
**Méta-modélisation UML pour les environnements
virtuels procéduraux et collaboratifs**

Rapport de stage

Nicolas MARION — (ENIB)

— Juin 2006 —



*Rapport de stage de Master 2 Recherche Informatique de
l'École Nationale d'Ingénieurs de Brest, cohabilité par l'IFSIC,*

effectué par Nicolas MARION,

encadré par Ronan QUERREC

au Centre Européen de Réalité Virtuelle,

EA-3883 – LISYC

Laboratoire d'Informatique des SYstèmes Complexes

de février à juin 2006.

Table des matières

Remerciements	iii
Introduction	1
1 Contexte du stage	3
1.1 Étude bibliographique	3
1.2 Présentation de l’environnement existant	4
2 Interprétation d’UML pour la modélisation d’EV	7
2.1 Différents diagrammes UML	7
2.1.1 Collaboration	8
2.1.2 Les interactions	9
2.1.3 Diagrammes d’activité	10
2.2 UML et MASCARET	12
2.2.1 Le paquet Organisations	13
2.2.2 Le paquet Environnement	14
2.2.3 Le paquet Agent	14
2.3 Représentation des procédures par des diagrammes d’activités	15
2.3.1 Représentation en UML	15
2.3.2 Algorithme de suivi de procédure	18
2.4 Conclusion et perspectives	20
3 Organisation des agents	23
3.1 Différents modèles d’organisation existants	23
3.1.1 AGR	23
3.1.2 Moise+	24
3.1.3 MASCARET	26
3.1.4 OMNI	27
3.2 Problèmes rencontrés avec l’organisation existante	29
Conclusion	33
Références	35

Table des figures

1	Une procédure collaborative sous GASPAR.	2
1.1	Du modèle UML à l'environnement virtuel.	5
1.2	L'architecture de MASCARET.	6
2.1	Une collaboration en UML	8
2.2	Un exemple de diagramme de séquence avec UML2.0	9
2.3	Un exemple de diagramme de communication avec UML2.0	10
2.4	Un diagramme d'activité en UML	11
2.5	Un diagramme d'activité en UML, organisé en niveau pour distinguer les rôles.	11
2.6	Le modèle de GASPAR.	12
2.7	La collaboration représentant l'équipe de déplacement sur le porte-avions.	13
2.8	Le paquet Environnement de GASPAR.	14
2.9	La machine à états de Plaque de protection	14
2.10	Le paquet Agent de GASPAR.	15
2.11	Les classes du paquet Agent	15
2.12	Le modèle des activités implémenté dans VEHA.	16
2.13	Une procédure de GASPAR représentée par un diagramme d'activité.	17
2.14	Un exemple d'action décomposée en opérations et intégrée dans une procédure.	21
3.1	Le modèle d'organisation AGR	24
3.2	Un exemple de hiérarchie de buts avec Moise+	25
3.3	Le modèle générique de MASCARET	26
3.4	Le modèle complet de l'environnement social de MASCARET	27
3.5	Configuration du porte-avions pour une procédure de catapultage sous GASPAR.	29

Remerciements

Je souhaite tout d'abord remercier les personnes qui m'ont permis de suivre la formation de Master 2 Recherche Informatique de l'IFSIC. En particulier M. François BODIN, responsable du Master et M. Djamil SARNI, responsable de la filière brestoise du Master.

Je souhaite également à remercier M. Jacques TISSEAU, directeur du Centre Européen de Réalité Virtuelle (CERV) pour nous avoir accueilli dans son laboratoire et M. Pierre DE LOOR qui s'est très bien occupé de nous durant cette année de Master.

Je tiens aussi à remercier mon maître de stage M. Ronan QUERREC pour sa disponibilité et son aide précieuse, M. Alexandre BOUDINOT pour sa patience et sa bonne humeur malgré le surcroît de travail que je lui ai apporté ainsi que Mme Karen MARTINAUD et Mme Malory SENEQUIER pour avoir accepté de relire mon étude bibliographique.

Enfin, je remercie Fabrice HARROUËT pour les outils qu'il a mis à notre disposition, les étudiants de Master pour la bonne humeur qu'ils ont fait régner dans notre bureau et Tim pour avoir réussi à nous divertir à chaque pause déjeuner.

Introduction

De nombreux domaines de formation, tels que la conduite automobile ou la formation des pompiers professionnels, nécessitent la mise en situation des apprenants. Ceux-ci doivent acquérir non seulement les connaissances nécessaires, mais encore de véritables compétences. Pour devenir progressivement efficace en situation, l'apprenant doit apprendre par l'action. Or, dans de nombreuses situations, la mise en situation des apprenants, peut s'avérer coûteuse (d'un point de vue matériel) ou risquée (d'un point de vue humain). C'est le cas lorsqu'il s'agit d'apprendre à réagir face à des accidents, des événements peu prévisibles ou des dysfonctionnements. En effet, la reproduction d'une telle situation est non seulement coûteuse et risquée, elle peut aussi être impossible à recréer au moment souhaité (conditions climatiques par exemple).

D'un autre côté, la simulation informatique permet d'immerger les apprenants dans des environnements dans lesquels ils peuvent essayer, choisir, prendre des initiatives, échouer et recommencer. Les environnements informatiques de formation utilisant la réalité virtuelle sont particulièrement adaptés à ce problème, car ils permettent de recréer des situations réalistes, complexes, en maîtrisant parfaitement les conditions de la simulation. L'ensemble de ces environnements est souvent regroupé sous l'acronyme EVF¹ (Environnement Virtuel de Formation).

J'ai effectué mon stage au CERV (Centre Européen de Réalité Virtuelle), et plus précisément au sein de l'équipe SPI (Simulation Participative et Immersive). Cette équipe développe des environnements virtuels destinés à la formation, dont une grande partie porte sur l'apprentissage de tâches procédurales et collaboratives, qui sont sujets de mon étude.

Durant mon stage je me suis appuyé sur l'application GASPARG (Gestion d'Aviation Sur Porte-Avions par la Réalité virtuelle). Le but de cette application est de simuler en temps réel et de valider les flux aviations sur les futurs navires français et cela avant leur construction. L'optimisation de ces flux dépend bien sûr de la configuration du navire, mais également des procédures réalisées par le personnel. La Figure 1 montre un exemple de procédure collaborative sous GASPARG.

Cette application met en évidence l'importance du travail collaboratif pour l'efficacité du système, et donc l'importance de la formation du personnel au travail collaboratif et procédural, qui est un objectif à plus long terme de GASPARG.

Un des principaux avantages des EVF est qu'il est inutile de réunir tous les par-

¹En anglais on parle de VET (Virtual Environment for Training)



FIG. 1 – Une procédure collaborative sous GASPAR.

ticipants humains nécessaires pour recréer la situation (chose qui peut se révéler très difficile). L'application doit donc simuler en temps réel et de manière réaliste l'environnement physique de l'apprenant ainsi que son environnement social. L'environnement social (les autres personnels de l'équipe) qui ne participe pas à la simulation est alors joué par des agents autonomes.

Dans le premier chapitre de ce rapport, je précise le contexte du stage et présente l'environnement sur lequel je me suis appuyé durant le stage.

Dans le deuxième chapitre, je détaille l'interprétation de la sémantique d'UML que nous avons réalisé pour pouvoir modéliser des environnements virtuels destinés à des tâches procédurales et collaboratives.

Dans le troisième chapitre, je m'intéresse à la manière dont on doit organiser les agents afin de leur permettre de réaliser des tâches procédurales et collaboratives à travers l'étude de plusieurs modèles d'organisations multi-agents.

Enfin, en conclusion, je résume le travail que j'ai effectué durant mon stage et présente en perspective les travaux qu'il reste à réaliser sur le projet.

Chapitre 1

Contexte du stage

1.1 Étude bibliographique

Dans le cadre du Master Recherche, j'ai effectué une étude bibliographique qui porte sur l'étude d'organisations sociales multi-agents pour les environnements virtuels de formation. En effet, les EVF qui sont développés au CERV sont fondés sur des systèmes multi-agents (SMA). Dans ce genre de SMA, l'environnement est scindé en deux parties distinctes en interaction : l'environnement physique et l'environnement social. La présence de ces deux environnements faisant intervenir différents types d'agents (plus ou moins réactifs ou rationnels en fonction de l'environnement dans lequel ils agissent), fait du système un SMA hétérogène, dans lequel il devient impératif de structurer les interactions à l'aide d'une organisation. Mon étude bibliographique porte plus précisément sur la modélisation d'organisation pour l'environnement social, destiné à la gestion des activités humaines.

Durant cette étude, j'ai constaté que le fait d'appliquer un modèle d'organisation à un EVF entraîne l'apparition de plusieurs contraintes liées d'une part à l'aspect environnement virtuel (et donc à la réalité virtuelle), et d'autre part à l'aspect environnement de formation. Nous pouvons distinguer quatre types de contraintes :

Simulation : Le modèle d'organisation sociale étant destiné à la gestion des activités humaines, les agents évoluant dans l'environnement doivent adopter un comportement réaliste, afin que l'apprenant se sente immergé dans l'environnement.

Conception : Le modèle doit pouvoir permettre au concepteur de l'environnement (le formateur par exemple) d'écrire et de modifier les règles de l'organisation dynamiquement, sans modifier le code du programme afin de pouvoir activer ou désactiver certains aspects de la simulation et ainsi s'adapter à un apprenant plus ou moins novice.

Réification : Le modèle doit être capable de répondre aux questions de l'apprenant sur l'organisation sociale du système. Si par exemple l'apprenant demande quels sont les rôles qui constituent son équipe, le modèle doit utiliser sa connaissance de l'organisation pour lui répondre. La structure du système doit donc être réifiée.

Participation : Dans l'environnement virtuel de formation, l'utilisateur du système (l'apprenant) doit pouvoir agir sur le système comme un agent à part entière. La

contrainte vient du fait que l'on peut lui faire prendre le contrôle d'un agent de manière dynamique et même demander à cet agent d'assister l'apprenant dans sa tâche ou de le remplacer. Cette affectation dynamique des rôles pose de nouvelles contraintes au niveau de l'organisation.

Dans une deuxième partie de mon étude bibliographique, j'ai montré que les notions de collaboration entre rôles ainsi que les agencements d'actions, qui font partie intégrante d'un environnement pour la formation à des tâches procédurales et collaboratives, sont des sujets qui ont déjà été étudiés dans le domaine du génie logiciel, et plus particulièrement dans la norme UML¹ (norme la plus utilisée dans ce domaine).

De plus, UML prend une part importante dans le développement des environnements virtuels du CERV. Tout d'abord, la quasi-totalité de l'environnement virtuel est décrite en UML, comme c'est le cas de beaucoup d'applications de nos jours. Cependant, la représentation de l'environnement en UML ne sert pas uniquement à sa description. En effet, le modèle UML créé est traité de manière automatique et sert directement à la création de l'environnement virtuel. Ce procédé, décrit plus en détail dans la section suivante nécessite une interprétation de la sémantique d'UML afin d'adapter la norme UML à la description d'environnements virtuels.

Mon stage s'est donc articulé autour de deux points essentiels :

- ▷ Étudier la norme UML et en interpréter la sémantique dans le but de pouvoir modéliser les notions de procédure et de collaboration dans les environnements virtuels en UML.
- ▷ Réfléchir sur un modèle multi-agents intégrant les notions de collaboration, de procédure, d'équipe... tout en gardant en tête les quatre contraintes identifiées précédemment.

1.2 Présentation de l'environnement existant

GASPAR est un des environnements virtuels qui s'appuie sur le projet MASCARET du CERV. Le projet MASCARET (Multi-Agent System for Collaborative and Adaptive Realistic Environment for Training) est dérivé des travaux de thèse de Ronan QUERREC [Querrec, 2002]. Comme son nom l'indique, le but de ce projet est de développer des EVF réalistes, axés sur le travail collaboratif et basés sur des systèmes multi-agents.

