple de fonctionnement d'organisation virtuelle basée sur

le modèle de contrat ECF ici aussi abordé en Annexe A.1.2. Nous finissons par $\mathcal{S}-\mathcal{M}$OISE$^+$ contrôlant les organisations de type $\mathcal{M}$OISE$^+$.

4.1.1 MadKit

La plate-forme MadKit [50] implémente les modèles de type AGR (cf. Section 3.1.2). L'utilisateur définit les agents, cependant aucune spécification ne lui permet de définir des règles, elles sont donc codées à l'intérieur des agents. L'autonomie vis-à-vis de ces règles est donc plus difficile à mettre en place et le contrôle du respect des règles est encore plus compliqué.

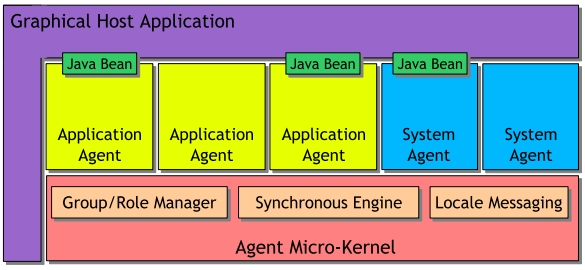


FIG. 4.1 – Architecture de MadKit (provenant de [50])

La plate-forme MadKit est composée de plusieurs couches (cf. FIG. 4.1) : une micro plate-forme dédiée au fonctionnement des agents (*Agent Micro-Kernel*), un ensemble d'agents fournissant des services d'agentification (*System Agent*) et une interface graphique (*Graphical Host Application*). MadKit inclut également des modèles d'agents standards, ils peuvent être utilisés tels quels ou servir de base de développement à d'autres agents. Ce sont eux (*Application Agent*) qui vont agir dans les organisations. Quant aux agents du système, ils servent à créer des groupes spéciaux permettant l'accès des agents d'application à d'autres groupes. Ce sont par exemple les agents *Entrance Group*, *GateKeeper* ou encore *GroupManager*.

Les agents pouvant évoluer au sein de la plate-forme doivent être programmés soit en Java soit à l'aide d'un langage de script parmi Python, Scheme (Kawa), BeanShell ou encore Jess. Si la plate-forme permet à une API de prendre en compte un script pour faire fonctionner un agent, un agent ne peut néanmoins pas prendre en compte un script pour être organisé. Il n'y a pas d'interprétation d'un modèle d'organisation externe, l'organisation est spécifiée au sein de chaque agent et c'est en créant des groupes et en jouant des rôles grâce à leurs méthodes que les agents vont instancier l'organisation.

Les agents dans MadKit héritent de la classe abstraite `AbstractAgent`, ou d'une de ses sous-classes, et possèdent les capacités suivantes : contrôle et cycle de vie, service de messagerie et vue organisationnelle. Cette dernière va leur permettre de jouer des rôles et de rejoindre des groupes. Cette vue est contrôlée par les agents eux-mêmes par rapport à un référentiel interne. La supervisation par un système d'arbitrage externe est de ce fait impossible.

Comme abordé dans la Section 3.1.2, le modèle AGR a donné lieu à l'extension AGREEN [9, 8]. Dans MadKit chaque organisation se voit associer une classe AGREENS (pour *AGREEN Service*) ajoutant un ensemble de normes supervisées définissant ainsi une Institution Électronique. Ce service a pour objectif de maintenir l'état d'un environnement social (les rôles, les obligations, etc.) cohérent et de traiter les actes institutionnels (nommer, destituer, obliger, interdire, donner des permissions, etc.) qui lui sont destinés.

4.1.2 KARMA

La FIG. 4.2 représente le cadre d'exécution Teamcore [94] pour la construction d'organisation d'agents. Dans ce contexte, et en tenant compte de ce que nous avons vu dans la Section 3.1.4, KARMA (*Knowledgeable Agent Resources Manager Assistant*) [83, 82] joue le rôle du *middleware* permettant aux agents d'interpréter et de se conformer aux programmes TOP.

Teamcore est une infrastructure ré-utilisable et indépendante du domaine d'application supportant les spécifications faites avec TOP (cf. Section 3.1.4). La plate-forme fournit entre autre des "emballages" (*wrappers*) regroupés en une couche de mandataires (*proxies*), prêts à augmenter les capacités sociales des agents du domaine, et un assistant de gestion des ressources des agents nommé KARMA (cf. FIG. 4.2). Cet assistant de gestion fait partie de ce que nous considérons comme un système d'arbitrage. Nous y trouvons les *middle agents* regroupant des agents offrant des services particuliers comme un service de pages blanches (*Agent Naming Service*) ou un service de mise en relation entre un ensemble de capacités et un agent (*AMatchMaker*), etc. ...

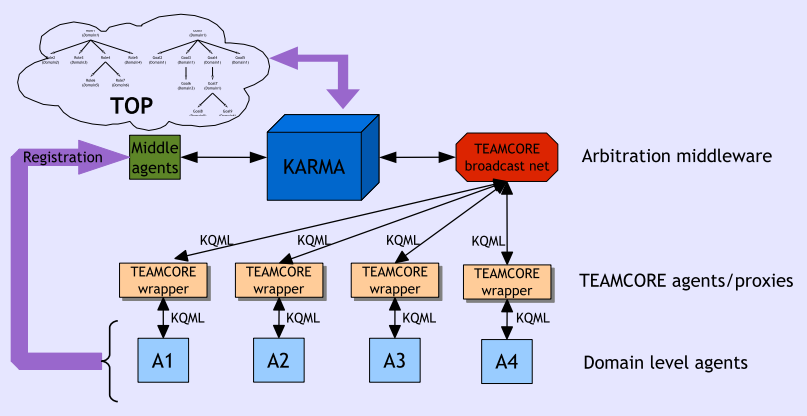


FIG. 4.2 – Cadre d'exécution des agents Teamcore avec le *middleware* KARMA (découlant de [94])

Comme mentionné dans la section 3.1.4 le programme TOP alloue des rôles organisationnels à des plans d'équipe. KARMA [82] déduit les conditions à remplir pour ces rôles au sein de l'organisation en se basant sur cette allocation. KARMA cherche alors les agents dont les capacités correspondent aux exigences ci-dessus. Pour cela KARMA dispose de plusieurs sources d'agents parmi les services de registre ou autres *middle agents*. De cette façon KARMA obtient une liste d'agents pertinents que le développeur du programme TOP assigne aux rôles dans l'organisation spécifiée[1]. KARMA vérifiera si cette allocation respecte les exigences du plan.

Au sein de la couche *proxy*, chacun des *wrappers* contient le module d'équipe de travail STEAM, étudié en Section 3.1.4, responsable du raisonnement sur les équipes de travail de type Teamcore. D'après le module STEAM, les *wrappers* génèrent automatiquement les actions de coordination requises en exécutant leurs tâches. Les règles STEAM leur permettent également de communiquer automatiquement entre eux afin d'assurer une exécution cohérente de leurs tâches, pour disséminer les informations importantes aux autres membres de l'équipe concernés et récupérer une stabilité après une erreur d'un membre. Le module d'interface d'un *wrapper* permet quant à lui de communiquer avec son agent du domaine en traduisant l'état de l'exécution de l'équipe en tâches individuelles.

Cette plate-forme permet de définir un ensemble de contraintes avec TOP et de faire en sorte qu'elles soient respectées par un ensemble d'agents hétérogènes sans imposer d'architecture. Pour ce

[1]KARMA peut également faire l'allocation des agents aux rôles automatiquement en choisissant le premier agent disponible possédant toutes les capacités exigées.

faire, les agents doivent comprendre l'organisation ; c'est le rôle des *wrappers*. KARMA permet de vérifier que les agents alloués à des rôles possèdent bien les capacités requises pour exécuter leurs tâches. Ici, nous n'avons pas de notion de normes à respecter. Les contraintes sont l'allocation des rôles sur des tâches. Si un agent joue un rôle, il a une tâche particulière à exécuter. La plate-forme ne se place pas dans un contexte d'agents autonomes. En fait, ce que nous nommons ici 'agent du domaine' pourrait très bien être autre chose qu'un agent. Les agents principaux ici sont les *wrappers*. Ils sont fournis par la plate-forme et doivent exécuter les règles STEAM et imposent leurs résultats aux agents qu'ils représentent. Ils ne peuvent donc pas ne pas respecter leurs contraintes, et, de ce fait, il n'existe pas de mécanisme de détection de violation ni d'application de sanction.

4.1.3 Contractual Agent Society

Les *Contractual Agent Societies* ou CAS [25] sont des Systèmes Multi-Agents permettant la coordination des comportements des agents au travers d'un ensemble de contrats sociaux[2] (également nommés *social norms*) renforcés par des mécanismes de contrôle social (*social institutions*).

La FIG. 4.3 représente une architecture conceptuelle d'une place de marché CNET [92] suivant les principes de CAS. La place de marché elle-même consiste en un ensemble d'agents homogènes se faisant mutuellement confiance incluant le *matchmaker*, le *socialization agent*, le *notary agent*, et le *reputation agent* et des contrats sociaux définissant les droits et les devoirs des agents.

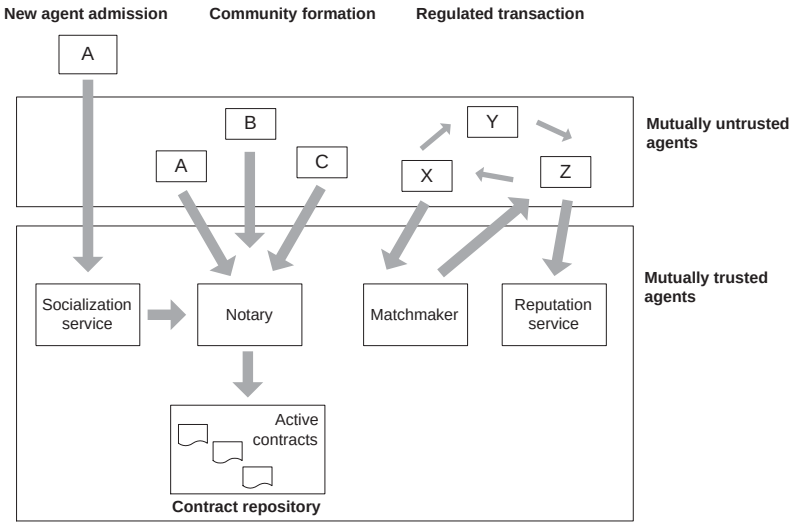


FIG. 4.3 – Architecture conceptuelle pour les places de marché CNET suivant CAS (provenant de [25])

Les contrats sociaux sont créés afin de spécifier les différents services de la place de marché (les protocoles et les politiques de contrôle social par exemple) ainsi que les partenariats entre les agents eux-mêmes. Pour cela les agents provenant de l'extérieur (potentiellement hétérogènes et non sûrs) doivent tout d'abord négocier un contrat social avec le *socialization agent*. Puis ils communiquent entre eux afin de négocier un nouveau contrat social qui va définir leur partenariat comme une communauté dans la place de marché. Le *notary agent* sert de médiateur à la négociation entre les agents et devra stocker le contrat et résoudre les éventuels conflits.

Le nouveau contrat définit les termes du partenariat mais hérite également de toutes les politiques de gestion de la place de marché CNET telles que les sanctions imposées pour l'annulation ou la violation d'un contrat. Le *notary agent* supervise le déroulement des contrats en mettant en place un

[2]La définition des contrats sociaux CAS est disponible en Annexe A.1.1.

mécanisme de contrôle social. Ce mécanisme définit les différentes classes d'exception (les déviations par rapport au comportement "normal" convenu) et peut définir des sanctions pour certaines ou toutes ces exceptions. Il appliquera les sanctions prescrites en cas d'annulation ou de violation de contrat.

Aucune architecture d'agent n'est imposée par le cadre d'exécution CAS car c'est un système ouvert donc n'importe quel agent provenant de n'importe quel système et développé sans standard peut intégrer une CAS. Cependant une sentinelle est associée à chaque agent lors de son enregistrement pour intégrer le système. Ces sentinelles jouent le rôle de superviseur d'engagements en observant et en influençant le comportement des agents en fonction des contrats sociaux qu'ils doivent respecter et afin de garantir le bon fonctionnement du système dans son ensemble. Chaque sentinelle est un interpréteur des graphes d'état-transition composant les contrats (cf. Section A.1.1).

Ces travaux restent en grande partie théoriques. En 2000 lors de la parution de l'article [25], seules les sentinelles existaient et servaient de contrôle social. Depuis, les travaux n'ont semble-t-il pas beaucoup évolué. Bien que la contrainte à base de contrats sociaux soit limité (pas de lien avec d'autres spécifications structurelles ou fonctionnelles par exemple) ce modèle inclut un superviseur, sous les traits du notaire, détectant et sanctionnant le non respect des contrats. La détection se fait sur l'évolution du graphe d'état-transition associé. Un seul notaire doit gérer la vérification de la création de tous les contrats, leur stockage, la détection d'éventuelles violations et l'application des sanctions. Dans la section suivante, nous présentons une architecture utilisant également les contrats comme spécification de contrainte et un notaire pour la création de contrats mais au lieu d'avoir un superviseur par agent, l'architecture en fournit un par contrat.

4.1.4 Organisations Virtuelles

Les travaux de Henrique Cardoso et Eugénio Oliviera [18, 87, 17, 16] utilisent les contrats pour spécifier les comportements d'organisations différentes et distribuées afin de créer des Organisations Virtuelles (VO). Pour cela ils se basent sur la définition des contrats de Matthias Sallé [88] (cf. Annexe A.1.2) et se servent d'une Institution Électronique pour gérer la coordination des flux de données (*workflow*) de chaque organisation.

Une Organisation Virtuelle est considérée par les auteurs comme un consortium temporaire de différentes organisations individuelles qui coopèrent pour atteindre un but commun. L'Institution Électronique est vue dans ce cas là comme un cadre d'exécution assistant le cycle de vie de l'Organisation Virtuelle. Elle fournit un ensemble de services institutionnels couvrant les fonctions de formation et d'action des Organisations Virtuelles. Ces services composent un système de coordination qui assiste les interactions d'agents logiciels représentant les différentes organisations individuelles.

La Fig. 4.4 représente les principaux services fournis par des agents disponibles dans l'Institution Électronique. Les services de *Negotiation mediation* permettent d'assister la formation d'une Organisation Virtuelle. Cela comprend l'utilisation des protocoles de négociation et des *templates* de contrats appropriés. L'instanciation des *templates* de contrat est le résultat du processus de négociation. Afin d'avoir une négociation compréhensible par les agents, un service de concordance des ontologies (*Ontology matching*) est également mis en place. Les contrats sont enregistrés dans l'Institution Électronique via le service de Notaire (*Notary*) qui est responsable de leur validation en fonction des normes institutionnelles. Ensuite, un service de *Contract Monitor* (un par contrat) assiste l'exécution des engagements contractuels par chaque partenaire de l'Organisation Virtuelle.

L'agent *IO-WfM* a pour but de superviser, d'un point de vue inter-organisationnel, l'exécution du plan de travail défini dans chaque contrat en interagissant avec les partenaires responsables de l'exécution de la tâche. Les compétences de l'agent *IO-WfM* lui permettent entre autre de : (i) synchroniser le *workflow* inter-organisationnel, c'est à dire déterminer si le processus est prêt à effectuer la prochaine tâche en imposant une synchronisation parmi les *workflows* individuels ; (ii) re-diriger des

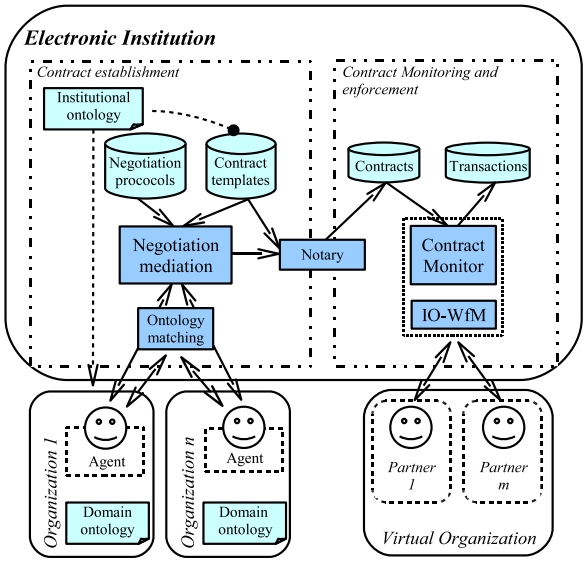


FIG. 4.4 – Services d'une Institution Électronique (provenant de [16])

étapes du *workflow* suite à l'apparition de situations inattendues ou à des tâches non-accomplies ; (iii) donner un retour à propos de l'exécution du plan de travail du contrat en condition réelle fournissant ainsi des indications pouvant avoir une répercussion sur la re-définition du contenu des contrats futurs.

L'Institution Électronique est également responsable du contrôle du respect des contrats. Elle peut appliquer des sanctions dans le cas où un agent ne tient pas compte des contrats. Ces sanctions doivent par contre être prévues dans le contrat ou définies institutionnellement. Toutefois, et pour que ce genre de mécanisme soit efficace, l'Institution Électronique garde en mémoire une réputation mise à jour en fonction des performances de chaque participant aux contrats.

Ce modèle de supervision des contraintes fonctionne bien en tant que superviseur mais intègre tous les inconvénients de l'utilisation des contrats sociaux comme moyen de contraindre les agents. En effet, les contrats (basés sur ECF) sont définis autour d'un opérateur déontique utilisant la référence à deux agents et à une action à effectuer. Les agents ne sont pas organisés et l'action à faire n'est pas structurée. Après les différents points de vue de contraintes que nous avons vues dans le chapitre précédent, cette spécification est uniquement normative. Un contrat est un ensemble de normes, et pour sanctionner une norme non respectée, il suffit de détecter le passage du graphe associé à un opérateur dans un état non voulu. Cependant, il n'est pas précisé comment l'état des opérateurs des normes évolue et grâce à qui. Une autre façon de faire est d'empêcher que l'agent autonome n'enfreigne les contraintes par l'impossibilité d'exécuter les actions non autorisées. C'est le principe présenté dans la section suivante.

4.1.5 $\mathcal{S}-\mathcal{M}$oise$^+$

$\mathcal{S}-\mathcal{M}$OISE$^+$ [60] est un *middleware* gérant les organisations $\mathcal{M}$OISE$^+$ (cf. Section 3.1.5). Il fournit à chaque agent évoluant dans la société une *OrgBox* considérée comme une vue partielle de l'organisation. Il sert d'interface entre les agents hétérogènes provenant de l'extérieur et l'organisation. Il tente ainsi de réduire l'espace entre les contraintes organisationnelles et l'autonomie des agents. Ce logiciel garantit que tous les agents suivent l'organisation et respectent les contraintes sans qu'ils soient développés dans un langage spécifique ou suivant une architecture particulière.

Le middleware organisationnel $\mathcal{S}-\mathcal{M}$OISE$^+$ est composé de deux principaux éléments comme

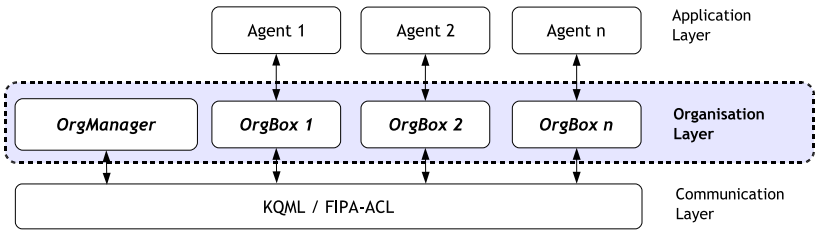


FIG. 4.5 – Composants de l'architecture $\mathcal{S}-\mathcal{M}$OISE$^+$(provenant de [60])

illustré sur la FIG. 4.5. L'agent *OrgManager* maintient l'état cohérent et consistant de l'entité organisationnelle (OE) (cf. Section 3.1.5). C'est lui qui agit directement sur l'entité organisationnelle afin de créer les groupes et les schémas, qui assignent les rôles et les missions aux agents (sur demande des agents). Il reçoit aussi les messages des agents voulant agir sur l'OE via leur *OrgBox* respective et effectue les changements seulement s'ils ne violent pas les contraintes organisationnelles.

Une *OrgBox* est l'interface que les agents utilisent pour accéder à la couche organisationnelle et de ce fait la couche de communication. Elle peut être comparée au *TEAMCORE wrapper* permettant d'accéder au middleware KARMA (cf. Section 4.1.2) ou à l'agent *IO-WfM* des organisations virtuelles vues dans la section précédente permettant d'agir sur les contrats. L'*OrgBox* doit être utilisée pour agir sur l'OE (adopter un rôle, s'engager sur une mission, satisfaire un but, etc.), envoyer un message à un autre agent ou connaître l'état de l'OE. L'*OrgBox* est informée par l'*OrgManager* quand l'état d'un schéma lié à l'*OrgBox* de l'agent change. L'*OrgBox* permet donc de ne pas imposer une architecture particulière aux agents évoluant dans l'organisation même s'il faut faire en sorte que l'agent puisse interagir avec son *OrgBox* et comprenne le fait d'adopter un rôle avant de pouvoir s'engager sur des missions pour atteindre des buts.

$\mathcal{S}-\mathcal{M}$OISE$^+$ permet donc de faire en sorte que les agents suivent les contraintes spécifiées par l'organisation (l'OS) notamment le respect des cardinalités, des liens entre rôles et les normes. $\mathcal{S}-\mathcal{M}$OISE$^+$ rend possible l'externalisation et l'interprétation de l'organisation, empêchant ainsi le codage des contraintes au sein même des agents. La supervision de l'organisation se fait par contre par un seul *OrgManager*. De plus, les expressions déontiques sont imposées aux agents sans leur laisser le choix de ne pas respecter ces expressions ou toute autre contrainte. De ce fait, des violations ne peuvent pas être détectées et encore moins être sanctionnées puisqu'elles ne peuvent pas avoir lieu. Un système de détection de violation ainsi que l'application de sanction s'avèrent donc pour le moment inutiles.

4.2 Systèmes de gestion d'Institution

Nous étudions dans cette section le système de gestion du modèle OMNI ainsi qu'AMELI mettant en œuvre les institutions définies avec ISLANDER.

4.2.1 OMNI

La plate-forme OMNI (Organizational Model for Normative Institutions) [32, 33, 104] est censée couvrir d'après Virginia Dignum et Javier Vázquez-Salceda tous les aspects qu'un Système Multi-Agents ouvert ou non doit pouvoir fournir.

OMNI fait la différence entre deux sortes d'agents : les agents institutionnels jouant les rôles d'aménagement de la plate-forme en offrant des services particuliers et définis dans les *templates* d'architecture mis à disposition pour spécifier des structures de rôles et d'interactions (cf. la Section 3.2.2) et les agents externes jouant les rôles opérationnels et qui devront suivre et respecter les

contraintes organisationnelles et normatives.

Selon le *template* choisi pour spécifier une organisation avec OMNI, certains agents institutionnels seront présents et aideront les agents externes à évoluer au sein de l'organisation. Les *templates* fournis définissent au choix un marché avec des agents de *matchmaking*, de réputation et d'identification et un *market master*, un réseau avec un agent de *matchmaking* également, un garde-barrière, un notaire et un agent de *monitoring*, etc. Les agents institutionnels vont en fait jouer ces rôles spéciaux.

Les contrats d'interactions instancient les structures d'interactions et définissent ainsi les liens de communication réels entre agents. Ces contrats sont définis en LCR (cf. Annexe A.1.3) et sont basés sur des diagrammes d'états/transitions. Ces diagrammes permettent le contrôle de l'exécution des normes. En effet, la détection d'une violation de norme se fait quand le diagramme atteint un état d'exception définissant également la sanction à appliquer. Un type spécial d'agents est en charge de l'exécution des normes : les *Police Agents* qui sont une extension du concept des *Guardian Agents* proposé par Fox et Das [45].

L'approche d'OMNI, définissant des rôles que les agents externes devront jouer et des protocoles qu'ils devront suivre, permet de ne pas avoir à faire des hypothèses sur leur architecture afin qu'ils aient les capacités de s'organiser. En effet, la définition des ontologies avec OMNI permet aux agents de comprendre l'organisation. Les agents institutionnels contrôlent le respect des contraintes en fonction des comportements des agents externes. Ils les perçoivent comme des boîtes noires et ne les considèrent que par rapport à l'évolution des contrats sur lesquels ils sont engagés.

4.2.2 AMELI

Le système d'Institution Électronique IDE-eli (*Integrated Development Environment for Electronic Institutions*) [90] permet de définir de manière centralisée une Institution Électronique avec ISLANDER, d'implémenter les agents qui y participeront avec aBUILDER, de simuler l'exécution avec SIMDEI et enfin d'exécuter et contrôler le fonctionnement de l'Institution Électronique avec AMELI.

AMELI [39, 7] est une plate-forme logicielle pour exécuter les Institutions Électroniques spécifiées avec le langage ISLANDER. La plate-forme facilite la participation des agents dans l'institution en faisant en sorte qu'ils respectent les conventions institutionnelles. AMELI fait partie de l'architecture du système d'Institution Électronique en tant que couche sociale comme illustré sur la Fig. 4.6, servant ainsi d'intermédiaire aux agents de la couche agent dans leurs interactions via la couche de communication. Nous avons vu dans le chapitre précédent qu'une Institution Électronique spécifiée avec ISLANDER était basée essentiellement sur une définition des interactions. C'est donc logiquement que nous trouvons AMELI comme système d'arbitrage afin de contrôler que les interactions entre les agents respectent bien les protocoles des scènes, leurs déplacements entre les scènes, etc.

Comme nous pouvons le voir sur la Fig. 4.6, AMELI est constitué d'un ensemble d'agents jouant les médiateurs. Il y a quatre types d'agents différents :

- *Institution Manager* (IM) ; il est en charge de démarrer l'Institution Électronique, d'autoriser les agents à entrer et il ne peut y en avoir qu'un par Institution Électronique.
- *Transition Manager* (TM) ; il est en charge de gérer les transitions en contrôlant les mouvements des agents entre les scènes. Il y a un *Transition Manager* par transition.
- *Scene Manager* (SM) ; il est responsable de l'exécution d'une scène et contrôle que les agents suivent le protocole défini pour chaque scène. Il y a un *Scene Manager* par exécution de scène.
- *Governor* (G) ; il joue l'intermédiaire entre un agent externe (provenant de la couche agent) et l'institution à laquelle il participe. Il y a un *Governor* par agent externe.

Au sein d'AMELI, ces agents sont séparés en deux groupes, les agents publics, accessibles par les agents externes, c'est-à-dire les *Governors* et les agents privés, non accessibles de l'extérieur et

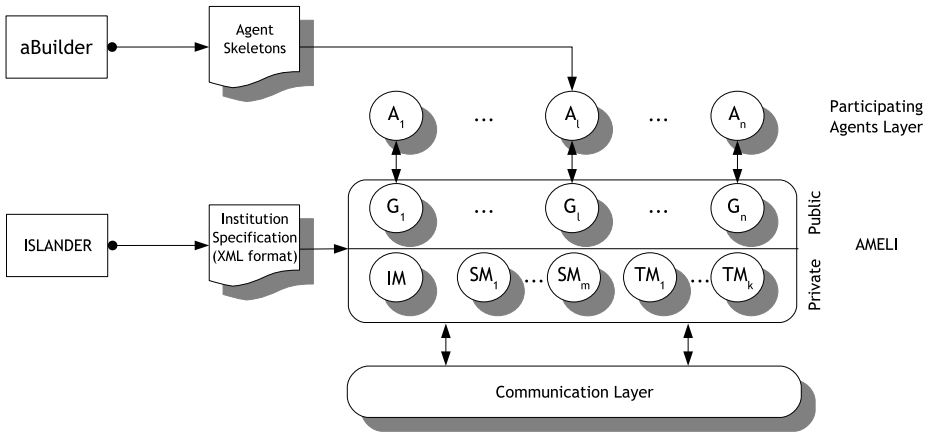


FIG. 4.6 – Architecture du système d'Institution Électronique IDE-eli (provenant de [7])

composés par les autres agents. Les agents externes peuvent être construits à l'aide de l'outil aBUILDER. Ce dernier est capable de générer une structure d'agent à partir de la spécification du rôle que l'agent jouera et des interactions auxquelles il participera. Cette génération définit ce que les agents peuvent faire. Le développeur leur ajoutera les mécanismes de décision leur permettant d'être des agents autonomes.

AMELI propose donc un ensemble de *Managers* capables de superviser chacun une partie de la spécification ISLANDER plus un ensemble de *Governors* permettant de faire en sorte que les agents venant agir au sein de l'institution puissent interpréter la spécification ISLANDER et interagir comme il se doit avec les autres agents. Cette spécification concerne principalement les interactions entre agents, de ce fait, ne peuvent être supervisés que le respect des protocoles dans les scènes et le respect des transitions entre scènes. Il n'y a donc pas de contrôle sur les actions que peuvent avoir les agents. Même si AMELI en tant qu'IAS d'institution électronique fournit l'arbitrage de sa spécification, cette dernière n'est pas complète (voir chapitre précédent). De plus, la structure de l'organisation, pourtant spécifiée avec ISLANDER, n'a pas de *Manager* associé et n'est donc pas supervisée.

4.3 Synthèse

Nous venons de voir les différents IAS qui existent dans le domaine des SMA supervisant un ensemble d'AAI contraints à l'aide des modèles SIM que nous avons vus dans le chapitre précédent. L'IAS et les AAI nous donnent donc deux critères de comparaison. De plus, pour l'IAS, les points caractéristiques sur lesquels nous nous basons sont les services proposés, leurs liens avec le SIM et la rigidité de ce lien. Pour les agents autonomes, nous pouvons comparer si leur architecture est imposée ou non (s'ils sont hétérogènes), la façon dont ils peuvent interpréter et tenir compte des spécifications et enfin leurs liens avec le système d'arbitrage. Nous n'étudierons pas les liens entre les agents et la spécification de contraintes puisque la plupart du temps, les agents n'accèdent pas directement à la modélisation faite avec le SIM.

4.3.1 Système d'Arbitrage d'Institution

L'IAS est plus qu'un simple système de contrôle surveillant que les contraintes soient respectées, c'est aussi une façon d'instancier et d'exécuter les spécifications de structure, de fonctionnalités et de normes en faisant intervenir un ensemble d'agents. Pour cela des services sont offerts par des agents spécialisés ou directement par la plate-forme d'exécution. Comme repris dans le TAB. 4.1, ces agents se contentent d'offrir des facilités aux agents externes pour évoluer au sein de la société, comme un

système de pages jaunes ou de pages blanches, un système de matchmaking, etc. comme le font les agents et services de Teamcore, CAS, ECF/VO et OMNI. Par contre, certains systèmes possèdent des agents gérant également la société en fonction des spécifications. C'est le cas de MadKit avec les *group* et *role managers* entre autre, de l'agent IO-WfM et du *contract monitor* pour le système d'organisations virtuelles, l'*OrgManager* pour $\mathcal{S}-\mathcal{M}$OISE$^+$ et les *Managers* en général pour OMNI. Cela ne signifie pas pour autant que les autres systèmes ne fournissent pas de moyen de se conformer aux spécifications. En effet, la deuxième ligne du tableau reprend le module de l'IAS en lien avec le SIM et permettant réellement de l'arbitrer. Tous les systèmes possèdent un service ou un agent dédié, sauf AGR. Les AAI accèdent directement au SIM sans arbitrage. Tous ces modules permettent de gérer l'accès des AAI à la société sur base du SIM. KARMA sélectionne une liste d'agents éligibles selon les spécifications afin d'atteindre le but de la société, contrairement aux autres où ce sont les AAI qui font, directement ou indirectement, appel à ces modules.

IAS	AGR/MadKit	TOP/Teamcore	CAS	ECF/VO	Moise+/S-Moise+	OMNI	Islander/Ameli
Services	System agents	Middle agents	2 agents + 2 services	1 agent + 4 services	1 agent + 1 service	Institutional agents	4 agents
Module d'Arbitrage		KARMA	Notaire	Contract Monitor	OrgManager	Police Agents	Managers
Degré d'Arbitrage	Contrôle	Direction	Supervision	Supervision	Contrôle	Supervision	Contrôle

TAB. 4.1 – Tableau comparatif des Systèmes d'Arbitrage

L'absence d'information pour MadKit, en ce qui concerne le module d'arbitrage du SIM pour les IAS, est due au fait que nous considérons que les informations du SIM sont implémentées directement au sein des agents. C'est en fonction de leurs spécifications internes que les AAI vont créer des groupes et des rôles et que d'autres agents les rejoindront. Le lien entre les spécifications et le système d'arbitrage se faisant par l'implémentation des agents, les spécifications ne peuvent ni être arbitrées et ne peuvent ni être violées. Nous sommes face à un contrôle dans le sens où celui-ci impose un comportement aux agents tandis que la supervision laisse le choix à l'agent mais le sanctionne en cas de non respect. Concernant Teamcore, les AAI vont être dirigés dans la mesure où KARMA n'attend pas qu'un agent agisse ou demande l'autorisation d'agir. C'est KARMA qui ordonne aux agents éligibles d'agir. Le degré d'arbitrage se situe au niveau du contrôle pour $\mathcal{S}-\mathcal{M}$OISE$^+$ et AMELI. Les autres modèles fournissant dans leurs spécifications la possibilité de définir des sanctions, leur IAS respectif met en place des services de détection et de sanction de violation sous les traits d'un *contract monitor* pour le système d'organisations virtuelles et de *police agent* pour OMNI. Le module d'arbitrage met donc en place ici une supervision. Les travaux ne disent cependant pas comment en ce qui concerne CAS.

4.3.2 Agents Autonomes

CAS est censé permettre à des agents hétérogènes de joindre la société d'agents à condition de passer au préalable un contrat avec cette même société. Afin de pouvoir interpréter correctement les spécifications des contrats et afin d'avoir les capacités à négocier les contrats avec la société et avec les autres agents, une sentinelle est fournie à chaque agent rejoignant le système. Afin que les agents puissent interpréter les spécifications et puissent interagir avec les autres agents, trois solutions sont possibles, comme reporté sur le TAB. 4.2 : intégrer le système d'interprétation dans l'agent (MadKit), lui fournir le moyen de comprendre à l'aide des ontologies (VO, OMNI, AMELI), imposer une interprétation toute prête (Teamcore) ou faire en sorte que chaque agent ait son propre interpréteur (CAS, $\mathcal{S}-\mathcal{M}$OISE$^+$). La solution adoptée par MadKit impose d'avoir des AAI capables d'interpréter directement le SIM. Cela enlève forcément une part d'hétérogénéité aux agents. Tandis que les agents interagissant au sein d'un cadre d'exécution Teamcore sont hétérogènes avec un ensemble de capacités différentes. Cependant, leur intervention dans l'organisation se fait si une tâche leur correspond et que KARMA les considère les plus aptes à l'exécuter. Leur autonomie est réduite

dans le sens où KARMA leur impose le comportement à adopter. Chaque AAI hétérogène d'une organisation Teamcore doit passer par un *wrapper* afin d'interpréter et de simplifier les échanges de messages entre l'AAI et KARMA. Comment être sûr que les AAI sauront également communiquer avec l'interpréteur ? L'utilisation des ontologies permet de définir syntaxiquement les actions de l'IAS accessibles aux agents. Ensuite, le système d'Organisations Virtuelles par exemple permet de mettre en correspondance l'ontologie d'un agent hétérogène avec l'ontologie de l'Institution Électronique.

AAI	AGR/MadKit	TOP/Teamcore	CAS	ECF/VO	Moise+/S-Moise+	OMNI	Islander/Ameli
Architecture	Imposée	Libre	Libre	Libre	Conseillée	Libre	Conseillée
Interprétation de la SIM	Intégrée	KARMA	Sentinelle	Ontologie matching	OrgBox	Ontologie	Ontologie
Accès à l'IAS	Group/Role Manager	Teamcore wrapper + STEAM	Sentinelle	IO-WfM agent	OrgBox	Institutional agents	Governor

TAB. 4.2 – Tableau comparatif des Agents Autonomes agissant au sein d'Institution (AAI)

Nous mentionnions précédemment que le lien entre les AAI et le SIM ne sert pas de point de comparaison faute d'existence de celui-ci. En effet, les agents ont rarement de lien direct avec le SIM. Afin que le SIM soit supervisé et son respect contrôlé, il faut généralement un intermédiaire représenté ici par l'IAS. La plupart des systèmes étudiés fournissent une composante permettant aux agents de pouvoir interagir avec l'IAS. C'est le cas pour MadKit mettant en place des agents spéciaux (ici le *role manager* et *le group manager* permettant d'accéder à l'Institution), pour Teamcore avec ses *Teamcore Wrappers*, pour CAS avec les sentinelles, pour le modèle d'Organisations Virtuelles avec l'agent IO-WfM, pour $\mathcal{S}-\mathcal{M}$OISE$^+$ avec l'OrgBox et pour AMELI avec les Governors. Les AAI plus ou moins hétérogènes peuvent ainsi en interprétant le SIM via l'IAS s'organiser. L'interprétation de le SIM et le lien avec l'IAS se fait grâce à un seul élément pour $\mathcal{M}$OISE$^+$ (OrgBox) et CAS (sentinelle). OMNI, AMELI, VO utilisent les ontologies pour interpréter le SIM. Le composant faisant le lien avec l'IAS intégrant l'ontologie, cela équivaut à avoir un seul élément. Cependant, et comme dit précédemment, le *matching* entre l'ontologie de l'institution et l'ontologie de l'AAI est fait seulement par VO. KARMA quant à lui dirige les agents externes. L'interpréteur ne laisse donc pas le choix à l'AAI, il décide lesquels vont intervenir sans être pour autant le contact direct avec chaque AAI. Il laisse cela aux *Teamcore Wrappers*.

Chapitre 5

Synthèse

Nous avons vu dans le Chapitre 2 la définition que donne Douglass North [76] des Institutions à savoir le moyen de définir les règles du jeu d'une société par un ensemble de contraintes. Nous étendons cette définition et considérons une ***Institution Électronique*** basée sur les agents comme étant une *organisation* d'agents autonomes (***Acteurs Autonomes de l'Institution*** ou ***AAI***) dans laquelle leurs comportements sont gouvernés par des *normes* et contrôlés par un *système d'arbitrage* (***Système d'Arbitrage d'Institution*** ou ***IAS***). L'organisation normée est définie grâce à un ***Modèle de Spécification d'Institution*** (***SIM***).

À l'aide de tout ce que nous venons de voir dans cette première partie, nous modifions la Fig. 3.11 de la synthèse du Chapitre 3 à l'aide des éléments de la synthèse du Chapitre 4 afin d'obtenir le modèle conceptuel d'une architecture d'une Institution Électronique rassemblant les différents éléments de l'état de l'art, représentée par la Fig. 5.1.

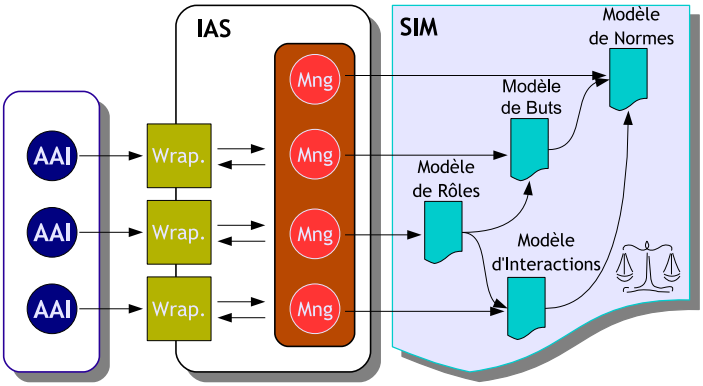


Fig. 5.1 – Modèle conceptuel d'une Institution Électronique

De manière générale, une Institution Électronique spécifie un ensemble de contraintes. Les agents voulant évoluer au sein de la société d'agents doivent rejoindre l'Institution et donc se conformer aux contraintes. Pour cela et comme étant la plupart du temps hétérogènes, ils doivent interagir avec l'Institution via un *wrapper* servant d'interface entre l'agent et l'Institution. Ensuite, ils doivent utiliser un ensemble de services pour être autorisés à agir au sein de la société (soit directement, soit via le système d'arbitrage). Pour ce faire, le système d'arbitrage se réfère à la spécification de l'Institution afin de faire en sorte que l'évolution de la société évolue dans un état prévu par la spécification.

Nos objectifs sont, d'une part, de pouvoir faire en sorte qu'un ensemble d'agents autonomes et hétérogènes soit capable de raisonner sur un SIM spécifié de façon explicite et déclarative, afin de

décider s'il la respectera ou non. D'autre part, nous voulons que l'IAS ainsi que les liens qu'il a avec le SIM et les agents autonomes permettent un arbitrage et un renforcement des contraintes grâce à la mise en place de sanctions et à une ouverture de l'organisation des agents à d'autres agents hétérogènes. Nous avons de ce fait étudié un ensemble de modèles correspondant à l'une ou l'autre des composantes d'une Institution Électronique dans les deux précédents chapitres afin de voir lesquels répondaient à nos attentes.

Modèle de Spécification d'Institution

Le Modèle de Spécification d'Institution doit spécifier explicitement l'organisation normative des agents de l'application selon différentes dimensions ajoutant ainsi une part importante de flexibilité. Ceci permet également à tout agent hétérogène de pouvoir prendre en compte les contraintes et agir au sein de l'organisation.

Les différentes dimensions de spécification que nous avons identifiées et qui sont reprises sur la Fig. 5.1 ne sont pas présentes dans tous les modèles étudiés dans l'état de l'art. La définition d'un ensemble de rôles structurant les agents est prise en compte par AGR, STEAM et $\mathcal{M}$OISE$^+$. Cependant, en plus de définir une hiérarchie de rôles (STEAM), de rassembler les rôles en groupes et de définir une contrainte de compatibilité entre rôles obligeant un agent à jouer deux rôles (AGR), $\mathcal{M}$OISE$^+$ permet d'autres contraintes telles les cardinalités et la définition d'autres liens entre les rôles.

La définition d'un ensemble de buts spécifiant le fonctionnement des agents est prise en compte par TÆMS, STEAM et $\mathcal{M}$OISE$^+$. Ce dernier ajoute la notion de mission regroupant un ensemble de buts (ou de tâches) et la possibilité de définir le nombre d'agents autorisés à s'engager sur les missions. De plus, $\mathcal{M}$OISE$^+$ permet de définir si les sous-buts d'un but doivent être atteints de manière séquentielle, en parallèle ou selon un choix.

La définition d'un ensemble de scènes d'interactions dans lesquelles se trouvent les agents est prise en compte par OMNI et ISLANDER. Cependant, la façon dont ces deux modèles spécifient un enchaînement de scènes contraint seulement les agents individuellement au lieu de l'organisation dans son ensemble. ISLANDER ne permet pas à ses agent d'évoluer dans plusieurs scènes en même temps. Pour notre part, nous considérons plutôt une scène comme un contexte influençant le comportement d'un agent ou d'un groupe d'agents, l'obligeant à respecter certains protocoles (de fonctionnement et de communication) et pouvant être actif en même temps que d'autres contextes. Aucun des modèles étudiés précédemment ne permet ceci.

La définition d'un ensemble de normes réglant le comportement des agents est prise en compte par OMNI, ISLANDER et $\mathcal{M}$OISE$^+$. Cependant, bien qu'ayant une dimension normative, OMNI ne définit pas un ensemble de normes dans une spécification à part. Chaque définition de norme se trouve au sein de la définition d'un rôle ou d'une scène d'interaction. Quant aux normes de ISLANDER et de $\mathcal{M}$OISE$^+$, elles ne définissent pas de sanction, ni d'émetteur de la norme, ni de contrainte temporelle pour ISLANDER, et ce dernier ne permet pas de différencier les obligations des permissions.

$\mathcal{M}$OISE$^+$ mis à part, les modèles étudiés dans le Chapitre 3 ne nous conduisent pas à une vision homogène et complète de la définition du comportement d'une organisation d'agents qui la rendrait explicite et permettrait ainsi une supervision par un IAS. Cependant, $\mathcal{M}$OISE$^+$ ne satisfait pas nos objectifs dans sa définition des normes (ou expressions déontiques) en n'intégrant pas de sanction, de plus, il ne présente pas de modèle d'interaction.

Système d'Arbitrage d'Institution

Le Système d'Arbitrage d'Institution doit, selon nous, être générique et flexible afin de permettre la supervision de tout type d'Institution Électronique. Comme illustré sur la Fig. 5.1, il est constitué d'un ensemble d'entités institutionnelles dont les capacités permettent de superviser le respect de chaque spécification (c'est-à-dire les différentes dimensions de SIM). Pour cela chaque agent organisé

et supervisé par l'Institution Électronique est associé à une entité permettant à l'agent de s'intégrer à l'organisation et d'interpréter le SIM, tandis que chaque modèle de spécification du SIM est associé à une entité permettant de superviser son respect. Ces deux sortes d'entités interagissent afin de faire en sorte que les agents se comportent convenablement sur l'organisation et afin de détecter et de sanctionner les violations du SIM.

La plupart des systèmes de gestion étudiés dans l'état de l'art ne possède qu'un module unique d'arbitrage pour toute l'organisation, à savoir KARMA pour Teamcore, un Notaire pour CAS, un *Contract Monitor* pour le système d'Organisations Virtuelles et enfin un OrgManager pour $\mathcal{M}$OISE$^+$. OMNI fournit des agents *de police* et ISLANDER des *Managers*.

Le SIM doit pouvoir définir en plus des spécifications les sanctions encourues en cas de non respect comme le fait OMNI par exemple. Cette partie doit être prise en compte par les agents afin de savoir ce qui leur en coûterait de violer une norme et par l'IAS afin de réellement appliquer les sanctions en cas de violation. Ce n'est pas le cas du modèle Teamcore qui dirige l'Organisation en choisissant à un moment précis quel agent doit agir, ou à $\mathcal{S}-\mathcal{M}$OISE$^+$ et AMELI qui se servent du modèle de spécification d'institution pour contrôler le fonctionnement des agents. Ce contrôle entraîne le fait que les agents sont obligés de respecter les contraintes réduisant ainsi l'autonomie des agents externes et de ce fait leur capacité à enfreindre les contraintes.

Aucun système d'arbitrage étudié dans le Chapitre 4 ne répond à nos objectifs car aucun ne permet la supervision individuelle des différents modèles de spécification en même temps qu'une prise en charge des agents hétérogènes voulant être organisés. De plus ces systèmes ne définissent pas explicitement la tâche d'arbitrage permettant l'ouverture à d'autres agents que ceux fournis par le système aux rôles de superviseurs ni la personnalisation de l'activité d'arbitrage.

Interdépendance AAI, SIM et IAS

Si nous prenons en compte la synthèse des modèles de spécification et d'arbitrage d'institution que nous avons faite dans les chapitres précédents, nous en déduisons que ce sont $\mathcal{M}$OISE$^+$/$\mathcal{S}-\mathcal{M}$OISE$^+$, OMNI et ISLANDER/AMELI qui répondent le mieux à notre problématique. Cependant ils n'atteignent pas entièrement nos objectifs. En effet, $\mathcal{M}$OISE$^+$/$\mathcal{S}-\mathcal{M}$OISE$^+$ et ISLANDER/AMELI ne laissent pas la possibilité aux agents de ne pas respecter les contraintes et donc, comme le prévoit OMNI, d'appliquer une sanction. Cela réduit l'autonomie des agents externes. ISLANDER/AMELI et OMNI ne permettent pas de spécifier une réelle organisation des agents avec une structure riche et détaillée ni un ensemble de fonctionnalités qui seront ensuite attribuées aux éléments de la structure. Enfin, aucun de ces modèles ne permet de définir des contextes de contraintes différents ayant une influence de plus haut niveau sur les comportements des agents. Cependant, au lieu de définir un ensemble de contextes vus comme des états dans lesquels serait l'organisation, il faudrait voir les contextes comme un moyen d'adapter à différentes situations les expressions normatives. Par exemple une norme obligeant une action dans un contexte ne sera peut être pas valable dans un autre contexte.

Le SIM est axé seulement sur les interactions (GAIA, AGR, ISLANDER) ou sur les tâches (STEAM, TAEMS, $\mathcal{M}$OISE$^+$) des AAI et rarement sur les deux (OMNI). Il faudrait pour cela définir différentes dimensions indépendantes des AAI (contrairement à AGR ou OMNI) et de l'IAS et définissant tous les aspects du comportement des acteurs d'une institution. Une autre remarque que nous pouvons faire concerne le degré d'arbitrage de l'IAS. Il est souvent trop contraignant puisqu'il s'agit de contrôle pour MAdKit, $\mathcal{S}-\mathcal{M}$OISE$^+$ et AMELI et de direction (dans le sens de 'diriger') pour KARMA.

Deuxième partie

$\mathbf{Ma}\mathcal{B}_{\mathbf{eli}}$

Dans la partie précédente, nous avons fait un état de l'art sur les différentes composantes d'une Institution Électronique. Nous avons en effet vu que, dans le domaine des Systèmes Multi-Agents, des modèles existent pour contraindre des agents autonomes et pour les contrôler. Par rapport à notre spécification des besoins en terme d'ouverture, de flexibilité et de contrôle, les systèmes étudiés ne répondent cependant pas à nos objectifs. Nous proposons donc notre propre modèle d'Institution Électronique : MA$\mathcal{B}_{\text{ELI}}$.

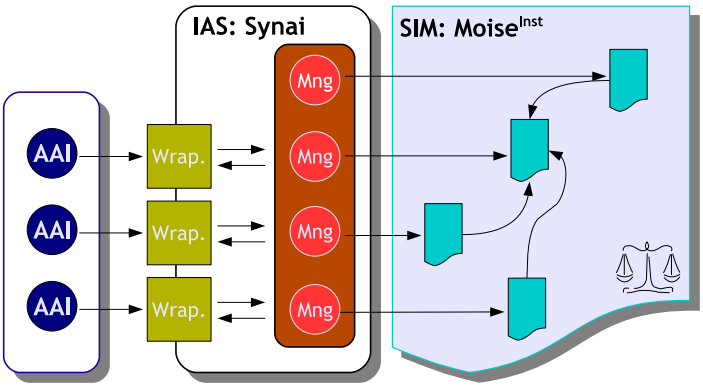


FIG. 5.2 – Vision globale de MA$\mathcal{B}_{\text{ELI}}$

Comme nous venons de le voir dans la partie précédente et comme illustré sur la FIG. 5.2, ce modèle est composé d'un modèle de spécification d'institution (SIM) nommé $\mathcal{M}$OISE^{Inst} et d'un système d'arbitrage (IAS) nommé $\mathcal{S}$YNAI afin de contraindre et de superviser un ensemble d'agents autonomes (AAI) propres à chaque domaine d'application. Un de nos objectifs étant l'ouverture, nous désirons que n'importe quel agent puisse rejoindre l'Institution Électronique à condition qu'il sache comment interagir avec $\mathcal{S}$YNAI. C'est pour ceci que nous ne proposerons pas d'architecture d'agent.

$\mathcal{M}$OISE^{Inst} répond à notre problématique tout en comblant les carences des modèles précédemment étudiés. $\mathcal{M}$OISE^{Inst} permet de spécifier de façon explicite et déclarative des organisations que les agents pourront ou non prendre en compte. Pour cela, en plus des dimensions structurelles et fonctionnelles permettant de spécifier les rôles des agents ainsi que leur fonction, $\mathcal{M}$OISE^{Inst} possède une dimension normative associant une obligation ou une permission à un rôle et une fonction. Le non-respect d'une norme est prévu grâce aux sanctions associées. Une dimension contextuelle définit, comme pour OMNI et ISLANDER, un ensemble de scènes (que nous nommons cependant contextes) faisant évoluer l'organisation dans sa totalité et influençant le comportement des agents via son intervention sur l'activation des normes.

Le système $\mathcal{S}$YNAI permet quant à lui de mettre en œuvre et d'arbitrer l'évolution des agents au sein de l'organisation. Le degré d'arbitrage permet un renforcement des normes avec une mise en place de sanctions en cas de violation sans imposer leur respect. L'arbitrage qui est considéré ici est une supervision et non un contrôle avec connotation de blocage. Le système d'arbitrage ne bloque pas les actions, il prévient et sanctionne en cas de faute. Nous donnons ici le même sens à l'arbitrage qu'à l'arbitrage d'un sport, où un arbitre supervise une partie et sanctionne le cas échéant. La supervision se fait de la part de $\mathcal{S}$YNAI par un ensemble de composants prenant chacun en compte une dimension du modèle $\mathcal{M}$OISE^{Inst}.

L'ouverture de MA$\mathcal{B}_{\text{ELI}}$ s'obtient grâce aux liens existant entre $\mathcal{M}$OISE^{Inst}, $\mathcal{S}$YNAI et les agents autonomes. Les agents autonomes agissent au sein de l'organisation via la supervision de $\mathcal{S}$YNAI. D'autres agents autonomes et hétérogènes peuvent rejoindre l'organisation, prendre en compte les normes régissant l'activité au sein de l'organisation grâce à leur spécification explicite et déclarative. Chaque norme est violable et l'agent qui ne respectera pas une norme le fera en tout état de cause puisqu'il aura pris en compte la sanction, associée à la définition de la norme, qu'il risque.

La partie qui suit est structurée en trois chapitres. Nous détaillons dans un premier temps le modèle $\mathcal{M}$OISE^{Inst} et justifions ses évolutions par rapport à $\mathcal{M}$OISE^{+} dont il découle. Puis nous expliquons comment à partir d'une telle spécification nous avons pu définir un ensemble d'agents institutionnels composant le système d'arbitrage capable de superviser l'organisation des agents du domaine. Puis, afin de mettre à la disposition des futurs utilisateurs de MA$\mathcal{B}_{\text{ELI}}$ une API, nous présentons l'implémentation et la simulation de notre modèle d'Institution Électronique.

Chapitre 6

$\mathcal{M}$oise^{Inst} : Modèle de Description d'Organisation et d'Institution Multi-Agents

Au sein du modèle d'Institution Électronique que nous proposons et auquel nous consacrons cette partie, se trouve une spécification organisationnelle normative de l'institution décrite avec le modèle $\mathcal{M}$OISE^{Inst}. Cette spécification permet de structurer le fonctionnement de l'application que nous voulons modéliser avec une Institution Électronique et de régler le comportement de chaque agent y participant. Ces règles peuvent influencer non seulement les agents, mais aussi un groupe d'agents ou encore une société complète d'agents. Cette spécification sert aussi bien à contraindre le comportement des agents qui seront les acteurs principaux dans une application, mais également les agents qui, au sein de MA$\mathcal{B}_{\text{ELI}}$, contrôlent le respect des contraintes issues de $\mathcal{M}$OISE^{Inst} (agents de la couche $\mathcal{S}$YNAI).

Nous détaillons dans ce chapitre le modèle $\mathcal{M}$OISE^{Inst} défini à partir du modèle $\mathcal{M}$OISE^{+} étudié dans la section 3.1.5. La principale caractéristique de ces deux modèles est le découpage de l'organisation en dimensions telles qu'abordées dans OMNI. Les dimensions sont appelées ici des *spécifications*. Le passage de $\mathcal{M}$OISE^{+} à $\mathcal{M}$OISE^{Inst} a fait apparaître une nouvelle spécification et a apporté des modifications aux autres. Nous verrons en détail chacune de ces spécifications et comment elles complètent la vision et la supervision de l'organisation.

6.1 Vue globale de $\mathcal{M}$oise^{Inst}

Si nous revenons à notre problématique située dans le domaine des Systèmes Multi-Agents, nous en déduisons le besoin de contraindre des agents autonomes et de faire en sorte que ceux-ci, même dotés de capacités de raisonnement, donc avec la possibilité de faire des choix et ainsi d'enfreindre les règles, respectent ces contraintes. Il nous faut non seulement trouver un moyen d'exprimer ces contraintes, mais également les rendre interprétables et par les agents du domaine (les agents qui ont pour objectif que le système global atteigne un but précis) et par des agents superviseurs regroupés dans un système d'arbitrage.

Nous voulons ainsi qu'un ensemble d'agents puisse évoluer, supervisés par un système d'arbitrage et tous deux prenant en compte l'expression des contraintes. Nous entendons exprimer aussi bien des contraintes de fonctionnement (structure de l'organisation, schémas de fonctionnement) que des contraintes de comportement (normes). Pour ceci il faut, d'une part, pouvoir spécifier les buts des agents dans le schéma global de fonctionnement de la société dans laquelle ils évolueront, d'autre part, définir un ensemble d'expressions normatives régissant le comportement des agents. C'est-à-dire

que nous allons construire des expressions déontiques basées sur le fonctionnement des agents et que nous les soumettrons à un ensemble de conditions de validité. Ainsi, nous pouvons définir que sous certaines conditions, un agent a l'autorisation ou l'obligation d'exécuter une action ou un ensemble d'actions. Nous avons vu également dans l'état de l'art que les entités ne sont pas contraintes directement la plupart du temps, mais à travers le rôle qu'elles jouent dans la société. C'est le cas pour certains modèles SMA (cf. Chapitre 3 sur les modèles multi-agents organisationnels et normatifs). Nous devons donc structurer les agents en rôles appartenant à des groupes et nous pouvons exprimer des contraintes sur cette structure en ajoutant des liens entre rôles et des contraintes de cardinalité et de compatibilité.

Le modèle organisationnel $\mathcal{M}$OISE$^+$ (**M**odel of **O**rganization for mult**I**agent **S**yst**E**m)[1] [**?**] met en place cette vision suivant trois spécifications différentes. Il offre en effet la possibilité de spécifier le fonctionnement global attendu d'une organisation d'agents (spécification fonctionnelle) ainsi que la structure de cette organisation en terme de rôles, de groupes et de liens (spécification structurelle). Une spécification déontique met en relation ces deux spécifications en explicitant les permissions, obligations et interdictions de la mise en oeuvre de la spécification fonctionnelle dans le cadre de la structure définie pour l'organisation. C'est-à-dire qu'elle définit des expressions déontiques mettant en relation un opérateur déontique, un rôle et une mission. Ainsi nous pouvons exprimer, par exemple, une obligation pour le rôle r de réaliser la mission m sous la forme $O(r, m)$.

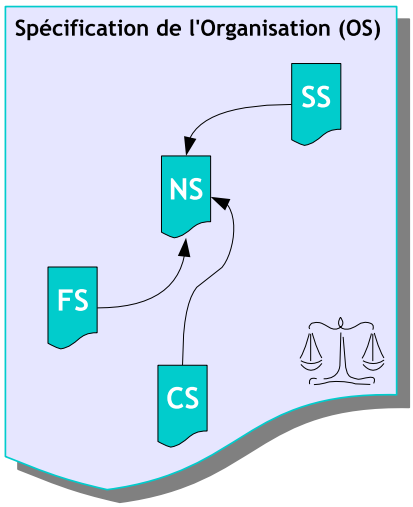


FIG. 6.1 – Aperçu de $\mathcal{M}$OISE^{Inst}

Notre modèle de spécification d'institution $\mathcal{M}$OISE^{Inst} est basé sur $\mathcal{M}$OISE$^+$ et l'étend. En effet, nous considérons que les expressions déontiques telles qu'elles sont utilisées dans $\mathcal{M}$OISE$^+$ sont trop restrictives par rapport à la définition des règles que nous recherchons. En ajoutant des informations au sein de cette spécification, nous la transformons en un ensemble de normes et donc en spécification normative. Nous pouvons grâce à cette spécification "attacher" un ensemble de normes à un rôle qu'un agent jouera. Cet agent devra donc respecter ces normes. Cependant, nous pouvons approfondir la modélisation en considérant que selon le contexte dans lequel il se trouve, l'agent sera confronté à des règles différentes. Si nous reprenons l'exemple du code de la route, la conduite sur une autoroute belge est un contexte (ou un ensemble de contextes : autoroute et Belgique) différent de la conduite sur une route nationale française. Les mêmes règles de restriction de vitesse ne seront pas appliquées de la même façon. Pour cette raison, nous ajoutons une spécification contextuelle. La spécification normative est basée sur les trois autres spécifications pour exprimer un ensemble de

[1] http ://www.lti.pcs.usp.br/moise/

normes et $\mathcal{M}$OISE^{Inst} peut être représenté comme sur la figure 6.1.

En plus de définir l'organisation des agents selon des spécifications, nous définissons des niveaux d'organisation (individuel, social et collectif). Ainsi, notre modèle est constitué de :

- Une Spécification Structurelle (*Structural Specification* ou *SS*) définissant les rôles (niveau individuel) que les agents joueront et les relations entre ces rôles (niveau social), ainsi que les groupes (niveau collectif), niveau de structuration supplémentaire, auxquels appartiennent les rôles.
- Une Spécification Fonctionnelle (*Functional Specification* ou *FS*) définissant les buts (niveau individuel) qui devront être atteints par l'organisation. Ils sont regroupés en missions (niveau social) au sein de schémas sociaux (niveau collectif).
- Une Spécification Contextuelle (*Contextual Specification* ou *CS*) définissant les différents contextes dans lesquels l'organisation va évoluer, ainsi que les transitions d'un contexte à l'autre.
- Une Spécification Normative (*Normative Specification* ou *NS*) étendant la spécification déontique (DS) de $\mathcal{M}$OISE$^+$ à l'aide entre autres de conditions d'activation et de validité. Les normes qui y sont spécifiées définissent clairement les droits et les devoirs de chaque rôle (niveau individuel) ou groupe (niveau collectif), dans un contexte particulier sur une mission (niveau social) donnée.

Ces quatre éléments constituent la Spécification Organisationnelle (OS). Formellement, l'OS est un ensemble constitué de sous-ensembles représentant chacun une spécification particulière :

$$\mathcal{OS} = \langle \mathcal{SS}, \mathcal{FS}, \mathcal{CS}, \mathcal{NS} \rangle$$

L'Organisation en elle-même est construite par l'affectation d'un ensemble d'agents sur l'OS instanciant ainsi cette dernière en ce que nous appelons une Entité Organisationnelle que nous décrirons en détail en Section 7.1. Nous aborderons ce point plus tard dans ce chapitre et en détail dans le suivant. Les notions utilisées pour définir l'OS sont elles-mêmes des spécifications qui seront ensuite instanciées par les agents. Pour faciliter la lecture, nous utiliserons tout au long de ce chapitre les notions de groupe, rôle, but, etc. au lieu de spécification de groupe, spécification de rôle, spécification de but etc. Nous parlerons alors d'instance de groupe, d'instance de rôle, d'instance de but, etc, (notions relatives à l'entité organisationnelle) quand il sera nécessaire de faire la distinction entre les deux[2].

Nous allons maintenant étudier en détail chacune des spécifications de $\mathcal{M}$OISE^{Inst}. Nous commencerons par la Spécification Structurelle qui permet de définir comment les agents seront structurés les uns par rapport aux autres et cela avant même de savoir quelles seront leurs fonctionnalités.

6.2 Spécification Structurelle

> La Spécification Structurelle (SS) exprime la structure en terme de ***rôles***, de ***liens*** entre rôles et de ***groupes***. Un ensemble de contraintes exprime en plus la ***portée des liens*** et la ***cardinalité*** des rôles et des groupes.

Cette spécification se représente sous différentes formes. Tout d'abord de façon graphique grâce à un formalisme dont un résumé se trouve sur la Fig. 6.2 permettant de représenter les éléments importants. Nous verrons son utilisation lors de la description d'un exemple. En Annexe B nous détaillons le modèle complet avec le langage BNF[3]. Nous allons décrire cette spécification de manière plus formelle grâce à la théorie des ensembles. Nous pouvons découper les éléments de description de la SS en trois composants : (i) ***les entités structurelles***, (ii) les ***liens*** entre ces entités structurelles

[2]Nous utilisons de façon indifférenciée dans le manuscrit les termes 'spécification' et 'type'
[3]Backus-Naur Normal Form

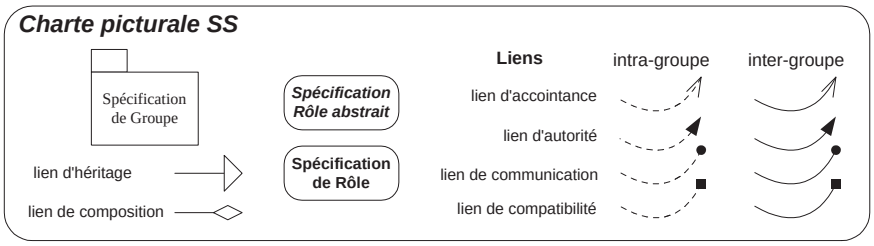


FIG. 6.2 – Éléments de la représentation graphique de la SS

et (iii) leurs ***contraintes de cardinalité***. Formellement, la SS est un couple constitué de l'ensemble de ses rôles $\mathcal{R}_{ss}$ et de l'ensemble de ses groupes $\mathcal{G}r_{ss}$.

$$\mathcal{SS} = \langle \mathcal{R}_{ss}, \mathcal{G}r_{ss}$$

Les rôles de $\mathcal{R}_{ss}$ et les groupes de $\mathcal{G}r_{ss}$ sont des entités structurelles dont nous allons voir la définition dans la section suivante.

6.2.1 Entités structurelles

> Une ***entité structurelle*** e est l'abstraction d'un agent ou d'une collectivité d'agents. Elle appartient à $\mathcal{SE}_{ss}$, ensemble des entités structurelles de la $\mathcal{SS}$ d'une modélisation $\mathcal{M}\text{OISE}^{Inst}$.

$$e \in \mathcal{SE}_{ss}$$

Comme nous le verrons par la suite, une entité structurelle sera affectée à des tâches, soumise à des normes, influencée par les contextes. $\mathcal{SE}_{ss}$ est l'union de $\mathcal{R}_{ss}$, ensemble des rôles de SS avec $\mathcal{G}r_{ss}$, ensemble des groupes de SS.

$$\mathcal{SE}_{ss} = \mathcal{R}_{ss} \cup \mathcal{G}r_{ss}$$

Rôles

> Un ***rôle*** est l'abstraction d'un agent dans la SS. Il sert de point d'ancrage d'un agent à un ensemble de contraintes qu'il doit suivre à partir du moment où il accepte de jouer ce rôle.

Un rôle r est représenté par un identifiant. Contrairement au modèle OperA [30], les contraintes influençant le rôle ne sont pas définies directement dans le rôle. Il appartient à $\mathcal{R}_{ss}$, ensemble des rôles de la SS.

$$r \in \mathcal{R}_{ss}$$

> Un ***rôle abstrait*** est un rôle qui ne peut pas être joué par un agent.

Un rôle abstrait r_{abs} appartient à $\mathcal{R}_{abs_{ss}}$, ensemble des rôles abstraits de la SS. $\mathcal{R}_{abs_{ss}}$ est un sous-ensemble de l'ensemble des rôles $\mathcal{R}_{ss}$.

$$r_{abs} \in \mathcal{R}_{abs_{ss}}$$
$$\mathcal{R}_{abs_{ss}} \subset \mathcal{R}_{ss}$$

Une relation d'***héritage*** relie deux rôles entre eux. Si un rôle r' hérite d'un rôle r (noté $r \sqsubset r'$), et que r est différent de r', alors r' reçoit les propriétés de r, et r' est un sous-rôle de r. L'ensemble des sous-rôles d'un rôle r est $\mathcal{SR}_r$. Un rôle ne peut pas hériter de lui-même et un rôle abstrait ne peut pas hériter d'un rôle non abstrait.

$$\forall\, r \in \mathcal{R}_{ss}, \forall\, r' \in \mathcal{R}_{ss} : r \sqsubset r' \wedge r \neq r' \Rightarrow r' \in \mathcal{SR}_r$$
$$\forall\, r \in \mathcal{R}_{ss}/\mathcal{R}_{abs_{ss}}, \forall\, r' \in \mathcal{R}_{abs_{ss}} \Rightarrow r\neg \sqsubset r'$$

La structure de base entre rôles se fait par liens d'héritage et non pas par liens hiérarchiques comme nous pouvons le trouver dans les modèles ISLANDER [38] ou TOP [84]. Dans notre cas, un rôle fils héritant d'un rôle père va hériter des attributs de son père, c'est-à-dire les liens et les normes qui concernent son père.

Groupes

> Un ***groupe*** gr est l'abstraction d'un collectif d'agents. Il est défini par un sept-uplet constitué de $\mathcal{R}_{gr}$, ensemble des rôles non abstraits pouvant être joués dans les groupes créés à partir de gr, de $\mathcal{SG}r_{gr}$, ensemble des sous-groupes du groupe gr, $\mathcal{L}_{gr}^{intra}$, ensemble des liens ayant une portée intra-groupe, $\mathcal{L}_{gr}^{inter}$, ensemble des liens ayant une portée inter-groupe, nr_{gr} et ng_{gr} cardinalités respectivement des rôles et des sous-groupes de gr, na, cardinalité de gr, c'est-à-dire le nombre d'agents pouvant y jouer un rôle.

Tout groupe gr appartient à $\mathcal{G}r_{ss}$, ensemble des groupes de la SS. L'ensemble des rôles appartenant à un groupe est un sous-ensemble des rôles de la SS. L'ensemble des sous-groupes d'un groupe est un sous-ensemble des groupes de SS.

$$gr \in \mathcal{G}r_{ss}$$
$$gr = \langle \mathcal{R}_{gr}, \mathcal{SG}r_{gr}, \mathcal{L}_{gr}^{intra}, \mathcal{L}_{gr}^{inter}, nr_{gr}, ng_{gr}, na \rangle$$
$$\mathcal{R}_{gr} \subset \mathcal{R}_{ss}$$
$$\mathcal{SG}r_{gr} \subset \mathcal{G}r_{ss}$$

Si un groupe est le sous-groupe d'aucun autre groupe, alors il est considéré comme le groupe "racine" de la SS. Un groupe ne peut être le sous-groupe que d'un seul groupe.

Un rôle de $\mathcal{R}_{ss}$ non abstrait appartient obligatoirement à au moins un groupe tandis que les rôles abstraits de $\mathcal{R}_{abs_{ss}}$ n'appartiennent à aucun groupe.

$$\forall\, r \in \mathcal{R}_{ss}/\mathcal{R}_{abs_{ss}}, \exists\, gr \in \mathcal{G}r_{ss} : r \in \mathcal{R}_{gr}$$
$$\forall\, gr \in \mathcal{G}r_{ss}, \mathcal{R}_{gr} \cap \mathcal{R}_{abs_{ss}} = \emptyset$$

Un groupe permet de contraindre un ensemble de rôles non-abstraits. Un agent sera donc contraint par les rôles qu'il adoptera et par les groupes dans lesquels il jouera ses rôles, mais sera influencé par les contraintes provenant des super-rôles de ses propres rôles et des super-groupes de ses propres groupes. Nous verrons dans la section 6.2.4 les différents types de contraintes pouvant être applicables aux entités structurelles. Voyons pour le moment les liens possibles entre les rôles.

6.2.2 Instances

Avant d'aborder les liens de la SS spécifiés entre deux rôles et qui concernent les agents lors de l'instanciation, nous devons préciser ce que nous entendons par "instance". L'instanciation de l'OS consiste à faire intervenir un ou plusieurs agents au sein de l'organisation afin qu'ils se comportent comme le spécifie l'OS.

Une ***instance de rôle*** est un agent jouant un rôle. Une ***instance de groupe*** regroupe les instances de rôles d'un même groupe.

Une instance de groupe appartenant à l'ensemble des instances de groupe $\mathcal{G}roup$ est différente d'une instance de rôle dans le sens où un groupe ne peut pas être instancié par un agent mais par un ensemble d'instances de rôles. De ce fait, l'instanciation d'un groupe se formalise de la façon suivante :

$$instantiate : \mathcal{G}r_{ss} \rightarrow \mathcal{G}roup$$

Nous définissons $\mathcal{A}$ comme étant l'ensemble des agents présents dans le système, $\mathcal{R}ole$ comme l'ensemble des instances de rôles et $role \in \mathcal{R}ole$ une instance de rôle. L'ensemble $\mathcal{R}ole$ est le résultat d'une fonction d'instanciation *instantiate* prenant en paramètre l'ensemble des agents, l'ensemble des spécifications des instances de groupes et l'ensemble des spécifications de rôle de la SS.

$$instantiate : \mathcal{A} \times \mathcal{G}roup \times \mathcal{R}_{ss} \rightarrow \mathcal{R}ole$$

Une instance de groupe *group* est constituée de $\mathcal{R}ole_{group}$, l'ensemble des instances de rôles de *group*, de $\mathcal{SG}roup_{group}$, l'ensemble des instances des sous-groupes de *group*, de $\mathcal{L}ink^{intra}_{group}$, l'ensemble des liens intra-groupe du groupe *group* et enfin de $\mathcal{L}ink^{inter}_{group}$, l'ensemble des liens inter-groupe liens des instances de rôles du groupe *group* et d'autres instances de groupes issues de la même spécification de groupe.

$$\begin{array}{l}\mathcal{R}ole = \langle \mathcal{A}, \mathcal{R}_{ss} \rangle \\ \mathcal{G}roup = \langle \mathcal{R}ole, \mathcal{L}ink_{group} \rangle \\ group \in \mathcal{G}roup \\ group = \langle \mathcal{R}ole_{group}, \mathcal{SG}roup_{group}, \mathcal{L}ink^{intra}_{group}, \mathcal{L}ink^{inter}_{group} \rangle \\ \mathcal{R}ole_{group} \subset \mathcal{R}ole \\ \mathcal{SG}roup_{group} \subset \mathcal{G}roup\end{array}$$

6.2.3 Liens

Les liens définissent le niveau social de la SS.

Un ***lien*** définit une relation entre deux rôles au sein d'une même spécification de groupe. Il est défini par un rôle source, un rôle cible et un ***type***.

Un lien est caractérisé par ses rôles ***source*** et ***destination***, par sa ***portée*** (inter ou intra-groupe), par son ***type*** et enfin par l'***extension*** dans les sous-groupes de ce lien. Les liens peuvent être de quatre types différents, (i) lien d'***accointance***, (ii) lien de ***communication***, (iii) lien d'***autorité*** et (iv) lien de ***compatibilité***. Considérons un groupe $gr \in \mathcal{G}r$, ses liens sont formellement représentés par :

$$\begin{array}{l}\mathcal{L}_{gr} = \mathcal{L}^{intra}_{gr} \cup \mathcal{L}^{inter}_{gr} \\ \forall\, l \in \mathcal{L}_{gr}, l = link(r_s, r_d, t)\end{array}$$

où $r_s \in \mathcal{R}_g$ est le rôle source, $r_d \in \mathcal{R}_g$ est le rôle destination, et $t \in \{acq, com, aut, comp\}$ est le type de lien.

Types de lien

- Dans le lien d'accointance ($t = acq$), les agents jouant le rôle source r_s sont autorisés à ***représenter*** des agents jouant le rôle cible r_d.
- Dans un lien de communication ($t = com$), les agents jouant le rôle source r_s sont autorisés à ***communiquer*** avec les agents jouant le rôle cible r_d.
- Dans un lien d'autorité ($t = aut$), les agents jouant le rôle source r_s sont autorisés à ***contrôler*** les agents jouant le rôle cible r_d.
- Dans un lien de compatibilité ($t = comp$), les agents jouant le rôle r_s sont autorisés à ***jouer*** le rôle r_d. Par défaut, les rôles d'une SS ne sont pas compatibles.

Un lien d'autorité implique l'existence d'un lien de communication, qui implique lui même l'existence d'un lien d'accointance tout en conservant la portée du lien :

$$link(r_s, r_d, aut) \Rightarrow link(r_s, r_d, com)$$
$$link(r_s, r_d, com) \Rightarrow link(r_s, r_d, acq)$$

Portée des liens

La portée des liens d'un groupe gr est identifiée par les liens internes à ce groupe (***liens intra-groupe***) $\mathcal{L}_{gr}^{intra}$ et par des liens externes à ce groupe (***liens inter-groupe***) $\mathcal{L}_{gr}^{inter}$. Suivant les contraintes de cardinalités (que nous allons voir dans la section suivante) nous pouvons avoir plusieurs instances de la spécification d'un groupe. Les agents sont liés via le rôle qu'ils jouent et grâce à des instances de liens. Une instance de lien est le résultat de la fonction *instantiate* portant sur une spécification de lien et deux instances de rôle (jouées par des agents) :

$$group = instantiate(gr)$$
$$instantiate : \mathcal{L}_{gr} \times \mathcal{R}ole \times \mathcal{R}ole \rightarrow \mathcal{L}ink_{group}$$

Un lien intra-groupe de gr établit que le ou les agents jouant le rôle source dans l'instance du groupe sont reliés à tous les agents jouant le rôle destination dans cette même instance de groupe gr ou dans une de ces instances de ses sous-groupes.

$$gr \in \mathcal{G}r_{ss}$$
$$group_1 = instantiate(gr)$$
$$r \in \mathcal{R}_{gr}, r' \in \mathcal{R}_{gr}$$
$$role_1 = instantiate(r_a, \alpha)$$
$$\forall\, l \in \mathcal{L}_{gr}^{intra} : l = link(r, r', t), \forall\, \alpha \in \mathcal{A} : instantiate(\alpha, r, group_1) \Rightarrow \forall\, \alpha' \in \mathcal{A} \wedge$$
$$instantiate(\alpha', r', group_1), \exists\, link \in \mathcal{L}ink_{group_1}^{intra} : link = link(\alpha, \alpha' t)$$

Un lien inter-groupe de gr établit quant à lui que la portée du lien d'un agent jouant le rôle source porte sur tous les agents jouant le rôle cible indépendamment des instances de groupe de gr au sein desquels ces agents jouent un rôle. Ainsi :

$$\forall\, l \in \mathcal{L}_{gr}^{inter} : l = link(r, r', t), \forall\, \alpha \in \mathcal{A} : instantiate(\alpha, r, group_1) \Rightarrow \forall\, \alpha' \in \mathcal{A} \wedge$$
$$instantiate(\alpha', r', group_2) : group_2 = instantiate(gr), \exists\, link \in \mathcal{L}ink_{group_1}^{inter} : link = link(\alpha, \alpha', t)$$

Par exemple, une compatibilité intra-groupe $link(r, r', comp) \in \mathcal{L}_{gr}^{intra}$ établit qu'un agent jouant le rôle r dans une instance de groupe $group_1$ du groupe gr est autorisé à jouer aussi le rôle r' dans la même instance de groupe $group_1$. Une compatibilité inter-groupe $link(r, r', comp) \in \mathcal{L}_{gr}^{inter}$ établit qu'un agent jouant le rôle r dans l'instance de groupe $group_1$ du groupe gr est autorisé à jouer aussi r' dans une autre instance de groupe $group_2$ du groupe gr.

