Apprentissage de procédures en environnements virtuels

Ronan Querrec

Université de Bretagne Occidentale

— Habilitation à Diriger des Recherches —

Section informatique

Apprentissage de procédures en environnements virtuels

Ronan Querrec

Version provisoire

Rapporteurs

Marc	Cavazza	Professeur	IVE Lab Teesside(England)
Indira	Mouttapa Thouvenin	Maître de conférences (HDR)	Heudiasyc Compiègne
Michèle	Joab	Professeur	LIRMM Montpellier

Examinateurs

Pascal	Leroux	Professeur	LIUM Le Mans
Pierre	Chevaillier	Maître de conférences (HDR)	ENIB/CERV Brest

Directeur

Jacques	Tisseau	Professeur	ENIB/CERV Brest

Ronan QUERREC

11, rue Kerargroas
F-29200 Brest
tel : +33 (0)6 77 07 91 37
e-mail : querrec@enib.fr
url : http://www.enib.fr/~querrec

Laboratoire d'Informatique des Systèmes Complexes (LISyC)
Centre Européen de Réalité Virtuelle (CERV)

Technopôle Brest-Iroise
25, rue Claude Chappe
BP 38 F-29280 Plouzané
tel : +33 (0)2 98 05 89 50
url : http://www.cerv.fr
url : http://www.lisyc.univ-brest.fr

Table des matières

Chapitre 1

Introduction

Mes travaux de recherche s'inscrivent dans les domaines de l'ingénierie des connaissances et de la réalité virtuelle. Les Environnements Virtuels d'Apprentissage Humain (EVAH) en constituent le cadre applicatif.

Depuis ma thèse, mes activités de recherche portent sur la modélisation des connaissances pour les agents autonomes en environnement virtuel. L'objectif est de concevoir des modèles génériques autorisant l'explicitation des connaissances nécessaires à l'exécution des comportements de ces agents. L'objet de mes travaux est la simulation d'activités humaines à l'aide de la réalité virtuelle. Les environnements virtuels sont directement compréhensibles par les utilisateurs humains, c'est à dire que l'on n'utilise pas de métaphores pour représenter les concepts simulés. Plus précisément, les activités considérées sont les tâches procédurales et collaboratives. Grâce à ces modèles, les utilisateurs humains peuvent « *prendre la place* » des agents en temps réel dans un but de simulation ou de formation. C'est essentiellement ce dernier usage qui motive nos recherches. Dans ce contexte, les connaissances explicitées à l'aide des modèles proposés décrivent le modèle « *métier* » à transmettre mais également les connaissances spécifiques à la didactique ou à la pédagogie qui servent aux comportements d'agents autonomes jouant les rôles du formateur.

Durant mes travaux de thèse [Querrec et al., 2004], j'ai démontré, à l'instar d'autres travaux, que la réalité virtuelle possédait des propriétés intéressantes pour la formation. En effet, la réalité virtuelle permet de former du personnel sur du matériel sensible, coûteux ou difficilement disponible. Elle permet également de mieux contrôler les conditions de la situation simulée et introduit de nouvelles fonctionnalités pédagogiques telles que le rejeu ou la mise en relief d'éléments de l'environnement. Toutefois à l'époque, nous n'étions pas en mesure de maîtriser et d'évaluer réellement l'apprentissage. Nous pouvons considérer les modèles de l'époque comme de bonnes bases pour la conception d'environnements virtuels pour l'entraînement, mais pas pour l'apprentissage.

Partant de ce constat, mes travaux cherchent à inscrire l'apprentissage au cœur du modèle informatique. Ainsi la problématique générale de mes travaux peut s'exprimer sous la forme de ces deux questions :

– Comment évaluer l'efficacité de l'apprentissage ?

– Quels modèles pour concevoir une situation d'apprentissage en réalité virtuelle qui assure la réalisation des objectifs pédagogiques ?