Avant de présenter l'architecture et le fonctionnement de MASCARET, nous allons d'abord voir à quoi il sert.

Comme nous l'avons expliqué dans la section précédente, pour créer un environnement virtuel, il doit être modélisé entièrement en UML. Ensuite, ce modèle est traité automatiquement dans le but de créer l'environnement. C'est à ce niveau qu'intervient MASCARET. MASCARET prend en entrée le modèle UML, traite tous les éléments qui le composent, puis instancie les objets nécessaires à la création de l'environnement

¹Unified Modeling Language : Superstructure version 2.0, Object Management Group (<http://www.uml.org>)

virtuel. Ce procédé est illustré sur la Figure 1.1.

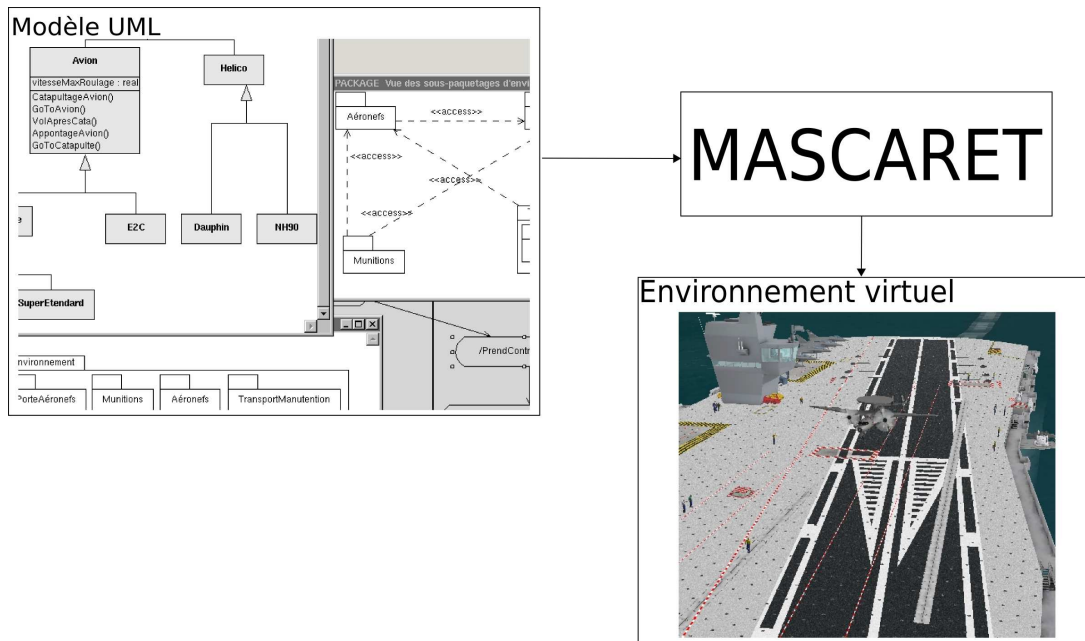


FIG. 1.1 – Du modèle UML à l’environnement virtuel.

Voyons maintenant l’architecture interne du projet MASCARET. Le projet MASCARET est organisé en trois couches, comme le montre la Figure 1.2 :

La couche bas-niveau : C’est ici que se trouve la plate-forme multi-agents utilisée pour la simulation de l’environnement. Au CERV, la plate forme utilisée est **ARéVi** (Atelier de Réalité Virtuelle) développée par Fabrice Harrouët². Une autre plate-forme multi-agents pourrait être utilisée pour simuler l’environnement.

La couche intermédiaire : À ce niveau, nous trouvons trois paquets :

- ▷ **VEHA** (Virtual Environment supporting Human Activities) est un profil UML qui permet de représenter (et donc de réifier) dans l’environnement tous les composants du modèle UML créé (classes, attributs, opérations...).
- ▷ Le paquet **Agent** est un paquet qui représente les notions d’agent, de rôle, d’organisation, de comportement... Ce paquet correspond à l’ancien modèle d’organisation de la thèse de Ronan Querrec.
- ▷ Le paquet **VirtualHuman** permet de représenter des humains virtuels et leurs activités (animations).

La couche haut-niveau : La couche haut niveau permet d’enrichir l’environnement représenté à la couche intermédiaire. C’est à ce niveau qu’on trouve le paquet **Procedural**, qui permet aux agents du système de suivre des procédures.

²<http://www.enib.fr/~harrouet/>

Sur la Figure 1.2, nous avons également représenté un paquet ITS utilisant MASCARET. Ce paquet s'appuie sur les connaissances situées à la couche haut-niveau et la couche intermédiaire pour effectuer l'apprentissage.

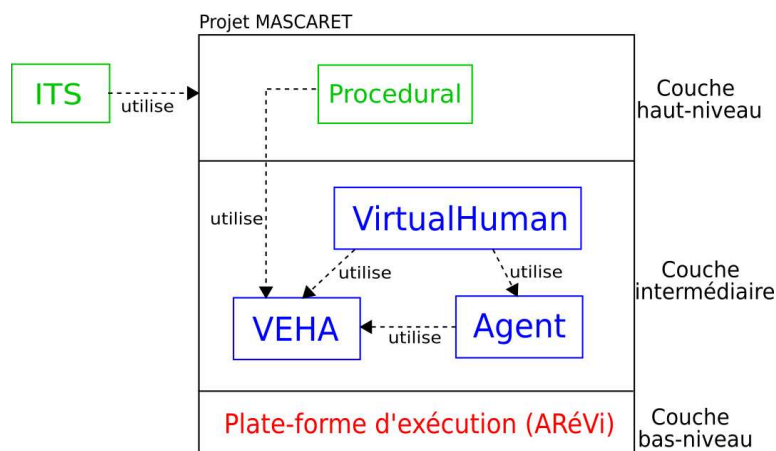


FIG. 1.2 – L'architecture de MASCARET.

Mon travail a porté tout particulièrement sur le paquet **Procedural**. Ce paquet permet d'introduire dans l'environnement les notions liées aux procédures. Ce genre de notion n'existant pas en UML, nous voyons ici la nécessité de l'interprétation sémantique d'UML pour la modélisation de l'environnement virtuel. C'est ce qui est décrit dans le chapitre suivant.

Chapitre 2

Interprétation d'UML pour la modélisation d'EV

Comme nous l'avons vu dans le chapitre précédent, dans MASCARET, un environnement virtuel est totalement décrit à travers un modèle UML. Cela présente des avantages au niveau de la conception de l'environnement puisqu'il est alors inutile de créer ou modifier du code pour modifier cet environnement, il suffit de changer le modèle UML à l'aide de son modèleur favori.

UML est un langage de modélisation permettant de modéliser non seulement la structure d'une application mais également son comportement. Le rapprochement entre UML et les systèmes multi-agents a déjà été exploité ; nous pouvons par exemple citer le projet AUML¹, qui a pour but d'étendre UML afin de pouvoir décrire des comportements d'agents. Certaines idées de ce projet ont d'ailleurs été intégrées dans la norme UML 2.0. Cependant, la norme UML n'intégrant pas directement les notions de procédure, d'équipe et même d'agent, il devient alors nécessaire d'interpréter la sémantique d'UML pour l'adapter au cas qui nous intéresse : la conception d'un EVF pour l'apprentissage de tâches procédurales et collaboratives. Cela comprend la description d'équipes, de procédures, d'objets, de ressources, de types d'agents...

Ce chapitre est organisé en quatre sections. La première fait un tour d'horizon des différents diagrammes UML qui peuvent être utilisés pour représenter ces différentes notions (cette partie est extraite de mon étude bibliographique). Dans la deuxième partie, je présente les travaux que j'ai effectué sur la modélisation d'un environnement en UML dans MASCARET, et je détaille le travail sur les procédures dans la troisième partie. Enfin, la quatrième partie conclut et offre des perspectives et des améliorations qui pourraient être effectuées sur le modèle UML utilisé par MASCARET.

2.1 Différents diagrammes UML

L'objectif de cette partie est d'étudier quelques diagrammes UML et de voir comment nous pouvons les interpréter dans le cas qui nous intéresse. Nous étudions d'abord comment les notions d'organisation et de rôle peuvent être traitées par les collabora-

¹Agent UML (<http://www.auml.org>)

tions en UML. Ensuite, nous voyons que les interactions peuvent être représentées par des diagrammes de séquence ou de communication. Enfin, nous examinons les activités en UML, et les rapprochons avec la notion de procédure dans un environnement virtuel.

2.1.1 Collaboration

En UML, une collaboration décrit une structure d'éléments en collaboration (rôles), chacun réalisant une fonction spécialisée, et qui accomplit collectivement une fonctionnalité désirée. Les collaborations sont généralement utilisées pour expliquer comment un ensemble d'instances en coopération réalise une ou plusieurs tâches. Typiquement, une collaboration décrit les aspects d'un objet uniquement dans le contexte de la collaboration. Par exemple, un objet peut jouer plusieurs rôles simultanément dans plusieurs collaborations, mais chaque collaboration ne représentera que les aspects de cet objet qui sont pertinents dans ce contexte.

Une collaboration spécifie quelles propriétés les instances doivent posséder pour pouvoir participer à la collaboration. Un rôle peut être défini par une ou plusieurs interfaces. Une interface est la description d'un ensemble de propriétés (caractéristiques externes observables) requises ou fournies par une instance. Un objet peut jouer un rôle défini par une interface à partir du moment où ce classifieur réalise l'interface (il a toutes les propriétés requises). La Figure 2.1 représente la notation d'une collaboration en UML.

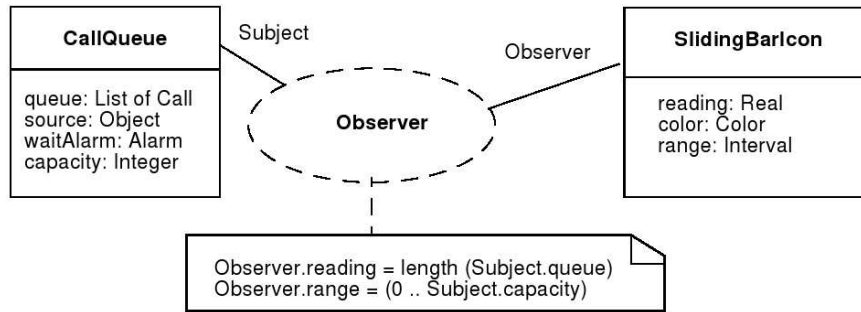


FIG. 2.1 – Une collaboration en UML

Cette figure décrit une collaboration (*Observer*) qui fait participer deux objets (*CallQueue* et *SlidingBarIcon*). Ces objets interviennent dans la collaboration en jouant un rôle (*Subject* pour *CallQueue* et *Observer* pour *SlidingBarIcon*). Chacun des objets jouant un rôle doit présenter des capacités qui sont représentées par une liste d'attributs (*reading*, *color* et *range* pour *SlidingBarIcon* par exemple). De même, plusieurs contraintes sur ces capacités peuvent être fixées (ici par exemple, il faut que l'attribut *range* de l'objet qui joue le rôle *Observer* soit compris entre 0 et la valeur de l'attribut *capacity* de l'objet qui joue le rôle *Subject*).

Ces notions de rôle et de collaboration sont utilisées également dans le domaine des SMA. En effet, nous pouvons utiliser cette représentation pour représenter des agents jouant différents rôles au sein d'une équipe ainsi que les contraintes qui régissent l'affectation de ces rôles aux agents.

2.1.2 Les interactions

Les interactions entre les agents sont le concept primordial des SMA. Cette notion d'interaction est également définie en UML. Pour UML, une interaction est une unité de comportement qui se concentre sur les échanges observables d'informations entre des éléments. Les messages échangés sont souvent des demandes d'exécution d'opérations.