Extension des liens

L'extension des liens permet de spécifier si les contraintes exprimées sont propagées dans les sous-groupes. Ainsi, la fonction *extendsToSubgroups* précise si les liens intra-groupes et inter-groupes d'un groupe gr sont valides dans ses sous-groupes $\mathcal{SG}r_{gr}$.

$$
\begin{array}{l}
\mathbb{B} = \{0, 1\} \\
extendsToSubgroups : \mathcal{L} \mapsto \mathbb{B} \\
l \in \mathcal{L}_{gr}^{intra} \wedge extendsToSubgroups(l) = 0 \Rightarrow \forall\, sgr \in \mathcal{SG}r_{gr} : l \in \mathcal{L}_{sgr}^{intra} \\
l \in \mathcal{L}_{gr}^{intra} \wedge extendsToSubgroups(l) = 1 \Rightarrow \nexists sgr \in \mathcal{SG}r_{gr} : l \in \mathcal{L}_{sgr}^{intra} \\
l \in \mathcal{L}_{gr}^{inter} \wedge extendsToSubgroups(l) = 0 \Rightarrow \forall\, sgr \in \mathcal{SG}r_{gr} : l \in \mathcal{L}_{sgr}^{inter} \\
l \in \mathcal{L}_{gr}^{inter} \wedge extendsToSubgroups(l) = 1 \Rightarrow \nexists sgr \in \mathcal{SG}r_{gr} : l \in \mathcal{L}_{sgr}^{inter}
\end{array}
$$

Héritage sur les liens

Les liens sont hérités entre rôles pères et rôles fils et s'appliquent de la manière suivante :

$$
\begin{array}{l}
link(r_s, r_d, t) \wedge r_s \neq r_d \wedge r_s \sqsubset r'_s \Rightarrow link(r'_s, r_d, t) \\
link(r_s, r_d, t) \wedge r_s \neq r_d \wedge r_d \sqsubset r'_d \Rightarrow link(r_s, r'_d, t)
\end{array}
$$

6.2.4 Contraintes de cardinalité

> Les ***contraintes de cardinalité*** spécifient le nombre maximal et le nombre minimal d'instances de rôle ou de groupe qu'une spécification de groupe accepte.

Trois types de cardinalités sont définis au sein d'un groupe : (i) une cardinalité de rôles (nr), (ii) une cardinalité de sous-groupes (ng) et (iii) une cardinalité d'agents (na). Un groupe dans la mise en œuvre d'une spécification de groupe sera considéré comme bien formé s'il respecte ces trois cardinalités.

> La ***cardinalité des rôles*** de gr est une fonction partielle nr_{gr} spécifiant les nombres minimum et maximum d'instances des rôles de l'ensemble $\mathcal{R}_{gr}$ que le groupe gr accepte.

$$nr_{gr} : \mathcal{R}_{gr} \nrightarrow \mathbb{N} \times \mathbb{N}$$

Par défaut, la cardinalité nr_{gr} d'un rôle $r \in \mathcal{R}_{gr}$ est $(0, \infty)$.

> La ***cardinalité des sous-groupes*** de gr est une fonction partielle ng_{gr} spécifiant les nombres minimum et maximum d'instances des sous-groupes de l'ensemble $\mathcal{SG}r_{gr}$ que le groupe gr accepte.

$$ng_{gr} : \mathcal{SG}r_{gr} \nrightarrow \mathbb{N} \times \mathbb{N}$$

Par défaut, la cardinalité ng_{gr} d'un groupe $gr' \in \mathcal{SG}_{gr}$ est $(0, \infty)$.

> La ***cardinalité d'agents*** est une fonction partielle na spécifiant les nombres minimum et maximum d'agents pouvant jouer un rôle au sein d'une instance de groupe gr, c'est-à-dire le nombre total d'instances des rôles de gr et des rôles des sous-groupes de gr.

$$na : \mathcal{G}r_{ss} \nrightarrow \mathbb{N} \times \mathbb{N}$$

Cette cardinalité est un des principaux changements apportés à la SS de $\mathcal{M}$OISE^{+} pour obtenir celle de $\mathcal{M}$OISE^{Inst}. Elle est particulièrement intéressante quand nous avons des rôles compatibles présents au sein d'un même groupe. Par défaut, la cardinalité na d'un groupe gr est égale à la somme des cardinalités nr_{gr} et la somme des produits de la cardinalité ng_{gr} par la cardinalité na des sous-groupes de gr, c'est-à-dire $\sum_{\forall\, r \in \mathcal{R}_{gr}} nr_{gr}(r) + \sum_{gr' \in \mathcal{SG}r_{gr}} (ng_{gr}(gr') \times na(gr'))$.

6.2.5 Exemple

L'exemple que nous avons choisi et qui illustrera l'utilisation de notre modèle dans ce chapitre et le suivant est celui des "Robots Footballeurs" issu de [55] et servant également d'illustration du modèle $\mathcal{M}$OISE^{+}.

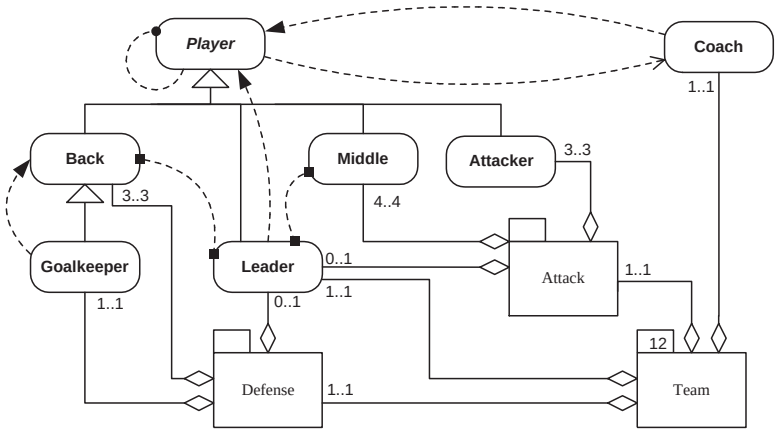


FIG. 6.3 – SS de l'exemple des "Robots Footballeurs"

LA FIG. 6.3 illustre graphiquement la SS de notre exemple. Ainsi nous voyons globalement qu'il y aura deux grands types de rôles, les joueurs (rôle "Player") et l'entraineur (role "Coach"). Le rôle "Player" étant un rôle *abstrait*, les agents ne pourront pas le jouer. Les rôles d'arrière ("Back"), de milieu ("Middle") et d'attaquant ("Attacker") héritent du rôle "Player". Le rôle de "Leader" hérite aussi du rôle "Player" quant au rôle de gardien de but "Goalkeeper", il hérite du rôle "Back". Ces rôles non-abstraits appartiennent à des groupes. Le groupe racine est celui représentant l'équipe de football dans sa totalité (groupe "Team"), c'est-à-dire les joueurs et l'entraîneur. Le rôle "Coach" a donc un lien de composition avec le groupe "Team". Les joueurs sont répartis en deux sous-groupes, le groupe "Attack" pour les attaquants et le groupe "Defense" pour les défenseurs. Ici le choix est fait de considérer les milieux de terrain comme des attaquants. C'est pourquoi le rôle "Middle" appartient au groupe "Attack" au même titre que le rôle "Attacker". Les rôles "Back" et "Goalkeeper" quant à eux appartiennent au groupe "Defense". Enfin, le capitaine d'équipe (celui jouant le rôle de "Leader") pouvant être n'importe quel joueur, il appartient aux groupes "Defense", "Attack" et "Team". Par contre les liens qu'il a avec les autres rôles et ses cardinalités le diffèrent des autres rôles. Formellement cela se représente par ceci :

$\mathcal{G}r_{ss} = \{Team, Defense, Attack\}$
$\mathcal{R}_{ss} = \{Player, Coach, Back, Goalkeeper, Leader, Middle, Attacker\}$
$\mathcal{R}_{abs_{ss}} = \{Player\}$
$\mathcal{SR}_{Player} = \{Back, Goalkeeper, Leader, Middle, Attacker\}$
$\mathcal{R}_{Team} = \{Coach, Leader\}$
$\mathcal{R}_{Defense} = \{Back, Goalkeeper, Leader\}$
$\mathcal{R}_{Attack} = \{Leader, Middle, Attacker\}$
$\mathcal{SG}_{Team} = \{Defense, Attack\}$

Tous les liens de ce modèle sont des liens intra-groupes. C'est à dire qu'ils relient des agents évoluant au sein d'une et une seule équipe. Ainsi, l'entraîneur aura le contrôle de tous les joueurs de son équipe et ces mêmes joueurs prendront en compte l'existence de l'entraîneur (liens d'autorité et d'accointance entre les rôles "Player" et "Coach"). Grâce au lien de communication du rôle "Player" sur lui-même, tous les joueurs sont capables de communiquer entre eux. Grâce aux liens d'héritage, un attaquant pourra par exemple communiquer avec le gardien de but. L'implication de ces liens entraîne que les joueurs auront une représentation des autres joueurs et que l'entraîneur pourra communiquer et représenter ses joueurs. Parmi les joueurs, le "Leader" et le "Goalkeeper" auront chacun un lien d'autorité sur tous les joueurs pour le premier et sur les arrières pour le second. Enfin, le rôle "Leader" est compatible avec les rôles "Back" et "Middle". Nous considérons ici que le capitaine d'équipe étant un membre de l'équipe et un joueur peut aussi être un arrière, y compris le gardien grâce au lien d'héritage, un milieu de terrain mais pas un attaquant. Les liens entre rôle sont formalisés de la façon suivante :

$\mathcal{L}^{intra}_{Defense} = \{link(Goalkeeper, Back, aut), link(Leader, Back, comp)\}$
$\mathcal{L}^{intra}_{Attack} = \{link(Leader, Middle, comp)\}$
$\mathcal{L}^{intra}_{Team} = \{link(Player, Coach, acq), link(Coach, Player, aut), link(Player, Player, com),$
$link(Leader, Player, aut)\}$

Les cardinalités des groupes "Defense" et "Attack" sont de $(1, 1)$ ce qui veut dire que pour une instance du groupe "Team" (pour une équipe de football), il ne pourra y avoir en son sein qu'une et une seule attaque et une et une seule défense. Les cardinalités de rôles expriment le fait qu'il doit y avoir 1 gardien, 3 défenseurs et éventuellement 1 leader dans le groupe de défense. Dans le groupe d'attaque, il doit y avoir 3 attaquants, 4 milieux de terrain et éventuellement 1 leader. Quoi qu'il en soit, il faut qu'il y ait un leader dans l'équipe (soit en défense, soit en attaque). Si tous les rôles sont joués en respectant leur cardinalité maximale, nous avons 14 membres du groupe "Team" $((1+3+1)*1+1+(4+3+1)*1=14)$. La cardinalité du groupe permet d'éviter qu'effectivement 14 agents viennent jouer un rôle au sein de l'équipe. Ainsi les cardinalités *na* du groupe "Team" sont définies à $(12, 12)$ restreignant l'accès à seulement 12 agents. Ainsi, pour que les autres cardinalités soient respectées, il faudra que certains agents jouent deux rôles. A noter que ce modèle nous permet d'avoir un leader par groupe de défense et d'attaque. Les cardinalités qui sont spécifiées sont formalisées de la façon suivante :

$ng_{Team}(Defense) = (1, 1)$
$ng_{Team}(Attack) = (1, 1)$
$nr_{Team}(Coach) = (1, 1)$
$nr_{Team}(Leader) = (1, 1)$
$na_{Team} = (12, 12)$
$nr_{Defense}(Goalkeeper) = (1, 1)$
$nr_{Defense}(Back) = (3, 3)$
$nr_{Defense}(Leader) = (0, 1)$
$nr_{Attack}(Leader) = (0, 1)$
$nr_{Attack}(Middle) = (4, 4)$
$nr_{Attack}(Attacker) = (3, 3)$

Nous pouvons également spécifier le groupe "Attack" de la façon suivante :

$Attack = (\{Leader, Middle, Attacker\}, \emptyset, \{link(Leader, Middle, comp)\}, \emptyset, \{(0,1),(4,4),(3,3)\}, \emptyset)$

Maintenant que nous avons contraint les agents en structurant leur organisation, il nous faut définir ce que cette organisation doit faire, c'est à dire les buts que l'ensemble de ces agents devra atteindre. Ceci se fait au sein de la Spécification Fonctionnelle.

6.3 Spécification Fonctionnelle

La Spécification Fonctionnelle (FS) définit un ensemble de schémas sociaux considérés comme les buts collectifs à atteindre par l'organisation. De ces schémas nous pouvons en déduire des missions sur lesquelles les agents devront s'engager. Un schéma social exprime le processus que les agents doivent suivre pour atteindre le but collectif représenté par le but racine du schéma. Il peut donc être représenté par un arbre dont les feuilles sont les buts pouvant être atteints par un agent seul (niveau individuel). Les missions regroupent, *a priori*, les buts en ensembles cohérents, qui devront être accomplis par les agents. Nous retrouvons le même découpage en buts et en sous-buts que dans TOP [84] et TÆMS [54].
La représentation graphique se fait en respectant la charte de la FIG. 6.4. Nous décrivons la FS en BNF dans l'Annexe B.

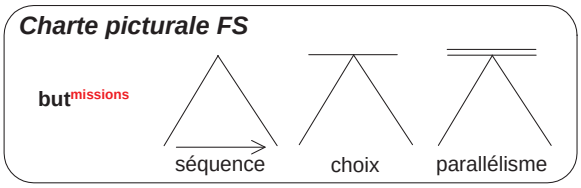


FIG. 6.4 – Éléments de la représentation graphique de la FS

> Les composants de base d'une ***Spécification Fonctionnelle*** sont les ***schémas sociaux*** constitués de ***buts*** eux mêmes regroupés en ***missions***.

Formellement, la FS d'une organisation est un ensemble de schémas sociaux $\mathcal{S}$, un ensemble de buts $\mathcal{G}o$ ainsi qu'un ensemble de préférences $\mathcal{PR}$ sur les missions des schémas sociaux :

$$\mathcal{FS} = \langle \mathcal{S}, \mathcal{G}o, \mathcal{PR} \rangle$$

6.3.1 Schémas sociaux

> Un ***schéma social*** est la structure d'un objectif global de l'organisation, décomposé en buts structurés en plans et regroupés en missions.

Un schéma social $s \in \mathcal{S}$ se définit de la façon suivante :

$$\text{s} = \langle \mathcal{G}o_s, Rac_s, \mathcal{M}_s, \mathcal{P}_s, mo_s, nm_s \rangle$$

où :
– $\mathcal{G}o_s \subset \mathcal{G}o$ est l'ensemble des buts (collectifs) du schéma s ;
– $Rac_s \in \mathcal{G}o_s$ est le but racine du schéma s ;
– $\mathcal{M}_s$ est l'ensemble des étiquettes de missions du schéma s ;

- $\mathcal{P}_s$ est l'ensemble des plans globaux issus de l'arbre de décomposition de buts ;
- mo_s est une fonction qui spécifie pour chaque mission l'ensemble des buts qui lui est rattaché ;
- nm_s est une cardinalité spécifiant les instances d'un schéma social au travers du nombre (minimum, maximum) d'agents qui peuvent s'engager sur chaque mission.

6.3.2 Instances

> Une ***instance de mission*** est un agent engagé sur une mission. Une instance de mission regroupe un ensemble d'***instances de buts*** structurées au sein d'une ***instance de schéma.***

Nous définissons $\mathcal{M}ission$ comme l'ensemble des instances de missions et $mission \in \mathcal{M}issions$ une instance de mission. L'ensemble $\mathcal{M}ission$ est le résultat d'une fonction d'instanciation *instantiate* prenant en paramètre l'ensemble des agents et l'ensemble des spécifications de mission. Une instance de groupe *mission* est quant à elle composée d'un ensemble d'instances de rôles *goal*.

$$
\begin{array}{l}
instantiate : \mathcal{A} \times \mathcal{M} \nrightarrow \mathcal{M}ission \\
instantiate : \mathcal{G}o \nrightarrow \mathcal{G}oal \\
mission \in \mathcal{M}ission \\
goal \in \mathcal{G}oal
\end{array}
$$

Une instance de schéma *scheme* appartenant à l'ensemble des instances de schéma $\mathcal{S}cheme$ représente un cadre de fonctionnement des agents. Lorsqu'un schéma est instancié, tous ses buts y sont instanciées de sorte que les agents puissent s'engager sur des missions et ainsi également les instancier. Nous avons alors $scheme \in \mathcal{S}cheme, scheme = \langle \mathcal{G}oal_{scheme}, \mathcal{M}ission_{scheme} \rangle$. Des instances de schémas concernant la même spécification de schéma peuvent coexister afin de permettre l'exécution de deux buts racines en même temps (répondre à deux questions différentes par exemple, ou marquer deux buts si deux ballons se trouvent sur un même terrain de football par exemple).

6.3.3 Buts

> Un ***but*** g est un état final constituant un objectif vers lequel l'organisation tend. Un but peut être défini par un ***plan*** décomposant le but en sous-buts.

$$
\begin{array}{l}
g \in \mathcal{G}o \\
g = \langle \mathcal{SG}o_g, op \rangle \\
op \in \{\text{`,'}, \text{` | '}, \text{` || '}\}
\end{array}
$$

où $\mathcal{SG}o_g$ est l'ensemble des sous-buts du but g. L'ensemble des sous-buts associé à l'opérateur *op* constituent le plan du but. L'opérateur *op* peut prendre trois valeurs différentes :

- *séquence* (',') : le plan $g1 = (\{g2, g3\}, ,)$ signifie que le but $g1$ sera atteint si le but $g2$ puis le but $g3$ sont atteints ;
- *choix* ('|') : le plan $g4 = (\{g5, g6\}, |)$ signifie que le but $g4$ sera atteint si seulement un des deux buts $g5$ ou $g6$ est atteint ;
- *parallélisme* ('||') : le plan $g7 = (\{g8, g9\}, ||)$ signifie que le but $g7$ sera atteint si à la fois les buts $g8$ et $g9$ sont atteints, chacun pouvant être atteints en même temps.

Une amélioration de $\mathcal{M}$OISE^{Inst} est de pouvoir faire référence à un schéma dans un plan.

$$\mathcal{SG}o \subset \mathcal{G}o \cup \mathcal{S}$$

Cela veut dire que si le plan du but g_1 est une séquence du but g_2 et du schéma sch_1, alors le but g_1 sera atteint quand le but g_2 puis le but racine du schéma sch_1 seront atteints. Ceci est intéressant quand le même ensemble de buts est utilisable plusieurs fois dans la FS. Nous factorisons ainsi un ensemble de buts.

$sch_1 \in \mathcal{S}, g_1 \in \mathcal{R}_{sch_2}, g_2 \in \mathcal{R}_{sch_2}$
$g_1 = (g_2, sch_1)$
$status(g_2) = satisfied \Rightarrow status(Rac_{sch_1}) = possible$
$status(Rac_{sch_1}) = satisfied \Rightarrow status(g_1) = satisfied$

Un but a un cycle de vie. Lorsqu'il est instancié au sein d'un schéma, il devient possible. Puis soit il est impossible à atteindre, soit il est satisfait. Une fois satisfait, le cycle de vie du but se termine. Nous représentons ce cycle par la Fig. 6.5.

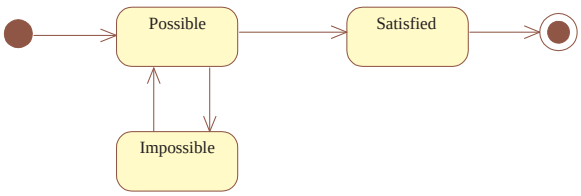


FIG. 6.5 – Diagramme d'état UML représentant le cycle de vie d'un but

La fonction *status* permet de savoir dans quel état de son cycle de vie se trouve une instance de l'ensemble des but d'une Organisation $\mathcal{G}oal$.

$status : \mathcal{G}oal \nrightarrow \{possible, impossible, satisfied\}$

6.3.4 Missions

Une ***mission*** est un ensemble de buts cohérents appartenant à un même schéma sur laquelle un agent peut s'engager.

Si nous considérons le schéma s, l'ensemble des missions $\mathcal{M}_s$ est constitué de mission m_i elles-mêmes constituées de buts grâce à la fonction mo_s. Pour chaque mission m_i, une cardinalité est définie grâce à la fonction nm_s, spécifiant ainsi le nombre d'agents pouvant être engagés sur la mission en même temps. Par défaut, la cardinalité est $(0, \infty)$.

$m \in \mathcal{M}_s$
$mo_s : \mathcal{M}_s \rightarrow \mathbb{P}(\mathcal{G}o_s)$
$nm_s : \mathcal{M}_s \nrightarrow \mathbb{N} \times \mathbb{N}$

FIG. 6.6 – Diagramme d'état UML représentant le cycle de vie d'une mission

Le cycle de vie de l'instance d'une mission définit les états *commited* et *accomplished* dont l'enchaînement se trouve sur la FIG. 6.6. Si un agent $\alpha \in \mathcal{A}$ s'engage sur l'instance $mission \in \mathcal{M}ission$ d'une mission, il s'engage à exécuter toutes les instances de buts $goal_i$ de $mission$ ($goal_i \in mission$) et α essaiera de satisfaire un but $goal_i$ seulement lorsque la précondition du but pour $goal_i$ est vérifiée. Les missions ayant une cardinalité autorisant plusieurs agents à s'engager dessus sont accomplies si tous les buts des missions sont atteints par au moins un agent ce qui se traduit formellement par :

$\forall\, goal \in \mathcal{G}oal_{mission}, status(goal) = satisfied \vee status(goal) = impossible \Rightarrow status(mission) = accomplished$

6.3.5 Préférences

Les relations de préférences $pr \in \mathcal{PR}$ de la FS définissent un ordre sur les missions et sont notées $\prec$. Si nous avons une préférence $pr_a = m_1 \prec m_2$, alors la mission m_1 a une préférence sociale par rapport à la mission m_2. Un agent doit donc en priorité s'engager sur m_1 si le choix lui est donné. Les préférences pouvant porter sur des missions de schémas sociaux différents, cet opérateur peut être utilisé pour spécifier des préférences entre schémas. La notion de préférence provient du modèle $\mathcal{M}$OISE$^+$. C'est une notion cruciale mais ne faisant pas partie des objectifs de cette thèse, nous nous n'étendrons pas dessus.

Si une mission m fait référence au but racine d'un schéma social, alors l'agent qui s'engagera sur cette mission est l'agent qui a la permission de créer ce schéma et de commencer son exécution. Nous abordons rapidement dans l'exemple qui suit la façon de spécifier un ensemble de schémas sociaux.

6.3.6 Exemple

La FS que nous présentons ici est celle de l'exemple des "Robots Footballeurs" déjà abordé dans la section 6.2.5.

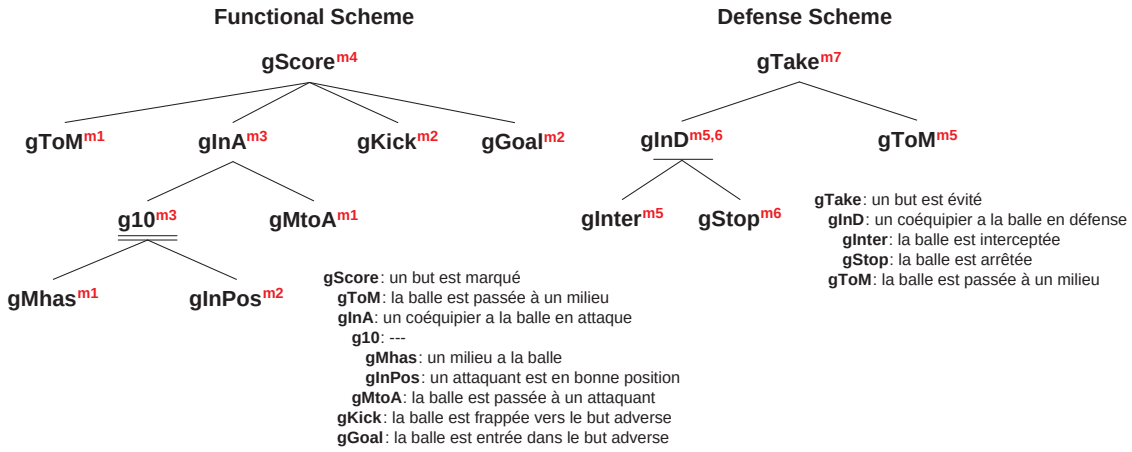


FIG. 6.7 – FS de l'exemple des "Robots Footballeurs"

La FIG. 6.7 représente la FS de la Spécification Organisationnelle d'une équipe de football. Tous les buts utilisés y sont également décrits. La spécification est construite sur deux schémas sociaux, un schéma de défense dont le but racine est de ne pas prendre de but (*gTake*) et un schéma d'attaque (appelé schéma fonctionnel car est considéré comme le schéma principal de la FS) dont le but racine est de marquer un but (*gScore*). Le but *gScore* est satisfait lorsque les buts de passer le ballon à un milieu de terrain (*gToM*), puis de passer à un attaquant en position de tir (*gInA*), de tirer au but (*gKick*) et enfin de tromper le gardien adverse (*gGoal*), sont satisfaits séquentiellement. De même le but *gInA* est atteint quand les buts *g10* puis *gMtoA* sont atteints. Enfin le but *g10* sera atteint si un milieu de terrain a le ballon (*gMhas*) et en même temps un attaquant est en bonne position (*gInPos*) afin que le but *gMtoA* spécifiant qu'un milieu fasse la passe à un attaquant puisse être atteint.

$\mathcal{S} = \{Functional, Defense\}$
$\mathcal{G}o = \{gScore, gTake, gToM, gInD, gInA, gInter, g10, gStop, gMhas, gToM, gInPos, gMtoA, gKick, gGoal\}$
$\mathcal{G}o_{Functional} = \{gScore, gToM, gInA, g10, gMhas, gInPos, gMtoA, gKick, gGoal\}$
$Rac_{Functional} = gScore$
$gScore = (\{gToM, gInA, gKick, gGoal\}, ,)$
$gInA = (\{g10, gMtoA\}, ,)$
$g10 = (\{gMhas, gInPos\}, ||)$

$\mathcal{G}o_{Defense} = \{gTake, gInD, gToM, gInter, gStop\}$
$Rac_{Defense} = gTake$

Le but racine du schéma de défense est quant à lui composé d'un plan séquentiel des buts *gInD* et *gToM* spécifiant que pour ne pas prendre de but, le ballon doit être récupéré par un défenseur puis passé à un milieu de terrain. Le but de récupérer le ballon est lui atteint quand un défenseur intercepte la balle ou que le gardien l'arrête, d'où le plan de *gInD* constitué d'un choix entre *gInter* et *gStop*. Les plans du schéma de défense sont spécifiés formellement comme ceci :

$gTake = (gInD\ ,\ gToM)$
$gInD = (gInter\ |\ Stop)$

Nous remarquons que le but *gToM* fait partie des deux schémas. C'est un moyen que nous avons pour synchroniser les deux schémas et faire ainsi en sorte de structurer des schémas différents pour l'attaque et la défense. Un autre moyen de synchronisation dynamique est de se baser sur les contextes (l'équipe est dans un contexte d'attaque ou dans un contexte de défense) afin de définir quelles missions doivent être accomplies dans quel contexte.

Id.	***Buts de la mission***	***Card.***	***Description***
m1	gToM, gMhas, gMtoA	1..4	Mission d'un milieu de terrain
m2	gInPos, gKick, gGoal	1..3	Mission d'un attaquant
m3	gInA, g10	1..7	Passer la balle à un attaquant
m4	gScore	1..7	Marquer un but
m5	gInD, gInter, gToM	1..3	Mission d'un défenseur
m6	gInD, gStop	1..1	Mission du gardien de but
m7	gTake	1..4	Éviter de prendre un but

TAB. 6.1 – Définition des missions de la FS de l'exemple des "Robots Footballeurs"

Les missions que nous définissons pour cet exemple sont reportées dans la TAB. 6.1. Sur la FIG. 6.7, chaque but indique la (les) mission(s) à laquelle (auxquelles) il appartient. Les buts sont regroupés de façon cohérente dans des missions. Ainsi pour le schéma fonctionnel nous pouvons regrouper les buts que devront atteindre les milieux de terrain et les buts que devront atteindre les attaquants afin de définir leur mission respective à savoir *m1* et *m2*. Pour ces deux missions, nous définissons une cardinalité de $(1, 4)$ pour les milieux de terrain puisque nous aurons 4 joueurs jouant ce rôle dans l'équipe (cf. la SS de l'exemple dans la section précédente) et (1,3) pour les attaquants pour les mêmes raisons que pour les milieux de terrain. Ainsi, pour un schéma fonctionnel (considéré comme une phase d'attaque) la mission des attaquants sera accomplie si le but d'avoir un attaquant en position de tir, le but de tirer au but et le but de marquer le but sont atteints par au moins un des agents jouant le rôle d'attaquant et engagé sur la mission. De même nous regroupons les buts *gInA* et *g10* en une mission d'amener le ballon dans la zone d'attaque, le but *mScore* en une mission de marquer un but et pour le schéma de défense, les buts *gInD*, *gInter* et *gToM* en une mission pour les défenseurs, les buts *gInD* et *gStop* en une mission pour le gardien de but et le but *gTake* en une mission d'éviter de prendre un but.

Formellement la spécification des missions pour les deux schémas de la FS se fait comme suit :

$\mathcal{M}_{Functional} = \{m1, m2, m3, m4\}$
$mo_{Functional}(m1) = \{gToM, gMhas, gMtoA\}$
$nm_{Functional}(m1) = (1, 4)$
$mo_{Functional}(m2) = \{gInPos, gKick, gGoal\}$
$nm_{Functional}(m2) = (1, 3)$
$mo_{Functional}(m3) = \{gInA, g10\}$
$nm_{Functional}(m3) = (1, 7)$
$mo_{Functional}(m4) = \{gScore\}$
$nm_{Functional}(m4) = (1, 7)$

$\mathcal{M}_{Defense} = \{m5, m6, m7\}$
$mo_{Defense}(m5) = \{gInD, gInter, gToM\}$
$nm_{Defense}(m5) = (1, 3)$
$mo_{Defense}(m6) = \{gInD, gStop\}$
$nm_{Defense}(m6) = (1, 1)$
$mo_{Defense}(m7) = \{gTake\}$
$nm_{Defense}(m7) = (1, 4)$

Les missions sont maintenant prêtes à être assignées comme droit ou devoir aux différents rôles à l'aide d'opérateurs déontiques. C'est ce que propose la Spécificaton Déontique de $\mathcal{M}$OISE$^+$. Nous considérons que cette spécification doit se faire avec plus de précisions, notamment en prenant en compte le contexte dans lequel l'agent se trouve. C'est pour cela que nous allons définir un ensemble de contextes dans lesquels l'organisation se retrouvera et qui aura une influence sur la définition des droits et devoirs. Cette Spécification Contextuelle que nous allons maintenant présenter ajoute également de la dynamicité en définissant un enchaînement d'états déclenchés par l'apparition d'événements.

6.4 Spécification Contextuelle

6.4.1 Concepts et définitions

Nous avons utilisé dans le Chapitre 2 le terme de "contexte". Les contextes sont utilisés pour définir les différentes situations des acteurs dans lesquelles ils jouent un rôle . Le fait de jouer un rôle plonge l'acteur dans un contexte différent. Dans le Chapitre 3, STEAM utilise des contextes qu'il définit comme un ensemble de règles "if ... then ... else". Cependant les contextes ne sont pas toujours perçus comme étant simplement l'état d'un environnement prédéfini avec un ensemble de ressources d'interactions fixées. Ils peuvent aussi être un moyen d'influencer le comportement d'agents intelligents. Turner dit dans [98] qu'un contexte est toute configuration identifiable des dispositifs environnementaux liée aux missions et aux agents et utilisée pour prévoir un comportement.

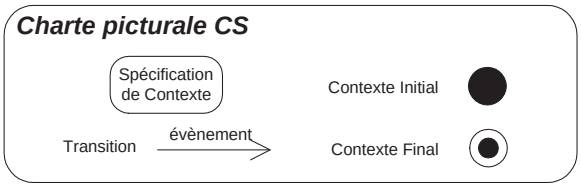


FIG. 6.8 – Éléments de la représentation graphique de la CS

Si nous reprenons notre exemple du code de la route, les contextes "route" et "autoroute" font partie d'un contexte de "conduite sur route" différent du contexte de "conduite sur circuit fermé". La transition d'un contexte à un autre se fait en passant l'entrée du circuit et au sein du contexte de "conduite sur route", en passant un péage autoroutier. Nous avons ainsi des contextes composés de sous-contextes liés par des transitions déclenchées lorsqu'un événement survient. Cette spécification ressemble ainsi à un diagramme d'état-transition UML puisqu'elle reprend les notions

d'états (contextes) et de transitions avec possibilité d'avoir des états hybrides. Ainsi la représentation graphique de cette spécification utilisera les éléments de la FIG 6.8. Comme les autres spécifications, nous pouvons exprimer la CS à l'aide de BNF mettant en scène les deux concepts principaux de *contexte* et de *transition* que nous allons détailler dans ce qui suit.

Ci-dessous nous précisons notre définition de la CS, puis nous verrons celles des contextes et des transitions dans les sections suivantes.

> Une *Spécification Contextuelle* (CS) est l'ensemble des ***contextes*** dans lesquels l'Organisation peut évoluer *a priori* ainsi que l'ensemble des ***événements*** déclenchant le changement de contexte.

Formellement, une Spécification Contextuelle se compose de $\mathcal{E}$, l'ensemble des événements pouvant survenir au sein de l'Organisation et de $\mathcal{C}$, l'ensemble des Contextes.

$$\mathcal{CS} = \langle \mathcal{E}, \mathcal{C} \rangle$$

6.4.2 Contextes

> Un ***Contexte*** est l'identification d'un état particulier dans lequel l'Organisation se trouve. Ce contexte peut être composé de sous-contextes liés entre eux par des ***transitions*** déclenchées par des ***événements*** et faisant passer l'Organisation d'un contexte à un autre.

Le plus souvent, la CS est composée d'un contexte global composé lui-même de sous-contextes. Nous ne représentons pas ce contexte global graphiquement afin de ne pas alourdir le schéma. Formellement, un contexte se définit ainsi :

$$c = \langle \mathcal{SC}_c, trans_c, init_c, final_c \rangle$$
$$init_c \in \mathcal{SC}_c$$
$$final_c \in \mathcal{SC}_c$$

où $\mathcal{SC}_c$ est l'ensemble des sous-contexte d'un contexte c, $trans_c$ est une fonction faisant passer d'un contexte de $\mathcal{SC}_c$ à un autre contexte de $\mathcal{SC}_c$ en fonction d'un événement et $init_c$ et $final_c$ sont les sous-contextes spéciaux de début et de fin du contexte c. Il ne peut y avoir qu'un seul contexte initial et qu'un seul contexte final par contexte.

Dans une instance d'Organisation, dès lors qu'un contexte est rendu actif par un événement, il appartient à l'ensemble $\mathcal{AC}ontext$ définissant les instances de contextes. Avec la décomposition des contextes en sous-contextes, l'Organisation peut très bien se trouver dans plusieurs contextes actifs en même temps. De ce fait, les contraintes liées à chaque contexte s'additionnent. Il faudra veiller cependant à ne pas arriver à des situations où chaque contexte définit des normes contradictoires pour un même rôle. Nous verrons cela dans la définition des normes.

6.4.3 Transitions

> Les transitions définissent la manière dont l'organisation passe d'un contexte à un autre en fonction d'un ***événement*** déclencheur.

Les transitions sont l'application de la fonction partielle *trans* faisant correspondre à un sous-contexte d'un contexte et à un événement de la CS un autre sous-contexte du contexte.

$$trans_c : \mathcal{SC}_c \times \mathcal{E} \nrightarrow \mathcal{SC}_c$$
$$\forall\, event \in \mathcal{E}, \forall\, c_i \in \mathcal{SC}_c : trans_c(c_i, event) \neq init$$
$$\forall\, event \in \mathcal{E}, \forall\, c_j \in \mathcal{SC}_c : trans_c(final, event) \neq c_j$$

L'ensemble des contextes et des transitions d'un contexte définit une suite non interrompue de contextes partant du contexte initial et arrivant au contexte final. Par ce cheminement entre contextes, nous ajoutons une dynamicité supplémentaire à celle définie au sein d'un schéma fonctionnel de la Spécification Fonctionnelle.

6.4.4 Exemple

La CS, comme les autres spécifications d'une modélisation $\mathcal{M}\text{OISE}^{Inst}$, est optionnelle. Mais en général, c'est la seule dont on peut vraiment se passer. Pour l'exemple des "Robots Footballeurs", le but de synchronisation *gToM* permet de faire le lien entre les deux schémas. Par contre nous pouvons nous poser la question de savoir si les agents doivent essayer d'accomplir leurs missions tout le temps ou seulement de temps en temps. Si nous laissons la spécification telle quelle, il se peut que des défenseurs viennent jusque dans la zone d'attaque pour intercepter le ballon ou que des milieux de terrain situés à l'arrière fassent une passe à un attaquant en bonne position à l'autre bout du terrain. Pour éviter cela, nous définissons trois contextes étant actifs en fonction de la position du ballon sur le terrain.

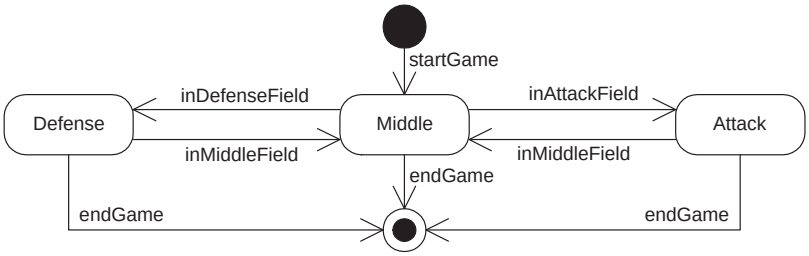


FIG. 6.9 – CS de l'exemple des "Robots Footballeurs"

La FIG. 6.9 représente la CS pour l'exemple des "Robots Footballeurs". Nous y retrouvons les deux contextes spéciaux *init* et *final*. Le contexte initial possède une transition se déclenchant lorsque la partie commence et activant le contexte "Middle". Le ballon étant mis en jeu au centre du terrain, c'est tout à fait logique. Ensuite, selon les événements, le contexte "Defense" ou "Attack" sera actif. Ces événements seront créés en fonction de la position du ballon sur le terrain. A ce stade de la spécification, nous ne nous intéressons ni comment, ni par qui ces événements seront créés. La partie pouvant se terminer où que soit le ballon, tous les contextes ont une transition déclenchée par le même événement faisant passer l'Organisation dans le contexte final mettant ainsi fin au cycle de vie de l'Organisation. En effet, quand la CS existe, elle permet de définir un cycle de vie. Tant que l'état final du CS global n'est pas atteint, l'Organisation continue d'évoluer. La CS de cet exemple est définie formellement ci-dessous :

$$
\begin{array}{l}
\mathcal{CS} = \{\{startGame, inDefenseField, inAttackField, inMiddleField, endGame\}, \{playtime\}\} \\
\mathcal{SC}_{playtime} = \{Defense, Middle, Attack\} \\
trans_{playtime}(init, startGame) = Middle \\
trans_{playtime}(Middle, inDefenseField) = Defense \\
trans_{playtime}(Middle, inAttackField) = Attack \\
trans_{playtime}(Middle, endGame) = final \\
trans_{playtime}(Defense, inMiddleField) = Middle \\
trans_{playtime}(Defense, endGame) = final \\
trans_{playtime}(Attack, inMiddleField) = Middle \\
trans_{playtime}(Attack, endGame) = final
\end{array}
$$

La Spécification Contextuelle actuelle que nous proposons dans $\mathcal{M}\text{OISE}^{Inst}$ peut être considérée, *a priori*, comme trop peu détaillée. En effet, par rapport à ISLANDER [105, 37] étudié précédemment

et sa *"Performative Structure"*, les transitions entre les contextes (appelés scènes) ne font pas intervenir une synchronisation entre les rôles et nous ne définissons pas de protocole à suivre au sein de ces contextes c'est-à-dire nous ne spécifions pas ce qui se passe pour chacun des rôles au sein de ces contextes. Cependant, ce manque peut être comblé par notre FS qui ne trouve par contre pas d'équivalent dans ce langage de description d'Institution Électronique. De plus, cette *"Performative Structure"* et ces protocoles peuvent être vus comme des spécifications très rigides d'une Institution et peuvent enlever une part de dynamicité et de flexibilité que nous recherchons dans notre modèle.

Nous avons maintenant tous les éléments pour pouvoir contraindre un agent. Selon le rôle qu'il jouera ou le groupe auquel il appartiendra, nous pouvons lui permettre ou l'obliger de réaliser ou non une mission dans un contexte particulier. Il nous faut pour cela lier ces trois composants avec un opérateur déontique afin de définir de cette manière une norme.

6.5 Spécification Normative

Nous venons de décrire dans les sections précédentes la façon dont nous définissons un ensemble de contraintes pour les agents à l'aide de trois spécifications. Lors de l'exécution de l'Organisation, ils devront ainsi jouer des rôles au sein de groupes tel que spécifié dans la SS, atteindre un ensemble de buts regroupés en missions définis dans la FS et enfin évoluer dans un ensemble de contextes tel qu'il est prévu dans la CS. Il nous faut maintenant lier ces spécifications afin de pouvoir exprimer des règles concernant un rôle ou un groupe accomplissant une mission dans un contexte particulier. Afin d'établir un lien entre SS, FS et CS comme illustré sur la Fig. 6.1 nous définissons la Spécification Normative (NS) composée d'un ensemble de normes. Nous avons vu dans le Chapitre 2 les différents sens accordés aux contraintes normatives telles que les règles, les lois ou les normes et dans la Section 3.2.1 différentes définitions de normes dans le domaine des SMA. Nous utilisons ici le sens du mot norme considéré dans le domaine légal et nous nous inspirons de la définition de [13]. Remarquons que le principe du découpage du modèle en spécifications évite de lier les normes directement aux agents.

> La Spécification Normative est composée d'un ensemble de normes mettant en relation la SS, la FS et éventuellement la CS via un contexte, un émetteur, un porteur, une mission et un opérateur déontique.

Considérons les ensembles $\mathcal{C}_{cs}$, $\mathcal{SE}_{ss}$ et $\mathcal{M}_{fs}$, regroupant les contextes, les entités structurelles et les missions de l'Organisation. Quant à $\mathcal{NS}$, il est l'ensemble des normes de l'organisation.

$$
\begin{array}{l}
n \in \mathcal{NS} \\
n = \varphi \rightarrow op(cont, issuer, bearer, m, sanc, w, tc) \\
op \in \{obl, per, for\} \\
cont \in \mathcal{C}_{cs} \\
issuer \in \mathcal{SE}_{ss} \\
bearer \in \mathcal{SE}_{ss} \\
m \in \mathcal{M}_{fs} \\
sanc \in \mathcal{NS} \\
w \in \mathbb{N}
\end{array}
$$

avec :
- φ est une expression des conditions de validité de la norme ;
- op est l'opérateur déontique définissant une Obligation, une Permission ou une Interdiction ;
- $cont$ est un contexte spécifié dans la CS précisant l'activation de la norme ;
- $issuer$ est une entité structurelle spécifiée dans la SS, considérée comme l'émettrice de la norme ;
- $bearer$ est une entité structurelle spécifiée dans la SS, considérée comme la porteuse de la norme ;
- m est une mission spécifiée dans la FS précisant l'action sur laquelle porte la norme ;
- $sanc$ est une sanction à appliquer en cas de violation de la norme ;
- w est la pondération de la norme ;

– tc est une contrainte temporelle définissant la date ou la durée de validité de la norme.

Nous réduisons la définition d'une norme à $op(cont, issuer, bearer, m)$ dans ce qui suit et jusqu'à la Section 6.5.2 afin de faciliter la lecture des expressions mathématiques.

Nous venons de définir qu'une norme de la Spécification Normative d'une Institution Électronique de type $\mathcal{M}\text{OISE}^{Inst}$ représente une expression des droits et des devoirs en termes d'obligations (O), de permissions (P) et d'interdictions (F) des rôles joués par les participants d'une société [99] en faisant le lien entre les rôles et les groupes de la SS et les missions de la FS dans un contexte particulier de la CS. En plus de cela, nous associons à ces normes des sanctions (pénalités et récompenses) qui guident les comportements des agents pour leur évolution dans le système. Nous ajoutons également la possibilité de conditionner une norme. Enfin nous gardons la possibilité de définir une contrainte temporelle de validation de la norme provenant de $\mathcal{M}\text{OISE}^{+}$ et nous ajoutons également une pondération permettant, nous allons le voir, d'éviter que deux normes ne deviennent conflictuelles.

Il est plus difficile de donner une représentation graphique pour cette spécification. Une façon de faire est de reprendre dans un tableau chacune des informations définissant une norme. Nous pouvons également regrouper les normes en fonction de ce sur quoi elles portent, par exemple dans le tableau 6.2, nous regroupons les normes en fonction du contexte dans lequel elles s'appliquent. Nous pourrions faire la même chose avec le porteur de la norme ou encore combiner les deux. Nous allons voir en détail dans ce qui suit chacun des liens avec les autres spécifications ainsi que les informations complémentaires néanmoins indispensables à la définition, à l'interprétation et au suivi des normes.

			$\mathcal{CS}$	$\mathcal{SS}$		$\mathcal{FS}$	$\mathcal{NS}$		
n	φ	op	$cont$	$issuer$	$bearer$	m	$sanc$	w	tc
N1	- - -	obl	$Begin$	r_1	r_0	$m1$	- - -	1	- - -
N3	- - -	obl	$Game$	r_2	gr_1	$m2$	- - -	1	- - -
N4	- - -	obl	$Game$	r_2	r_4	$m4$	- - -	1	- - -
N7	- - -	for	$Game$	r_3	r_4	$m16$	- - -	2	- - -
N8	- - -	obl	End	r_3	r_5	$m5$	- - -	3	- - -
N9	- - -	for	End	r_2	r_4	$m4$	- - -	1	- - -

TAB. 6.2 – Exemple de représentation de la NS

6.5.1 Liens avec les autres spécifications

La Spécification Normative utilise les éléments des autres spécifications pour définir les normes. Nous voyons dans les sections suivantes le détail de chacun des liens existant entre la définition des normes et les éléments de la CS, de la SS, de la FS mais également de la NS elle-même.

Lien avec la CS

La définition du contexte d'une norme permet de définir l'état de l'organisation dans lequel la norme sera applicable. Elle nous permet aussi de donner indirectement une durée de vie à notre norme. En effet, si nous considérons un contexte c_1 déclenché par l'événement ev_1 et prenant fin avec l'application d'une transition déclenchée par l'événement ev_2, nous pouvons dire que la norme est active entre l'apparition des événements ev_1 et ev_2.

Même si l'organisation peut être dans plusieurs contextes au même moment, une norme ne concerne qu'un et un seul contexte à la fois. Il peut donc arriver dans des systèmes assez complexes d'analyser et de vérifier si un agent peut se retrouver à jouer deux rôles différents, étant influencé par des contextes différents et devant donc respecter deux normes contradictoires. L'agent se retrouve donc en présence d'un conflit. Cela peut être résolu grâce à la pondération de la norme que nous

détaillons dans la section 6.5.2. Il est toutefois possible de ne pas préciser le contexte dans lequel la norme est active. Les normes restent ainsi applicables durant toute la durée de vie de l'Organisation quel que soit l'état d'une éventuelle CS.

Enfin, la composition des contextes en sous-contextes entraîne que si une norme est active dans un contexte et ce, quel que soit l'opérateur déontique, alors elle l'est aussi dans un de ses sous-contextes :

$$c_1 \in \mathcal{C}_{os}$$
$$op(c_1, issuer, bearer, m) \wedge c_2 \in \mathcal{SC}_{c_1} \Rightarrow op(c_2, issuer, bearer, m)$$

Liens avec la SS

Le porteur d'une norme (*bearer*) est une entité structurelle de la SS sur laquelle s'applique la norme. C'est-à-dire qu'une norme peut s'appliquer sur un rôle ou un groupe et dans ce cas, tous les rôles appartenant à ce groupe doivent respecter la norme. L'agent jouant le rôle impliqué dans la norme ou jouant un rôle appartenant à un groupe impliqué dans la norme devra se comporter de telle manière que la partie fonctionnelle de la norme devienne ou reste vraie.

Une norme est également définie vis-à-vis d'une autre entité structurelle considérée comme l'entité émettrice de la norme (*issuer*). Cet autre rôle ou groupe qui sera désigné ici aura en charge de surveiller le respect de la norme. C'est à cette entité que revient la tâche de vérifier que la partie fonctionnelle de la norme devienne ou reste vraie. Si l'entité structurelle est un groupe, n'importe quel rôle appartenant au groupe a la responsabilité de détecter une violation. En cas de non-respect d'une norme, le(s) rôle(s) *issuer* doi(ven)t déclencher et éventuellement appliquer une sanction si cette dernière est définie.

Quel que soit l'opérateur de la norme, si le porteur est un rôle alors l'héritage des rôles fera qu'une norme concernant le rôle *bearer* concernera également le rôle $bearer'$ si $bearer'$ hérite de *bearer*. Idem pour l'émetteur de la norme :

$$op(cont, issuer, bearer, m) \wedge bearer \in \mathcal{R} \wedge bearer \sqsubset bearer' \Rightarrow op(cont, issuer, bearer', m)$$
$$op(cont, issuer, bearer, m) \wedge issuer \in \mathcal{R} \wedge issuer \sqsubset issuer' \Rightarrow op(cont, issuer', bearer, m)$$
$$op(cont, issuer, bearer, m) \wedge bearer \in \mathcal{R} \wedge issuer \in \mathcal{R} \wedge bearer \sqsubset bearer' \wedge issuer \sqsubset issuer' \Rightarrow op(cont, issuer', bearer', m)$$

Quel que soit l'opérateur de la norme, si le porteur est un groupe alors la composition des groupes en rôles ou en sous-groupes fera qu'une norme concernant le groupe *bearer* concernera également le rôle ou le sous-groupe $bearer'$ si $bearer'$ appartient au groupe *bearer* ou est un sous-groupe de *bearer*. Idem pour l'émetteur de la norme :

$$op(cont, issuer, bearer, m) \wedge bearer \in \mathcal{G}r \wedge bearer' \in bearer \Rightarrow op(cont, issuer, bearer', m)$$
$$op(cont, issuer, bearer, m) \wedge issuer \in \mathcal{G}r \wedge issuer' \in issuer \Rightarrow op(cont, issuer', bearer, m)$$
$$op(cont, issuer, bearer, m) \wedge bearer \in \mathcal{G}r \wedge issuer \in \mathcal{G}r \wedge bearer' \in bearer \wedge issuer' \in issuer \Rightarrow op(cont, issuer', bearer', m)$$

Lien avec la FS

La partie fonctionnelle des normes renseigne la mission m sur laquelle la norme porte. C'est-à-dire que l'agent ou le groupe d'agents devant respecter cette norme aura à s'engager sur la mission dont il est question ici et devra l'accomplir. Son comportement sera alors respectueux de la Spécification Normative, ainsi l'agent ne risquera pas de se faire sanctionner par le ou les autre(s) agent(s) concerné(s) par cette norme.

Lien avec la NS

Dans une norme, nous pouvons faire référence à une autre norme (*sanc*) qui sera alors considérée comme une sanction. Nous définissons alors une norme de la façon suivante :

$$n = op(cont, issuer, bearer, m, sanc)$$
$$sanc \in \mathcal{NS}$$
$$sanc \neq n$$

Pour le porteur de la norme (*bearer*), cette référence pointe vers la norme qui sera valide, donc vers la mission à accomplir par l'émetteur (*issuer*) s'il ne respecte pas cette contrainte. En d'autres termes, ce peut être un aperçu de ce que le porteur risque s'il n'obéit pas. Pour l'émetteur, c'est la norme qu'il devra respecter afin de sanctionner celui ou ceux qu'il surveille. La mission d'une sanction fait souvent partie d'un schéma spécifique. Ainsi, pour que la mission référencée par la norme soit exécutée il faut auparavant que le schéma correspondant soit instancié. Une sanction a les mêmes caractéristiques qu'une norme, ce qui veut dire qu'elle peut également être violée et elle même faire référence à une sanction et ainsi de suite. Les sanctions pouvant être positives ou négatives, il est alors possible de créer un arbre de comportements avec deux choix (respect ou non de la norme) pour chaque nœud.

6.5.2 Informations normatives complémentaires

Nous venons de voir les références que font chaque norme de la NS aux autres spécifications de l'OS. Une norme concerne un porteur pour une mission donnée, sous la supervision d'un émetteur pouvant appliquer une sanction et tout ceci prenant effet dans un contexte. Cependant, il nous faut pouvoir définir si cette norme est un droit ou un devoir et éventuellement si sa validité est conditionnée ou limitée dans le temps. Enfin, une pondération nous permet de définir le poids d'une norme par rapport à une autre. Nous allons maintenant détailler chacune de ces informations.

Opérateur déontique

L'opérateur déontique *op* permet de définir la relation déontique activée par la norme. Elle peut prendre les valeurs *obl* pour une obligation, *per* pour une permission et *for* pour une interdiction.

$$n = op(cont, issuer, bearer, m, sanc)$$
$$op \in \{obl, per, for\}$$

Conditions

Les conditions permettent de définir si une norme est valide ou non. Elles sont exprimées sous forme de combinaisons booléennes de prédicats sur l'état de l'Organisation et évalués par les agents *issuer* et *bearer* pour déterminer l'applicabilité de la norme. Les prédicats de base utiles pour toute sorte d'application sont par exemple ceux testant la violation d'une norme (*violated*) ou encore les fonctions accédant aux propriétés de l'Organisation telles que *number* retournant le nombre d'agents appartenant à un groupe, ou *cardinalityMax* retournant le nombre d'agents maximum pouvant appartenir à un groupe. Une description complète des conditions en BNF se trouve en Annexe B.4.

Une norme *sanc* étant référencée par une norme n comme étant une sanction doit avoir dans ses conditions l'expression $violated(n)$ de telle sorte que :

$$n = \varphi \rightarrow op(cont, issuer, bearer, m, sanc)$$
$$sanc : violated(n) \rightarrow op(cont', issuer', bearer', m')$$

Une norme considérée comme une sanction ne possède pas forcément de référence à une sanction, ce champ étant facultatif.

Pondération

Dans une organisation complexe faisant intervenir de nombreux rôles et de nombreux contextes, comment pouvons-nous nous assurer qu'un agent jouant deux rôles différents ne se retrouve pas dans un ensemble de contextes l'obligeant à respecter deux normes contradictoires ? Si le cas survient,

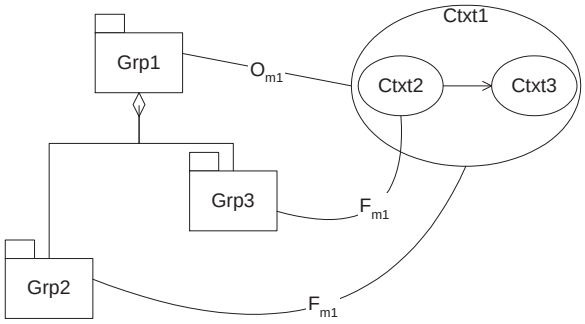


FIG. 6.10 – Exemple de conflit entre normes

comment l'agent peut-il faire son choix ?

Prenons l'exemple illustré sur la Fig. 6.10. Nous voyons que la SS a défini un groupe *Grp1* composé des sous-groupes *Grp2* et *Grp3*. De son côté, la CS a défini un contexte *Ctxt1* composé de deux sous-contextes *Ctxt2* et *Ctxt3*. Enfin, la NS définit une Obligation pour le groupe *Grp1* de faire la mission *m1* dans le contexte *Ctxt1* ainsi que l'Interdiction pour les *Grp1* et *Grp2* de faire *m1* dans respectivement *Ctxt2* et *Ctxt1*. Imaginons maintenant qu'un agent joue un rôle appartenant au groupe *Grp3* et que la société se trouve dans le contexte *Ctxt2*. La norme influençant cet agent directement est l'Interdiction de faire *m1* dans le contexte *Ctxt2*. Le groupe *Grp3* et le contexte *Ctxt2* définissant une composition, les normes sont également héritées. Ainsi l'agent est aussi influencé par la norme exprimant une Obligation de faire *m1* dans le contexte *Ctxt1*. L'agent est donc à la fois obligé et interdit de faire *m1*. Il devra dans tous les cas violer une norme, mais laquelle ?

Pour cela w définit la pondération d'une norme à l'aide d'un entier exprimant ainsi les normes comme suit :

$$n = \varphi \rightarrow op(cont, issuer, bearer, m, sanc, w)$$
$$w \in \mathbb{N}$$

Cet attribut permet de donner aux normes un ordre de priorité dans la NS pour un rôle donné. En supposant que le concepteur définisse des pondérations différentes, l'agent aura des indications sur la norme la plus importante à respecter en cas de conflit. Si cet attribut n'est pas suffisant, l'agent peut également raisonner sur la sanction encourue afin de choisir de violer la norme qui déclenchera la sanction la plus faible. Enfin si le conflit existe encore, l'agent devra faire un choix aléatoire. Mais cela dépend de la structure interne des agents et de leurs capacités de raisonnement, ce qui n'est pas de notre ressort pour les agents du domaine. Nous détaillerons dans le Chapitre 7 les agents institutionnels de la couche $\mathcal{S}$YNAI.

Contrainte temporelle

Les normes intégrant la définition d'une contrainte temporelle tc se représentent de la façon suivante :

$$n = \varphi \rightarrow op(cont, issuer, bearer, m, sanc, w, tc)$$
$$tc = (tOp, d)$$
$$tOp \in \{<, >, =\}$$
$$d \in \mathcal{D}$$

où $\mathcal{D}$ est l'ensemble des expressions de temps (date ou durée). Ainsi nous pouvons exprimer qu'une norme est valide avant une date, après une date ou durant une période. Cet attribut est optionnel. En effet, nous avons vu précédemment que l'expression d'une durée de validité pour une norme pouvait

se faire grâce à l'attribut du contexte. Si aucune contrainte de temps n'est exprimée, alors la norme est valide pendant toute la durée durant laquelle l'Organisation restera dans le contexte validant la norme. Si aucune contrainte de temps ni aucun contexte n'est exprimé, alors la norme est valide durant toute la durée de vie de l'Organisation.

6.5.3 Cycle de vie des normes

Afin que les spécifications de normes contraignent le comportement des agents en exprimant leurs droits et leurs devoirs, il faut qu'elles soient instanciées, c'est-à-dire qu'elles deviennent actives. Une norme devient active lorsque le contexte de sa spécification fait partie des contextes actifs. L'ensemble des normes actives $\mathcal{AN}orm$ est un couple composé de l'ensemble des normes et de leur statut.

$$\mathcal{AN}orm = \langle \mathcal{NS}, status \rangle$$

Une fois active, une norme peut ensuite être valide, respectée ou violée. C'est le résultat de la fonction *status*, combinaison de l'ensemble des contextes actifs, des conditions de validité, des contraintes temporelles et des instances de mission, qui détermine l'état de la norme.

$$status : \mathcal{AN}orm \times \mathcal{AC}ontext \times \varphi \times tc \times \mathcal{M}ission \rightarrow \{valid\ ,\ active\ ,\ respected\ ,\ violated\}$$

L'enchaînement des états d'une norme définit son cycle de vie (cf. Fig. 6.11). Dès qu'une instance de norme est créée, elle est active. Pour qu'une instance de norme devienne valide, il faut qu'elle soit active et que les conditions de validité soient vérifiées. A partir de ce moment, la norme sera respectée ou violée si la mission est accomplie ou non dans le laps de temps spécifié par la contrainte temporelle. La détection de la violation d'une norme est différente si l'opérateur déontique spécifie une obligation, une interdiction ou une permission.

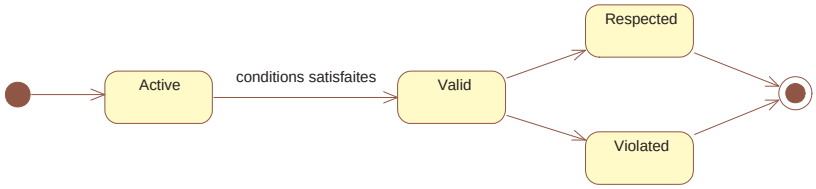


Fig. 6.11 – Diagramme d'état UML représentant le cycle de vie d'une norme

Obligation

- L'instance de norme est respectée quand la mission est accomplie par le porteur de la norme dans les temps spécifiés.
- L'instance de norme est violée quand la mission est accomplie par le porteur de la norme en dehors des délais spécifiés.
- L'instance de norme est violée quand la contrainte de temps est dépassée et que la mission n'est pas accomplie par le porteur de la norme.

De plus :

- Notre spécification des normes nous permet de définir un groupe comme porteur d'une norme. Dans ce cas-ci, lors de la vérification, si au moins un rôle du groupe respecte la norme, alors la norme est respectée pour le groupe entier.
- Notre spécification des normes nous permet d'omettre une contrainte temporelle. Si c'est le cas, la fin de l'activité du contexte concerné par la norme correspond à la date avant laquelle la norme doit être respectée.
- Quand une norme change d'état et ne devient plus valide ou plus active, une vérification du respect des normes concernées par ce changement est effectuée.

Interdiction

- L'instance de norme est respectée tant que la mission n'est pas accomplie par le porteur de la norme dans les limites de la contrainte temporelle.
- L'instance de norme est violée quand la mission concernée est accomplie dans les limites de la contrainte temporelle.

Nous considérons que les missions qui ne sont pas contraintes par une interdiction sont autorisées.

Permission

Une permission est considérée comme une obligation ne pouvant pas être violée. Les permissions sont utiles dans les systèmes où l'on considère que les missions non autorisées sont interdites. Comme précisé ci-dessus, nous ne nous trouvons pas dans ce contexte. Nos agents ne restreignent pas leurs actions à ce qui est autorisé ou obligé mais à toutes les missions qu'ils sont capables d'accomplir et qui ne leur sont pas interdites.

6.5.4 Exemple

Reprenons l'exemple des "Robots Footballeurs". Le Tab. 6.3 définit les normes que devront respecter les agents qui feront partie de l'Organisation. Elles expriment plus particulièrement les devoirs de chaque joueur en fonction du poste qu'il occupe et de la position du ballon sur le terrain (les contextes).

n	φ	op	cont	issuer	bearer	m	sanc	w	tc
N01	---	obl	Attack	Coach	Attack	m4	N10	1	---
N02	---	obl	Attack	Coach	Attacker	m2	---	1	---
N03	---	per	Attack	Coach	Middle	m1	---	1	---
N04	---	obl	Attack	Coach	Attack	m3	---	1	---
N05	---	obl	Middle	Coach	Middle	m1	---	1	---
N06	---	per	Middle	Coach	Back	m5	---	1	---
N07	---	obl	Defense	Coach	Player	m7	---	1	---
N08	---	obl	Defense	Coach	Back	m5	---	1	---
N09	---	obl	Defense	Coach	Goalkeeper	m6	---	1	---
N10	violated(N01)	obl	---	Coach	Player	m7	---	1	---

Tab. 6.3 – NS de l'exemple des "Robots Footballeurs"

Le concepteur pensant que pour un équipe de football, le plus important est de marquer un but, la première norme définie ici spécifie que le groupe "Attack" a l'Obligation de s'engager et donc d'accomplir la mission *m4* (laquelle, rappelons-le, est composée du seul but de marquer un but) dans le contexte "Attack" (c'est à dire quand le ballon est dans la zone d'attaque) sans contrainte de temps ni de condition de validité. La norme concernant le groupe "Attack", tous les agents jouant un rôle appartenant au groupe "Attack" ("Attacker", "Middle" et "Leader") seront soumis à cette norme et devront la respecter s'ils le peuvent. C'est à dire qu'ils devront accomplir la mission *m4*, donc faire en sorte qu'un but soit marqué. Pour ce faire, comme nous l'avons vu dans la FS, quatre sous-buts doivent être atteints. Cette seule norme déclenchant l'obligation d'atteindre tous les buts du schéma pour marquer un but pourrait suffire. Mais dans ce cas, il ne serait pas spécifié qui doit atteindre les sous-buts. D'autres normes doivent être définies. A noter également que l'émetteur de la norme est le rôle "Coach", chose normale étant donné que c'est le seul à avoir l'autorité sur tous les autres rôles. Il est d'ailleurs l'émetteur de toutes les normes de cette NS. Cette première norme se formalise comme suit :

$$N01 = obl(Attack, Coach, Attack, m4, N10, 1,)$$

La deuxième norme oblige le rôle "Attacker" à accomplir la mission *m2* toujours dans le contexte "Attack". On restreint donc ici les agents qui peuvent atteindre un ensemble de sous-buts de *gScore*. Nous précisons en fait que dans un contexte de ballon dans la zone d'attaque, les attaquants doivent

accomplir la mission d'attaquer. De même les normes N05, N08 et N09 obligent le rôle "Middle" à accomplir la mission de milieu de terrain quand le ballon est dans le milieu du terrain, le rôle "Back" à défendre quand le ballon est en zone de défense et le gardien à arrêter le ballon quand le ballon est également en zone de défense. Cependant les normes N03 et N06 autorisent (Permission) le rôle "Middle" à accomplir la mission de milieu de terrain quand le ballon est en zone d'attaque et le rôle "Back" à accomplir la mission de défense quand le ballon est en milieu de terrain. Enfin la norme N04 oblige tous les membres du groupe "Attack" à faire en sorte de passer le ballon à un attaquant dans un contexte d'attaque et la norme N07 oblige tous les joueurs de l'équipe à éviter de prendre un but quand le ballon est dans la zone de défense. Même si le rôle "Player" ne peut pas être joué par un agent, l'héritage fait que tous les sous-rôles "Back", "Leader", "Middle", "Attacker" et même "Goalkeeper" sont soumis à cette norme. Si la norme n'est pas respectée, c'est à dire si l'équipe prend un but, tous les joueurs sont responsables. La formalisation des normes se fait ainsi :

$$
\begin{aligned}
&N02 = obl(Attack, Coach, Attacker, m2, , 1,)\\
&N03 = per(Attack, Coach, Middle, m1, , 1,)\\
&N04 = obl(Attack, Coach, Attack, m3, , 1,)\\
&N05 = obl(Middle, Coach, Middle, m1, , 1,)\\
&N06 = per(Middle, Coach, Back, m5, , 1,)\\
&N07 = obl(Defense, Coach, Player, m7, , 1,)\\
&N08 = obl(Defense, Coach, Back, m5, , 1,)\\
&N09 = obl(Defense, Coach, Goalkeeper, m6, , 1,)
\end{aligned}
$$

Dans cet exemple, toutes les normes sont des obligations ou des permissions ne concernant pas deux fois la même mission dans le même contexte et nous n'avons pas de sous-contextes. De ce fait, aucune norme ne peut être en conflit avec une autre. Cela explique la pondération à '1' de toutes les normes. L'absence de définition d'une *deadline* pour toutes les normes fait en sorte qu'une fois activées par l'entrée dans le contexte correspondant, les normes restent valides pendant toute la durée de l'activation de ce contexte. La norme N10 n'ayant pas de contexte, celle-ci est donc active pendant toute la durée de vie de l'organisation. Cependant, elle ne sera valide que selon une certaine condition, celle-ci étant que la norme N01 ne soit pas respectée. Nous avons donc affaire ici à une sanction. D'ailleurs, la norme N01 fait référence à cette norme N10 en tant que sanction. Ainsi, si la norme N01 n'est pas respectée, c'est-à-dire que pour une raison ou pour une autre un but n'est pas marqué, tous les joueurs doivent se mettre en défense et éviter de se prendre un but et donc pour cela récupérer la balle. Cela se traduit formellement de la façon suivante :

$$N10 = violated(N01) \rightarrow obl(, Coach, Player, m7, , 1,)$$

6.6 Synthèse

Nous avons présenté dans ce chapitre le modèle $\mathcal{M}\text{OISE}^{Inst}$, une extension de $\mathcal{M}\text{OISE}^{+}$ [**?**] nous permettant de prendre en compte une dimension normative plus complète que la dimension déontique originale ainsi qu'un ensemble de contextes ajoutant un autre niveau de contrainte du comportement des agents. Ainsi les Spécifications Structurelles et Fonctionnelles structurent le fonctionnement que chaque agent possède en définissant que la capacité d'un agent à atteindre un certain but sera liée dans cette organisation à l'accomplissement d'une mission, et que sa capacité à communiquer ne devra pas être utilisée avec des agents jouant certains rôles. Quant aux Spécifications Contextuelles et Normatives, elle définissent sur base de ce fonctionnement structuré les comportements que les agents devront avoir suivant des contextes et des conditions de validité. Ainsi, nous voyons le comportement comme un sous-ensemble normé des capacités de fonctionnement des agents. Cette séparation en différentes spécifications que propose $\mathcal{M}\text{OISE}^{Inst}$ permet une plus grande flexibilité au niveau du modèle global via la possibilité de manipuler chacune de ces spécifications indépendamment les unes des autres. Les normes, éléments essentiels de $\mathcal{M}\text{OISE}^{Inst}$ sont définies comme des relations entre un rôle ou un groupe et une mission dans un contexte donné ajoutant ainsi à la spécification déontique de $\mathcal{M}\text{OISE}^{+}$ un ensemble d'informations inspirées par différents travaux.

Les opérateurs déontiques nous permettent de différencier un droit (permission) d'un devoir (obligation) qui définissent les limites de comportement des agents comme dans [27]. Nous inspirant de [88], nous avons étoffé la contrainte posée par une norme par l'intermédiaire d'une limite temporelle, d'une condition d'activation de la norme, ainsi que par l'émetteur de la norme. Puis, comme dans [38], nous avons ajouté une condition supplémentaire de validité en faisant référence à un contexte. Enfin, les organisations complexes mettant en œuvre plusieurs rôles non-abstraits compatibles et héritant les uns des autres avec des schémas fonctionnels faisant intervenir de nombreux buts héritant, et ceci dans beaucoup de contextes, peuvent faire apparaître des conflits. Une pondération allouée à chaque norme définit ainsi un ordre de priorité entre les normes. Cette aide au choix d'une norme à respecter en cas de conflit peut également se faire avec un raisonnement sur les sanctions encourues. Les sanctions ont été ajoutées à l'origine pour faire en sorte d'informer les agents des risques qu'ils encouraient à enfreindre une norme. Ainsi leur décision serait prise en sachant quelle action serait mise en place contre eux. Cependant, avoir une société d'agents auto-organisée et auto-supervisée n'étant pas notre but, d'autres agents faisant partie intégrante de la plate-forme d'Institution Électronique doivent intervenir.

Ainsi, deux sortes d'agents évoluent dans notre organisation : les agents du domaine et les agents institutionnels. $\mathcal{M}$OISE^{Inst} ne cherche pas à imposer une homogénéité des agents qui constituent l'organisation. Cependant nous pouvons spécifier les fonctionnalités de la couche intergicielle ($\mathcal{S}$YNAI) dédiée à la régulation du système. En effet, MA$\mathcal{B}_{\text{ELI}}$ fournit un ensemble d'agents de supervision dont la mission est de détecter les violations de normes et de punir les agents fautifs. Le méta-modèle $\mathcal{M}$OISE^{Inst} a l'avantage de fournir tout ce dont nous avons besoin pour structurer les capacités de ces agents en missions afin de définir des comportements adéquates en fonction des violations des agents du domaine. De plus, les liens d'autorités faciliteront l'application de sanctions. Nous allons donc maintenant décrire la couche $\mathcal{S}$YNAI en présentant ces agents institutionnels qui joueront le rôle de superviseur ainsi que leur modèle d'organisation $\mathcal{M}$OISE^{Inst} propre.

Chapitre 7

*S*ynai : Système de Supervision d'Institution Multi-Agents

Le modèle $\mathcal{M}$OISE^{Inst} que nous venons d'étudier nous permet de spécifier des Institutions Électroniques basées sur une organisation normée. La spécification est l'aspect statique de l'organisation. En effet, nous définissons seulement les différentes contraintes structurelles, fonctionnelles, contextuelles et normatives qui influenceront un ensemble d'agents. Comme nous l'avons abordé dans la problématique (cf. Chapitre 1), les agents autonomes soumis à un ensemble de contraintes peuvent ne pas les respecter. Les agents seront effectivement soumis aux contraintes lorsque la Spécification de l'Organisation (l'OS) sera instanciée en une Organisation. C'est à ce moment qu'il faut contrôler que l'instance de l'organisation correspond à sa spécification. C'est le rôle du système d'arbitrage que nous proposons dans MA$\mathcal{B}_{\text{ELI}}$, capable d'arbitrer tout type de modélisation d'Institution Électronique faite avec $\mathcal{M}$OISE^{Inst}.

Avant d'entrer dans les détails de ce système d'arbitrage nommé $\mathcal{S}$YNAI, nous allons voir dans la première section la façon dont l'OS est instanciée en une Organisation. Puis dans la section 7.2, nous donnerons un aperçu des différentes actions qu'un agent peut mettre en place au sein de l'Organisation. La section 7.3 définira le système d'arbitrage tandis que les sections 7.4 et 7.5 entreront en détail dans les mécanismes d'arbitrage.

7.1 Instance d'Organisation

Le modèle $\mathcal{M}$OISE^{Inst} permet de définir des instances d'organisation sur lesquelles sont basées des Institutions Électroniques. L'élément principal est l'OS, à savoir la Spécification de l'Organisation. Cette spécification peut être appliquée à une société d'agents afin de les contraindre en les structurant et en définissant les différentes fonctionnalités qu'ils pourront et ne pourront pas faire au sein de cette société. Une même spécification peut servir pour plusieurs sociétés d'agents. Nous appelons ***Entité Organisationnelle*** (***OE***) ou ***Instance d'Organisation*** l'application d'une OS $\mathcal{M}$OISE^{Inst} sur une société d'agents.

> Une ***entité organisationnelle*** *oe* appartient à l'ensemble $\mathcal{OE}$ instanciant une spécification organisationnelle ***OS***. Elle est constituée d'une structure d'agents (***Structure***), d'un fonctionnement d'agents (***Functioning***) et des contextes et normes actifs (***Active Contexts*** et ***Active Norms***).

Comme nous le constatons sur la FIG. 7.1 une OS est instanciée en OE à partir du moment où un ensemble d'agents viennent jouer des rôles et accomplir des missions au sein de l'organisation. Formellement, nous représentons ceci de la manière suivante :

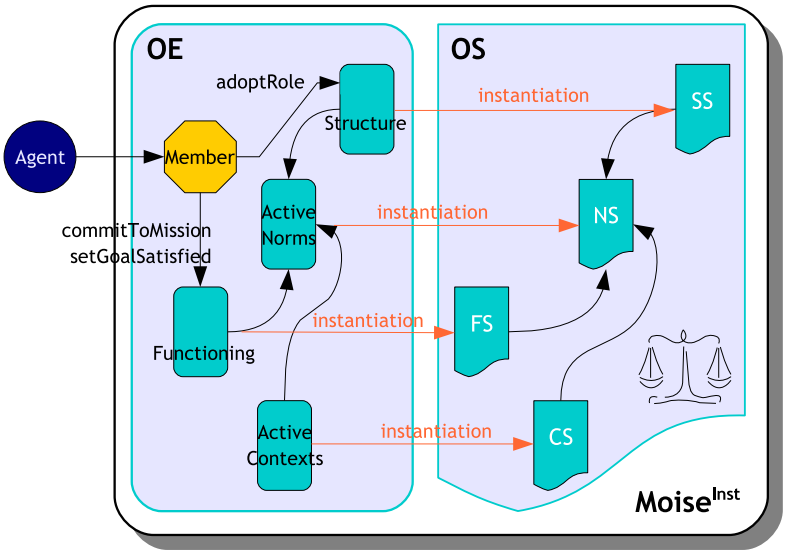


FIG. 7.1 – Instance d'une organisation $\mathcal{M}\text{OISE}^{Inst}$

$$
\begin{array}{l}
\mathcal{OE} = \langle \mathcal{OS}, \mathcal{A} \rangle \\
oe \in \mathcal{OE} \\
oe = \langle \mathcal{S}tructure, \mathcal{F}unctioning, \mathcal{AC}ontext, \mathcal{AN}orm, member \rangle \\
\mathcal{S}tructure = \langle \mathcal{G}roup, \mathcal{R}ole \rangle \\
\mathcal{F}unctioning = \langle \mathcal{M}ission, \mathcal{G}oal, \mathcal{S}cheme \rangle \\
\mathcal{AC}ontext \subset \mathcal{C}_{cs} \\
\mathcal{AN}orm = \langle \mathcal{NS}, status \rangle \\
member : \mathcal{OE} \rightarrow \mathcal{A} \\
member(\mathcal{OE}) \subset \mathcal{A}
\end{array}
$$

où $\mathcal{S}tructure$ est l'instanciation de la SS, $\mathcal{G}roup$ est l'ensemble des instances de groupes, $\mathcal{R}ole$ est l'ensemble des instances de rôles (cf. Section 6.2.2), $\mathcal{F}unctioning$ est l'instance de la FS, $\mathcal{M}ission$ est l'ensemble des instances de missions, $\mathcal{G}oal$ est l'ensemble des instances de buts, $\mathcal{S}cheme$ est l'ensemble des instances de schémas, $\mathcal{AC}ontext$ est l'ensemble des contextes actifs, $\mathcal{AN}orm$ est l'ensemble des normes ayant un statut et enfin *member* est une fonction retournant tous les agents appartenant à une instance d'organisation. Pour toutes les instances, nous définissons la fonction *spec* qui à une instance fait correspondre sa spécification. C'est la fonction inverse de la fonction *instantiate*. Nous définissons formellement ces deux fonctions de la manière suivante :

$$
\begin{array}{l}
instantiate : \mathcal{OS} \rightarrow \mathcal{OE} \\
spec : \mathcal{OE} \rightarrow \mathcal{OS}
\end{array}
$$

Les agents membres d'une Organisation peuvent jouer plusieurs rôles et s'engager sur plusieurs missions au sein de cette Organisation. Nous définissons pour cela plusieurs fonctions. Les rôles pouvant être joués au sein d'instances de groupes, nous avons besoin de la fonction *group* permettant de connaître les instances de groupe d'une OE et de la fonction $igroup_{gr}$ retournant toutes les instances d'un groupe de $\mathcal{SG}r_{gr}$ au sein d'une instance du groupe gr.

$$
\begin{array}{l}
group : \mathcal{OE} \rightarrow \mathcal{G}roup \\
igroup_{gr} : \mathcal{SG}r_{gr} \times \mathcal{G}roup \rightarrow \mathcal{G}roup
\end{array}
$$

Concernant les rôles, la fonction *role* permet de connaître toutes les instances de rôles appartenant à une instance de groupe, la fonction $irole_r$ retourne toutes les instances de rôle d'une spécification

de rôle r appartenant à une instance de groupe et enfin *prole* retourne toutes les instances de rôle qu'un agent joue au sein d'une Organisation.

$$role : \mathcal{G}roup \rightarrow \mathcal{R}ole$$
$$irole_r : \mathcal{G}roup \rightarrow \mathbb{P}(\mathcal{R}ole)$$
$$prole : \mathcal{A} \times \mathcal{OE} \rightarrow \mathcal{R}ole$$

Sur base de ces fonctions, nous définissons un ensemble de prédicats permettant de savoir :
- si un agent a est membre de l'OE :
  $a \in \mathcal{A}, oe \in \mathcal{OE}, is\_member(a, oe) = true$ si et seulememt si $a \in member(oe)$
- si un agent a joue un rôle dans l'OE :
  $oe \in \mathcal{OE}, a \in member(oe), is\_player(a, oe) = true$ si et seulement si $prole(a, oe) \in \mathcal{R}ole$
- si un agent a joue un rôle r dans l'instance de groupe g de l'OE :
  $oe \in \mathcal{OE}, a \in member(oe), g \in groupe(oe), r \in \mathcal{R}_{spec(g)}, is\_player\_role(a, r, g, oe) = true$ si et seulement si $irole_r(g) \in prole(a, oe) \wedge irole_r(g) \in role(g)$
- si deux rôles sont compatibles : $r \in \mathcal{R}_{ss}, r' \in \mathcal{R}_{ss}, is\_compatible(r, r') = true$ si et seulement si $\exists\, l \in \mathcal{L}ink : l = link(r, r', comp)$

Les agents membres d'une Organisation peuvent s'engager sur plusieurs missions au sein de cette Organisation. Nous définissons pour cela plusieurs fonctions. Les missions devant être accomplies au sein d'instances de schémas, nous avons besoin de la fonction *scheme* permettant de connaître toutes les instances de scheme d'une OE, de la fonction *igoal* retournant toutes les instances des buts d'une instance du schéma et de la fonction *pgoal* retournant toutes les instances des buts d'une instance de mission d'une instance de schéma.

$$scheme : \mathcal{OE} \rightarrow \mathcal{S}cheme$$
$$igoal : \mathcal{S}cheme \rightarrow \mathbb{P}(\mathcal{G}oal)$$
$$pgoal : \mathcal{S}cheme \times \mathcal{M}ission \rightarrow \mathcal{G}oal$$

Concernant les missions, la fonction *mission* permet de connaître toutes les instances de missions appartenant à une instance de schéma ....

$$mission : \mathcal{S}cheme \rightarrow \mathcal{M}ission$$
$$imission_m : \mathcal{S}cheme \rightarrow \mathcal{M}ission$$
$$pmission : \mathcal{A} \times \mathcal{OE} \rightarrow \mathcal{M}ission$$

Sur base de ces fonctions, nous définissons un ensemble de prédicats permettant de savoir :
- si un agent a est membre de l'OE :
  $a \in \mathcal{A}, oe \in \mathcal{OE}, is\_member(a, oe) = true$ si et seulememt si $a \in member(oe)$
- si un agent a est engagé sur une mission de l'OE :
  $oe \in \mathcal{OE}, a \in member(oe), is\_committed(a, oe) = true$ si et seulement si $pmission(a, oe) \in \mathcal{M}ission$
- si un agent a est engagé sur une mission m d'une instance de schéma s de l'OE :
  $oe \in \mathcal{OE}, a \in member(oe), sscheme(oe), m \in \mathcal{M}_{spec(s)}, is\_committed\_mission(a, m, s, oe) = true$ si et seulement si $imission_m(s) \in pmission(a, oe) \wedge imission_m(s) \in mission(s)$

Les actions qu'un agent peut exécuter sur l'OE concernent aussi bien sa structure et son fonctionnement que l'instance d'organisation dans sa globalité ainsi que les Contextes et les Normes actifs. Nous les détaillons dans la section suivante.