Malgré l'existence de critères objectifs, tels que la réussite ou non d'un exercice, l'efficacité réelle de l'apprentissage ne peut s'évaluer que par l'expérimentation. Je m'inspire alors de celles menées par l'équipe de psychologie et de sciences de l'éducation du Centre Européen de Réalité Virtuelle. Une première étude sur l'apprentissage de procédures de maintenance (projet GVT [Gerbaud et al., 2008]) à l'aide de la réalité virtuelle m'inspire plus particulièrement. Lors de cette expérimentation, l'apprenant doit réaliser une procédure qu'il ne connait pas. Il effectue dix essais à la suite puis, après une période de six jours, il effectue à nouveau trois essais. Le temps total de consultation des instructions (ou plus précisément le temps d'exposition des instructions) ainsi que le temps total d'exécution de la procédure sont calculés. Les courbes présentées sur la figure 1.1 montre l'évolution de la moyenne du temps d'exposition et du temps d'exécution en fonction des essais.

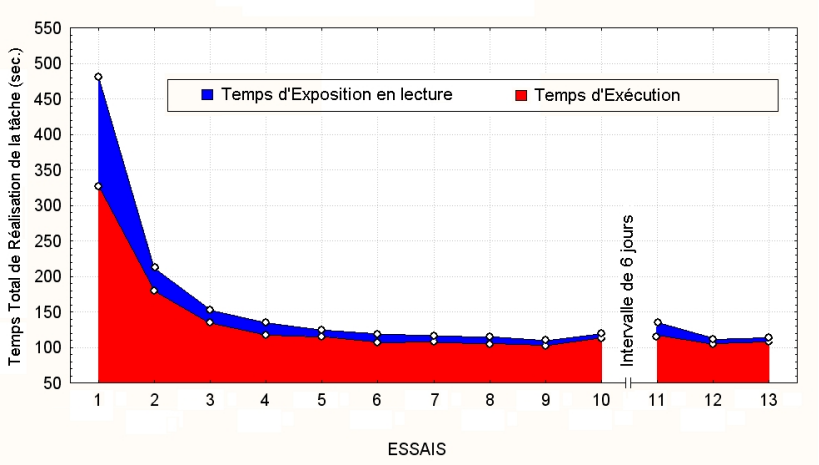


Figure 1.1 – Courbe d'apprentissage de procédures de maintenance en environnement virtuel (d'après [Gerbaud et al., 2008])

Cette première expérimentation montre que l'apprentissage en environnement virtuel suit la même tendance que la courbe d'apprentissage montrée par [Anderson, 1995] en environnement réel. Ainsi on peut observer que le temps de consultation des instructions et le temps d'exécution de la procédure par l'apprenant diminue avec le nombre d'essais. On considère que lorsque ces indicateurs n'évoluent plus l'apprenant a appris. Pour vérifier que les connaissances sont stockées en mémoire à long terme, la même expérience est reconduite quelques semaines plus tard ; si les temps sont équivalents aux temps réalisés lors des derniers essais, alors on considère que les connaissances sont bien acquises. Toutefois la figure 1.1 montre que l'acquisition de connaissances est coûteuse en temps dès les premiers essais. Mon objectif est de contrôler la situation d'apprentissage en environnement virtuel afin de réduire le nombre d'essais et/ou le temps des premiers essais.

Houssaye [Houssaye, 1988] décrit une situation d'apprentissage par un triangle dont les sommets sont le « savoir » à transmettre, le formateur et l'apprenant. Ce triangle est connu sous le nom de triangle didactique ou forme ternaire de l'apprentissage. Dans [Buche et al., 2005], nous reprenons ce triangle dans le contexte de la réalité virtuelle (figure 1.2). La situation

d'apprentissage se concrétise par une simulation en réalité virtuelle (un exercice de sapeurs-pompiers dans l'exemple de la figure 1.2). Cette situation simulée est construite à partir des trois sommets et de leurs relations. Notre objectif est de modéliser, de réifier et d'instrumenter les trois sommets et les trois côtés du triangle.

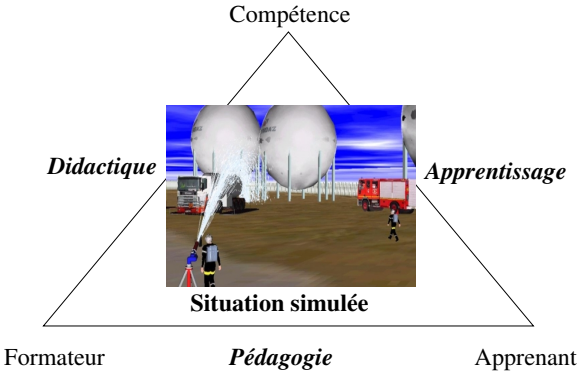


Figure 1.2 – Situation d'apprentissage en réalité virtuelle.