Il existe différents diagrammes UML permettant de représenter des interactions. Ici, nous étudions les diagrammes de séquence et les diagrammes de communication.

Diagrammes de séquence

Le diagramme de séquence est le plus commun pour représenter des interactions. Il se concentre sur la séquence de messages qui sont échangés entre plusieurs objets (représentés par des lignes de vie). Le diagramme de séquence peut décrire les interactions entre des classes (diagramme générique), ou peut être spécialisé pour décrire les interactions entre des objets (instances). La figure 2.2 représente un diagramme de séquence en UML 2.0.

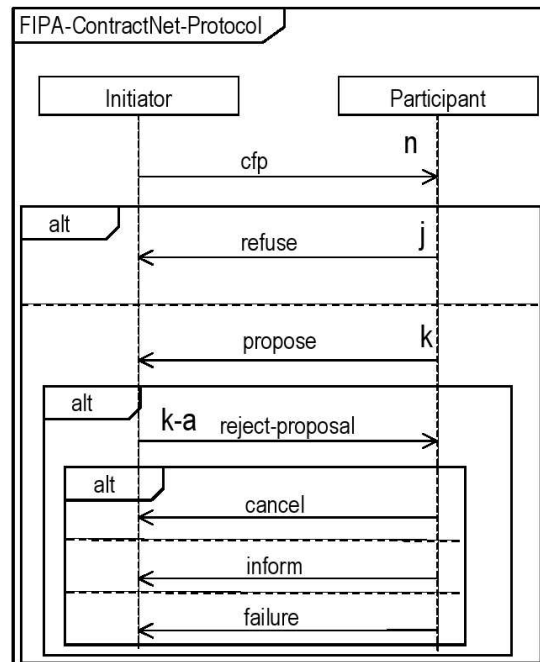


FIG. 2.2 – Un exemple de diagramme de séquence avec UML2.0

Ce diagramme représente la séquence de messages échangés entre deux classes (*Initiator* et *Participant*), prenant part au protocole ContractNet de la FIPA². Les lignes verticales sortant des classes représentent leur ligne de vie. Les flèches entre les lignes de vie décrivent les échanges de messages. Nous voyons par exemple que le rôle Initiator envoie le message *cfp* à *n* participants et qu'il reçoit le message *refuse* de *j* participants

²The Foundation for Intelligent Physical Agents (<http://www.fipa.org>)

et le message *propose* de k participants.

Un diagramme de séquence peut être utilisé pour décrire des interactions entre les agents. Pour cela, nous pouvons considérer que chaque ligne de vie représente l'activité d'un rôle ou d'un agent jouant ce rôle (si le diagramme est générique ou non). Cette notation permet de contrôler des interactions par envois de messages et donc de représenter par exemple des protocoles d'interaction entre agents.

Diagrammes de communication

Le diagramme de communication (appelé diagramme de collaboration dans UML 1.x) décrit les interactions entre des lignes de vie en se concentrant sur l'architecture de la structure interne et comment elle correspond par envoi de messages.

Le diagramme de communication correspond à un diagramme de séquence simple qui n'utilise pas de mécanisme de structuration. Le séquençage des messages se fait via un schéma de numérotation. Contrairement au diagramme de séquence, ce diagramme suppose qu'il existe un ordonnancement strict des messages. La Figure 2.3 représente un diagramme de communication décrivant une interaction entre plusieurs objets. Ce diagramme montre les envois de messages (expéditeur, destinataire), ainsi que la manière dont ils sont séquençés.

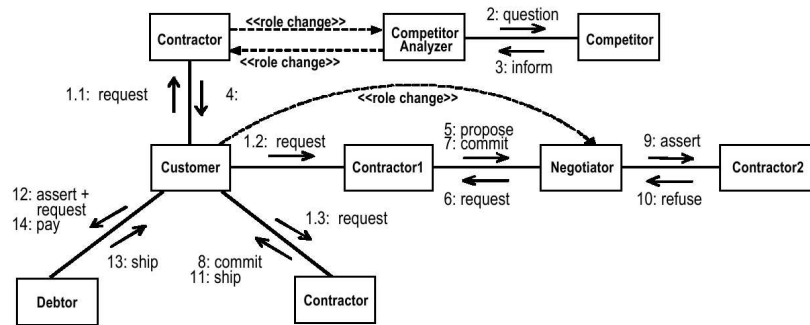


FIG. 2.3 – Un exemple de diagramme de communication avec UML2.0

Ce type de diagramme peut être appliqué à la modélisation de SMA de la même manière que le diagramme de séquence. Nous pouvons cependant remarquer qu'on ne peut pas modéliser d'interaction entre des groupes dans ces deux diagrammes, la notion de groupe n'étant pas spécifiée en UML [Bauer et Odell, 2004].

2.1.3 Diagrammes d'activité

Une activité spécifie comment des comportements sont coordonnés en utilisant des modèles de flot. L'objectif d'un diagramme d'activité est donc de représenter l'ordre et les conditions d'exécution de comportements de plus bas niveau. Les actions coordonnées par ce genre de diagramme peuvent démarrer pour plusieurs raisons : l'exécution d'une autre action est terminée, des données ou des objets deviennent disponibles ou parce qu'un évènement externe vient interrompre le flot d'exécution. La Figure

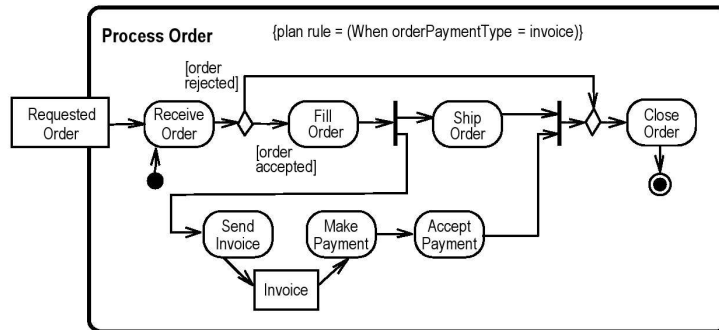


FIG. 2.4 – Un diagramme d’activité en UML

2.4 représente un diagramme d’activité simple décrivant l’exécution d’un processus de commande.

Ce genre de diagramme peut être utilisé pour la conception de SMA, car il permet de décrire comment un ou plusieurs agent atteignent un but en suivant un plan. UML 2.0 permet même d’indiquer quel rôle jouera chaque partie du plan en organisant le diagramme d’activité sur plusieurs niveaux (swimlanes). Nous pouvons ainsi distinguer plus aisément de quel rôle dépend chaque partie du diagramme d’activité. La Figure 2.5 représente le même diagramme d’activité que précédemment, organisé ainsi.

Un diagramme d’activité peut donc nous permettre de représenter comment des agents atteignent un but. Cependant, la notion de but en elle même n’existe pas dans UML. Il y a donc deux manières de penser un but dans un diagramme d’activité. Soit on considère que le noeud final du diagramme d’activité est le but, car c’est le point final du processus. Soit on se place à un niveau supérieur et considère le but à travers la notion de service rendu par le système [Bauer et Odell, 2004]. Dans MASCARET, ce diagramme est utilisé pour décrire des procédures.

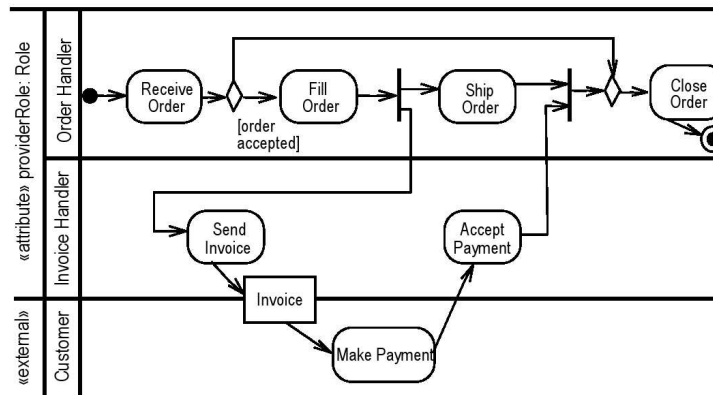


FIG. 2.5 – Un diagramme d’activité en UML, organisé en niveau pour distinguer les rôles.

2.2 UML et MASCARET

Après avoir décrit l'utilisation que l'on pourrait faire des différents diagrammes UML, je présente dans cette partie comment le modèle de l'environnement virtuel est décrit en UML, et donc l'interprétation de la sémantique des différents diagrammes UML que l'on utilise. L'exemple que j'utilise ici est GASPARG. En effet, c'est l'application utilisant MASCARET qui est la plus aboutie.

Le modèle qui décrit l'environnement virtuel est organisé en trois paquets :

Organisations : Ce paquet contient la description des équipes d'agents de l'environnement virtuel.

Environnement : Ce paquet contient la description des objets qui sont représentés dans l'environnement virtuel.

Agent : Ce paquet contient la description de toutes les classes d'agents qui constituent l'environnement social de l'environnement virtuel.

La Figure 2.6 montre le modèle de GASPARG avec cette organisation.

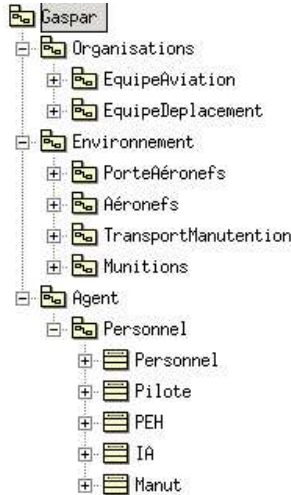


FIG. 2.6 – Le modèle de GASPARG.

Avant de décrire le contenu de ces différents paquets, je vais expliquer comment nous passons du modèle UML à un modèle exploitable par la plateforme. Pour cela, nous utilisons le langage XMI³. XMI est un procédé de sérialisation qui permet de décrire des objets UML sous forme XML. Ce qui est intéressant, c'est que le XMI est un format standard, et que la plupart des modeleurs UML permettent, après avoir créé un modèle UML de l'exporter au format XMI (donc sous la forme d'un document XML). De notre côté, nous avons intégré à VEHA un parseur XMI, qui permet à partir du document XMI de recréer les classes et les objets décrits dans le modèle UML.

Une fois le modèle XMI interprété de manière automatique par MASCARET, les éléments de ce modèle interprétés à différents niveaux de sémantique. Dans VEHA,

³XML Metadata Interchange. (<http://www.omg.org/technology/documents/formal/xmi.htm>)

nous utilisons une classe appelée **ModelLoader** qui interprète les paquets, les classes, les attributs. . . Puis, si nous sommes dans le cas d'un modèle utilisé dans le cadre d'environnements pour l'apprentissage de tâches collaboratives et procédurales, le modèle est également traité par le paquet **Procedural** qui ajoute de la sémantique via la classe **TeamLoader** en introduisant les notions d'équipes, de procédures. . . L'interprétation sémantique du modèle UML créé dépend donc des différents *loaders* qui traitent le modèle pour enrichir sa sémantique.

Ce que je décris ci-dessous représente l'organisation d'un modèle comme celui de GASPARET, qui doit être interprété par le **ModelLoader** et le **TeamLoader**. Tous les modèles décrivant des environnements pour l'apprentissage de tâches collaboratives et procédurales doivent respecter cette organisation pour être traités par MASCARET.

2.2.1 Le paquet Organisations

Ce paquet décrit les équipes d'agents qui peuvent être formées dans l'environnement virtuel. Chaque équipe est représentée par un paquet. À l'intérieur de chaque équipe, nous pouvons distinguer deux choses : **une collaboration**, qui décrit la structure organisationnelle de l'équipe, et **des diagrammes d'activités**, qui représentent les procédures que peut jouer cette équipe.