7.2 Actions sur l'Organisation

Afin que les agents puissent agir au sein de l'OE, ils doivent passer par leur représentation au sein même de l'Organisation. Ainsi, et comme illustré sur la FIG. 7.1, les agents voulant par exemple adopter un rôle au sein de la Structure de l'Organisation doivent appeler la méthode *adoptRole* de

leur "Member"[1]. Ci-après, nous détaillons les différentes actions permettant tout d'abord de créer et de détruire ces "Member", puis les actions concernant la Structure et le Fonctionnement de l'Organisation et enfin celles concernant la gestion de l'activité des Contextes et des Normes. Les agents qui vont agir au sein de l'Organisation peuvent aussi bien être des agents du domaine (AAI), mais aussi être, comme nous le verrons dans ce chapitre, des agents composant le système d'arbitrage (IAS). Les actions que nous allons voir peuvent ainsi être exécutées dans un but d'arbitrage et de supervision ou pour que l'Organisation atteigne son objectif. A partir de maintenant et contrairement au chapitre précédent, nous nommerons groupes, rôles, schémas, etc, les instances de groupe, les instances de rôles, les instances de schémas, etc.

Dans ce qui suit, nous allons décrire un ensemble d'actions selon cinq critères :

- *Description de l'action* : décrit en langage naturel ce que doit faire l'action.
- *Arguments* : ce sont les entités passées en paramètre qui seront utilisées afin d'exécuter l'action.
- *Pré-conditions* : ce sont les conditions requises afin que l'action puisse s'exécuter.
- *Tests institutionnels* : ce sont des conditions requises pour l'exécution de l'action mais non bloquantes contrairement aux pré-conditions. Ces tests sont effectués par le système d'arbitrage et peuvent déclencher une détection de violation si ils ne sont pas satisfaits.
- *Pré-conditions* : ce sont les conditions devant être vérifiées une fois l'action exécutée.

7.2.1 Actions organisationnelles

Les actions organisationnelles concernent l'Organisation dans son ensemble. Elle s'appliquent donc à une Entité Organisationnelle *oe* que nous omettons de mettre dans les arguments pour des raisons de facilité de lecture. Les actions disponibles permettent de gérer l'entrée et la sortie des agents de l'Organisation. Ainsi, l'action *createMember* permet d'ajouter l'agent passé en paramètre à la liste des membres de cette OE[2].

Action : *createMember*	
Description de l'action	Entrée d'un agent dans l'Organisation
Arguments	Agent $a \in \mathcal{A}$
Pré-conditions	
Tests institutionnels	
Post-conditions	L'agent a fait partie des membres de l'OE $is\_member(a, oe)$

Pour sortir de l'Organisation, il faut que l'agent ne soit plus engagé sur des missions ou ne soit pas encore en train de jouer des rôles. L'action *deleteMember* peut alors être exécutée et effacer l'agent de la liste des membres.

[1]Un agent ne sait pas obligatoirement comment mettre en place une action d'adoption de rôle. Nous considérons donc l'exécution d'une action comme étant l'envoie d'un message d'un agent à son "Member" et l'exécution réelle de la méthode correspondante par ce dernier.

[2]Nous notons que ce genre d'action ne peut être exécutée que par des agents supervisant l'Organisation, à savoir ceux de l'IAS

Action : *deleteMember*	
Description de l'action	Sortie d'un agent de l'organisation
Arguments	Agent $a \in \mathcal{A}$
Pré-conditions	L'agent a fait partie des membres de l'OE $is\_member(a, oe)$
Tests institutionnels	Aucune mission ne concerne l'agent a $\neg is\_committed(a, oe)$ Aucun rôle ne concerne l'agent a $\neg is\_player(a, oe)$
Post-conditions	L'agent a ne fait plus partie des membres de l'OE $\neg is\_member(a, oe)$

La conséquence de cette action est que l'agent ne soit plus représenté au sein de l'Organisation et de ce fait ne peut plus agir en son sein. Il peut cependant toujours faire partie de la société.

7.2.2 Actions structurelles

Les actions que nous allons décrire ici concernent la Structure de l'Organisation. Pour chaque action, nous donnons les arguments éventuels, les pré-conditions au déclenchement de l'action ainsi que les post-conditions. Nous décrivons également les tests institutionnels qui seront faits par le système d'arbitrage afin de vérifier si la spécification n'est pas violée lors de son instanciation.

L'action *createGroup* permet à un agent de créer un groupe et se traduit par l'instanciation d'une spécification de groupe de la SS passée en paramètre. La spécification de groupe appartient obligatoirement à un ensemble de spécifications de sous-groupes et de ce fait le sous-groupe ne peut pas être créé si son groupe père n'a pas été créé auparavant. Les pré-conditions requièrent également que le nombre d'instances de groupe ne dépasse pas la cardinalité de groupe maximum.

Action : *createGroup*	
Description de l'action	Création d'une instance de groupe
Arguments	Spécification de groupe $sgr \in \mathcal{G}r_{ss}$ Instance de groupe $igr \in \mathcal{G}roup$
Pré-conditions	Si igr est instanciée et que sgr est un sous-groupe de igr, alors sgr ne doit pas être déjà instanciée et appartenir à igr $igr \in group(oe) \wedge sgr \in \mathcal{SG}r_{spec(igr)} \Rightarrow instantiate(sgr) \notin igroup_{spec(igr)}(sgr, igr)$ Si sgr n'est pas un sous-groupe de igr, alors sgr ne doit pas être déjà instanciée $sgr \notin \mathcal{SG}r_{spec(igr)} \Rightarrow instantiate(sgr) \notin group(oe)$
Tests institutionnels	Si sgr est le sous-groupe de igr, le nombre d'instances de groupe sgr est inférieur à sa cardinalité maximale de groupe $card(igroup_{spec(igr)}(sgr, igr)) < ng_{spec(igr)}(sgr)$
Post-conditions	Si igr existe, sgr est instanciée en sous-groupe de igr $igr \in groupe(oe) \Rightarrow instantiate(sgr) \in igroup_{spec(igr)}(sgr, igr)$ Si igr n'existe pas, sgr est instanciée comme groupe racine de l'OE $igr \notin groupe(oe) \Rightarrow instantiate(sgr) \in group(oe)$

De même qu'il est possible de créer un groupe, l'action *removeGroup* permet de détruire l'instance de groupe passée en paramètre. Les pré-conditions requièrent une instance de groupe n'étant composée ni de sous-groupes ni de rôles.

Action : *removeGroup*	
Description de l'action	Destruction d'une instance de groupe
Arguments	Instance de groupe $group \in \mathcal{G}roup$
Pré-conditions	L'instance de groupe ne possède pas de sous-groupes $\forall\, gr' \in \mathcal{SG}r_{spec(group)}, igroup_{spec(group)}(gr', group) = \emptyset$ L'instance de groupe ne possède pas d'instances de rôles $role(group) = \emptyset$
Tests institutionnels	
Post-conditions	L'instance de groupe $group$ n'appartient plus à l'OE $group \notin group(oe)$

En plus des groupes, il est possible d'instancier des rôles grâce à l'action *adoptRole*. Pour cela, la spécification du rôle, l'instance du groupe auquel le rôle va appartenir et l'agent qui va jouer le rôle sont les paramètres de l'action. Les pré-conditions vérifient que le rôle existe bien et n'est pas abstrait tandis que les tests institutionnels vérifient que l'agent ne va pas violer une spécification en agissant ainsi sur l'Organisation.

Action : *adoptRole*	
Description de l'action	Adoption d'un rôle par un agent dans une instance de groupe
Arguments	Spécification de rôle $r \in \mathcal{R}_{ss}$ Instance de groupe $group \in \mathcal{G}roup$ Agent $a \in \mathcal{A}$
Pré-conditions	L'agent a est un membre de l'OE $is\_member(a, oe)$ Le rôle n'est pas abstrait $r \notin \mathcal{R}_{abs_{ss}}$ Le rôle n'est pas déjà joué par ce membre dans ce groupe $\neg is\_player\_role(a, r, group, oe)$
Tests institutionnels	La cardinalité de rôle pour le rôle r de la spécification du groupe $group$ est supérieur au nombre d'instances du rôle r $card(irole_r(group)) < nr_{spec(group)}(r)$ Le rôle est compatible avec les rôles déjà joués par le membre $\forall\, role' \in prole(a, oe), is\_compatible(r, spec(role'))$
Post-conditions	L'agent a joue le rôle r dans l'instance de groupe $group$ de l'OE $is\_player\_role(a, r, group, oe)$ Les instances de liens intra-groupe concernant le rôle à adopter d'une part et un rôle instancié au sein de la même instance de groupe d'autre part font partie de l'ensemble des instances de lien intra-groupe $(\forall\, r' \in \mathcal{R}_{ss}, \forall\, l \in \mathcal{L}^{intra}_{spec(group)} : l = link(r, r', t)) \wedge (\forall\, role' \in \mathcal{R}ole, \forall\, a' \in \mathcal{A} : is\_player\_role(a', spec(role'), group, oe)) \Rightarrow instantiate(l, role', irole_r(a, group)) \in \mathcal{L}ink^{intra}_{group}$ Les instances de liens inter-groupe concernant le rôle à adopter d'une part et un rôle instancié au sein des instances de la même spécification de groupe d'autre part font partie de l'ensemble des instances de lien inter-groupe $(\forall\, r' \in \mathcal{R}_{ss}, \forall\, l \in \mathcal{L}^{intra}_{spec(group)} : l = link(r, r', t)) \wedge (\forall\, group' \in \mathcal{G}roup : group' = instantiate(spec(group)) \wedge group \neq group') \wedge (\forall\, role' \in \mathcal{R}ole, \forall\, a' \in \mathcal{A} : is\_player\_role(a', spec(role'), group', oe)) \Rightarrow instantiate(l, role', irole_r(a, group)) \in \mathcal{L}ink^{inter}_{group}$

L'action *giveUpRole* permet la destruction d'une instance de rôle passée en paramètre. Pour que cela ne rende pas l'Organisation incohérente, il faut que l'agent ne joue pas un rôle étant autorisé ou obligé à être engagé et à accomplir une mission. Afin de faire des tests sur les missions et les normes les concernant, nous définissons les fonctions suivantes permettant de connaître le porteur d'une norme ainsi que son opérateur déontique et la mission concernée :

$bearer : \mathcal{NS} \rightarrow \mathcal{SE}$
$op : \mathcal{NS} \rightarrow \{obl, per, for\}$
$m : \mathcal{NS} \rightarrow \mathcal{M}$

Action : *giveUpRole*	
Description de l'action	Abandon d'un rôle par un agent dans une instance de groupe
Arguments	Instance de rôle $role \in \mathcal{R}ole$ Instance de groupe $group \in \mathcal{G}roup$ Agent $a \in \mathcal{A}$
Pré-conditions	L'agent a est un membre de l'OE $is\_member(a, oe)$ Le rôle *role* est joué par l'agent a dans le groupe *group* de l'OE $is\_player\_role(a, spec(role), group, oe)$
Tests institutionnels	Pour toutes les obligations concernant les missions non accomplies sur lesquelles l'agent est engagé, le rôle de la norme n'est pas le rôle que le membre veut abandonner $\forall mission \in \mathcal{M}ission, \forall scheme \in \mathcal{S}cheme : is\_committed\_mission(a, spec(mission), scheme, oe), \forall norm \in \mathcal{AN}orm : (mission(norm) = spec(mission)) \wedge (op(norm) = obl) \Rightarrow bearer(norm) \neq spec(role)$
Post-conditions	L'agent a ne joue plus le rôle *role* au sein du groupe *group* de l'OE $\neg is\_player\_role(a, spec(role), group, oe)$ Les instances de liens concernant le rôle à abandonner d'une part et un rôle instancié au sein d'une quelconque instance de groupe d'autre part ne font pas partie de l'ensemble des instances de lien $(\forall role' \in \mathcal{R}ole, \forall link \in \mathcal{L}ink_{group} : link = link(role, role', t)) \Rightarrow link \notin \mathcal{L}ink_{group}$

Nous remarquons qu'il n'y a aucune action structurelle permettant la manipulation des liens entre instances de rôles.

7.2.3 Actions fonctionnelles

La première action fonctionnelle à exécuter au sein de l'Organisation est la création d'un schéma d'exécution afin d'instancier le fonctionnement des agents de la société. Cela est possible avec l'action *createScheme* prenant en paramètre sa spécification.

Action : *createScheme*	
Description de l'action	Création et démarrage d'une instance de schéma
Arguments	Spécification de schéma $s \in \mathcal{S}_{fs}$
Pré-conditions	
Tests institutionnels	
Post-conditions	L'instance de schéma s fait partie de l'ensemble des instances de schéma de l'OE $instantiate(s) \in scheme(oe)$ Les instances de buts composent l'instance de schéma $\forall\, g \in \mathcal{G}o_s \Rightarrow instantiate(g) \in igoal(instantiate(s))$

La création d'un schéma implique la création des instances de buts correspondant à l'exécution du schéma. Les instances de but ont un état indiquant si le but est atteint, non-atteint ou impossible à atteindre. Une fois que le but racine du schéma est atteint ou déclaré comme impossible à atteindre, l'action *finishScheme* permet de supprimer l'instance de schéma. Les pré-conditions requièrent donc que le schéma ne soit pas déjà terminé et qu'aucun agent ne soit engagé sur une mission du schéma afin que des agents ne se retrouvent pas engagés sur des missions dont les buts n'existent plus.

Action : *finishScheme*	
Description de l'action	Arrêt et suppression d'une instance de schéma
Arguments	Instance de schéma $scheme \in \mathcal{S}cheme$
Tests institutionnels	
Pré-conditions	Aucun agent n'est engagé sur une mission de l'instance de schéma *scheme* $mission(scheme) = \emptyset$
Post-conditions	L'instance de schéma ne fait pas partie de l'ensemble des instances de schéma de l'OE $scheme \notin scheme(oe)$ Les instances de buts de l'instance de schéma *scheme* ne font pas partie de l'ensemble des buts de l'OE $igoal(scheme) = \emptyset$

Il est cependant possible de stopper le schéma au cours de son exécution alors que des agents sont toujours engagés sur des missions afin d'atteindre des buts. Il suffit pour cela que l'agent voulant arrêter l'exécution d'un schéma en cours, exécute l'action *abortScheme* de son "Member".

Action : *abortScheme*	
Description de l'action	Abandon d'une instance de schéma en cours
Arguments	Instance de schéma $scheme \in \mathcal{S}cheme$
Pré-conditions	
Tests institutionnels	
Post-conditions	Aucun agent n'est engagé sur une mission de l'instance de schéma *scheme* $mission(scheme) = \emptyset$ L'instance de schéma ne fait pas partie de l'ensemble des instances de schéma de l'OE $scheme \notin scheme(oe)$ Les instances de buts de l'instance de schéma *scheme* ne font pas partie de l'ensemble des buts de l'OE $igoal(scheme) = \emptyset$

Le fait d'abandonner l'exécution d'une instance de schéma désengage les membres des agents de leurs missions. Pour qu'un agent s'engage sur une mission, il faut qu'il exécute l'action *commitToMission*. Cette création d'instance de mission prend en paramètre l'agent et la spécification de mission sur laquelle il veut s'engager.

Action : *commitToMission*	
Description de l'action	Engagement d'un agent sur une mission
Arguments	Agent $a \in \mathcal{A}$ Spécification de mission $m \in \mathcal{M}_{fs}$ Instance de schéma $scheme \in \mathcal{S}cheme$
Pré-conditions	L'agent a est un membre de l'OE $is\_member(a, oe)$ L'agent n'est pas déjà engagé sur la mission $\neg is\_committed\_mission(a, m, scheme, oe)$
Tests institutionnels	Il existe une norme obligeant ou autorisant l'accomplissement de la spécification de mission m par une spécification de rôle instanciée par le membre de l'agent $\exists\, norm \in \mathcal{N}orm : (m(norm) = m) \wedge (op(norm) = obl \vee op(norm) = per\}) \wedge (\forall\, group \in \mathcal{G}roup, \exists\, role \in \mathcal{R}ole : role = is\_player\_role(a, bearer(norm), group, oe))$ Le nombre d'agent engagé sur la mission m est inférieur à la cardinalité maximale de la mission m $card(imission_m(scheme) < nm_{spec(scheme)}(m)$
Post-conditions	L'agent a est engagé sur la mission m faisant partie du schéma *scheme* de l'OE $is\_committed\_mission(a, m, scheme, oe)$

Une fois que toutes les instances des buts ont été atteintes par un agent ou que l'exécution de l'instance de schéma est abandonnée, les agents doivent se désengager des missions sur lesquelles ils étaient. Pour cela, l'action *uncommitToMission* doit être exécutée en passant l'instance de schéma en paramètre.

Action : *uncommitToMission*	
Description de l'action	Désengagement d'un agent sur une mission
Arguments	Instance de mission $mission \in \mathcal{M}ission$ Instance de schéma $scheme \in \mathcal{S}cheme$ Agent $a \in \mathcal{A}$
Pré-conditions	L'agent a est un membre de l'OE $is\_member(a, oe)$ L'agent est engagé sur la mission $is\_committed\_mission(a, spec(mission), scheme, oe)$
Tests institutionnels	La mission est accomplie $status(mission) = accomplished$
Post-conditions	L'agent a n'est plus engagé sur mission $mission$ au sein du schéma $scheme$ de l'OE $\neg is\_committed\_mission(a, spec(mission), scheme, oe)$

En temps normal, l'agent ne doit pas se désengager d'une mission avant d'avoir satisfait toutes les instances de buts de la mission. Pour ce faire, il doit exécuter l'action *setGoalSatisfied* en passant en paramètre l'instance de but concernée. Il faut bien entendu que l'agent soit engagé sur la mission pour que l'action puisse être exécutée.

Action : *setGoalSatisfied*	
Description de l'action	Instance de but atteint par un agent
Arguments	Instance de but $goal \in \mathcal{G}oal$ Agent $a \in \mathcal{A}$
Pré-conditions	L'agent a est un membre de l'OE $is\_member(a, oe)$ L'agent est engagé sur une mission contenant ce but $\forall scheme \in \mathcal{S}cheme, \exists mission \in \mathcal{M}ission : is\_committed\_mission(a, spec(mission), scheme, oe) \wedge goal \in pgoal(mission, scheme)$ La satisfaction du but par cet agent ne termine pas une mission provoquant une violation d'une norme $(status(goal) = satisfied) \Rightarrow \nexists mission : goal \in pgoal(mission, scheme), status(mission) = accomplished \wedge m(norme) = mission \rightarrow status(norme) = violated$
Post-conditions	L'état de l'instance de but est "satisfait" par agent a $status(goal) = satisfied$

Pour déclarer le but impossible à atteindre, l'action *setGoalImpossible* doit être exécutée.

Action : *setGoalImpossible*	
Description de l'action	Instance de but déclarée impossible à atteindre
Arguments	Instance de but $goal \in \mathcal{G}oal$ Agent $a \in \mathcal{A}$
Pré-conditions	L'agent a est un membre de l'OE $is\_member(a, oe)$
Tests institutionnels	
Post-conditions	L'état de l'instance de but est déclaré "impossible" par agent a $status(goal) = impossible$

Si au cours de l'évolution de l'Organisation, un but impossible redevient possible, l'action *setGoalPossible* doit être exécutée.

Action : *setGoalPossible*	
Description de l'action	Instance de but déclarée possible à atteindre
Arguments	Instance de but $goal \in \mathcal{G}oal$ Agent $a \in \mathcal{A}$
Pré-conditions	L'agent a est un membre de l'OE $is\_member(a, oe)$ L'état de l'instance de but est "impossible" $status(goal) = impossible$
Tests institutionnels	
Post-conditions	L'état de l'instance de but est déclaré "possible" par agent a $status(goal) = possible$

7.2.4 Actions contextuelles

Au contraire de l'action normative et même si les Contextes Actifs ne sont pas une instanciation des contextes, la seule action disponible permet de mettre à jour les Contextes Actifs en fonction de l'événement passé en paramètre. En effet l'action *changeContext* permet de changer les contextes actifs si l'événement passé en paramètre permet de déclencher des transitions définies dans la CS.

Action : *changeContext*	
Description de l'action	Met à jour les contextes actifs
Arguments	Événement $e \in \mathcal{E}$ Agent $a \in \mathcal{A}$
Pré-conditions	L'agent a est un membre de l'OE $is\_member(a, oe)$
Tests institutionnels	
Post-conditions	Pour tous les contextes actifs dont le résultat de la fonction de transition par l'événement e donne un contexte cible, le contexte actif est enlevé de l'ensemble des contextes actifs et le contexte cible y est ajouté par l'agent a $\forall\, contexte_{source} \in \mathcal{AC}ontext : trans(contexte_{source}, e) \neq \emptyset \Rightarrow \mathcal{AC}ontext = \mathcal{AC}ontext \cup \{trans(contexte_{source}, e)\} - \{contexte_{source}\}$

7.2.5 Actions normatives

Les actions normatives permettent d'agir sur les instances de normes et notamment sur leur statut. Ce sont les agents faisant partie de l'IAS qui peuvent exécuter ces actions de leur "Member". Lorsqu'une norme est instanciée, cela veut dire que le contexte pour lequel elle est définie devient actif. De ce fait, une instance de norme commence son cycle de vie en étant active. Ensuite, pour qu'elle puisse influencer le comportement des agents, il faut qu'elle devienne valide. Pour cela, l'action *setNormValid* doit être exécutée.

Action : *setNormValid*	
Description de l'action	Instance de norme est valide
Arguments	Instance de norme $norm \in \mathcal{AN}orm$ Agent $a \in \mathcal{A}$
Pré-conditions	L'agent a est un membre de l'OE $is\_member(a, oe)$ La norme est active $status(norm) =$ "$active$" Les conditions de validation de la norme sont vraies $cond(norm) = true$
Tests institutionnels	
Post-conditions	La norme est déclarée valide par agent a $status(norm) =$ "$valid$"

Une fois que la norme est déclarée valide et afin de suivre son cycle de vie, une norme est soit respectée, soit violée. Pour cela, les actions *setNotmRespected* et *setNormViolated* doivent être exécutées.

Action : *setNormRespected*	
Description de l'action	Instance de norme est respectée
Arguments	Instance de norme $norm \in \mathcal{AN}orm$ Agent $a \in \mathcal{A}$
Pré-conditions	L'agent a est un membre de l'OE $is\_member(a, oe)$ La norme est valide $status(norm) =$ "$valid$"
Tests institutionnels	
Post-conditions	La norme est déclarée respectée par agent a $status(norm) =$ "$respected$"

Action : *setNormViolated*	
Description de l'action	Instance de norme est violée
Arguments	Instance de norme $norm \in \mathcal{AN}orm$ Agent $a \in \mathcal{A}$
Pré-conditions	L'agent a est un membre de l'OE $is\_member(a, oe)$ La norme est valide $status(norm) =$ "$valid$"
Tests institutionnels	
Post-conditions	La norme est déclarée violée par agent a $status(norm) =$ "$violated$"

Nous remarquons que les actions concernant les normes ne possèdent pas de tests institutionnels. Cela s'explique en deux temps : (1) tout d'abord, ces actions étant exécutées par les agents institutionnels de l'IAS, elles respectent tout le temps l'OS et ne font donc pas l'objet de tests institutionnels ; (2) par contre, ces actions sont la conséquence des tests institutionnels pour les actions *commitToMission* et *setGoalSatisfied*. Si le résultat des tests institutionnels de ces deux actions est que les normes sont violées, alors la norme concernée doit être mise à jour. Comme nous l'avons vu dans la Section 6.5.3 du Chapitre 6, cette détection de la violation ne se fait que via la supervision de la mission concernée par la norme. Nous verrons dans le chapitre suivant les moyens mis en œuvre

par les agents pour détecter tous les types de violation.

7.3 Agents de Supervision d'Institution Multi-Agents

L'OS à partir de laquelle l'Organisation est instanciée définit un ensemble de contraintes restreignant le champ d'action des agents. Pour que les agents respectent la spécification, il faut que toutes les pré-conditions associées aux actions soient vraies et que les tests institutionnels ne provoquent pas de violation de l'OS. C'est cela que nous considérons comme une activité d'arbitrage devant être mise en place par l'IAS. Notre solution est de fournir un *middleware* d'agents de supervision entre la demande des agents d'agir sur l'Organisation et leur "Member" mettant en place les actions demandées comme illustré sur la FIG. 7.2. Son nom est $\mathcal{S}$YNAI et son rôle est donc de superviser l'action des agents sur l'OE, c'est-à-dire que les agents de supervision feront les tests institutionnels des actions que nous venons de voir. Ses caractéristiques principales sont les suivantes :

- $\mathcal{S}$YNAI est constitué d'un ensemble d'Agents fournis par MA$\mathcal{B}_{\text{ELI}}$.
- Les agents de $\mathcal{S}$YNAI sont des agents cognitifs capables de raisonner afin de résoudre des conflits ou des situations incohérentes et de sanctionner les agents du domaine. Cette capacité ne serait pas possible avec un ensemble de services à la place des Agents de Supervision.
- Les agents de $\mathcal{S}$YNAI constituent un système d'arbitrage prenant en compte les spécifications d'une modélisation avec $\mathcal{M}$OISE^{Inst}. Leur comportement est non seulement dirigé par une OS $\mathcal{M}$OISE^{Inst} dont la FS définit des buts de supervision.
- Chaque instance d'Organisation définie avec $\mathcal{M}$OISE^{Inst} est supervisée par un système d'arbitrage unique et chaque système d'arbitrage supervise une seule instance d'Organisation. Lorsqu'une OS est instanciée, une OE et une instance de $\mathcal{S}$YNAI (un ensemble unique d'agents de supervision de l'OE) sont créées.
- Chaque instance d'une spécification de l'Organisation est supervisée par un agent spécifique de $\mathcal{S}$YNAI. C'est justement la FS de leur propre OS ainsi que les protocoles, que chaque agent doit suivre, qui permet d'associer un agent à la supervision d'une spécification particulière.
- Chaque agent du domaine venant faire partie de l'Organisation afin d'y jouer un rôle et accomplir des missions ne peut interagir qu'avec un agent d'interface le représentant au sein d'une instance d'Organisation. Un agent du domaine devra communiquer avec un *wrapper* afin de pouvoir exécuter les actions de son "Member".

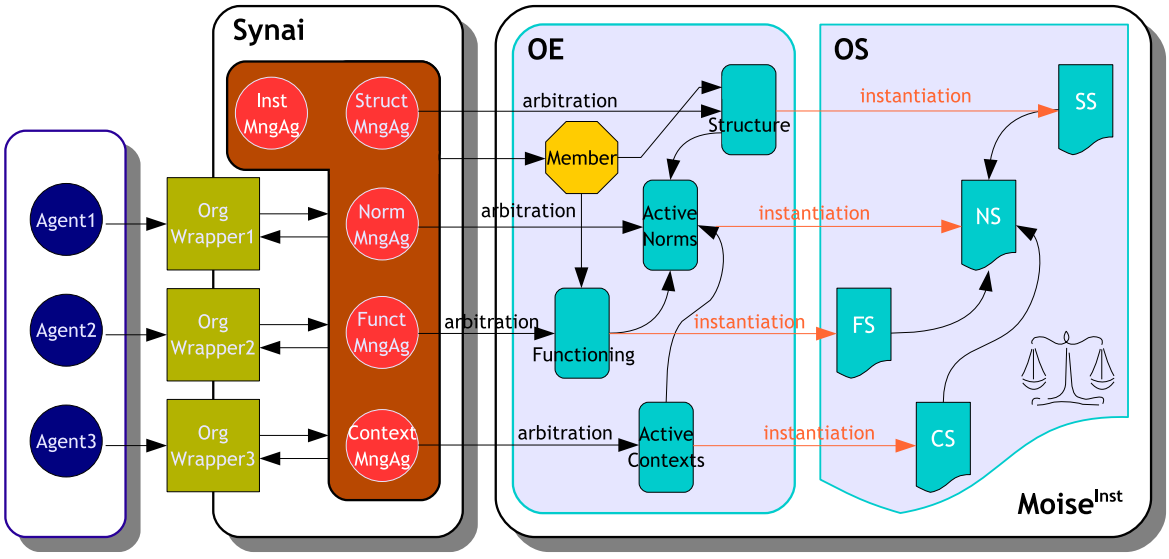


FIG. 7.2 – Synthèse sur la position de $\mathcal{S}$YNAI au sein de MA$\mathcal{B}_{\text{ELI}}$

OrgWrapperAg	Médiateur entre les agents du domaine et les agents de $\mathcal{S}$YNAI
StructManagerAg	Manager gérant l'adoption de rôles et l'accès aux groupes
FunctManagerAg	Manager gérant l'engagement sur les missions et le changement d'état des buts de la mission
ContextManagerAg	Manager gérant les changements de contexte en fonction des événements déclencheurs de transition
NormManagerAg	Manager gérant le respect de norme en fonction des rôles et des contextes
InstManagerAg	Manager gérant l'accès à l'organisation et la mise en place des sanctions.

TAB. 7.1 – Identification des agents de $\mathcal{S}$YNAI

Les Agents de Supervision de $\mathcal{S}$YNAI sont listés dans le TAB. 7.1. Plus précisément, si nous regardons la FIG. 7.2 de droite à gauche, nous voyons qu'un agent du domaine peut communiquer avec l'agent *InstManagerAg* de l'organisation qu'il veut rejoindre et, une fois que celui-ci sera créé, avec son agent *OrgWrapperAg*. Au sein de $\mathcal{S}$YNAI, il existe un agent *OrgWrapperAg* par agent du domaine et par institution. Si un même agent de domaine participe à deux institutions, il aura deux agents institutionnels comme interlocuteurs pour interagir avec les deux organisations. L'agent du domaine communique par envoi de message avec son *OrgWrapperAg* qui lui même communique avec l'ensemble des autres agents institutionnels. Tous les superviseurs peuvent communiquer entre eux. Sur le schéma, les liens de communication ne sont pas représentés afin de rendre la figure plus lisible. Les agents *InstManagerAg*, *StructManagerAg*, *FunctManagerAg*, *NormManagerAg* et *ContextManagerAg* sont capables de communiquer les uns avec les autres. Ces agents reçoivent donc de la part des agents du domaine reliés par les agents *OrgWrapperAg* des demandes d'actions au sein de l'Organisation (cf. section précédente pour la description des actions). Ces agents traitent les requêtes en acceptant ou non (selon la configuration de l'arbitrage) pour ensuite agir sur les instances de spécification de l'organisation. Nous reviendrons sur ce point plus tard mais nous pouvons d'ores et déjà préciser que l'*InstManagerAg* est dédié entre autre à la mise en application des normes (*enforcement*) et à l'application de sanctions tandis que les autres agents sont des agents de détection.

Si nous voulons que ces superviseurs interagissent avec les agents du domaine sur un même terrain d'entente, il faut qu'ils aient la même représentation et interprétation de l'Organisation. C'est pour cela que les Agents de Supervision, au même titre que les agents du domaine, doivent être organisés à l'aide de $\mathcal{M}$OISE^{Inst}. Cependant, avant de donner le modèle d'organisation de $\mathcal{S}$YNAI, voyons dans les sections suivantes une brève description de chaque Agent de Supervision.

7.3.1 OrgWrapperAg

C'est l'agent qui sert d'interface entre l'agent du domaine et les agents institutionnels. Il en existe un par agent du domaine et par Institution. Il reçoit les messages des agents du domaine et en fonction de la requête et de la composante de l'organisation que cela concerne, il fait suivre le message au *Superviseur* correspondant. Les messages sont donc triés en fonction de l'événement transporté (que nous allons étudier dans la Section 7.5). Cela implique qu'il connaisse les différents protocoles de communication à utiliser avec les agents institutionnels. Ainsi, pour une demande de l'agent du domaine, il mettra en place un dialogue avec un ou plusieurs agents institutionnels afin de retourner une réponse à son émetteur. Nous considérons que l'agent *OrgWrapperAg* fournit l'ensemble des méthodes de base pour communiquer avec les autres agents comme c'est le cas dans TeamCore [94] avec le TeamcoreWrapper.

Comme nous verrons dans la section 7.4, cet agent, comme tous les autres agents de $\mathcal{S}$YNAI, a son comportement spécifié avec une organisation de type $\mathcal{M}$OISE^{Inst}. Il est donc capable d'interpréter l'OS, de jouer un rôle, de s'engager sur des missions et de satisfaire des buts. Les agents doivent exécuter les actions du "Member" qui leur est associé afin d'agir au sein de l'organisation. L'agent

OrgWrapperAg, en tant que représentant de l'agent du domaine, va faire en sorte que son "Member" adopte des rôles, s'engage sur des missions, et ait la possibilité de violer des contraintes au nom de cet agent.

7.3.2 StructManagerAg

Cet agent gère l'instance de la Spécification Structurelle en créant et en détruisant des instances de groupes et de rôles. Pour ce faire, il doit mettre en place à chaque fois un mécanisme de vérification pour faire en sorte que la Structure de l'Organisation respecte ce qui a été défini dans la SS. Par exemple, si un agent du domaine veut jouer un rôle au sein d'une instance de groupe possédant déjà le nombre maximum de joueurs autorisés, alors l'agent *StructManagerAg* détecte que la SS vient d'être violée. Cet agent est utilisé pour mettre en place les actions *createGroup*, *removeGroup*, *adoptRole* et *giveUpRole* abordées dans la Section 7.2.

Si l'agent *StructManagerAg* détecte que les pré-conditions des différentes actions qu'il peut mettre en place ne sont pas respectées, il envoie alors un message à l'agent *InstManagerAg* pour l'avertir d'une violation de la SS.

7.3.3 FunctManagerAg

Cet agent gère les engagements des agents de l'Organisation sur des missions définies dans la Spécification Fonctionnelle. Suivant les rôles qu'il joue et les contextes dans lesquels il se trouve, l'agent *FunctManagerAg* permettra ou non aux agents de s'engager sur des missions en mettant en œuvre les actions *commitToMission* et *uncommitToMission* et en agissant sur les buts via la mise en œuvre des actions *setGoalSatisfied* et *setGoalImpossible* (cf. Section 7.2). L'agent *FunctManagerAg* s'occupe également de créer des instances de schémas d'exécution (action *createScheme*). Quand le but racine du schéma est atteint, l'instance du schéma est terminée et est détruite (action *finishScheme* ou *abortScheme*).

De la même façon que l'agent *StructManagerAg*, cet agent doit vérifier que toutes les conditions soient satisfaites pour que les agents faisant la demande d'action fonctionnelle sur l'Organisation respectent bien les contraintes issues de la FS et de la NS. L'agent doit pour cela exécuter un ensemble de tests permettant de déterminer si les pré-conditions des actions demandées sont satisfaites. Si ce n'est pas le cas il doit prévenir l'agent *InstManagerAg* d'une violation de la FS et agir comme la stratégie d'arbitrage le prévoit (nous aborderons les stratégies d'arbitrage dans la section 7.4).

Les agents *StructManagerAg* et *FunctManagerAg* permettent aux agents du domaine participant à l'Organisation de jouer des rôles et d'atteindre des buts sous leur supervision. Ils agissent donc, indirectement certes, sur les instances des spécifications structurelles et fonctionnelles. Par contre, les agents du domaine n'agissent pas sur les instances des spécifications contextuelles et normatives mais subissent leurs conséquences. Les contextes et les normes actifs vont déterminer à un moment donné les droits et les devoirs de chaque agent en fonction de leurs rôles joués et suivant les missions qu'ils veulent accomplir. Le *ContextManagerAg* informe des contextes courant et le *NormManagerAg* détecte les violations de normes.

7.3.4 ContextManagerAg

L'agent *ContextManagerAg* est capable de mettre en place l'action *changeContext*. Pour cela, il va prendre en compte tous les messages d'information envoyés par les agents et contenant les événements définis dans l'Organisation qui sont diffusés à tout le monde. Pour chaque message que *ContextManagerAg* recevra, il déclenchera l'action *changeContext* de son "Member". Ainsi, comme expliqué en Section 7.2, si une transition basée sur cet événement existe, il appliquera le changement de contexte. Il mettra ainsi à jour la liste des Contextes Actifs et avertira l'Organisation du changement en envoyant un message à tous les agents.

Les événements pris en compte par le *ContextManagerAg* dépendent de la CS de l'application dans laquelle il sera utilisé.

7.3.5 NormManagerAg

C'est lorsque les agents du domaine vont agir au sein de la société qu'ils vont devoir respecter les normes. Pour agir, il faut qu'ils s'engagent sur des missions et atteignent des buts. Ils doivent dans un premier temps interagir avec l'agent *FunctManagerAg*. C'est ce dernier qui va ensuite envoyer des demandes à l'agent *NormManagerAg* pour vérifier que les actions sont autorisées par les normes actives. Nous verrons plus en détail l'enchaînement des messages dans la section 7.5. L'événement de demande de vérification de norme est *checkNorms* et lorsque l'agent *NormManagerAg* le reçoit dans un message il met en œuvre l'action du même nom (cf. Section 7.2).

Cet événement sera envoyé de l'agent *FunctManagerAg* à l'agent *NormManagerAg* quand un autre agent voudra s'engager sur une mission d'une instance de schéma ou quand un agent du domaine voudra modifier l'état d'un but appartenant à une mission d'une instance de schéma. Nous verrons cela en détail dans la section consacrée aux protocoles de supervision. Dans les deux cas, l'Agent Institutionnel vérifiera s'il existe une instance de rôle autorisant l'agent à exécuter la mission donc à s'engager dessus ou à atteindre des buts faisant partie de cette mission. S'il n'existe pas d'autorisation (une permission ou une obligation pour cette mission), alors l'agent est en infraction et de ce fait, l'agent *InstManagerAg* doit être prévenu.

En supplément de cette tâche réactive, l'agent *NormManagerAg* doit accomplir une tâche plus pro-active ne faisant pas l'objet d'une demande d'un autre agent institutionnel. En effet, une norme peut être violée parce qu'elle n'est plus active ou valide et que la mission n'a pas été accomplie dans le cas d'une obligation par exemple. La description des façons de détecter une violation de norme est abordée dans le chapitre précédent à la Section 6.5.3. La détection est influencée par ce qui déclenche le test (changement de contexte, demande d'un agent) et par le type de la norme (obligation, permission, interdiction). Nous entrerons dans les détails dans le chapitre suivant consacré à l'implémentation du modèle et des agents de $\mathcal{S}$YNAI.

7.3.6 InstManagerAg

Cet agent a la charge de gérer l'Institution dans son ensemble, c'est-à-dire à la fois les agents du domaine, et plus particulièrement leurs entrées et sorties de l'Organisation, mais également les autres agents institutionnels en leur déléguant la surveillance de chaque instance de spécification avec une obligation de retour en cas de violation de contrainte.

Les actions concernant l'Organisation, mises en place par l'*InstManagerAg* sont *createMember* et *deleteMember*, permettant la création ou la destruction d'un *OrgWrapperAg* et de son "Member" associé, permettant à l'agent du domaine qui en fait directement la demande de pouvoir interagir dans l'Organisation. Ainsi, un agent voulant intégrer l'Organisation devra en premier lieu contacter l'agent institutionnel *InstManagerAg* et lui envoyer un message contenant l'événement *createMember* indiquant qu'il veut faire en sorte que la méthode du même nom soit exécutée. Il fait de même avec un message contenant l'événement *deleteMember* quand il veut quitter l'Organisation.

Cet agent étant le superviseur général de $\mathcal{S}$YNAI, c'est lui qui coordonne l'arbitrage mais c'est également lui qui prend l'initiative de créer et de détruire les instances de schémas concernant $\mathcal{S}$YNAI en envoyant les événements *createScheme*, *abortScheme* et *finishScheme* à l'agent *FunctManagerAg*. Cela veut dire que non seulement les agents de supervision vont devoir communiquer entre eux pour se coordonner sur l'arbitrage (demande de vérification de norme, annonce de violation de norme, etc) mais également pour agir directement sur l'Organisation (adoption de rôle, satisfaction de but). En effet, comme nous avons pu l'annoncer dans cette section, les agents de $\mathcal{S}$YNAI ont leur Organisation spécifiée avec $\mathcal{M}$OISE^{Inst}. C'est ce que nous allons voir dans la partie suivante, tandis que nous

détaillerons comment les agents de supervision interagissent les uns avec les autres lorsque nous aborderons les protocoles de supervision dans la section 7.5.

7.4 Spécification de $\mathcal{S}$ynai avec $\mathcal{M}$oise^{Inst}

De la même façon que nous spécifions une organisation d'agents à l'aide de $\mathcal{M}$OISE^{Inst} afin de définir comment ils sont structurés les uns par rapport aux autres, les buts qu'ils doivent atteindre et les règles qu'ils doivent respecter, nous définissons l'organisation que les agents de $\mathcal{S}$YNAI devront respecter. Nous spécifions cette organisation institutionnelle également avec $\mathcal{M}$OISE^{Inst}. Cela permet de rendre explicite l'Organisation du Système d'Arbitrage et d'apporter plus de flexibilité. L'OS du Système d'Arbitrage que nous fournissons avec MA$\mathcal{B}_{\text{ELI}}$ est une solution qui peut être changée par l'utilisateur si elle ne lui convient pas. De plus, avoir une représentation explicite de l'OS de $\mathcal{S}$YNAI permet d'ajouter d'autres agents capables d'interpréter l'organisation et d'épauler les agents existants. Les agents du domaine et les agents de supervision faisant partie d'une seule organisation, il faut pouvoir regrouper la spécification organisationnelle du domaine et la spécification organisationnelle du système d'arbitrage en une seule spécification organisationnelle complète de l'Institution Électronique. Avoir une organisation de $\mathcal{S}$YNAI spécifiée avec $\mathcal{M}$OISE^{Inst} facilite cela.

Nous allons présenter dans cette section les SS, FS, CS et NS composant une OS possible de $\mathcal{S}$YNAI. Dans chaque spécification d'Institution Électronique utilisant le modèle MA$\mathcal{B}_{\text{ELI}}$, la spécification organisationnelle du système d'arbitrage sera celle-ci par défaut et quelle que soit l'application dans laquelle notre modèle sera utilisé. Ceci est possible grâce à une spécification générique que nous faisons de $\mathcal{S}$YNAI. Nous allons également voir dans les sections suivantes les règles d'intégration entre l'OS de $\mathcal{S}$YNAI et celle de l'application dans laquelle elle est utilisée.

7.4.1 Structure des agents (SS)

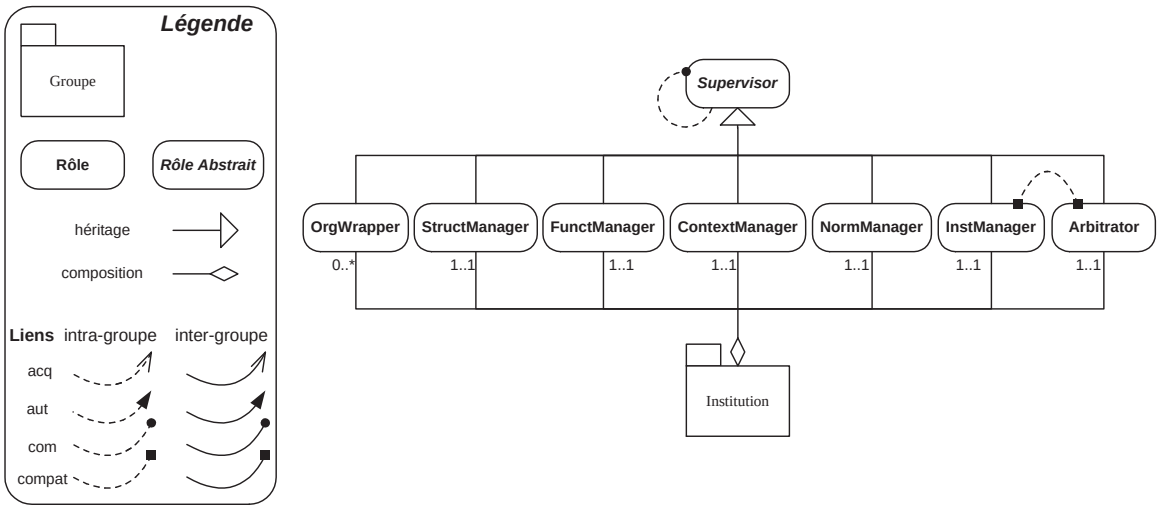


FIG. 7.3 – SS de $\mathcal{S}$YNAI

Chaque agent de $\mathcal{S}$YNAI devra jouer un rôle particulier. Ces rôles sont représentés sur la FIG. 7.3. Il est évident que l'agent *OrgWrapperAg* jouera le rôle d'"OrgWrapper", l'agent *StructManagerAg* celui de "StructManager" et ainsi de suite. Les rôles "InstManager" et "Arbitrator" étant compatibles entre eux, c'est l'agent *InstManagerAg* qui les jouera. Tous ces rôles sont au même niveau et héritent tous du rôle abstrait "Supervisor". Comme précisé dans la description de la structure de $\mathcal{S}$YNAI, tous les rôles pourront communiquer les uns avec les autres grâce au lien de communication défini sur

le rôle “Supervisor”. Ces rôles appartiennent au groupe “Institution” et ont tous la cardinalité de 1..1, sauf le rôle “OrgWrapper” puisqu'il y aura dans l'instance de cette organisation autant d'agents *OrgWrapperAg* que d'agents du domaine. Il faut donc qu'ils puissent tous jouer un rôle.

Règles d'intégration

Afin d'intégrer la Spécification Structurelle du domaine (la SS de l'application) à la Spécification Structurelle du système d'arbitrage (la SS de $\mathcal{S}$YNAI), les règles suivantes doivent être respectées :

- Tous les rôles du domaine héritent du rôle abstrait de base “soc”.
  $\forall r \in \mathcal{R}_{ss_{domain}}, \nexists r' \in \mathcal{R}_{ss_{domain}} : r \sqsubset r' \Rightarrow r \sqsubset r_{soc}$
- Il existe un lien d'autorité allant du rôle “Supervisor” du système d'arbitrage au rôle de base “soc” du domaine.
  $\exists\, l \in \mathcal{L}_{Institution} : l = link(r_{Supervisor}, r_{soc}, aut)$
- Le groupe “Institution” est composé du groupe racine du domaine.
  $gr_{root} \in \mathcal{G}r_{ss_{domain}} : \forall\, gr' \in \mathcal{G}r_{ss_{domain}}, gr \notin \mathcal{SG}r_{gr'} \Rightarrow gr_{root} \in \mathcal{SG}_{Institution}$
- L'ensemble des entités structurelles de l'Organisation est composé de l'ensemble des entités structurelles de la SS du domaine et de l'ensemble des entités structurelles de la SS du système d'arbitrage.
  $\mathcal{E}_{ss} = \mathcal{E}_{ss_{domain}} \cup \mathcal{R}_{ss_{synai}}$

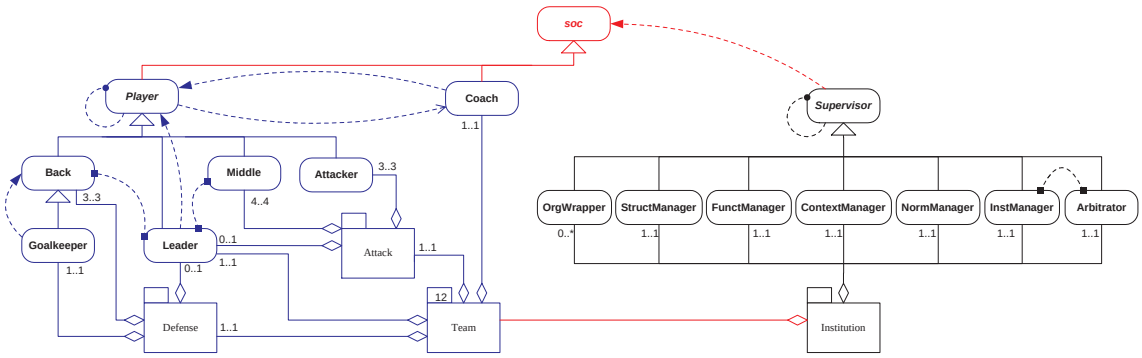


FIG. 7.4 – SS globale de l'exemple des “Robots Footballeurs”

Si nous prenons l'exemple des “Robots Footballeurs”, après l'application des règles d'intégration, nous obtenons une SS d'Institution représentée par la FIG. 7.4. Nous retrouvons bien le rôle “soc” dont les rôles “Player” et “Coach” héritent et le lien d'autorité existant entre le rôle “Supervisor” et le rôle “soc” donnant à tous les rôles de supervision l'autorité sur les rôles du domaine. De plus le groupe “Institution” est composé du groupe de base “Team”. Cela n'affecte pas spécialement les rôles et les groupes du domaine mais sert à lier les deux spécifications et à représenter le fait que le groupe de base de l'Institution est composé des rôles de supervision d'une part et des groupes et sous-groupes à arbitrer d'autre part.

Maintenant que nous avons assemblé en une seule spécification la définition de la structure d'une Institution Électronique, il nous faut définir des fonctionnalités.

7.4.2 Fonctionnalités des agents (FS)

Quand nous voudrons utiliser le modèle MA$\mathcal{B}_{\text{ELI}}$ au sein d'une application, les fonctionnalités de cette dernière seront divisées en deux ; les fonctionnalités propres au domaine d'application (par exemple “marquer un but” pour l'exemple des “Robots Footballeurs”) et celles propres à l'arbitrage du fonctionnement des agents du domaine. Les fonctionnalités du domaine, donc la définition de la

FS, sont propres à chaque application tandis que les fonctionnalités d'arbitrage que nous proposons par défaut pour $\mathcal{S}$YNAI sont communes à toutes les applications. Ainsi, nous fournissons une méthode d'arbitrage au sein de MA$\mathcal{B}_{\text{ELI}}$ utilisant un ensemble de buts possibles issus d'une analyse du processus de détection, de contrôle et de sanction de spécification effectuée avec $\mathcal{M}$OISE^{Inst}.

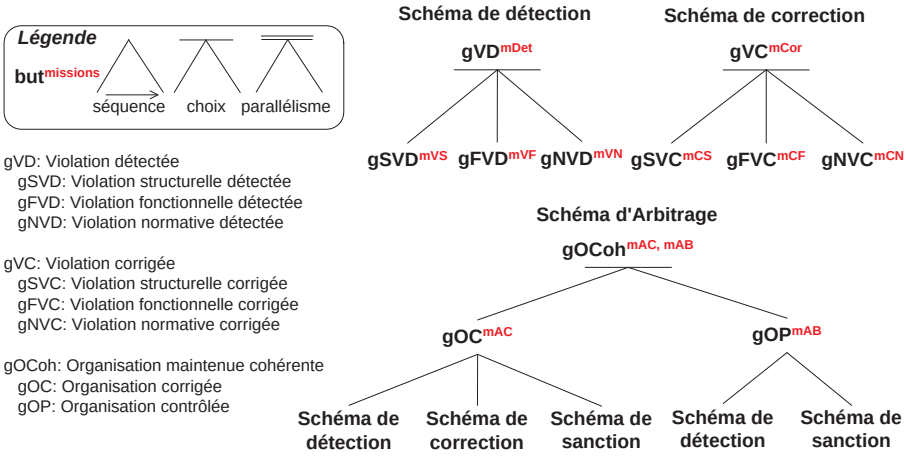


FIG. 7.5 – FS de $\mathcal{S}$YNAI

Le principe de cette méthode d'arbitrage permet à l'utilisateur de choisir la stratégie d'arbitrage qu'il veut utiliser. Comme illustré sur la FIG. 7.5, nous fournissons deux possibilités dont les buts sont de garder l'organisation cohérente en corrigeant les éventuelles violations ou de garder l'organisation cohérente en prévenant la violation. Nous pouvons parler ici de contrôle. La différence entre les deux stratégies se situe au niveau de la présence ou non de la fonction de correction. Soit l'Organisation est contrôlée et aucune violation ne peut survenir car elle serait détectée et bloquée, soit l'Organisation autorise les violations et une fois qu'elles ont été détectées, elles sont ensuite corrigées. Ainsi, par rapport au schéma fonctionnel "Schéma d'arbitrage" illustré sur la figure, le but global est de maintenir l'organisation dans un état cohérent. Un état de l'organisation cohérent correspond à une instance d'Organisation respectant les contraintes définies dans la spécification de l'institution. Pour atteindre ce but racine, le but *gOC* ou le but *gOP* doit être atteint en fonction du type d'arbitrage voulu. Nous avons vu dans la section précédente concernant la description des agents institutionnels que la détection d'un état incohérent se faisait lorsqu'un agent du domaine envoyait un message pour agir sur l'instance de l'organisation. Suivant l'agent qui atteint le but, l'incohérence sera considérée comme la résultante d'une violation de la SS (but *gSVD*), de la FS (but *gFVD*) ou de la NS (but *gNVD*).

Une fois la violation détectée et potentiellement corrigée (buts *gSVC*, *gFVC* et *gNVC*) l'agent fautif doit être puni. L'éventuelle correction pourrait se faire en parallèle de la sanction, mais pour des facilités de modélisation, les deux ou trois schémas s'exécutent en séquence. Nous considérons qu'au sein de l'arbitrage, même si la détection des violations est identique, les sanctions peuvent différer suivant les applications. Les nouvelles fonctionnalités apportées par $\mathcal{M}$OISE^{Inst} nous permettent de faire référence dans le schéma d'arbitrage à un schéma de sanction défini dans la FS du domaine. Ainsi, la détection par les agents institutionnels d'une violation de la part des agents du domaine entraîne la création et l'exécution d'une instance de schéma de sanction défini par l'utilisateur pour l'utilisation de MA$\mathcal{B}_{\text{ELI}}$ dans son application.

Les **Règles d'intégration** consistent pour la FS de l'Institution Électronique à regrouper l'ensemble des schémas d'exécution de la FS du domaine et l'ensemble des schémas de la FS de $\mathcal{S}$YNAI.

$$\mathcal{FS} = \langle \mathcal{S}_{fs_{domain}} \cup \mathcal{S}_{fs_{synai}}, \mathcal{G}o_{fs_{domain}} \cup \mathcal{G}o_{fs_{synai}}, \mathcal{PR}_{fs_{domain}} \cup \mathcal{PR}_{fs_{synai}} \rangle$$

La FIG. 7.6 nous montre le résultat de cette intégration pour l'exemple des "Robots Footballeurs". La FS de l'exemple ne comportant pas de schéma de sanction, lorsque les buts *gOC* ou *gOP* seront atteints, les agents de $\mathcal{S}$YNAI devront exécuter seulement les schémas de détection et éventuellement de correction. Nous avons remplacé sur la figure les références faites à ces schémas au sein du schéma d'arbitrage par les schémas eux-mêmes (entourés de pointillés). Une fois les règles appliquées, la FS de l'institution définit à la fois le fonctionnement des agents du domaine et celui des agents institutionnels.

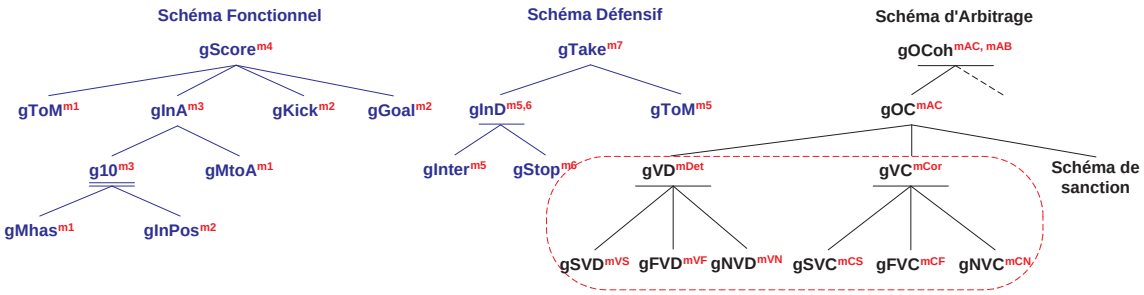


FIG. 7.6 – FS globale de l'exemple des "Robots Footballeurs"

A l'aide des sous-buts *gOC* et *gOP*, nous donnons la possibilité d'arbitrer une organisation de deux façons différentes. Le choix du but à atteindre se fera en fonction de la mission sur laquelle l'agent institutionnel *InstManagerAg* se sera engagé. Les agents institutionnels respectant leurs contraintes, *InstManagerAg* sera engagé sur une mission qu'il sera autorisé à accomplir. Nous verrons dans la Section 7.4.4 que c'est au moment de la définition des normes que le lien est fait entre les contextes et les schémas d'arbitrage afin qu'un contexte oblige les agents de supervision à superviser d'une façon plutôt que d'une autre en accomplissant les missions adéquates. Ainsi, la possibilité de choisir la stratégie d'arbitrage se fait en partie grâce à la CS de $\mathcal{S}$YNAI que nous allons maintenant développer.

7.4.3 Stratégies d'arbitrage (CS)

La Spécification Contextuelle est le moyen de faire en sorte que les agents de Supervision exécutent tel ou tel schéma d'arbitrage par le biais des normes. Nous définissons ici deux contextes différents correspondant aux deux stratégies d'arbitrage spécifiées dans la FS, à savoir l'arbitrage correctif ou l'arbitrage préventif.

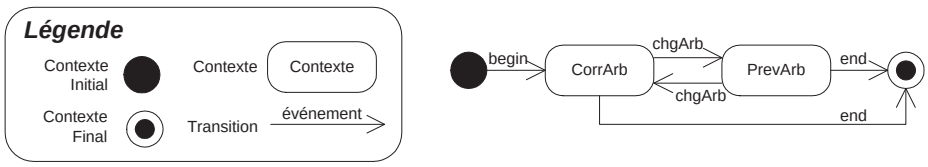


FIG. 7.7 – CS de $\mathcal{S}$YNAI

La FIG. 7.7 représente la CS de $\mathcal{S}$YNAI composée de deux contextes. Le contexte "CorrArb" correspond à un contexte d'arbitrage avec correction tandis que le contexte "PrevArb" correspond à un contexte d'arbitrage avec blocage des violations. Lorsque l'Organisation est créée, suivant l'événement qui lui est envoyé, elle se trouvera dans un des deux contextes d'arbitrage. Ensuite, pendant toute la durée de vie de l'Organisation, le passage d'un contexte à un autre sera possible par l'envoi d'un message comportant l'événement "chgArb". De même, l'Organisation pourra se retrouver dans le

contexte final pour ensuite être détruite, à partir de n'importe quel contexte.

Règles d'intégration

Pour obtenir la CS de l'Institution, les règles suivantes doivent être appliquées :

- L'ensemble des événements de la CS est la jonction de l'ensemble des événements de la CS du domaine et de l'ensemble des événements de la CS de $\mathcal{S}$YNAI.
  $\mathcal{E}_{cs} = \mathcal{E}_{cs_{domain}} \cup \mathcal{E}_{cs_{synai}}$
- L'ensemble des contextes de la CS est constitué d'un seul contexte nommé "GlobalContext".
  $\mathcal{C}_{cs} = GlobalContext$
- L'ensemble des sous-contextes de "GlobalContext" est composé de l'ensemble des contextes de la CS de $\mathcal{S}$YNAI et de l'ensemble des contextes de la CS du domaine.
  $\mathcal{SC}_{GlobalContext} = \mathcal{C}_{cs_{domain}} \cup \mathcal{C}_{cs_{synai}}$

Nous représentons sur la FIG. 7.8 la CS de l'Institution pour l'exemple des "Robots Footballeurs". Nous avons omis de représenter sur ce schéma le contexte de base des CS du domaine et de $\mathcal{S}$YNAI constituant l'ensemble des sous-contextes du contexte "GlobalContext". Ces deux contextes de bases sont eux-mêmes constitués des sous-contextes "Defense", "Middle" et "Attack" pour la CS du domaine et des sous-contextes "CorrArb" et "PrevArb" pour la CS de $\mathcal{S}$YNAI. Avec cette CS globale d'Institution, l'Organisation peut se trouver en même temps dans un contexte d'arbitrage par correction ou d'arbitrage préventif et dans un contexte de phase de défense ("Defense"), de phase de milieu de terrain ("Middle") ou de phase d'attaque ("Attack"), tout en étant dans le contexte global "GlobalContext".

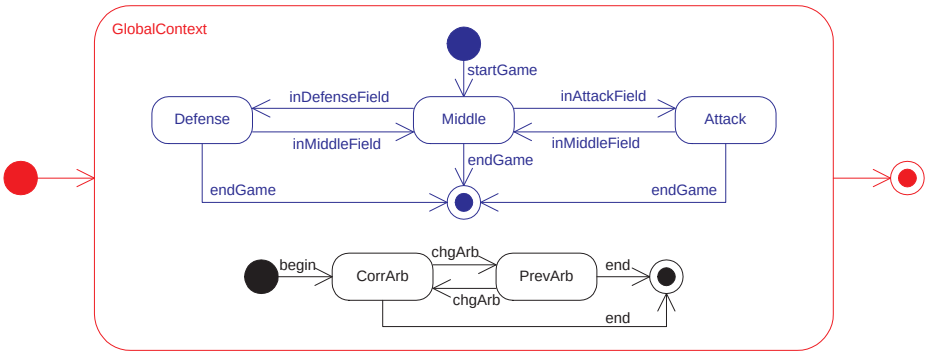


FIG. 7.8 – CS globale de l'exemple des "Robots Footballeurs"

Les contextes ne contraignent pas en eux-mêmes les agents. Le type d'arbitrage ne peut pas être défini seulement grâce à la spécification de plusieurs contextes. Il faut lier les schémas d'arbitrage aux contextes pour définir dans quel contexte tel schéma devra être exécuté ou non. C'est dans les normes de la NS que les règles de fonctionnement des agents institutionnels sont définies.

7.4.4 Règles de fonctionnement des agents (NS)

La définition de la NS de $\mathcal{S}$YNAI définit les buts à atteindre des agents institutionnels en fonction des missions qu'ils seront autorisés ou obligés d'accomplir. Pour la spécification du système d'arbitrage que nous proposons, nous autorisons les agents à s'engager sur les missions qui leur permettront d'arbitrer les agents du domaine. Comme nous considérons que les agents institutionnels respectent tout le temps leurs normes, celles-ci ne définissent pas de sanctions. De même, comme elles doivent être applicables durant toute la durée de vie de l'organisation, elles ne comprennent pas de date limite

de validité.

context	id	w.	condition	issuer	bearer	deOp	mission	deadline	sanction
CorrArb	NA01	1	---	InstManager	Arbitrator	O	mAC	---	---
CorrArb	NA02	1	---	Supervisor	Arbitrator	O	mCor	---	---
CorrArb	NA03	1	---	Supervisor	StructManager	O	mCS	---	---
CorrArb	NA04	1	---	Supervisor	FunctManager	O	mCF	---	---
CorrArb	NA05	1	---	Supervisor	NormManager	O	mCN	---	---
BlocArb	NA06	1	---	Supervisor	Arbitrator	O	mAB	---	---
---	NA07	1	---	Supervisor	Arbitrator	O	mDet	---	---
---	NA08	1	---	Supervisor	StructManager	O	mVS	---	---
---	NA09	1	---	Supervisor	FunctManager	O	mVF	---	---
---	NA10	1	---	Supervisor	NormManager	O	mVN	---	---

TAB. 7.2 – NS de $\mathcal{S}$YNAI

Les normes de la NS du système d'arbitrage sont répertoriées dans le tableau 7.2. Nous les avons regroupées par contexte. Nous constatons ainsi avec les normes NA03, NA04 et NA05 que si l'organisation se trouve dans le contexte "CorrArb", les agents jouant les rôles "StructManager", "FunctManager" et "NormManager" devront réparer les violations de la structure (mission mCS), des fonctionnalités de l'organisation (mission mCF) ou des normes (mission mCN). Par contre, si nous nous trouvons dans le contexte "BlocArb", nous obligeons seulement "Arbitrator" a garder une organisation cohérente (norme NA06). Enfin, quel que soit le contexte, les rôles "StructManager", "FunctManager" et "NormManager" doivent détecter les violations en accomplissant respectivement les missions mVS, mVF et mVN (normes NA08, NA09 et NA10).

context	id	w.	condition	issuer	bearer	deOp	mission	deadline	sanction
Attack	N01	1	---	Coach	Attack	O	m4	---	N10
Attack	N02	1	---	Coach	Attacker	O	m2	---	---
Attack	N03	1	---	Coach	Middle	P	m1	---	---
Attack	N04	1	---	Coach	Attack	O	m3	---	---
Middle	N05	1	---	Coach	Middle	O	m1	---	---
Middle	N06	1	---	Coach	Back	P	m5	---	---
Defense	N07	1	---	Coach	Player	O	m7	---	---
Defense	N08	1	---	Coach	Back	O	m5	---	---
Defense	N09	1	---	Coach	Goalkeeper	O	m6	---	---
	N10	1	*violated(N01)*	Coach	Player	O	m7	---	---
CorrArb	NA01	1	---	InstManager	Arbitrator	O	mAC	---	---
CorrArb	NA02	1	---	Supervisor	Arbitrator	O	mCor	---	---
CorrArb	NA03	1	---	Supervisor	StructManager	O	mCS	---	---
CorrArb	NA04	1	---	Supervisor	FunctManager	O	mCF	---	---
CorrArb	NA05	1	---	Supervisor	NormManager	O	mCN	---	---
BlocArb	NA06	1	---	Supervisor	Arbitrator	O	mAB	---	---
	NA07	1	---	Supervisor	Arbitrator	O	mDet	---	---
	NA08	1	---	Supervisor	StructManager	O	mVS	---	---
	NA09	1	---	Supervisor	FunctManager	O	mVF	---	---
	NA10	1	---	Supervisor	NormManager	O	mVN	---	---

TAB. 7.3 – NS globale de l'exemple des "Robots Footballeurs"

Les **Règles d'intégration** spécifient que la NS globale de l'Institution est le regroupement en un seul ensemble des normes du domaine avec les normes du système d'arbitrage.

$$\mathcal{NS} = \mathcal{NS}_{domain} \cup \mathcal{NS}_{synai}$$

Les normes étant référencées différemment(NXX pour le domaine et NAXX pour $\mathcal{S}$YNAI), il n'y a pas de risque de conflit. Nous avons décrit dans le TAB. 7.3 l'ensemble des règles pour l'exemple des

"Robots Footballeurs".

Nous avons spécifié dans cette section l'organisation de $\mathcal{S}$YNAI avec $\mathcal{M}$OISE^{Inst}. Les Agents Institutionnels ont donc des buts à atteindre dont un en particulier, celui de garder l'Organisation cohérente en détectant les susceptibles violations de spécifications. Pour satisfaire ces buts, il faut que les agents jouent également leurs rôles d'intermédiaire entre les agents du domaine et l'Organisation. Les agents de $\mathcal{S}$YNAI doivent donc communiquer avec les agents de l'application et en fonction des résultats des tests, agir sur l'Organisation. Les tests ont été abordés dans la section 7.2 de ce chapitre et seront formalisés dans le chapitre suivant. Nous allons donc voir dans ce qui suit la spécification des interactions entre agents.

7.5 Protocoles d'Interaction pour la Supervision d'Institution

Nous allons définir dans cette section la façon dont les agents de $\mathcal{S}$YNAI doivent interagir entre eux de manière théorique. Il y a trois grandes phases de communication durant la durée de vie d'une organisation de type $\mathcal{M}$OISE^{Inst}, à savoir i) l'entrée dans l'organisation, ii) l'activité dans l'organisation et iii) la sortie de l'organisation. Pour chacune de ces étapes, des interactions ont lieu entre les agents du domaine (donc l'agent *OrgWrapperAg*) présents pour agir afin d'atteindre un objectif propre à l'application modélisée et les agents de $\mathcal{S}$YNAI dont le but est de vérifier que les agents du domaine atteignent bien ces objectifs. Pour chacune des étapes citées précédemment, nous avons défini un ensemble de protocoles de supervision. Nous allons les détailler dans les sections qui suivent. Nous verrons également en quoi consiste un arbitrage pouvant intervenir dans n'importe quelle étape du cycle de vie d'un agent au sein d'une organisation.

Le problème auquel nous devons faire face est de mettre en place une communication entre des agents du domaine, hétérogènes, ne connaissant pas forcément le protocole à suivre, et les agents de $\mathcal{S}$YNAI. Le fait d'avoir un agent *OrgWrapperAg* permet de régler la communication avec les agents institutionnels. Nous n'avons plus qu'à nous soucier des communications au sein de $\mathcal{S}$YNAI, l'agent *OrgWrapperAg* étant également un agent institutionnel. Il faut toutefois que l'agent du domaine venant d'arriver commence par communiquer avec un agent, *InstManagerAg* en l'occurrence. La communication entre les agents consiste en un échange de messages composés d'un événement ayant une signification particulière pour les agents. L'utilisation du terme 'événement' peut paraître abusif dans le sens où ce que nous nommons événements représentent le langage de contenu pour les communications entre les superviseurs et l'*OrgWrapperAg*. Nous les définissons dans ce qui suit.

7.5.1 Définition des événements institutionnels

Les agents du domaine vont interagir avec les agents de $\mathcal{S}$YNAI (via leur *OrgWrapperAg* respectif) afin d'agir au sein de l'organisation. Pour cela ils vont devoir communiquer. Cela se fait grâce à l'envoi de messages respectant le standard FIPA-ACL pour la définition des protocoles. Nous verrons dans le chapitre suivant que les messages sont au format KQML, car ayant été définis de la sorte dans $\mathcal{M}$OISE$^+$. Ces messages vont contenir des événements permettant aux agents soit de faire une requête d'action au sein de l'Organisation (relative à la structure ou au fonctionnement), soit de faire des demandes d'information (à propos de la structure, du fonctionnement ou des normes). Les agents de $\mathcal{S}$YNAI vont devoir non seulement répondre aux requêtes d'actions ou d'informations mais aussi interagir entre eux pour se coordonner sur les activités de supervision. Les événements dont nous parlons ici constituent, avec les événements définis dans la CS, l'ensemble des événements pouvant survenir au sein de l'Organisation.

Nous distinguons deux sortes d'événements contenus dans les messages définissant les protocoles de communication : ceux permettant aux agents de demander l'état de l'Organisation et ceux permettant aux agents de se lancer des requêtes sur la possibilité d'agir sur l'Organisation. Nous allons voir ces deux catégories dans lesquelles nous regroupons tous les événements déclenchant les actions étudiées au début de ce chapitre.

Événements relatifs aux demandes d'action sur l'Organisation

Ces événements ont généralement le même nom que les actions décrites dans la Section 7.2 qu'ils sont censées déclencher. Les Superviseurs recevant ces requêtes d'action vont tout d'abord mettre en place une série de tests afin de vérifier qu'aucune violation de la spécification de l'organisation n'a lieu. Ces tests sont les tests institutionnels nécessaires à la détection ou non d'une violation lors de l'exécution des actions. Comme les actions, les événements relatifs aux demandes d'action sont regroupés en trois groupes, celui concernant les actions sur la structure, celui concernant les actions sur le fonctionnement et enfin celui concernant la coordination entre les agents de supervision.

Événements de requête d'action	**Événements de réponses**
createMember	*welcome*
deleteMember	*goodbye*
Événements de demande d'action sur la Structure de l'Organisation	**Événements de réponses**
adoptRole	*roleAdopted* *roleRefused*
giveUpRole	*roleGivenUp*
Événements de demande d'action sur le Fonctionnement de l'Organisation	**Événements de réponses**
createScheme	*schemeCreated*
finishScheme	*schemeFinished*
commitToMission	*missionForbidden* *missionAuthorized* *missionRefused* *missionCommited*
uncommitToMission	*missionUncommited*
setGoalSatisfied	*goalRefused* *goalNotCommitted* *goalCommitted* *goalSatisfied*
checkNorms	*missionForbidden* *missionAuthorized* *goalRefused* *goalSatisfied*

Événements relatifs aux informations concernant l'Organisation

Événements de demande d'information	**Événements de réponse**
availableRole	*roleAvailable*
commitableMission	*missionCommitable*
Événements d'information	**Événements de réponse**
NSViolated	*violationTreated*
FSViolated	*violationTreated*
SSviolated	*violationTreated*
Tous les événements définis et utilisés au sein de la CS	*contextChanged*

Maintenant que nous avons vu les différents événements pouvant servir de réponse à l'envoi d'autres événements, nous allons définir comment ces ensembles d'échanges sont organisés en protocoles que les agents devront suivre afin de communiquer entre eux et se comprendre. Nous allons pour cela regrouper les différents protocoles selon les phases de communication définies en début de section.

7.5.2 Entrée dans l'Organisation

Afin de rejoindre une Organisation, un agent du domaine doit obtenir la création d'un agent *OrgWrapperAg* lui permettant de communiquer avec les autres agents (du domaine et de $\mathcal{S}$YNAI) et d'agir sur l'organisation en exécutant les méthodes de l'objet "Member" qui lui est associé. Les communications avec les autres agents du domaine faisant partie de la même organisation sont fonction des liens existant entre les rôles que chacun des agents jouent. Une fois l'agent entré dans l'organisation, il doit adopter un rôle au sein d'un groupe afin d'accomplir un ensemble de mission en atteignant des buts.

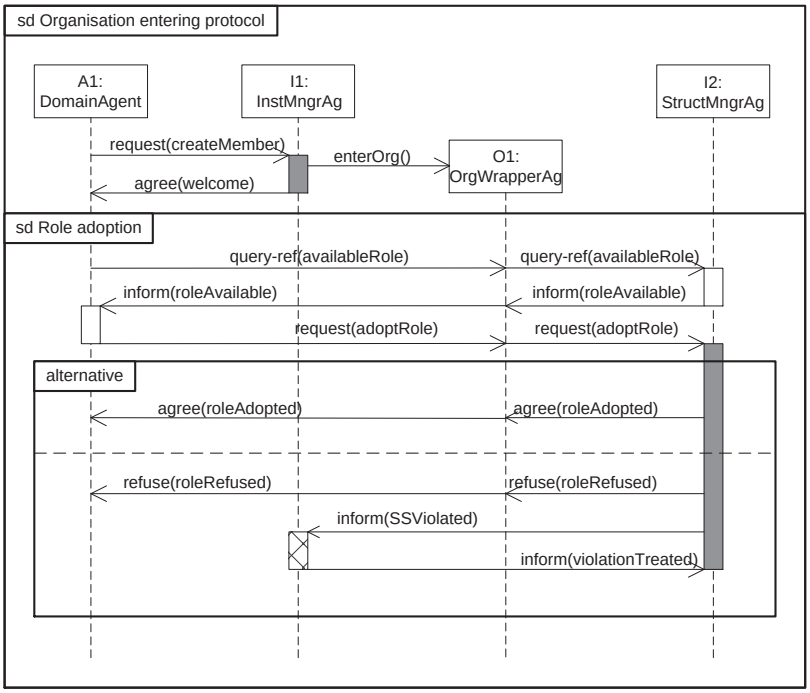


FIG. 7.9 – Protocoles de la phase d'entrée dans l'Organisation définis en AUML

Sur le diagramme de séquence de la FIG. 7.9, nous retrouvons ces deux protocoles d'entrée dans l'organisation et d'adoption de rôle. L'agent du domaine envoie un message à l'agent *InstManagerAg* qui, rappelons-le, gère l'instance d'organisation dans sa globalité et crée un agent *OrgWrapperAg* qui deviendra le seul interlocuteur pour l'agent du domaine. Il est à noter que les éléments en gris représentent les processus mis en place à la suite de la réception d'un événement de demande d'action sur l'Organisation nécessitant l'application de tests institutionnels (cf. Section 7.2).