Sommet compétence : Dans notre cas, le savoir à transmettre est un ensemble de connaissances déclaratives (savoirs) et de connaissances procédurales (savoir-faire) mais également des comportements à respecter dans l'environnement (savoir-être). Il s'agit donc bien de compétences et ce type d'apprentissage peut clairement tirer partie d'une situation d'apprentissage en réalité virtuelle. Nous proposons de réifier ces compétences dans la situation d'apprentissage. Ainsi les participants (formateurs ou apprenants) pourront les manipuler en cours d'exercices. Ces connaissances portent sur la structure de l'environnement virtuel et les comportements des entités qui le peuplent, ainsi que sur les procédures que les apprenants doivent acquérir pour le manipuler. Notre proposition se place dans le domaine des *Intelligent Virtual Environments* (IVE) [Aylett and Luck, 2000] ou des Environnements Virtuels Informés (EVI) [Thouvenin, 2009]. Le principe est de donner aux objets, non seulement des propriétés géométriques pour qu'ils puisssent être visualisés et être manipulés « physiquement », mais aussi des propriétés sémantiques [Lugrin and Cavazza, 2007]. La compétence porte également sur les procédures à réaliser. Une procédure est considérée ici comme un agencement explicite *a priori* des interactions entre les participants et l'environnement. Les solutions que nous proposons ne peuvent pour l'instant pas s'appliquer aux tâches telles que l'élaboration de stratégies ou de tactiques collaboratives comme dans [Joab et al., 2002] par exemple. L'expert du domaine exprime alors la compétence à acquérir sous forme de diagrammes de classe et d'activités Uml à l'aide de Mascaret (figure 1.3). Cette proposition est détaillée dans le chapitre 2

Sommets apprenant et formateur : Les deux autres sommets représentent des utilisateurs humains (formateur et apprenant) pour lesquels nous n'avons pas à ce jour de modèles explicites.

Relation pédagogie : La première relation qui nous intéresse est la relation de pédagogie qui lie le formateur à l'apprenant. Le principe est de fournir des assistances au for-

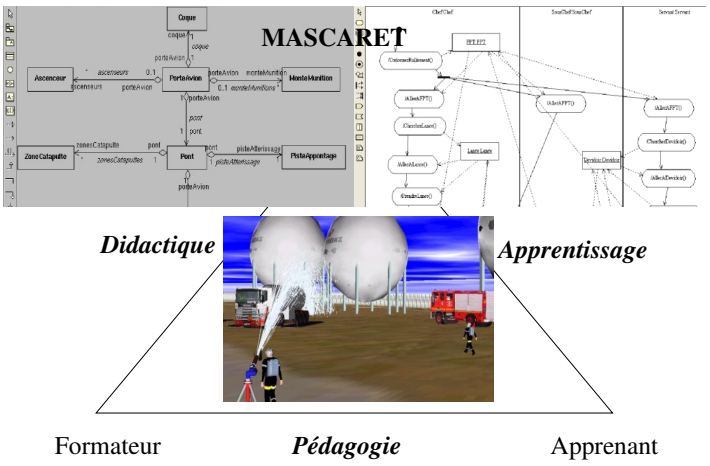


Figure 1.3 – Modélisation de la compétence dans la situation d'apprentissage à l'aide de Mascaret.

mateur afin d'adapter la situation pédagogique en temps réel. La difficulté est de proposer la meilleure assistance au moment opportun. Ces travaux se situent dans le domaine des *Intelligent Tutoring Systems* (ITS) [Wenger, 1987] tels que Steve [Rickel and Johnson, 1999] ou RobotTeach [Leroux, 1999]. Nous proposons un nouveau modèle d'ITS que nous nommons pegase (PEdagogical Generic and Adaptive SystEm) [Buche et al., 2009] (figure 1.4). L'originalité de Pegase par rapport aux autres ITS est qu'il est indépendant des domaines d'applications. Il s'agit d'un système d'agents dont la base de connaissances, la structure et les comportements sont exprimés à l'aide de Mascaret. Les connaissances de ces agents portent sur la compétence métier à faire acquérir et sur des stratégies pédagogiques. Nos travaux sur Pegase sont détaillés au cours du chapitre 3.