La collaboration : structure organisationnelle

Ici, nous avons choisi de représenter la structure organisationnelle de l'équipe par une collaboration. La Figure 2.7 montre la collaboration qui représente une équipe de GASPARET. Cette collaboration se décrit sous la forme d'une ellipse en pointillé dans laquelle est inscrit le nom de l'équipe. Chaque rôle de l'équipe est représenté par un rectangle relié à la collaboration par un trait. Dans le rectangle est inscrit le nom du rôle et la classe à laquelle un agent doit appartenir pour jouer ce rôle.

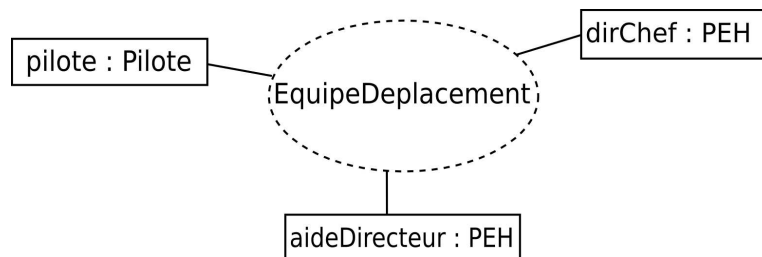


FIG. 2.7 – La collaboration représentant l'équipe de déplacement sur le porte-avions.

Nous avons choisi de représenter une équipe par une collaboration car cette représentation nous permet de visualiser de manière simple quels rôles sont impliqués dans l'équipe et de quelles classes sont les agents qui jouent ces rôles. Malheureusement, cette représentation n'est pour l'instant pas exploitée par MASCARET car le modèleur UML utilisé pendant mon stage (Objectteering⁴) ne permet pas d'exporter les collaborations en XMI.

⁴<http://www.objectteering.com/>

Un diagramme d'activité : une procédure

Un paquet représentant une équipe peut contenir des diagrammes d'activité qui sont alors interprétés comme des procédures (par le paquet **Procedural**). Représenter des procédures par des diagrammes d'activité a constitué une grosse partie de mon stage, c'est pourquoi les détails de cette représentation sont expliqués dans la partie 2.3.

2.2.2 Le paquet Environnement

Le paquet **Environnement** regroupe les objets de l'environnement. Chaque objet est représenté par une classe, et possède donc des attributs et des opérations. Un objet peut également posséder une machine à états. Cette machine à états décrit alors les différents états que peut prendre un objet et les transitions possibles entre ces états. La Figure 2.8 présente pour exemple le paquet environnement de GASPARG, et la Figure 2.9 montre la machine à états de l'objet **Plaque de protection**. Sur cette dernière figure, nous voyons que le changement d'état peut être déclenché par la réception d'un signal (**Ouvre** ou **Ferme**), ce qui nous intéresse pour la modification d'états des ressources dans une procédure, comme décrit dans la section 2.3.

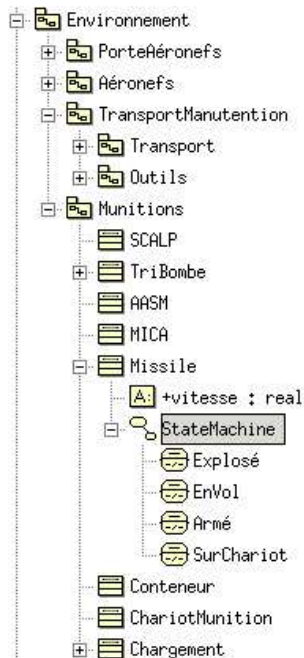


FIG. 2.8 – Le paquet **Environnement** de GASPARG.

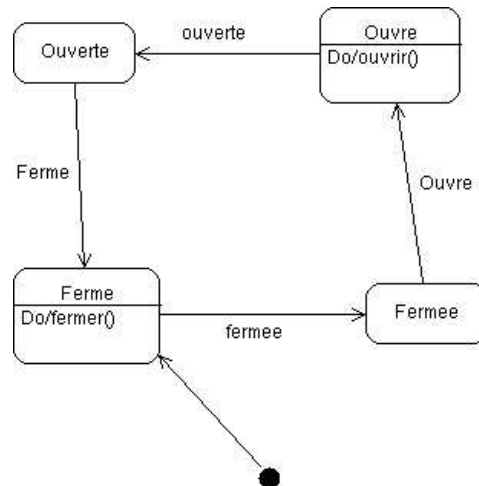


FIG. 2.9 – La machine à états de **Plaque de protection**.

2.2.3 Le paquet Agent

Le dernier paquet qui décrit notre environnement est le paquet **Agent**. Ce paquet décrit toutes les classes d'agents qui peuvent se trouver dans l'environnement virtuel. Chaque classe décrit les actions qu'un agent de cette classe peut effectuer.

Les Figures 2.10 et 2.11 montrent la composition du paquet **Agent** de GASPARG.



FIG. 2.10 – Le paquet **Agent** de GASPAR.

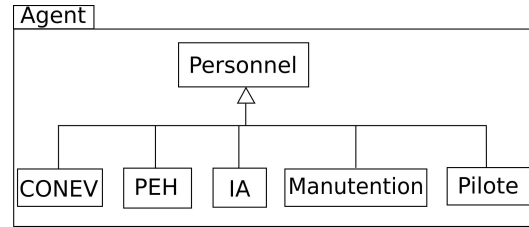


FIG. 2.11 – Les classes du paquet **Agent**.

2.3 Représentation des procédures par des diagrammes d'activités

2.3.1 Représentation en UML

Pour représenter des procédures en UML, plusieurs possibilités peuvent être explorées. En effet, je me suis intéressé à deux diagrammes UML : le diagramme d'activité, et le diagramme de séquence. Cependant, le diagramme de séquence possède quelques inconvénients par rapport au diagramme d'activité. En effet, le diagramme de séquence ne permet pas d'envoyer des signaux, ni de modifier l'état d'un objet, ce qui rend impossible la modification d'état de ressources (comme nous le verrons par la suite). De plus, la représentation sous forme de diagramme de séquence est moins claire visuellement lorsqu'on utilise les boucles ou les conditions par exemple.

Lorsque j'ai débuté mon stage, le modèle utilisé dans MASCARET pour représenter les procédures était le modèle des graphes d'activité d'UML 1.4. Cependant, le modèle n'était pas complètement implémenté (il était par exemple impossible de faire des branchements conditionnels ou des synchronisations). De plus, l'algorithme de suivi de procédure utilisé ne permettait pas de faire reboucler la procédure.

Une grosse partie de mon travail de stage a consisté à étudier le modèle des activités d'UML 2.0, de le comparer avec celui d'UML 1.4, puis de l'implémenter dans VEHA. La Figure 2.12 montre le modèle des activités que j'ai implémenté pour la représentation des procédures. Ce modèle s'inspire du modèle des activités d'UML 2.0, mais en le simplifiant car de nombreuses notions ne sont pas nécessaires pour représenter des procédures.

Ci-dessous, je décris les différentes classes, à quoi elles servent, et leur correspondance avec l'ancien modèle d'UML 1.4.

- **ActivityGraph** : Cette classe s'appelle en fait Activity dans UML 2.0 mais je l'ai appelé ActivityGraph pour éviter les confusions (avec la classe Activity d'ARéVi par exemple). Elle consiste en un ensemble de noeuds (ActivityNode) reliés par des transitions (ActivityEdge), qui permettent d'organiser un flux de contrôle (ControlFlow) qui organise l'exécution des noeuds et un flux de données (ObjectFlow) qui organise les données produites ou utilisées par les noeuds. C'est cette classe que l'on assimile à une procédure.

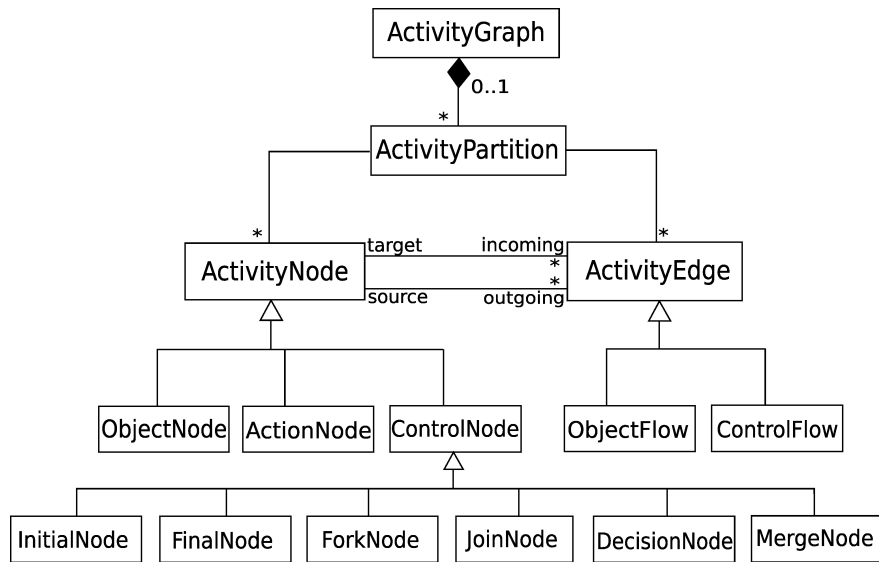


FIG. 2.12 – Le modèle des activités implémenté dans VEHA.

- **ActivityPartition** : Une ActivityPartition est un groupe qui identifie des actions qui ont des caractéristiques communes. Une ActivityPartition correspond à une Partition d’UML 1.4. Le regroupement des actions n’affecte pas les flux dans le graphe, il fournit juste une vue de l’activité. En ce qui nous concerne, une ActivityPartition représente l’activité d’un rôle. L’ActivityGraph est divisé en plusieurs ActivityPartition, ce qui représente le fait qu’une procédure fait intervenir plusieurs rôles qui se partagent l’activité.
- **ActivityNode** : Un ActivityNode est une classe qui représente des points dans le flux d’une activité, connectés par des arcs (ActivityEdge). Un ActivityNode en lui-même possède peu de sémantique, mais ses enfants (ActionNode, ObjectNode, ControlNode) sont plus intéressants. Un ActivityNode peut correspondre à un StateVertex d’UML 1.4.
- **ActivityEdge** : Un ActivityEdge est une connexion directe entre des ActivityNode, qui possède donc une source et une destination, qui est parcourue par le flux d’activité. La classe ActivityEdge possède deux enfants : ControlFlow représente un arc sur lequel circule le flux de contrôle de l’activité (déroulement des actions), et ObjectFlow sur lequel circule le flux de données (objets...).
- **ActionNode** : Un ActionNode est un ActivityNode qui exécute une action. Lorsque le flux d’exécution arrive à l’entrée du noeud, l’action associée est exécutée, et lorsque celle-ci est terminée, le flux d’exécution sort du noeud par les ActivityEdge sortants. Un ActionNode peut posséder des pré et post-conditions. Un ActionNode équivaut à un ActionState d’UML 1.4.
- **ControlNode** : Un ControlNode est un ActivityNode qui sert à organiser le flux dans une activité. Cela comprend : InitialNode, FinalNode, ForkNode, JoinNode, MergeNode et DecisionNode. Cela correspond au PseudoState d’UML 1.4. Nous pouvons alors réaliser des synchronisations ou des branchements conditionnels dans une procédure.

- **ObjectNode** : Un ObjectNode est un ActivityNode qui indique qu’une instance d’un classier particulier, éventuellement dans un état particulier, peut être disponible à un point particulier de l’activité. Cela permet de représenter le passage d’objets en entrée ou sortie de noeuds dans le graphe. Cela correspond aux ObjectFlowState d’UML 1.4. Nous utilisons les ObjectNode pour représenter les ressources d’une procédure.