Afin d'atteindre des buts au sein de l'Organisation, l'agent du domaine va vouloir connaître via son *OrgWrapperAg* les rôles qu'il peut jouer en envoyant l'événement *availableRole*. L'agent *StructManagerAg* répond avec une vue de la Structure de l'Organisation afin que l'agent connaisse la structure de l'organisation et son instanciation actuelle. De ce fait, il pourra vouloir adopter un rôle disponible ou non. Son autonomie est préservée. Ayant fait son choix, il envoie l'événement *adoptRole* avec en paramètre le groupe qu'il veut rejoindre et le rôle qu'il veut jouer. L'agent *StructManagerAg* vérifie l'état de la Structure par rapport à sa spécification et décide si oui ou non l'agent du domaine peut, via son agent *OrgWrapperAg*, jouer ce rôle au sein de l'organisation. S'il ne peut, une instance de rôle est créée, sinon, un message de violation de la SS est envoyé au *InstManagerAg* (événement *SSViolated*). Nous verrons à la fin de cette section comment un message de ce genre est traité par le système d'arbitrage. Maintenant qu'il fait partie de l'organisation, notre agent du domaine va pouvoir

agir au sein de l'organisation en atteignant des buts et en accomplissant des missions.

7.5.3 Activité au sein de l'organisation

Pour une organisation de type $\mathcal{M}\text{OISE}^{Inst}$, une activité consiste en l'engagement sur une mission visant à atteindre les buts qui la composent. Quand un agent prétend avoir effectivement atteint un but, ce n'est pas de notre ressort ni de celui des agents institutionnels de vérifier que c'est effectivement vrai ou de prétendre le contraire. Du point de vue de l'organisation, cela consiste seulement à annoncer qu'un agent a atteint un but en changeant l'état d'une instance. Pour ce faire, l'agent *OrgWrapperAg* doit envoyer un message à l'agent *FunctManagerAg*, d'une part pour connaître les missions sur lesquelles il peut s'engager et d'autre part pour effectivement s'engager sur une mission. Ceci étant fait, il n'aura plus qu'à atteindre les buts composant sa ou ses missions.

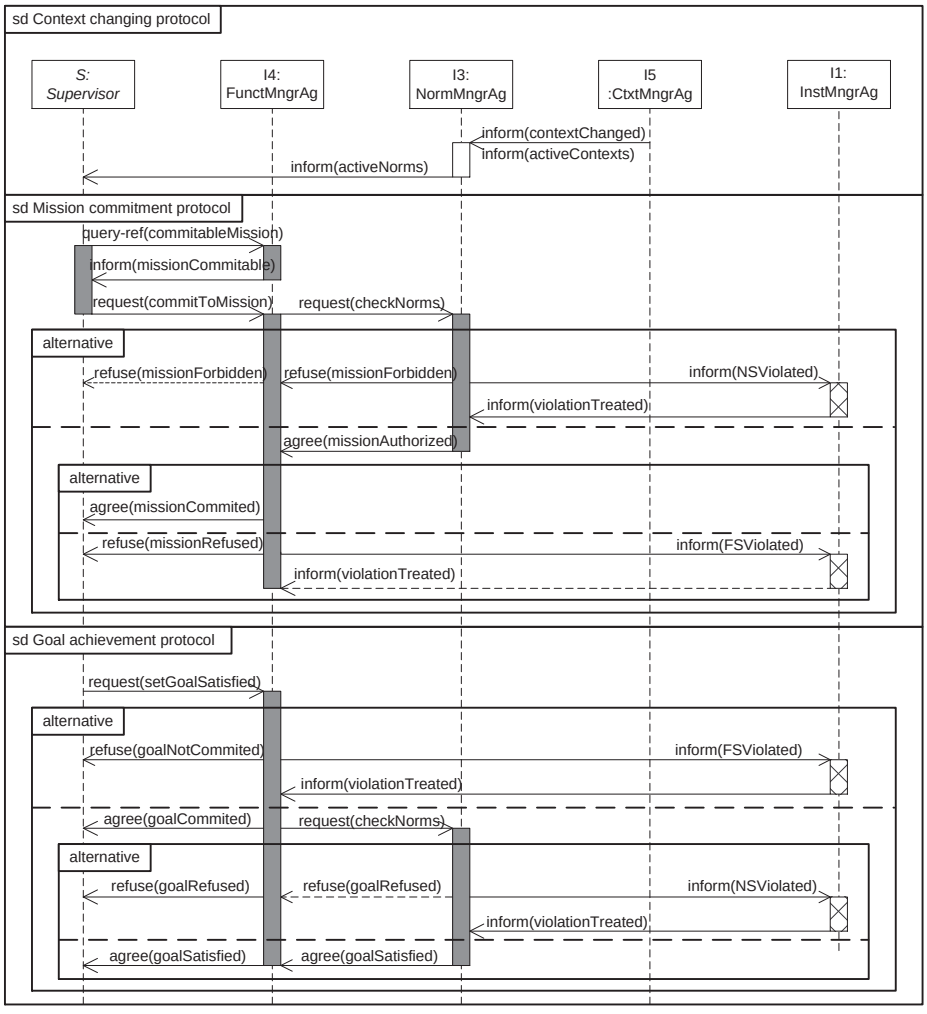


FIG. 7.10 – Protocoles d'activité au sein de l'organisation définis en AUML

L'agent *Supervisor* qui est présent sur le diagramme représente n'importe quel agent de $\mathcal{S}\text{YNAI}$. Nous aurions simplement pu mettre *OrgWrapperAg* mais les agents institutionnels aussi doivent s'engager sur leurs missions et atteindre leurs buts (nous en verrons une partie dans la description du protocole d'arbitrage). De ce fait, nous n'avons pas représenté les interactions entre l'agent du do-

maine et son *OrgWrapperAg*.

Sur le diagramme de séquence de la FIG. 7.10 nous considérons qu'un agent s'engage sur une mission puis atteint un but de cette mission. Nous avons cependant commencé par faire apparaître le message que le *ContextManagerAg* envoie au *NormManagerAg* l'avertissant que le contexte a changé et de ce fait que les normes ont pu potentiellement être mises à jour. L'agent du domaine (via son *OrgWrapperAg*) est au courant des normes actives le concernant quand il en fait la demande et à chaque changement d'état. Il est important de noter que les trois protocoles présentés sur le diagramme et décrits dans ce paragraphe peuvent avoir lieu dans un ordre différent. Il n'y a pas de contrainte d'enchaînement spécifique entre ces protocoles.

Chaque agent pouvant interpréter une organisation de type $\mathcal{M}$OISE^{Inst} sait qu'il doit s'engager sur des missions. Pour cela, il doit d'abord connaître les missions sur lesquelles il peut le faire. L'envoi de l'événement *commitableMission* permet de recevoir en retour, en paramètre de l'événement *missionCommitable*, les missions disponibles, c'est-à-dire les missions obligées ou autorisées pour le rôle que l'agent est en train de jouer. Comme pour l'adoption de rôle, l'agent *FunctManagerAg* retourne une vue du Fonctionnement de l'Organisation. Ainsi, l'agent du domaine connaissant (via son *OrgWrapperAg*) les missions spécifiées, celles déjà instanciées et celles qu'il est autorisé à accomplir, peut faire une demande d'engagement sur n'importe quelle mission (événement *commitToMission*). Bien entendu, le *FunctManagerAg* vérifie dans un premier temps que la mission est bien autorisée (événement *checkNorms* déclenchant la vérification de l'existence d'une obligation ou d'une permission concernant la mission) et dans un deuxième temps qu'elle peut bien être instanciée. Si l'un des deux tests ne passe pas, une violation de NS ou de FS est envoyée à *InstManagerAg*. Nous verrons plus tard à quoi correspond le traitement représenté par une barre d'activation hachurée. De même, si le test ne passe pas, l'événement *missionRefused* est envoyé au *OrgWrapperAg* qui enverra un message compréhensible à l'agent du domaine. Si l'engagement sur la mission est accepté, l'agent du domaine est averti positivement (*missionCommited* pour le *OrgWrapperAg*). Le même processus a lieu pour la mise à jour d'un but en but atteint. La seule différence (mis à part les messages) est l'ordre des tests, d'abord on regarde si le but fait bien partie d'une mission sur laquelle l'agent est bien engagé et ensuite si ce but appartient bien à une mission autorisée (*checkNorms*).

Nous venons de voir au sein de ces protocoles l'envoi de messages contenant les événements *SSViolated*, *FSViolated* ou *NSViolated* permettant de donner l'alerte quant à une violation d'une spécification de $\mathcal{M}$OISE^{Inst}. Le traitement de ces violations est représenté sur les diagrammes par un processus hachuré. Nous allons maintenant détailler cette activité d'arbitrage et ses conséquences.

7.5.4 Activité d'arbitrage

Nous avons vu précédemment que le but racine du schéma d'arbitrage de la FS de $\mathcal{S}$YNAI est de maintenir l'organisation cohérente. Quelle que soit la stratégie d'arbitrage choisie, et de ce fait le sous-but à atteindre, une détection de violation doit avoir lieu à travers l'exécution du schéma de détection. Pour chaque activité d'arbitrage, c'est-à-dire pour chaque détection de violation, une instance de schéma est exécutée. Les buts feuilles du schéma de détection de violation sont atteints quand l'agent envoie l'événement *SSViolated* ou *FSViolated* ou *NSViolated* à l'agent *InstManagerAg*. En fait, en même temps qu'il enverra ce message, en tant que joueur de rôle ayant une mission à accomplir, il enverra au *FunctManagerAg* un message alertant qu'il a atteint un but de détection de violation.

Prenons l'exemple d'une détection de violation de la structure comme représentée sur la FIG. 7.11. Nous avons repris à côté une version simplifiée du schéma d'exécution de la détection et de la correction d'une violation. En l'occurrence, on considère que nous nous trouvons dans un contexte d'arbitrage préventif, c'est-à-dire qu'aucune action de correction n'est mise en place. Les agents institutionnels acceptent ou n'acceptent pas les requêtes des agents du domaine. Même si ces derniers ont la liberté de vouloir violer leur contrainte, leurs actions sont bloquées par les agents du système

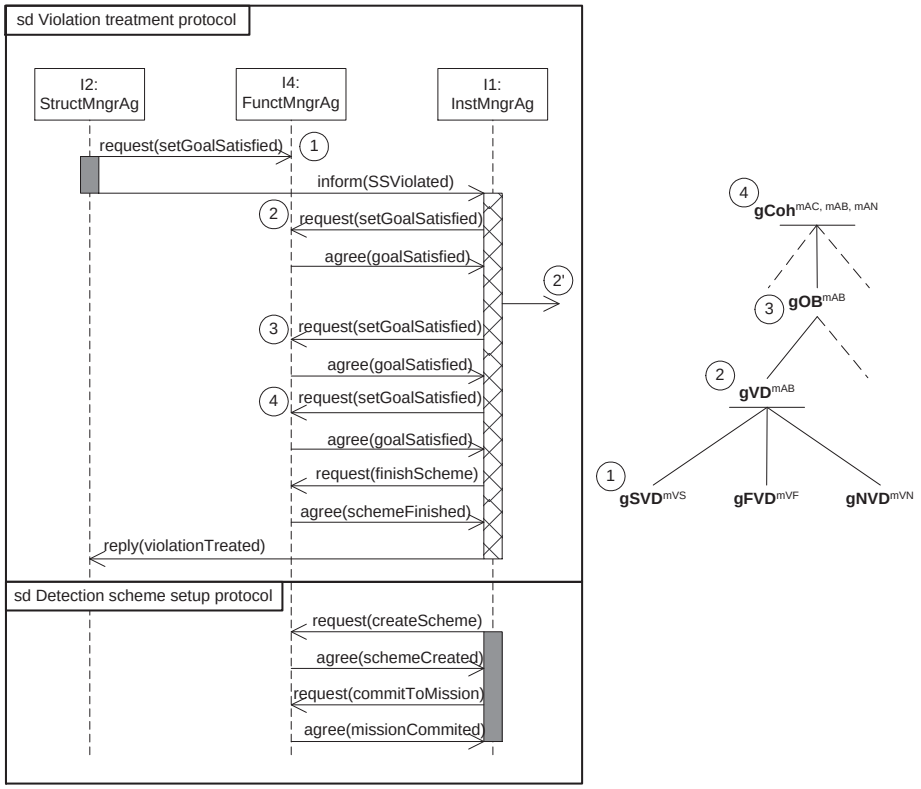


FIG. 7.11 – Protocole d'arbitrage d'une violation de SS défini en AUML

d'arbitrage. Dans ce contexte, l'agent *StructManagerAg* détecte une violation de la SS. De ce fait, il atteint le but composant sa mission et envoie donc l'événement *setGoalSatisfied* au *FunctManagerAg* en passant en paramètre le but "gSVD" (étape 1). En même temps, il avertit l'agent *InstManagerAg* qu'une violation a eu lieu. Ce dernier considère donc que le but "gVD" est atteint (d'où l'envoi de l'événement) puisque son plan a été consommé (étape 2). Ceci étant, pour que la mission mAB soit accomplie, une sanction doit être appliquée. La flèche partant dans le vide représente en fait la création, l'exécution et la terminaison d'un schéma de sanction par le *InstManagerAg* en tant que rôle "Arbitrator" (étape 2´). Cet agent peut ensuite atteindre le but "gOB" puisque une violation a été détectée et sanctionnée (étape 3). Enfin le dernier but "gCoh" est atteint également puisque correspondant au choix de stratégie d'arbitrage (étape 4). La mission mAB est donc accomplie et le schéma d'arbitrage terminé (événement *finishScheme*). L'arbitrage est terminé (pour cette violation) et l'agent *InstManagerAg* le fait donc savoir au détecteur en envoyant l'événement *violationTreated*.

Le schéma d'arbitrage étant terminé plus aucune mission de détection, de correction et de punition de violation ne peut avoir lieu. C'est pourquoi il faut instancier ce schéma une nouvelle fois après chaque fin d'exécution. Ceci correspond à la deuxième partie du diagramme de séquence de la FIG. 7.11. L'agent *InstManagerAg* crée un nouveau schéma et s'engage sur la mission principale. Pour que les autres agents s'engagent de nouveau sur les missions contenant les buts feuilles du schéma, il faut soit que ces agents détectent automatiquement que ces missions sont disponibles, soit que l'agent *InstManagerAg* les avertisse en usant d'un quelconque moyen de coordination. Nous verrons ceci dans la suite de cette section après avoir abordé les interactions survenant lors de la sortie d'un agent du domaine de l'organisation.

7.5.5 Sortie de l'Organisation

Comme toute chose, une organisation a une durée de vie, mais pour qu'elle se termine il faudrait que plus aucun agent n'en fasse partie. La participation d'un agent au sein d'une organisation a un début (nous l'avons vu) et une fin lorsque cet agent décide de quitter l'organisation. Pour ce faire, il ne doit plus être engagé sur des missions ni jouer de rôle. Sa sortie de l'organisation se traduira par la suppression de l'agent le représentant parmi les agents de $\mathcal{S}$YNAI, à savoir un agent *OrgWrapperAg*.

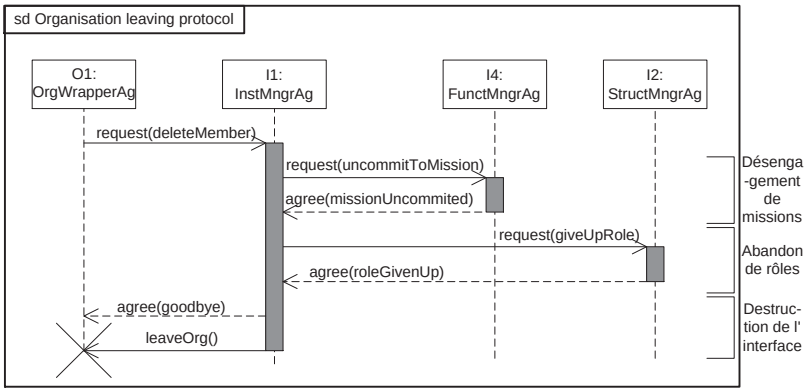


FIG. 7.12 – Protocole de sortie de l'organisation définit en AUML

Nous illustrons sur le diagramme de séquence de la FIG. 7.12 le cas où l'agent qui décide de sortir de l'organisation joue encore des rôles et doit encore accomplir des missions. La réaction de l'agent *InstManagerAg* est de le désengager des missions sur lesquelles il est et de lui faire abandonner ses rôles. Une alternative aurait été de faire un test sur les missions et rôles et interdire la sortie de l'organisation tant que la situation n'aurait pas été débloquée. Ici en l'occurrence, l'*OrgWrapperAg* envoie l'événement de sortie *deleteMember* ce qui déclenche chez l'*InstManagerAg* l'envoi des messages *uncommitMission* et *giveUpRole* pour toutes les missions sur lesquelles l'agent du domaine est engagé et tous les rôles que cet agent joue. Une fois libéré des contraintes de structure et de fonctionnement (et donc de ce fait de ces normes) *InstManagerAg* envoie un dernier message à *OrgWrapperAg* qui sera transféré à l'agent du domaine avant de détruire son point d'entrée avec l'organisation. Il n'est plus représenté au sein de l'organisation donc il n'existe plus.

7.6 Synthèse

Nous avons défini dans ce chapitre le rôle de $\mathcal{S}$YNAI au sein de MA$\mathcal{B}_{\text{ELI}}$. Il sert en effet de *middleware* entre les agents du domaine et l'organisation au sein de laquelle ils évoluent et sur laquelle ils vont agir. Son but est de faire en sorte que les actions mises en place par les agents amènent l'Organisation dans un état cohérent avec sa spécification. Pour ce faire, un ensemble d'agents spécialisés interceptent les accès à l'Organisation et testent leur validité quant au respect de la spécification. Nous avons parlé du rôle et du but de $\mathcal{S}$YNAI. Il est en effet possible de définir une spécification d'organisation pour les agents de $\mathcal{S}$YNAI. Nous définissons également un ensemble de protocoles contraignant les agents à envoyer des messages contenant des événements particuliers afin de communiquer entre eux.

Nous avons vu jusqu'à présent comment contraindre les agents du domaine avec $\mathcal{M}$OISE^{Inst} et comment les contrôler avec $\mathcal{S}$YNAI. Nous devons maintenant fournir l'API permettant à des agents de prendre en compte une spécification $\mathcal{M}$OISE^{Inst} ainsi que les agents de $\mathcal{S}$YNAI. Pour tester leur fonctionnement, nous pouvons utiliser un outil capable de simuler un contrôle du respect des contraintes.

C'est ce que nous allons voir dans le chapitre suivant.

Chapitre 8

Mise en œuvre de Ma$\mathcal{B}_{\text{eli}}$

Nous avons présenté dans le Chapitre 6 le modèle $\mathcal{M}$OISE^{Inst} constituant le coeur de notre modèle d'Institution Électronique. C'est en effet en fonction d'un modèle conforme à $\mathcal{M}$OISE^{Inst} que différents agents vont pouvoir évoluer et agir au sein d'une Organisation afin de respecter leurs contraintes. Jusqu'à présent, nous avons représenté le modèle et les organisations en découlant par des schémas et sous forme d'expressions formelles. Nous avons également décrit le modèle de description d'Institution $\mathcal{M}$OISE^{Inst} à l'aide du langage BNF dont le détail se trouve en Annexe B. Ce qu'il nous faut à présent, c'est pouvoir décrire une OS $\mathcal{M}$OISE^{Inst} de manière assez simple pour être compris par un utilisateur et suffisamment structuré pour être interprété par un programme informatique. Une façon efficace de faire est de nous baser sur un document structuré et de définir le langage MoiseIML (Moise Institution Markup Language).

Une fois une OS obtenue à l'aide de ce langage de description, il faut pouvoir l'implémenter (d'un point de vue programmation) afin de l'instancier (d'un point de vue organisationnel) pour qu'un ensemble d'agents agissent en son sein. Nous fournissons pour cela une API permettant le passage d'une description sous forme de document structuré à un ensemble d'objets. Les objets représentent ainsi l'Organisation et peuvent être manipulés par des agents.

Dans ce chapitre, nous présentons dans un premier temps la façon dont nous avons défini le langage structuré MoiseIML et dans un deuxième temps comment nous sommes passés de ce langage capable de définir des OS sous forme de documents structurés à une API capable d'implémenter ces OS en un ensemble d'objets pour pouvoir ensuite l'instancier en Organisation. Puis nous nous intéresserons à l'implémentation des agents de $\mathcal{S}$YNAI. Nous finirons par la présentation d'un outil de test des Institutions Ma$\mathcal{B}_{\text{ELI}}$ permettant d'illustrer d'une part la mise en œuvre de nos API et d'autre part de tester l'efficacité du contrôle des agents de $\mathcal{S}$YNAI.

8.1 Description de Spécification d'Organisation $\mathcal{M}$oise^{Inst}

L'implémentation du modèle $\mathcal{M}$OISE^{Inst} passe par la définition d'une structure de données permettant aux utilisateurs de décrire une organisation grâce au langage MoiseIML. Pour cela nous allons définir la structure des documents à partir de la description BNF du modèle $\mathcal{M}$OISE^{Inst} puis nous la transformerons en un langage de programmation objet.

8.1.1 Langage de description d'Institution MoiseIML

Dans notre cas, nous utilisons en tant que structure de données décrivant une organisation un fichier XML basé sur une DTD. Ce choix se justifie par le fait qu'il était plus simple de transformer la DTD existante décrivant $\mathcal{M}$OISE^{+} que de créer un schéma XML pour la description de $\mathcal{M}$OISE^{Inst}. Une DTD représente le modèle $\mathcal{M}$OISE^{Inst} et les fichiers XML conformes à cette DTD décrivent les

instances d'organisation. Le passage de la description BNF du modèle $\mathcal{M}$OISE^{Inst} à la DTD suit un ensemble de règles de transformation que nous décrivons ci-dessous.

Règles de transformation

Pour obtenir la DTD, à chaque élément de vocabulaire du langage correspond une balise XML :
- les éléments structurés deviennent des balises `<!ELEMENT>` ;
- les éléments représentant des données de base (entier, chaîne de caractères, booléen, etc.) deviennent des éléments `<!ATTLIST>` ;
- les expressions régulières spécifiant le nombre de fois qu'une occurrence peut être présente sont traduites en expression XML à savoir + pour une ou plusieurs fois, $*$ pour une ou plusieurs fois et ? pour zéro ou une fois.

Exemple

En appliquant ces principes à notre modèle $\mathcal{M}$OISE^{Inst}, nous obtenons la DTD que nous détaillons en Annexe C. Nous reprenons dans la FIG. 8.1 une partie du modèle afin d'illustrer les principes de transformation. La version simplifiée de $\mathcal{M}$OISE^{Inst} possède une OS composée d'un identifiant, d'une SS, d'une FS, d'une CS et d'une NS. Puis, nous définissons la composition d'une SS par un ensemble de groupes et de rôles (sans prendre en compte les liens qui existent entre eux).

Langage BNF	***Fichier DTD***

<OS> ::= '(os' :id <osId>
:ss <SS> :fs <FS> :cs <CS> :ns <NS>')'
	`<!ELEMENT OS`
	`(SS?, FS?, CS?, NS?)`
	`>`

<osId> ::= <string>	`<!ATTLIST OS`
	`osId CDATA #REQUIRED`
	`>`

<SS> ::= '(SS':role <role>* :group <group>*')'	`<!ELEMENT SS Role*, Group*)>`

<role> ::= '(':id <roleId>
[:extends <roleId>]')'
	`<!ELEMENT Role (extends?)>`
	`<!ELEMENT extends EMPTY>`
	`<!ATTLIST extends`
	`roleId CDATA #REQUIRED`
	`>`

<roleId> ::= <string>	`<!ATTLIST Role`
	`roleId CDATA #REQUIRED`
	`>`

<group> ::= '(':id <groupId>')'	`<!ELEMENT Group EMPTY>`

<groupId> ::= <string>	`<!ATTLIST Group`
	`groupId CDATA #REQUIRED`
	`>`

FIG. 8.1 – Correspondances entre les définitions de l'OS en BNF et sous forme de DTD

Sur la FIG. 8.1 nous visualisons la correspondance entre les informations en BNF et la DTD. Nous y retrouvons l'OS composé des éléments SS, FS, CS et NS ainsi que des attributs qui structurent chacun de ces éléments.

Ce format de structure de données a l'avantage d'être un standard dans le domaine de l'interopérabilité. A l'aide du modèle et d'une organisation en découlant, il nous est possible soit d'afficher des informations en traitant les données contenues soit de transformer de nouveau le fichier XML en structures objet grâce à un *parser* XML.

8.1.2 API orientée objet

Le langage de programmation utilisé pour implémenter à la fois les organisations et les agents (qu'ils soient du domaine ou de la couche $\mathcal{S}$YNAI) est JAVA. L'implémentation en JAVA de la description de la spécification de l'organisation nous permet ensuite de l'instancier afin d'être accessible par les agents.

Règles de transformation

Le passage d'un fichier DTD a une API Java respecte également quelques règles :
- chaque balise `<!ELEMENT>` XML devient une classe JAVA ;
- chaque attribut d'élément devient un attribut de classe ;
- les expressions régulières pour exprimer les nombres d'occurrences d'attributs sont transformées en cardinalités de lien.

Le traitement des données (attributs) se fait via des méthodes qui permettent d'avoir accès en lecture et en écriture à ces données (méthodes *get()* et *set()*). D'autres méthodes sont ajoutées afin de donner la possibilité de manipuler des objets autrement qu'en agissant sur leurs attributs. Par exemple, pour un objet de type `Role`, il peut s'agir d'une méthode capable de retourner toutes les normes relatives à ce rôle. Nous allons voir dans la section suivante l'API de $\mathcal{M}$OISE^{Inst} où nous détaillerons alors toutes ses méthodes.

Exemple

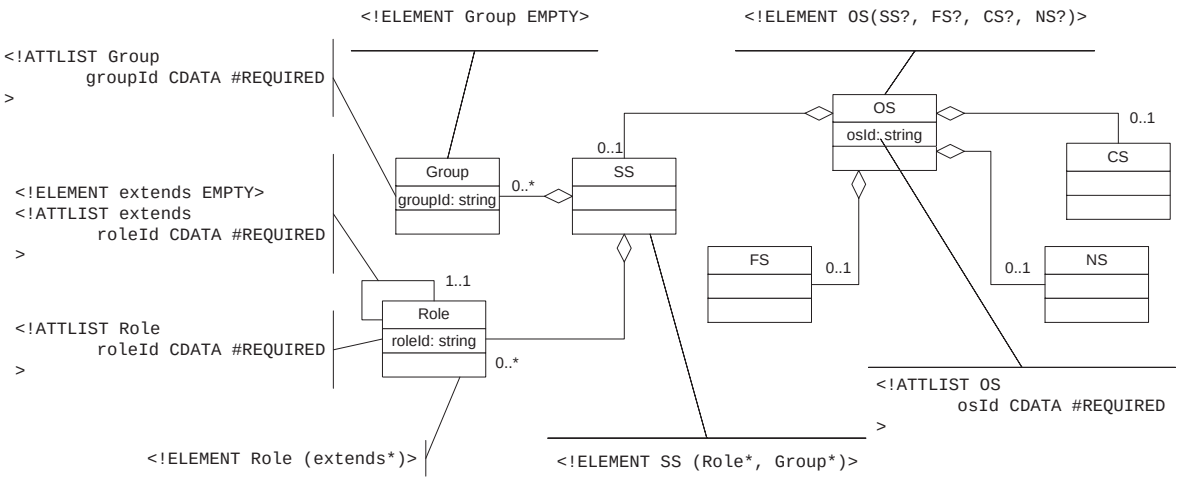


FIG. 8.2 – Correspondances entre les définitions de l'OS sous forme de DTD et d'API JAVA

Nous pouvons voir sur le FIG. 8.2 la correspondance entre le contenu du fichier DTD pour la version simplifiée de l'OS définie précédemment et son équivalent en classes JAVA, ici illustrée sous forme de diagramme de classes UML. On y retrouve la structure de données OS devenant la classe JAVA `OS`. Cette dernière est associée aux classes `SS`, `FS`, `CS` et `NS` avec une cardinalité de "0..1". On retrouve avec ces cardinalités les expressions régulières définissant que chaque spécification est optionnelle au sein de l'OS. Les attributs des structures deviennent des attributs de classe, en l'occurrence ici des identifiants. Nous n'avons pas fait apparaître pour chaque attribut les méthodes *get()* et *set()* permettant d'accéder à la valeur de l'attribut.

Nous allons maintenant entrer dans le détail de l'API JAVA de $\mathcal{M}$OISE^{Inst} considérée comme le modèle $\mathcal{M}$OISE^{Inst}. Une instanciation en programmation, c'est à dire la création d'objets, permet de

définir une organisation $\mathcal{M}$OISE^{Inst}. Cependant l'API JAVA de $\mathcal{M}$OISE^{Inst} permet aussi de définir les classes qui donneront les objets représentant une instance de l'OS. En plus de la décrire, l'API fournit les classes et les méthodes pour agir au sein de l'Organisation. Ainsi, l'API JAVA $\mathcal{M}$OISE^{Inst} est utilisée pour la description de l'OS et pour l'exécution de son instance.

8.2 Implémentation d'Organisation $\mathcal{M}$oise^{Inst}

L'API de $\mathcal{M}$OISE^{Inst} est composée de deux principaux *packages* permettant la définition de la spécification d'une organisation d'une part (package `moise.os`) et la définition de l'organisation d'autre part (package `moise.oe`). Pour fonctionner ces deux packages utilisent des classes communes regroupées dans le package `moise.common`. Détaillons à présent chacun d'entre eux.

8.2.1 Package moise.common

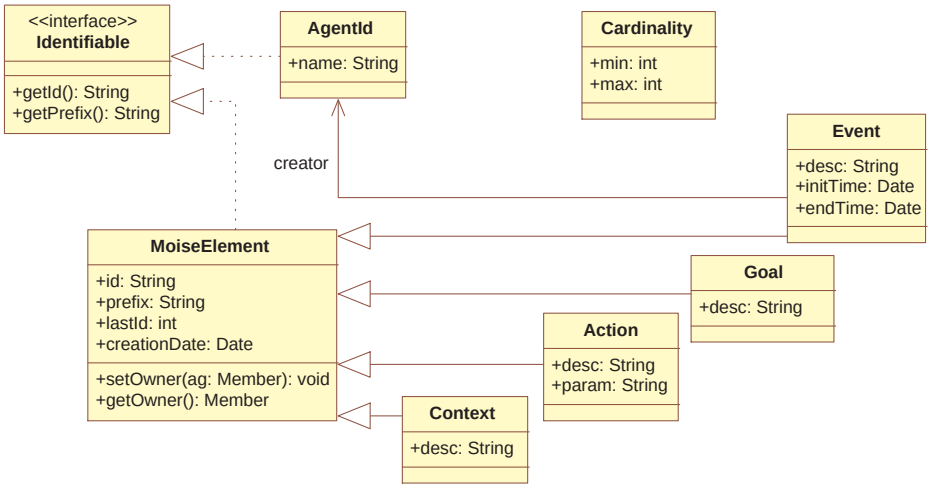


FIG. 8.3 – Diagramme de classes du package `moise.commom`

Nous trouvons dans ce package les classes de base sur lesquelles les classes de spécification d'organisation sont construites. Ces classes sont décrites dans un diagramme de classes illustré sur la FIG. 8.3. Toutes les classes représentant un élément de spécification de $\mathcal{M}$OISE^{Inst} héritent de la classe `MoiseElement`. Elle fournit un identifiant, un préfix et une date de création de l'objet. Les classes `Goal`, `Context`, `Event` et `Action` héritent de `MoiseElement`. Elles représentent des données de base utilisées dans plusieurs spécifications d'organisation. Une description est ajoutée à ces tâches. La classe `Action` a également un paramètre d'exécution tandis que la classe `Event` possède des dates de début et de fin.

8.2.2 Package moise.os

La FIG. 8.4 est une vue simplifiée du diagramme de classes du package `moise.os`. Ce dernier est composé principalement de la classe `OS` et des sous-packages `moise.os.ss`, `moise.os.fs`, `moise.os.cs` et `moise.os.ns` représentant comme leur nom l'indique, chaque vue d'une spécification d'organisation.

La classe `OS` représente la spécification de l'organisation. Elle est constituée des quatre spécifications SS, FS, CS et NS (liens vers les classes de même nom). Elle possède un attribut indiquant la référence à un fichier XML (URI) sur base duquel le modèle de spécification d'organisation JAVA va être construit. La méthode *addOS()* permet d'ajouter à chacune de ses spécifications les spécifications

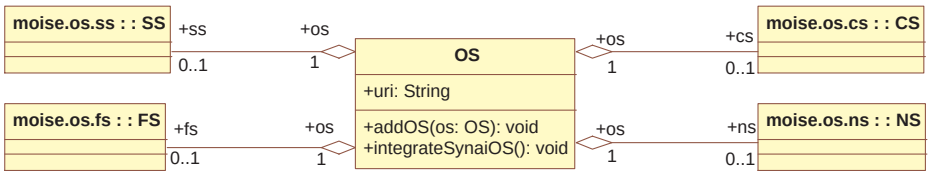


FIG. 8.4 – Diagramme de classes du package `moise.os`

d'une autre OS passée en paramètre. Cette méthode est utile lorsqu'il faut faire se joindre deux spécifications d'organisation. La méthode *integrateSynaiOS()* quant à elle utilise la méthode *addOS()* afin de pouvoir ajouter les spécifications de l'organisation de $\mathcal{S}$YNAI (définies dans un fichier XML fournies avec l'API) aux spécifications de l'organisation définies par l'utilisateur. Cette intégration se fait en respectant les différentes règles d'intégration requises pour chaque spécification et énoncées à la section 7.4 du Chapitre 7.

Nous allons maintenant détailler individuellement chaque *package* relatif à une spécification.

Package moise.os.ss

Comme illustré sur le diagramme de classes de la FIG 8.5, le package `moise.os.ss` se compose des classes `SS`, `SEntitySpec`, `GroupSpec`, `RoleSpec`, `RoleRel` et `Link`.

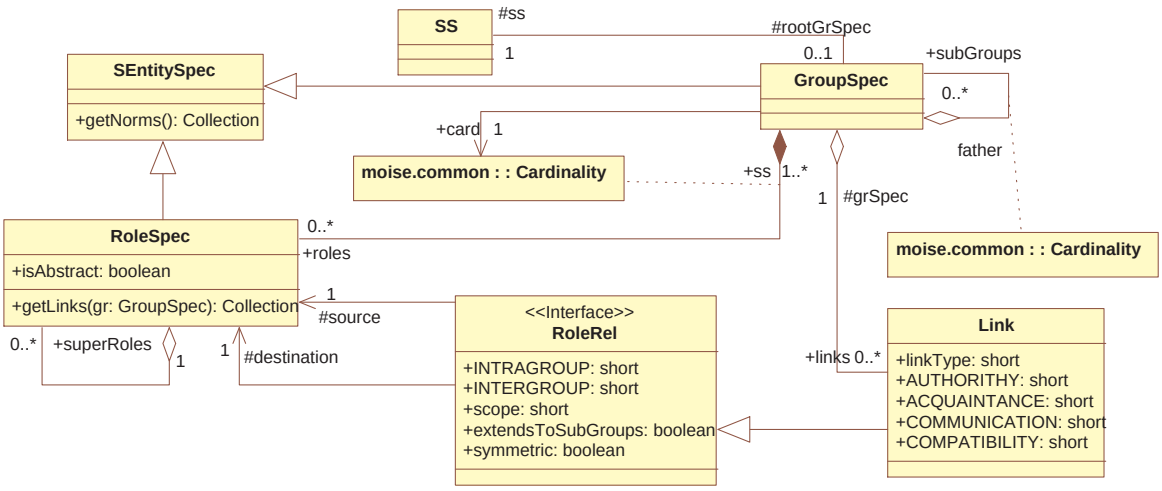


FIG. 8.5 – Diagramme de classes du package `moise.os.ss`

- La classe `SS` représente la spécification structurelle de la spécification de l'organisation. Elle fait référence à un groupe racine grâce au lien avec la classe `GroupSpec`.
- La classe `SEntitySpec` représente une entité structurelle. La méthode *getNorms()* permet de connaître toutes les normes concernant l'objet de type `SEntitySpec` depuis lequel est exécutée la méthode. La méthode retourne sans distinction toutes les normes où l'objet apparaît en tant que *issuer* ou en tant que *bearer*. La séparation si besoin est doit être faite ensuite. Les classes `RoleSpec` et `GroupSpec` héritent de la classe `SEntitySpec`.
- La classe `GroupSpec` représente la spécification d'un groupe. Elle possède un lien avec la classe `Cardinality` lui permettant de définir sa propre cardinalité (combien d'agents pourront appartenir à ce groupe). Un groupe étant composé de sous-groupes, il est lié à lui même avec `Cardinality` en classe d'association. Elle possède également d'une part un lien d'agrégation

avec la classe `RoleSpec` car un groupe est composé de rôles, et d'autre part avec la classe `Link` pour définir les liens entre ses rôles à'intérieur groupe.
- La classe `RoleSpec` représente la spécification d'un rôle. Elle est composée d'elle même afin de définir ses super rôles ainsi que d'un attribut définissant si le rôle est abstrait ou non. Elle exhibe une méthode permettant la récupération des liens référençant ce rôle en tant que rôle source ou rôle cible. Comme pour la méthode *getnorms()* de la classe `SEntitySpec`, la distinction entre les liens n'est pas faite.
- La classe `RoleRel` est une classe abstraite définissant la portée d'un lien entre deux rôles (liens source et destination), si le lien est extensible aux sous-groupes et s'il est symétrique.
- La classe `Link` hérite de la classe `RoleRel` et définit en plus le type de la relation entre les rôles : lien de communication, d'acquaintance, d'autorité ou de compatibilité.

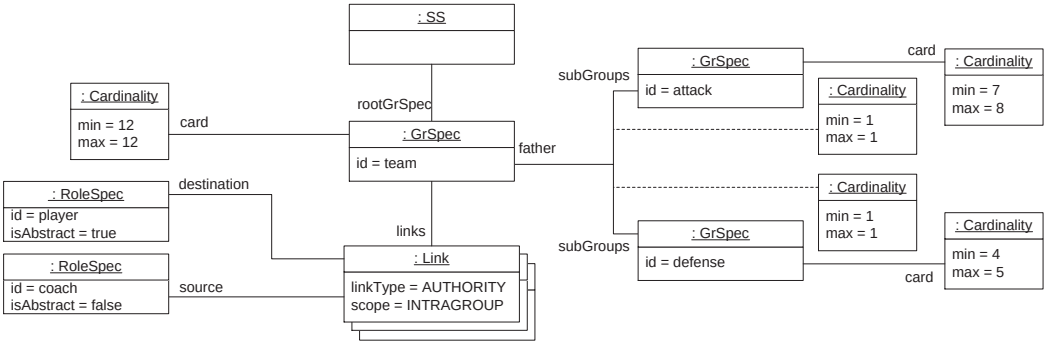


FIG. 8.6 – Diagramme d'instances partiel de la SS pour l'exemple des "Robots Footballeurs"

Si nous instancions ce package à l'aide de l'exemple des "Robots Footballeurs" abordé dans le Chapitre 6, nous obtenons le diagramme d'instances de la FIG. 8.6. Nous voyons ainsi que l'objet SS est composé du groupe "team". Ce groupe est quant à lui lié à un ensemble d'objets de type `Link` faisant référence chacun à deux rôles. Par soucis de lisibilité, nous ne représentons que deux rôles sur le diagramme (la totalité des rôles étant représentée sur le diagramme de la FIG. 8.7). Le groupe "team" est également composé des deux sous-groupes "attack" et "defense". Des objets `Cardinality` sont définis sur les liens entre le groupe racine et ses sous-groupes signifiant qu'un groupe "team" est composé d'un et un seul groupe "attack" et d'un et un seul groupe "defense" ; mais aussi liés avec les spécifications de groupe pour définir le nombre de joueurs minimum et maximum au sein de chaque groupe.

En ce qui concerne les rôles, nous illustrons leur structure sur le diagramme de la FIG. 8.7. Le rôle "player" est lié aux rôles "back", "leader", "middle" et "attacker" en tant que super-rôle. De même "back" est le super-rôle de "goalkeeper". Chaque rôle appartient à un ou plusieurs groupes, suivant les liens existants entre ces deux types d'objet. Ils ont leurs cardinalités exprimées grâce aux objets d'association de type `Cardinality`. Par exemple, le rôle "attacker" est présent trois, et seulement trois, fois dans le groupe "attack".

Package moise.os.fs

Comme illustré sur le diagramme de classes de la FIG 8.8, le package moise.os.fs se compose des classes `FS`, `SchemeSpec`, `GoalSpec`, `PlanSpec` et `MissionSpec`.

- La classe `FS` représente la spécification fonctionnelle. Une FS étant un ensemble de schémas fonctionnels, elle est associée à la classe `SchemeSpec`. Elle est également liée à la classe `MissionSpec` afin de définir l'ensemble de préférences sur les missions de la FS.

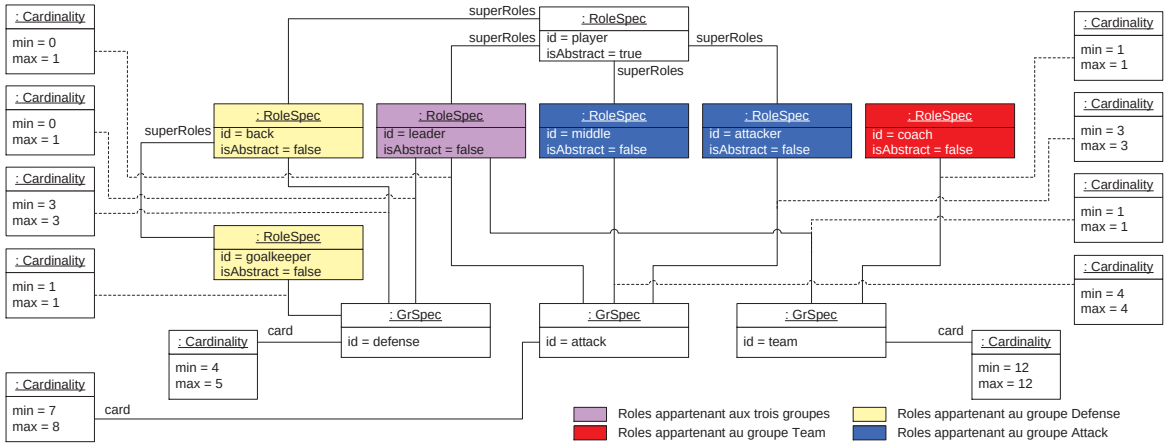


FIG. 8.7 – Diagramme d'instances partiel de la SS - Structure des rôles

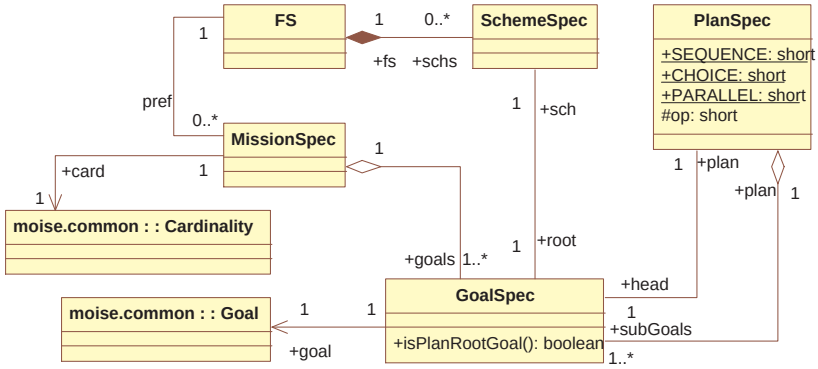


FIG. 8.8 – Diagramme de classes du package `moise.os.fs`

- La classe `SchemeSpec` représente un schéma d'exécution de l'organisation. Elle possède un lien avec la classe `GoalSpec` afin de définir le but racine du schéma.
- La classe `GoalSpec` représente un but. La méthode *isPlanRootGoal()* permet de savoir si ce but est le but racine d'un plan (si ce but possède un plan). Si le but a effectivement un plan, il est défini par le lien vers la classe `PlanSpec`.
- La classe `PlanSpec` représente l'ensemble des sous-buts spécifiant un autre but. La classe est composée de la classe `GoalSpec` représentant l'ensemble des sous-buts. Elle possède également un lien avec cette même classe pour définir le but racine du plan (le but qui est décrit par le plan en question). Cette classe possède également comme attribut un opérateur définissant le type du plan (parallèle, séquentiel, choix).
- Enfin la classe `MissionSpec` est composée de la classe `GoalSpec` (ensemble de buts définissant la mission) et fait référence à la classe `Cardinality` afin de définir le nombre d'agents pouvant s'engager en même temps sur la mission.

Si nous instancions ce package à l'aide de l'exemple des "Robots Footballeurs" nous obtenons le diagramme d'instances de la FIG. 8.9. Nous voyons ainsi que l'objet FS est composé d'un ensemble d'objets de type `SchemeSpec` et en particulier celui identifié comme étant le schéma "Functional_SCH". Ce schéma définit comme but racine le but "gScore".

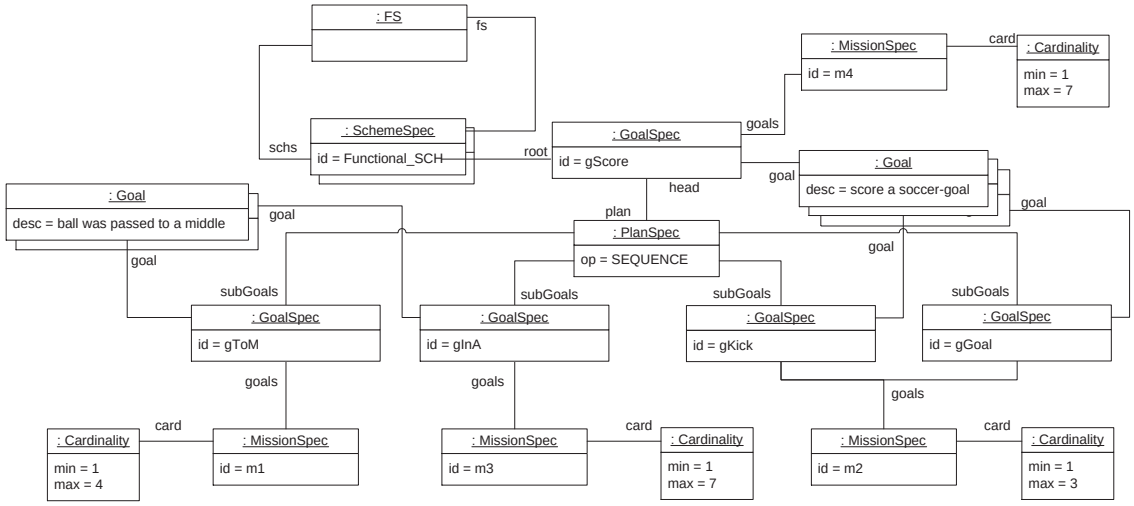


FIG. 8.9 – Diagramme d'instances de la FS pour l'exemple des "Robots Footballeurs"

Ce but est lié à un objet de type `PlanSpec` dont l'opérateur est la séquentialité (attribut *op*). Ce plan est composé des buts "gToM", "gInA", "gKick" et "gGoal" afin de spécifier le but "gScore". Tous les buts appartenant à ce plan sont normalement eux aussi spécifiés par un plan. Nous ne les avons pas représentés ici faute de place.

Les objets de type `MissionSpec` sont liés aux objets de buts les constituant. Ici nous avons les missions "m1", "m2", "m3", "m4" composées respectivement des buts "gToM", "gKick" et "gGoal", "gInA", "gScore". Les missions sont liées à des cardinalités indiquant combien d'agents vont pouvoir accomplir les missions.

Package moise.os.cs

Comme illustré sur le diagramme de classes de la FIG 8.10, le package moise.os.cs se compose des classes `CS`, `ContextSpec` et `Transition`.

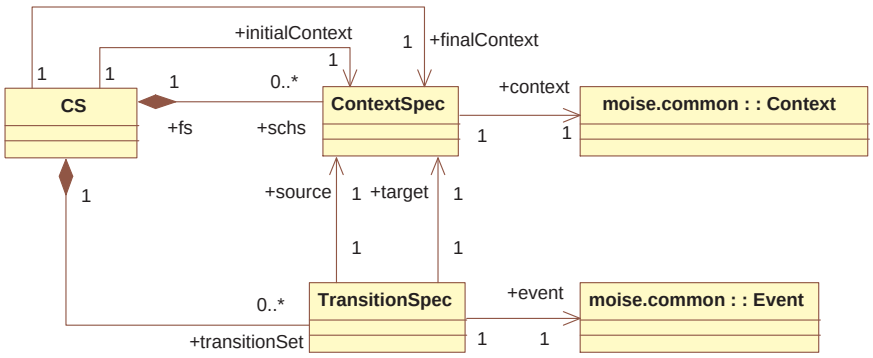


FIG. 8.10 – Diagramme de classes du package `moise.os.cs`

– La classe `CS` représente la spécification contextuelle d'une organisation. Elle est composée de la classe `ContextSpec` et de deux liens avec cette même classe pour définir le contexte initial et le contexte final de la spécification. Elle est également composée de la classe `TransitionSpec` définissant ainsi toutes les transitions qui peuvent avoir lieu.

– La classe `TransitionSpec` représente le passage d'un contexte source à un contexte destination. Deux liens avec la classe `ContextSpec` spécifient cela. Une transition étant déclenchée par un événement, un lien existe avec la classe `Event` du `package moise.common`.
– La classe `ContextSpec` représente un contexte de l'organisation. Pouvant être composé de sous-CS, cette classe possède un lien d'agrégation avec la classe `CS`.

Si nous instancions ce package à l'aide de l'exemple des "Robots Footballeurs" nous obtenons le diagramme d'instances de la Fig. 8.11. Nous voyons ainsi que l'objet CS est composé d'un ensemble d'objets de type `ContextSpec` et de type `TransitionSpec`. Par rapport à la représentation graphique que nous faisions de cette spécification dans la Section 6.4 du Chapitre 6, nous avons deux contextes en plus : les contextes identifiés par "Begin" et "End". En effet, comme l'objet CS doit pointer sur un contexte initial et un contexte final, nous ne pouvions pas respecter le schéma en définissant les deux contextes identifiés par "Attack" et "Defense" comme contextes initiaux et finaux. Ainsi le contexte initial est "Begin" et suivant l'événement qui décidera de qui engage la partie, la transition liée à cet événement fera en sorte que les agents se retrouveront soit dans un contexte de stratégie offensive soit dans un contexte de stratégie défensive. Mais quel que soit le contexte, un événement de fin de partie ("endGame") déclenchera une transition vers le contexte "End". Les transitions font ainsi référence à deux contextes (source et cible) ainsi qu'à un événement tandis que les spécifications de contextes font référence à une définition de contexte (objet de type `Context`).

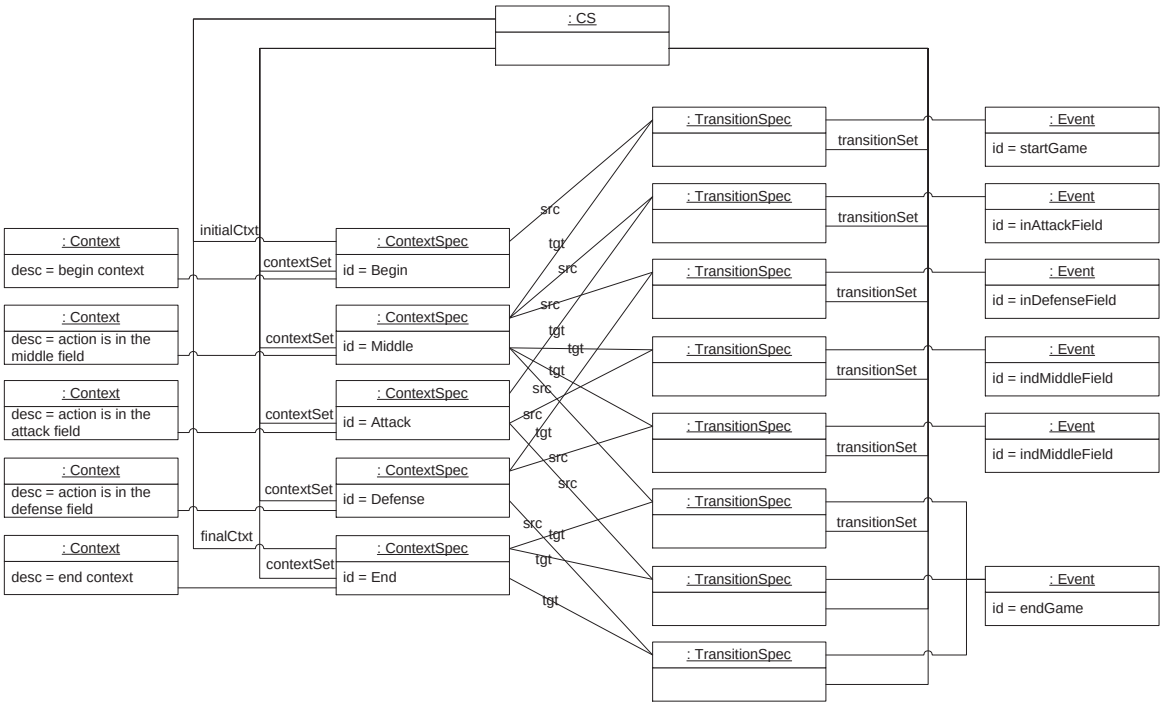


Fig. 8.11 – Diagramme d'instances de la CS pour l'exemple des "Robots Footballeurs"

Package moise.os.ns

Comme illustré sur le diagramme de classes de la Fig 8.12, le package moise.os.ns se compose des classes `NS`, `NormSpec`, `TimeConstraint` et `Condition`.

– La classe `NS` représente la spécification normative de l'organisation. Elle est composée de la classe `NormSpec` avec une cardinalité de "0..*".

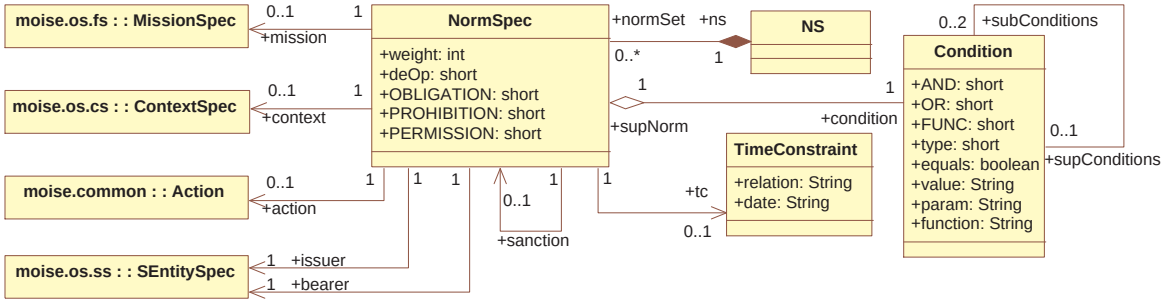


FIG. 8.12 – Diagramme de classes du package `moise.os.ns`

- La classe `NormSpec` est composée des classes `Condition` et `TimeConstraint` afin de définir la validité d'une norme. Elle possède également deux liens différents avec la classe `SEntitySpec` du package `moise.os.ss` pour définir le porteur et l'émetteur de la norme. La référence à cette classe permet d'avoir aussi bien un objet de type `RoleSpec` qu'un objet de type `GroupSpec` en tant qu'émetteur et porteur de la norme. La classe `NormSpec` possède plusieurs liens dont un avec la classe `ContextSpec` du package `moise.os.cs` pour définir le contexte d'activation de la norme ainsi que des liens avec les classes `MissionSpec` du package `moise.os.fs` et `Action` du package `moise.common` afin de définir sur quoi porte la norme. Enfin la classe `NormSpec` est liée à elle même pour faire référence à une autre norme en tant que sanction. Ses attributs définissent l'opérateur déontique (*deOp*) de la norme et son poids.
- La classe `Condition` permet d'exprimer les conditions de validité de la norme. Elle possède l'attribut `type` définissant si la condition est une fonction de base (`type = FUNC`) ou une expression composée de plusieurs conditions séparées par l'opérateur logique ET (`type = AND`) ou l'opérateur logique OU (`type = OR`). Si la condition est une fonction, alors la norme est valide si un paramètre (attribut `param`) est égal ou non (attribut `equals`) à une certaine valeur (attribut `value`). Si la condition joue le rôle d'opérateur logique, alors l'objet est obligatoirement lié à deux autre objets conditions (cardinalité de "0..2").
- La classe `TimeConstraint` permet de définir une contrainte temporelle. Cette classe possède donc les attributs `relation` et `date` afin de spécifier si la norme doit être respectée avant, pendant ou après (attribut `relation`) une date (attribut `date`). L'attribut date est de type `string` car il peut être une date exacte, une échelle de temps ou un contexte.

Si nous instancions ce package à l'aide de l'exemple des "Robots Footballeurs" nous obtenons le diagramme d'instances de la FIG. 8.13. Nous voyons ainsi que l'objet NS est composé d'un ensemble d'objets de type `NormSpec` eux-mêmes faisant référence à tous les objets que la norme implique dans sa définition. Ici la norme identifiée comme "N01" est mise en avant. Elle spécifie que les agents jouant un rôle au sein du groupe "Attack" ont l'obligation (attribut *deOp*) d'accomplir la mission "m4" (mission de marquer un but) si l'Organisation est dans un contexte de stratégie d'attaque (contexte "Attack"). Nous notons qu'aucun objet de type `TimeConstraint` est lié à la spécification de norme ce qui veut dire qu'elle sera valide pendant toute la durée de l'activité du contexte "Attack". La norme "N1" est également associée à une autre norme identifiée "N10" par un lien de sanction. Cette norme est valide à condition que la fonction *violated()* retourne "true" quand la norme "N01" est passée en paramètre. En clair, si la mission de marquer un but n'est pas accomplie, la mission de ne pas prendre de but doit être accomplie.

8.2.3 Package moise.oe

Toutes les classes que nous venons de voir dans le package `moise.os` permettent de créer des objets spécifiant une Organisation. De cet ensemble d'objets, nous pouvons en déduire une ou plusieurs instanciations représentant une Organisation. Voyons comment elle se traduit en terme d'API Java.

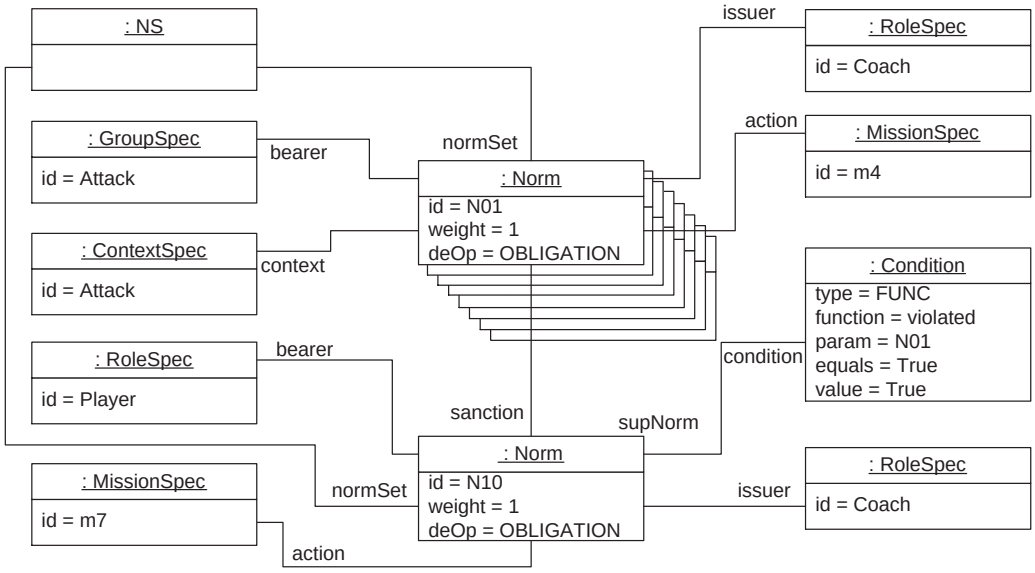


FIG. 8.13 – Diagramme d'instances de la NS pour l'exemple des "Robots Footballeurs"

Comme illustré sur le diagramme de classes UML de la FIG. 8.14 le package `moise.oe` est composé des classes `OE` et `Member` (cf. la description des composantes d'une Organisation dans la Section 7.1) ainsi que des sous-packages `moise.oe.structure`, `moise.oe.functioning`, `moise.oe.contexts` et `moise.oe.norms` dont seules les classes principales ont été mises sur le diagramme afin d'en faciliter la lecture. Chaque sous-package est détaillé dans les sections suivantes. Pour chaque classe, nous verrons le lien qu'elle a avec la classe de sa spécification, les autres classes du package avec lesquelles elle est associée et enfin ses différentes fonctions.

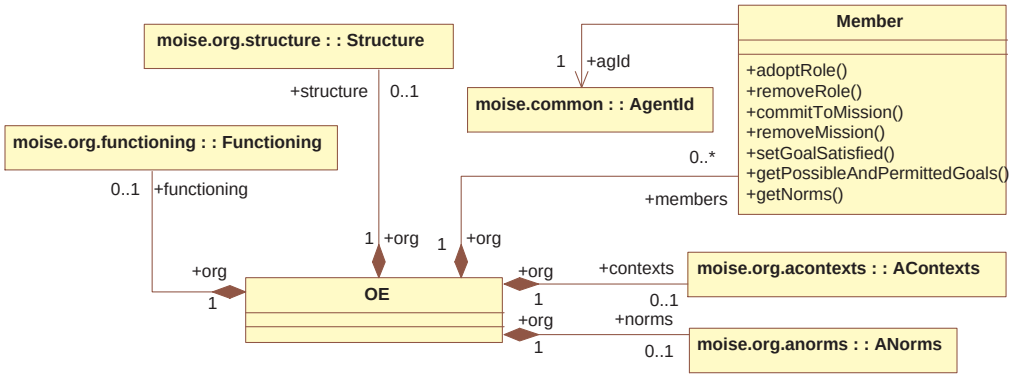


FIG. 8.14 – Diagramme de classes du package `moise.oe`

– La classe `OE` représente une instance d'organisation.
  ▷ Elle fait référence à la spécification d'une organisation grâce au lien avec la classe `OS` du `package moise.os`.
  ▷ Pour être dynamique, elle est associée à la classe `Member` pouvant être instanciée en objet entre 0 et n fois. Une Organisation est également définie par les liens de composition avec les classes `Structure`, `Functioning`, `AContexts` et `ANorms` des sous-packages `moise.oe.structure`,

`moise.oe.functioning`, `moise.oe.acontexts` et `moise.oe.anorms`.

– La classe `Member` est la représentation d'un agent au sein de l'Organisation afin de pouvoir l'interpréter et intéragir avec.
  ▷ Cette classe décrit un agent dans l'organisation, de ce fait elle est liée à la classe d'identification d'agent `AgentId` du package `moise.common`.
  ▷ Pour décrire les actions d'un agent au sein de l'organisation, la classe possède les méthodes suivantes :
    ○ *adoptRole()* permettant l'adoption de rôles se traduisant par la création d'un objet de type `Role` ;
    ○ *removeRole()* permettant l'abandon d'un rôle (destruction de l'objet de type `Role` correspondant) ;
    ○ *commitToMission()* permettant l'engagement sur une mission (création d'un objet de type `Mission`) ;
    ○ *removeMission()* permettant l'abandon de l'accomplissement d'une mission engagée (destruction d'un objet de type `Mission`) ;
    ○ *setGoalSatisfied()* permettant le changement d'état d'une instance de but ;
    ○ *getPossibleAndPermittedGoals()* permettant de connaître tous les buts appartenant aux missions sur lesquelles l'agent est engagé possibles à atteindre (cf. les instances de buts dans la description du package `moise.oe.functioning` ci-après) ;
    ○ *getNorms()* permettant de connaître toutes les normes contraignant le fonctionnement de l'agent en fonction des rôles qu'il est en train de jouer.

Package moise.oe.structure

Comme illustré sur le diagramme de classes de la Fig. 8.15, le package `moise.oe.structure` est composé des classes `Structure`, `Group` et `Role`.

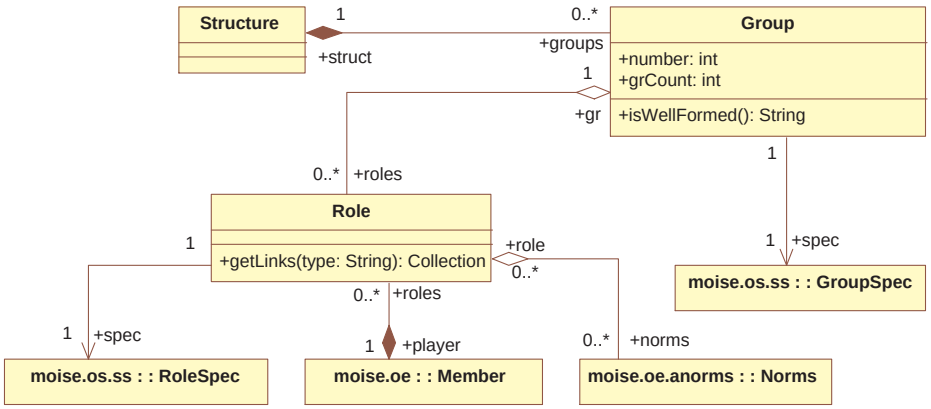


FIG. 8.15 – Diagramme de classes du package `moise.oe.structure`

Elle est associée à la classe `Role` définissant ainsi l'ensemble des rôles qu'un agent joue au sein de l'organisation ainsi que de la classe `Mission` définissant l'ensemble des missions sur lesquelles l'agent s'engage.

– La classe `Group` représente l'instanciation d'un groupe.
  ▷ Elle fait référence à sa spécification grâce au lien avec la classe `GroupSpec`.
  ▷ Elle est associée à elle-même afin de représenter les instances de ses sous-groupes ainsi que l'instance de son éventuel super-groupe. Elle est également associée à la classe `Role`.
  ▷ Ses attributs permettent de connaître l'identifiant unique de l'objet (attribut *number*) ainsi que le nombre d'instances de groupes créées (attribut statique *grCount*).

▷ Sa méthode *isWellFormed()* permet de savoir si le groupe respecte sa spécification en terme de nombre d'instances de rôles créées appartenant à cette instance de groupe ainsi qu'aux instances de ses sous-groupes.

– La classe `Role` représente l'instance de rôle joué par un agent dans l'organisation.
  ▷ Pour cela elle fait référence à sa spécification via la classe `RoleSpec`.
  ▷ Afin de définir qu'un rôle est joué par un agent au sein de l'organisation la classe `RoleSpec` est associée à la classe `Member`, à l'instance de groupe à laquelle elle appartient ainsi qu'à un ensemble d'instance de normes relatives à cette instance de rôle. Nous entrerons dans les détails de ce lien lors de la description du package `moise.oe.anorms`. Il n'existe pas de lien avec le package `moise.oe.functioning` pour définir qu'un rôle s'engage sur une mission. En effet, ce sont les agents qui jouent un ou plusieurs rôles et c'est également les agents qui s'engagent sur les missions. Le lien se fait donc via la classe `Member` qui est composée d'un ensemble d'instances de rôles et d'un ensemble d'instances de missions.
  ▷ Sa méthode *getLinks()* permet de connaître tous les liens concernant sa spécification de rôle et de savoir ainsi avec qui l'agent peut communiquer, quel agent il peut voir, sur quel agent il peut avoir autorité et quel autre rôle il peut jouer (cf. description des liens en Section 6.2.3).

Package moise.oe.functioning

Comme illustré sur le diagramme de classes de la Fig. 8.16, le package `moise.oe.functioning` est composé des classes `Functioning`, `Scheme`, `GoalInstance`, `Plan` et `Mission`.

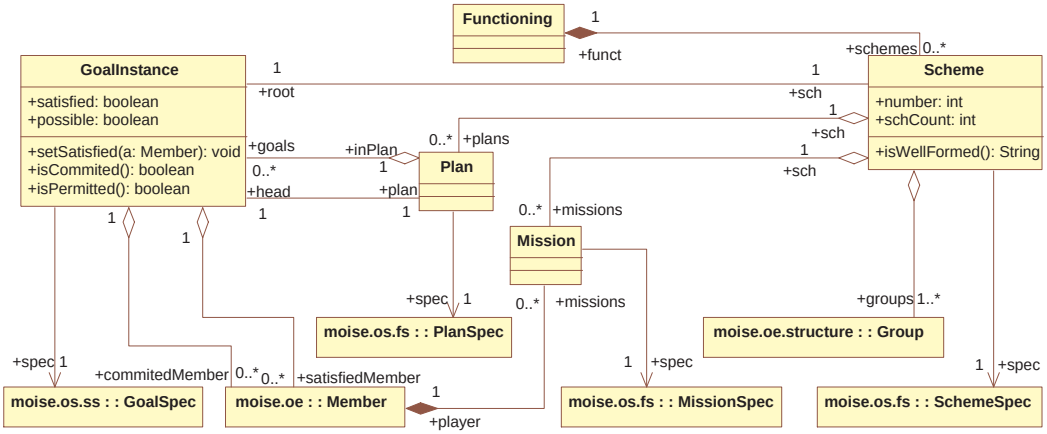


FIG. 8.16 – Diagramme de classes du package `moise.oe.functioning`

– La classe `Scheme` représente l'instanciation d'un schéma d'exécution.
  ▷ Elle fait référence à sa spécification via la classe `SchemeSpec` du package `moise.os.fs`.
  ▷ Elle est associée à un ensemble d'instances de plans (classe `Plan`), à un ensemble d'instances de missions (classe `Mission`), à un ensemble d'instances de groupes responsables du schéma (classe `Group`) et à une instance de but (classe `GoalInstance`) étant le but racine à satisfaire du schéma.
  ▷ Ses attributs lui permettent de définir un identifiant unique (*number*) ainsi qu'un compteur d'instance de schéma (*schCount*).
  ▷ Comme pour la classe `Group`, la méthode *isWellFormed()* vérifie que le schéma satisfait la spécification en terme de nombre d'agents engagés sur ses missions.

– La classe `GoalInstance` représente l'instance du but que les agents devront atteindre.
  ▷ Elle fait référence au schéma auquel le but appartient ainsi qu'à la classe `GoalSpec` pour sa

spécification.
  - ▷ Elle est associée à la classe `Member` afin de définir les membres de l'organisation qui sont engagés sur la mission possédant ce but et ceux qui ont satisfait ce but. Cette classe est liée à la classe `Plan` de deux manières différentes pour définir le plan auquel le but appartient et le plan qu'il est nécessaire d'exécuter pour atteindre le but en question.
  - ▷ Pour cela deux attributs permettent de savoir si le but est atteint (attribut *satisfied*) ou s'il est impossible à atteindre (attribut *possible* = false).
  - ▷ La méthode *setSatisfied()* permet de satisfaire ce but en faisant passer le membre qui est à l'origine de l'ensemble des membres engagés à l'ensemble des membres qui ont satisfait le but. Les méthodes *isPermitted()* et *isCommittable()* renvoient la valeur "true" si le but est permis, c'est-à-dire que dans le schéma, son but précédent est atteint pour la première méthode et si le but n'est pas déjà satisfait, est permis, est possible, n'a pas de super-buts déjà satisfaits et n'a pas de super-buts impossibles et que ces sous-buts sont tous satisfaits dans le cas d'un plan séquentiel ou parallèle ou que un de ces sous-buts est satisfait dans le cas d'un plan de choix pour la deuxième méthode.

– Le classe `Mission` représente l'engagement d'un agent sur une mission. Elle est donc liée aux classes `MissionSpec` pour la référence à sa spécification et associée à la classe `Member` pour la référence à l'agent. Elle fait également référence à l'instance de schéma a laquelle elle appartient.

– La classe `Plan` représente une instance de plan à exécuter pour atteindre une instance de but. Elle est associée à la classe `GoalInstance` pour la définition de l'ensemble des buts à atteindre et est liée à cette même classe comme référence au but racine du plan en question. Elle est également associée au schéma auquel elle appartient et enfin elle fait référence à sa spécification (classe `PlanSpec`) provenant du package `moise.os.fs`.

Package moise.oe.acontexts

Comme illustré sur le diagramme de classes de la Fig. 8.17, le package `moise.oe.acontexts` est composé de la seule classe `AContexts`.

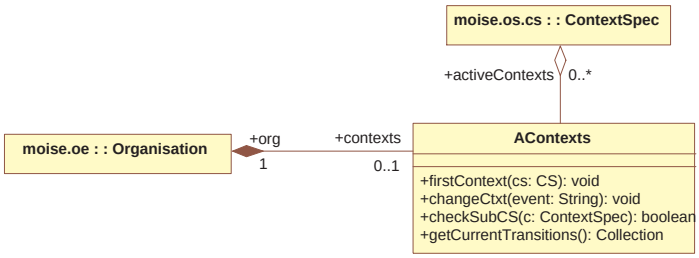


FIG. 8.17 – Diagramme de classes du package `moise.oe.acontexts`

– La classe `AContexts` sert d'interface de gestion de la dynamicité des contextes.
  - ▷ Un contexte n'étant pas instancié, nous avons seulement un ensemble de contextes actifs au sein de l'Organisation. Cela se traduit par l'association avec la classe `ContextSpec` et avec la classe `Organisation`.
  - ▷ La gestion des contextes se fait grâce aux méthodes suivantes :
    - ○ *firstContext()* consistant à initialiser l'ensemble des contextes actifs avec le contexte initial défini dans la CS ;
    - ○ *changeCtxt()* permettant d'exécuter une transition qui serait définie par un événement donné et dont le contexte source serait un contexte appartenant aux contextes actifs ;
    - ○ *checkSubCS()* consiste à vérifier que tous les sous-CS d'un contexte particulier sont dans un état final ;

- *getCurrentTransitions()* permet de connaître toutes les transitions qui peuvent avoir lieu en fonction de l'ensemble des contextes actifs.

Package moise.oe.anorms

Comme illustré sur le diagramme de classes de la Fig. 8.18, le package `moise.oe.anorms` est composé des classes `ANorms` et `Norm`.

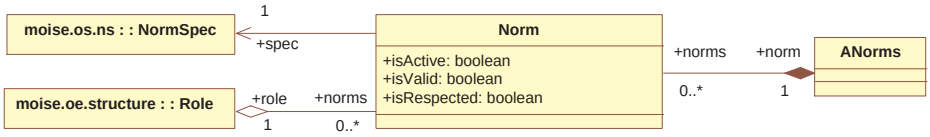


FIG. 8.18 – Diagramme de classes du package `moise.oe.anorms`

– La classe `Norm` représente une instance de norme pour une instance de rôle en particulier.
  ▷ Elle fait référence à sa spécification grâce au lien avec la classe `NormSpec`.
  ▷ Afin d'avoir une instance de norme définie pour un agent jouant un rôle, une association lie cette classe avec la classe `Role`.
  ▷ La classe `Norm` possède trois attributs de type booléen permettant de savoir si la norme est active (attribut *isActive*), si elle est valide (attribut *isValid*) et si elle est respectée (attribut *isRespected*).

Nous venons de voir dans cette section l'API permettant de spécifier et d'instancier une Organisation de type $\mathcal{M}$OISE^{Inst} en JAVA. Une partie des actions que nous avons vues dans la Section 7.2 peut être appelée par l'intermédiaire des objets de type `Member` afin d'agir sur l'Organisation. Les agents faisant partie de l'Organisation peuvent appeler ces méthodes relatives aux rôles, aux missions et à la satisfaction de buts. Notre but est de contrôler ces actions et de faire en sorte que les objets issus du package `moise.oe` respectent bien leur spécification. De ce fait, un agent ne doit pas pouvoir agir directement sur son objet `Member` sans qu'il soit supervisé. Il devra pour cela commiquer tout d'abord avec les agents de la couche $\mathcal{S}$YNAI. Nous allons maintenant décrire l'API JAVA de $\mathcal{S}$YNAI.

8.3 Mise en œuvre des agents de $\mathcal{S}$ynai

Nous allons voir dans cette section les agents qui vont composer la couche $\mathcal{S}$YNAI de notre modèle d'Institution Électronique. Nous allons commencer par voir globalement leur architecture, puis leur mécanisme de traitement des informations plus en détail et enfin nous décrirons l'API de $\mathcal{S}$YNAI ainsi que le cycle de vie des agents de supervision.

8.3.1 Architecture des Superviseurs

Les Superviseurs appartiennent à la couche $\mathcal{S}$YNAI modélisée selon une organisation $\mathcal{M}$OISE^{Inst}. Ils doivent donc pouvoir interpréter et également agir sur cette organisation. Pour cela, et comme vu dans la section précédente, ils sont représentés par un objet de type `Member`. En plus de savoir s'organiser entre eux, ces agents institutionnels doivent pouvoir traiter des demandes d'action provenant des agents du domaine en vérifiant qu'elles respectent bien leurs pré-conditions (cf. Section 7.2).