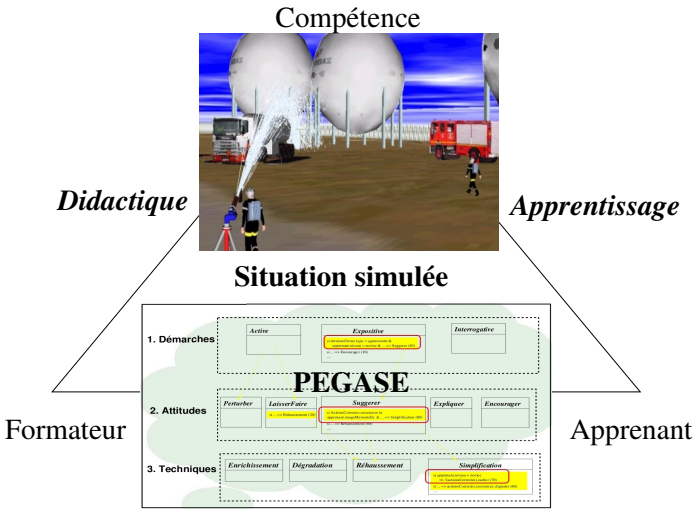


Figure 1.4 – Modélisation de la relation pédagogique à l'aide de Pegase.

Relation didactique : La seconde relation que nous étudions est la relation didactique entre le formateur et la compétence. Il s'agit de concevoir des outils destinés au formateur pour préparer un scénario qui guide l'apprenant vers les objectifs pédagogiques au sein de la situation simulée. Dans le domaine des EIAH, la norme IMS-LD fait référence pour décrire les scénarios pédagogiques. Comme pour la relation pédagogique, notre démarche est de fournir un modèle de scénario indépendant du domaine (tel que Formid [Guéraud et al., 2004]) et nous proposons pour cela le modèle poseidon (PedagOgical ScEnario for vIrtual and informeD envirONment) [Marion et al., 2009] (figure 1.5). Poseidon fournit une sémantique opérationnelle du scénario pédagogique dans le cadre de son exécution en environnement virtuel. L'indépendance au domaine est obtenue grâce à la réification de la compétence à l'aide de Mascaret. Ce modèle fait l'objet du chapitre 4.

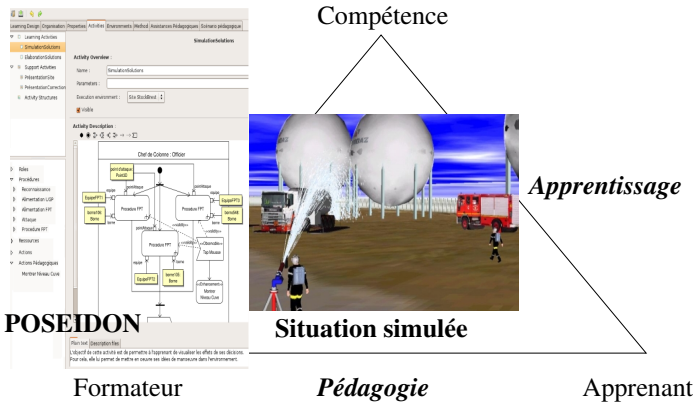


Figure 1.5 – Modélisation de la relation didactique à l'aide de Poseidon.

Relation apprentissage : La dernière relation est celle de l'apprentissage qui lie l'apprenant à la compétence. Nous considérons que les outils fournis au sein de l'environnement virtuel vont permettre cet apprentissage. Toutefois, nous considérons que des expérimentations sont nécessaires afin de valider l'efficacité des modèles proposés pour l'apprentissage. Pour cela nous proposons un outil de conception d'expérimentations destinées à la formation aux tâches procédurales : Tip-Exe [Ganier and Querrec, 2008] (figure 1.6). Tip-Exe à été utilisé pour concevoir une première étude comparant l'efficacité d'un apprentissage sur un dispositif virtuel et un apprentissage sur le dispositif réel. Cette expérimentation est simplement abordée dans le chapitre de conclusion de ce mémoire.