Ce modèle nous permet donc de représenter des procédures. La Figure 2.13 décrit un extrait d’une procédure de GASPARG qui utilise ces notions.

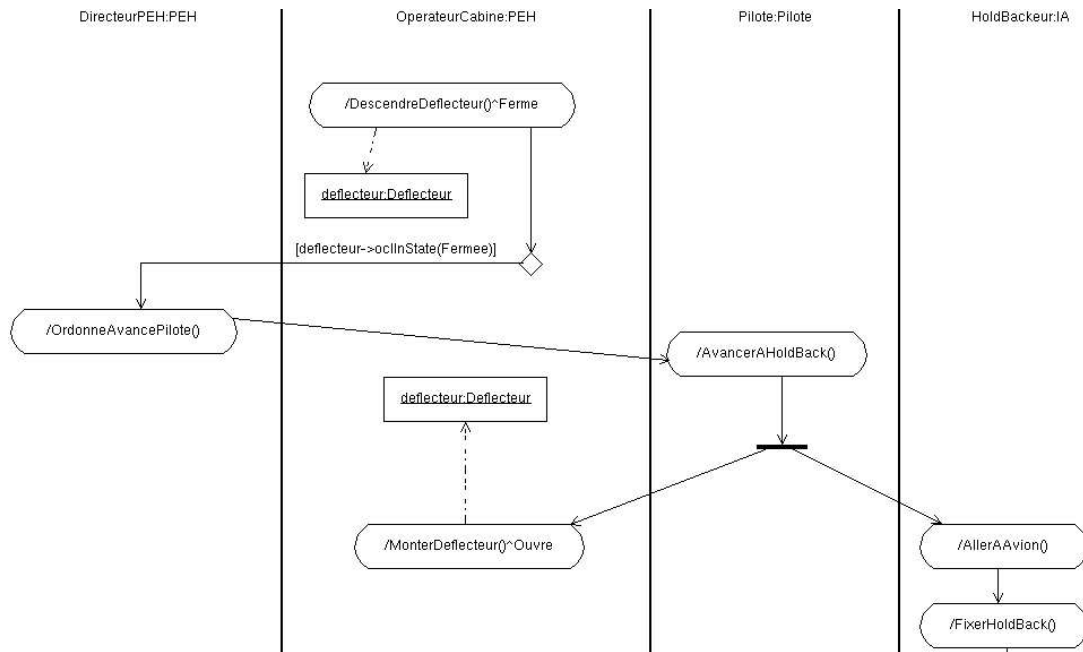


FIG. 2.13 – Une procédure de GASPARG représentée par un diagramme d’activité.

Ceci est un extrait de la procédure de catapultage de GASPARG. Elle illustre les notions que j’ai décrites précédemment. Je vais expliquer plus précisément chaque point intéressant.

On voit ici que cette procédure implique quatre rôles qui sont représentés par les quatre partitions. Les actions que doivent effectuer les rôles sont représentées par les ActionNodes (rectangles arrondis), et l’enchaînement entre les actions par les ActivityEdges (flèches). Les ressources de la procédure sont représentées par les ObjectNodes (rectangles).

Si on considère la première action de cette procédure (DescendreDeflecteur), on constate qu’elle pointe vers la ressource **deflecteur**. De manière générale, lorsqu’un ActionNode pointe vers un ObjectNode, cela signifie que l’action porte sur cette ressource. Dans ce cas précis, on constate également que cette action envoie un signal (un message) à la ressource. On le voit au nom de l’action qui s’écrit : **nomAction^nomSignal**. Comme l’action pointe vers la ressource **deflecteur**, le signal **Ferme** sera envoyé à l’objet **deflecteur**. Ce signal sera traité par la machine à états de l’objet en question

qui changera d'état ou non, en fonction des conditions de changement d'état de la machine à états ⁵. Si un `actionNode` envoie un signal mais ne pointe vers aucune ressource, le signal est envoyé en diffusion à toutes les entités de l'environnement. Cela représente la manière que l'on a choisi pour modifier l'état d'une ressource dans une procédure représentée par un diagramme d'activité. C'est un choix qui illustre l'interprétation de la sémantique d'UML que nous avons réalisée pour l'adapter à la représentation d'un environnement virtuel.

Ensuite, on peut voir un branchement conditionnel. L'action **OrdonneAvancePilote** sera réalisée uniquement si la condition `deflecteur->oclInState(Fermee)` est vraie, et donc si le déflecteur est fermé. Comme le branchement ne comporte qu'une branche, la procédure testera cette condition jusqu'à temps qu'elle soit vraie. La condition est exprimée en OCL ⁶. **VEHA** intègre un interpréteur OCL simplifié qui permet de traiter ce genre de condition.

Ce qui va donc se passer ici, c'est que l'action **DescendreDeflecteur** va envoyer le signal **Ferme** au déflecteur. La machine à états du déflecteur va traiter ce signal et faire passer le déflecteur dans l'état **Ferme** qui va réaliser l'action **Fermer** du déflecteur. On voit alors le déflecteur se fermer. Pendant ce temps, la procédure ne fait plus rien car on attend que le déflecteur soit dans l'état **Fermee**. Dès que c'est le cas, on passe à la suite de la procédure.

L'exemple décrit ici est un exemple de procédure assez simple, cependant tous les autres noeuds de contrôle ont également été implémenté (*Join, Fork, Initial, Final, Decision, Merge*).

Si nous nous sommes intéressé ici à l'application des diagrammes d'activités pour représenter les procédures (car c'est un bon exemple d'interprétation de la sémantique d'UML), nous pouvons les utiliser directement à partir de **VEHA**. En effet en UML, l'exécution d'une opération peut être décrite par un diagramme d'activité, ce qui est possible dans **VEHA** puisque le modèle des activités que j'ai implémenté est directement accessible.

2.3.2 Algorithme de suivi de procédure

Chaque agent qui prend part à une procédure dans MASCARET est doté d'un comportement procédural : le **ProceduralBehavior**. C'est ce comportement qui lui permet de suivre le déroulement d'une procédure et de réaliser les bonnes actions au bon moment. J'ai travaillé sur ce comportement afin de refaire l'algorithme de suivi de procédures qui comportait quelques inconvénients. En effet, avec l'algorithme précédent, il était impossible de faire reboucler une procédure car les actions effectuées étaient marquées d'un *drapeau* les indiquant comme terminée, et elles ne pouvaient donc plus être exécutées à nouveau. De plus l'algorithme parcourait la procédure depuis le début à chaque exécution du **proceduralBehavior**. En plus de prendre du temps, cette méthode était inadaptée pour gérer les branchements conditionnels. En effet, si une condition change alors que l'on est déjà plus loin dans le graphe, comme

⁵La machine à états du déflecteur est représentée sur la Figure 2.9. C'est celle de la plaque de protection car le déflecteur hérite de plaque de protection. C'est d'ailleurs pour cela que le déflecteur peut être dans l'état *Fermee* ou *Ouverte*

⁶Object Constraint Language (<http://www.omg.org/technology/documents/formal/ocl.htm>)

l'algorithme re-parcourt le graphe en entier, il ne prendra pas la même branche et ne trouvera donc pas le noeud en cours.

J'ai donc mis au point un nouvel algorithme à base de pointeurs sur les noeuds du graphe. Chaque procédure possède autant de pointeurs que de noeuds en cours d'exécution (par défaut un seul, ce qui peut augmenter si on rencontre des Fork par exemple...). Les pointeurs possèdent un état qui peut être un des trois suivants :

- TODO : Le noeud n'a pas encore été évalué.
- DOING : L'action du noeud est en cours de réalisation.
- DONE : Le noeud est réalisé, on peut passer au suivant.

Le `proceduralBehavior` (comportement procédural de l'agent, qui tourne en boucle) se déroule comme ceci : Pour chaque procédure, on récupère tous les pointeurs. Pour chacun de ces pointeurs, on regarde si il pointe vers un noeud qui concerne l'agent (si il joue le rôle qui doit traiter le noeud). Si il est concerné, il traite le noeud en conséquence. Si c'est un `ControlNode` (Fork, Join...), il lui demande ses noeuds suivants, il enlève le pointeur sur ce noeud et en ajoute un en état TODO sur ses suivants. Si c'est un `ActionNode` on regarde l'état du pointeur. S'il est en état TODO, l'agent lance l'exécution de l'action, s'il est en état DOING, il ne fait rien tant que l'action n'est pas finie, sinon le pointeur passe en DONE. Et si il est en état DONE, le pointeur sur ce noeud est retiré et on ajoute un pointeur en état TODO sur tous les noeuds suivants. S'il n'y a plus aucun pointeur dans la procédure, alors elle est terminée. Le pseudo-code de cet algorithme est décrit sur le listing 1.

Ce à quoi il faut faire attention ici c'est que les agents ne travaillent pas sur la même instance de procédure. Chacun d'eux possède une copie de la procédure. En effet, il est prévu que les agents peuvent s'exécuter sur des machines différentes, il doivent donc posséder une instance de la procédure sur leur machine. De plus, cela peut permettre de représenter le fait que des agents possèdent une vision différente de la procédure. À chaque fois qu'un agent modifie l'état de la procédure (ajout ou suppression d'un noeud), il doit donc envoyer un message aux autres agents de l'équipe qui joue la procédure pour qu'ils fassent de même et donc soient au courant de l'état d'avancement de la procédure. C'est pourquoi à la fin du `proceduralBehavior`, les agents doivent gérer les messages.

De plus, avant de lancer une action on vérifie que le rôle n'est pas inhibé. Si un rôle est inhibé, cela signifie qu'il est joué par un utilisateur humain (un apprenant par exemple). Dans ce cas, on ne lance pas l'exécution de l'action puisque c'est l'utilisateur qui devra le faire. Cependant, si le rôle est inhibé, on continue quand même à suivre l'évolution de la procédure. Cela permet à un éventuel ITS de savoir l'action que l'utilisateur devrait être en train de réaliser et donc si ce qu'il fait est correct ou non. De plus, cela permet de rendre la main de manière dynamique à l'agent, qui peut alors reprendre la procédure à l'endroit où l'utilisateur l'a laissée.

Cet algorithme corrige les inconvénients de celui utilisé précédemment. En effet, on peut très bien faire reboucler la procédure puisque dès qu'une action est terminée, on retire le pointeur sur le noeud qui peut donc accepter un autre pointeur plus tard. De plus, on ne parcourt plus le graphe depuis le début à chaque fois puisqu'on reprend

```
1 pour tous les pointeurs de la procédure faire
2   si le noeud pointé me concerne alors
3     si c'est un noeud de contrôle alors
4       On demande les noeuds suivants;
5       pour tous les noeuds suivants faire
6         | On ajoute un pointeur en état TODO sur le noeud suivant;
7       fin
8       On retire le(s) pointeur(s) sur le noeud courant;
9     fin
10    sinon si c'est un noeud d'action alors
11      si le noeud est en état DOING alors
12        | si l'exécution de l'action est terminée alors
13          | On met le pointeur en état DONE
14        | fin
15      fin
16      si le noeud est en état DONE alors
17        | On retire le pointeur sur le noeud courant;
18        pour tous les noeuds suivants faire
19          | On ajoute un pointeur en état TODO sur le noeud suivant;
20        | fin
21      fin
22      si le noeud est en état TODO alors
23        | si le rôle n'est pas inhibé alors
24          | On lance l'exécution de l'action;
25        | fin
26      fin
27    fin
28  fin
29 fin
30 Gestion des messages;
```

directement aux noeuds en cours grâce aux pointeurs.

2.4 Conclusion et perspectives

Ce que nous avons vu dans ce chapitre, c'est comment à partir d'un modèle créé sur un modeleur UML, nous arrivons à décrire un environnement virtuel. Nous avons montré comment un même modèle pouvait être interprété avec une sémantique différente en fonction de l'usage que nous souhaitons en faire. Par exemple pour GASPARG, après une interprétation bas-niveau (classes, attributs, méthodes...) effectuée par VEHA, nous ajoutons du sens au niveau procédural grâce au paquet **Procedural**.