La FIG. 8.19 représente les différents composants que les Superviseurs possèdent. Tout d'abord et afin de pouvoir faire en sorte que les actions des agents du domaine soient cohérentes avec la spécification de l'Organisation, il faut que les Superviseurs possèdent une base de connaissances concernant les pré-conditions, sur l'état de l'organisation et sur sa spécification. Ce sont les deux

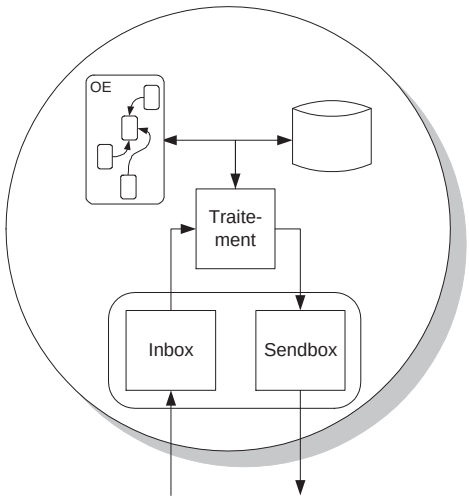


FIG. 8.19 – Représentation graphique de l'implémentation des agents de SYNAI

composants sur lesquels va se baser le traitement des demandes d'actions. Ces demandes sont reçues par un message et la réponse est envoyée sous la même forme. La succession des messages (les protocoles abordés dans la Section 7.5) font également partie de la base de connaissance de chaque agent. De ce fait, les protocoles de communication ne sont pas communs à tous les agents mais partagés entre eux au sein de l'OS via une spécification des interactions par exemple. C'est cependant une perspective intéressante (cf. Chapitre 11).

Les agents de supervision ainsi que les agents du domaine doivent évoluer et communiquer au sein d'une plate-forme d'exécution multi-agents. Nous avons choisi SACI dont nous allons voir rapidement l'API avant de voir celle de la couche SYNAI.

8.3.2 Package saci

Nous utilisons la plate-forme d'exécution à agents SACI [56] (Simple Agent Communication Infrastructure) afin de faire communiquer les agents. L'envoi des messages est transparent grâce à l'utilisation d'un *Facilitator* servant de service de pages blanches, de service de pages jaunes et de gestionnaire de la société. Au sein de SACI, les messages sont construits avec KQML.

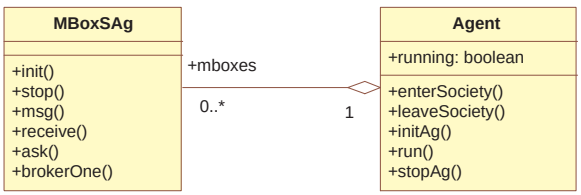


FIG. 8.20 – Diagramme de classes du package saci

Sur FIG. 8.20, nous représentons les deux classes principales sur lesquelles nous allons nous baser pour définir nos agents. Le package SACI fournit entre autre les classes `MBoxSAg` et `Agent`. Les agents sont mis en œuvre par spécialisation de la classe `Agent`. Les méthodes de cette classe permettent entre autre aux agents de s'initialiser (méthode *initAg()*), d'entrer dans une société SACI (méthode *enterSociety()*) ainsi que d'en sortir (méthode *leaveSociety()*), et d'exécuter leur cycle de vie (méthode

run()). Cette dernière méthode doit bien entendu être surchargée afin d'obtenir le fonctionnement des agents que nous souhaitons. Cette classe utilise la classe `MBoxSAg` pour envoyer et recevoir des messages grâce aux deux liens qui les associent. Cette classe fournit en effet une file de messages aux agents. Les méthodes de la classe `MBoxAg` permettent de manipuler des messages en émission ou en réception.

Ces classes vont nous servir de base pour la création d'agents capables de s'exécuter et de communiquer sur une plate-forme SACI en exhibant également des capacités à s'adapter et à superviser des organisations de type $\mathcal{M}$OISE^{Inst}. C'est ce que nous allons voir dans la section suivante avec la description de l'API JAVA du package $\mathcal{S}$YNAI.

8.3.3 Package synai

Comme illustré sur le diagramme de classes UML de la FIG. 8.21 le package `synai` est composé des classes `OrAgent`, `Supervisor` et `OrgWrapper`.

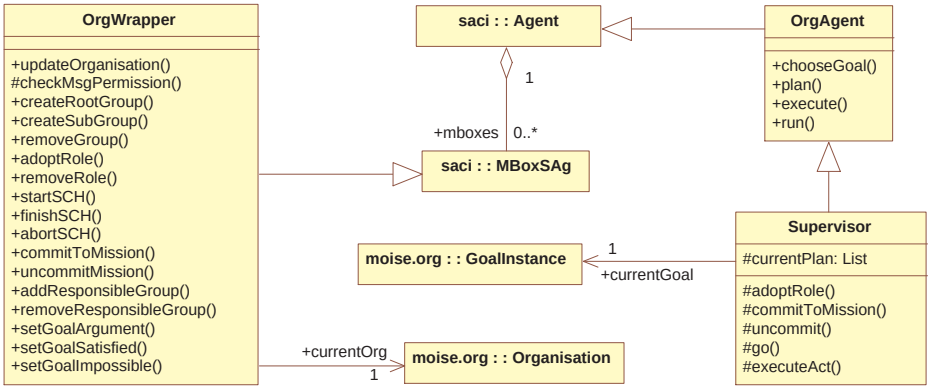


FIG. 8.21 – Diagramme de classes du package `synai`

- Afin d'obtenir des agents institutionnels capables de prendre en compte une organisation, nous faisons hériter de la classe `Agent` de `saci` la classe `OrgAgent` à laquelle nous ajoutons des méthodes relatives à son activité au sein d'une Organisation de type $\mathcal{M}$OISE^{Inst} :
  - *chooseGoal()* permettant à l'agent de choisir le but qu'il va tenter d'atteindre dans l'Organisation ;
  - *plan()* permettant à l'agent de découper un but en sous-buts ;
  - *execute()* permettant à l'agent de mettre en place la ou les actions lui permettant d'atteindre un but.

  Ces trois méthodes sont écrites pour définir le comportement générique d'un agent évoluant dans une organisation $\mathcal{M}$OISE^{Inst}. Elles doivent donc être surchargées afin d'obtenir un comportement spécifique.

- La classe `Supervisor` hérite de la classe `OrgAgent`. Elle représente l'agent de base qui constitue la couche $\mathcal{S}$YNAI. Cette classe est associée à la classe `GoalInstance` afin de définir le but courant que le Superviseur doit atteindre. Il possède un attribut listant l'ensemble des identifiants des sous-buts qu'il faut atteindre pour satisfaire le but courant. En plus des méthodes héritées de la classe `OrgAgent`, la classe `Supervisor` exhibe les méthodes suivantes :
  - *adoptRole()* permettant à l'agent d'envoyer une requête d'adoption de rôle,
  - *commitToMission()* permettant à l'agent d'envoyer une requête d'engagement sur une mission,
  - *uncommit()* permettant à l'agent d'envoyer une requête de désengagement d'une mission,
  - *executeAct()* permettant à l'agent d'exécuter une tâche d'un plan d'action.

Ces méthodes sont utilisées par les méthodes héritées et surchargées afin de définir le comportement des agents lors de l'exécution de leur cycle de vie. Le choix d'un but se fera après s'être engagé sur une mission (donnant ainsi à l'agent un ensemble de buts à atteindre). Une fois le plan du but à atteindre défini, la méthode *executeAct()* met en place les actions permettant la satisfaction des sous-buts. Nous allons voir ceci plus en détail dans la suite de cette section.

– La classe `OrgWrapper` hérite de la classe `MBoxSAg` du package `saci`. Ainsi, l'*OrgWrapperAg* n'est pas un agent à proprement parler mais plutôt un service offert à l'agent du domaine. Il contient une copie de l'Organisation (association avec la classe `Organisation` du package `moise.oe`) qui permet l'accès au représentant de l'agent au sein de l'Organisation (classe `Member`). La classe `OrgWrapper` permet donc aux Superviseurs d'envoyer et de recevoir des messages mais également d'agir sur l'Organisation grâce aux méthodes qu'elle exhibe.

Initialisation des agents de $\mathcal{S}$ynai

Un fois qu'un agent a été créé et avant qu'il ne fonctionne au sein d'une société SACI et d'une organisation $\mathcal{M}$OISE^{Inst}, il faut l'initialiser. Nous considérons ici qu'une société SACI construite sur une organisation $\mathcal{M}$OISE^{Inst} existe déjà et que lors de sa création l'agent rejoint cette société. Lors de son initialisation, un agent va chercher à adopter un rôle afin de pouvoir agir dans l'organisation. En tant qu'agent de $\mathcal{S}$YNAI, il va adopter un des rôles de supervision du groupe Institution.

Nous ne définissons ici qu'un seul agent ayant toutes les facultés de supervision c'est pourquoi il n'existe qu'une seule classe. Nous aurons ainsi plusieurs Superviseurs tous capables de jouer n'importe quel rôle de supervision. C'est lors de l'instanciation de $\mathcal{S}$YNAI que ces agents sont créés et nous les initialiserons en leur faisant adopter chacun un rôle de supervision grâce à l'appel de la méthode *adoptRole()*. Les Superviseurs sont alors prêts à s'exécuter en mettant en œuvre de manière itérative leur cycle de vie.

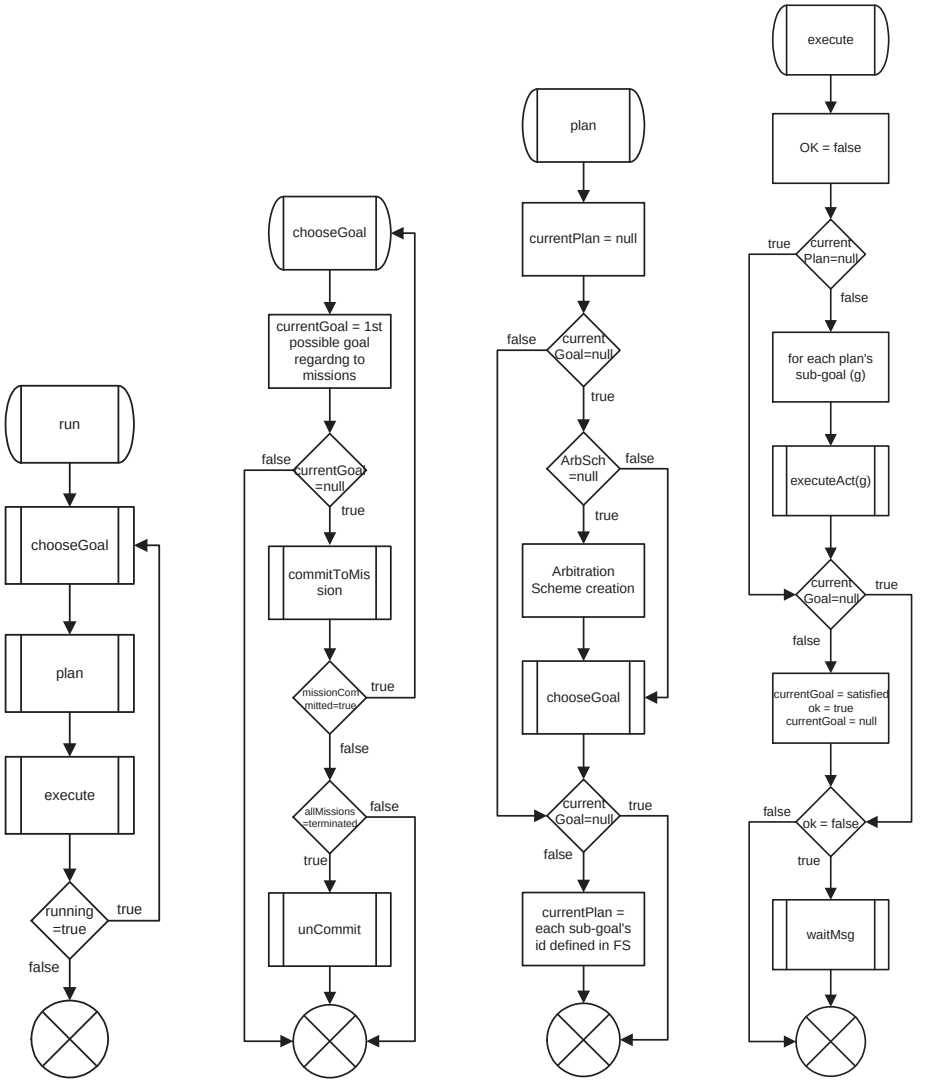
Cycle de vie des agents de $\mathcal{S}$ynai

Après avoir été initialisés, les superviseurs sont démarrés grâce à la méthode *run()* représentant l'exécution du cycle de vie des agents. Cette méthode consiste pour un superviseur à choisir un but possible et autorisé par l'organisation (*chooseGoal()*), de déduire le plan du but choisi (*plan()*) et enfin d'en exécuter toutes les actions (*execute()*). Cet enchaînement est illustré sur la FIG. 8.22(a). Cet algorithme étant la base du cycle de vie de l'agent, une fois le plan exécuté, il choisit un autre but et va faire en sorte de l'atteindre, et ainsi de suite.

La FIG. 8.22(b) illustre la façon dont l'agent choisit son but. Le principe est de définir un but courant qu'il est autorisé à atteindre en fonction du rôle qu'il joue et des missions sur lesquelles il est engagé. Si un but courant est effectivement défini, alors il va pouvoir ensuite en déduire son plan. Sinon, il faut qu'il s'engage sur une mission autorisée afin d'avoir effectivement des buts à atteindre.

La façon de définir le plan d'un but est illustrée la FIG. 8.22(c). Afin d'atteindre le but courant qu'il vient de choisir, l'agent va créer un plan composé de tous les identifiants des sous-buts de son but courant (il va devoir accéder à la FS pour cela). Par contre, si son but courant n'est pas encore choisi, et qu'il n'existe aucune instance du schéma d'arbitrage alors il en crée une afin d'y choisir un but à atteindre.

Enfin, une fois le schéma créé, le but choisi et son plan défini, il faut exécuter ce dernier grâce à *execute()*. La FIG. 8.22(d) illustre cette méthode. Si l'on considère que l'agent a son but courant défini, alors pour chaque but composant le plan, il va exécuter la méthode *executeAct()* lui permettant de mettre en place une action pour satisfaire le but. Il est à noter que les actions dont nous parlons ici sont propres aux agents et ne correspondent pas aux actions de l'Organisation décrites dans la section 7.2. Ensuite, s'il existe toujours un but courant, il est renseigné comme satisfait et il n'y a



(a) Exécution (b) Choix d'un but (c) Plan du but (d) Exécution du plan

FIG. 8.22 – Organigrammes du cycle de vie des Superviseurs

plus de but courant. L'agent va pouvoir tenter d'atteindre un autre but.

Le but d'un superviseur est de répondre aux requêtes des agents. Ainsi, un des buts qu'il doit atteindre est la détection d'une violation. Suivant la requête qu'il va recevoir et suivant son rôle au sein de l'Organisation, l'exécution de chaque but du plan va se dérouler différemment. En fait, il va exécuter une tâche afin d'atteindre un but. C'est ce que nous détaillons maintenant.

Mécanisme d'exécution des tâches des Superviseurs

Dans la Section 7.2 du chapitre précédent, nous avons détaillé les actions qu'un agent appartenant à l'Organisation était susceptible de mettre en œuvre s'il respectait certaines conditions. Cette vérification des pré-conditions est faite par les Superviseurs (cf. Section 7.3) lors d'une phase de tests. Les tests sont bien sûr différents selon l'action qui va être exécutée au sein de l'Organisation. Ainsi, selon le rôle joué par le superviseur et selon l'*événement relatif à une demande d'action sur l'Organisation* reçu, il mettra en place les tests adéquats. Si les tests sont passés avec succès, alors l'action

est exécutée par l'objet `Member` de l'agent faisant la requête, sinon, le Superviseur prévient l'agent *InstManagerAg* qu'une violation a lieu et l'agent *FunctManagerAg* qu'il a atteint son but courant.

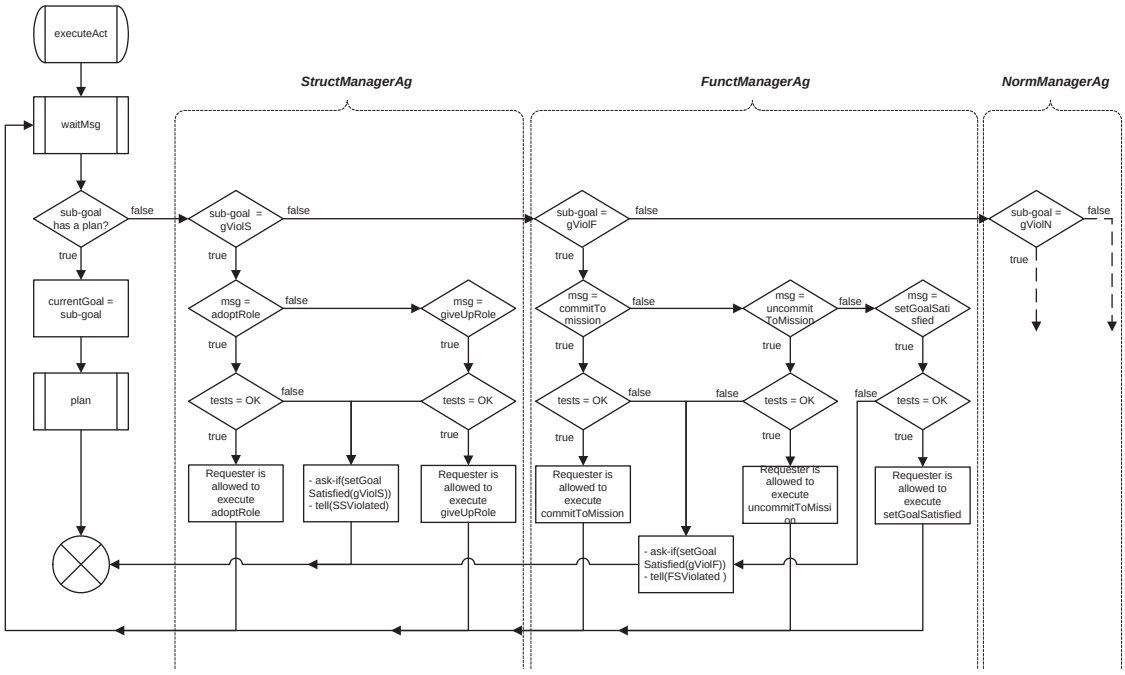


FIG. 8.23 – Organigramme de la méthode d'exécution des tâches

La méthode *executeAct()* que nous illustrons par l'organigramme de la FIG. 8.23 associe un identifiant de but à une tâche concrète de l'agent lui permettant d'atteindre le but. Cependant, si le sous-but à atteindre a lui aussi un plan, il va falloir atteindre chacun de ses sous-buts. S'il n'a pas de sous-buts, alors l'exécution de la tâche correspondante va pouvoir commencer. Suivant le but à atteindre (déduit du rôle que l'agent joue), sa tâche se traduit par le traitement d'un événement que l'agent aura reçu d'un agent du domaine demandant l'exécution d'une action dans l'Organisation. En fonction de l'action, une batterie de tests est effectuée. Si le résultat est positif, le superviseur autorise l'exécution de l'action d'une part et n'ayant pas atteint sont but, il continue à exécuter sa tâche courante. Nous retournons dans ce cas en mode de réception de message afin de recevoir une nouvelle demande d'un agent du domaine.

A l'inverse, si les tests sont négatifs, alors il y a violation et de ce fait, la tâche de l'agent l'ayant détectée est exécutée et son but satisfait. Il en informe donc l'agent *FunctManagerAg* afin de mettre à jour l'état de son but (ou sous-but) courant et prévient également le *InstManagerAg* qu'une violation a eu lieu. Ce dernier s'occupera de mettre en œuvre les mécanismes de correction et de sanction. A ce moment là, l'exécution de la tâche est terminée et l'agent peut continuer son cycle de vie. Nous venons d'illustrer le fait que c'est également dans cette méthode que les protocoles abordés dans la Section 7.5 sont mis en œuvre.

Nous avons représenté ici que les détections de violations se déroulant lors d'une demande de la part d'un agent du domaine. En ce qui concerne l'agent *NormManagerAg*, il doit également détecter les violations de normes ne faisant pas forcément l'objet d'une demande du *FunctManagerAg* avec l'envoi d'un événement "checkNorms". Nous consacrons la section suivante à cela.

Mécanisme de détection des violations des normes

Afin de vérifier de façon pro-active qu'une norme est respectée, l'agent *NormManagerAg* doit se baser sur plusieurs éléments de sa spécification (cf. section 6.5) :

1. La mission car c'est ce sur quoi porte vraiment la norme : une norme spécifie si la mission doit être atteinte ou non.
2. Les contraintes temporelles : une norme va dire si la mission doit être atteinte ou non, et quand.
3. Les conditions de validité : une norme va dire si la mission doit être atteinte ou non, quand et sous quelles conditions.
4. Les conditions d'activation : une norme va dire si la mission doit être atteinte ou non, quand, sous quelles conditions et dans quel contexte.

De plus, la spécification des normes est basée sur la logique déontique. Grâce à un opérateur déontique, nous spécifions si la norme exprime une obligation, une permission ou une interdiction. Nous avons également vu dans la section 8.2.3 que l'instance d'une norme, c'est à dire une norme prête à être utilisée dans une instance d'organisation, possédait trois variables booléennes informant sur l'activation de l'instance de norme, sur la validité et sur son respect. Les valeurs de ces variables vont définir à quel moment de leur cycle de vie les normes se trouvent (cf. Section 6.5.3). Ces valeurs vont être modifiées par l'agent *NormManagerAg* au cours de son propre cycle de vie.

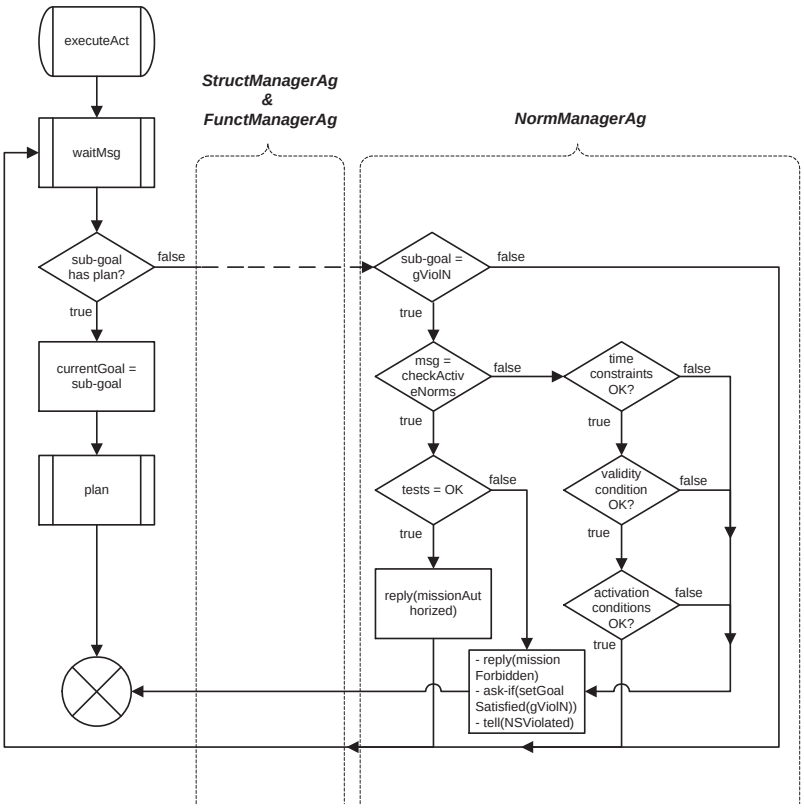


FIG. 8.24 – Organigramme de la méthode d'exécution des tâches pour un agent NormManager

La FIG. 8.24 illustre la méthode *executeAct()* du point de vue de l'agent *NormManagerAg*. On y retrouve d'une part la phase de tests suite à une demande explicite et également, la phase de test pro-active se basant sur un éventuel changement de chacune des contraintes composant une norme. Ainsi soit l'agent reçoit un message contenant l'événement "checkNorms" et dans ce cas il vérifie si la

mission est autorisée par l'agent demandant, soit il n'y a pas de message et il vérifiera tour à tour et pour chaque instance de norme le respect des contraintes temporelles, des conditions de validité et des conditions d'activation selon l'opérateur déontique de la norme. Si une contrainte n'est pas conforme à la spécification normative, alors une violation est détectée et envoyée à l'agent *InstManagerAg*.

Nous venons de passer en revue l'API du modèle $\mathcal{M}$OISE^{Inst} capable d'implémenter un ensemble d'objets représentant une spécification de modèle d'organisation qui lui même une fois instancié à l'aide d'un ensemble d'objets issus du package `moise.oe` de l'API $\mathcal{M}$OISE^{Inst} peut être manipulé par un ensemble d'agents. Nous avons également décrit l'API de la couche $\mathcal{S}$YNAI capable de contrôler la manipulation de l'organisation par des agents du domaine. Nous allons maintenant mettre en pratique ces API en construisant un outil nommé SimOE permettant de tester le bon fonctionnement de l'Organisation et sa supervision.

8.4 Outil de Test de Spécification d'Institution

SimOE permet de tester l'exécution d'une organisation $\mathcal{M}$OISE^{Inst} au travers une Institution de type MA$\mathcal{B}_{\text{ELI}}$. C'est à dire que nous allons pouvoir lire un modèle d'organisation construit selon le principe des quatre spécifications, le visualiser, l'instancier en agissant sur l'organisation via la création d'agents et enfin apprécier l'arbitrage automatique de $\mathcal{S}$YNAI. L'organisation que nous donnons en entrée du simulateur est sous la forme d'un fichier XML. Ce fichier XML est pour le moment créé à la main, sans outil d'aide à la création d'organisation de type $\mathcal{M}$OISE^{Inst}. Une des perspectives d'évolution de cet outil serait de pouvoir créer directement et graphiquement son organisation avant de pouvoir la simuler.

Dès le lancement de l'exécutable, un fichier XML est demandé et si le fichier est bien formé et est conforme à la DTD de $\mathcal{M}$OISE^{Inst}, alors une fenêtre reprenant toutes les informations de l'organisation s'affiche (cf. FIG. 8.25). A partir de ce moment, deux vues sont possibles, la vue passive de l'OS et la vue active de l'OE que l'on peut sélectionner grâce aux menus représentés ci-contre. Détaillons chacune d'entre elles.

8.4.1 Représentation de la spécification

La visualisation de l'OS permet de faire une vérification des informations sur l'OS spécifié par l'utilisateur dans le fichier XML qu'il a donné à lire au simulateur (cf. FIG. 8.25). Sur la partie gauche de la fenêtre un arbre structuré suivant les quatre spécifications donne tous les éléments de l'OS. Les feuilles de cet arbre sont les rôles, les groupes, les buts, les missions, les contextes et les normes. En sélectionnant une de ces feuilles le détail de l'élément choisi s'affiche dans le fenêtre de droite. L'affichage se fait en transformant du XML généré par chaque objet de l'API en HTML via des fichiers de transformation XSL. Des liens hypertextes permettent de naviguer dans les descriptions et ainsi de visualiser le détail d'autres éléments de cette OS. Pour les rôles et les groupes, nous affichons les normes qui les concernent. La spécification organisationnelle n'est pour l'instant pas éditable.

La spécification de l'organisation de l'utilisateur sera prise en compte ainsi que celle fournie avec MA$\mathcal{B}_{\text{ELI}}$ spécifiant l'organisation de $\mathcal{S}$YNAI. L'OS affichée intégrera alors automatiquement les spécifications de $\mathcal{S}$YNAI suivant les règles énoncées dans la section 7.4. Nous trouverons donc dans la SS, le groupe Institution et les rôles de Supervision, dans la FS les schémas de détection, de correction, de sanction et d'arbitrage, dans la CS le contexte global composé de la CS de l'utilisateur et de la CS d'arbitrage (les différentes stratégies) et enfin dans la NS toutes les normes. Cette partie de la fenêtre permet de visualiser la partie fixe de l'application en matière d'organisation d'agents. Voyons maintenant comment apporter de la dynamicité à la simulation.

8.4.2 Instanciation de l'OS en Organisation

Cette partie permet de visualiser le résultat de l'action de l'utilisateur sur l'organisation via un ensemble d'agents. Nous avons pour cela dans la partie gauche le listing des agents, des groupes et

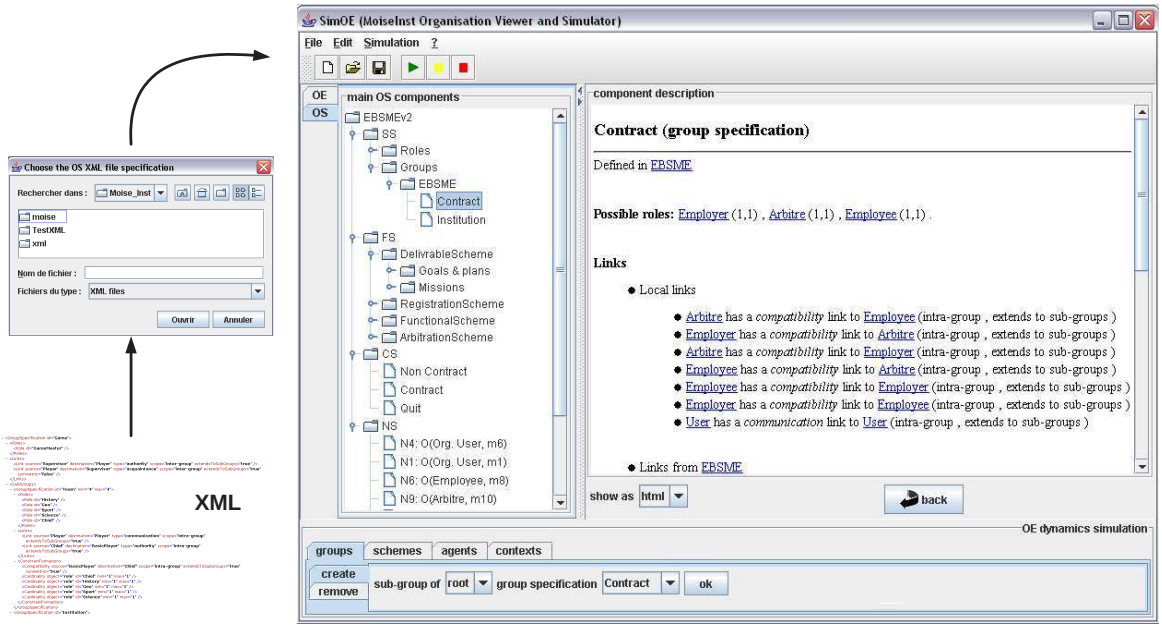


FIG. 8.25 – Spécification Organisationnelle (OS)

des schémas créés au sein de l'organisation. La partie de droite affiche les informations sur chacun des éléments ci-dessus. La partie inférieure de la fenêtre permet d'agir sur l'organisation. Les trois parties de la fenêtre sont visibles sur la FIG. 8.27.

FIG. 8.26 – Barre de contrôle de la simulation

L'instanciation de l'OS en Organisation se fait automatiquement à la lecture de la spécification de l'organisation. L'Organisation obtenue est cependant vide, il faut créer les groupes, démarrer les schémas et faire en sorte que les agents agissent. La commande "Start Simulation" du menu "Simulation", ou la barre de simulation (FIG. 8.26), permet la création automatique des agents de $\mathcal{S}$YNAI et leur initialisation au sein de l'organisation via la création de leur groupe, le démarrage de leurs schémas, l'adoption des rôles institutionnels et l'engagement sur les missions de supervision. En même temps, les premiers contextes de la CS deviennent actifs. Ceci étant fait, l'utilisateur peut visualiser l'Organisation (l'état des instances de norme par exemple) ou agir dessus. Nous listons ci-dessous l'ensemble des possibilités offertes à l'utilisateur vis à vis de ses instances d'organisation que ce soit en terme d'action sur l'organisation ou en terme de visualisation de l'état de l'organisation.

Actions sur l'organisation

- Création/Destruction des instances de groupe : FIG. 8.27.a).
- Création/Destruction des instances de schéma : FIG. 8.27.b).
- Création/Destruction des agents du domaine : FIG. 8.27.c).
- Adoption/Abandon de rôles par les agents du domaine : FIG. 8.27.d).
- Engagement/Désengagement des agents du domaine sur des missions : FIG. 8.27.e).
- Changement d'état des instances de buts : FIG. 8.27.g).
- Changement de contexte : FIG. 8.27.f).

Visualisation de l'organisation
- Affichage des normes valides et de leur statut (respectées ou non) selon les contextes.
- Affichage des normes valides et de leur statut selon les agents concernés.
- Affichage des normes valides et de leur statut selon les groupes concernés.

Ainsi nous pouvons tester qu'une société d'agents (dont les actions sont décidées par l'utilisateur) sera automatiquement supervisée et contrôlée par $\mathcal{S}$YNAI. La création d'un agent entraîne l'instanciation de trois objets : un agent de base capable d'envoyer des messages aux Superviseurs via son `OrgWrapper` afin d'agir sur l'organisation via son `Member`. L'utilisateur peut simuler la création d'événements provoquant le changement des contextes actifs. Une commande spéciale permet d'envoyer les événements adéquats provoquant le changement de stratégie d'arbitrage.

L'utilisateur décide des actions des agents qu'il aura créés. Il peut leur faire adopter des missions sur lesquelles ils ne sont pas autorisés. Cela lui permet ainsi de stimuler le fonctionnement des agents institutionnels et vérifier leur capacité à superviser. En effet ces derniers traitent la demande qui passe par eux et affichent leur résultat dans la partie de droite de la fenêtre d'application. Suivant la stratégie d'arbitrage choisie, si l'action demandée est interdite, elle sera faite puis défaite ou interdite directement. L'agent *NormManagerAg* indique également par un affichage de messages quand des normes sont détectées comme violées autrement que sur une demande du *FunctManagerAg*.

Du point de vue de l'outil de test de l'organisation normative, les commandes de pause et de stop sont inutiles et ne fonctionnent donc pas. L'enregistrement d'une organisation en cours de simulation ne fonctionne pas non plus. Contrairement aux contrôles de la simulation, cette fonction de sauvegarde pourrait s'avérer utile dans différentes applications. Nous allons d'ailleurs pouvoir le constater dans la troisième partie de ce rapport traitant de la validation par l'utilisation dans deux cas concrets de notre modèle d'Institution Électronique.

8.5 Synthèse

Dans les Chapitres 6 et 7, nous avons vu le modèle $\mathcal{M}$OISE^{Inst} et son utilisation par les agents de la couche $\mathcal{S}$YNAI. Dans ce chapitre, nous avons présenté une API JAVA permettant l'implémentation de modèles organisationnels. Ces modèles sont mis en œuvre à partir d'une description faite en XML. Nous avons également présenté l'API JAVA de $\mathcal{S}$YNAI permettant l'implémentation d'un ensemble d'agents de supervision capables d'agir au sein de la mise en œuvre d'un modèle organisationnel. Ils peuvent s'y organiser et agir afin d'atteindre leurs buts, communiquer entre eux et superviser les actions d'un ensemble d'agents du domaine eux aussi pouvant s'organiser et agir au sein de l'Organisation.

Comme nous l'avons précisé précédemment dans ce chapitre, tous les agents de type `Supervisor` possèdent un mécanisme de détection de violation. Pour les agents *StructManagerAg* et *FunctManagerAg*, cela consiste à effectuer les tests institutionnels correspondant aux actions (de type structurelle ou fonctionnelle) sur l'organisation faisant l'objet d'une requête de la part d'un autre agent. La requête est soit acceptée ou une violation est détectée. Le *NormManagerAg* quant à lui, en plus d'exécuter les tests institutionnels d'actions fonctionnelles, agit de façon pro-active afin de vérifier que les contraintes temporelles, de validité et d'activité soient toujours respectées pour chaque norme. Pour tous les superviseurs, la découverte d'une violation équivaut à la satisfaction de leur but principal, c'est-à-dire atteindre le sous-but du schéma d'arbitrage qui fera en sorte que le but principal de garder l'organisation cohérente soit atteint.

L'outil de test des organisations normatives SimOE permet de tester[1] le bon fonctionnement des organisations spécifiées avec le modèle $\mathcal{M}$OISE^{Inst} ainsi que le fonctionnement des agents du domaine et d'arbitrage. Une fois un arbitrage simulé à l'aide de cet outil SimOE, nous pouvons implémenter MA$\mathcal{B}_{\text{ELI}}$ directement dans l'application. Nous allons maintenant voir deux cas réels d'application

[1] selon une procédure de tests laissée libre de choix au développeur

provenant de deux domaines totalement différents néanmoins complémentaires. La complémentarité de ces applications permettra de valider notre modèle et d'illustrer sa capacité d'adaptation à des domaines variés et pour des configurations différentes.

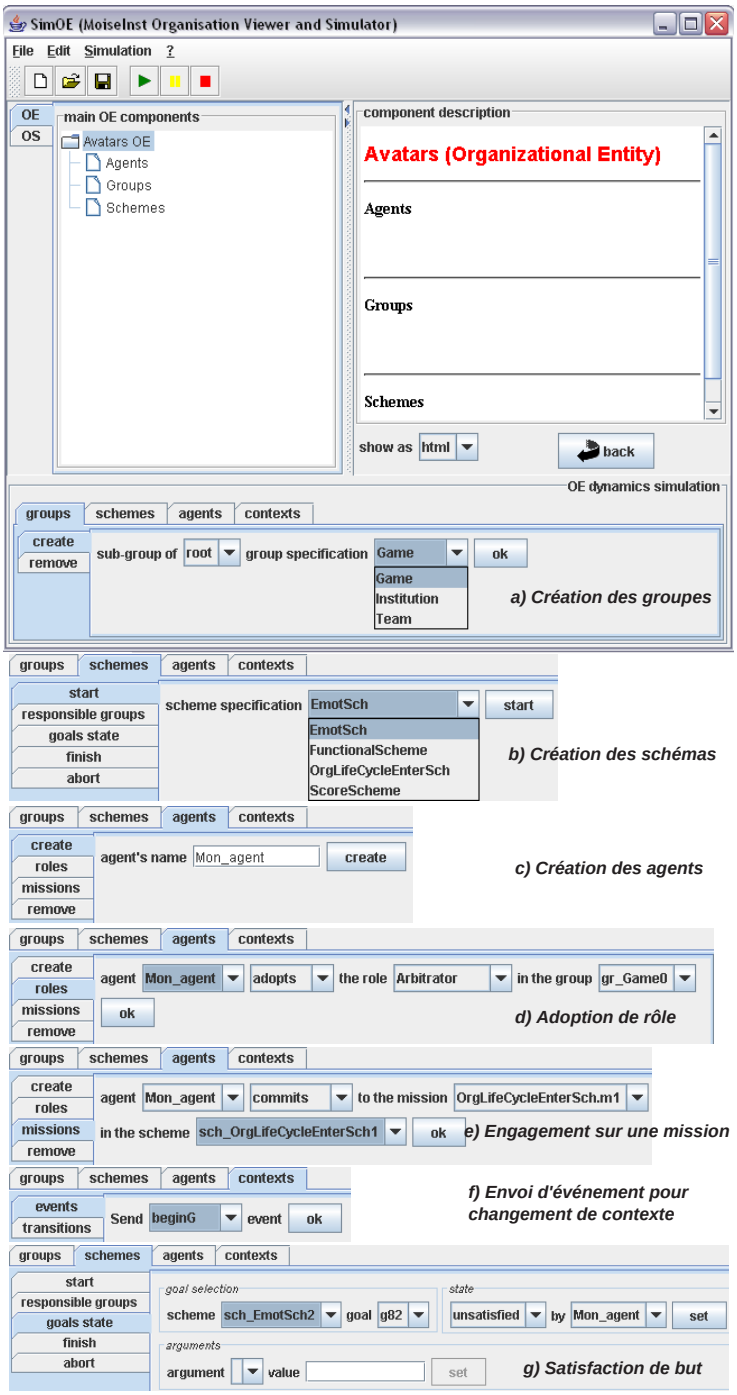


FIG. 8.27 – Simulation d'action sur l'Organisation

Troisième partie

Réalisation et Expérimentation

Nous avons présenté en détail dans la deuxième partie de ce mémoire un modèle d'Institution Électronique nommé MA$\mathcal{B}_{\text{ELI}}$, un langage de description de modèle d'Institution Électronique nommé MoiseInstML et un ensemble d'API Java permettant l'instanciation des modèles et leur contrôle avec un ensemble d'agents de supervision. Un outil de tests nous permet de vérifier d'une part que les spécifications d'organisation issus du modèle $\mathcal{M}$OISE^{Inst} sont exploitables et correctement interprétables et d'autre part que la couche $\mathcal{S}$YNAI fournie par l'Institution Électronique remplit bien ses fonctions en détectant les violations de spécification. Notre but est de montrer dans cette troisième partie l'application du modèle à deux domaines différents dont l'un concerne la création et l'exécution de contenu pour la télévision interactive et l'autre la gestion de contrats électroniques dans une plate-forme de e-commerce. Dans les deux cas, la même problématique d'arbitrage d'entités autonomes devant respecter des contraintes est importante. De plus, ces deux applications sont complémentaires par leur différence de portée de l'Institution.

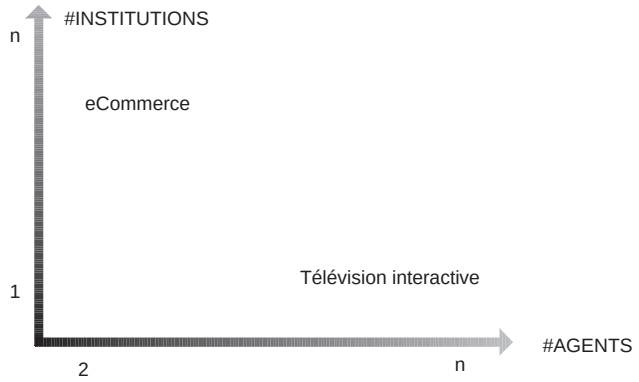


FIG. 8.28 – Portée en terme de nombre d'agents et d'institutions des applications de validation

Ainsi et comme illustré sur le schéma de la FIG. 8.28, nous allons montrer que MA$\mathcal{B}_{\text{ELI}}$ peut couvrir une grande partie de la portée qu'un modèle d'Institution Électronique peut avoir. Nous verrons ainsi comment dans le domaine de la télévision interactive, nous pouvons faire évoluer au sein d'une seule institution une équipe organisée de plusieurs agents. Nous verrons également qu'avec ce même modèle nous pouvons faire évoluer un faible nombre d'agents (deux) dans plusieurs institutions. Ces deux cas de figure fonctionnant, l'utilisation de ce modèle, pour organiser et contrôler une importante quantité d'agents au sein de plusieurs institutions, est possible et envisageable. Des tests de performance devront être cependant menés pour une application réelle.

Enfin cette troisième partie nous permettra d'illustrer comment un ensemble d'agents fournis par

MA$\mathcal{B}_{\text{ELI}}$ via $\mathcal{S}$YNAI peut superviser un ensemble de contraintes définies à l'aide de $\mathcal{M}$OISE^{Inst}. Il nous permettra également, grâce à la flexibilité du modèle, de voir comment le changement d'une simple contrainte (cardinalité de la SS, plan de la FS, norme de la NS) affectera le comportement des agents institutionnels. Ces agents institutionnels, qu'ils soient plongés dans un domaine de *e-contracting* ou de la télévision interactive arbitreront de la même façon, conséquence de la généricité que nous leur avons fournie.

Nous allons tout d'abord appliquer MA$\mathcal{B}_{\text{ELI}}$ dans une application de jeu de télévision interactive pour arbitrer un ensemble d'objets multimédias dans le Chapitre 9 puis nous l'appliquerons dans une application commerce électronique afin d'arbitrer un ensemble d'entités assistant l'utilisateur pour la gestion de contrats électroniques dans le Chapitre 10.

Chapitre 9

Arbitrage d'Objets Multimédias Interactifs

Nous allons détailler dans ce chapitre l'intégration et l'utilisation du modèle d'Institution Électronique MA$\mathcal{B}_{\text{ELI}}$ dans le domaine des animations multimédias interactives. Ce domaine a longtemps spécifié le comportement des objets impliqués dans les scènes selon des scénarios préétablis et rigides [78, 106]. Afin d'obtenir des scènes plus flexibles, les objets sont considérés depuis peu comme des agents autonomes capables de s'adapter au contexte. C'est le cas notamment des approches développées par le Centre de Recherche Public Henri Tudor au travers des travaux menés sur le concept RAMO (Reactive and Adpative Multimedia Object) [86, 85].

Nous nous plaçons ainsi dans le cadre des objets multimédias adaptatifs, c'est-à-dire d'objets ayant les capacités à adapter leur comportement en fonction de l'environnement dans lequel ils se trouvent. En fonction des interactions entre les objets, ou simplement en fonction du contexte dans lequel un objet se trouve, des contraintes sur son comportement sont définies. Ainsi, lors de son exécution, un objet doit s'adapter au contexte pour respecter les contraintes de comportement. Par exemple, nous pouvons imaginer qu'un objet multimédia simulant la réalité soit soumis à un ensemble de contraintes, comme les lois de la physique afin d'éviter à un objet de s'envoler dans une scène, mais également les lois régissant des sociétés d'objets comme le code de la route. Nous aimerions faire en sorte qu'un objet multimédia autonome évoluant dans une scène selon des objectifs personnels et globaux respecte un certain nombre de contraintes.

Etant autonomes, les objets multimédias basés sur les agents peuvent "décider" de privilégier un objectif à atteindre par rapport à une contrainte à respecter et de ce fait l'enfreindre. Pour éviter d'avoir affaire à une société d'objets instables et pour garder un comportement global cohérent, "conforme" au déroulement du scénario défini, il faut mettre en place un système d'arbitrage capable d'imposer un comportement. Pour cela, il doit contrôler le bon respect des contraintes et sanctionner le cas échéant. Il est également crucial d'accompagner l'autonomie des objets en spécifiant leurs droits et leurs devoirs dans les différents contextes de la scène où ils sont immergés.

Dans ce chapitre, nous présentons comment l'utilisation de MA$\mathcal{B}_{\text{ELI}}$ peut être un élément de réponse à ce besoin. Nous montrons également ici l'utilisation d'une Institution pour l'Organisation et le contrôle de plusieurs agents. Le chapitre est organisé comme suit. Nous abordons en premier lieu les motivations qui nous poussent à intégrer MA$\mathcal{B}_{\text{ELI}}$ au sein de cette application (Section 9.1). Nous détaillons l'OS définie à l'aide de $\mathcal{M}$OISE^{Inst} dans les Sections 9.2 et 9.3. Enfin nous abordons l'implémentation de l'Institution Électronique (Section 9.4) en illustrant son fonctionnement selon un exemple de résultats d'exécution (Section 9.5).

9.1 Motivations

L'application concerne un jeu télévisuel interactif. Elle est l'extension de l'application utilisée comme démonstrateur du projet européen ITEA Jules Verne dont le but était de mettre en place un langage de description d'objets multimédias réactifs et adaptatifs. Une fois l'application rapidement décrite, nous en déduirons nos objectifs concernant son "agentification".

9.1.1 Présentation de l'application

La Fig. 9.1.1 donne un aperçu du jeu que nous allons étudier ici. Il consiste en une série de "question – réponse" opposant deux équipes. Une est composée de joueurs réels présents sur le plateau de l'émission et l'autre est composée de téléspectateurs représentés par des Avatars. Chaque Avatar est directement contrôlé par son téléspectateur. Un directeur de jeu (QuizMaster), non visible à l'écran est en fait un être humain qui choisit les questions à poser et les changements de manches. Il est épaulé par un assistant virtuel (le GameMaster) régulant le jeu et contraignant les joueurs à respecter les règles.

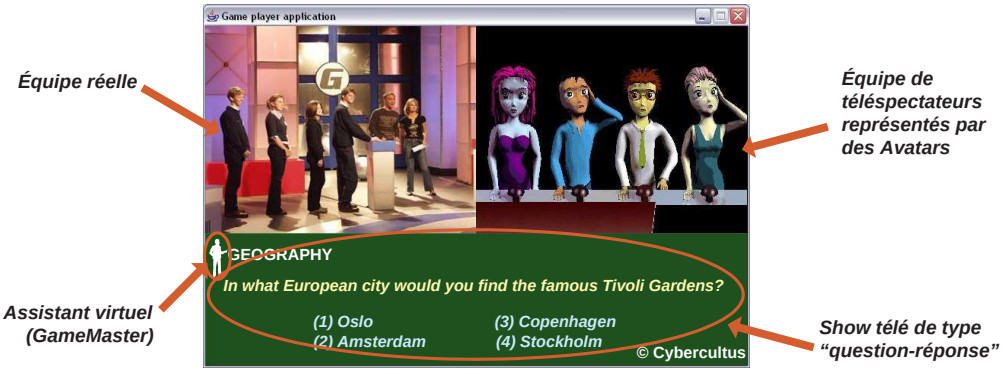


Fig. 9.1 – Interface cliente de l'application des Avatars

Ce jeu est en fait collectif bien que les téléspectateurs ne se connaissent pas, ne se voient pas et interagissent uniquement au travers de leur Avatar. Un téléspectateur doit pouvoir décider de répondre alors que ce n'est pas son tour et risquer de se faire sanctionner. Ainsi, afin de se retrouver dans des conditions réelles de jeu, les Avatars devront pouvoir ne pas respecter les règles du jeu. La version actuelle du jeu met en place un contrôle total de l'Avatar par son utilisateur. L'Avatar est autonome vis-à-vis des règles du jeu. C'est une des raisons pour laquelle nous avons retenu cette application. En effet elle nous libère du développement des mécanismes de délibération internes à un agent sur la décision de respecter ou non une norme. Les téléspectateurs prennent la décision en fonction des normes qui leur seront communiquées par leur Avatar.

Les règles du jeu sont les suivantes :
- Chaque équipe est composée de quatre membres ayant chacun une spécialité différente choisie entre Histoire, Géographie, Science et Sport.
- Les questions posées font partie d'une catégorie correspondant aux spécialités des joueurs.
- Un des membres de chaque équipe est nommé Chef.
- Les équipes répondent aux questions chacune à leur tour.
- Une partie se joue en trois manches.
- Pendant la première manche, tous les joueurs de l'équipe peuvent répondre.
- Pendant la deuxième manche, seuls les joueurs ayant leur profil correspondant à la catégorie de la question sont autorisés à répondre. Cependant, les autres joueurs de l'équipe peuvent eux

aussi répondre en sachant qu'une réponse juste fait gagner moins de point et qu'une réponse fausse en fait perdre.
– Pendant la troisième manche, seul le chef d'équipe peut répondre.

L'utilisateur au travers de son Avatar est libre d'agir comme il le souhaite au sein de son équipe. Il peut enfreindre les règles du jeu et risquer de faire sanctionner son équipe. Pour qu'il se fasse sanctionner, il faut que la violation d'une règle du jeu soit détectée. Ce n'est pas le rôle de l'assistant virtuel. Il doit y avoir un intermédiaire capable de superviser les interactions entre les joueurs et l'assistant virtuel afin de contrôler que les règles du jeu soient respectées.

9.1.2 Objectifs

L'objectif principal de ces travaux est de faire en sorte que les téléspectateurs respectent les règles. Pour cela, ils sont représentés au sein du jeu par les Avatars basés sur des agents les rendant ainsi autonomes. Il nous faut également définir un ensemble de contraintes spécifiant à la fois la structure de l'équipe, son fonctionnement et les règles du jeu qu'elle doit respecter. Les Avatars évoluent au sein de cette organisation et en subissent les contraintes. Ils sont le moyen de rendre les contraintes présentes auprès des téléspectateurs. Enfin, les joueurs pouvant toujours ordonner à leur Avatar d'enfreindre les règles de l'équipe, il faut les superviser à l'aide d'un système d'arbitrage.

Pour cela et comme illustré sur le schéma de la FIG. 9.2, nous ajoutons aux objets multimédia RAMO une composante 'agent' leur permettant de prendre en compte une OS décrite selon $\mathcal{M}$OISE^{Inst} et d'être supervisables par l'ensemble des agents institutionnels de $\mathcal{S}$YNAI.

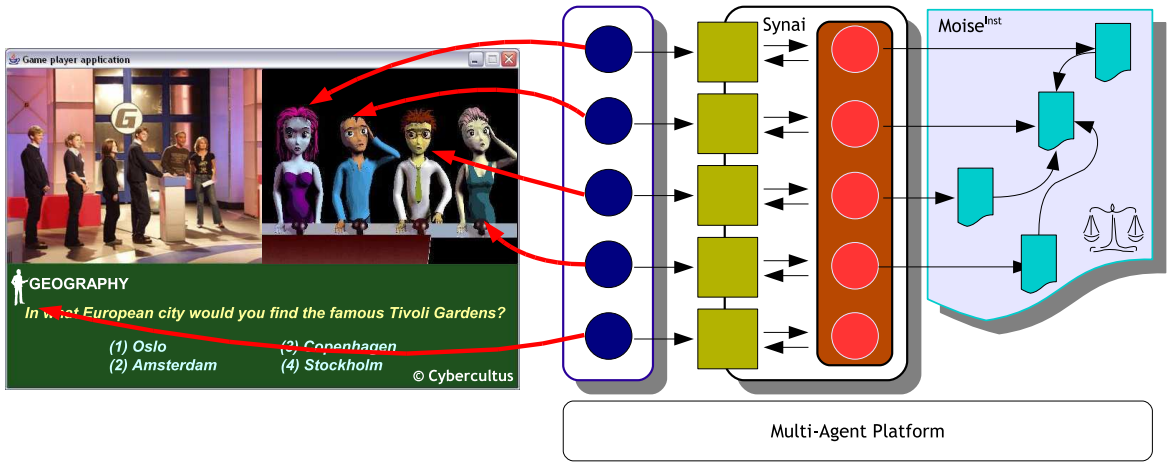


FIG. 9.2 – Vue globale de l'organisation multi-agent pour la télévision interactive

En combinant le modèle de développement multi-agents MA$\mathcal{B}_{\text{ELI}}$ avec d'autres technologies de développement d'applications multimédia interactives (cf. Fig. 9.2), nous obtenons un système à deux couches : (i) un Système Multi-Agents dédié au domaine d'application dans lequel opère un ensemble d'agents autonomes (les avatars assistant le téléspectateur dans le jeu), (ii) un "intergiciel" multi-agent institutionnel, $\mathcal{S}$YNAI, pour la gestion des règles du jeu et la régulation du système. Ces deux couches interprètent la spécification explicite des règles du jeu, programmée avec le modèle $\mathcal{M}$OISE^{Inst}. Les Avatars sont ainsi capables d'interpréter et de présenter à leur utilisateur la spécification décrite avec $\mathcal{M}$OISE^{Inst}. Celui-ci a ensuite la possibilité de la prendre en compte ou non et de faire agir son Avatar en conséquence. L'intergiciel institutionnel se base sur cette spécification pour superviser et contrôler le fonctionnement des Avatars. Les deux couches reposent sur la plate-forme d'exécution et de communication agents SACI [56].

La composante 'agent' des Avatars ainsi que les couches $\mathcal{M}$OISE^{Inst} et $\mathcal{S}$YNAI doivent intégrer les chaînes de production et d'exécution de contenu multimédia interactif et adaptatif.

9.2 Vision Générale de l'Application

L'application iDTV comporte deux phases principales pour l'exécution d'un show télévisé : la production du contenu avec la définition des règles du show et l'exécution de ce contenu.

9.2.1 Production de contenu multimédia interactif et adaptatif

La phase de production définit statiquement les composants d'un show. Comme illustré sur la FIG. 9.3 l'outil de composition de show permet de préciser les éléments graphiques de la scène, à savoir les rendus d'Avatars, du pupitre, des buzzers, ainsi que le type de jeu et les règles qui seront utilisées. Cet outil rend possible la composition d'un show à partir d'éléments existants et déjà définis. Il ne permet pas de fixer des règles d'interactions entre les éléments comme on pourrait l'imaginer dans la définition d'une scène multimédia composée d'objets adaptatifs [86]. Néanmoins, nous apportons avec $\mathcal{M}$OISE^{Inst} la possibilité de spécifier une organisation normative et d'obtenir ainsi un ensemble de liens et de contraintes entre les éléments "autonomes" de la scène, en l'occurrence, les Avatars. Ces règles définissent les rôles que les joueurs peuvent avoir et ce qu'ils peuvent et ne peuvent pas faire durant la partie.

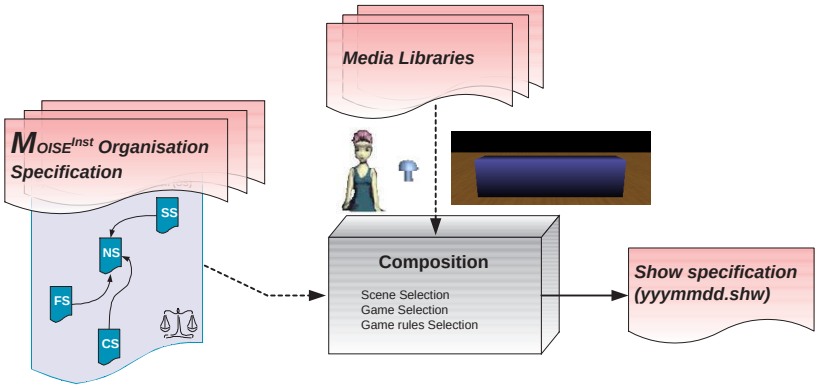


FIG. 9.3 – Vue simplifiée de la production de contenu de télévision interactive

Nous obtenons à la sortie de cette chaîne une spécification de show. Il n'est fait pour le moment aucune mention d'Avatars autrement que via le rendu graphique qu'ils auront. Nous savons qu'ils évolueront dans une scène définie, qu'ils devront réagir de telle façon à tels événements de la scène, et qu'ils devront respecter un certain nombre de contraintes pour évoluer dans la scène. L'instanciation de l'organisation $\mathcal{M}$OISE^{Inst} correspond à l'intervention des objets multimédias autonomes jouant le rôle de joueurs et endossant le costume d'Avatar et à l'exécution de la partie.

9.2.2 Exécution de contenu multimédia interactif et adaptatif

A partir de la spécification de show issue de la phase de production de contenu multimédia, la phase d'exécution crée une version déployable du jeu. La définition du show intègre maintenant des informations relatives à l'organisation des objets multimédias qu'il faudra interpréter et exécuter. Nous faisons intervenir pour cela le package $\mathcal{M}$OISE^{Inst} de l'API MA$\mathcal{B}_{\text{ELI}}$. Ainsi et comme illustré sur la FIG. 9.4, la structure de définition de show sera interprétée par les API JAVA de l'application et de $\mathcal{M}$OISE^{Inst} afin de faire fonctionner un ensemble d'Avatars ayant une composante multimédia et

une composante agent. La composante agent des Avatars ne peut fonctionner qu'au sein d'une plate-forme agents permettant également à un ensemble d'agents de supervision de contrôler les Avatars. L'API JAVA de SACI et le package SYNAI seront ainsi utilisés comme base de fonctionnement de l'application.

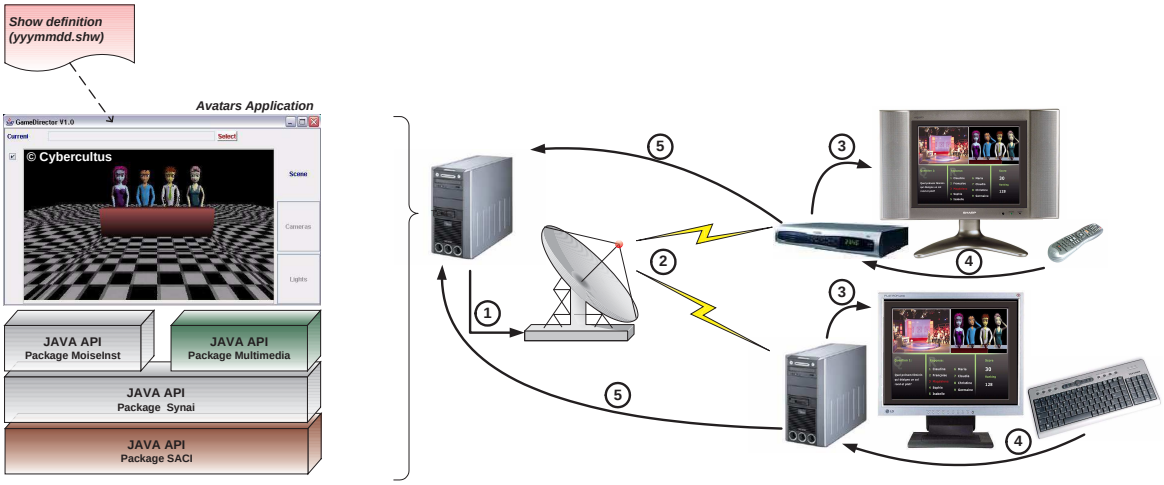


FIG. 9.4 – Vue simplifiée de l'exécution de contenu télévision interactive

Cette première partie d'exécution de code JAVA se fait sur une machine serveur. Les interfaces que nous voyons sur la FIG. 9.4 sont visibles seulement par l'utilisateur mettant en scène son show (le QuizMaster). Le fonctionnement consiste en un flux de données vidéo (1) de l'émetteur envoyé par différents canaux et réceptionné par les utilisateurs (2). Du côté utilisateur, suivant s'il reçoit le flux via le satellite dans le cadre d'une vraie application de télévision interactive ou par Internet dans le cas d'une version web de cette application, une set-top-box (démodulateur et décodeur) ou une application fonctionnant sur l'ordinateur de réception interprète le flux. Ensuite, l'utilisateur interagit directement avec la partie s'exécutant sur la partie cliente (PC ou Set-top-box) (4). Ses actions sont retournées au serveur soit via le canal de retour du satellite, soit via une ligne téléphonique ou une connexion Internet pour une utilisation avec des technologies plus classiques (5).

Nous allons maintenant étudier dans les deux prochaines sections le cœur de ces deux phases à savoir la spécification des règles du jeu via la définition d'un modèle organisationnel $\mathcal{M}$OISE^{Inst} puis son interprétation et exécution grâce à un ensemble d'API Java, et notamment celle concernant l'architecture des Avatars eux mêmes.

9.3 Spécification de l'Organisation avec $\mathcal{M}$oise^{Inst}

Durant la phase de définition du jeu, le producteur du show doit spécifier non seulement les décors et toute la partie multimédia, mais aussi l'organisation des Avatars. Il doit pour cela utiliser le modèle $\mathcal{M}$OISE^{Inst}. Grâce à celui-ci, le producteur définit la structure de l'équipe, les fonctions que celle-ci pourra avoir, les contextes dans lesquels l'organisation pourra se trouver et enfin les règles qui guideront les actions des agents selon les contextes via les quatre vues de spécifications fournies par $\mathcal{M}$OISE^{Inst}. Nous donnons en détail un exemple de spécification pour le scénario des Avatars dans les sous-sections qui suivent. Nous voyons dans TAB. 9.1 les événements, les contextes et les buts que nous avons définis pour cette application en particulier et qui seront utilisés dans l'OS. Nous reviendrons sur les buts lors de la description de la FS et sur les contextes et les événements lors de la description de la CS.

Goal	
g1	Team joined
g2	Game played
g2a	All questions handled
g2b	Question handled
g3	Team quit
g4	Topic handled
g4x	*Category* topic handled
g4x1	*Category* question asked
g4x2	*Category* question answered
g5	Answer evaluated
g6	Sanction applied
g61	Player ejected
g62	Team disqualified
g7	Score changed
g71	Score increased
g72	Score decreased
g8	Emotion shown
g81	Be happy
g82	Be sad

Context	
Begin	Joining context
Game	Game context
End	Quitting context
Round1	First round of the game
Round2	Second round of the game
Round3	Third round of the game
TurnStart	Temporaly context to define which team will start
TurnStop	Temporaly context to define which team will stop
MyTurn	Turn of the Avatars team
NotMyTurn	Turn of human beings team

Event	
beginG	Begins the game
endG	Ends the game
chgRd	Changes round
chgT	Changes turn
avT	It is Avatars' turn
hmT	It is human beings' turn

TAB. 9.1 – Vocabulaire utilisé pour la spécification d'une application de jeu télévisé interactif

9.3.1 Spécification Structurelle

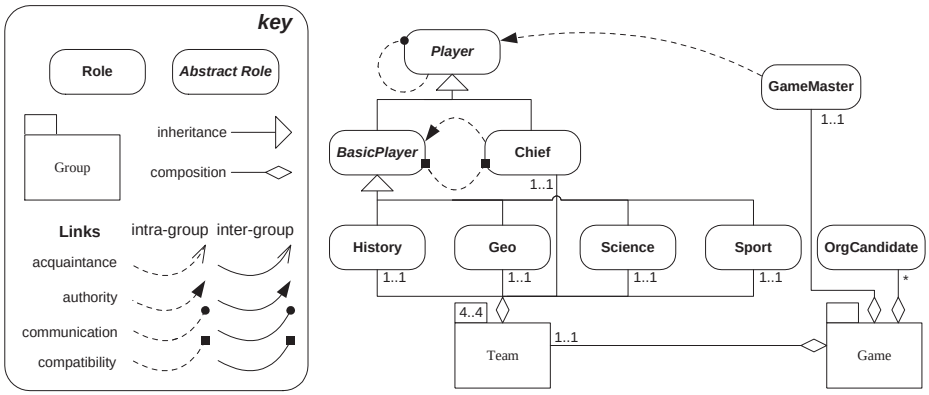


FIG. 9.5 – Spécification Structurelle du scénario Avatars

La SS du scénario des Avatars est représentée par la FIG. 9.5. Le groupe racine est le groupe "Game" composé d'un seul groupe "Team" (cardinalité "1..1"). Ce groupe représente une équipe de quatre Avatars (cardinalité "4..4"). Elle devra affronter l'équipe réelle. L'équipe est composée des cinq rôles "History", "Geo", "Sport", "Science" et "Chief". Ils peuvent être adoptés une et une seule fois dans le groupe (cardinalité "1..1"). Les rôles "History", "Geo", "Sport", "Science" héritent de "BasicPlayer". Les deux rôles "BasicPlayer" et "Chief" sont compatibles. Cependant comme le rôle "BasicPlayer" est abstrait un Avatar pourra jouer en même temps le rôle de chef et l'un des rôles héritant de "BasicPlayer". Du fait des cardinalités, les quatre Avatars de l'équipe joueront chacun un rôle différent et l'un d'eux sera le chef. Le rôle "Chief" possède également un lien d'autorité sur le rôle "BasicPlayer" ce qui donne la possibilité au chef d'avoir l'autorité sur tout Avatar de l'équipe. Tous les rôles cités précédemment héritent (directement ou indirectement) du rôle abstrait "Player".

Le groupe "Game" est également composé des rôles "OrgCandidate" et "GameMaster". Le téléspectateur voulant participer à la partie se connecte au système et doit passer par la première étape créer son Avatar. Une fois son Avatar créé, il peut rejoindre une partie si cette dernière n'est pas déjà en cours. Dans le cadre de la partie, l'Avatar joue d'abord le rôle d'un candidat ("OrgCandidate") avant de pouvoir jouer le rôle d'un joueur au sein d'une équipe. En tant que candidat, il sera soumis à des droits et devoirs comme par exemple celui de ne pas rejoindre la partie en cours. Le rôle "GameMaster" a, quant à lui, une cardinalité de "1..1" puisqu'il représente le seul Avatar assistant le directeur du jeu. De par son rôle de *maître du jeu*, il existe un lien d'autorité entre lui et le rôle "Player" lui donnant la possibilité d'avoir autorité sur tous les rôles du groupe "Team".

La structure de l'équipe étant définie, voyons les fonctions que ces rôles auront à mettre en œuvre via la Spécification Fonctionnelle.

9.3.2 Spécification Fonctionnelle

Le but global de l'organisation est de faire en sorte qu'une partie soit jouée correctement. Pour cela et comme représenté sur la FS de la Fig. 9.6, plusieurs schémas fonctionnels doivent être mis en place. Le plus important est celui relatif au fonctionnement de l'organisation dans sa globalité à savoir manipuler un ensemble de questions afin de jouer la partie ("Functional Scheme"). Pour que le jeu se déroule, il faut que des questions soient posées et que des réponses soient données ("Question Scheme"). De plus, un score doit être calculé ("Score Scheme") et des sanctions appliquées en cas de faute de jeu ("Sanction Scheme"). Enfin, nous avons décidé de pouvoir gérer les émotions exprimées par les Avatars ("Emotion Scheme") et les entrées et sorties des joueurs dans l'équipe ("OrgEnter Scheme" et "OrgExit Scheme").

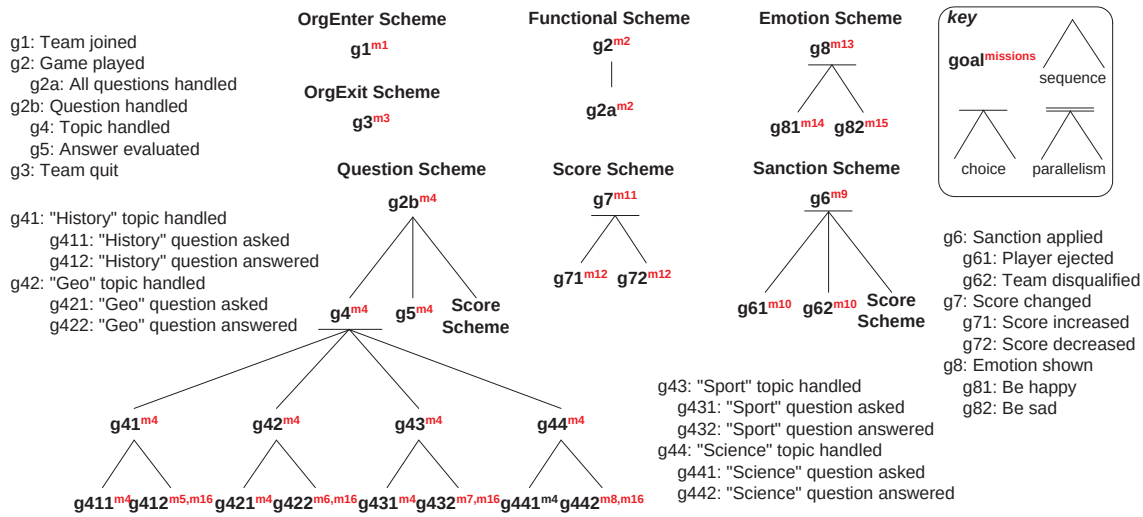


Fig. 9.6 – Spécification Fonctionnelle du scénario des Avatars

Les buts qui sont utilisés au sein de la FS proviennent du vocabulaire défini dans le Tab. 9.1. Ils sont ici organisés en plan au sein des schémas. Le schéma social "Functional Scheme" a ainsi comme but racine le fait de jouer une partie (but "g2"). Ce but est atteint quand un ensemble de questions a été traité, c'est-à-dire que des questions ont été posées et des réponses ont été données (but "g2a").

L'exécution d'un certain nombre d'instance de schéma "Question Scheme" dont le but racine est de traiter une question permet d'atteindre le but racine "g2a" du "Functional Scheme".

– Une question est traitée (but "g2b") quand tour à tour un thème est abordé (but "g4"), une réponse est évaluée (but "g5") et le score est changé (schéma "Score Changed").
– Aborder un thème s'atteint en choisissant entre Histoire, Géographie, Science et Sport (buts "g41", "g42", "g43" et "g44").
– Aborder un thème choisi se fait en posant une question de ce thème (buts "g411", "g421", "g431" et "g441") et en y répondant (buts "g412", "g422", "g432" et "g442"). Le but de répondre à une question est atteint quelle que soit la véracité de la réponse.

Le schéma "Score Scheme" possède comme but racine le fait d'agir sur le score (but "g7"). Agir sur le score se traduit par un choix entre le baisser (but "g71") ou l'augmenter ("g72"). De même, montrer son émotion (but racine "g8" du schéma "Emotion Scheme") est un choix entre montrer un visage heureux (but "g81") ou montrer un visage triste (but "g82"). Le "Sanction Scheme" dont le but est d'appliquer une sanction (but "g6") est un choix entre éliminer le joueur (but "g61"), disqualifier l'équipe (but "g62") ou agir sur le score (schéma "Score Scheme"). Les buts "g1" et "g3" composent à eux seuls les schémas d'entrée et de sortie du groupe "Team" ("OrgEnter Scheme" et "OrgExit Scheme").

Au sein de cette FS, les missions regroupent les buts appartenant à un même schéma permettant à un agent d'atteindre un ensemble de buts cohérents entre eux. Par exemple, pour le schéma "Question Scheme", nous définissons la mission "m4" qui consiste à poser une question. Elle regroupe tous les buts relatifs à la manipulation d'une question à savoir les buts "g2b", "g4", "g41", "g42", "g43", "g44", "g411', "g421", "g431" et "g441" et à son évaluation (but "g5"). Nous définissons la mission de répondre à une question quelle que soit sa thématique en regroupant les buts "g412", "g422", "g432" et "g442". Nous définissons également quatre autres missions relatives au fait de répondre à une question sur une thématique particulière.

9.3.3 Spécification Contextuelle

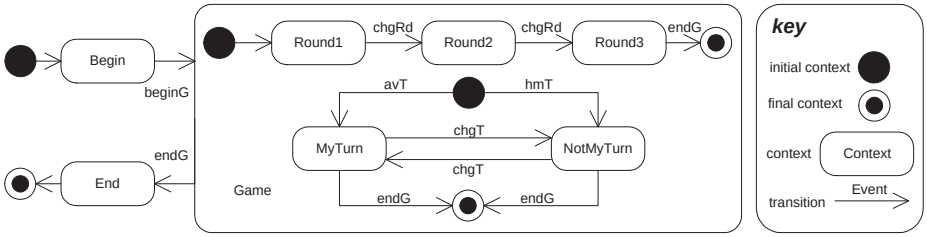


FIG. 9.7 – Spécification Contextuelle du scénario des Avatars.

La CS de l'application iDTV est représentée graphiquement par la FIG. 9.7. Les événements utilisés ici sont définis et décrits dans le TAB. 9.1. Ils permettent l'enchaînement entre les différents contextes de la façon suivante :

1. Une partie débute dans un contexte de synchronisation "Begin" qui permet aux téléspectateurs de se connecter au système, de créer leur Avatar et de constituer l'équipe. L'événement *beginG* déclenche le début de la partie.
2. Le macro-contexte "Game" définit la période de temps durant laquelle les Avatars sont en train de jouer. Les sous-CS de "Game" débutent lorsque le contexte "Game" devient actif.
   – Une sous-CS représente les différentes manches (contextes "Round1", "Round2" et "Round3"). La transition entre chaque manche se fait sur l'apparition de l'événement *chR*.
   – Une autre sous-CS représente les tours de réponse (contextes "MyTurn" et "NotMyTurn"). Elle définit si c'est au tour ou non des Avatars de répondre. Les événements avT et hmT débutent la CS soit par le tour des Avatars soit par le tour des joueurs réels. Ensuite, l'événement chT déclenche un changement de tour.

– Un sous-contexte de manche et un sous-contexte de tour peuvent être actifs en même temps. Le macro-contexte "Game" est également actif dans chaque sous-contexte. Ainsi l'Organisation peut avoir en même temps les contextes "Game", "Round2" et "NotMyTurn" actifs. L'événement *endG* termine les sous-CS.

3. L'événement *endG* déclenche aussi le passage au contexte "End". C'est le contexte dans lequel les Avatars quittent leur équipe.

9.3.4 Spécification Normative

La NS du scénario des Avatars est représentée par le Tab. 9.2. Cette spécification nous permet d'exprimer sous forme de normes les différentes règles du jeu ainsi que divers comportements que les Avatars doivent ou peuvent avoir avant, pendant et après la partie. Nous les regroupons en plusieurs catégories.

context	id	w.	condition	issuer	bearer	deOp	mission	deadline	sanction
Begin	N01	1	nb(Team)<max(Team)	Supervisor	OrgCandidate	O	m1	---	---
End	N02	1	---	Supervisor	Team	O	m3	---	---
Game	N03	1	---	Supervisor	OrgCandidate	F	m1	---	N17
Game	N04	1	---	Supervisor	Team	F	m3	---	---
Game	N05	1	---	Supervisor	GameMaster	O	m2	---	---
Game	N06	1	---	Supervisor	GameMaster	O	m4	---	---
Game	N07	1	---	Supervisor	Team	P	m13	---	---
Game	N08	2	---	Supervisor	Team	F	m16	---	N18
Round1	N09	3	---	Supervisor	Team	P	m16	< answer_delay	---
Round2	N11	1	---	Supervisor	History	P	m5	< answer_delay	---
Round2	N12	1	---	Supervisor	Geo	P	m6	< answer_delay	---
Round2	N13	1	---	Supervisor	Sport	P	m7	< answer_delay	---
Round2	N14	3	---	Supervisor	Science	P	m8	< answer_delay	---
Round3	N10	1	---	Supervisor	Chief	P	m16	< answer_delay	---
NotMyTurn	N15	1	---	Supervisor	Team	F	m16	---	---
NotMyTurn	N16	1	---	Supervisor	Team	F	m14	---	---
Game	N17	1	violated(N02)	Supervisor	GameMaster	O	m9	---	---
Game	N18	1	violated(N08)	Supervisor	GameMaster	O	m11	---	---

Tab. 9.2 – Spécification Normative du scénario des Avatars

Gestion des entrées et sorties

- ▷ *Le rôle "OrgCandidate" est obligé de rejoindre une équipe avant le début de la partie.* Cela se traduit par la norme N01. Cette norme possède une condition testant s'il reste des rôles à jouer dans le groupe "Team" en faisant la différence entre le nombre d'Avatars dans le groupe et le nombre de places disponibles.
- ▷ La norme N02 exprime le fait que *les membres du groupe "Team" doivent quitter l'équipe en fin de partie.*
- ▷ *Il est interdit de rejoindre une équipe ou de la quitter pendant la partie*, ce qui est spécifié par les normes N03 et N04. Si un Avatar rejoint tout de même l'équipe durant la partie (ce qui est normalement impossible étant donnée la condition sur N01), la N17 devient valide en tant que sanction et *oblige le "GameMaster" à éjecter l'Avatar fautif de l'équipe.*

Comportement général durant le jeu

- ▷ L'Avatar assistant le QuizMaster doit faire en sorte que la partie soit jouée, le *"GameMaster" est obligé de traiter un ensemble de questions en les posant, puis en évaluant les réponses et en mettant à jour le score.* Pour cela nous définissons les normes N05 et N06.
- ▷ Afin de spécifier les règles de chaque manche, *il est interdit au groupe "Team" de répondre aux questions dans le contexte "Game"* (N08). Si cette norme n'est pas respectée, la sanction N18 est appliquée. Cette sanction est conditionnée par le fait que la norme N08 soit violée. Si N08

est violée, alors le rôle "GameMaster" a l'*Obligation d'agir sur le score et ce durant toute la durée de la partie.*

Comportement particulier durant les manches

Le fait d'interdire par défaut aux joueurs de répondre aux questions durant la partie va nous permettre de définir des permissions (et non pas des obligations car le but n'est pas de contraindre les Avatars à répondre mais de faire en sorte qu'ils répondent au moment voulu). Ce sont des permissions de répondre aux questions concernant des rôles en particulier et dans des sous-contextes du jeu avec une priorité plus élevée donc prioritaires à l'interdiction précédente.