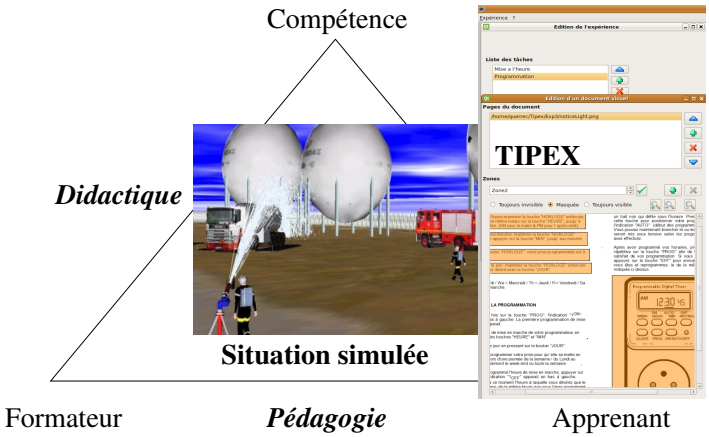


Figure 1.6 – Validation de l'apprentissage à l'aide de Tip-Exe.

Chapitre 2

Modélisation de la compétence en l'environnement virtuel

Dans ce chapitre, nous proposons une solution pour modéliser le premier sommet du triangle pédagogique tel que présenté en introduction : la compétence (figure 2.1). Ces travaux poursuivent mes travaux de thèse [Querrec, 2001] et ont donné lieu à l'encadrement de masters et post-doctorat [Chevaillier et al., 2009]. Ils se poursuivent également dans le cadre de la thèse de H. Thrin [Trinh et al., 2010] en co-encadrement avec Pierre Chevaillier.

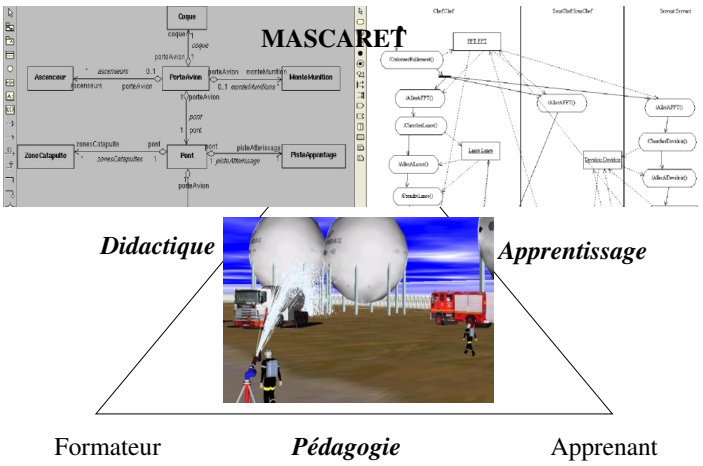


FIGURE 2.1 – Modélisation de la compétence dans la situation d'apprentissage.

L'objectif est de rendre la compétence explicite dans la simulation de manière à la rendre manipulable par les agents autonomes ou par les utilisateurs humains. Dans le cadre de nos travaux, la compétence est constituée de connaissances déclaratives et de connaissances procédurales. Ces dernières sont exécutables, il s'agit de comportements d'entités ou de procédures métiers par exemple. Ces deux types de connaissances doivent être explicites dans la simulation. Les connaissances procédurales servent, de plus, à décrire l'exécution de la simulation et constituent en conséquence la situation simulé au centre du triangle.

Nous présentons par la suite notre proposition : Mascaret. Mascaret regroupe plusieurs métamodèles Veha, Have, Behave et Mats. Veha (*Virtual Environment for Human Activities*) permet de donner une définition formelle des entités qui composent un environnement virtuel, de leur sémantique, des relations topologiques qui les relient, et de leurs comportements. Ces environnements sont destinés à héberger des activités humaines. Ces activités sont jouées par des personnages virtuels en collaboration avec des utilisateurs humains. Les métamodèles Have (*Human Activities in Virtual Environment*) et Behave (pour *Be* Have) décrivent les modèles d'agents et d'activités humaines que nous proposons. Ces trois métamodèles sont décrits par la suite. Le métamodèle Mats (*MultiAgent Tutoring Systems*) qui utilisent ces trois métamodèles à vocation pédagogique fait l'objet des deux chapitres suivant.