Un des principaux avantages de ce mécanisme est que toutes les notions du système sont réifiées. En effet, rien n'est codé *en dur*. Ceci est très important pour un EVF,

comme je l'ai précisé dans l'introduction. Par exemple, si un apprenant qui s'exerce sur une procédure commet une erreur, nous sommes capable de lui expliquer précisément pourquoi puisque le système contient la description de la procédure, l'enchaînement des actions, les rôles qui doivent jouer chaque action, les ressources nécessaires. . .

L'objectif de ce procédé à terme (notamment en ce qui concerne les procédures) est de devoir écrire le strict minimum de code. L'idéal serait de limiter le code aux opérations des agents. En effet, nous serons toujours obligé de coder les opérations élémentaires (marcher, lever les bras. . .). Cependant, il faudrait réussir à décrire les actions⁷ sans devoir les coder, ce qui est le cas actuellement. En effet, on devrait toujours réussir à décrire une action par un enchaînement d'opérations (marcher jusqu'à tel endroit, faire tel geste, attendre. . .), et par conséquent décrire les actions elles aussi par des sous-diagrammes d'activités que l'on pourrait intégrer aux diagrammes d'activités représentant les procédures. On obtiendrait alors plusieurs diagrammes d'activités avec des niveaux de granularité différents, comme celui de la Figure 2.14. Le problème à ce niveau serait de savoir où s'arrêtent les opérations et où commencent les actions.

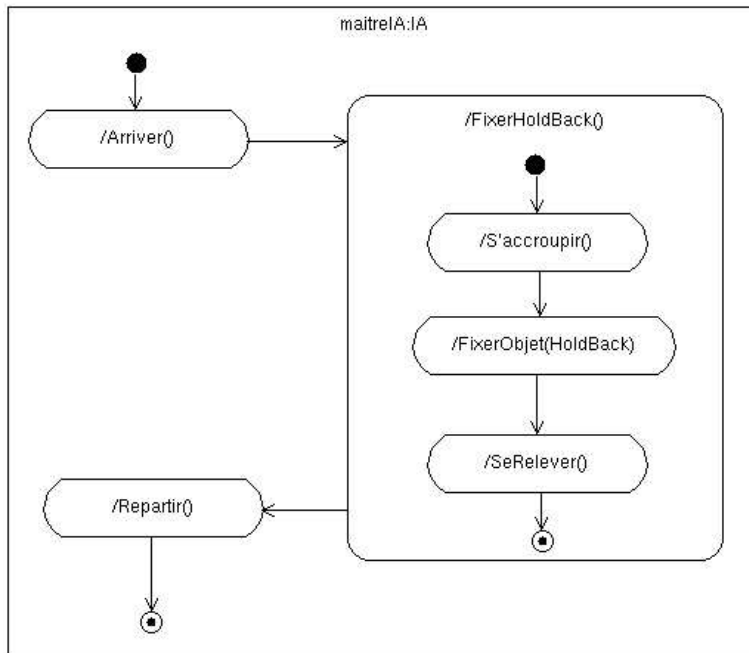


FIG. 2.14 – Un exemple d'action décomposée en opérations et intégrée dans une procédure.

⁷Une opération est une action élémentaire alors qu'une action est en théorie un enchaînement d'opérations que l'on utilise dans une procédure. L'action **FixerHoldBack** par exemple consiste à s'accroupir, accrocher un objet puis se relever (trois opérations élémentaires).

Chapitre 3

Organisation des agents

Dans le chapitre précédent, nous avons vu la manière dont le modèle UML était transformé en modèle numérique grâce à au format XML. Cependant, il reste un problème qui concerne le modèle numérique lui même. En effet, même si nous arrivons à représenter les notions d'équipes, de rôle ou d'agent, il nous reste à déterminer comment organiser ces notions dans le modèle.

Dans ma bibliographie, j'ai étudié différents modèles d'organisation sociale pour les systèmes multi-agents afin de tirer les avantages et les inconvénients de chacun dans le cas d'étude qui nous intéresse (EVF pour l'apprentissage de tâches procédurales et collaboratives).

Dans la première section de ce chapitre, je présente différents modèles d'organisation sociale multi-agents que je critique en fonction des contraintes identifiés en introduction.

Dans la deuxième section, je décris les problèmes liés à l'organisation que l'on rencontre avec le modèle actuel dans MASCARET et je propose quelques pistes pour les résoudre sur lesquelles il faudrait réfléchir.

3.1 Différents modèles d'organisation existants

3.1.1 AGR

Le modèle AGR (Agent, Groupe, Rôle) [Gutknecht et al., 2004], est l'évolution du modèle AALAADIN. Comme le décrit la Figure 3.1, l'organisation dans ce modèle s'articule autour des notions d'agent, de groupe et de rôle.

Agent : Dans ce modèle, un agent est une entité capable d'agir et de communiquer, qui peut jouer un ou plusieurs rôles dans un ou plusieurs groupes. Il n'existe aucune contrainte sur la structure interne de l'agent.

Groupe : Un groupe est un ensemble d'agents qui partagent des caractéristiques. Il est utilisé pour diviser l'organisation en groupant des agents dont l'activité se rejoint. Deux agents peuvent communiquer uniquement s'ils appartiennent au même groupe.

Rôle : Un rôle est la représentation abstraite de la fonction d'un agent dans un groupe. Un rôle est local à un groupe et n'a plus aucun sens en dehors de ce contexte. Pour

jouer un rôle dans le groupe, l'agent doit postuler pour ce rôle, et l'affectation se fait en fonction de ses compétences.

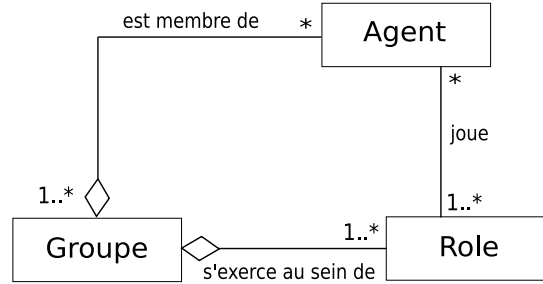


FIG. 3.1 – Le modèle d'organisation AGR

D'après les auteurs, ce modèle est volontairement très générique pour pouvoir s'adapter à tous les types de systèmes multi-agents. C'est pourtant ce qui peut lui être reproché. En effet, sa structure minimaliste ne permet pas de l'utiliser tel quel. Il s'agit plutôt d'un patron de conception à partir duquel on pourrait créer un modèle vraiment implémentable, doté d'une sémantique plus précise [Querrec, 2002].

La notion de groupe notamment ne constitue en fait qu'une liste des agents qui le constituent. Ce concept pourrait être étendu dans le but de faire du groupe, un ensemble de travail offrant aux agents un lieu privilégié d'interactions ou comme on l'utilise dans MASCARET, une équipe.

De plus, le modèle AGR ne fournit aucun moyen de représenter les fonctions d'un rôle ou les compétences nécessaires pour pouvoir jouer ce rôle.

3.1.2 Moise+

Dans [Hübner et al., 2002], les auteurs expliquent qu'il existe deux familles de modèles organisationnels pour les SMA. D'un côté, on trouve les modèles qui s'appuient sur des plans globaux, qui s'intéressent au fonctionnement de l'organisation. De l'autre côté, il y a les modèles qui se concentrent plutôt sur la structure du système, sur les rôles et les relations entre eux.

Le modèle Moise+ (Model of Organization for multi-Agent SystEms) est une tentative d'unification de ces deux familles, on y retrouve donc une spécification structurelle (structure du SMA) et une spécification fonctionnelle (buts à atteindre). Afin d'explicitier la manière dont les agents collaborent pour concourir aux buts communs, ces deux spécifications sont reliées par une spécification déontique.

Spécification structurelle : définit les relations entre les agents à travers les notions de groupes, rôles et liens.

Spécification fonctionnelle : explicite comment un SMA atteint généralement son but global (décomposition du but en plans, puis distribution de missions aux agents).

Spécification déontique : décrit les permissions et les obligations des rôles pour les missions.

La spécification structurelle est organisée sur trois niveaux : le niveau individuel, le niveau social et le niveau collectif. Le niveau individuel est constitué d'un ensemble de rôles. Chaque rôle décrit le comportement attendu d'un agent lorsqu'il choisit de jouer ce rôle. Le niveau social définit les relations entre les rôles (liens) qui viennent contraindre et structurer les agents en fonction des rôles qu'ils jouent. Enfin, le niveau collectif rassemble les rôles en groupes.

On s'aperçoit ici que la spécification structurelle de Moise+ équivaut au modèle AGR. Nous pouvons donc formuler les mêmes remarques que précédemment.

La spécification fonctionnelle dans Moise+ est structurée en différents schémas sociaux. Un schéma social est un ensemble de plans globaux organisé en missions. Les plans globaux définissent des arborescences de buts collectifs. La Figure 3.2 représente un exemple d'arborescence de buts, qui utilise des opérateurs de séquence (choix et parallélisme). Une mission est définie comme un ensemble cohérent de buts de l'arborescence. L'agent qui accepte une mission s'engage à réaliser tous les buts de celle-ci.

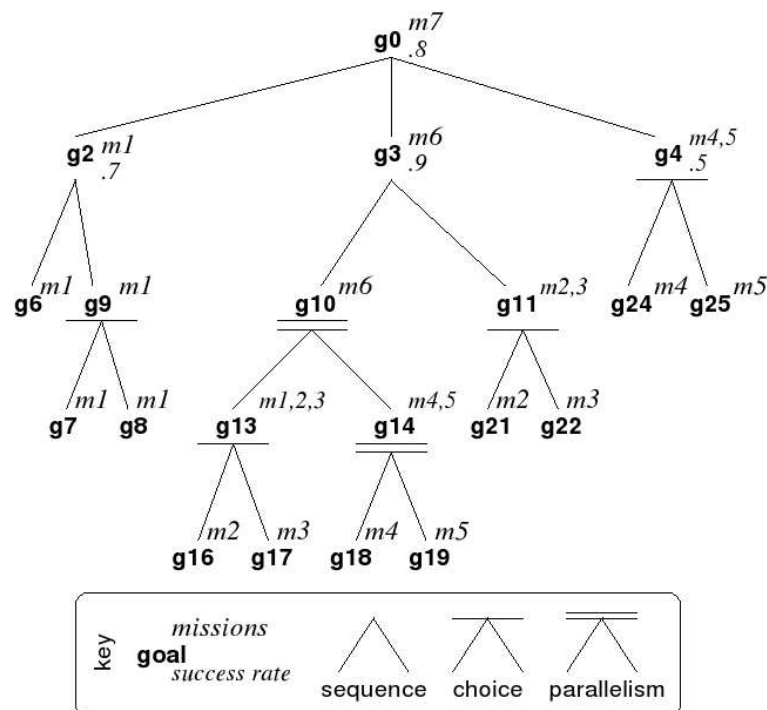


FIG. 3.2 – Un exemple de hiérarchie de buts avec Moise+

Enfin, la spécification déontique de Moise+ définit la relation des rôles vers les missions en termes de permissions et d'obligations. Si un agent a une obligation envers une mission, il a également une autorisation pour cette mission.

3.1.3 MASCARET

Le modèle MASCARET (MultiAgent System for Collaborative and Adaptive Realistic Environment for Training) [Querrec, 2002], est un modèle qui a été mis au point spécialement pour les EVF. C'est le modèle actuel utilisé par le projet MASCARET (et donc par GASPARET). Il comprend un modèle générique d'organisation qui est ensuite dérivé de deux manières pour s'adapter d'une part à l'environnement physique, et d'autre part à l'environnement social.