- ▷ *Dans le contexte "Round1", le groupe "Team" est autorisé à répondre à toutes les questions*, ce qui se traduit par la norme N09.
- ▷ *Dans le contexte "Round2", chaque spécialiste peut répondre aux questions de sa catégorie.* Nous le spécifions par quatre normes différentes visibles dans le TAB. 9.2 sous les identifiants N11, N12, N13 et N14.
- ▷ *Dans le contexte "Round3", le rôle "Chief" est autorisé à répondre à toutes les questions* que nous spécifions par la norme N10.

Pour toutes ces normes nous définissons une contrainte temporelle obligeant les Avatars à donner une réponse avant la fin du temps imparti. La violation de ces normes n'est pas sanctionnée puisque ce sont des permissions. Si un Avatar ne répond pas, il perd uniquement l'occasion de gagner des points. S'il répond alors qu'il ne le devait pas (violation de N08 sans autorisation d'une autre norme au poids plus élevé) son score est alors diminué.

Gestion des tours de réponse et des émotions

- ▷ Une règle du jeu énonce que chaque équipe doit répondre à son tour. Nous définissons donc l'*Interdiction pour l'équipe de répondre à une question lorsque ce n'est pas son tour* par la norme N15.
- ▷ Nous pouvons spécifier quelques contraintes grâce aux normes pour adapter le comportement multimédia des Avatars. Ainsi *il est permis aux membres de l'équipe de montrer leurs émotions* ce que nous traduisons par la norme N07. Par contre *il leur est interdit de montrer un visage heureux lorsque c'est au tour de l'autre équipe de répondre aux questions.* Nous spécifions cette règle grâce à la norme N16. Bien entendu la contrainte d'un comportement typiquement multimédia ne peut se faire que si ce comportement est exprimé sous forme de buts dans la FS et qu'une correspondance entre ces buts et une méthode spécifique à l'objet multimédia est faite au sein de l'Avatar.

La vérification de la cohérence des normes est à la charge du concepteur du modèle. En l'occurrence, des conflits potentiels peuvent survenir entre les normes N15 et N09 parce que ces dernières obligent les Avatars appartenant au groupe "Team" à accomplir, respectivement à ne pas accomplir la mission "m16". Nous avons le même cas entre les normes N15 et N14, N8 et N9, et enfin N8 et N14. De ce fait et afin d'éviter que les agents aient à faire un choix entre deux normes conflictuelles à respecter, nous spécifions un ordre de priorité représenté par *w.* dans le tableau. La valeur *1* correspond à la priorité la plus haute. Pour supprimer les conflits entre les normes, nous diminuons l'ordre de priorité des normes N9 et N14.

Nous allons aborder maintenant l'instanciation des Avatars et du modèle que nous venons de présenter à l'aide des API de $\mathcal{M}$OISE^{Inst} et $\mathcal{S}$YNAI pour la partie agents et l'API de l'application existante pour la partie multimédia.

9.4 Implémentation des Avatars

L'application existante dans laquelle nous intégrons le modèle d'Institution Électronique basé sur des agents est construite autour d'un *package* (API) d'objets multimédia. Les Avatars doivent être constitués d'une partie multimédia provenant de ce package et d'une partie agent pouvant prendre en compte une modélisation $\mathcal{M}$OISE^{Inst}. Nous allons voir dans un premier temps l'architecture des objets multimédia autonomes puis les spécificités de comportement dans le cas où l'Avatar assiste le téléspectateur ou le QuizMaster.

9.4.1 Architecture des Avatars

En premier lieu, la composante agent des Avatars nécessite une plate-forme d'exécution multi-agents pour évoluer. La plate-forme SACI étant utilisée pour faire communiquer et faire interagir les agents de $\mathcal{S}$YNAI entre eux. Les Avatars devront également l'utiliser. Ainsi, la composante agent des Avatars est construite sur base de la classe d'agents fournie par le package `saci` (cf. la section 8.3.2 du Chapitre 8) et ont la même base d'agent organisationnel que les agents de $\mathcal{S}$YNAI. Comme illustré sur le diagramme d'état de la FIG. 9.8, la classe `Avatar` hérite de la classe `OrgAgent` provenant du package `synai`. L'Avatar possède donc toutes les capacités d'organisation et de décision d'un agent. Il est lié à la classe `ShowPlayer` pour sa composante multimédia. La classe `ShowPlayer` caractérise un objet multimédia représentant un personnage. De ce fait, ses attributs renseignent sa position, le pseudo du joueur et son équipe.

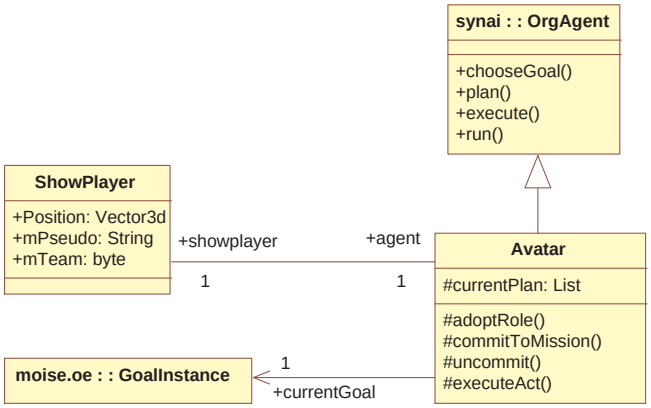


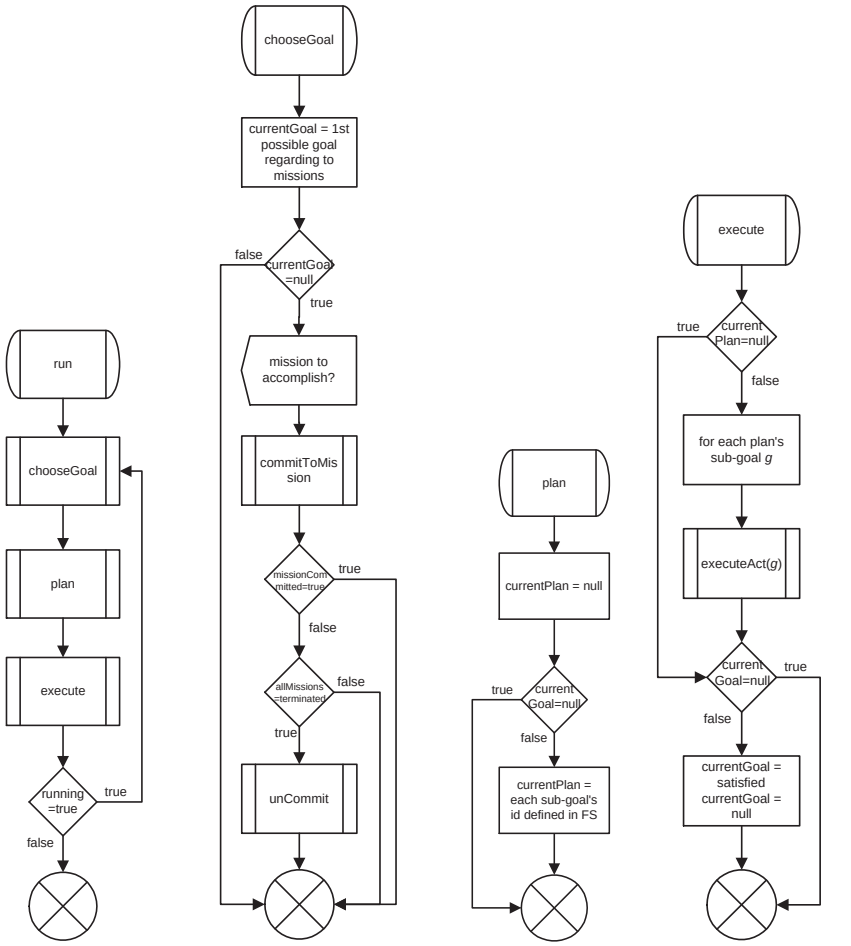
FIG. 9.8 – Diagramme de classe de la représentation d'un Avatar

Nous distinguons deux types d'Avatars : celui qui assiste et représente le téléspectateur (Avatar de type *Player*) et celui qui représente le QuizMaster (Avatar de type *GameMaster*).

9.4.2 Exécution des Avatars de type *Player*

Les Avatars de ce type doivent exhiber un comportement de jeu. Leur cycle de vie doit donc leur faire adopter un rôle puis satisfaire des buts en répondant aux questions jusqu'à la fin de la partie. Bien entendu, une grande partie des méthodes nécessite une interaction avec le téléspectateur, ce dernier contrôlant son Avatar.

Une fois l'Avatar initialisé, son cycle de vie commence grâce à l'exécution de la méthode *run()* héritée de la classe `OrgAgent`. Comme nous l'avions défini dans la Section 8.3.3 du Chapitre 8, cette méthode est une suite d'appels itératifs des fonctions *chooseGoal()*, *plan()* et *execute()* (cf. FIG. 9.9(a)). Ces méthodes sont surchargées pour fonctionner dans notre domaine et pour cette application en particulier. Nous détaillons ces méthodes dans les sections suivantes.



(a) Exécution (b) Choix d'un but (c) Plan du but (d) Exécution du plan

FIG. 9.9 – Organigramme du cycle de vie des Avatars de type *Player*

Méthode *chooseGoal()*

La FIG. 9.9(b) illustre cette méthode. Elle consiste à faire le choix d'un but à atteindre. Ce but est choisi parmi ceux de la mission sur laquelle l'Avatar est engagé. Le choix de la mission à accomplir se fait par le téléspectateur. Si le choix d'un but ne se fait pas c'est peut être parce que la mission sur laquelle l'Avatar est engagé est accomplie. Si c'est effectivement le cas, il s'en désengage.

Méthode *plan()*

Contrairement à la méthode *plan()* des agents de SYNAI, les Avatars de type *Player* ne sont pas responsables de la création des schémas d'exécution. C'est le rôle du *GameMaster* de créer un schéma et de faire en sorte que les buts soient satisfaisables pour chaque nouvelle question posée. Ainsi, si l'Avatar a un but courant défini, alors le plan courant fait référence aux sous-buts du but courant. Nous illustrons cette méthode grâce à la FIG. 9.9(c).

Méthode *execute()*

Si le plan courant n'est pas vide alors, pour chaque identifiant de sous-buts du plan courant, on appelle la méthode *executeAct()*. Le but courant est donc déclaré satisfait. Nous illustrons cette méthode avec la FIG. 9.9(d).

Méthode *executeAct()*

Cette méthode fait le lien entre un but faisant partie d'un plan et l'action que l'Avatar doit mettre en œuvre pour le satisfaire. L'organigramme de la FIG. 9.10 illustre l'algorithme dont les grands principes sont les suivants :

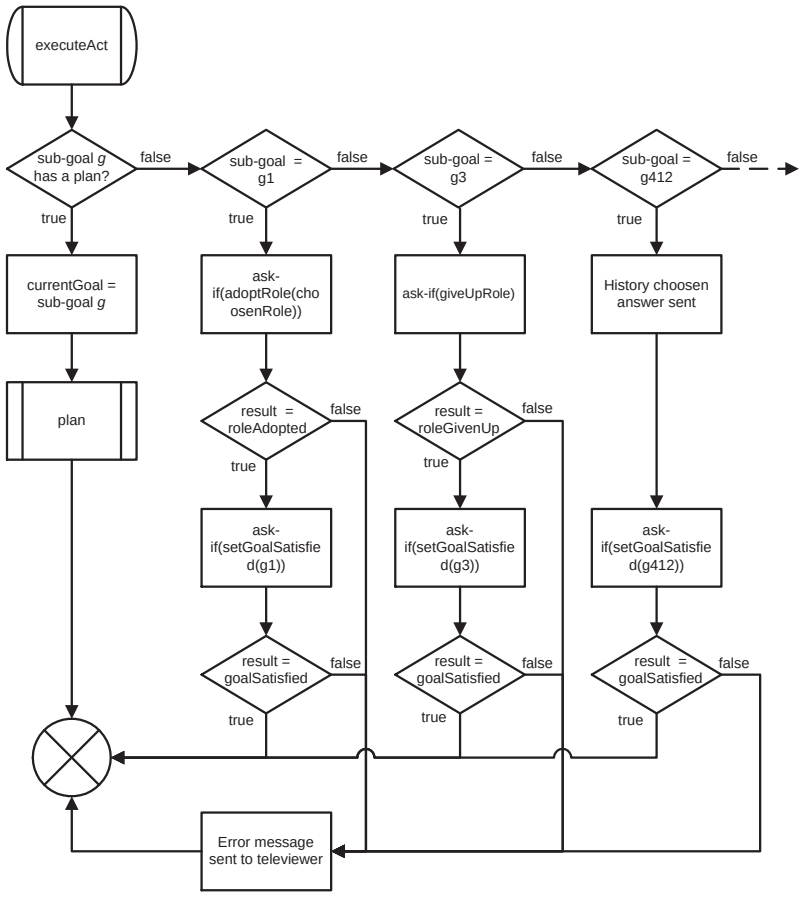


FIG. 9.10 – Organigramme de la méthode d'exécution des tâches pour un Avatar de type *Player*

- Soit le but courant possède des sous-buts, il faut alors définir son plan (appel de la méthode *plan()*).
- Soit le but courant est un but feuille (but sans plan), et alors l'action correspondante est mise en place par l'Avatar :
  - Si le but est "g1", alors une demande d'adoption de rôle est envoyée et si le rôle est effectivement adopté, une demande de satisfaction du but "g1" est envoyée.
  - Si le but est "g3", alors une demande d'abandon du rôle en cours est envoyée et si le rôle est effectivement abandonné alors une demande de satisfaction du but "g3" est envoyée.
  - Si le but est "g412", alors la réponse est envoyée ainsi qu'une demande de satisfaction de ce but. Il en est de même pour les buts "g422", "g432" et "g442".
  - Si le but est "g81" ou "g82", un rendu d'Avatar est choisi et une demande de satisfaction de but est envoyée.

  Pour tous les buts, si un problème survient un message d'erreur est envoyé au téléspectateur.

Nous remarquons que les buts "g412", "g422", "g432" et "g442" sont considérés comme satisfaits par les Avatars par la simple action de donner une réponse. Peu importe qu'elle soit bonne ou mauvaise

puisque cela n'influence pas le comportement des Avatars. Par contre, il se peut que la demande de satisfaction ne soit pas acceptée car cela violerait une spécification de l'organisation mais la raison ne sera pas le contenu même de la réponse. La raison peut être que la réponse a été donnée alors que ce n'était pas le tour de l'équipe ou qu'elle ne soit pas une question de la bonne catégorie. On pourrait par contre imaginer pour la suite des travaux, une version de l'application laissant une part d'autonomie plus large aux Avatars. Ces derniers auraient une base de connaissance leur permettant de conseiller le téléspectateur à propos de la réponse à choisir.

9.4.3 Exécution des Avatars de type *GameMaster*

L'Avatar de ce type a quelques différences avec les Avatars de type *Player*. En effet l'exécution des tâches est différente puisqu'ils ne jouent pas le même rôle. Le *GameMaster* est créé pendant l'initialisation de l'application, lors de la création de la couche $\mathcal{S}$YNAI. Au lieu d'adopter le rôle "Org-Candidate", il adopte le rôle "GameMaster". Son cycle de vie est celui d'un agent organisationnel. Ses méthodes *chooseGoal()*, *plan()* et *execute()* sont implémentées de la même manière que pour les Avatars de type *Player*, déjà décrites ci-dessus et dans la FIG. 9.9. Le *GameMaster* a par contre sa propre implémentation de la méthode *executeAct()*. Elle permet de mettre en œuvre l'action correspondante au but courant. Les actions que le *GameMaster* peut exécuter et qui sont relatives aux buts "g411", "g421", "g431" et "g441" suivent le principe d'exécution suivant :

Action : *relative au but "g4x1"*	
Description	Fait en sorte d'atteindre le but "*Category* question asked"
Arguments	Question choisie par le QuizMaster
Pré-conditions	
Post-conditions	Envoi de la question choisie par le QuizMaster aux membres des équipes Demande de satisfaction du but "g4x1"

L'exécution des actions concernant les autres buts suit le même principe. Que ce soit pour l'évaluation des réponses :

Action : *relative au but "g5"*	
Description	Fait en sorte d'atteindre le but "Answer evaluated"
Arguments	Réponse envoyée par un Avatar
Pré-conditions	
Post-conditions	Comparaison avec la réponse attendue et déduction de la véracité de la réponse reçue Demande de satisfaction du but "g5"

... ou la mise à jour des scores :

Action : *relative au but "g71"*	
Description	Fait en sorte d'atteindre le but "Score increased"
Arguments	Type de mise à jour
Pré-conditions	
Post-conditions	Si la mise à jour se fait dans le cadre d'une sanction alors le score de l'équipe est augmenté de 2 points Si la mise à jour ne se fait pas dans le cadre d'une sanction alors le score de l'équipe est augmenté de 4 points Demande de satisfaction du but "g71"

Action : *relative au but "g72"*	
Description	Fait en sorte d'atteindre le but "Score decreased"
Arguments	Type de mise à jour
Pré-conditions	
Post-conditions	Si la mise à jour se fait dans le cadre d'une sanction alors le score de l'équipe est diminué de 4 points Demande de satisfaction du but "g72"

Nous avons défini dans la Section 9.3.2 les sanctions auxquelles le schéma d'arbitrage de $\mathcal{S}$YNAI fait référence. Cependant, les capacités des agents de $\mathcal{S}$YNAI ne permettent pas de les mettre en œuvre. Ces buts devront donc être atteints par l'agent jouant le rôle *GameMaster* grâce à l'exécution des actions suivantes :

Action : *relative au but "g61"*	
Description	Fait en sorte d'atteindre le but "Player ejected"
Arguments	Avatar fautif
Pré-conditions	
Post-conditions	L'Avatar ne joue plus aucun rôle au sein de l'Organisation Demande de satisfaction du but "g61"

Action : *relative au but "g62"*	
Description	Fait en sorte d'atteindre le but "Team disqualified"
Arguments	
Pré-conditions	
Post-conditions	L'équipe des joueurs réels gagne la partie Demande de satisfaction du but "g62"

Pour toutes les exécutions d'action, si la requête de satisfaction de but est acceptée, alors le *GameMaster* peut exécuter un autre but ou se mettre en attente d'une nouvelle demande de mission de son *QuizMaster*. Sinon, un message d'erreur est adressé à ce dernier. Nous remarquons également que pour ces actions, aucun test institutionnel n'est requis. En effet, ces actions représentant les capacités des agents, c'est lors de leurs mises en application pour atteindre des buts que des tests institutionnels sont faits afin de vérifier que les agents sont autorisés à agir.

Le *matching* fait par les agents afin d'exécuter une action pour atteindre un but revient à considérer que les buts sont codés en dur dans les agents. Pour le moment les buts sont spécifiés par un identifiant et une description textuelle au sein de l'OS. Les perspectives d'amélioration pourront permettre d'associer une sémantique à un but. Ainsi, à l'aide d'une ontologie référencée par la spécification de l'organisation, il serait possible à l'agent de déterminer l'action à mettre en place pour atteindre le but sans forcément qu'une correspondance soit faite entre un but $\mathcal{M}$OISE^{Inst} et une action. Cela nous permettrait d'avoir des agents cognitifs plutôt que des agents réactifs.

En ajoutant aux objets multimédias des capacités d'organisation normative et aux définitions de show télévisé des spécifications de contraintes organisationnelles, nous obtenons un ensemble d'entités multimédias normées, réactives et adaptatives interprétant et respectant des règles définies en dehors de chaque objet. Avec l'ajout d'une couche de supervision, le respect de ces règles est contrôlé indépendamment de leur définition. L'avantage de cette solution est de pouvoir changer les règles du jeu (dans notre contexte de jeu télévisé) sans avoir à intervenir sur le comportement des Avatars ni

sur la capacité de supervision et de détection des violations des Superviseurs. Nous nous rapprochons ainsi des outils de composition de scènes multimédias permettant à un utilisateur d'ajouter un ensemble d'objets multimédias dans une scène, de définir quelques règles de fonctionnement de cette scène et de laisser un comportement global émerger de l'évolution de chaque objet.

9.5 Expérimentation et validation

Afin d'expérimenter et valider l'utilisation de notre modèle d'Institution Électronique dans le contexte du multimédia, nous allons étudier les résultats que provoquent un fonctionnement normal des agents et un comportement non-respectueux des règles en fonction de la spécification que nous avons décrite précédemment. En étudiant la différence de comportement des agents du domaine (les Avatars) et des agents de $\mathcal{S}$YNAI, nous pouvons tirer des conclusions sur la validité d'une partie de notre modèle.

9.5.1 Expérimentation

Nous décrivons dans un premier temps le déroulement d'une partie avec sa spécification puis dans un deuxième temps son exécution.

Définition du show télévisé

Comme abordé en partie dans la Section 9.2.1, les interfaces logicielles se trouvant sur la FIG. 9.11 permettent au producteur du show (le créateur du contenu), de choisir une date de diffusion du show dans un calendrier et de le définir ensuite. Cela se traduit par une composition de la scène en choisissant trois spécifications : la définition du jeu en lui-même (ici le jeu de "question – réponse"), les décors qui apparaîtront à l'écran ainsi qu'un ensemble de rendu pour les Avatars que les téléspectateurs pourront choisir et enfin la définition des règles du jeu via une spécification d'Institution à l'aide du langage MoiseIML. Le résultat de ces sélections est la création d'une spécification du show référençant les autres définitions utilisées pour ce show et pour une date précise. La date définie est mise en valeur dans le calendrier. C'est via ce calendrier que l'utilisateur va ensuite pouvoir sélectionner le show à exécuter.

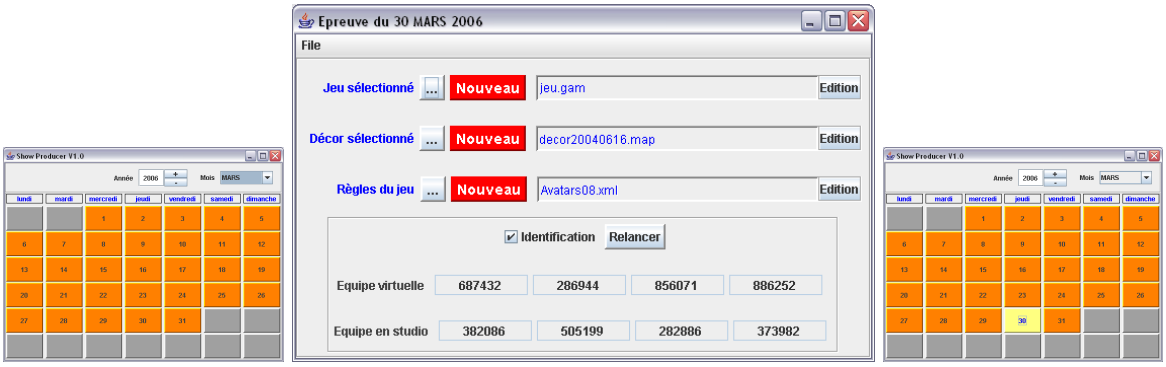


FIG. 9.11 – Interfaces pour la création d'un show

Initialisation du show télévisé

Maintenant que le show est spécifié, et avant de pouvoir l'exécuter, il faut lancer la plate-forme d'exécution multi-agent SACI permettant à la composante agent des Avatars de pouvoir évoluer et communiquer, notamment avec les agents de $\mathcal{S}$YNAI. Une fois SACI démarrée, il faut lancer la partie

"GameDirector" de l'application. Cette partie permet de choisir dans le calendrier le show défini précédemment et ainsi initialiser son exécution. Le fichier de définition du show est pris en compte, notamment la spécification d'Institution afin de créer une société SACI, créer une spécification d'organisation et son instance, les lier à la société SACI, créer un ensemble d'agents de supervision, les faire rejoindre la société et les faire adopter les rôles de supervision. A ce moment là, tous les groupes de l'organisation sont créés et le contexte "Begin" est actif. Seule la norme N01 est valide et applicable.

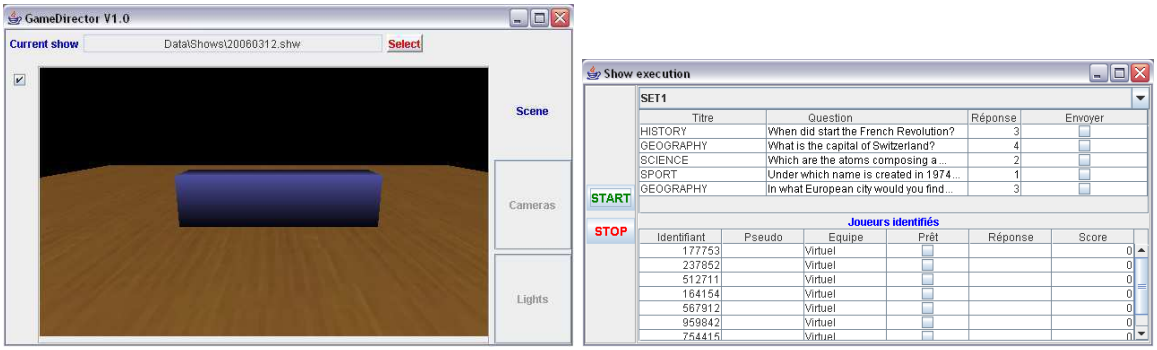


FIG. 9.12 – Interface pour l'initialisation de l'instanciation du show

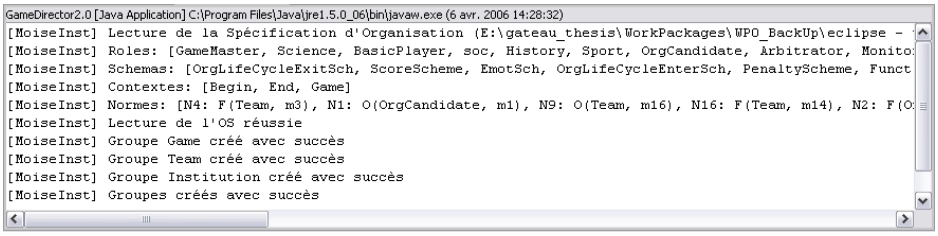


FIG. 9.13 – Affichage du résultat de la création de l'Organisation $\mathcal{M}$OISE^{Inst}

Nous retrouvons le résultat de cet enchaînement de processus concernant les agents dans la console d'exécution visible sur la FIG. 9.13. La FIG. 9.12 représente le show tel qu'il a été défini avec les décors sur l'interface de gauche et l'interface de contrôle sur la partie de droite permettant entre autre de voir quand un Avatar s'est connecté au jeu, de poser les questions et de changer de manche. Le bouton "START" permet de lancer l'exécution de la partie et de pouvoir choisir les questions à poser aux joueurs.

A ce stade, le *QuizMaster* n'a plus qu'à attendre que des joueurs (des téléspectateurs) se connectent au système afin de pouvoir démarrer la partie. Les joueurs doivent s'identifier avec un code pour être reconnus par le système. Le *QuizMaster* peut voir quel joueur est connecté et prêt à jouer grâce à la colonne "Prêt" de la partie inférieure de l'interface d'exécution. Voyons les étapes à suivre pour créer son Avatar et être prêt à jouer du côté client.

Connexion d'un téléspectateur

Lorsque l'utilisateur exécute la partie cliente de l'application, une connexion se fait entre sa set-top-box ou son ordinateur et le système. Il doit alors donner un certain nombre de renseignements que nous retrouvons en partie sur les interfaces de gauche et du milieu de la FIG. 9.14 :

– le pseudonyme par lequel il sera reconnu,

– son numéro d'identification afin de rejoindre une partie,
– le rôle qu'il veut jouer au sein de l'équipe,
– l'aspect qu'aura son Avatar choisi parmi les rendus sélectionnés lors de la définition du show.

Lorsque ces étapes sont passées, un Avatar est créé sur le serveur. La partie agent de l'Avatar va rejoindre la société SACI et adopter directement le rôle "OrgCandidate" au sein du groupe "Game". La partie objet multimédia sera affichée à l'écran et l'Avatar mettra en place des comportements définis par cet objet.

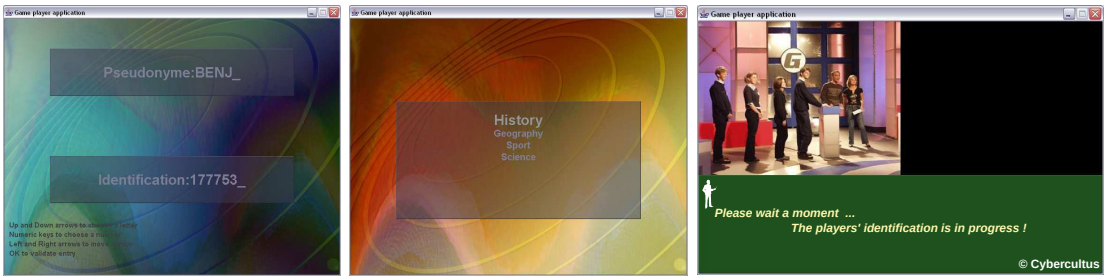


FIG. 9.14 – Interfaces pour l'identification d'un téléspectateur

L'Avatar tentera ensuite d'adopter le rôle sélectionné par l'utilisateur. Si le rôle est déjà joué, l'Avatar viole une contrainte structurelle liée à la cardinalité du rôle et l'utilisateur doit choisir un autre rôle parmi les rôles restant disponibles. Il peut soit annuler sa participation à cette partie, soit choisir un autre rôle. Si le rôle de chef n'est pas encore adopté, le système va lui demander s'il veut le jouer. Si tous les rôles sont déjà joués, il viole par contre la norme N01. Ce choix étant fait, l'interface d'exécution du téléspectateur est affichée sans aucune interaction possible jusqu'à ce que la partie soit démarrée (écran de droite de la FIG. 9.14).

Exécution du show télévisé

Le *QuizMaster* peut, grâce à une interface de *monitoring* exécutable via SACI, visualiser les sociétés et les agents créés au sein de SACI et en sélectionnant une société de type $\mathcal{M}\text{OISE}^{Inst}$ voir l'état de l'organisation. Ainsi sur la FIG. 9.15, nous pouvons constater que tous les agents sont créés (les quatre joueurs ainsi que les Superviseurs), que les groupes sont créés et que le JOUEUR3 a adopté le rôle "History" au sein d'une instance du groupe "Team". Tout au long de la partie, nous pouvons ainsi vérifier les contextes actifs ainsi que les normes actives, les normes respectées et les normes violées. Quand tous les joueurs sont connectés au système, le *QuizMaster* peut démarrer la partie en appuyant sur le bouton "START" de son interface d'exécution (cf. FIG. 9.16). Le *GameMaster* va alors envoyer un message à la société d'agents contenant les événements "beginG" et "avT" pour débuter la partie et faire commencer les Avatars. Le superviseur *ContextManagerAg* va alors changer de contexte et va faire en sorte que les contextes "Game", "Round1" et "MyTurn" soient actifs. De ce fait, les normes allant de N03 à N09 ainsi que les normes N17 et N18 deviennent actives.

La partie étant débutée, une interface du *QuizMaster* (interface de gauche de la FIG. 9.12) va afficher les Avatars des utilisateurs dans le décor choisi au préalable par le producteur. Cette scène est ensuite diffusée et affichée dans la fenêtre en haut à droite de l'interface utilisateur tandis que la fenêtre de gauche (interface de droite de la FIG. 9.16) sert à diffuser le flux vidéo provenant du studio, émis par la chaîne de télévision (l'équipe réelle sur le plateau en l'occurrence). En sélectionnant une question, le *QuizMaster* demande au *GameMaster*, de l'envoyer aux joueurs via leurs Avatars et de l'afficher sur l'écran des joueurs. Ceci étant fait, le *GameMaster* atteint un de ses buts. Les téléspectateurs ont alors 10 secondes pour répondre à la question en sélectionnant une des réponses proposées. Du côté du *QuizMaster*, la réponse choisie par chaque téléspectateur s'affiche dans la cellule "Réponse" de chaque ligne des Avatars.

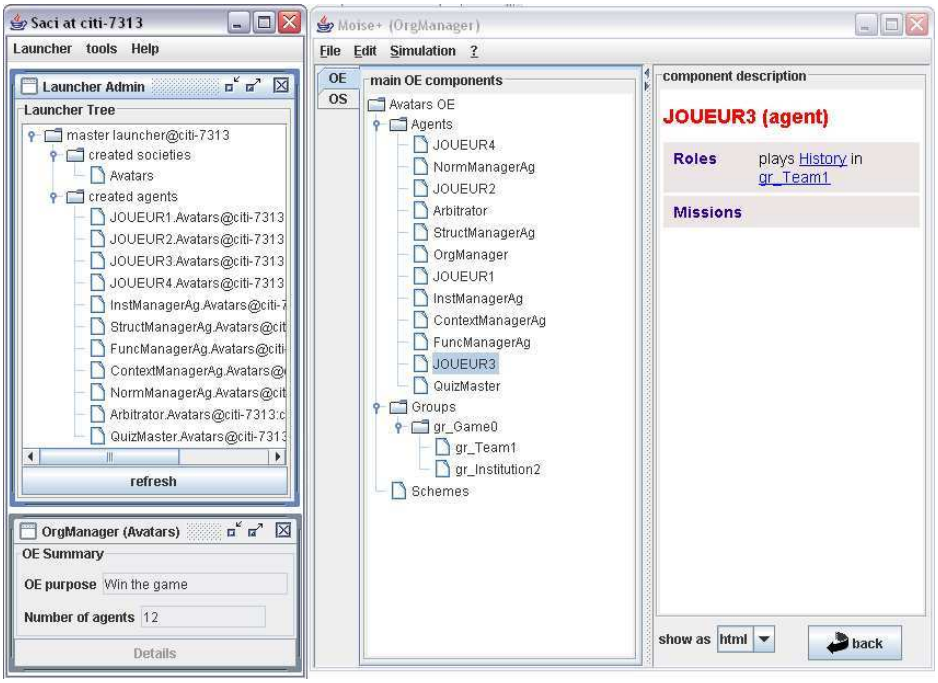


FIG. 9.15 – Interface pour le monitoring de la société SACI

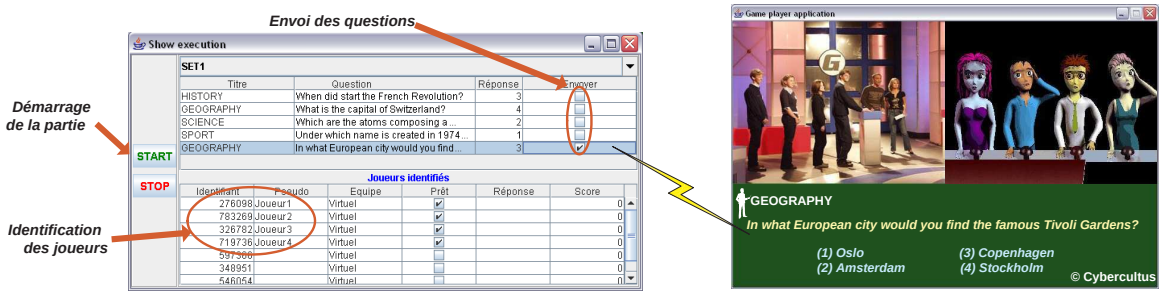


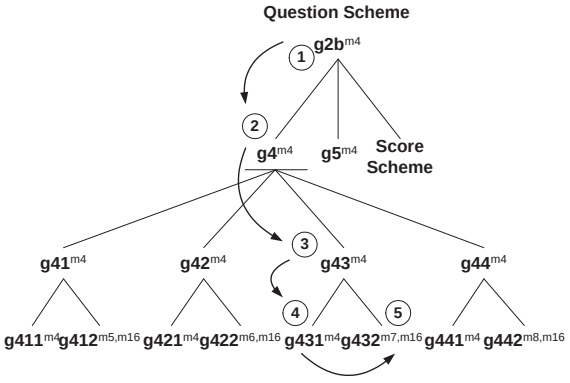
FIG. 9.16 – Interface pour la gestion de la partie

Une fois la réponse évaluée par le *GameMaster* en la comparant avec la réponse juste (colonne "Réponse" des questions), le score est mis à jour pour chacun des joueurs (colonne "Score") et la somme donne le score de l'équipe qui est affiché entre chaque question dans la partie inférieure de l'interface cliente de l'application. Le *QuizMaster* change de manche en sélectionnant un nouvel ensemble de questions (dénommé SET1, SET2 et SET3). En faisant cela, le *QuizMaster* fait en sorte que son Avatar envoie un message contenant l'événement "chgRd" à toute la société d'agents. L'agent de Supervision *ContextManagerAg* en recevant ce message déclenche un changement de contexte et passe donc dans le sous-contexte "Round2". Au niveau des normes, la norme N09 devient inactive et celles allant de N11 à N14 deviennent actives.

Violation de contraintes institutionnelles

Nous nous trouvons maintenant dans le contexte de la deuxième manche où seuls les joueurs ayant le rôle adéquat peuvent répondre aux questions. Le *QuizMaster* veut envoyer une question de Sport. Pour cela l'Avatar *GameMaster* va devoir exécuter une instance du schéma "Question Scheme". La

FIG. 9.17 représente cette instance. Le *QuizMaster* sélectionne, via son interface, la question qu'il veut poser. Cela déclenche chez l'Avatar jouant le rôle de *GameMaster*, conformément à la norme N06, le fait de s'engager sur la mission m4 correspondant à l'étape (1) de la FIG. 9.17. Il choisit donc le but "g2b" pour l'accomplir. Le plan de ce but est composé des sous-buts "g4", "g5" et "g7" (but racine du schéma "Score Scheme"). Son but courant à exécuter est donc "g4" et n'étant pas un but feuille il définit son plan d'action pour atteindre ce but (étape (2) de l'exécution de l'instance de schéma).

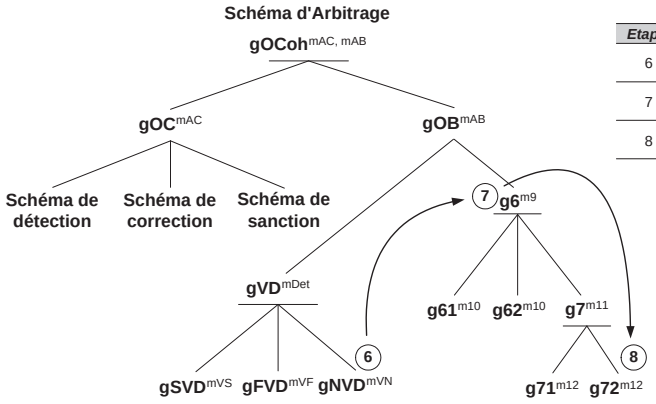


Etape	Normes	Actions
1	N06	commitToMission(m4); chooseGoal(); plan()
2		currentGoal = g4; plan();
3		currentGoal = g43; plan();
4		executeAct(g431)
5	N08; N13	commitToMission(m7); chooseGoal(); plan(); executeAct(g432);

FIG. 9.17 – Fonctionnement de l'Organisation de l'application IDTV - Questions

Le *QuizMaster* a choisi de poser une question de Sport. Son Avatar en est donc informé, de ce fait, la définition du plan du but "g4" équivaut à exécuter le but "g43" (manipuler une question de Sport) qui devient le nouveau but à atteindre du *GameMaster* et dont le plan se compose des buts "g431" et "g432" (étape (3) de l'exécution de l'instance de schéma). Lors de l'étape (4), le *GameMaster* doit exécuter le plan du but "g4" mais n'est autorisé à atteindre que le but "g431". L'exécution de ce but lui fait afficher la question de type Sport aux joueurs afin que ces derniers puissent répondre. L'exécution de la tâche mettra le statut du but "g431" à satisfait. Pour que le but "g43" soit atteint, il faut maintenant que le but "g432" le soit lui aussi. Les Avatars sont en attente d'une action éventuellement demandée par un utilisateur, il faut donc qu'ils choisissent un but à atteindre et pour cela ils attendent que les joueurs les contrôlant les renseignent sur les missions sur lesquelles ils doivent s'engager (en l'occurrence la mission m7). Nous en sommes à l'étape (5) de l'exécution de l'instance de schéma et nous considérons que le joueur dont l'Avatar joue le rôle "Sport" décide de répondre. Les normes N08 et N13 contraignent son action à ce moment là. Il ne doit pas s'engager sur la mission m16 mais peut s'engager sur la mission m7. Son Avatar s'engage donc sur la mission m7 et choisit dans cette mission le but qu'il est possible d'atteindre à savoir "g432". La définition et l'exécution de son plan lui fait exécuter la tâche "g432" se traduisant par l'envoi de la réponse et la demande de changement de statut du but "g432".

Considérons une autre question de Sport et le même enchaînement d'étape jusqu'à la phase (5). Imaginons aussi que ce n'est pas le joueur Sport qui réponde mais le joueur Science. Le joueur ordonne donc à son Avatar jouant le rôle "Science" d'envoyer une réponse. Nous nous trouvons alors dans le cadre du protocole détaillé sur la FIG. 7.10 de la Section 7.5 du Chapitre 7. L'Avatar, via son *Org-WrapperAg* va demander au *FunctManagerAg* de pouvoir s'engager sur la mission m7 afin de répondre à la question (donc d'atteindre le but "g432"). A la réception de ce genre de message, le Superviseur, en charge de la supervision du fonctionnement de l'Organisation, va demander au *NormManagerAg* de vérifier si une norme autorise cela. Or à cette étape, les normes N08 et N13 sont à prendre en compte et aucune n'autorise un Avatar jouant le rôle "Science" à accomplir la mission m7. L'engagement sur la mission est donc refusé et un message de violation de norme est envoyé au *InstManagerAg*.



Etape	Normes	Actions
6	NA12	setGoalSatisfied(gNVD); msg(NSViolated)
7	NA07; N18	chooseGoal(); currentGoal = g6; plan(); currentGoal = g7; plan()
8	N18	commitToMission(m12); chooseGoal(); plan(); executeAct(g72)

FIG. 9.18 – Fonctionnement de l'Organisation de l'application IDTV - Supervision

Comme illustré sur la FIG. 9.18 représentant la partie de Supervision du fonctionnement de l'Organisation des Avatars, l'agent de Supervision qui devait atteindre le but "gNVD" (détection d'une violation normative) devra maintenant exécuter la suite du plan de "gOB" consistant à garder une organisation cohérente par blocage des actions de violation. L'étape (7) consiste donc, sous l'influence de la norme institutionnelle NA12, à exécuter le schéma de sanction défini pour le scénario des Avatars. Le but racine de ce schéma possède un plan permettant de faire un choix entre plusieurs sanctions possibles. Dans notre cas, la norme N18 définie par l'utilisateur pour ce jeu, oblige l'Avatar *GameMaster* à agir sur le score du joueur. Cette action, en étape (8) consiste aussi au choix entre augmenter ou diminuer le score. La diminution est choisie et le but "g72" est atteint.

La suite de l'exécution du fonctionnement de l'Organisation permet aux joueurs d'atteindre les buts autorités à chaque question en donnant une réponse à la question et aux superviseurs en détectant et en sanctionnant les violations. Le *GameMaster* finit par atteindre le but "g2a" quand toutes les questions ont été posées. C'est également à ce moment là que l'événement "endG" est envoyé parce que le *QuizMaster* l'aura décidé. L'Organisation entrera alors dans le contexte "End" et permettra aux joueurs de quitter l'équipe et donc de se déconnecter du serveur.

Durant toute la durée de la partie, les Avatars ont l'occasion de choisir le but "g8". Ce but leur permet de montrer leurs émotions en montrant un visage triste ou bien en affichant un visage heureux. Ces buts sont étroitement liés aux capacités de la composante multimédia des Avatars. Pour renforcer l'autonomie (un objet multimédia autonome ne devrait *a priori* pas attendre la demande d'un utilisateur pour adapter son comportement graphique), les méthodes de comportement de l'objet multimédia et l'exécution des tâches "g81" et "g82" devraient être plus liées qu'elles ne le sont actuellement. Une sémantique plus poussée devrait pouvoir faire le lien entre la définition du vocabulaire des buts, contextes, événements et actions et une réelle action au niveau de chaque Avatar, qu'elle soit appelée par la composante agent ou la composante multimédia. Ce n'est pas le cas pour le moment. Nous considérons donc qu'une fois la réponse donnée, l'utilisateur peut choisir si son Avatar exhibe une émotion ou non. S'il le fait alors que c'est l'autre équipe qui répondait, et que l'Avatar montre un visage heureux, alors la norme N16 est violée et la sanction N17 est appliquée.

9.5.2 Validation

Grâce à $\mathcal{M}$OISE^{Inst}, nous pouvons définir un modèle organisationnel contraignant des entités autonomes selon quatre axes. La Spécification Structurelle nous permet de structurer les différents acteurs en membres d'une équipe, en candidats potentiels pour faire partie de l'équipe et en gestionnaire de la partie. Différents rôles sont créés au sein de l'équipe. La structure en groupes et en rôles nous permet de contraindre les Avatars en définissant une hiérarchie, en pouvant appliquer des cardinalités pour

le nombre limite de participants et en pouvant appliquer des normes différentes pour chacune des entités structurelles. La Spécification Fonctionnelle permet de définir clairement les buts à atteindre au sein de l'organisation et la façon de les atteindre. Ces buts clairement identifiés et regroupés en missions nous permettent de les normer. La Spécification Contextuelle rend possible la définition d'un enchaînement de contextes, lesquels liés aux entités structurelles et aux missions, nous permettent de définir un scénario guidant le comportement des acteurs de la scène multimédia que nous voulons contrôler. La spécification de l'Organisation nous permet en effet d'extérioriser les relations entre agents et le fonctionnement dans l'Organisation des objets multimédia initiant ainsi une autonomie et une réutilisabilité dans d'autres scènes.

En définissant un système de supervision à l'aide du même modèle $\mathcal{M}$OISE^{Inst}, la détection et surtout la sanction sont facilitées. En effet, il est possible avec la composition des schémas de faire en sorte que le schéma d'arbitrage prenne en compte le schéma de sanction défini par l'utilisateur. Nous fournissons ainsi le même système d'arbitrage pour tous les modèles mais nous laissons la possibilité à l'utilisateur de personnaliser les sanctions à appliquer. Ici, les sanctions adéquates sont d'agir sur le score ou d'éjecter le joueur de l'équipe. Ces schémas sont définis par l'utilisateur mais seront exécutés par les superviseurs (ou tout du moins initiés).

Le modèle $\mathcal{M}$OISE^{Inst} ne permet pas de définir une spécification des interactions entre les Avatars contrairement à d'autres modèles. Le comportement des Avatars se résume à agir sur l'Organisation via leur *OrgWrapper* et est supervisé par $\mathcal{S}$YNAI. Une spécification des interactions n'est donc pas dans ce cas nécessaire. La dynamicité de l'organisation se retrouve dans l'enchaînement des contextes et les normes qui leurs sont liées.

9.6 Synthèse

Dans ce chapitre nous avons présenté l'utilisation de MA$\mathcal{B}_{\text{ELI}}$ au sein d'une application de télévision interactive. Cette application consiste en un jeu télévisé de "question – réponse" opposant une équipe de joueurs réels à une équipe de téléspectateurs représentés par des Avatars. Le méta-modèle $\mathcal{M}$OISE^{Inst} nous a permis de définir les règles du jeu en spécifiant le comportement individuel et collectif d'un ensemble d'objets multimédias autonomes au sein d'une institution. Ces objets multimédias sont composés d'une partie agent héritant de la classe `OrgAgent` de SACI. Leur cycle de vie est quasiment le même que celui des agents de $\mathcal{S}$YNAI. Leur méthode d'exécution fait une correspondance entre un but à atteindre et une action à mettre en œuvre pour atteindre ce but.

Nous distinguons deux types d'Avatar, les joueurs assistant les téléspectateurs et le *GameMaster* assistant le directeur du jeu (le *QuizMaster*). Le *GameMaster* a la particularité de faire partie des agents du domaine, donc sous l'autorité des agents institutionnels, en ayant lui aussi l'autorité sur tous les autres agents du domaine. Ce rôle particulier lui donne le droit de participer à la supervision de l'Organisation via la mise en œuvre des sanctions définies spécialement pour cette application. Les agents génériques de $\mathcal{S}$YNAI ne peuvent pas connaître le processus lié à ces sanctions particulières. Malgré tout, la supervision d'une partie se fait d'une façon centralisée et globale et se déroule convenablement.

Nous venons donc d'illustrer le fait de pouvoir contraindre et superviser un ensemble d'agents organisés et structurés à l'aide d'une seule institution. Afin de vérifier maintenant que MA$\mathcal{B}_{\text{ELI}}$ n'est pas liée au domaine du multimédia en particulier, que les agents de $\mathcal{S}$YNAI supervisent de la même manière un ensemble d'agents soumis à différentes institutions, nous utilisons notre modèle d'Institution Électronique dans un autre domaine d'application. Nous allons donc maintenant valider MA$\mathcal{B}_{\text{ELI}}$ avec son utilisation dans une application de commerce électronique.

Chapitre 10

Arbitrage pour la Gestion de Contrats Électroniques

Après avoir validé notre modèle MA$\mathcal{B}_{\text{ELI}}$ dans le contexte d'une application mettant en scène plusieurs agents au sein d'une seule institution, nous étudions l'intégration de ce même modèle dans le contexte d'une application de gestion de contrats électroniques utilisant deux (ou trois) agents au sein de plusieurs institutions différentes. Les institutions représentant des contrats électroniques différents, nous allons donc avoir un petit nombre d'agents impliqués dans un grand nombre d'institutions.

Le domaine des services électroniques et notamment celui des contrats électroniques en ligne connaît un intérêt grandissant de la part des entreprises avec l'apparition et le développement des entreprises virtuelles sur Internet. Certaines applications B2B (*Business to Business*) se contentent de digitaliser les contrats papier et de les signer numériquement (thématique du *e-contracting*) [69]. D'autres dont le nombre est de plus en plus important ajoutent aux clauses des contrats de l'information compréhensible par des machines afin d'améliorer la manipulation de ces contrats électroniques [66].

Dans cet esprit, le paradigme des Systèmes Multi-Agents a été introduit afin d'atteindre un niveau d'automatisation dans la création, l'exécution et la gestion des contrats électroniques par des agents au nom des utilisateurs. Les agents jouent un rôle d'assistant gérant les contrats car ils ne peuvent bien entendu pas produire le service concerné par le contrat. Ils ne peuvent pas non plus signer légalement un contrat. Considérant les agents comme des gestionnaires de contrats, certains travaux traitent de la négociation automatique des questions liées au contenu (certains éléments compris dans les clauses, comme le prix ou le délai par exemple) et à leur exécution [28, 80]. Les contrats produits de cette façon consistent alors en un accord digital dans lequel les droits et les devoirs en terme de livrable de chaque partie contractuelle ainsi que les coûts et les délais imposés aux participants du contrat sont explicitement représentés. Ces droits et devoirs sont imposés mais non modifiables en cours d'exécution. De tels contrats aboutissent souvent à des relations inflexibles entre les participants et sont dans ces cas contraires aux exigences des nouveaux paradigmes métier visant à améliorer la compétitivité des entreprises à l'aide des entreprises virtuelles dynamiques [53]. En matière de flexibilité, la possibilité de pouvoir agir sur une spécification de contrat au même titre que sur le contrat lui-même lors de son exécution est une thématique abordée par les travaux sur $\mathcal{M}$OISE$^+$ [59] traitant de la ré-organisation.

Dans ce chapitre nous présentons donc une application de commerce électronique nommée EBSME[1] pour laquelle notre objectif est d'y intégrer un modèle de contrat électronique basé sur $\mathcal{M}$OISE^{Inst} et dont les instances seront gérées par $\mathcal{S}$YNAI. Dans la section suivante, nous présentons en détail la

[1] EBSME : Electronic Business for the Small and Medium Enterprises

spécification d'un contrat à l'aide de $\mathcal{M}$OISE^{Inst}. Nous abordons ensuite l'implémentation des assistants qui auront à interpréter les contrats (les agents qui auront à agir sur les Organisations) ainsi que l'intégration de l'API globale de MA$\mathcal{B}_{\text{ELI}}$. Ces différents éléments nous permettent de définir une plate-forme de commerce électronique. Nous présentons un exemple d'utilisation de cette plate-forme afin d'expérimenter et de valider l'utilisation de notre modèle d'Institution Électronique dans le domaine du *e-Commerce*.

10.1 Motivations

Les consommateurs de services via Internet (comptabilité, traduction, transcription, etc...) ont des critères de choix qui sont en général la rapidité, la qualité du service, le prix et surtout le degré de sécurité dans lequel les transactions s'effectuent. Ces exigences engendrent une difficulté au niveau des PME voulant se lancer dans le domaine des *e-services* puisque celles-ci, dans la plupart des cas, ont un manque de connaissance technique et n'ont souvent pas l'infrastructure logicielle adaptée. C'est pour répondre à ces attentes que la plate-forme de eBusiness sécurisée EBSME (Electronic Business for the Small and Medium Enterprises) [65] construite à base de composants *Open Source* et permettant la gestion et l'exécution de contrats électroniques a été proposée.

10.1.1 Présentation globale de l'application

La plate-forme EBSME est une application web écrite en JAVA et utilisant la plate-forme J2EE (Java Platform, Enterprise Edition) suivant une architecture trois tiers (nous reviendrons sur les spécificités de son architecture dans la section 10.3.1) permettant à un ensemble d'utilisateurs de s'inscrire et de pouvoir, dans un cadre sécurisé, manipuler un ensemble de contrats électroniques. Selon [48], les différentes phases qui régissent la gestion d'un contrat électronique sont les suivantes :

- Une phase d'enregistrement (ou phase pré-contractuelle) lors de laquelle les clients et les fournisseurs identifient les produits/services qu'ils cherchent à vendre ou à acheter et diffusent leurs catalogues.
- Une phase de création (ou phase contractuelle) où une relation formelle est créée entre un acheteur et un vendeur. Cette phase englobe la négociation du contrat. A l'issue de la négociation, la signature du contrat peut s'effectuer.
- Une phase d'exécution du contrat (ou phase post-contractuelle) pendant laquelle les biens et les services sont délivrés et validés (une tierce partie digne de confiance peut éventuellement intervenir durant cette phase en cas de conflit).

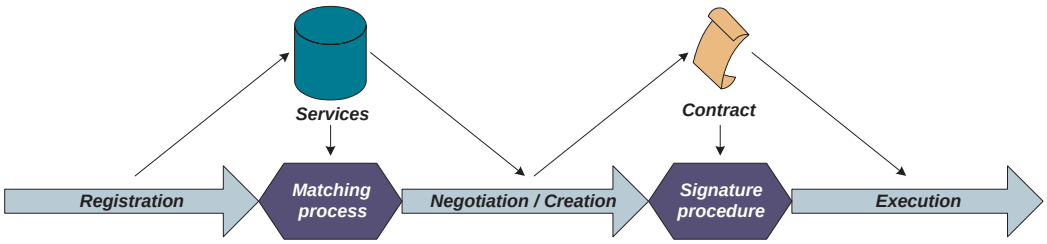


FIG. 10.1 – Modules pour le support du management de contrats

L'application EBSME respecte ces différentes phases comme illustré sur la FIG. 10.1. Les utilisateurs peuvent en effet s'inscrire à la plate-forme et créer un profil en spécifiant les compétences qu'ils offrent et celles qu'ils recherchent. Ces compétences concernent dans notre cas des capacités à comprendre et à traduire certains langages. Elles sont stockées dans une base de services sur laquelle des traitements peuvent être effectués afin que chaque utilisateur puisse trouver des profils lui correspondant dans ses recherches de services. Ainsi un utilisateur donneur d'ordre (employeur) ayant besoin d'effectuer des services se mettra en contact avec un utilisateur exécutant (employé) ayant

les compétences adéquates. Ils pourront négocier les clauses d'un contrat électronique qui sera créé et signé afin de permettre au fournisseur de service de satisfaire les besoins d'un client. Ces rôles sont définis dans le contrat électronique auquel est ajouté un troisième utilisateur qui servira de juge ou d'arbitre en cas de conflit entre les deux principales parties contractantes. L'employé s'engage à exécuter un service pour l'employeur défini en termes de livrables (les éléments à traduire à savoir un tome, un chapitre, etc.), de coût (l'employé est-il payé à la fin du contrat ou par livrables fournis ?) et de délais (dans combien de temps les livrables doivent-ils être livrés ?). Une fois le contrat signé par les trois utilisateurs, il doit être exécuté en respectant les clauses.

La plate-forme EBSME est donc le moyen de coordonner l'enregistrement, la négociation et la création d'accord sous forme de contrats électroniques, ainsi que leur signature et leur exécution.

Afin d'améliorer cette application en ajoutant plus d'autonomie dans l'exécution et la gestion des différents contrats électroniques, nous aimerions faire en sorte que chaque utilisateur puisse être assisté par une entité logicielle à laquelle il peut déléguer une part de son contrôle sur les contrats. Ainsi, les assistants pourraient, sous la supervision et le paramétrage de l'utilisateur, proposer de manière pro-active des contrats basés sur des *templates* définis par l'utilisateur tout en essayant de respecter les contrats courants actifs. Nous proposons pour cela de construire les assistants avec des agents autonomes et de les contraindre à l'aide du modèle d'organisation normatif $\mathcal{M}\text{OISE}^{Inst}$ représentant ainsi un contrat électronique. La supervision par un système d'arbitrage évitera une gestion de contrats éphémères.

10.1.2 Objectifs

Afin de répondre aux motivations énoncées ci-dessus, nous pensons que l'intégration du modèle $\text{MA}\mathcal{B}_{\text{ELI}}$ au sein d'EBSME est une solution adéquate. Afin de pouvoir utiliser des agents en tant qu'assistants, nous devons enrichir et formaliser le modèle de contrat électronique utilisé dans EBSME pour que les assistants des utilisateurs puissent interpréter et raisonner sur les contrats. Nous devons également ajouter un système de supervision permettant de contrôler l'exécution des contrats et de détecter leur non-respect étant données l'autonomie des assistants et leur capacité à proposer des alternatives aux contrats en cours afin de répondre aux objectifs des utilisateurs (délais, coût, etc.).

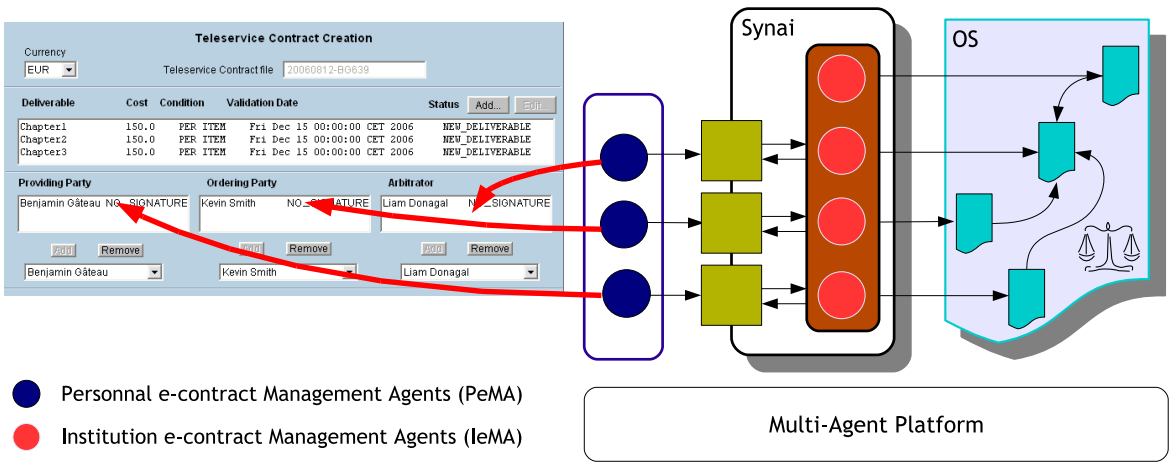


FIG. 10.2 – Vue globale de l'organisation multi-agent pour la gestion de contrats électroniques

Ainsi et comme illustré sur la FIG. 10.2, l'objectif est de considérer EBSME comme un Système Multi-Agents organisé en quatre couches : (i) une couche de *Gestion de Contrats Électroniques* servant d'interface homme-machine et via laquelle l'utilisateur interagit avec le système pour gérer ses contrats (sur la figure, l'interface représentée est une partie de l'interface de création de contrat), (ii) une couche

d'*Exécution de Contrats Électroniques* représentant et gérant l'exécution des contrats électroniques au nom de leurs parties représentées, (iii) une couche de *Supervision de Contrats Électroniques* qui supervise et contrôle l'exécution des contrats électroniques, (iv) une *Plate-Forme d'Exécution Agent* qui gère la distribution et les échanges de messages entre les agents. Les contrats que les assistants des utilisateurs devront respecter et sur lesquels le système de Supervision se basera pour contrôler l'activité des assistants sont spécifiés à l'aide de $\mathcal{M}$OISE^{Inst}.

Les deux couches centrales seront implémentées avec deux types d'agent :
- dans la couche d'exécution des contrats, des agents personnels de gestion des contrats électroniques (nommés PeMA pour *Personnal e-contract Management Agents*) consultent les contrats et agissent sur les livrables au nom de leur utilisateur.
- dans la couche de supervision des contrats, des agents institutionnels de gestion des contrats électroniques (nommés IeMA pour *Institution e-contract Management Agents*) supervisent et contrôlent l'action des PeMA sur les contrats électroniques.

Il est important de noter que les IeMA sont représentés par le middleware $\mathcal{S}$YNAI et donc les agents étant génériques, ce sont les mêmes que ceux présentés dans la Section 7.3 du Chapitre 7, ce sont les mêmes que les agents de Supervision utilisés dans l'application de iDTV du chapitre précédent.

Les PeMA jouent les rôles d'employeur ou d'employé et doivent de ce fait atteindre les buts exprimés dans leurs contrats actifs. Sur la FIG. 10.2 nous voyons que les entités PeMA font d'un côté partie du Système Multi-Agents et de l'autre sont exécutés dans le système et affichés sur l'interface. Ici, nous pouvons constater les rôles que les assistants des utilisateurs "Gâteau", "Smith" et "Donagal" joueront au sein du contrat électronique "20060812-BG639". Ils interagissent les uns avec les autres pour se coordonner et gérer l'exécution de leurs contrats courants. Chaque PeMA appartenant à un et un seul utilisateur a pour but d'agir sur les contrats selon les ordres de son utilisateur, le tenir au courant de l'évolution de l'état des contrats auxquels il participe, l'alerter à l'approche des dates limites auxquelles il est soumis en tant que participant au contrat et lui donner des conseils sur les décisions que l'utilisateur doit prendre quant à la réalisation ou non d'une partie ou de l'entièreté du contrat.

Les contrats étant définis grâce au modèle $\mathcal{M}$OISE^{Inst}, nous faisons ici aussi la distinction entre la spécification du contrat et l'instanciation du contrat. La spécification est un modèle de contrat qu'il faut instancier avec ses propres valeurs pour définir un contrat utilisable par les assistants des utilisateurs. Une spécification de contrat faîte à partir d'une OS $\mathcal{M}$OISE^{Inst} est un *template* de contrat. Une instance d'Organisation $\mathcal{M}$OISE^{Inst} au sein de laquelle les assistants jouent les rôles d'employeur et d'employé est un contrat actif. Les concepteurs et administrateurs de la plate-forme mettent à disposition des utilisateurs certains templates de contrat que ces derniers pourront instancier avec leurs propres valeurs afin de pouvoir ensuite les signer et les exécuter.

Nous avons vu globalement le principe de l'application EBSME et les grandes lignes que nous suivrons afin d'y intégrer notre modèle d'Institution Électronique. Nous allons donc voir maintenant la façon dont un contrat est spécifié à l'aide de $\mathcal{M}$OISE^{Inst} et comment il sera instancié pour être exécuté.

10.2 Spécification des contrats sur base de $\mathcal{M}$oise^{Inst}

Un contrat définit explicitement les participants et les services d'une part et l'explication des processus métier qui doivent être mis en place pour remplir le contrat d'autre part. Les processus métier regroupent le fonctionnement du contrat ainsi que les différentes obligations et permissions relatives au fonctionnement que les utilisateurs doivent respecter pour être conformes aux contrats.

Nous formalisons une instance de contrats comme suit : un identifiant unique (:id), un en-tête (:header), un ensemble de conditions générales (:conditions), une structure (:Struct), un fonctionnement (:Funct) et des normes actives (:ActNorms) :

⟨Contract⟩ : := '(' Contract :id ⟨cId⟩ :header ⟨header⟩ :conditions ⟨condition⟩* :Struct ⟨Struct⟩ :Funct ⟨Funct⟩ :ActNorms ⟨ActNorms⟩ ')'

La structure, le fonctionnement et les normes d'un contrat sont les instances d'une spécification faite avec $\mathcal{M}$OISE^{Inst}. En effet, la plate-forme EBSME propose aux utilisateurs des *templates* de contrat ne spécifiant que ces trois composants. L'établissement d'un contrat électronique sur base d'un template de contrat se fait via la définition de son en-tête et de ses conditions générales qui vont définir un contexte d'exécution. Ces deux ensembles instancient les spécifications structurelles, fonctionnelles et normatives qui décrivent les éléments du template de contrat pour obtenir une structure (l'organisation des participants), un fonctionnement (les clauses du contrat) et finalement des normes actives (les méthodes d'exécution du contrat).

10.2.1 Entête d'un contrat électronique

L'en-tête (:header) spécifie les participants et les services concernés par le contrat. Les informations sont renseignées par l'utilisateur via l'interface web de EBSME. La formalisation en BNF d'un entête de contrat est :

⟨header⟩ : := '(' :participant ⟨participant⟩* :service ⟨service⟩* ')'

Un en-tête de contrat consiste ainsi en : (i) une liste de couples d'identifiant d'agents et de rôles venant de la SS faisant ainsi la correspondance entre les agents et le rôle qu'ils jouent dans l'Organisation, i.e dans le contrat (:participant), (ii) une liste d'identifiants de services et d'attributs concernés par le contrat. Les services apparaissent comme des buts au sein de la FS comme partie du processus métier du contrat (:service). La valeur des attributs de service a été définie durant la négociation du contrat.

⟨participant⟩ : := '(' ⟨idAgt⟩ ⟨roleId⟩ ')'*
⟨service⟩ : := '(' ⟨idS⟩ ⟨typeS⟩ ⟨issue⟩* [⟨property⟩*] ')'
⟨issue⟩ : := '(' ⟨IssueName⟩ ⟨value⟩')'
⟨property⟩ : := ⟨object⟩ | ⟨address⟩ | ...

Si nous reprenons l'exemple du contrat illustré par l'interface de la Fig. 10.2, son entête définit que le contrat est passé entre les agents assistant les utilisateurs "Gâteau", "Smith" et "Donagal", jouant respectivement les rôles d'employeur, d'employé et d'arbitre. Le service concerné par ce contrat est le service de traduction "20060812-BG639" de type 'delivery' dont les livrables (⟨issue⟩) sont "Chapter1", "Chapter2" et "Chapter3" ayant chacun une valeur de 150 euros.

Une fois l'en-tête défini, le contrat est ensuite basé sur un cœur organisationnel (template de contrat électronique) du contrat représenté par une OS de type $\mathcal{M}$OISE^{Inst}. Le vocabulaire sur lequel l'OS se base est mince puisque nous n'avons pas de CS, donc pas de contexte ni d'événement. Nous avons quelques buts dont nous donnons la signification dans la section 10.2.3.

10.2.2 Spécification Structurelle

La SS d'un template de contrat électronique est représentée graphiquement par la Fig. 10.3. Le groupe racine est le groupe "Contract". Il est composé des trois rôles "Employer", "Employee" et "DomArbitrator" pouvant chacun être joué par un agent au plus (cardinalité de "1..1"). Ces trois rôles sont compatibles c'est-à-dire qu'un assistant pourra jouer les trois rôles en même temps, mais la portée de ces liens est 'inter-groupe' ce qui signifie que les assistants peuvent jouer ces trois rôles en même temps, mais pas dans la même instance de groupe (pas dans la même instance de contrat). Un utilisateur peut, en effet, être engagé sur plusieurs contrats en même temps en tant qu'employeur ou employé mais ne peut pas jouer ces deux rôles au sein du même contrat.

D'autres liens contraignent les rôles mais avec une portée intra-groupe cette fois-ci (donc au sein d'un même contrat). Le rôle "DomArbitrator" a l'autorité sur les rôles "Employer" et "Employee"

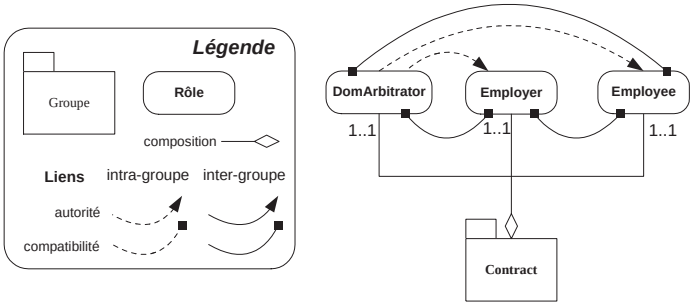


FIG. 10.3 – Spécification Structurelle d'un template de contrat électronique

permettant ainsi un contrôle de l'assistant jouant ce rôle sur les assistants jouant les deux autres rôles. Nous rappelons qu'un lien d'autorité implique un lien de communication.

La structure des participants au contrat étant définie, voyons les fonctions que ces rôles auront à mettre en œuvre (les processus métier à réaliser pour exécuter le contrat) grâce à la Spécification Fonctionnelle.

10.2.3 Spécification Fonctionnelle

La FS d'un template de contrat est basée sur un ensemble de buts dont voici la description :

Goal	
gAllDelExecuted	Tous les livrables sont exécutés, le contrat est terminé
gDelTerm	Le livrable est terminé
gSubmitted	Le livrable est soumis
gValidated	Le livrable est validé
gPaid	Le livrable est payé
gSancApplied	La sanction est appliquée
gRepDecr	La réputation est baissée
gPayArb	La sanction financière est payée
gRollback	La correction est appliquée

La FS d'un template de contrat est représentée graphiquement par la FIG. 10.4. Elle spécifie les tâches globales qui sont concernées par les contrats électroniques instanciant ce template. Ces tâches composent différents schémas et sont regroupées en missions qui devront être accomplies par les assistants des utilisateurs.

- Au sein du "Schéma d'exécution de contrat" La mission "mUser" concerne tous les participants au contrat. Elle est composée du but racine "gAllDelExecuted". Ce but est atteint quand tous les livrables sont exécutés.
- La réalisation d'un livrable correspond à l'exécution d'une instance du "Schéma d'exécution de livrable". Ce schéma a pour but racine le but "gDelTerm" qui est atteint quand le livrable est effectivement terminé. Pour terminer un livrable, il faut atteindre les sous-buts "gSubmitted", "gValidated" et "gPaid" composant le plan du but racine. Ces sous-buts spécifient qu'un livrable est terminé quand il a été soumis, validé et payé. Ce schéma ne définit pas le processus métier mais la procédure commune à l'employeur et à l'employé pour tout type de contrat. Ainsi, les deux parties contractuelles du contrat doivent se partager l'exécution du livrable, d'où la séparation des buts en deux missions "mEe" pour les buts concernant l'employé et "mEr" pour les buts concernant l'employeur.
- La mission "mArb" concerne l'arbitre du contrat (le tiers de confiance). Elle est composée de tous les buts du "Schéma de sanction". Le schéma de sanction est composé du but racine

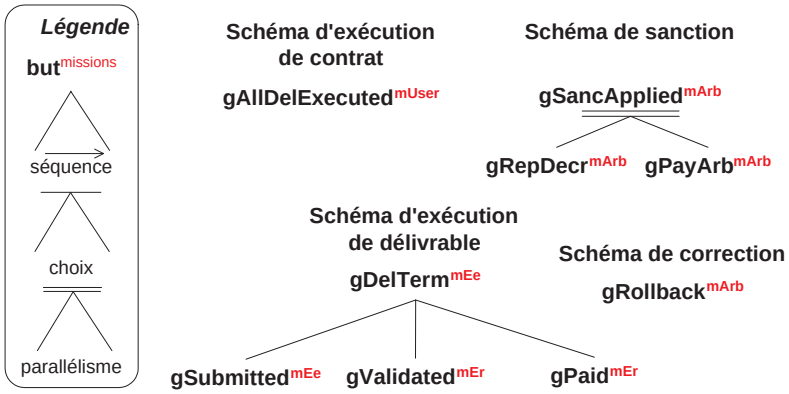


FIG. 10.4 – Spécification Fonctionnelle d'un template de contrat

"gSancApplied" qui est atteint quand une baisse de la réputation du fautif est appliquée (but "gRepDecr") et qu'une amende est payée (but "gPayArb").
- La mission "mArb" est également définie dans le schéma de correction et est composée du seul but "gRollback". De la même façon que le schéma de sanction définit les sanctions à appliquer, ce schéma définit les corrections à apporter à l'organisation en cas de détection d'une violation (cf. Section 7.4.2).

Cette spécification fonctionnelle est commune à tous les contrats qui seront instanciés. Plus globalement, elle permet d'exécuter de façon générique tout type de contrat électronique puisqu'elle ne décrit pas précisément de processus métier. Les clauses définies grâce à une NS se contentent également de spécifier la procédure commune. Avant de les détailler, il est à noter que nous avons trouvé inutile de définir une CS (optionnelle au sein de $\mathcal{M}$OISE^{Inst}) car dans le cas présent, la dynamique du contrat n'en nécessite pas.

10.2.4 Spécification Normative

Nous considérons la NS comme la définition des clauses du contrat. Nous spécifions ici ce que doivent et ne doivent pas faire les participants au contrat. Une clause est une norme concernant un rôle du contrat par rapport à une mission du contrat.

context	id	weight	condition	issuer	bearer	deOp	mission	deadline	sanction
---	N01	1	---	Supervisor	Employer	O	mUser	< *end_contract*	N05
---	N02	1	---	Supervisor	Employee	O	mUser	< *end_contract*	N05
---	N03	1	---	Supervisor	Employee	O	mEe	< *end_del*	N05
---	N04	1	---	Supervisor	Employer	O	mEr	< *end_del*	N05
---	N05	1	violated(?N)	Supervisor	DomArbitrator	O	mArb	---	---

FIG. 10.5 – Spécification Normative d'un template de contrat

La NS de l'application est représentée par la FIG. 10.5. Elle permet de définir les droits et les devoirs des participants au contrat pour l'exécution :
- ▷ Les rôles "Employee" et "Employer" doivent exécuter tous les livrables du contrat (normes N01 et N02).
- ▷ Le rôle "Employee" a l'*Obligation de soumettre un livrable sur la plate-forme avant la fin du temps imparti* que nous traduisons par la norme N03.
- ▷ Le rôle "Employer" a l'*Obligation de valider le livrable soumis par l'employé et de le payer dans la limite de temps défini pour ce livrable* que nous traduisons par la norme N04.

Si les normes ci-dessus ne sont pas respectées, le troisième participant au contrat doit intervenir :
▷ La sanction N05 a comme condition le fait qu'une des normes précédentes (allant de N01 à N04) soit violée. *Si une de ces normes est violée, alors le rôle "DomArbitrator" a l'Obligation d'appliquer une sanction*, c'est-à-dire d'atteindre les buts "gRepDecr" et "gPayArb". Aucune contrainte de temps n'est définie pour cette norme.

Comparée à la NS de l'application iDTV du chapitre précédent, la NS ci-dessus comporte moins de normes et seulement des Obligations. Ceci s'explique par la simplicité de la SS et de la FS ainsi que l'absence de CS. Il y a donc potentiellement moins de combinaisons à faire entre les rôles, les missions et les contextes. Les normes sont peu nombreuses et restent très générales car il est difficile de spécifier des règles précises pour une instance de contrat dans un *template*. En proposant une série de *templates* plus spécifiques, nous pourrons par exemple faire la différence entre les contrats relatifs à un service et à un produit. Puis, nous pourrons préciser les services en fonction de s'ils sont de traduction, graphiques, etc. Plus on détaillera le type de contrat, plus il sera facile d'avoir une FS précise proche du processus métier. On pourra même laisser la possibilité aux utilisateurs de définir leurs propres spécifications de contrat. Ainsi, la définition des normes pourrait être également plus précise en mettant en relation les rôles de chacune des parties avec les missions de la FS afin de définir les clauses du contrat.

Maintenant que nous avons défini un modèle de contrat, voyons comment les assistants vont pouvoir l'instancier pour créer de vrais contrats exécutables et comment ces assistants sont eux-même implémentés pour pouvoir à la fois assister l'utilisateur dans ses prises de décision et également gérer l'exécution de ses différents contrats. Nous allons voir cela dans la section suivante traitant de l'implémentation de EBSME.

10.3 Implémentation d'EBSME

Afin de pouvoir intégrer des assistants ayant une composante agent dans l'architecture EBSME existante, il faut bien entendu étudier en premier lieu cette architecture. Nous allons donc dans cette section présenter l'architecture d'EBSME, puis l'intégration des agents au sein de la plate-forme et enfin l'implémentation des assistants basée sur les agents.

10.3.1 Architecture d'origine d'EBSME

L'application d'EBSME est construite selon une architecture 3-tiers classique utilisant le *framework* J2EE. Comme illustré sur la Fig. 10.6 les utilisateurs accèdent aux informations stockées dans la base de données via une interface Web construite sur une applet JAVA. Les communications entre le client et le serveur utilisent le protocole SOAP et une connexion SSL. La gestion des informations (contrôle de la consistance, etc ...) se fait par des EJB (Enterprise Java Bean). Enfin les informations concernant les utilisateurs, les compétences qu'ils recherchent ou qu'ils proposent, les contrats en cours, etc. sont stockés dans une base de données relationnelle.

Le serveur d'application J2EE (ou JEE pour *Java Enterprise Edition* depuis sa version 5) est un modèle de gestion de transactions pour composants qui permet la définition et l'implémentation d'EJB. Les EJB sont des modèles de composants se trouvant côté serveur. Écrits en Java, ils sont portables, réutilisables et déployables et encapsulent des données (*entity bean*) et les traitements (*session bean*) de données . Ils s'exécutent dans un conteneur EJB qui va leur fournir des services tels que les transactions ou la persistance. D'autres informations concernant les EJB ainsi que tous les EJB utilisés au sein de EBSME sont disponibles dans l'Annexe D.