2.1 Positionnement

Mascaret est un métamodèle permettant de décrire l'environnement virtuel, non pas en tant qu'espace géométrique (tel qu'un graphe de scène) mais en fournissant une sémantique permettant à des agents artificiels ou humains de manipuler une représentation commune de l'environnement et d'y interagir conjointement pour atteindre leurs buts. Un agent – au sens large – a besoin de connaitre les objets qui compose l'environnement virtuel, leur accessibilité, leurs propriétés, leur comportement, et ainsi de savoir comment interagir avec eux. Les connaissances des agents s'appuient sur une représentation sémantique de l'environnement qui porte sur des concepts (des *classes*) ou des objets instanciés d'un domaine particulier, ayant ou non une représentation tangible dans l'environnement virtuel. Par exemple, un agent – ou un humain – peut savoir qu' *une vis se visse avec un tournevis et que la propriété forme de vis doit être identique à la propriété forme de tournevis*, ce qui peut déclencher alors un comportement de recherche d'un tournevis (localisation de l'objet, recherche d'un chemin pour y accéder puis préhension) et l'exécution de l'opération *visser* avec le tounevis t_i sur la vis v_j (déclenchement d'une opération de rotation et de translation de la vis, accompagné éventuellement d'un changement d'état). L'objectif de Mascaret est de rendre ces différents éléments de modélisation inspectables en ligne et directement utilisables pour simuler l'évolution de l'environnement virtuel. Trois types de connaissances doivent être exprimés dans l'environnement virtuel (on parle alors d'environnement virtuel informé et structuré) :

– les concepts du domaine de l'application,
– la structuration de l'environnement et les possibilités d'interaction avec les entités
– le comportement de ces dernières.

1. *Les concepts du domaine.* Il s'agit ici de la description sémantique des concepts du domaine d'activité abordé, de leur structure et de leurs relations. Classiquement ces connaissances sont représentées à l'aide d'un langage d'ontologie, tel que OWL [Patel-Schneider et al., 2004], ou de logiques descriptives [Baader et al., 2003]. Dans le domaine de la réalité virtuelle, le projet Matrics [Aubry et al., 2007] utilise les ontologies pour l'annotation de maquettes virtuelles dans le domaine de la conception mécanique. La spécificité de ce projet est que la maquette est conçue de manière collaborative. Le fait d'annoter la maquette par des données ayant un sens pour le système permet de proposer aux intervenants des assistances pour le suivi de la conception. Ces langages sont bien adaptés à la description des connaissances mais n'ont pas

de sémantique opérationnelle, ce qui les rend inaptes à la modélisation comportementale.

VR-WISE [Pellens et al., 2006] permet également de décrire l'environnement virtuel en explicitant les concepts du domaine aux niveaux classes et objets. VR-WISE s'appuie sur un langage graphique et permet de décrire les propriétés, les relations topologiques et les comportements des objets. Les concepts proposés dans ce modèle sont suffisamment formalisés pour être opérationnalisés dans le cadre de la simulation en réalité virtuelle. Toutefois cette description reste d'une expressivité limitée et ne permet pas l'exécution de comportement « métier » complexe.

2. *Les possibilités d'interaction et la structuration de l'environnement.* La spécificité des environnements virtuels est qu'ils sont composés d'objets physiques (virtuellement), en interaction les uns avec les autres et manipulés par les utilisateurs. La modélisation de ces interactions revêt plusieurs aspects : 1. comment les actions motrices des utilisateurs se traduisent en modifications de l'environnement virtuel (interfaçage sensorimoteur), 2. quelles parties des objets sont sensibles aux interactions, 3. quel est le « protocole » d'interaction (proximité des objets, enchaînement d'actions à réaliser, modalités de ces actions) 4. quels sont les comportements des participants à l'interaction. Dans cette section nous n'envisageons pas le premier problème qui est également un point clé de la réalité virtuelle et augmentée et un sujet d'étude en soi.