Le modèle générique de MASCARET, représenté sur la Figure 3.3 reprend la notion de rôle, qui désigne ici les différentes responsabilités des agents dans l'organisation. Chaque rôle impose des pré-requis, une liste de capacités qu'un agent doit posséder pour pouvoir jouer ce rôle. Dans MASCARET, un groupe s'appelle une organisation. Dans le modèle dérivé appliqué à l'environnement social représenté Figure 3.4, on voit que les groupes utilisés sont en fait des équipes. En effet, MASCARET s'applique à des EVF pour les tâches collaboratives. De plus, les équipes gérées sont hiérarchiques, les agents doivent donc pouvoir gérer les relations avec leurs supérieurs et leurs subordonnés. Comme le décrit la Figure 3.4, une équipe doit remplir plusieurs missions. Une mission consiste en un ensemble de buts et chaque agent de l'équipe qui effectue cette mission doit oeuvrer pour la réalisation de ces buts. Une mission sert de cadre à la coordination des agents.

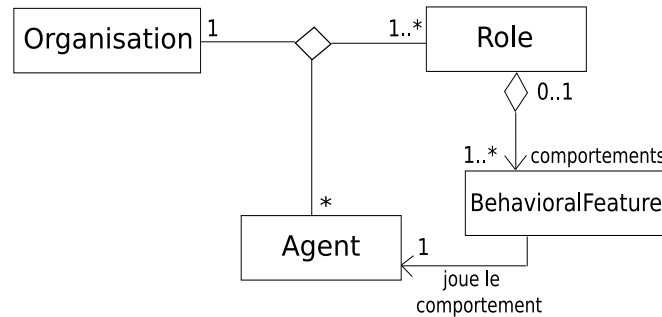


FIG. 3.3 – Le modèle générique de MASCARET

MASCARET s'intéresse plus particulièrement aux EVF dans lesquels les missions sont des procédures et sont alors représentées par des ensembles de contraintes qui décrivent l'agencement des actions. La procédure décrit les interactions métiers entre les agents dans le cas optimal, et laisse aux agents la responsabilité de se construire des plans implicites.

Pour participer efficacement à cette organisation, chaque agent est doté d'un comportement collaboratif et d'un comportement organisationnel.

Comportement collaboratif : C'est ce comportement qui est utilisé par l'agent pour participer à la réalisation des buts de l'équipe (suivi de la procédure...).

Comportement organisationnel : Le comportement organisationnel permet à l'agent d'intégrer la structure de l'organisation dans son raisonnement. Dans une organisation hiérarchique, si un agent est face à un problème (que le comportement

collaboratif ne lui permet pas de résoudre), il en réfère à son supérieur.

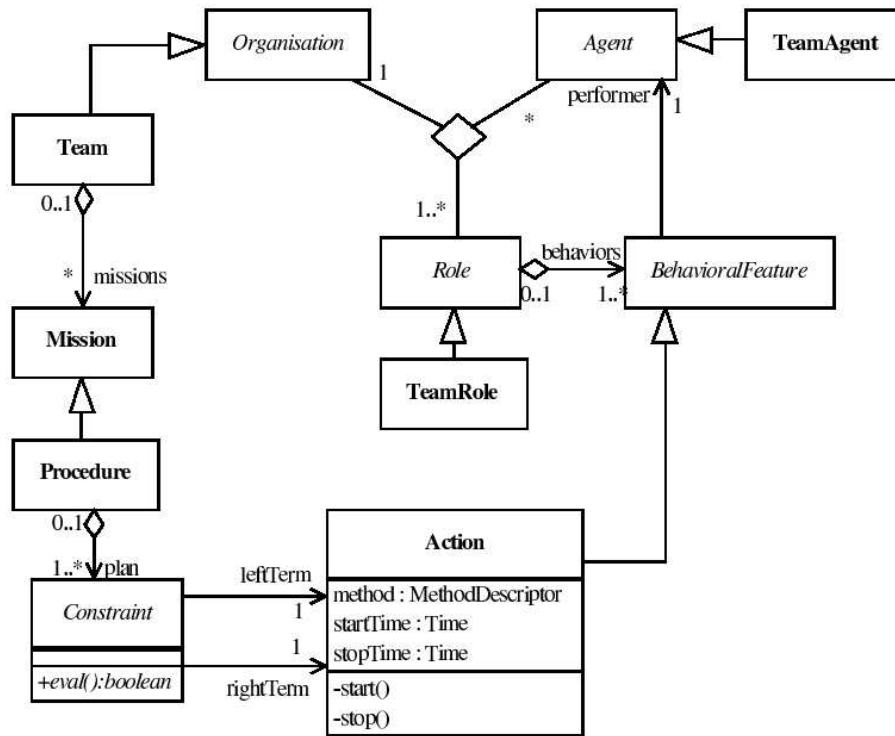


FIG. 3.4 – Le modèle complet de l’environnement social de MASCARET

Le principal avantage de MASCARET dans le cadre de notre étude vient du fait que c’est un modèle conçu pour s’appliquer à des EVF. Ainsi, il prend déjà en compte les contraintes que nous avons présentées en introduction (inhibition de phénomènes, prise de contrôle dynamique. . .). Cependant, son plus grand défaut vient du fait que le modèle créé est un modèle générique qui se dérive en deux modèles distincts pour l’environnement physique et l’environnement social. Malgré l’élégance de cette démarche, cela conduit à des modèles compliqués du fait de la dépendance au modèle générique. Nous pensons que les environnements physique et social, bien qu’étant tous deux des SMA, sont assez différents et doivent être organisés de manière différente. C’est pourquoi nous nous intéressons ici uniquement à l’organisation dans l’environnement social. Les différents problèmes que l’on rencontre à l’utilisation de ce modèle avec GASPARET sont détaillés dans la section 3.2.

3.1.4 OMNI

OMNI (Organizational Model for Normative Institutions) [Vázquez-Salceda et al., 2004], est plus qu’un simple modèle d’organisation sociale, c’est un *framework* pour modéliser des organisations d’agents. OMNI a été créé avec pour objectif d’équilibrer les besoins globaux de l’organisation avec l’autonomie des agents. Cela signifie que le modèle d’or-

ganisation ne doit pas être trop centré agent (il devient alors difficile de modéliser les interactions des agents dans la société), ni trop centré organisation (sinon les agents se retrouvent fixés dans des protocoles rigides).

Pour cela, le cadre d'OMNI se décompose en trois dimensions :

La dimension organisationnelle : Elle décrit la structure d'une organisation et peut être vue comme un moyen de gérer des dynamiques complexes. Elle met en place les relations entre trois modèles : le modèle organisationnel qui spécifie les caractéristiques de la société en termes de structure sociale (rôles) et de structure d'interaction (scripts de scènes), le modèle social qui fixe l'acceptation des rôles par les agents dans des contrats sociaux qui décrivent leurs droits, leurs devoirs et leurs responsabilités, et le modèle d'interaction dans lequel des scènes d'interaction concrètes sont créées dynamiquement par les agents, basées sur les scripts identifiés auparavant.

La dimension normative : Elle représente les normes et les règles auxquelles les agents doivent adhérer. Après avoir défini les normes au niveau de l'organisation, elles sont transformées en règles interprétables par les agents (qui savent alors quoi faire, mais pas comment). Enfin, on définit les mécanismes finaux pour que les agents interprètent ces règles. Il y a alors deux approches, soit on crée un interpréteur que tous les agents devront posséder, soit on transforme les règles en protocoles à inclure dans les contrats d'interaction.

La dimension ontologique : Elle définit à la fois le contenu et le langage de communication sur plusieurs niveaux d'abstraction. On commence par définir tous les concepts du *framework* lui-même tels que les normes, règles, rôles, groupes... Ensuite on spécifie le contenu de la communication, ou connaissance du domaine. Enfin, on décrit les actes de langages spécifiques utilisés par les agents pour communiquer.

Dans OMNI, en plus des agents externes (agents qui pour une raison ou une autre peuvent décider ou non de jouer un rôle dans la société), il y a aussi des agents institutionnels (qui jouent des rôles de "facilitation"), qui veillent au bon fonctionnement de la société. Par exemple, un agent peut avoir pour rôle de noter chaque agent en terme de performance afin d'établir un annuaire de confiance, ou encore être responsable de décider qui entre dans la société ou non.

Une très bonne idée d'OMNI est de représenter explicitement la dimension ontologique. En effet dans un SMA, la communication est un des éléments les plus important. Une bonne organisation ne sert à rien si les agents sont incapables de se transmettre correctement des informations. La communication est un domaine pour lequel on peut s'inspirer de l'expérience humaine. En effet, de nombreuses études existent et nous permettent d'identifier les principes fondamentaux d'une bonne communication dans un groupe. Par exemple, même si cela paraît évident, avoir un langage commun et précis est un élément indispensable pour une bonne compréhension. De même, la topologie d'un réseau de communication affecte les performance d'un groupe [[Anzieu et Martin, 1973](#)].

Le problème que l'on rencontre avec OMNI est que la gestion des objectifs de l'organisation se fait à l'intérieur de sa dimension organisationnelle (et donc concerne

la structure du SMA). Cela signifie que l'organisation est conçue avec un but précis et ne peut alors être utilisée dans un autre but sans subir de modification. Cela peut être assez gênant dans un environnement dynamique dans lequel les buts peuvent changer.

3.2 Problèmes rencontrés avec l'organisation existante

Les problèmes que nous décrivons ci-dessous sont des problèmes qui sont apparus au fur et à mesure de l'utilisation de MASCARET avec GASPARG. Ces problèmes sont principalement liés à la relation entre les agents, les rôles, les équipes et les procédures, et donc liés au modèle d'organisation.

Tout d'abord, il faut s'intéresser à la notion d'équipe. Qu'est-ce qu'une équipe ? Dans GASPARG, nous pouvons affirmer qu'une équipe est un groupe d'agents qui réalisent une procédure ensemble. Cependant, si les rôles qui composent une équipe sont fixes (structure organisationnelle), il n'en est pas de même pour les agents qui jouent ces rôles (entité organisationnelle). Il se pose donc le problème de l'affectation des rôles aux agents. Lorsqu'on lance GASPARG, on lui fournit plusieurs fichiers de configuration, (au format XML) dont un qui lui indique le rôle que va jouer chaque agent, et un autre qui lui indique quelle procédure doit être lancée, à quelle heure et avec quelle ressource. Pour décrire le problème, je vais prendre un exemple simple, celui du catapultage d'un rafale sur le porte-avions. La Figure 3.5 représente la configuration du porte-avions lors du catapultage de plusieurs rafales.

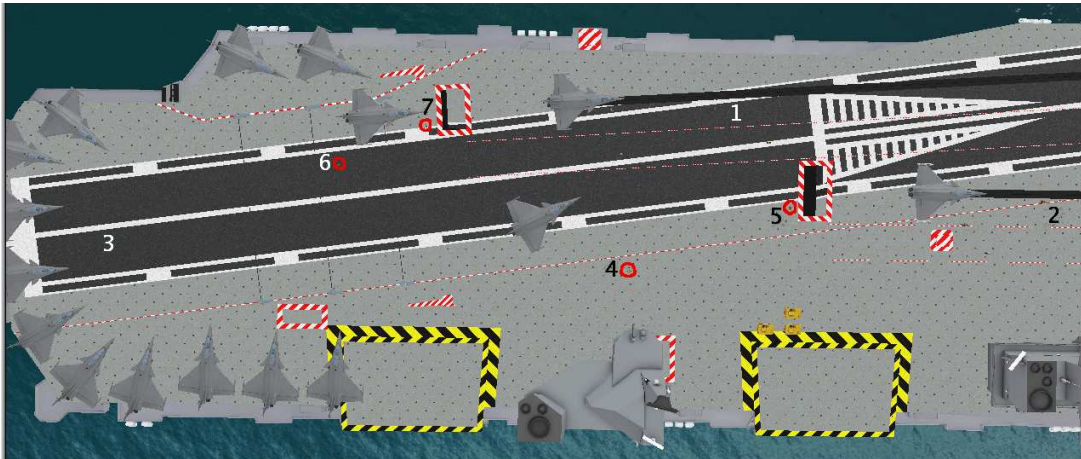


FIG. 3.5 – Configuration du porte-avions pour une procédure de catapultage sous GASPARG.