Les utilisateurs ont leurs principales informations représentées par un EJB. C'est sur cet EJB qu'il va falloir agir pour ajouter une composante agent capable de s'organiser et d'agir de manière autonome. De même, les informations d'origine des contrats sont représentées par l'EJB "Contract". Il faut également ajouter les informations concernant l'Organisation $\mathcal{M}$OISE^{Inst} afin d'obtenir notre

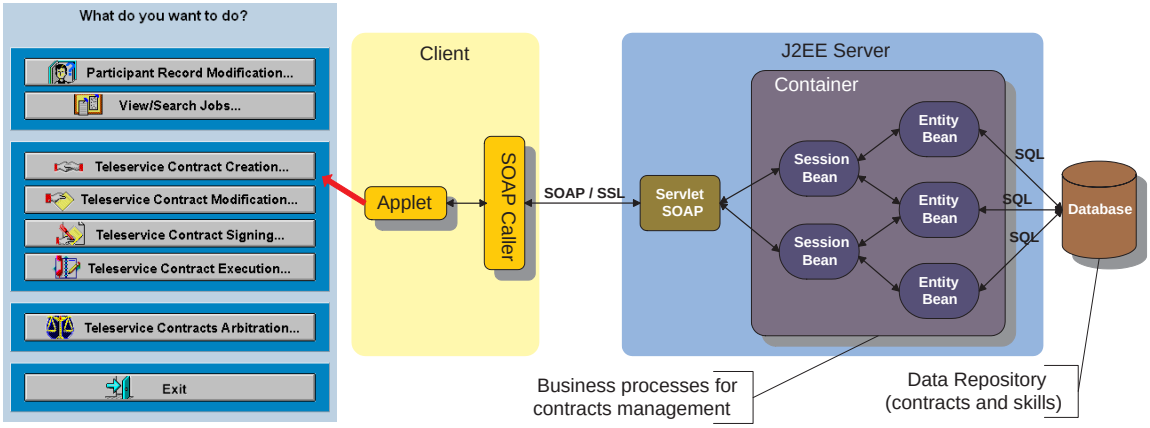


FIG. 10.6 – Architecture d'EBSME

représentation enrichie des contrats. De plus, il faudra agir sur les "ContractController" afin qu'ils puissent également superviser les modifications à apporter sur la partie organisationnelle des contrats. Voyons comment intégrer notre modèle basé sur un SMA au sein de la plate-forme J2EE.

10.3.2 Intégration du Système Multi-Agents

Les fonctionnalités de la plate-forme J2EE en matière de transactions permettent un arbitrage avec correction (cf. les stratégies d'arbitrage de la Section 7.4.3). En effet, un processus de correction implique la nécessité de revenir dans l'état antérieur cohérent. Si l'organisation d'agents représentant un contrat est modifié sans respecter ses contraintes, le système d'arbitrage le détecte et doit ensuite corriger l'erreur. Ainsi, pour intégrer une technologie multi-agents, nous avons besoin de transformer l'Organisation (l'instance de contrat) en son équivalent EJB afin de pouvoir faire des transactions sur son état. Les agents agissant dessus doivent alors également devenir des EJB.

En nous basant sur les propos de Brantschen et Haas [14] concernant l'utilisation des agents dans le monde J2EE et comme représenté sur le bas de la FIG. 10.7 pour les PeMAs, nous intégrons une plate-forme agents (ici SACI) au sein de la plate-forme J2EE et nous composons les *session beans* avec des agents. Nous donnons ainsi aux EJB la capacité de prendre des décisions, d'être organisés et d'agir sur des contrats structurés suivant un modèle $\mathcal{M}$OISE^{Inst}. Mais pour intégrer les agents, certaines de leurs capacités spécifiques doivent être ajoutées. Un composant "Glue" permet de faire le lien entre l'interface des agents et les EJB fournis par la plate-forme J2EE afin d'avoir les capacités des agents interfacées par les EJB. Ainsi, nous accumulons les avantages de J2EE et des SMA à savoir les transactions et l'Organisation.

La FIG. 10.7 représente l'architecture du système de management des contrats de notre application intégrant un SMA. Par rapport à la FIG. 10.6 les EJB représentant les PeMAs et les IeMAs exhibent en plus la propriété provenant de leur composante agent leur permettant de s'organiser. Ainsi les PeMAs ont d'une part une composante EJB accessible par la partie cliente de l'application et d'autre part une composante Agent leur permettant d'interpréter les informations d'organisation stockées dans la base de données et d'agir dessus afin de faire évoluer le contrat. Pour cela, les PeMAs doivent passer par les IeMAs jouant le rôle d'intermédiaire.

Les IeMAs sont construits sur la même base que les PeMA, à savoir une partie agent pour interpréter et agir sur les Organisations et une partie EJB pour mettre à jour les Organisations stockées dans la base de données et accessibles via des *Entity Beans*. Les PeMAs communiquent avec les "Org-Wrapper" fournis avec $\mathcal{S}$YNAI. Comme ils concernent exclusivement la plate-forme agent, ils n'ont pas d'équivalent EJB. Nous retrouvons sur le schéma parmi les *Entity Beans* ceux qui ont été ajoutés

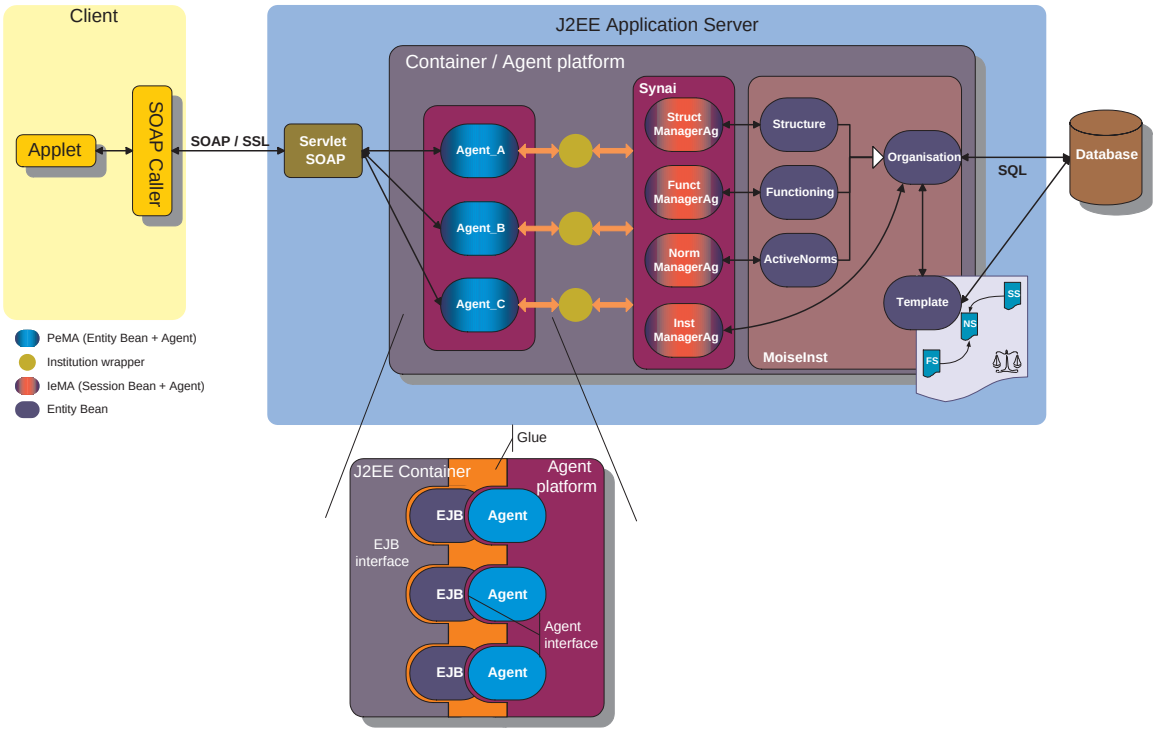


FIG. 10.7 – Architecture de EBSME intégrant les agents

pour représenter les contrats, à savoir pour commencer "Organisation" qui représente la partie organisationnelle d'un contrat et "Template" représentant la spécification de l'organisation et sur laquelle les IeMAs se basent pour contrôler la cohérence de l'Organisation. Nous avions vu que chaque IeMA contrôlait une instance de spécification en particulier. Il nous faut donc composer l'*Entity Bean* représentant une Organisation avec ses trois composantes pour un contrat. Chaque IeMA est donc un "Session Bean" accédant aux informations spécifiques de l'organisation via les "Entity Bean". L'EJB `TemplateController` non représenté sur la figure est ajouté pour gérer l'accès aux templates de contrats.

Il faut modifier les classes `Contract` (EJB de type entity) et `ContractController` (EJB de type session) afin de faire en sorte qu'en plus de l'identifiant, du statut, d'un document d'accord de collaboration, d'une devise, d'une description et des identifiants des participants, nous sauvegardions dans la base de données l'organisation normative représentant la dynamique d'exécution du contrat. Pour cela nous ajoutons dans la base de données un attribut *OrganisationId* dans la table "ContractBeanTable" ainsi qu'une table "OrganisationBeanTable" comme illustrée sur la FIG. 10.8 reprenant en partie le nouveau schéma de base de données d'EBSME. Dans la table "OrganisationBeanTable" nous avons entre autre une référence vers une table "TemplateBeanTable" que nous ajoutons également. En plus de ces deux nouvelles tables, nous ajoutons toutes celles nécessaires à la description des sous-structures composant une spécification d'organisation et une instance d'organisation que nous n'avons pas repris sur ce schéma pour en faciliter la lisibilité. A noter également qu'un attribut de réputation est ajouté aux informations du participant et sera mis à jour par l'utilisateur jouant le rôle de "DomArbitrator" durant l'exécution d'un contrat.

Grâce à cette architecture, il est possible d'avoir accès aux nouvelles formes de contrats électroniques basées sur le modèle $\mathcal{M}$OISE^{Inst} et de pouvoir agir dessus depuis l'interface cliente de l'application.

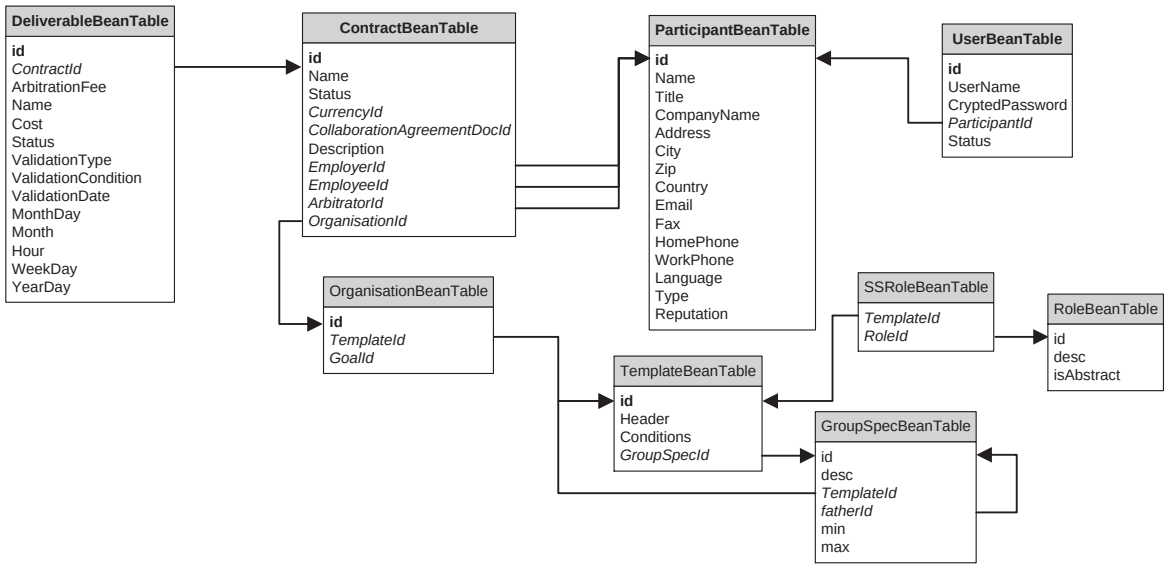


FIG. 10.8 – Schéma de la base de données de EBSME prenant en compte le modèle $\mathcal{M}$OISE^{Inst}

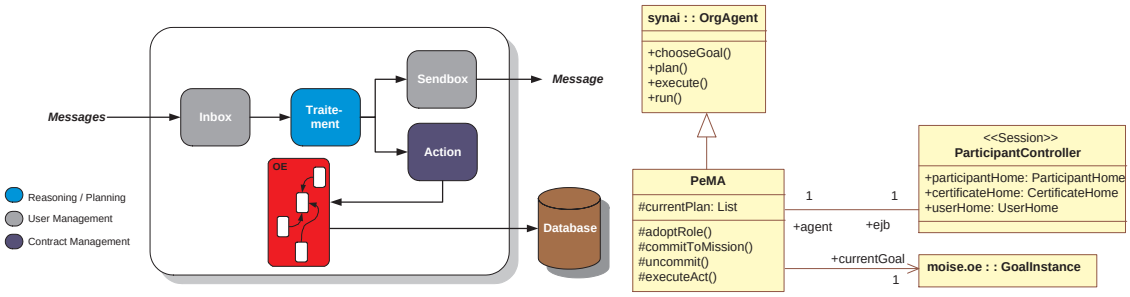
10.3.3 Implémentation

Nous venons de voir dans la section précédente la spécification de l'architecture J2EE globale de EBSME. Le cœur de l'application est d'une part l'enrichissement des contrats électroniques et d'autre part l'ajout de la composante agent aux *Session Bean* gérant l'accès aux informations de l'utilisateur. Nous allons faire en sorte que ces PeMAs puissent à la fois être organisés au sein d'un contrat et agir sur ces organisations. Les contrats sont représentés comme nous l'avons vu précédemment par des *Entity Bean*, toutes les informations seront donc stockées dans une base de données. Les PeMAs, contrôlés par les IeMAs vont agir sur les informations des contrats. Nous allons donc aborder l'architecture et l'implémentation de ces PeMAs avant d'aborder l'implémentation des contrats électroniques et des IeMAs.

Implémentation des assistants

Techniquement, les assistants des utilisateurs (les PeMA) sont composés d'une partie provenant de l'EJB de type Session et d'une partie provenant de l'agent. Si l'on considère que les PeMA sont composés des modules de gestion de l'utilisateur, de gestion des contrats et celui de raisonnement et de planning, nous pouvons répartir ces capacités comme suit : la partie de gestion de l'utilisateur se fera par la partie EJB, celle de raisonnement par la partie agent et celle de gestion des contrats, par l'action des agents sur l'organisation via les transactions EJB supervisées par les IeMA. Nous représentons ceci grâce à la FIG. 10.9(a). Il est évident que l'architecture d'un PeMA ressemble aux agents de $\mathcal{S}$YNAI. Elle pourrait se différencier par l'ajout d'un module plus spécifique de négociation de contrat ou de recherche de profil adéquate aux attentes de l'utilisateur, mais que nous ne prenons pas en compte ici.

Donc comme précisé précédemment l'assistant PeMA est constuit sur base d'un EJB et d'un agent. Le lien entre ces deux composantes est fait comme illustré sur la FIG. 10.9(b). Nous voyons que la classe `PeMA` hérite de la classe `OrgAgent` du package `synai`, lui apportant donc toutes les capacités d'un agent. Cette classe `PeMA` ajoute les fonctions et les attributs nécessaires pour l'évolution de l'agent au sein d'une organisation de type $\mathcal{M}$OISE^{Inst} au même titre que les Avatars dans le chapitre précédent. Comme les Avatars avec leur composante multimédia, les PeMA ont leur composante EJB leur permettant d'agir sur les informations des participants qu'ils assistent. De ce fait, la classe `PeMA` est associée au *Session Bean* "ParticipantController".



(a) Architecture d'un assistant (b) Diagramme de classe UML d'un assistant

FIG. 10.9 – Conception d'un assistant d'utilisateur PeMA

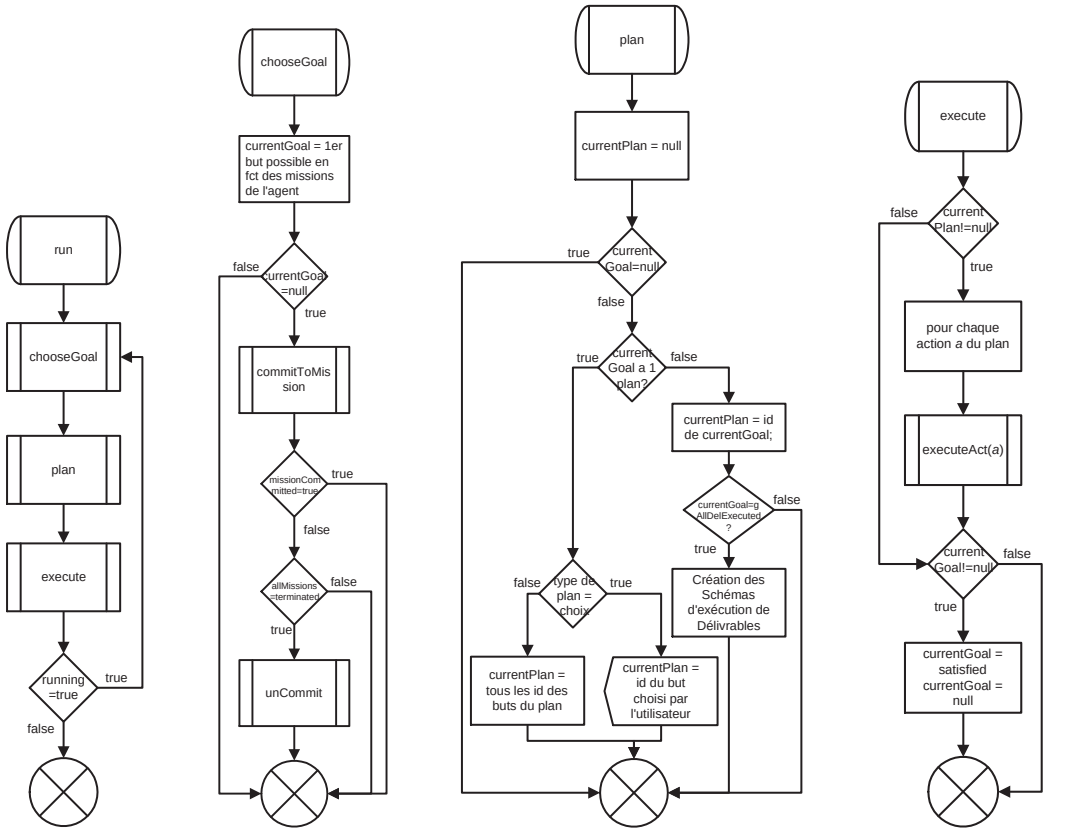
L'activation du cycle de vie du PeMA se fait via l'appel de la méthode *run()* que nous retrouvons sur la FIG. 10.10(a). Comme nous le voyons, cela se résume à faire appel itérativement aux trois fonctions *chooseGoal()*, *plan()* et *execute()* que nous allons maintenant décrire.

Méthode *chooseGoal()*
Le but courant est le premier but atteignable spécifié dans les missions sur lesquelles le PeMA est engagé. Si le but courant du PeMA n'a pas été défini, il tente de s'engager sur la première mission autorisée. S'il est effectivement engagé sur la mission alors il choisit de nouveau un but à satisfaire, si toutes les missions sont terminées il se désengage alors de ses missions. Nous illustrons cette méthode avec la FIG. 10.10(b).

Méthode *plan()*
Définir le plan d'un but courant signifie définir une liste d'identifiant représentant les sous-buts qu'il faut atteindre pour que le but courant soit lui aussi atteint. Les buts feuilles n'ont pas de plan et sont donc exécutés lors de l'appel de la méthode *executeAct()*. Mais si le but courant est un but racine qui n'a pas de plan, alors il ne pourra pas être exécuté. La méthode *plan()* spécifie que si le but courant existe et que le but courant n'a pas de plan, alors nous nous trouvons forcément dans le cas où le but courant est "gAllDelExecuted" ou "gRollback" et dans ce cas son identifiant est mis dans la définition du plan et si le but courant est "gAllDelExecuted" alors les schémas d'exécution de livrables sont créés. Si le but courant a un plan, alors soit son plan est défini avec les identifiants de ses sous-buts si le plan est de type séquence ou parallèle, soit son plan est défini avec un identifiant de but que l'utilisateur aura choisi si le plan est de type choix. Nous illustrons cette méthode grâce à la FIG. 10.10(c).

Méthode *execute()*
Si le plan courant n'est pas vide alors pour chaque but de l'ensemble des buts du plan courant, on appelle la méthode *executeAct()*, puis si le but courant existe alors il est spécifié comme satisfait et est mis à *null*. Nous illustrons cette méthode avec la FIG. 10.10(d).

Méthode *executeAct()*
Comme pour les Avatars de la section précédente et les superviseurs de $\mathcal{S}$YNAI, cette méthode fait le lien entre une identification d'action faisant partie d'un plan et l'action réelle que le PeMA doit mettre en place pour atteindre le but correspondant. Si un but du plan courant ne possède pas de sous-buts alors l'action correspondante est exécutée. Nous décrivons ci-dessous pour chaque but feuille de la FS d'un contrat l'action susceptible de le satisfaire.



(a) Exécution (b) Choix d'un but (c) Plan du but (d) Exécution du plan

FIG. 10.10 – Organigramme du cycle de vie des PeMA

Action : *relative au but "gAllDelExecuted"*	
Description	Fait en sorte d'atteindre le but "Tous les livrables sont exécutés, le contrat est terminé"
Arguments	
Pré-conditions	
Post-conditions	Demande de satisfaction du but "gAllDelExecuted"

Cette action permet de terminer un contrat. Si le but "gAllDelExecuted" est atteint alors le schéma d'exécution est terminé. L'action consiste à vérifier que tous les livrables composant un contrat ont été traités, c'est à dire dans l'ordre soumis, validés et payés. Les livrables doivent donc tous être dans l'état "Paid". Pour changer l'état d'un livrable, il suffit à l'assistant de le faire, après avoir soumis, validé ou payé le livrable. Ces actions sont reprises ci-dessous en une seule fois.

Action : *relative au but "gSubmitted"/"gValidated"/"gPaid"*	
Description	Fait en sorte d'atteindre le but "Le livrable est soumis"/"Le livrable est validé"/"Le livrable est payé"
Arguments	livrable
Pré-conditions	
Post-conditions	Envoi/validation/paiement du livrable Statut du livrable devient "submitted"/"gValidated"/"gPaid" Demande de satisfaction du but "gSubmitted"/"gValidated"/"gPaid"

Ces actions consistent donc à agir sur le livrable puis à en changer l'état afin d'atteindre les buts permettant de satisfaire le but "dDelTerm" mettant un terme au livrable. Il se peut que ces trois actions ne s'enchaînent pas. Il faut alors corriger et sanctionner. Nous basant sur les transactions offertes par les EJB, nous pouvons revenir dans l'état précédent de la base de données et annuler ainsi les mauvaises actions. L'action permettant cela est la suivante :

Action : *relative au but "gRollBack"*	
Description	Fait en sorte d'atteindre le but "La correction est appliquée"
Arguments	
Pré-conditions	
Post-conditions	La partie EJB du PeMA appelle la méthode *rollback* Demande de satisfaction du but "gRollBack"

Si le PeMA joue le rôle de "DomArbitrator", il doit pouvoir aussi atteindre des buts relatifs à l'arbitrage et notamment à la mise en œuvre de sanctions. Le but racine du schéma de sanction est composé de deux buts dont les actions doivent être exécutées en parallèle. La première consiste à baisser la réputation de l'utilisateur.

Action : *relative au but "gRepDecr"*	
Description	Fait en sorte d'atteindre le but "La réputation est baissée"
Arguments	PeMA fautif
Pré-conditions	
Post-conditions	La réputation du PeMA est baissée en fonction de l'importance de la norme violée Demande de satisfaction du but "gRepDecr"

Comme indiqué dans les post-conditions, la réputation sera baissée selon la norme violée. Ces informations sont négociées avant le début du contrat et complètent les informations concernant les tarifs, le montant des amendes etc. La demande de paiement d'une amende est une action permettant de satisfaire le deuxième but du schéma de sanction.

Action : *relative au but "gPayArb"*	
Description	Fait en sorte d'atteindre le but "La sanction financière est payée"
Arguments	PeMA fautif
Pré-conditions	
Post-conditions	Le PeMA fautif est puni par une pénalité financière Statut du livrable devient "Arbitrated" Demande de satisfaction du but "gPayArb"

Cette action fait en sorte que le PeMA fautif soit débité d'une pénalité financière dont le montant

aura été négocié avant le début de l'exécution du contrat.

Les changements de statut du livrable ainsi que la modification de la réputation d'un utilisateur ou l'annulation d'une transaction sont effectués par la partie EJB des PeMA. Les buts du schéma de sanction sont ici pris en compte par les PeMA car nous rappelons qu'un contrat représente un accord entre trois parties, l'employeur et l'employé et aussi l'arbitre en tant que troisième partie de confiance. Les IeMA détectent les violations du contrat tandis que le PeMA jouant le rôle de "DomArbitrator" est sollicité par une des deux autres parties en cas de litige sur un livrable. Si le type optionnel pouvait être intégré à la FS d'une spécification de contrat, un quatrième but serait inséré entre la soumission du livrable et sa validation. Si l'employeur n'est pas satisfait du livrable soumis par l'employé, alors il peut faire appel à l'arbitre afin que ce dernier tranche et donne raison à une des deux parties. Une re-négociation du contrat prendrait éventuellement effet, se traduisant par une ré-organisation au niveau de $\mathcal{M}$OISE^{Inst}. La négociation (et donc la re-négociation) ne faisant pas partie de la portée de nos recherches, de même que la ré-organisation, nous n'en dirons pas plus à propos de cette problématique. Voyons plutôt à présent l'implémentation des *templates* de contrat, des contrats électroniques en découlant, et de ceux qui les modifieront, les IeMAs.

Implémentation des contrats électroniques

Nous ajoutons aux informations du contrat d'origine une organisation de type $\mathcal{M}$OISE^{Inst} interprétable par les agents afin d'agir dessus pour les PeMA et de contrôler sa cohérence pour les IeMA. Au niveau des EJB, cela se traduit par l'ajout d'un attribut "organisationId" dans la classe `Contract` comme nous pouvons le constater sur la FIG. 10.11. Cet attribut fait référence à un nouvel EJB de type *Entity* nommé `Organisation` et représentant les informations contenues dans une instance d'organisation de type $\mathcal{M}$OISE^{Inst}. Par rapport à ce qui a été présenté dans le Chapitre 8 sur l'API de $\mathcal{M}$OISE^{Inst}, la structure des données n'est pas guidée pas les objets mais plutôt par leur stockage dans la base de données. De ce fait, nous ne pouvons pas reprendre toutes les classes de l'API $\mathcal{M}$OISE^{Inst} et les transformer en EJB de type *Entity* pour obtenir quelque chose de juste et de cohérent.

Ainsi sur la FIG. 10.11, l'EJB `Organisation` ne contient qu'un attribut liant l'instance à sa spécification, 'templateId', et une référence à l'identifiant d'une instance de but dont l'EJB n'est pas représenté ici pour faciliter la lisibilité. Nous trouvons également l'EJB `Template` représentant les informations contenues dans les *templates* de contrat. Cet EJB peut être vu comme l'équivalent composant de la classe OS de l'API $\mathcal{M}$OISE^{Inst}. La spécification des contrats fait référence par son attribut 'groupSpecId' à l'EJB représentant les spécifications de groupe. La suite des structures de données des *templates* de contrat et d'instances de contrat est passée sous silence ici. L'attribut :`header` définissant un contrat au niveau du modèle (cf. Section 10.2) est en fait un lien vers l'EJB `Contract`. Cet attribut est spécifique pour chaque instance de contrat et se trouve donc dans l'EJB `Organisation` et non dans `Template`.

De la même manière que nous avons l'EJB `ContractController` contrôlant entre autre l'accès aux informations de la table "Contract" via l'EJB `Contract`, il nous faut également des EJB capables de contrôler l'accès aux données représentant les organisations normatives des contrats. Nous créons donc des IeMA, EJB de type *Session*, dont la tâche est de contrôler l'accès aux données de l'organisation normative. Nous avons vu jusqu'à maintenant, du point de vue agent, que l'action sur l'organisation normative était contrôlée par les agents de supervision de $\mathcal{S}$YNAI. Nous lions donc aux IeMA une composante agent afin d'obtenir comme pour les PeMA des EJB autonomes basés sur les agents. Du point de vue spécification, sur le diagramme de classe complet des EJB, nous aurions l'EJB IeMA lié à tous les *Entity Bean* représentant l'organisation et sa spécification. Dans la pratique et lors de l'instanciation, nous aurions autant d'IeMA différents que nous avons de Superviseurs différents dans la couche $\mathcal{S}$YNAI.

Maintenant que nous avons étudié l'implémentation de toutes les composantes 'agentifiant' EBSME, nous allons voir dans la section suivante comment elles sont utilisées et exécutées. Nous y détaillons

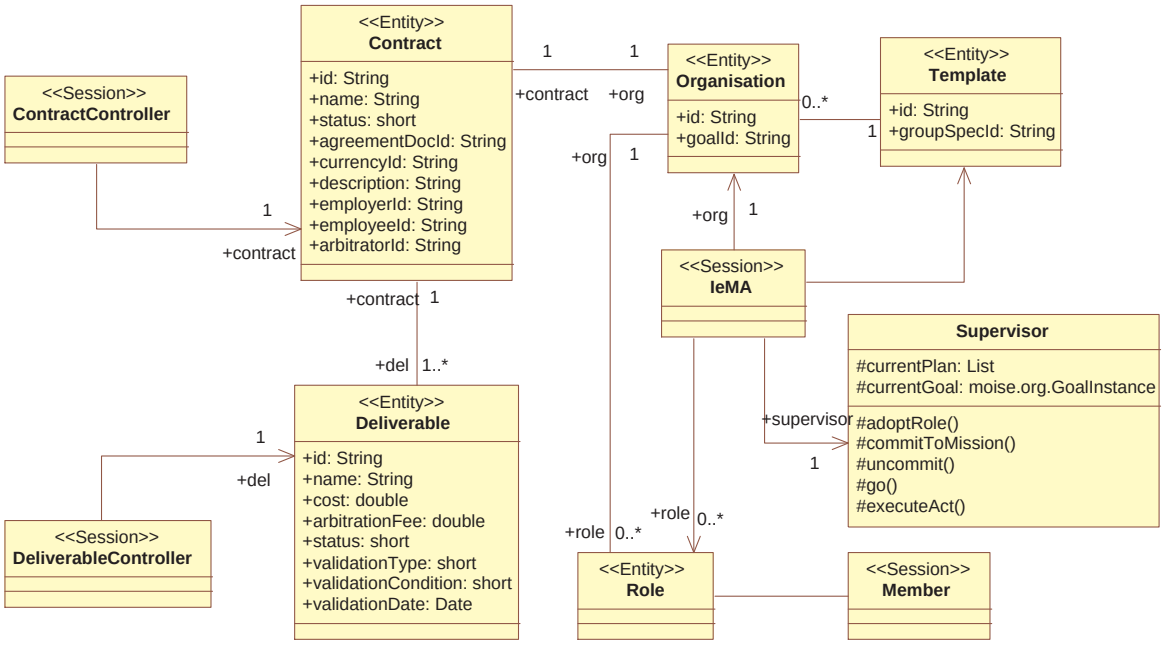


FIG. 10.11 – Diagramme de classe d'un contrat et des IeMA concernés

un scénario d'enregistrement d'un utilisateur, de création et d'exécution d'un contrat ce qui nous permet de voir à quel moment les PeMA et les IeMA d'un contrat sont créés. Nous verrons aussi lors de l'exécution d'un contrat comment chaque assistant va interagir avec les autres pour atteindre son objectif, celui de faire en sorte qu'un contrat se déroule sans accroc.

10.4 Expérimentation et validation

Nous voulons expérimenter et valider l'utilisation de notre modèle d'Institution Électronique dans le contexte d'une application de commerce électronique. Pour ceci, nous allons exécuter dans ce qui suit un scénario permettant à un utilisateur de s'enregistrer, de créer un contrat et de l'exécuter. Nous étudierons alors les résultats que provoque un fonctionnement normal des agents, c'est à dire lorsqu'ils respectent leurs contrats, et un comportement non-respectueux de la spécification du contrat. En analysant les comportements des agents du domaine (les PeMA) et les agents de SYNAI (les IeMA), nous pouvons tirer des conclusions sur la validité de notre modèle.

10.4.1 Exécution du scénario

Dans un premier temps nous décrivons la façon dont l'utilisateur va s'enregistrer, va créer et signer un contrat puis l'exécuter au même titre que les autres parties contractantes.

Enregistrement d'un utilisateur

Pour un utilisateur voulant fonctionner sur la plate-forme, il faut d'abord qu'il s'enregistre. Il commence donc par entrer un *login* et un *password*. Un EJB de type *Entity Bean* "Participant" reprenant les informations du participant, un EJB de type *Session Bean* "ParticipantController" qui via l'EJB "Participant" va mettre à jour les informations et un agent PeMA lié au "ParticipantController" seront alors créés. L'activation de l'EJB déclenchera le fonctionnement de l'agent. Il va commencer par rejoindre la société SACI utilisée par le système pour faire évoluer tous les agents des différents

utilisateurs. Ensuite, il va exécuter son cycle de vie.

Ceci étant fait, l'utilisateur va voir s'afficher l'interface de gauche de la Fig. 10.12. Le fait d'envoyer sur le serveur son certificat lui permettra de signer les documents de façon sûre. Une étude sur la possibilité d'utiliser un gestionnaire de clef (une *Public Key Infrastructure*) a été faite au CRP Henri Tudor [81]. L'utilisateur va pouvoir également enregistrer toutes les informations le concernant via le bouton "User Profile Registration" et à travers les interfaces de droite de la Fig. 10.12. Les trois grands thèmes des informations concernant l'utilisateur sont en jaune sur l'interface et correspondent aux EJB "`Participant`", "JobProfile" et "TranslatorSkills". Les mises à jour seront contrôlées par les EJB "JobProfileController" pour les informations sur les disponibilités et le titre de l'utilisateur ainsi que sur ses compétences en matière de traduction et par l'EJB "ParticipantController" pour les informations personnelles relatives à l'utilisateur. C'est lorsque cet EJB "ParticipantController" avec état va être créé qu'un agent va également être créé et lui être associé afin de fournir à l'utilisateur son PeMA. L'agent va commencer son cycle de vie mais ne trouvera pour le moment aucun but à atteindre. Il faut qu'il attende que l'utilisateur exécute un contrat.

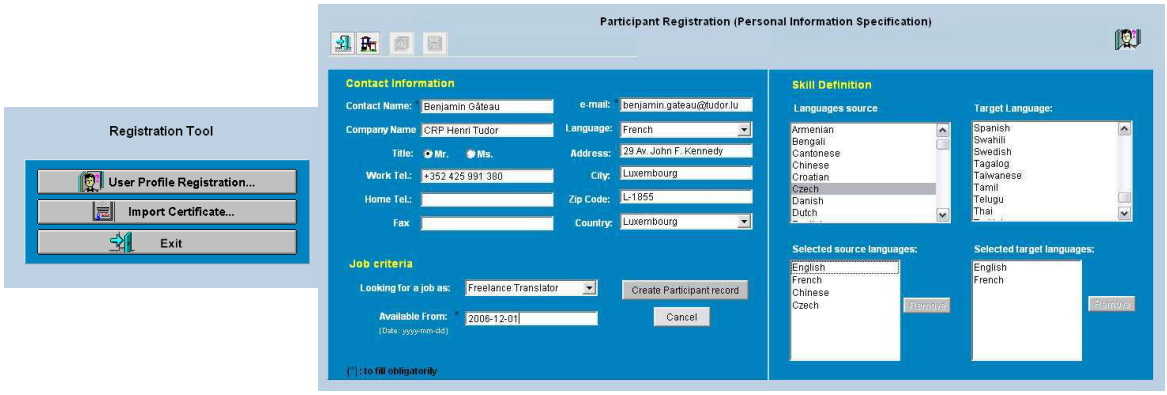


Fig. 10.12 – Interfaces pour l'enregistrement d'un nouvel utilisateur

Création d'un contrat

Une fois le nouvel utilisateur enregistré et connecté, le menu de la Fig. 10.13 va lui permettre d'agir au sein de la plate-forme. Il peut bien entendu modifier les informations le concernant, ses critères de recherche d'emploi ou bien encore ses compétences (1). Le bouton "View/Search Jobs" lui permet grâce à l'interface de droite de la Fig. 10.13 de trouver les autres utilisateurs correspondant à ce qu'il cherche ou à ce qu'il offre. Une fois un partenaire potentiel repéré, il peut le contacter en dehors de la plate-forme par e-mail ou par téléphone pour commencer une négociation afin de déboucher sur un éventuel accord. Les quatre boutons (2) du menu permettent de gérer tout ce qui est relatif aux contrats, à savoir leur création, leur modification, leur signature et leur exécution. L'exécution ne sera pas la même pour chaque contrat pour un utilisateur en fonction de son rôle au sein de l'accord. Le dernier bouton (3) permet de visualiser et de résoudre les conflits des contrats sur lesquels l'utilisateur joue le rôle d'arbitre. Voyons pour commencer la création d'un contrat.

Nous considérons donc que l'utilisateur a trouvé un partenaire disponible pour effectuer une traduction ainsi qu' un autre participant pour jouer le rôle de tiers de confiance. Il va donc créer un contrat. L'interface de la Fig. 10.14 illustre la façon dont le contrat se crée. La zone (1) permet de décider quelle devise est utilisée pour les paiements au sein du contrat, d'afficher l'identifiant du contrat et de sélectionner un document qui sera transféré sur le serveur en tant que contrat papier réel. Ces informations sont stockées dans la base de données au travers l'EJB "Contract" et contrôlées par l'EJB "ContractController". L'EJB "Contract" permet aussi de stocker les informations sur les participants aux contrats choisis dans la zone (3) par l'utilisateur. Bien entendu, l'utilisateur créant le

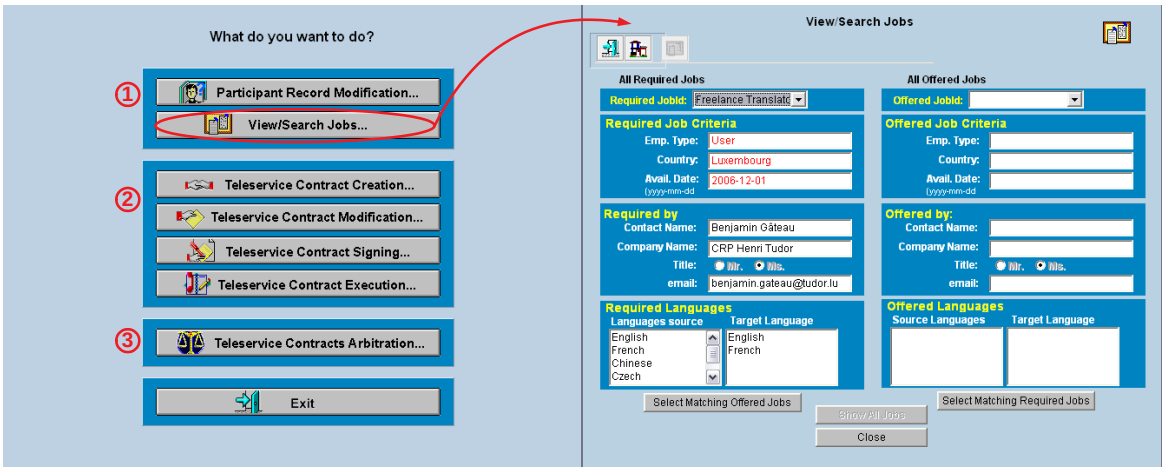


FIG. 10.13 – Menu principal d'un utilisateur et interface de recherche de job

contrat doit se sélectionner en tant qu'employeur. Il sélectionne ensuite l'employé et l'arbitre. Comme montré sur le diagramme d'instance de la FIG. 10.14, l'EJB "Contract" est lié à l'EJB "Deliverable" (2) et ses informations sont affichées dans la zone (2) de l'interface. L'utilisateur ajoute des livrables via l'interface de droite de la FIG. 10.14 en choisissant le nom du livrable, son coût, le coût d'un arbitrage, la méthode de validation (par livrable ou par contrat) et la date de rendu du livrable. Pour ce qui est de la description des tâches à effectuer pour obtenir le livrable, tout ceci doit être expliqué dans le document de collaboration abordé ci-dessus. Dans la zone (2), le statut du livrable peut être NEW_DELIVERABLE, PROVIDER_CONFIRMED, CUSTOMER_VALIDATED, CUSTOMER_REJECTED, ARBITRATION_REQUESTED, ARB_VALIDATED_DLVRBL ou ARB_REJECTED_DLVRBL.

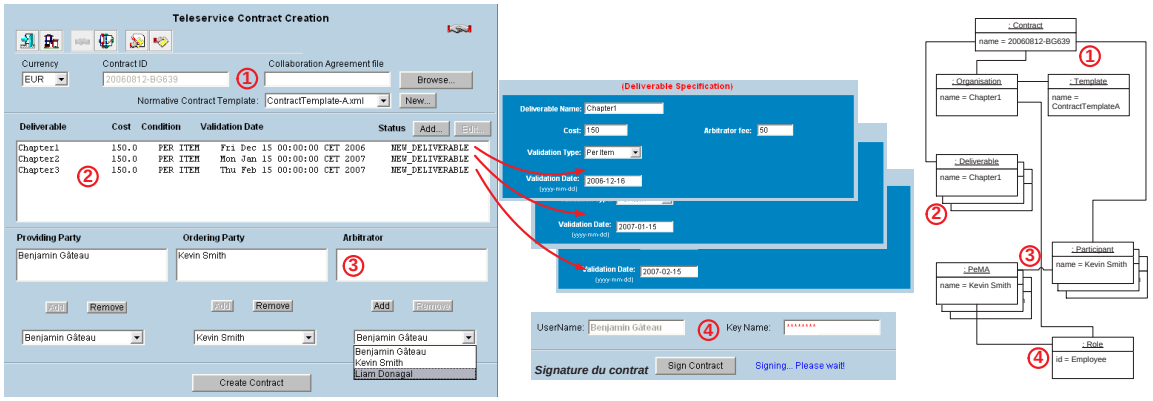


FIG. 10.14 – Interfaces de création d'un contrat

Nous n'avons pas abordé dans la zone (1) la liste permettant de choisir une spécification d'organisation normative en tant que *template* de contrat. L'interface permet de choisir un *template* existant ou d'en créer un autre. La création d'une nouvelle spécification se fait en chargeant un nouveau fichier xml sur le serveur, qui sera *parser* en EJB "Template" et stocké dans la base. Lorsque l'utilisateur validera le formulaire pour créer le contrat, une organisation $\mathcal{M}$OISE^{Inst} sera instanciée : un EJB "Organisation" sera créé ainsi que tous les autres EJB relatifs tels que les groupes, les buts, les normes et le schéma d'exécution de contrat. Les agents pourront alors adopter les rôles qui leur sont attribués dans le contrat. L'EJB "Organisation" référencera l'EJB représentant le *template* et il sera quant à

lui référencé par l'EJB "Contract". La mise à jour du participant via son "ParticipantController" va provoquer un appel de méthode de la partie agent du PeMA qui va faire la demande auprès de la couche SYNAI (créée avec le contrat) d'adopter un rôle au sein de cette nouvelle organisation. Une fois le rôle adopté, il peut choisir un nouveau but et commencer à s'exécuter.

Avant que le contrat puisse effectivement être exécuté, il faut que chaque partie signe le contrat. Du point de vue sécurité, la signature se fait en renseignant la clef privée du certificat que l'utilisateur aura envoyée lors de son enregistrement. En plus d'authentifier l'utilisateur, l'action de signer un contrat va déclencher une demande d'adoption du rôle défini dans le *template* par le PeMA. Si la requête est acceptée par les IeMA, alors un EJB "Role" va être créé afin de représenter la spécification de rôle que jouera le participant sur ce contrat. Nous retrouvons ces EJB sur le diagramme d'instance à droite de la FIG. 10.14. Puisque le PeMA joue un rôle, d'après son cycle de vie, il peut maintenant s'engager sur la mission du schéma d'exécution de contrat qui a été créé en même temps que le contrat. Lorsqu'il va définir le plan pour atteindre ce but, il va créer tous les schémas d'exécution de livrable (comme les agents sont trois à vouloir atteindre ce but, une vérification est faite pour ne créer qu'une seule fois les schémas d'exécution de livrable). Ici, trois différents schémas seront créés. Maintenant que le contrat est créé et signé, et que tous les éléments organisationnels sont en place, l'exécution peut avoir lieu.

Exécution normale d'un livrable

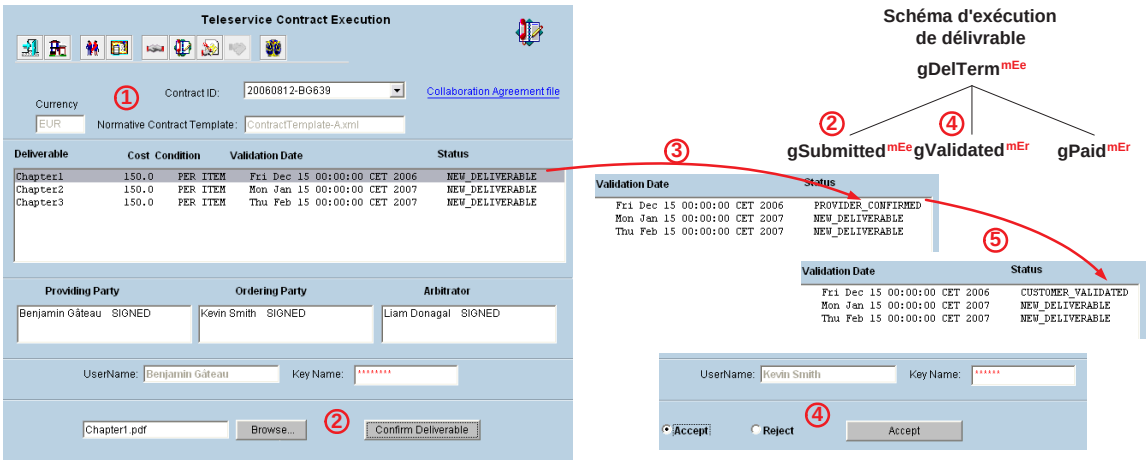


FIG. 10.15 – Interface pour l'exécution d'un contrat par l'employé

Du point de vue agent, l'exécution d'un livrable se fait en atteignant successivement trois buts. Premièrement, l'employé doit soumettre le document du livrable, puis l'employeur doit valider le document et enfin l'employeur doit payer l'employé. Nous retrouvons sur l'interface de la FIG. 10.15 les outils pour atteindre ces buts. La zone (1) donne quelques informations sur le contrat que l'on veut exécuter. Une liste permet de choisir le contrat concerné parmi tous les contrats sur lesquels l'utilisateur est impliqué. La devise utilisée ainsi que le *template* sont mentionnés et un lien permet de télécharger le document papier du contrat. En sélectionnant un livrable, l'utilisateur qui joue le rôle d'employé peut choisir un document à envoyer et confirmer qu'il a exécuté le livrable dans les temps. Pour cela il doit également renseigner sa clef privée afin de signer le document. Cette étape (2) correspond au fait d'atteindre le but "gSubmitted" du schéma représenté sur la partie droite de la FIG. 10.15. En effet, lorsque l'utilisateur va cliquer sur le bouton, la partie EJB de l'assistant va indiquer à l'agent que le but est atteint après avoir exécuté l'action correspondante (cf. Section 10.3.3). Celui-ci va donc faire une requête de satisfaction de but à l'IeMA adéquat, lequel une fois les vérifications faites va mettre à jour l'organisation et va permettre au "ParticipantController"

(la partie EJB du PeMA) de mettre à jour les informations relatives au livrable via le "Deliverable-Controller". L'assistant de l'employé n'a à ce moment là plus d'autres buts à satisfaire. Le statut du livrable est mis à jour comme montré à l'étape (3) de la FIG. 10.15.

L'employeur ne pourra effectuer, via la même interface, une action que si le livrable qu'il sélectionne aura été confirmé par l'employé. Après avoir bien entendu pris en considération le document soumis par l'employé, l'employeur va décider s'il accepte ou rejette le livrable. Dans ce cas-ci et comme illustré sur la figure à l'étape (4), l'employeur accepte le livrable et certifie sa décision en fournissant sa clef privée. Ceci aura pour conséquence de faire en sorte que le PeMA atteigne le but "gValidated" du schéma d'exécution du livrable. De la même façon que pour le PeMA de l'employé, l'assistant va faire une requête de satisfaction de but pour ensuite pouvoir mettre à jour l'organisation du contrat et faire en sorte que les autres *Session Bean* mettent à leur tour à jour le livrable. Cette dernière mise à jour s'illustre par le changement de statut visible à l'étape (5). Enfin pour parfaire l'exécution du livrable et pour atteindre le but "gDelTerm", il faut que l'employeur paie son employé. Une fois que sa validation est faite, un bouton permet d'afficher une fenêtre de paiement. L'employé n'a plus qu'à choisir le moyen de paiement, renseigner ses numéros de compte ou de carte si cela n'a pas été fait auparavant et il procède au paiement proprement dit. Cette partie de l'application n'est pas implémentée. Une fois la confirmation du paiement reçue, le but "gDelPaid" est atteint, du moins l'assistant fait une requête de satisfaction de but et ainsi de suite.

Nous venons de voir un scénario où tout se passait correctement et où aucune contrainte n'était enfreinte. L'Organisation est restée cohérente et les termes du contrat ne sont pas violés. De plus, aucun conflit n'est apparu entre les parties contractantes. Voyons maintenant un cas où cela arriverait.

Exécution anormale d'un livrable

Nous considérons comme anormale l'exécution d'un livrable lorsqu'il y a un conflit entre l'employeur et l'employé sur la validité du document envoyé et que l'intervention de l'arbitre est nécessaire ou lorsqu'une contrainte de l'organisation (structurelle, fonctionnelle ou normative) est violée par l'une des parties contractuelles (via son PeMA), qu'elle est détectée par les IeMA et que ceux-ci soient obligés de mettre en place un processus de correction et de sanction.

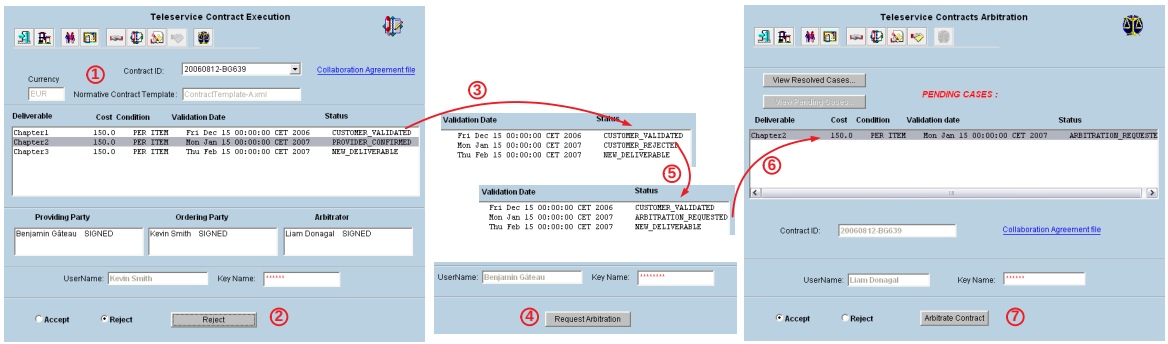
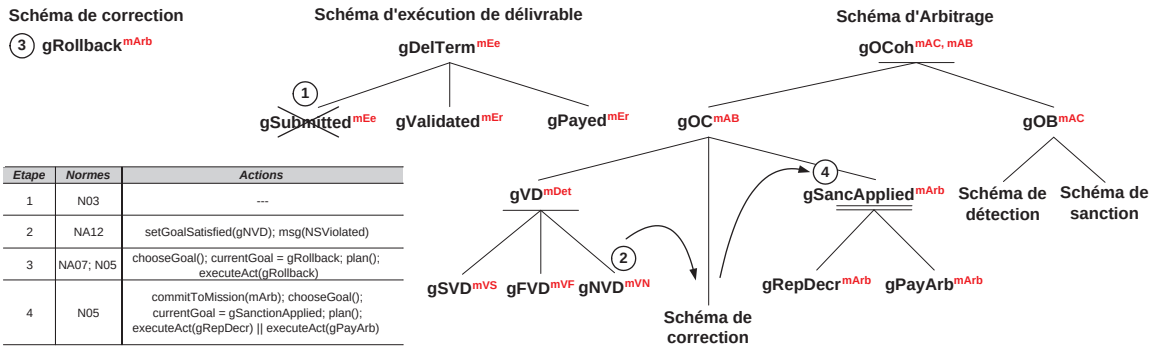


FIG. 10.16 – Interface pour l'exécution d'un contrat par l'employeur

Sur les interfaces de la FIG. 10.16 nous retrouvons notre contrat et ses participants. L'employé a soumis son document et a donc atteint le but "gSubmitted". L'employeur en phase (2) rejette le document et l'employé peut donc voir le statut du livrable en (3) comme étant CUSTOMER_REJECTED. La possibilité est donnée à l'employé de contester cette décision en faisant appel à l'arbitre du contrat. Ainsi en sélectionnant le livrable en question, il peut en phase (4) demander un arbitrage. En phase (5), l'état du livrable passe donc dans l'état ARBITRATION_REQUESTED.

En choisissant le menu “Teleservice Contracts Arbitration” de l’interface de la FIG. 10.13, le participant jouant le rôle de l’arbitre a un aperçu des livrables qui posent problème et appartenant aux contrats dont il est l’arbitre. Sur l’interface de droite de la FIG. 10.16, nous constatons que l’arbitre n’a qu’un seul cas à résoudre. Pour cela, il peut voir à quel contrat le livrable appartient et également télécharger le document d’accord de collaboration. Le document du livrable lui aura été transmis par e-mail et sera téléchargeable via une fenêtre d’accueil accessible par tous les participants dès leur connexion au système. En effet, les assistants personnels se chargent de prévenir, par mail et via un résumé des tâches à effectuer lors de la soumission d’un document, de la validation ou non de ce dernier, de la demande d’arbitrage, etc. La plate-forme EBSME nous sert dans nos travaux de recherche à valider notre modèle d’Institution Électronique. Nous sommes conscients que la solution en terme de commerce électronique n’est pas la solution idéale. L’utilisation d’un workflow serait notamment plus adapté pour le suivi des documents.

Quoi qu’il en soit, après avoir sélectionné un livrable, l’arbitre peut rendre son verdict et accepter ou non à son tour le document. Le livrable apparaîtra ensuite via l’interface d’exécution des contrats de l’employé et de l’employeur comme ayant le statut ARB_VALIDATED_DLVRBL ou ARB_REJECTED_DLVRBL. Si l’arbitre valide le livrable, l’employeur doit se soumettre à cette décision et atteint de ce fait le but “gValidated”. Il doit donc ensuite payer l’employé ainsi que les frais d’arbitrage concernant ce livrable. Par contre, si l’arbitre décide de rejeter le livrable, une phase de re-négociation et de modification du contrat en cours intervient ici. Cela peut aller de la prolongation de la date de rendu avec une réduction du coût jusqu’à l’annulation du livrable voire même du contrat. Cette problématique de ré-organisation automatique ne fait pas partie du domaine de nos travaux. Les frais d’arbitrage sont à la charge de l’employé. En fonction de la décision prise par le participant jouant le rôle de l’arbitre, la partie EJB de son assistant va automatiquement réduire la réputation du participant qui était en tort.



Etape	Normes	Actions
1	N03	---
2	NA12	setGoalSatisfied(gNVD); msg(NSViolated)
3	NA07; N05	chooseGoal(); currentGoal = gRollback; plan(); executeAct(gRollback)
4	N05	commitToMission(mArb); chooseGoal(); currentGoal = gSanctionApplied; plan(); executeAct(gRepDecr) || executeAct(gPayArb)

FIG. 10.17 – Exécution du schéma de supervision des IeMAs

L’exemple que nous venons de voir illustre un arbitrage fait par le participant et son EJB. Imaginons maintenant que pour le dernier livrable, l’employé n’ait toujours rien soumis le 15 février 2007. Or, la norme N03 stipule que l’employé doit soumettre un document avant la date limite de fin de livrable, c’est à dire le 15 février 2007 à minuit. L’IeMA responsable du management des normes est le *NornManagerAg* dont l’algorithme d’exécution des tâches est représenté sur la FIG. 8.23. Le but à atteindre pour ce Superviseur est “gNVD” (violation normative détectée). Tant qu’il ne reçoit pas de message de vérification de respect de norme active, il vérifie les contraintes temporelles, de validité et d’activation de toutes les normes actives. Or dans le cas présent, nous sommes le 16 février 2007 et le but “gSubmitted” appartenant à la mission “mEe” n’est pas satisfait et la mission non atteinte (phase (1) de l’exécution du schéma de supervision de la FIG. 10.17) et par conséquent la norme N03 est violée. Ainsi, le “NormManager” atteint son but “gNVD” et prévient “InstManager” de cela (2). Pour atteindre le but “gOC” (organisation corrigée), le schéma de correction doit être exécuté. Ce

schéma est composé d'une seule mission dont le but doit être satisfait par le rôle "DomArbitrator". Conformément à la norme N05, cet agent s'engage sur la mission "mArb" et tente de satisfaire le but. Comme détaillé dans la Section 10.3.3, pour atteindre le but "mArb", il faut entre autre effectuer un *Rollback* sur la base de données via l'EJB "ParticipantController". Enfin et toujours pour atteindre le but "gOC", le but "gSancApplied" doit être satisfait ce qui sera vrai quand le rôle "DomArbitrator" satisfera les deux buts de sanction servant à réduire la réputation du fautif d'une part et de lui faire payer une amende d'autre part. Alors les buts "gOC" et "gOCoh" sont satisfaits, le "NormManager" a accompli sa mission et un nouveau schéma d'arbitrage est créé contenant de nouvelles missions sur lesquelles s'engagent les IeMAs.

L'arbitrage via les agents est différent de l'arbitrage par l'utilisateur via son EJB même si les deux sont étroitement liés. Nous venons de voir différents scénari d'exécution de livrables mettant en lumière ces différents arbitrages. L'arbitrage par les EJB a été guidé par la FS de l'Organisation tandis que l'arbitrage des agents a été possible par la NS de l'Organisation et les IeMA guidés par les normes les contraignant. Nous allons maintenant en tirer nos conclusions.

10.4.2 Validation

Le méta-modèle organisationnel $\mathcal{M}\text{OISE}^{Inst}$ est basé sur quatre spécifications permettant de définir la structure d'une organisation, les fonctionnalités de cette organisation, les contextes dans lesquels l'organisation peut se trouver et enfin un ensemble de règles se basant sur les trois autres spécifications. Ce découpage en spécifications apporte plus de flexibilité. En effet, dans l'exemple d'application que nous venons de voir, la modélisation que nous faisons d'un *template* de contrat n'est basée que sur trois spécifications. Une spécification d'organisation est ensuite instanciée en organisation. Dans cet exemple, une spécification représente un *template* de contrat et son instance un contrat réel et actif. Ces organisations ont été liées aux contrats existants afin d'enrichir leur définition et permettre à des entités autonomes assistant les utilisateurs non seulement d'automatiser certaines actions mais aussi d'agir sous le commandement des participants ainsi que sous le contrôle d'un ensemble de superviseurs de contrats.

Une personne enregistrée dans le système peut évidemment passer plusieurs contrats et avoir donc plusieurs contrats à exécuter en même temps. La négociation des délais doit se faire pour le moment par l'être humain en fonction des contrats qu'il a déjà. L'assistant de l'utilisateur peut jouer des rôles différents dans plusieurs contrats. Nous avons donc affaire ici à un agent agissant sur plusieurs organisations à la fois. Nous représentons sur la Fig. 10.18 l'exemple d'un utilisateur ayant son PeMA impliqué sur deux contrats différents mais issus du même *template* de contrat. Il est à noter ici l'importance d'une bonne modélisation des organisations. En effet, un agent pourrait se retrouver dans une situation de conflit de normes. Cependant l'absence de contexte et d'interdiction facilite grandement l'empêchement de se trouver face à une interdiction et une obligation sur la même mission. La structure simple exclut également le fait d'avoir des normes différentes selon les rôles joués. Nous pourrions toutefois imaginer avoir en plus des institutions gérant des contrats une institution globale gérant la plate-forme elle-même. De ce fait, en plus de jouer un rôle ou non au sein d'un contrat, les assistants joueraient obligatoirement le seul rôle possible de l'organisation d'EBSME, et seraient contraints sur les actions que les agents pourraient avoir à faire en dehors d'un contrat comme par exemple chercher des partenaires potentiels et négocier pour aboutir à la création de nouveaux contrats (donc de nouvelles organisations dans lesquelles agir). L'intégration d'une spécification d'interaction serait alors nécessaire pour la négociation.

Dans le cadre des organisations représentant les contrats, une spécification des interactions n'est pas nécessaire étant donné le faible nombre d'échanges entre les agents eux-mêmes au sein du contrat. Les échanges se limitent à l'envoi des documents accompagnant les livrables. Un système de *workflow* est de ce fait plus utile ici et fait partie de la gestion des contrats via les composantes EJB des assistants des utilisateurs.

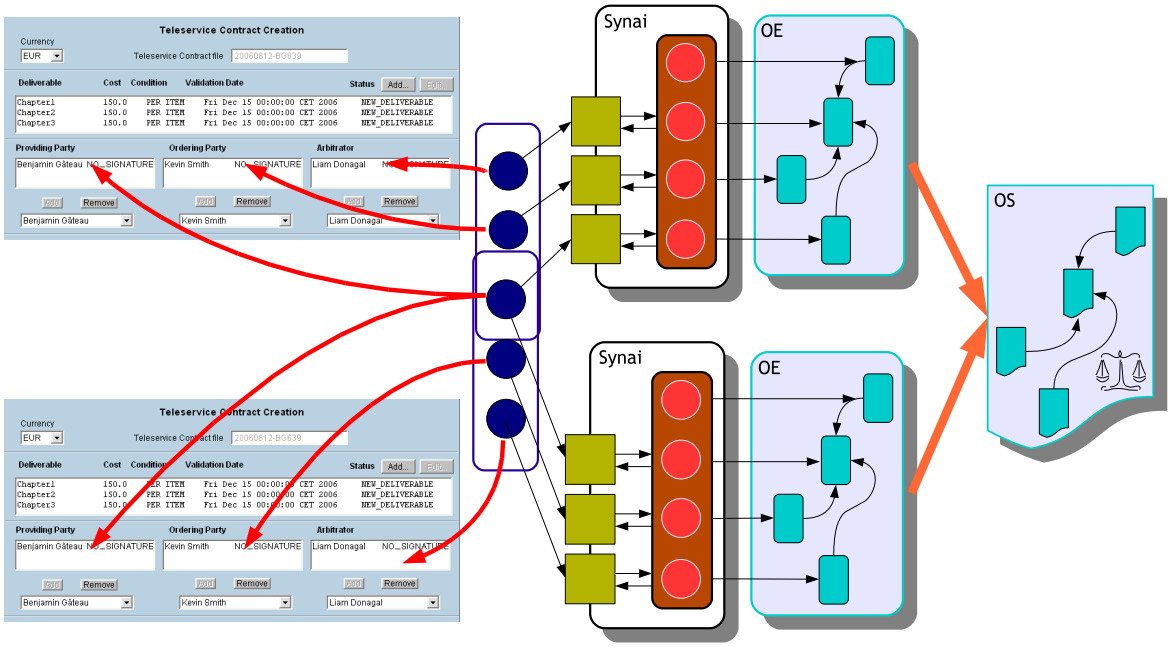


FIG. 10.18 – Exemple de “multi-institutions”

La modélisation complète d'un contrat à l'aide d'une organisation multi-agents est compliquée l'objectif est de définir le processus métier. De nombreux facteurs interviennent comme le type de services ou de produits sur lesquels porte le contrat, les montants, les processus d'obtention de chaque livrable, les conditions de validation de ces mêmes livrables. La solution proposée ici permet de spécifier à l'aide d'un modèle organisationnel les rôles des participants sur le contrat, le cycle de vie d'un livrable (sans entrer dans le détail de son élaboration et de sa validation), les sanctions appliquées en cas de non respect d'une partie ou de la totalité du contrat et enfin les règles de fonctionnement pouvant être vues comme des clauses. La représentation existante d'un contrat électronique (spécifiant notamment le nombre de livrables ainsi que leur date limite de rendu) est maintenue quoique liée à notre organisation. Ainsi, son exécution peut être supervisée par les agents de $\mathcal{S}$YNAI fournis avec le modèle. Dans notre cas, l'implémentation des agents a été quelque peu modifiée pour intégrer la plate-forme J2EE (de même que pour l'API du méta-modèle $\mathcal{M}$OISE^{Inst}), mais les mécanismes de détection et de décision restent les mêmes. Il est donc possible avec notre système d'arbitrage défini de manière générale de contrôler l'action d'agents sur un contrat électronique.

Ces agents de $\mathcal{S}$YNAI, contrairement aux agents assistant les utilisateurs sont entièrement autonomes et permettent ainsi un arbitrage automatisé sans intervention humaine. Si un agent détecte une erreur, l'erreur est corrigée et le fautif sanctionné. Un rapport d'erreurs peut ensuite être généré et un administrateur peut intervenir dans le cas d'erreurs impliquant l'annulation de contrat par exemple. Les protocoles d'actions concernant en particulier la législation des contrats ne sont pas pris en compte par les agents de supervision. Ils se contentent de superviser en interprétant les différentes spécifications de contraintes concernant le contrat y relatif. Si nous imaginons nous trouver dans le cas d'un contrat re-négocié en cours d'exécution avec une modification des délais par exemple, le “NormManager” s'adaptera et prendra en compte de la même manière cette norme modifiée.

Un point sur lequel les Systèmes Multi-Agents sont souvent attaqués est leur limite au niveau de la sécurité. En effet les agents offrant une certaine mobilité doivent utiliser des techniques pour garantir une exécution sûre de leur code sur le système ou mettre en place des barrières empêchant le serveur d'accéder à leur contenu. Cependant dans notre cas, les agents sont amenés à vivre sur le serveur en tant que composant des PeMAs et de leur architecture ; de plus leurs capacités d'actions

sont fournies. Ce sont donc les communications qui doivent utiliser des canaux sûrs. Dans une future version de la plate-forme utilisant les capacités de négociation des agents, des mécanismes de confiance entre agents devront être intégrés. L'ajout que nous avons fait d'une première version d'une valeur de réputation pour chaque participant est une première étape. Pour ce qui est de la sécurité initiale promue par EBSME, on est en droit de se demander ce qu'il advient de la sécurité assurée par le *framework* J2EE, des connexions SSL et de l'utilisation du protocole SOAP. Au premier abord, nous serions tentés de répondre que les agents agissant via leurs EJB associés au sein des PeMAs, le niveau de sécurité est inchangé. De plus, nous pensons que la sécurité des agents en général peut être obtenue à l'aide de solutions traditionnelles et fournie par un EJB et sa plate-forme J2EE dans notre cas. Nous ne l'avons cependant pas encore prouvé et les tests sont encore à effectuer.

10.5 Synthèse

Dans ce chapitre, nous avons utilisé $\mathcal{M}$OISE^{Inst} pour modéliser des contrats électroniques. Ces contrats sont pris en compte par des assistants basés sur des agents afin de gérer et aider à l'exécution des contrats passés par l'utilisateur. Un assistant peut donc être impliqué sur plusieurs contrats et donc devoir respecter des contraintes provenant d'institutions différentes. Le danger est donc plus grand de violer une norme de contrat. $\mathcal{S}$YNAI a été intégré à la plate-forme de commerce électronique afin de détecter, corriger et punir les violations de contrat.

Les API de MA$\mathcal{B}_{\text{ELI}}$ ont dû être adaptées afin de pouvoir être intégrées au sein de la plate-forme J2EE existante. Chaque classe du modèle $\mathcal{M}$OISE^{Inst} est transformée en *Entity Bean* lui donnant la possibilité d'être stockée dans la base de données. Cela a deux avantages, la possibilité de garder dans une mémoire physique les informations concernant les organisations relatives aux contrats et la possibilité de retourner à un état antérieur de l'organisation en cas de violation d'une norme.

Nous venons donc d'illustrer le fait de pouvoir contraindre et superviser un agent à l'aide de plusieurs institutions. L'application dans le domaine du commerce électronique montre que MA$\mathcal{B}_{\text{ELI}}$ n'est pas liée ou restreinte à un domaine en particulier et que les agents de $\mathcal{S}$YNAI supervisent de la même manière. La spécification très simple avec $\mathcal{M}$OISE^{Inst} est faite pour être instanciée par un nombre limité d'agents (3 par contrat). Nous avons un contexte de fonctionnement des agents agissant par petit groupe au sein d'un ensemble de plusieurs institutions. Après avoir validé MA$\mathcal{B}_{\text{ELI}}$ dans deux domaines d'application différents nous pouvons tirer les conclusions de cette partie expérimentale.

Quatrième partie

Conclusion & Perspectives

Chapitre 11

Conclusion

Dans ce chapitre, nous exposons les conclusions de cette thèse. Nous commençons par rappeler la problématique mise en avant tout au long de ce manuscrit, puis nous faisons la synthèse de nos contributions. Enfin, après avoir abordé les limites de notre modèle, nous terminons en exposant les perspectives de nos travaux.

11.1 Problématique et objectifs

Dans cette thèse nous nous sommes intéressés à l'élaboration de modèles globaux définissant et structurant l'activité commune d'agents autonomes. En effet l'autonomie d'un agent se traduit par sa capacité à déterminer ses propres buts face à la configuration de l'environnement et de ses motivations. Du fait de cette autonomie, des conflits liés à l'existence de buts propres à chacun des agents autonomes d'une société d'agents peuvent être incompatibles, complémentaires ou en compétition avec les objectifs de la société. Des modèles de gestion de conflits existent, et dans le cadre de ce travail, nous nous sommes intéressés aux modèles organisationnels.

Nous avons donc défendu la thèse selon laquelle la définition et la mise en place de contraintes globales influençant le fonctionnement d'agents autonomes doit être faite de manière à prendre en compte l'éventualité que les agents puissent ne pas respecter les contraintes selon les contextes et les objectifs individuels de chacun. Afin d'atteindre ces objectifs, nous avons proposé un modèle d'Institution Électronique devant comporter une description explicite et déclarative des contraintes à appliquer aux agents ainsi qu'un système de supervision et de renforcement de ces contraintes.

11.2 Contributions

Dans ce manuscrit, nous avons proposé un modèle d'***Institution Électronique*** que nous avons intitulé $\mathbf{Ma}\mathcal{B}_{\mathbf{eli}}$ (*Multi-Agent Based ELectronic Institution*) composé d'un ***Modèle de Spécification d'Institution*** (SIM) basé sur un modèle organisationnel et normatif ainsi que d'un ***Système d'Arbitrage d'Institution*** (IAS) vérifiant qu'un ensemble d'agents hétérogènes participant à l'organisation respectent la SIM et sanctionnant les agents si nécessaire.

11.2.1 Modèle de Spécification d'Institution

Le Modèle de Spécification d'Institution que nous avons présenté se nomme $\mathcal{M}\mathbf{oise}^{Inst}$ (*Model of Organisation for multI-agent SystEm in Institution*) et permet de définir une organisation normative. Les organisations définies avec $\mathcal{M}\text{OISE}^{Inst}$ se composent d'une Spécification Structurelle permettant aux agents de jouer un ensemble de rôles au sein de groupes et d'avoir un certain nombre de liens entre eux. Une organisation comporte également une Spécification Fonctionnelle définissant les buts de l'organisation selon un ensemble de schémas fonctionnels composés de buts regroupés en missions.

Enfin, les Spécifications Contextuelles et Normatives définissent les contextes dans lesquels les normes obligeant, autorisant ou interdisant les agents jouant des rôles, à accomplir des missions. Ce modèle organisationnel sert non seulement à spécifier l'organisation normative des agents à l'application mais également à décrire le comportement des agents du Système d'Arbitrage de l'Institution Électronique. Ceci permet d'obtenir une flexibilité de la spécification de l'organisation des agents du domaine ou du système d'arbitrage ainsi qu'une ouverture des instances d'organisation.

Par rapport à d'autres modèles du domaine étudiés dans l'état de l'art de cette thèse, ce modèle permet d'organiser les agents au travers de plusieurs définitions :

- Via la définition d'une structure d'agents comme le font AGR, STEAM ou $\mathcal{M}$OISE$^+$. Cependant, en plus de définir une hiérarchie de rôles (STEAM), de rassembler les rôles en groupes et de définir une contrainte de compatibilité entre rôles obligeant un agent à jouer deux rôles (AGR), $\mathcal{M}$OISE^{Inst} permet d'autres contraintes telles les cardinalités et la définition d'autres liens entre les rôles.
- Via la définition du fonctionnement des agents comme le font TÆMS, STEAM et $\mathcal{M}$OISE$^+$ mais en ajoutant la notion de mission regroupant un ensemble de buts (ou de tâches) et la possibilité de définir le nombre d'agents autorisés à s'engager sur les missions. De plus, $\mathcal{M}$OISE^{Inst} permet de définir si les sous-buts d'un but doivent être atteints de manière séquentielle, en parallèle ou selon un choix.
- Via la définition des différents contextes dans lesquels peuvent se trouver les agents comme le font OMNI et ISLANDER. La façon dont ces deux modèles spécifient un enchaînement de contextes (qu'ils appellent scènes) contraint seulement quelques rôles au lieu de l'organisation dans son ensemble et ne permet pas de définir pour ISLANDER des contextes dans lesquels les agents pourront évoluer en même temps.
- Via la définition des différentes normes réglant le comportement des agents comme le font OMNI, ISLANDER et $\mathcal{M}$OISE$^+$. Cependant, bien qu'ayant une dimension normative, OMNI ne définit pas un ensemble de normes dans une spécification à part. Chaque définition de norme se trouve au sein de la définition d'un rôle ou d'une scène d'interaction. Quant aux normes de ISLANDER et de $\mathcal{M}$OISE$^+$, elles ne définissent pas de sanction, ni d'émetteur de la norme, ni de contrainte temporelle pour ISLANDER, et ce dernier ne permet pas de différencier les obligations des permissions.

Ceci nous conduit à une vision plus homogène et beaucoup plus complète de la définition du comportement d'une organisation d'agents, la rendant explicite et permettant ainsi une supervision par les agents de $\mathcal{S}$YNAI.

11.2.2 Système de Supervision d'Institution

Nous avons proposé un Système d'Arbitrage pour les Institutions spécifiées avec $\mathcal{M}$OISE^{Inst} s'intitulant $\mathcal{S}$**ynai** (*SYstem of Normative Agents for Institution*). Ce système propose une solution d'organisation générique mais flexible et modifiable de la supervision d'une Institution Électronique. Elle est constituée d'un ensemble d'agents institutionnels fournis par le modèle MA$\mathcal{B}_{\text{ELI}}$ dont les capacités et les rôles permettent de superviser le respect de chaque spécification. Pour cela, chaque agent organisé et supervisé par MA$\mathcal{B}_{\text{ELI}}$ se verra attribué un agent institutionnel spécifique reliant ces demandes d'action sur l'organisation et interagissant avec les autres agents de $\mathcal{S}$YNAI afin de répondre positivement ou non à leur requête. En cas de refus, une violation est détectée et un dernier agent institutionnel s'occupera de sanctionner l'agent fautif.

$\mathcal{S}$YNAI est composé de plusieurs agents ayant chacun un but de supervision à atteindre (spécifié avec $\mathcal{M}$OISE^{Inst}) permettant une répartition des tâches et une flexibilité dans l'arbitrage. La plupart des autres systèmes de gestion étudiés dans l'état de l'art ne possède qu'un module unique d'arbitrage pour toute l'organisation, à savoir KARMA pour Teamcore, un Notaire pour CAS, un *Contract Monitor* pour le système d'Organisations Virtuelles et enfin un OrgManager pour $\mathcal{S}-\mathcal{M}$OISE$^+$. Le modèle $\mathcal{M}$OISE^{Inst} définissant des sanctions, $\mathcal{S}$YNAI est capable de superviser le respect des normes et

des contraintes en général contrairement au modèle Teamcore qui dirige l'Organisation en choisissant à un moment précis quel agent doit agir, ou à $\mathcal{S}-\mathcal{M}$OISE$^+$ et AMELI qui se servent du modèle de spécification d'institution pour contrôler le fonctionnement des agents. Ce contrôle entraîne le fait que les agents soient obligés de respecter les contraintes.

Tous ces agents doivent coopérer afin de superviser et de détecter les éventuelles violations de la spécification de l'organisation ; pour cela un ensemble d'événements échangés au sein de messages structurés par des protocoles d'interactions sont définis. Ces événements permettent le déclenchement ou non d'actions spécifiques sur l'organisation. La capacité de communiquer des agents de $\mathcal{S}$YNAI est, de ce fait, statique mais n'empêche pas le système d'arbitrage d'être générique. Nous avons illustré les aspects de cette généricité à travers la validation de deux applications issues de deux domaines différents.

11.2.3 Validation

La validation du modèle d'Institution Électronique s'est effectuée dans deux domaines différents mais sur deux applications complémentaires dans lesquelles les agents organisés ont été mis en situation d'autonomie vis-à-vis de la Spécification de l'Institution. Ces domaines sont ceux de la télévision numérique interactive et du *e-contracting*.

Dans le premier domaine, nous avons mis en œuvre, au travers d'une application de jeu télévisé, une Institution Électronique composée d'un ensemble d'Avatars représentant les téléspectateurs. La Spécification de l'Institution a permis de définir la structure de l'équipe des Avatars ainsi que son fonctionnement et les règles du jeu à respecter grâce aux normes. Une seule institution nous a suffis pour spécifier le comportement des Avatars, avant, pendant et après une partie ; le comportement de l'Avatar assistant le maître du jeu et posant les questions, vérifiant les réponses et ajustant les scores ainsi que le comportement des agents de $\mathcal{S}$YNAI supervisant le respect des règles du jeu et l'application des sanctions.