Le porte-avions représenté dans GASPARG comporte deux catapultes, une catapulte axiale (2) et une catapulte latérale (1). Les rafales sont garés sur un grand parking à l'arrière du porte-avions (3) et peuvent être catapultés sur n'importe laquelle des deux catapultes. Le catapultage d'un rafale se déroule en deux parties. Tout d'abord, on doit amener le rafale jusqu'à la catapulte, ce qui est le travail d'une équipe appelée "Équipe de déplacement", puis, le rafale est préparé puis catapulté par l' "Équipe aviation".

Nous nous intéressons pour cet exemple à la première partie de la procédure, et donc à l'équipe de déplacement.

Cet équipe est composée de trois rôles : le pilote du rafale, l'aide directeur qui guide le pilote pour sortir du parking (4 et 6), et le directeur chef (5 et 7) qui prend le relais et guide le pilote jusqu'à la catapulte. Le problème de cette équipe est que les agents qui jouent chaque rôle ne sont pas fixes, et peuvent changer à chaque catapultage. En effet, le rôle pilote sera joué par le pilote qui se trouve dans le rafale qui va être catapulté. Ensuite, celui qui va jouer le rôle d'aide directeur est celui qui est le plus proche du rafale à faire décoller. Enfin le directeur chef sera l'agent qui est placé devant le déflecteur correspondant à la catapulte que l'on va utiliser.

L'équipe est donc fortement dynamique car sa constitution peut changer entre chaque procédure. En effet, le pilote du rafale change à chaque fois, ce qui est normal, puis, en fonction de sa position dans le parking et de la catapulte utilisée, le reste de l'équipe peut être constituée des agents 4 et 5, 6 et 7, 4 et 7 ou 6 et 5. Il se pose alors le problème de l'affectation dynamiques de rôles. Actuellement, l'affectation des rôles est statique et codée en dur pour chaque procédure, ce qui n'est pas une solution satisfaisante. Une solution pourrait être de représenter les contraintes d'affectation de rôles par des contraintes OCL que l'on apporterait à l'organisation, comme nous avons vu qu'il était possible de le faire dans la section 2.1.1. Cependant, comme je l'ai déjà précisé, c'est impossible à réaliser dans l'état actuel des choses étant donné que le modèleur UML utilisé (Objecteering) n'exporte pas les collaborations. Il faudrait alors se pencher sur l'utilisation d'un autre modèleur.

Un autre problème qui peut être identifié au niveau de l'organisation concerne les *actions organisationnelles*. Ce que j'entends ici par action organisationnelle, c'est une action qui n'est pas métier, qui a un effet sur l'organisation. Jouer un rôle ou démarrer une procédure sont des actions organisationnelles. Agiter un drapeau n'est pas une action organisationnelle. L'exemple que je vais prendre pour illustrer le problème concerne le démarrage d'une procédure d'une équipe par une autre équipe. Typiquement, dans le catapultage que j'ai décrit juste au-dessus, j'ai dit qu'il y avait deux équipes, l'équipe de déplacement et l'équipe d'aviation. Une fois que l'équipe de déplacement a amené le rafale jusqu'au déflecteur, l'équipe d'aviation doit démarrer la procédure de préparation du rafale au catapultage. C'est donc en théorie l'équipe de déplacement qui doit *dire* à l'équipe d'aviation de démarrer sa procédure. C'est une action organisationnelle.

Actuellement, pour pouvoir faire ça dans MASCARET, nous sommes obligé de faire en sorte que le personnage qui joue le rôle qui effectue la dernière action dans la première équipe appartienne également à la deuxième équipe, même si il n'a rien à y faire, pour faire le lien entre les actions des deux procédures. C'est une solution qui n'est pas satisfaisante non plus. En effet, l'idéal serait de pouvoir faire communiquer les équipes afin que lorsque la première équipe termine sa procédure, elle envoie un message à la deuxième équipe pour la faire démarrer. Si cette solution est choisie, il restera encore à trouver la représentation UML appropriée.

D'une manière plus générale, il serait intéressant de faire communiquer les agents, et pour cela leur définir un langage commun (comme nous avons vu qu'il était intéressant de le faire dans OMNI par exemple, dans la section 3.1.4). Nous pourrions pour

cela utiliser la norme FIPA-ACL¹.

Enfin, le modèle d'organisation de base de MASCARET est censé permettre à un agent de réaliser plusieurs procédures en simultan . Cependant, cela necessiterait de sp cifier plus en d tail la mani re de g rer les proc dures simultan ment. En effet, certaines proc dures (que l'on peut qualifier d'exclusives) ne doivent pas  tre interrompues. D'autres au contraire n'imposent aucune contrainte de ce genre. De plus, dans certaines proc dures, il n'y a aucun probl me   ce que certains r les effectuent plusieurs proc dures alors que pour d'autres r les, c'est impossible. Il reste donc un gros travail de r flexion sur la gestion de plusieurs proc dures par les agents.

¹Foundation for Intelligent Physical Agents (<http://www.fipa.org>)

Conclusion

Comme nous l'avons vu dans le deuxième chapitre, la plupart de mon travail a porté sur la modélisation d'environnements virtuels en UML. J'ai d'abord étudié la sémantique de la nouvelle norme UML afin de voir quels diagrammes pouvaient être utilisés, et comment nous pouvions les interpréter pour modéliser l'environnement virtuel. Puis, je me suis plus particulièrement axé sur la représentation du modèle des procédures par des diagrammes d'activité. C'est ce qui m'a pris le plus de temps car c'est un problème assez complexe. En effet, il a fallu gérer tous les aspects du travail procédural (modification d'état de ressources, synchronisation d'actions ...). Puis, je me suis penché sur le comportement procédural des agents, en implémentant un algorithme de suivi de procédure permettant aux agents de réaliser la procédure à partir du graphe d'activité.

Cette partie orientée procédure est quasiment finalisée puisqu'il reste encore à essayer de réduire le code des actions en utilisant des sous-graphes d'activités décrivant des enchaînements d'opérations, comme je l'ai expliqué dans la section 2.4.

En ce qui concerne la partie organisation, il reste plus de travail. En effet le problème de l'organisation dans un système multi-agents est un problème complexe. Sur cette partie, mon travail a surtout consisté à identifier les problèmes de l'actuel modèle (que l'on repère surtout au fur et à mesure de l'utilisation de ce modèle), puis à réfléchir à des solutions. Nous avons vu que ces problèmes tournent beaucoup autour des notions de rôles, d'équipe et de procédure. En effet, même si on réussit à représenter les procédures et à les faire exécuter aux agents sans problèmes, il reste difficile de gérer l'affectation dynamique de rôles ou la gestion simultanée de plusieurs procédures par les agents.

De plus, il reste à enrichir la notion de communication de MASCARET entre les agents dans le but de passer d'un système coopératif² (les agents agissent ensemble dans le cadre d'une procédure définie) comme c'est le cas actuellement, à un système collaboratif³ (les agents agissent ensemble sans pour autant que leurs interactions aient été prédéfinies).

Enfin, même si j'en ai peu parlé, il reste à effectuer l'analyse de l'activité de l'environnement. Il faudrait pour cela remonter les actions effectuées par les agents afin de pouvoir visualiser ce qu'a fait chaque agent, chaque équipe, le temps pris par chaque procédure...

²Coopération : partage de tâches, chaque personne est responsable de sa partie du travail.

³Collaboration : engagement mutuel des participants dans un effort coordonné pour résoudre un problème.

Pour conclure ce document, je vais reprendre les quatre contraintes décrites dans le premier chapitre en expliquant l'impact de mon travail sur chacune d'entre elles :

Simulation : Nous avons dit que les agents évoluant dans l'environnement devaient adopter un comportement réaliste, afin que l'apprenant se sente immergé dans l'environnement. La partie de mon travail qui est en rapport avec cette contrainte est celle qui concerne le comportement procédural des agents (ProceduralBehavior). En effet, si les agents peuvent suivre une procédure de manière crédible, l'environnement est socialement plus réaliste.

Conception : Cette contrainte concerne la modification dynamique de l'environnement par le concepteur. La partie sur la modélisation en UML de l'environnement répond complètement à cette contrainte. L'environnement n'est plus décrit par du code informatique qu'il faut recompiler, mais par un modèle UML qu'il suffit de ré-exporter en XMI pour appliquer les modifications.

Réification : Le modèle doit être capable de répondre aux questions de l'apprenant sur l'organisation sociale du système. Le travail sur la modélisation UML entre encore en jeu ici. En effet, le fait que le modèle soit décrit en UML et non par du code implique que tous les éléments du système (y compris l'organisation sociale) sont réifiés dans le système.

Participation : Dans l'environnement virtuel, l'utilisateur peut prendre dynamiquement le contrôle d'un agent. Le travail que j'ai effectué sur le comportement procédural et plus particulièrement l'algorithme de suivi de procédure va dans ce sens. Comme nous l'avons vu dans la section [2.3.2](#), lorsque quelqu'un prend le contrôle d'un agent, l'agent continue à suivre la procédure mais laisse le soin à l'utilisateur de réaliser les actions. Cela permet au système de suivre l'évolution de l'apprenant et de savoir si ce qu'il réalise est correct ou non, et de pouvoir rendre la main dynamiquement.

Bibliographie

- [Anzieu et Martin, 1973,] Anzieu, D. et Martin, J.-Y. (1973). *La dynamique des groupes restreints*. 28
- [Bauer et Odell, 2004,] Bauer, B. et Odell, J. (2004). Uml 2.0 and agents : How to build agent-based systems with the new uml standard. 10, 11
- [Coutinho et al., 2005,] Coutinho, L., Sichman, J., et Boissier, O. (2005). Modeling organization in mas : A comparison of models.
- [Drogoul, 1993,] Drogoul, A. (1993). *De la simulation multi-agent à la résolution collective de problèmes*. PhD thesis, Université Paris VI.
- [Gutknecht et al., 2004,] Gutknecht, O., Ferber, J., et Michel, F. (2004). From agents to organizations : an organizational view of multi-agent systems. *Agent-Oriented Software Engineering IV*, 2935 :214–230. 23
- [Hübner et al., 2002,] Hübner, J., Sichman, J., et Boissier, O. (2002). Spécification structurelle, fonctionnelle et déontique d’organisations dans les systèmes multi-agents. *JFIADSMA’02*. 24
- [Lourdeaux, 2001,] Lourdeaux, D. (2001). *Réalité Virtuelle et Formation : Conception d’Environnements Virtuels Pédagogiques*. PhD thesis, Ecole des Mines de Paris.
- [Querrec, 2002,] Querrec, R. (2002). *Les systèmes multi-agents pour les environnements virtuels de formation. Application à la sécurité civile*. PhD thesis, ENIB, UBO. 4, 24, 26
- [Reenskaug, 1999,] Reenskaug, T. (1999). Uml collaboration semantics.
- [Richard, 2001,] Richard, N. (2001). *Description de comportements d’agents autonomes dans des mondes virtuels habités*. PhD thesis, ENST.
- [Vázquez-Salceda et al., 2004,] Vázquez-Salceda, J., Dignum, V., et Dignum, F. (2004). Organizing multiagent systems. Technical Report UU-CS-2004-015, Institute of Information and Computing Sciences, Utrecht University. 27