Dans le domaine du *e-contracting*, nous avons mis en place le modèle MA$\mathcal{B}_{\text{ELI}}$ dans une application de gestion de contrats électroniques au sein desquels les agents assistent les utilisateurs. La Spécification de l'Institution nous a permis de définir un modèle de contrat dont les instances sont chacune supervisées par une instance de $\mathcal{S}$YNAI. Nous avons mis en œuvre un ensemble de contrats faisant ainsi intervenir un petit nombre d'agents capables de gérer les différents contrats de leur utilisateur afin de les faire participer à plusieurs Institutions Électroniques en même temps et, de ce fait, les soumettre à des contraintes différentes voire conflictuelles.

11.3 Limites

Nous exposons dans cette section les critiques que nous pouvons faire sur notre modèle ainsi que sur sa validation.

11.3.1 Modèle de Spécification d'Institution

Analyse et spécification des actions et événements

En ce qui concerne le modèle $\mathcal{M}$OISE^{Inst}, nous avons vu que pour spécifier une organisation, nous avions besoin de spécifier également un ensemble d'événements, un ensemble d'actions et un ensemble de buts. Les événements sont ensuite utilisés au sein de la spécification contextuelle afin de définir les transitions entre contextes, les buts sont utilisés au sein de la spécification fonctionnelle afin de définir les objectifs des agents et enfin les actions sont utilisées au sein de la spécification normative comme éventuel objet sur lequel s'appliquent des normes.

Afin d'agir sur les instances d'organisation, nous avons défini un ensemble d'actions permettant aux agents via leur membre d'adopter des rôles et de satisfaire des buts. Ces actions sont mises en place lorsqu'un agent envoie un message contenant un événement à un agent institutionnel. Les actions et événements sont ici définis séparément des événements et des actions de l'OS alors qu'ils sont pourtant similaires.

Enfin, les agents dont nous avons défini l'architecture et le fonctionnement interne (les superviseurs de $\mathcal{S}$YNAI, les Avatars et les PeMA) mettent en place un ensemble d'actions (définies au sein de chaque agent) afin d'atteindre les buts sur lesquels ils sont engagés. Ces actions là ne sont pas déclenchées par des événements mais visent à satisfaire des buts. Nous avons donc un ensemble d'actions relatives à la spécification pouvant faire l'objet de normes, un ensemble d'actions déclenchées par des événements quand il s'agit de l'organisation et un ensemble d'actions permettant la satisfaction d'un but lorsqu'il s'agit de l'application au sein de laquelle l'Institution est utilisée. Comme nous définissons la structure et le fonctionnement des agents du domaine et de $\mathcal{S}$YNAI, nous devrions pouvoir définir l'action des différents agents. Ces actions seraient éventuellement déclenchées par un événement, satisferaient potentiellement un but et pourraient faire l'objet de normes.

Interactions entre agents

La définition que nous avons donné d'une Institution Électronique est construite autour d'un modèle organisationnel spécifiant la structure des agents ainsi que leur fonctionnement. Le modèle ISLANDER dont l'objectif était de faciliter les échanges entre agents au cours de séances d'enchères ou de négociation, définit les Institutions Électroniques autour d'un ensemble de scènes spécifiées par un ensemble de protocoles d'interaction que les agents devront suivre. Des démarches différentes ont donc abouti à des modèles complémentaires.

Tout comme une définition de rôle existe au sein de ISLANDER, les interactions au sein de MA$\mathcal{B}_{\text{ELI}}$ sont définies à l'intérieur des agents de $\mathcal{S}$YNAI, y compris au sein de l'OrgWrapper permettant aux agents du domaine de connaître les protocoles à respecter. L'internalisation de cette spécification empêche une définition des interactions entre les agents du domaine et leur supervision par un système d'arbitrage. Même si nous avons vu au travers de nos deux applications de validation que leur définition n'est pas essentielle, il serait intéressant de les externaliser de la définition des agents.

11.3.2 Mise en œuvre du modèle

La validation du modèle d'Institution Électronique a été faite dans les domaines de la télévision numérique interactive et du *e-contracting* dans lesquels les agents organisés étaient autonomes vis-à-vis de la spécification de l'organisation mais pas vis-à-vis de l'utilisateur. En considérant qu'un agent ait légalement le droit de signer un contrat et puisse automatiquement produire les délivrables, nous aurions pu imaginer tester les comportements des agents au sein de plusieurs institutions liées les unes aux autres. Contrairement à l'application de commerce électronique où les agents pouvaient être engagés sur plusieurs contrats (donc faisant partie de plusieurs institutions), le passage d'une institution à une autre ou l'évolution dans deux institutions pourrait être spécifié et supervisé par une autre institution.

11.4 Perspectives

Afin de répondre aux limites identifiées précédemment, nous exposons ici les perspectives d'évolution de nos travaux.

11.4.1 Spécification Ontologique

Afin de rendre cohérente la définition des différentes actions possibles au sein d'une organisation sur elle-même ou sur l'application l'utilisant, nous envisageons de mettre en place une autre spécification que nous nommerons ***Spécification Ontologique*** (*Ontological Specification* ou *OnS*). Cette spécification permettrait de définir une ontologie organisationnelle associée à un langage de contenu capable de spécifier la sémantique de chaque action ainsi que son lien avec les buts à atteindre. Seuls les événements définis au sein de la CS subsisteraient tandis que ceux permettant aux agents de communiquer seraient regroupés avec la définition des actions et utilisés comme tels au sein des messages construits avec un ACL.

11.4.2 Spécification Interactionnelle

Seuls les messages que peuvent s'échanger les agents de $\mathcal{S}$YNAI sont définis. Afin de donner la possibilité aux utilisateurs de spécifier les messages que les agents du domaine peuvent échanger (au sein des contextes par exemple), nous envisageons d'ajouter également une ***Spécification Interactionnelle*** (*Interactional Specification* ou *IS*). Cette spécification permettrait, comme pour ISLANDER, de définir des protocoles d'interaction au sein des contextes. En plus des liens entre les rôles définis au sein de la SS, les interactions pourraient également être contraintes en étant l'objet des normes. La règle pourrait se faire sur l'interdiction, l'obligation ou la permission d'utiliser un type de performatif ou d'envoyer un type de message vers un rôle particulier.

11.4.3 Validation multi-institutionnelle

Concernant la validation et l'utilisation du modèle MA$\mathcal{B}_{\text{ELI}}$ dans des applications concrètes, nous pourrions imaginer le besoin qu'un ensemble d'agents organisés avec $\mathcal{M}$OISE^{Inst} a de se ré-organiser et cela au travers d'une gestion multi-institutionnelle des agents. Prenons pour exemple un ensemble de véhicules autonomes devant collaborer afin de mettre en place un ensemble d'actions sur une surface étendue et soumise à de forts dénivelés. Les obstacles de l'environnement ainsi que la topologie de leur réseau de communication pourraient peut-être pousser les agents à se ré-organiser ou à créer une sous-organisation indépendante vis-à-vis de la première mais dépendante de l'objectif global de la société d'agents. Une institution globale gérant l'exécution de deux sous-institutions et le passage éventuel d'un agent d'une institution à une autre institution fait partie d'une thématique que nous nommons "Multi-Institution".

Publications

[2007]

- B. Gâteau, O. Boissier and D. Khadraoui. ***Organisation multi-agent normative : modélisation et infrastructure***. In Journées Francophones sur les Systèmes Multi-Agents (JFSMA'07), Carcassonne - France, 17-19 October 2007. Acceptation rate : 43%.

- O. Boissier and B. Gâteau. ***Normative Multi-Agent Organizations : Modeling, Support and Control, Draft Version***. In G. Boella, L. van der Torre, and H. Verhagen (eds.). Normative Multi-Agent Systems. Dagsthul Seminar Proceedings 07122, Internationales Begegnungs und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl - Germany, 2007.

[2006]

- B. Gâteau. ***Using a normative organisational model to specify and manage an institution for multi-agent systems***. In European Workshop on Multi-Agent Systems (EUMAS'06), Lisbon - Portugal, 14-15 December 2006. Acceptation rate : 78%.

- B. Gâteau, O. Boissier, D. Khadraoui and E. Dubois. ***Controlling an Interactive Game With a Multi-agent Based Normative Organizational Model***. In Coordination, Organization, Institutions and Norms in agent systems workshop (COIN) at the 17th European Conference on Artificial Intelligence (ECAI), Riva del Garda - Italy, 28 August 2006. Acceptation rate : 50%.

- B. Gâteau, O. Boissier and D. Khadraoui. ***Multi-Agent-Based Support for Electronic Contracting in Virtual Enterprises***. In IFAC Symposium on Information Control Problems in Manufacturing (INCOM), Saint-Etienne - France, 17-19 May 2006. Acceptation rate : 62%.

- B. Gâteau, O. Boissier, D. Khadraoui and E. Dubois. ***Arbitration of autonomous multimedia objects with a multi-agent system***. In IEEE International Conference on Information & Communications Technologies : from Theory to Applications (ICTTA'06). Damascus - Syria, 24-28 April 2006. ISBN : 0-7803-9522-0.

- B. Gâteau. ***An Interactive Multi-Agent based TV Game Show Implementatio***n. In Interoperability for enterprise software and applications : proceedings of the workshops and the doctorial symposium of the second IFAC/IFIP (I-ESA) International Conference : EI2N, WSI, IS-TSPQ 2006, Panetto, H., Boudjlida. N. ed., ISTE. Bordeaux - France, March 20-24, 2006. ISBN : 1-905209-61-4.

- S. Renault, B. Gâteau, D. Mathevon and Y. Naudet. ***Implementation of RAMO-based Multimedia Applications on MPEG-4 Platform***. In the IEEE International Conference on Consumer Electronics (ICCE'06), Las Vegas, Nevada - USA, 9-11 January 2006. ISBN : 0-7803-9459-3/06.

[2005]

- B. Gâteau, O. Boissier, D. Khadraoui and E. Dubois. $\mathcal{M}$**oise**^{Inst} ***: An Organizational Model for Specifying Rights and Duties of Autonomous Agents***. In European Workshop on Multi-Agent Systems (EUMAS'05), Brussels - Belgium, 7-8 December 2005. Acceptation rate : 70%.

- D. Khadraoui and B. Gâteau. ***Prototype for an Agent-Based Electronic Contracting using an Organizational Model***. In International Conference on Intelligent Agents, Web Technology and Internet Commerce (IAWTIC 2005), Vienna - Austria, 28-30 November 2005. ISBN : 1740-88247-4.

- B. Gâteau, O. Boissier, D. Khadraoui and E. Dubois. $\mathcal{M}$**oise**^{Inst} ***: An Organizational Model for Specifying Rights and Duties of Autonomous Agents***. In 1st International Workshop on Coordination and Organisation (CoOrg 2005) affiliated with the 7th International Conference on Coordination Models and Languages, Namur - Belgium, 20-23 April 2005.

[2004]

- B. Gâteau, D. Khadraoui, O. Boissier and E. Dubois. ***Contract Model for Agent Mediated Electronic Commerce (Poster)***. In 3rd international joint conference on Autonomous Agents & Multi-Agent Systems (AAMAS), Columbia University, New York City - USA, 19-23 July 2004. ISBN : 1-58113-864-4. Acceptation rate : 26%.

- B. Gâteau, D. Mathevon, D. Khadraoui, O. Boissier and E. Dubois. ***Transforming a Secure eCommerce Application into a Multi-Agent based Solution for e-Contracting***. In International Conference on Intelligent Agents, Web Technology and Internet Commerce - IAWTIC'2004, Canberra - Australia, 12-14 July 2004. ISBN : 1740-88189-3.

- B. Gâteau, D. Khadraoui and E. Dubois. ***Architecture e-Business sécurisée pour la gestion des contrats***. In 3ème Conférence sur la Sécurité et Architectures Réseaux (SAR), La Londe, Côte d'Azur - France, 21-25 June 2004.

- B. Gâteau, D. Khadraoui, O. Boissier and E. Dubois. ***Multi-Agent Organizational Model for e-contracting (Poster)***. In 6th International Conference on Enterprise Information Systems (ICEIS), Universidade Portucalense, Porto - Portugal, 14-17 April 2004. ISBN : 972-8865-00-7.

Cinquième partie

Annexes

Annexe A

Contrats Électroniques et Systèmes Multi-Agents

A.1 Contrats sociaux

Un contrat peut être défini globalement par une expression d'intention qui dirige le comportement des organisations et des individus [74]. Les contrats ont été proposés comme un moyen de rendre explicite la façon dont les agents interagissent avec et dans la société.

A.1.1 Contractual Agent Society

Dellarocas définit dans CAS (*Contractual Agent Society*) des contrats sociaux qu'ils nomment également *social norms* [25]. Ces contrats sociaux spécifient tous les éléments d'une CAS qui gouvernent les interactions d'un membre avec le reste de la société. Intuitivement ils définissent les droits et devoirs d'un agent au sein de la société dans laquelle il se trouve. Ils incluent les croyances, les valeurs, les objectifs, les protocoles et les polices que deux agents ou plus acceptent de respecter dans le contexte d'une relation sociale. La définition d'un contrat telle que l'entend Dellarocas est apparentée à la notion d'engagement social (*social commitment*) défini par Singh dans [91]. Un contrat se définit par une ou plusieurs clauses. Une clause C se représente par $C(x, c, b, s)$ avec :

- x = les agents contractant
- c = le contexte du groupe (distinct des parties contractantes, on peut considérer cela comme l'autorité du groupe)
- b = le corps du contrat (soit un ensemble de clauses C'(x', c', b', s') soit une clause primitive étant ce sur quoi porte cette clause (croyance, valeur, objectif, etc., cf. paragraphe précédent)
- s = le graphe d'états/transitions du contrat (représente les connexions entre les clauses des contrats et le contrôle social. L'objectif est de rester dans un état stable.)

Les sanctions sont appliquées aux agents qui sont responsables du passage à un état non stable (idem pour la récompense si passage à un état stable). Chaque clause est associée à un graphe. Les transitions du graphe du contrat sont fonction des sous-transitions des clauses. De même entre une clause et ses sous-clauses.

A.1.2 Electronic Contract Framework

Electronic Contract Framework (ECF) [88] est une plate-forme basée sur les agents pour l'automatisation des relations contractuelles. Ces travaux sont surtout situés dans le domaine du B2B et basés sur ceux de Castelfranchi *et al.* [19], de Dellarocas [25] et de Kollingbaum et Norman [67] qui considèrent que les agents peuvent évoluer dans un environnement réglementé et peuvent adapter leurs

comportements par rapport à leur perception des normes spécifiées dans les contrats électroniques.

Un contrat est composé d'une section informative regroupant un identifiant, un *mapping* entre les rôles du contrat et les identités réelles des contractants, une période de validité et une section de spécification comportementale regroupant un ensemble d'expressions normatives. Ces expressions sont considérées comme étant le comportement que chacune des parties est censée suivre. Ainsi les contrats ne sont pas des contraintes fortes sur le comportement des agents et nous avons affaire à une approche délibérative des expressions normatives. Ces expressions (ns) sont composées de deux parties : $ns : \varphi \rightarrow \theta_{s,b}(\alpha < \psi)$ voulant dire que si φ est vrai, alors s a {l'obligation, la permission, l'interdiction} (θ) envers b d'accomplir α {avant, jusqu'à ce} que ψ soit vrai. La partie φ peut être une condition de respect d'une autre expression normative. Les ns sont relatives à leurs sanctions associées.

Il existe deux types de classes de sanctions, les sanctions endogènes qui peuvent aboutir sur une continuité du contrat et les sanctions exogènes qui débouchent sur un état critique non résolvable automatiquement (cas de non respect d'une sanction endogène par exemple). Les sanctions peuvent entraîner une obligation ou une interdiction etc. : $ns' : \mathit{not\_fulfilled}(ns) \rightarrow \theta_{s,b}(\alpha' < \psi')$

Les opérateurs déontiques sont dynamiques, de ce fait on peut associer un cycle de vie à chaque opérateur (par exemple une obligation en état *agreed* peut ensuite être dans les états *refused*, *accepted*, *cancelled*, *in progress*, *rejected* ou encore *fulfilled*. Une observation se fait via une comparaison entre les comportements attendus (le cycle de vie) et les comportements actuels des parties contractantes. Les opérateurs ont deux types d'état, les états internes et les états partagés. Les agents d'un même contrat doivent avoir la même vision des états partagés de ses opérateurs. Il faut pour cela une synchronisation des vues ou des croyances de chacun. Un agent ne peut pas faire l'hypothèse aveuglement que chaque partie contractante va se comporter comme convenu (ce qui pourrait influencer leurs propres comportements). Un besoin de prédiction dynamique du comportement des partenaires est donc nécessaire. Pour ces deux besoins, Sallé propose le CFP (Contract Fulfilment Protocol), protocole collaboratif basé sur le cycle de vie des opérateurs déontiques et utilise la théorie des actes de langage. Ainsi un agent peut informer son partenaire qu'il accepte une obligation. Le partenaire peut ensuite demander si la norme a été respectée par l'initiateur du dialogue qui peut lui répondre par une information comme quoi il a respecté sa règle. La synchronisation d'état "réalisé" suit donc la prédiction des deux parties.

A.1.3 Logic for Contract Representation

Virginia Dignum définit dans [31] ainsi que dans le chapitre 3 de [30] les contrats sociaux comme des tuples composés d'agents, de clauses et d'étapes. En plus des parties impliquées directement dans le contrat, les auteurs supposent que les contrats peuvent également définir une troisième partie représentée par un ou plusieurs agents étant en charge de la supervision de l'exécution du contrat et l'application de sanctions le cas échéant.

Un contrat social SC est défini par le triplet $SC = (A, M, CC)$ où :
- A est un ensemble d'agents (il peut n'y en avoir qu'un seul, représentant ainsi l'engagement interne d'un agent).
- M est un ensemble (potentiellement vide) d'agents de supervision responsables du contrôle de la satisfaction du contrat et de l'application de sanctions.
- CC est un ensemble de clauses de contrats.

Une clause de contrat est une expression déontique qui décrit les conditions et une limite de temps pour une obligation, interdiction ou permission spécifique sur l'activité d'un agent jouant un rôle. Les clauses de contrat sont spécifiées formellement comme des expressions déontiques définies avec LCR (the Logic for Contract Representation). LCR est une logique décrivant les interactions dans les systèmes multi-agent rendant possible la description et la vérification de contrats spécifiant les interactions entre agents (cf. Chapitre 4 de [30]).

Les contrats d'interactions (cf. chapitre 3 de [30]) décrivent quant à eux l'interprétation réelle d'un script de scène en fonction des agents jouant les rôles d'une structure de rôles (nous reviendrons là-dessus dans la Section 3.2.2). C'est à dire qu'ils décrivent les conditions et les règles qui sont appliquées sur les interactions entre les agents d'une société.

Un contrat d'interaction IC est défini par le tuple $IC = (A, s, CC, P)$ où :

- A est un ensemble d'agents,
- s est une scène,
- CC est un ensemble de clauses de contrat et
- P est un protocole qui doit être suivi par les agents.

Ces modèles de contrat ne fournissent pas explicitement de spécification structurelle ou fonctionnelle. Les interactions entre agents ne sont pas abordées non plus à part pour LCR où les interactions sont spécifiées avec des contrats. Les clauses des contrats peuvent être considérées comme un ensemble de contraintes normatives ayant pour objet les agents eux-mêmes. La spécification des contrats est en général très abstraite et ne concerne que deux agents à la fois pour CAS et ECF et de 1 à n agents pour LCR (sans compter les agents servant de tiers de confiance). Ces spécifications abstraites de contrats font intervenir les notions d'agents. De ce fait, les définitions de contrainte sont très proches des agents et donc moins flexibles en matière de re-définition ou de ré-organisation des contraintes. Dans la dernière section de ce chapitre, nous allons voir si l'utilisation de contraintes organisationnelles associées à des expressions normatives permet de ne pas faire référence directement aux agents et donc leur imposer le respect des contraintes.

Annexe B

$\mathcal{M}$oise^{Inst} model specification with the BNF language

A $\mathcal{M}$OISE^{Inst} model is based on a vocabulary composed of a set of goals, of events, of contexts and of defined as following :

⟨goal⟩ ::= '(goal' :id ⟨goalId⟩ :label ⟨String⟩')'
⟨context⟩ ::= '(context' :id ⟨contextId⟩ :label ⟨String⟩')'
⟨event⟩ ::= '(event' :id ⟨eventId⟩ :label ⟨String⟩')'
⟨action⟩ ::= '(action' :id ⟨actionId⟩ :label ⟨String⟩')'

A goal is a state in which the system have to be. The agents have to act in order to reach goals. A context defines a particular state of the system. The contexts *Start* and *Stop* are keywords of the specification and represent initial and final contexts of the CS. An event defines a message sent to all in the system announcing something which happens. Finally an action is something that an agent is able to do.

An Organisation Specification (OS) of a $\mathcal{M}$OISE^{Inst} model is composed of :

- A Structural Specification (SS)
- A Functional Specification (FS)
- A Contextual Specification (CS)
- A Normative Specification (NS)

⟨OS⟩ ::= '(os' :id ⟨osId⟩ :ss ⟨SS⟩ :fs ⟨FS⟩ :cs ⟨CS⟩ :ns ⟨NS⟩')'

B.1 Structural Specification

In $\mathcal{M}$OISE^{Inst} concepts of role, role relation and group are used to build, respectively, the individual, social, and collective structural levels of an organisation with concepts of inheritance, compatibility, cardinality, and sub-groups.

⟨SS⟩ ::= '(SS' :role ⟨role⟩* :link ⟨link⟩* :group ⟨group⟩*')'
⟨sEntityId⟩ ::= ⟨roleId⟩ | ⟨groupId⟩
⟨group⟩ ::= '(' :id ⟨groupId⟩ :role ⟨roleId⟩* [:link ⟨linkId⟩*] [:subgroup ⟨group⟩*] [:constraint ⟨constraint⟩*] [:min ⟨integer⟩ :max ⟨integer⟩]')'
⟨role⟩ ::= '(' :id ⟨roleId⟩ [:extends ⟨roleId⟩]')'
⟨link⟩ ::= '(' :id ⟨linkId⟩ :source ⟨sEntityId⟩ :target ⟨sEntityId⟩ :type ⟨linkT⟩ :scope ⟨scope⟩ :extendsSG (true|false)')'
⟨linkT⟩ ::= 'Authority' | 'Communication' | 'Acquaintance'
⟨scope⟩ ::= 'intra-group' | 'inter-group'
⟨constraint⟩ ::= '('[:compatibility ⟨compatibility⟩*] [:cardinality ⟨cardinality⟩*]')'
⟨compatibility⟩ ::= '(' :role1⟨roleId⟩ :role2 ⟨roleId⟩ :scope ⟨scope⟩ :extendsSG (true | false)')'
⟨cardinality⟩ ::= '(' :object ('role'|'grSpec') :id ⟨sEntityId⟩ :max ⟨integer⟩ :min ⟨integer⟩')'

B.1.1 SEntityId

This represents a basic structural entity identification of an organisation. It can be a groupId or a roleId.

B.1.2 Group

This is the entity where a set of not abstract roles may be played.

Attributes

id	Identifier of the group.
role	Set of not abstract roles included in the group.
link	Set of links among roles in the group.
subgroups	Groups included in the defined group.
constraint	Define the compatibilities and cardinalities of the group (see after).
max	Define the maximum number of agents authorized in the group.
min	Define the minimum number of agents authorized in the group (by default, max and min are calculated with cardinalities constraints).

B.1.3 Role

A role means a set of constraints that an agent ought to follow when it accepts to enter a group playing that role.

Attributes

id	Identifier of the role (its name most of time).
extends	Inheritance link toward a sub-role.
abstract	Is the role abstract or not.

B.1.4 Link

Links directly constrain the agents. In case the link type is acquaintance, the agent playing the source role is allowed to have a representation of the agent playing the destination role. In a communication link, the agents are allowed to communicate together. In a authority link, the agent playing the source role is allowed to have authority on the agent playing the destination role, i.e., to control it. An authority link implies the existence of a communication link that implies the existence of an acquaintance link.

Attributes

target	The link destination.
source	The link source.
linkT	The link type ('Authority', 'Communication' or 'Acquaintance).
scope	Define if the roles implied in the link are in the same group ('intra-group') or in different groups ('inter-group').

B.1.5 Constraint

This can define constraints in a collective level as cardinalities and compatibilities.

Attributes

cardinality	The set of cardinality constraints.
compatibility	The set of compatibility constraints.

B.1.6 Cardinality

This can define the cardinality of role and sub-groups. We can specify the number (minimum, maximum) of roles that have to be played in the group. Analogously, we can specify the number of sub-groups in a group. By default, cardinality pairs are $(0, \infty)$.

Attributes

object	Specify the entity which is concerned by the constraint, a role or a group.
id	The identifier of the entity concerned.
max	Maximum number of entities in the organisation.
min	Minimum number of entities in the organisation.

B.1.7 Compatibility

This compatibility constraint states that the agents playing the role p_a are also allowed to play the role p_b (it is a reflexive and transitive relation). By default, two roles are not compatible.

Attributes

role1	The first role of the relation.
role2	The second role of the relation.
scope	Define is the roles implied in the link are in the same group ('intra-group') or in different groups ('inter-group').
extendsSG	Define if compatibility is the same in sub-groups.

B.2 Functional Specification

The FS in $\mathcal{M}$OISE^{Inst} is based on the concepts of missions (a set of global goals) and global plans (the goals in a structure). These two concepts are assembled in a Scheme.

⟨FS⟩ ::= '(FS' :schemes ⟨scheme⟩* [:preferences ⟨preference⟩*]')'
⟨scheme⟩ ::= '(' :id ⟨schemeId⟩ :root ⟨goalId⟩ [:plans ⟨plan⟩*] [:missions ⟨mission⟩*]')'
⟨plan⟩ ::= '(' :source ⟨goalId⟩ [:success ⟨int⟩] :operator ('sequence' | 'choice' | 'parallel') [:goals ⟨goalId⟩*][:schemes ⟨schemeId⟩*]')'
⟨mission⟩ ::= '(' :id ⟨missionId⟩ [:max ⟨integer⟩][:min ⟨integer⟩] :goals ⟨goalId⟩*')'
⟨preference⟩ ::= '(' :mission ⟨missionId⟩ :prefered ⟨missionId⟩')'

B.2.1 Scheme

It is essentially a goal decomposition tree where the root is the scheme goal and where the responsibilities for the sub-goals are distributed in missions.

Attributes

id	Identifier of the scheme.
root	The root goal of the scheme.
goals	The set of goals included in the scheme.
plans	The set of plans included in the scheme.
missions	The set of mission included in the scheme.

B.2.2 Plan

This concept assembles goals in a structure. Each goal may be decomposed in sub-goals through plans which may use three operators : sequence, choice, or parallelism.

Attributes

source	The root goal of the plan.
success	A certainty success degree of the plan.
operator	How the sub-goals have to be achieved : sequentially, by choice or parallel ?
goals	Sub-goals included in the plan.
schemes	Sub-scheme included in the plan as the same level as a simple sub-goal.

B.2.3 Mission

A mission is a set of coherent goals that an agent can commit to. It is possible to define a preference order among the missions (in the FS level).

Attributes

id	Identifier of the mission.
max	The maximum of agents authorized to commit the mission.
min	The minimum of agents authorized to commit the mission.
goals	The set of goals composed the mission.

B.3 Contextual Specification

The CS define a state chart (a state is a context) which the organisation will follow. A context can be composed of sub-contexts. A role can be in several contexts at the same time and these contexts have not composition links between them (one is not composed by another one). But a norm (see section about NS) concerns only one context.

⟨CS⟩ ::= '(CS' :context ⟨contextDesc⟩* :transition ⟨transition⟩*
:initialCtxt ⟨contextId⟩ :finalCtxt ⟨contextId⟩')'
⟨contextDesc⟩ ::= '(' :id ⟨contextId⟩ [:subcontext ⟨CS⟩*]')'
⟨transition⟩ ::= '(' :source ⟨contextId⟩ :target ⟨contextId⟩ [:event ⟨eventId⟩]')'

The *initialCtxt* attribute is an initial context. In a CS, the transition from an initial context may be labeled with the trigger event ; otherwise, it must be unlabeled. If it is unlabeled, it represents any transition from the enclosing CS. There can be at most one initial context in a CS.

The *finalCtxt* attribute is a special kind of context signifying that the enclosing CS is completed. If the enclosing CS is directly contained in a context and all other CS in the context also are completed, then it means that the entire context is completed. A final context cannot have any outgoing transitions.

B.3.1 ContextDesc

It describes a context with other contexts which compose it. *ContextDesc* is the composition of a *Context*.

Attributes

id	Identifier of the context.
subcontext	The set of sub-context composing a sub-CS.

B.3.2 Transition

Define the transition between a source context and a target context.

Attributes

id	Identifier of the transition.
source	The source context.
target	The target context.
event	The event triggering the transition.

La CS d'une OS décrit l'ensemble des contextes, a priori, au sein desquels l'organisation pourra évoluer ainsi que les règles de transition d'un contexte à l'autre. Elle est définie de la manière suivante :

'(CS' :context ⟨contextDesc⟩* :transition ⟨transition⟩* [:initialCtxt ⟨contextId⟩ :finalCtxt ⟨contextId⟩]')'

.

⟨contextDesc⟩ décrit un état global dans lequel l'Organisation se trouve. Sa définition

⟨contextDesc⟩ ::= '(' :id ⟨contextId⟩ [:subcontext ⟨CS⟩*]')'

permet d'inclure des sous-contextes définis de manière récursive. Deux contextes spéciaux (*Start* et *Stop*) peuvent également être référencés. Ils correspondent respectivement aux contextes initiaux et finaux de la CS (points noirs et points noirs entourés d'un cercle). Les différents sous-contextes peuvent évoluer en parallèle. ⟨transition⟩ définit une transition unidirectionnelle entre deux contextes de la manière suivante :

⟨transition⟩ ::= '(' :source ⟨contextId⟩ :target ⟨contextId⟩ [:event ⟨eventId⟩]')'.

Le déclencheur du franchissement de la transition correspond à l'apparition d'un évènement ⟨event⟩ dans l'Organisation. Les évènements dépendent de l'application.

B.4 Normative Specification

The NS specifies relations between SS, FS and CS as permissions and obligations of a role on a mission in a special context. The NS is composed of norms. Norms define deontic relation between two roles, within a context and on a mission. A condition is optional. Each norm has a weight which a priority order for agents in case of conflict between several norms. A norm can make reference to a sanction which is another norm.

⟨NS⟩ ::= '(NS' ⟨norm⟩*')'
⟨norm⟩ ::= '(Norm' :id ⟨normId⟩ :weight ⟨int⟩ [:conditions ⟨condition⟩]
:operator ⟨deonticRel⟩ :bearer ⟨sEntityId⟩ :issuer ⟨sentityId⟩ :context ⟨contextId⟩
:action ⟨deonticAct⟩ [:relation ⟨relation⟩ :deadline ⟨date⟩][:sanction ⟨normId⟩]')'
⟨condition⟩ ::= '('(⟨condition⟩ 'AND' ⟨condition⟩) | (⟨condition⟩ 'OR' ⟨condition⟩) |
(⟨function⟩ ['!'] '=' ⟨value⟩)')'
⟨function⟩ ::= 'violated('⟨normId⟩')' | 'respected('⟨normId⟩')' | 'number('⟨groupId⟩')' |
'cardinalityMax('⟨groupId⟩')'
⟨deonticRel⟩ ::= 'O' | 'P' | 'F'
⟨deonticAct⟩ ::= 'execute('⟨missionId⟩')' | ⟨actionId⟩
⟨relation⟩ ::= '>' | '<' | '='

B.4.1 Norm

A norm states that an agent playing the role implied is obliged, permitted, forbidden to commit to the mission considered in the context indicated and in the period listed.

Attributes

id	Identifier of the norm.
weight	The weight of the norm defined as an integer in order to define a kind of priority in the NS.
conditions	A set of conditions defining the validity of the norm.
operator	Define the deontic operator of the norm (Obligation, Permission, Prohibition).
bearer	Define the role concerned by the norm. Its value could equal a role or a group ID.
issuer	Define the structural entity which issues the norm and so controls if it is respected.
context	Define the context in which the norm is applied. A norm has only one context of application.
action	Define the action to do in order to commit, respect the norm.
relation	Define the time relation of the norm.
deadline	Define a time of validity for the norm.
sanction	Define a reference of another norm which is valid if this norm isn't respected.

B.4.2 Condition

This is the part which define if the norm is valid or not. The conditions could be composed of conditions and logic operators as AND or OR. A condition could be a test on a function result.

Attributes

function	Define the basic functions that agents from SYNAI could do.

Annexe C

Définition du modèle $\mathcal{M}$oise^{Inst} à l'aide d'une DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT OrganizationalSpecification (Property*, DomainKnowledgeSpecification,
   StructuralSpecification, FunctionalSpecification, ContextualSpecification*,
   NormativeSpecification)>
<!ATTLIST OrganizationalSpecification
   id CDATA #REQUIRED
>
<!ELEMENT DomainKnowledgeSpecification (Goal*, Context*, Event*, Action*)>
<!ELEMENT Context (#PCDATA)>
<!ATTLIST Context
   id CDATA #REQUIRED
>
<!ELEMENT Event (#PCDATA)>
<!ATTLIST Event
   id CDATA #REQUIRED
>
<!ELEMENT Goal (#PCDATA)>
<!ATTLIST Goal
   id CDATA #REQUIRED
>
<!ELEMENT Action (#PCDATA)>
<!ATTLIST Action
   id CDATA #REQUIRED
>
<!ELEMENT StructuralSpecification (Property*, RolesDefinition?, LinksType?,
                                   GroupSpecification?)>
<!ELEMENT RolesDefinition (Role)*>
<!ELEMENT Role (Property*, extends*)>
<!ATTLIST Role
   id CDATA #REQUIRED
>
<!ELEMENT extends EMPTY>
<!ATTLIST extends
   role CDATA #REQUIRED
>
<!ELEMENT LinksType (LinkType)*>
<!ELEMENT LinkType EMPTY>
```

```
<!ATTLIST LinkType
   id CDATA #REQUIRED
>
<!ELEMENT LoadGroupSpecification EMPTY>
<!ATTLIST LoadGroupSpecification
   uri CDATA #REQUIRED
>
<!ELEMENT GroupSpecification (Property*, Roles?, Links?, SubGroups?,
   ConstrainFormation?)*>
<!ATTLIST GroupSpecification
   id CDATA #REQUIRED
   max CDATA #IMPLIED
   min CDATA #IMPLIED
   arbitration (true|false) "false"
>
<!ELEMENT Roles (Role)*>
<!ELEMENT Links (Link)*>
<!ELEMENT Link (Property*)>
<!ATTLIST Link
   source CDATA #REQUIRED
   destination CDATA #REQUIRED
   type CDATA #REQUIRED
   scope (intra-group | inter-group) "intra-group"
   extendsToSubGroups (true | false) "false"
   symmetric (true | false) "false"
>
<!ELEMENT SubGroups (LoadGroupSpecification*, GroupSpecification*)>
<!ELEMENT ConstrainFormation (Compatibility*, Cardinality*)>
<!ELEMENT Compatibility EMPTY>
<!ATTLIST Compatibility
   source CDATA #REQUIRED
   destination CDATA #REQUIRED
   scope (intra-group | inter-group) "intra-group"
   extendsToSubGroups (true | false) "false"
   symmetric (true | false) "false"
>
<!ELEMENT Cardinality EMPTY>
<!ATTLIST Cardinality
   object (role | grSpec) #REQUIRED
   id CDATA #REQUIRED
   max CDATA #IMPLIED
   min CDATA #IMPLIED
>
<!ELEMENT FunctionalSpecification (Property*, Scheme*, Preference*)>
<!ELEMENT Scheme (Property*, Plan*, Mission*)>
<!ATTLIST Scheme
   id CDATA #REQUIRED
   rootGoal CDATA #IMPLIED
>
<!ELEMENT Plan (Property*, GoalId*)>
<!ATTLIST Plan
   headGoal CDATA #REQUIRED
   successRate CDATA #IMPLIED
   operator (sequence | choice | parallel) #REQUIRED
```

```
>
<!ELEMENT Mission (Property*, GoalId*)>
<!ATTLIST Mission
   id CDATA #REQUIRED
   max CDATA #IMPLIED
   min CDATA #IMPLIED
>
<!ELEMENT GoalId (#PCDATA)>
<!ELEMENT Preference EMPTY>
<!ATTLIST Preference
   mission CDATA #REQUIRED
   preferable CDATA #REQUIRED
>
<!ELEMENT Property EMPTY>
<!ATTLIST Property
   id CDATA #REQUIRED
   value CDATA #REQUIRED
>
<!ELEMENT ContextualSpecification (ContextDesc*, Transition*)>
<!ATTLIST ContextualSpecification
   initialCtxt CDATA #IMPLIED
   finalCtxt CDATA #IMPLIED
>
<!ELEMENT ContextDesc (ContextualSpecification*)>
<!ATTLIST ContextDesc
   id CDATA #REQUIRED
>
<!ELEMENT Transition EMPTY>
<!ATTLIST Transition
   id CDATA #REQUIRED
   source CDATA #REQUIRED
   target CDATA #REQUIRED
   eventId CDATA #IMPLIED
>

<!ELEMENT NormativeSpecification (Norm)*>
<!ELEMENT Norm (Condition*)>
<!ATTLIST Norm
   id CDATA #REQUIRED
   weight CDATA #REQUIRED
   operator (O | P | F) #REQUIRED
   bearer CDATA #REQUIRED
   issuer CDATA #REQUIRED
   context CDATA #REQUIRED
   actionT (mission | action) #REQUIRED
   action CDATA #REQUIRED
   timeRelation (before | after | during | when) #IMPLIED
   timeConstraint CDATA #IMPLIED
   sanction CDATA #IMPLIED
>
<!ELEMENT Condition (LogFunc | TestFunc)?>
<!ELEMENT LogFunc (Condition+)>
<!ATTLIST LogFunc
   operator (AND | OR) #REQUIRED
```

```
>
<!ELEMENT TestFunc EMPTY>
<!ATTLIST TestFunc
   id (violated | respected | number | cardinality) #REQUIRED
   param CDATA #IMPLIED
   equals (yes | no) #REQUIRED
   value CDATA #REQUIRED
>
```

Annexe D

Les EJB dans EBSME

D.1 Enterprise Java Bean

Un EJB permet de définir et d'implémenter des composants logiciels qui encapsulent souvent le deuxième-tiers ("application serveur") d'une application métier, où la logique métier est implémentée en utilisant des EJB, informant le premier-tiers ("front end systems") et en utilisant la base de données du troisième-tiers ("back end systems"), comme illustré sur Fig. D.1.

Deux types d'EJB existent. Ils fournissent les capacités suivantes :
- Les EJB de type "Session" (*Session Bean*) exécutent une tâche pour le client et peuvent être avec ou sans état. Un composant avec état est dédié à un certain client pendant toute la durée de son instanciation. Ainsi toutes les invocations d'une méthode par le client seront traitées par le même EJB.
- Les EJB de type "Entity" (*Entity Bean*) représentent un objet métier qui existe dans le système de stockage permanent (en général une ligne d'une table relationnelle ou un objet persistant dans une base de données). Ce sont des serveurs pour des *Session Beans* le plus souvent. Ils servent de *proxy* entre la logique métier (traitement) et les bases de données.

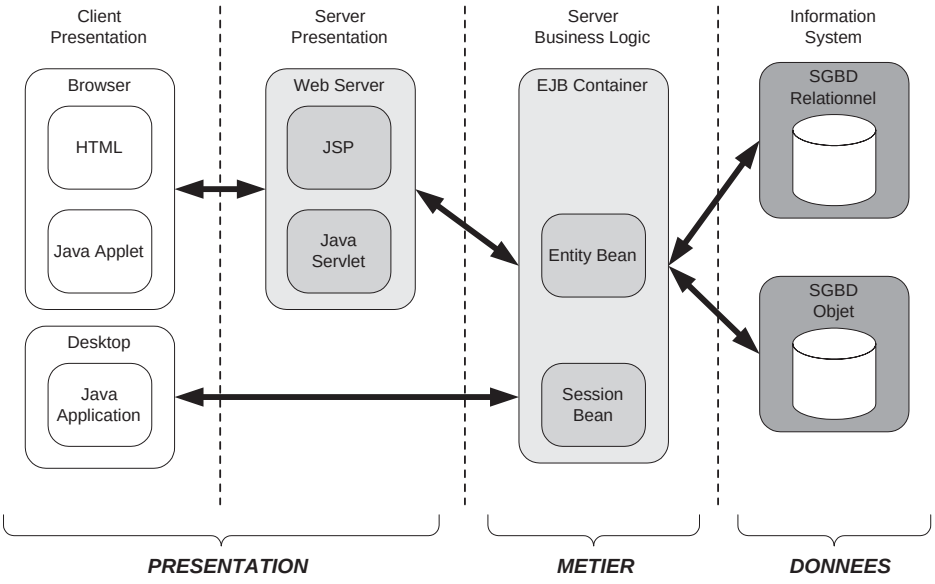


Fig. D.1 – Architecture J2EE 3-tiers

D.2 Les EJB de EBSME

Si nous revenons à notre application EBSME, nous avons un EJB de type entité pour chaque table de la base de données. Les EJB de type session permettent d'agir sur ces données via les EJB de type entité. L'applet JAVA cliente est donc une interface pour avoir un aperçu de ce qui se trouve dans la base de données et pour agir sur ces données par la création ou l'exécution de contrats électroniques par exemple.

Les EJB de la version non-agent d'EBSME et de type *Entity Bean* sont les suivants :
- *Participant* : les données d'un utilisateur.
- *User* : les données d'un utilisateur en relation avec l'utilisation du système (nom d'utilisateur, mot de passe hashé, ...).
- *Certificate* : le certificat d'un utilisateur permettant la vérification de sa signature.
- *TranslatorSkills* : les connaissances d'un utilisateur, en termes de langues connues.
- *JobProfile* : le profil d'un utilisateur par rapport à son travail (disponibilité, rôle, ...).
- *Signature* : une signature d'un document, associée au document et à un participant.
- *Document* : un document, lié soit à un contrat (accord de collaboration, le contrat papier réel) ou à un arbitrage (raison de la décision d'arbitrage).
- *Arbitration* : une décision d'arbitrage.
- *Contract* : un contrat électronique, lié à trois participants et à un document.
- *Deliverable* : un délivrable lié à un contrat.
- *Currency* : une devise reconnue par le système.

Ces EJB permettent d'accéder les tables de la base de données dont le schéma est donné sur la Fig. D.2.

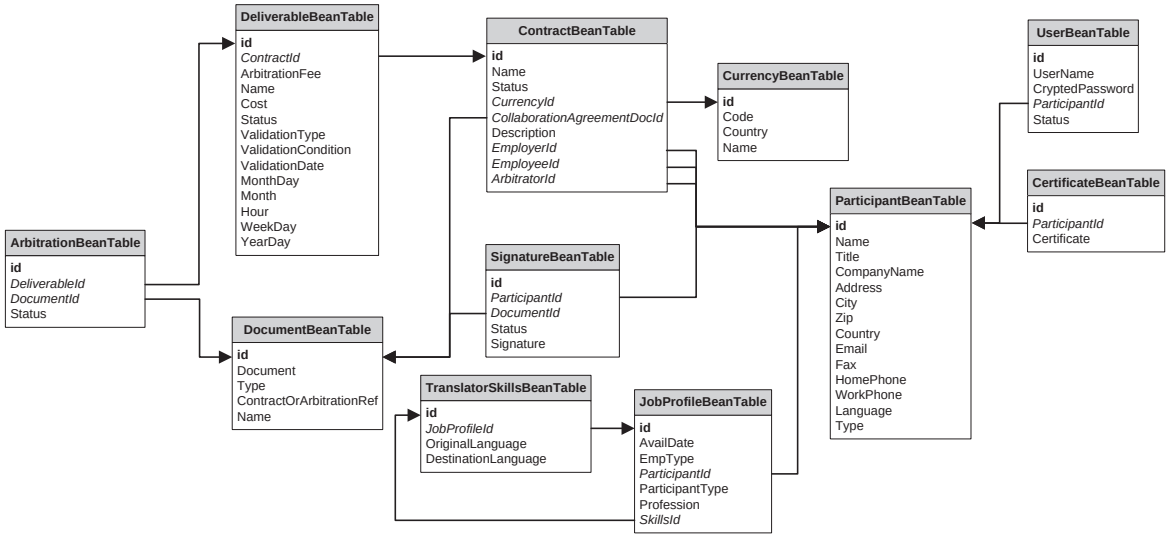


Fig. D.2 – Schéma de la base de données de l'application EBSME

Les *Session Bean* conçus pour réimplémenter la logique métier dans un composant serveur sont :
- *ContractController* : Ce contrôleur est en charge des entités "Contract". Il offre les fonctionnalités de consultation, d'ajout, de mise à jour et d'effacement de contrats.
- *CurrencyController* : Ce contrôleur gère les devises (entité "Currency") reconnues par l'application. Ces devises peuvent être choisies lors de la création d'un contrat. Il permet la consultation, l'ajout et l'effacement de devises.
- *DocumentController* : Le "DocumentController" permet la gestion des entités "Document" et de leurs signatures ("Signature"). Des documents peuvent être mis en place et consultés. Des

signatures peuvent être stockées, consultées et vérifiées à l'aide du certificat du participant impliqué.
- *DeliverableController* : Ce contrôleur est en charge de la gestion des délivrables via l'EJB "Deliverable", et des décisions d'arbitrage ("Arbitration") effectuées sur ceux-ci par un arbitre.
- *JobProfileController* : Le "JobProfileController" fournit les fonctions de gestion de profils de participant via l'EJB "JobProfile". Il effectue aussi les recherches d'employeurs et d'employés, et gère les connaissances des participants via l'EJB "TranslatorSkills".
- *ParticipantController* : Ce contrôleur gère les entités "Participant", "Certificate" et "User".

Le détail des entités et des contrôleurs conçus est présenté sur la FIG. D.3 sous forme de diagramme de classes du composant serveur. Nous retrouvons les *Entity Beans* en gris clair et les *Session Beans* en gris foncé. Les relations entre les entités y sont indiquées, ainsi que les relations entre les contrôleurs et les entités. Une relation du type "ejbref" indique qu'un contrôleur accède à une entité pour l'exécution d'une de ses méthodes. Nous retrouvons ainsi comme décrit précédemment des liens partant d'un contrôleur vers plusieurs entités différentes. Les entités possèdent les attributs dont les valeurs seront stockées dans la base de données. Les contrôleurs possèdent des méthodes pour accéder et modifier ces valeurs.

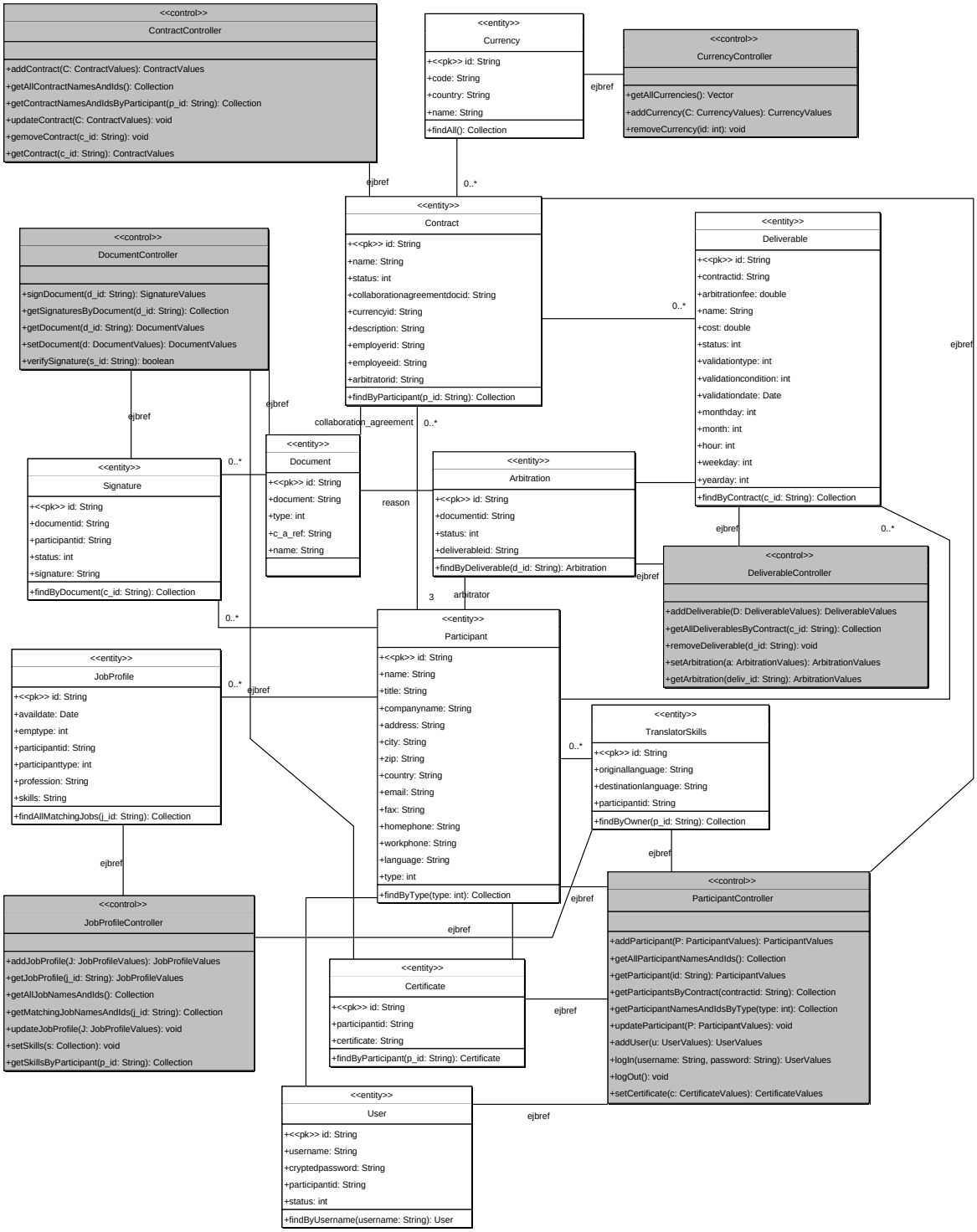


FIG. D.3 – Diagramme de classes du composant serveur

Bibliographie

[1] Grand dictionnaire terminologique. url : www.granddictionnaire.com.

[2] Oasis extensible access control markup language (xacml). url : www.oasis-open.org/committees/tc_home.php ?wg_abbrev=xacml.

[3] Organisation mondiale du commerce. url : www.wto.org.

[4] Wikipédia. url : fr.wikipedia.org.

[5] *Axis, Dictionnaire Encyclopédique*. Hachette, 1995.

[6] *Le petit Larousse illustré*. Larousse, 2003. ISBN 2-03-530204-8.

[7] Josep Ll. Arcos, Marc Esteva, Pablo Noriega, Juan A. Rodríguez-Aguilar, and Carles Sierra. Engineering open environments for multiagent systems. *Engineering Applications of Artificial Intelligence*, 18(2) :191–204, 2004.

[8] José Báez, Tiberiu Stratulat, and Jacques Ferber. Un modèle institutionnel pour sma organisationnel. In *Proceedings of Journées Francophones sur les Systèmes Multi-Agents (JFSMA'05)*, Calais - France, Novembre 2005. Hermès-Lavoisier.

[9] José Báez, Tiberiu Stratulat, and Jacques Ferber. A unified model for physical and social environments. In *Proceedings of the third Workshop on Environments for MAS (E4MAS'06@AAMAS06)*, Japan, 2006.

[10] Wolfgang Balzer and Raimo Tuomela. Social institutions, norms and practices. In *Proceedings of the Workshop on Norms and Institutions in Multi-Agent Systems, 4th International Conference on Multi-Agent Systems (Agents-2000)*, Barcelona, Spain, June 3 2000.

[11] D. E. Bell and L. J. La Paluda. Secure computer systems : Unified exposition and multics interpretation. Esd-tr-73-306, The MITRE Corporation, March 1976.

[12] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security*, 6(1) :71–127, 2003.

[13] Guido Boella and Leonardo Lesmo. Deliberate normative agents. In *Proceedings of the Workshop on Norms and Institutions in Multi-Agent Systems, 4th International Conference on Multi-Agent Systems (Agents-2000)*, Barcelona, Spain, June 2000.

[14] S. Brantschen and T. Haas. Agents in a j2ee world, April 2002. AgentLink News #9.

[15] Cosmin Carabelea, Olivier Boissier, and Adina Florea. Autonomy in multi-agent systems : A classification attempt. In Matthias Nickles, Michael Rovatsos, and Gerhard Weiß, editors, *Agents and Computational Autonomy - Potential, Risks, and Solutions - Postproceedings of the 1st International Workshop on Computational Autonomy - Potential, Risks, Solutions (AUTONOMY 2003)*, volume 2969 of *Lecture Notes in Computer Science*, pages 103–113. Springer, 2004. ISBN 3-540-22477-7.

[16] Henrique Lopes Cardoso, Paulo Leitão, and Eugénio Oliveira. An approach to inter-organizational worklow management in an electronic institution. In *Proceedings of InCom 2006, 12th IFAC Symposium on Information Control Problems in Manufacturing*, volume Vol I, pages 429–434, Saint-Etienne - France, May 2006.

[17] Henrique Lopes Cardoso, Andreia Malucelli, Ana Paula Rocha, and Eugénio Oliveira. Institutional services for dynamic organizations. In L.M. Camarinha-Matos, H. Afsarmanesh, and A. Ortiz, editors, *Collaborative Networks and Their Breeding Environments - Sixth IFIP Working Conference on Virtual Enterprises*, pages 521–528, Valencia - Spain, September 2005. Springer.

[18] Henrique Lopes Cardoso and Eugénio Oliveira. Virtual Enterprise Normative Framework within Electronic Institutions. In *Proceedings of 5th Int. Workshop on Engineering Societies in the Agents World (ESAW'04)*, Toulouse - France, October 2004.

[19] Cristiano Castelfranchi, Frank Dignum, Catholijn M. Jonker, and Jan Treur. Deliberative normative agents : Principles and architecture. In Nicholas R. Jennings and Yves Lespérance, editors, *Intelligent Agents VI – Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, Lecture Notes in Artificial Intelligence, pages 364–378. Springer-Verlag, 2000. ISBN 3-540-67200-1.

[20] Rosaria Conte, Cristiano Castelfranchi, and Frank Dignum. Autonomous norm acceptance. In *ATAL '98 : Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*, volume 1555 of *Lecture Notes In Computer Science*, pages 99–112. Springer-Verlag, 1999. ISBN 3-540-65713-4.

[21] Frédéric Cuppens and Alexandre Miège. Adorbac : An administration model for or-bac. *International Journal of Computer Systems Science and Engineering (CSSE)*, 19(3), May 2004.

[22] Keith Decker. *Foundations of Distributed Artificial Intelligence*, chapter Chapter 16 - TAEMS : A Framework for Environment Centered Analysis & Design of Coordination Mechanisms, pages 429–448. Wiley Inter-Science, January 1996.

[23] Keith S. Decker. *Environment centered analysis and design of coordination mechanisms.* PhD thesis, Graduate School of the University of Massachusetts, May 1995.

[24] Keith S. Decker and Victor R. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance, and Management*, 2(4) :215–234, 1993.

[25] Chrysanthos Dellarocas. Contractual agent societies : Negotiated shared context and social control in open multi-agent systems. In *Proceedings of the Workshop on Norms and Institutions in Multi-Agent Systems, 4th International Conference on Multi-Agent Systems (Agents-2000)*, Barcelona, Spain, June 2000.

[26] F. Dignum. Autonomous agents with norms. *Artificial Intelligence and Law*, 7(1) :69–79, 1999.

[27] Frank Dignum. Agents, markets, institutions, and protocols. In Frank Dignum and Carles Sierra, editors, *Agent-Mediated Electronic Commerce - The European AgentLink Perspective*, volume 1991 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2001. ISBN 3-540-41671-4.

[28] Frank Dignum and Carles Sierra, editors. *Agent-Mediated Electronic Commerce - The European AgentLink Perspective*, volume 1991 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2001. ISBN 3-540-41671-4.

[29] V. Dignum, J. Meyer, H. Weigand, and F. Dignum. An organizational-oriented model for agent societies. In *International Workshop on Regulated Agent-Based Social Systems : Theories and Applications (RASTA'02) at AAMAS*, Bologna, Italy, 16 July 2002.

[30] Viginia Dignum. *A Model for Organizational Interaction Based on Agents, Founded in Logic.* PhD thesis, Universiteit Utrecht, 2004. ISBN 90-393-3568-0.

[31] Virginia Dignum, John-Jules Meyer, and Hans Weigand. Towards an organizational model for agent societies using contracts. In C. Castelfranchi and W.L. Johnson, editors, *Proceedings of the first international joint conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, pages 694–695, Bologna, Italy, 2002. ACM Press. ISBN 1-58113-480-0.

[32] Virginia Dignum, Javier Vazquez-Salceda, and Frank Dignum. A model of almost everything : Norms, structure and ontologies in agent organizations. In Nicholas R. Jennings, Carles Sierra,

Liz Sonenberg, and Miling Tambe, editors, *3rd international joint conference on Autonomous Agents & Multi-Agent Systems (AAMAS)*, volume 3, pages 1496–1497, Columbia University, New York City - USA, 19-23 July 2004. ACM Press. ISBN 1-58113-864-4.

[33] Virginia Dignum, Javier Vazquez-Salceda, and Frank Dignum. OMNI : Introducing social structure, norms and ontologies into agent organizations. In Rafael H. Bordini, Mehdi Dastani, Jurgen Dix, and Amal El Fallah-Seghrouchni, editors, *Programming Multi-Agent Systems, Second International Workshop ProMAS 2004, New York, NY, USA, July 20, 2004 Selected Revised and Invited Papers*, volume 3346 of *Lecture Notes in Computer Science*, pages 181–198. Springer, 2005.

[34] Mark d'Inverno, Michael Luck, and Fabiola López y López. *Understanding Agent Systems*, chapter Normative Agents. Springer Series on Agent Technology. Springer-Verlag, second edition edition, 2004.

[35] Marc Esteva. *Electronic Institutions : from Specification to Development*. PhD thesis, Universitat Politecnica de Catalunya, 2003.

[36] Marc Esteva, David de la Cruz, and Carles Sierra. Islander : an electronic institutions editor. In *1rst international joint conference on autonomous agents and multiagent systems (AAMAS'02)*, volume 3, pages 1045–1052, Bologna, Italy, 2002. ACM Press. ISBN 1-58113-480-0.

[37] Marc Esteva, Julian A. Padget, and Carles Sierra. Formalizing a language for institutions and norms. In *Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, volume 2333 of *Lecture Notes in Artificial Intelligence*, pages 348–366. Springer-Verlag, 2002. ISBN 3-540-43858-0.

[38] Marc Esteva, Juan-Antonio Rodriguez-Aguilar, Carles Sierra, Pere Garcia, and Josep L. Arcos. On the formal specification of electronic institutions. In Frank Dignum and Carles Sierra, editors, *Agent-Mediated Electronic Commerce - The European AgentLink Perspective*, volume 1991 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2001. ISBN 3-540-41671-4.

[39] Marc Esteva, Bruno Rosell, Juan A. Rodriguez-Aguilar, and Josep Ll. Arcos. Ameli : An agent-based middleware for electronic institutions. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Miling Tambe, editors, *3rd international joint conference on Autonomous Agents & Multi-Agent Systems (AAMAS)*, volume 1, pages 236–243, Columbia University, New York City - USA, 19-23 July 2004. ACM Press. ISBN 1-58113-864-4.

[40] Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98)*, pages 128–135. IEEE Computer Society Press, 1998.

[41] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations : An organizational view of multi-agent systems. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *AOSE*, pages 214–230, 2003.

[42] Jacques Ferber, Fabien Michel, and José Báez. Agre : Integrating environments with organizations. In *Environments for Multi-agent Systems*, volume vol. 3374 of *Lecture Notes in Computer Science*, pages 48–56. Springer, 2005.

[43] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *Information and System Security*, 4(3) :224–274, 2001.

[44] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and Systems Sciences*, 18 :194–211, 1979.

[45] J. Fox and S. Das. Guardian agents for safety-critical systems. In V. Shankararaman, editor, *Workshop on Autonomous Agents in Health Care*, pages 25–34, 2000.

[46] Andrés García-Camino, Pablo Noriega, and Juan-Antonio Rodríguez-Aguilar. Implementing Norms in Electronic Institutions. In *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 667–673, Utrecht, The Nederlands, July 2005.

[47] Serban I. Gavrila and John F. Barkley. Formal specification for role based access control user/role and role/role relationship management. In *ACM Workshop on Role-Based Access Control*, pages 81–90, 1998.

[48] Andrew Goodchild, Charles Herring, and Zoran Milosevic. Business contracts for b2b. In *CAISE*00 Workshop on Infrastructure for Dynamic Business-to-Business Service Outsourcing*, Stockholm, June 5-6 2000.

[49] B. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86 :269–358, 1996.

[50] Olivier Gutknecht and Jacques Ferber. The MADKIT agent platform architecture. In *Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 48–55, 2000.

[51] Joseph Y. Halpern and Vicky Weissman. Using first-order logic to reason about policies. In *16th IEEE Computer Security Foundations Workshop (CSFW 2003)*, Pacific Grove, California, USA, June 2003.

[52] Henry Hexmoor, Cristiano Castelfranchi, and Rino Falcone, editors. *Agent Autonomy*. Multiagent Systems, Artificial Societies, and Simulated Organizations. Kluwer Academic Publishers, March 2003.

[53] Yigal Hoffner, Simon Field, Paul Grefen, and Heiko Ludwig. Contract-driven creation and operation of virtual enterprises. *Comput. Networks*, 37(2) :111–136, 2001.

[54] Bryan Horling, Victor Lesser, Regis Vincent, Tom Wagner, Anita Raja, Shelley Zhang, Keith Decker, and Alan Garvey. The TAEMS white paper. Technical Report MA 01003, Multi-Agent Systems Lab, Department of Computer Science, University of Massachusetts, Department of Computer Science, University of Massachusetts, 1999.

[55] Jomi Fred Hübner. *Um Modelo de Reorganização de Sistemas Multiagentes*. PhD thesis, Universidade de São Paulo, Escola Politécnica, 2003.

[56] Jomi Fred Hübner and Jaime Simão Sichman. Saci : Uma ferramenta para implementação e monitoração da comunicação entre agentes. In Maria Carolina Monard and Jaime Simao Sichman, editors, *International Joint Conference, 7th Ibero-American Conference on AI, 15th Brazilian Symposium on AI (Open Discussion Track)*, pages 47–56, São Carlos, 2000. ICMC/USP.

[57] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In Guilherme Bittencourt and Geber L. Ramalho, editors, *Proc. of the 16th Brazilian Symposium on Artificial Intelligence (SBIA'02)*, number 2507 in LNAI, pages 118–128. Springer, 2002.

[58] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Spécification structurelle, fonctionnelle et déontique d'organisations dans les systèmes multi-agents. In P. Mathieu and J.P. Müller, editors, *10èmes Journées Francophones Intelligence Artificielle Distribuée & Systèmes Multi-Agents (JFIADSMA'02)*, pages 205–216, Lille, France, 2002. Hermès.

[59] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Using the $\mathcal{M}$OISE$^+$model for a cooperative framework of mas reorganisation. In Ana L. C. Bazzan and Sofiane Labidi, editors, *Proc. of the 17th Brazilian Symposium on Artificial Intelligence (SBIA'04)*, number 3171 in LNAI, pages 506–515. Springer, 2004.

[60] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. $\mathcal{S}-\mathcal{M}$OISE$^+$: A middleware for developing organized multi-agent systems. In Olivier Boissier, Virginia Dignum, Eric Matson, and Jaime Simão Sichman, editors, *Proceedings of the International Workshop on Organizations in Multi-Agent Systems : From Organizations to Organization Oriented Programming in MAS (OOOP)*, volume LNCS 3913 of *Lecture Notes in Computer Science*. Springer, 2005.

[61] Sushil Jajodia, Pierangela Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *IEEE Symposium on Security and Privacy*, pages 31–42, 1997.

[62] Thomas Juan, Adrian Pearce, and Leon Sterling. Roadmap : extending the gaia methodology for complex open systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems (AAMAS'02)*, pages 3–10, New York, NY - USA, 2002. ACM Press.

[63] Anas Abou El Kalam, Rania El Baida, Philippe Balbiani, Salem Benferhat, Frédéric Cuppens, Yves Deswarte, Alexandre Miège, Claire Saurel, and Gilles Trouessin. Or-BAC : un modèle de contrôle d'accès basé sur les organisations. *Cahiers francophones de la recherche en sécurité de l'information*, 1(II) :30–43, 1er trimestre 2003.

[64] Hans Kelsen. *Théorie générale des normes.* Presses Universitaires de France - PUF, 1996. ISBN : 2130474020.

[65] Djamel Khadraoui and Eric Dubois. B2b econtract solution for teleservices. In *International Conference on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC2003)*, pages 12–14, Vienna, Austria, February 2003. ISBN 1740880692.

[66] Marjanca Koetsier, Paul W. P. J. Grefen, and Jochem Vonk. Contracts for cross-organizational workflow management. In *EC-WEB '00 : Proceedings of the First International Conference on Electronic Commerce and Web Technologies*, pages 110–121, London, UK, 2000. Springer-Verlag. ISBN 3-540-67981-2.

[67] Martin J. Kollingbaum and Timothy J. Norman. Supervised interaction : creating a web of trust for contracting agents in electronic environments. In C. Castelfranchi and W.L. Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, pages 272–279, Bologna, Italy, 2002. ACM Press. ISBN 1-58113-480-0.

[68] Butler Lampson. Protection. *ACM Operating System Reviews*, 8(1) :18–24, January 1974.

[69] Heli Laurikkala and Kari Tanskanen. Managing contracts in virtual project supply chains. In *PRO-VE '02 : Proceedings of the IFIP TC5/WG5.5 Third Working Conference on Infrastructures for Virtual Enterprises*, pages 93–100, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V. ISBN 1-4020-7020-9.

[70] H. J. Levesque, P. R. Cohen, and J. Nunes. On acting together. In *Proceedings of the National Conference on Artificial Intelligence*, Menlo Park, Calif., 1990. AAAI press.

[71] Fabiola López y López, Michael Luck, and Mark d'Inverno. Normative agent reasoning in dynamic societies. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Miling Tambe, editors, *3rd international joint conference on Autonomous Agents & Multi-Agent Systems (AAMAS)*, volume 2, pages 732–739, Columbia University, New York City - USA, 19-23 July 2004. ACM Press. ISBN 1-58113-864-4.

[72] OASIS Access Control TC members. eXtensible Access Control Markup Language 3 (XACML) Version 2.0. Technical report, OASIS, February 2005. Tim Moses (Ed.).

[73] Alexandre Miège. *Définition d'un environnement formel d'expression de politiques de sécurité. Modèle Or-BAC et extensions.* Informatique et réseaux, Ecole Nationale Supérieure des Télécommunications, PARIS, 27 June 2005.

[74] Michal Morciniec, Mathias Salle, and Brian Monahan. Towards regulating electronic communities with contracts. In *Proceedings of the Second Workshop on Norms and Institutions in MAS, 5th International Conference on Autonomous Agents*, Montreal, Canada, May 2001.

[75] Allen Newell. *Unified theories of cognition.* Harvard University Press, Cambridge, MA, USA, 1990. ISBN : 0-674-92099-6.

[76] Douglass C. North. *Institutions, Institutional Change and Economic Performance.* Political Economy of Institutions and Decisions. Cambridge University Press, 1990. ISBN 0521397340.

[77] Douglass C. North. Institutions. *Journal of Economic Perspectives*, 5(1) :97–112, Winter 1991.

[78] Christoph Oechslein, Franziska Klügl, Rainer Herrler, and Frank Puppe. Uml for behavior-oriented multi-agent simulations. In Babara Dunin-Keplicz and Edward Nawarecki, editors, *Proceedings of the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems (CEEMAS 2001)*, 2001.

[79] OFTA. *Systèmes Multi-Agents (ARAGO 29).* Lavoisier, February 2004. Rapport de synthèse du Groupe "Systèmes Multi-Agents" de l'Observatoire Français des Techniques Avancées coordonné par Yves Demazeau.

[80] Boris Padovan, Stefan Sackmann, Torsten Eymann, and Ingo Pippow. A prototype for an agent-based secure electronic marketplace including reputation tracking mechanisms. *International Journal of Electronic Commerce*, 6(4) :93–113, 2002.

[81] Sébastien Poggi. Mise en place d'une solution PKI. DESS SSIC, Université de Metz – UFR MIM, 2004.

[82] David V. Pynadath and Milind Tambe. An automated teamwork infrastructure for heterogeneous software agents and humans. *Autonomous Agents and Multi-Agent Systems*, 7(1-2) :71–100, 2003. ISSN : 1387-2532.

[83] David V. Pynadath, Milind Tambe, and Nicolas Chauvat. Rapid integration and coordination of heterogeneous, distributed agents for collaborative enterprises. In *Proceedings of the DARPA-JFACC Symposium on Advances in Enterprise Control*, pages 171–176, 1999.

[84] David V. Pynadath, Milind Tambe, Nicolas Chauvat, and Lawrence Cavedon. Toward team-oriented programming. In *Agent Theories, Architectures, and Languages*, pages 233–247, 1999.

[85] Simon Renault, Benjamin Gâteau, Damien Mathevon, and Yannick Naudet. Implementation of RAMO-based multimedia applications on MPEG-4 platform. In *IEEE International Conference on Consumer Electronics (ICCE'06)*, Las Vegas, Nevada - USA, January 9-11 2006. ISBN : 0-7803-9459-3/06.

[86] Simon Renault, Farid Meinkohn, Djamel Khadraoui, and Patrick Blandin. Reactive and adaptive multimedia object approach for interactive and immersive applications. In *Proceedings of the International Conference on Information & Communication Technologies : From Theory to Applications*, Damascus, Syria, April 19-23 2004.

[87] Ana Paula Rocha, Henrique Lopes Cardoso, and Eugénio Oliveira. *Virtual Enterprise Integration : Technological and Organizational Perspectives*, chapter Ch. XI - Contributions to an Electronic Institution supporting Virtual Enterprises' life cycle, pages 229–246. Idea Group Inc., 2005. ISBN 1-59140-406-1.

[88] Mathias Salle. Electronic contract framework for contractual agents. In *Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, pages 349–353. Springer-Verlag, 2002. ISBN 3-540-43724-X.

[89] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2) :38–47, 1996.

[90] Carles Sierra, Juan A. Rodriguez-Aguilar, Pablo Noriega Blanco-Vigil, Josep-Lluis Arcos-Rosell, and Marc Esteva-Vivancos. Engineering multi-agent systems as electronic institutions. *UPGRADE - The European Journal for the Informatics Professional*, V(4) :33–39, August 2004. ISSN 1684-5285.

[91] Munindar P. Singh. An ontology for commitments in multiagent systems : Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7 :97–113, 1999.

[92] Reid G. Smith. The contract net protocol : High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12) :1104–1113, December 1980.

[93] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7 :83–124, 1997.

[94] Milind Tambe, David V. Pynadath, Nicolas Chauvat, Abhimanyu Das, and Gal A. Kaminka. Adaptive agent integration architectures for heterogeneous team members. In *Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 301–308, 2000.

[95] Milind Tambe and Weixiong Zhang. Towards flexible teamwork in persistent teams. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS)*, 1998.

[96] Milind Tambe and Weixiong Zhang. Towards flexible teamwork in persistent teams : Extended report. *Autonomous Agents and Multi-Agent Systems*, 3(2) :159–183, 2000.

[97] Raimo Tuomela. *The Importance of Us : A Philosophical Study of Basic Social Norms.* Standford University Press, 1995. ISBN : 0804724229.

[98] Roy Turner. Context-mediated behavior for intelligent agents. *International Journal of Human-Computer Studies*, 48(3) :307–330, March 1998.

[99] A. Valente. *Legal Knowledge Engineering - A Modelling Approach*, volume 30 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 1995. ISBN 90-5199-230-0.

[100] J. Vázquez-Salceda, U. Cortes, J. Padget, A. Lopez-Navidad, and F. Caballero. The organ allocation process : a natural extension of the carrel agent mediated electronic institution. *AI Communications*, 16(3) :153–165, 2003.

[101] J. Vázquez-Salceda, J.A. Padget, U. Cortes, A. Lopez-Navidad, and F. Caballero. Formalizing an electronic institution for the distribution of human tissues. *Artificial Intelligence in Medicine*, 27(3) :233–258, March 2003.

[102] Javier Vázquez-Salceda. *The role of Norms and Electronic Institutions in Multi-Agent Systems*. Whitestein Series in Software Agent Technology. Springer-Verlag, softcover edition, 2004. ISBN 3-7643-7057-2.

[103] Javier Vázquez-Salceda and Frank Dignum. Modelling electronic organizations. In *Multi-Agent Systems and Applications III : 3rd. International/Central and Eastern European Conference on Multi-Agent Systems -CEEMAS'03-*, volume 2691 of *Lecture Notes in Artificial Intelligence*, pages 584–593. Springer-Verlag, 2003.

[104] Javier Vázquez-Salceda, Virginia Dignum, and Frank Dignum. Organizing multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 11(3) :307–360, November 2005. ISSN : 1387-2532.

[105] Oliver Vickers and Julian Padget. Skeletal jade components for the construction of institutions. In Julian Padget, Onn Shehory, David Parkes, Norman Sadeh, and William E. Walsh, editors, *Agent-Mediated Electronic Commerce IV - Designing Mechanisms and Systems (AMEC 2002 Workshop Revised Papers)*, volume 2531 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2002. ISBN 3-540-00327-4.

[106] Peter Jason Willemsen. *Behavior and scenario modeling for real-time virtual environments*. Computer science, Graduate College of The university of Iowa, 2000. Supervisor-Joseph K. Kearney.

[107] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 69–76, Seattle, WA, USA, 1999. ACM Press.

[108] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3) :285–312, 2000.

[109] Franco Zambonelli, Nicholas R. Jennings, Andrea Omicini, and Michael Wooldridge. *Agent-Oriented Software Engineering for Internet Applications*, chapter Chapter 13, pages 326–346. Springer, 2001. ISBN : 3-540-41613-7.

[110] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems : The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3) :317–370, July 2003. ISSN :1049-331X.

[111] Z. Zlatev. Examination of the negotiation domain. Technical report, EEMCS department, University of Twente, September 2002.

Ecole Nationale Supérieure des Mines
de Saint-Etienne

N° d'ordre : 442 I

Benjamin Gâteau

Title : Multi-Agent based Institutions Modelling and Supervision

Speciality : Computer Science

Keywords : Multi-Agent System, Electronic Institution, Arbitration System, Norm, Organisation

Abstract : The scientific domain of our research work is Multi-Agent System. In this domain, we are interested in the development of global models defining and structuring the common activity of autonomous agents. The autonomy of an agent results in its capacity to determine its own goals vis-a-vis the configuration of the environment and its motivations. Because of this autonomy, conflicts related to the existence of goals suitable for each autonomous agent of a society can be incompatible, complementary or in competition with the society objectives. Models of conflicts management exist and within the framework of this work, we focused on the organisational models.

We assert that the definition and implementation of global constraints influencing autonomous agents functioning must be done so as to take into account the possibility that the agents can not respect constraints according to individual contexts and objectives' of each one. In order to achieve these goals, we propose an Electronic Institution model composed of a constraints' description and specification model as well as a system of supervision and reinforcement of these constraints.

This model is used to define and structure the activity of a set of autonomous agents within a normed organisation thanks to $\mathcal{M}$OISE^{Inst}and to supervise the respect of the organisation specification with $\mathcal{S}$YNAI. This arbitration system is composed of a set of supervisor agents able to implement constraints' regulation and reinforcement mechanisms on the autonomous agents.

The model experimentation is done in two completely different domains. They concern the contents creation and execution for interactive television on a side and the management of electronic contracts in a e-commerce platform on the other side. They have the same arbitration of autonomous entities having to respect constraints issue. Moreover, both applications are complementary by their Institution scope.

Ecole Nationale Supérieure des Mines
de Saint-Etienne

N° d'ordre : 442 I

Benjamin Gâteau

Titre : Modélisation et Supervision d'Institutions Multi-Agents

Spécialité : Informatique

Mots clefs : Système Multi-Agent, Institution Electronique, Système d'Arbitrage, Norme, Organisation

Résumé : Le domaine scientifique dans lequel nos recherches s'insèrent est celui des Systèmes Multi-Agents. Dans ce domaine, nous nous intéressons à l'élaboration de modèles globaux définissant et structurant l'activité commune d'agents autonomes. En effet l'autonomie d'un agent se traduit par sa capacité à déterminer ses propres buts face à la configuration de l'environnement ainsi qu'à ses motivations. Du fait de cette autonomie, des conflits liés à l'existence de buts propres à chacun des agents autonomes d'une société d'agents peuvent être incompatibles, complémentaires ou en compétition avec les objectifs de la société. Des modèles de gestion de conflits existent, et dans le cadre de ce travail, nous nous sommes intéressés aux modèles organisationnels.

Nous défendons la thèse selon laquelle la définition et la mise en place de contraintes globales influençant le fonctionnement d'agents autonomes doit être faite de manière à prendre en compte l'éventualité que les agents puissent ne pas respecter les contraintes selon les contextes et les objectifs individuels de chacun. Afin d'atteindre ces objectifs, nous proposons un modèle d'Institution Électronique comportant une description explicite et déclarative des contraintes à appliquer aux agents ainsi qu'un système de supervision et de renforcement de ces contraintes.

Ce modèle permet de définir et de structurer grâce à $\mathcal{M}$OISE^{Inst} l'activité d'un ensemble d'agents autonomes au sein d'une organisation normée et de superviser son respect grâce au système d'arbitrage $\mathcal{S}$YNAI. Ce dernier est composé d'un ensemble d'agents de supervision capables de mettre en place des mécanismes de régulation et de renforcement de ces contraintes sur les agents autonomes.

L'application du modèle se fait dans deux domaines totalement différents puisqu'ils concernent la création et l'exécution de contenu pour la télévision interactive d'un côté et la gestion de contrats électroniques d'une plate-forme de e-commerce de l'autre. Ils exhibent cependant la même problématique d'arbitrage d'entités autonomes devant respecter des contraintes. De plus, ces deux applications sont complémentaires par leur différence de portée de l'Institution.