Le principe des *smart objects*, introduit par [Kallmann and Thalmann, 1998], est de définir des points remarquables sur la géométrie des objets et de les annoter d'informations indiquant le type d'interaction dans lequel ils peuvent intervenir ainsi que la manière dont cette interaction est effectuée, essentiellement des manipulations d'objets par un humanoïde virtuel. Un environnement ainsi informé permet à des agents d'exhiber des comportements contextualisés, donc variés, tout en ayant des comportements intrinsèquement simples [Doyle and Hayes-Roth, 1998, Doyle, 2002]. L'enrichissement de l'environnement par des annotations facilite l'écriture des scénarios d'agents tels que des personnages non joueurs [Gao et al., 2005]. Elles renforcent l'autonomie des agents et facilitent la planification de leurs actions [Gonçalves et al., 2001, Badawi and Donikian, 2004, Mathieu and Picault, 2005, Abaci et al., 2006, Sequeira et al., 2007]. Le fait d'informer l'environnement virtuel et d'en construire une structuration logique est également nécessaire pour les tâches de navigation ; la modélisation porte également sur des points informés et des zones [Lamarche and Donikian, 2004, Paris et al., 2006].

3. *Les comportements des entités.* Le dernier type de connaissance devant être explicités concerne le comportement des objets. Cette explicitation doit permettre aux agents d'analyser le comportement des objets, d'anticiper leur évolution, ainsi que de transmettre cette connaissance aux utilisateurs. Cette description doit également être exécutable afin d'assurer que l'objet évolue tel que son comportement a été décrit. Plusieurs langages ou modèles ont été proposés dans le domaine de l'animation comportementale, tels que HPTS [Donikian, 2001] et pourraient s'appliquer ici.

L'analyse de ces différents travaux montre qu'il existe des langages ou des modèles pour décrire chacune des connaissances intervenant dans la modélisation d'un environnement virtuel. Toutefois, ces langages n'explicitent pas l'ensemble des connaissances. Les langages d'ontologie ou les logiques descriptives, même s'ils explicitent des connaissances sur des comportements, n'en donnent pas une description exécutable. Les *smart objects* fournissent

des informations nécessaires à l'interaction ou la planification, mais sont très limités pour décrire la sémantique des concepts abstraits. Les modèles d'interaction dans les systèmes multi-agents n'intègrent pas les spécificités des environnements virtuels. Enfin les langages de description de comportement sont limités pour décrire les notions d'interactions ou de sémantique. Le problème majeur est l'impossibilité de faire formellement le lien entre les différentes facettes de la description de l'environnement virtuel.

Dans le cadre de la simulation d'activités humaines, l'environnement est peuplé d'agents autonomes. Ces agents possèdent des comportements qui leur permettent de prendre en compte l'environnement, de le modifier et d'interagir avec les autres agents et avec les utilisateurs humains. Il existe de nombreuses applications fondées sur les systèmes multi-agents. Ces applications couvrent un large champ, de l'écologie [Marilleau et al., 2008] aux relations humaines [Ochs et al., 2008]. De ces applications découlent de nombreux modèles d'agent, de système multi-agents et de plateformes telles que *Jade* [Rimassa, 2003], *Jack*[1] ou *Gaia* [Wooldridge et al., 2000]. Plusieurs travaux ont essayé de généraliser ces modèles et de proposer de méta-modèles d'agents ou de systèmes multi-agents [Hahn et al., 2009, Beydoun et al., 2009]. Les systèmes multi-agents étant utilisés pour simuler des activités humaines ou des systèmes automatiques (les agents représentant alors des logiciels ou des mécanismes) voire des systèmes physiques ou biologiques, il apparaît difficile de proposer des méta-modèles permettant de couvir l'ensemble de ces usages tout en gardant un langage efficace pour le modélisateur.

Dans la littérature, plusieurs propositions se fondent sur Uml. Ces modèles peuvent s'inspirer du méta-modèle Uml [Bauer et al., 2001] ou proposer des comportement d'agent interprétant de manière automatique des connaissances exprimées en Uml [Huget and Odell, 2004, Torres DaSilva et al., 2004, Ehrler and Cranefield, 2004]. Cependant les modèles proposés ne couvrent qu'une partie des types de comportement qui nous intéressent et surtout n'intègrent pas la modélisation de l'environnement, support de ces comportements.

De plus, *FIPA*[2] propose des modèles qui se veulent être une norme de fait à propos de la modélisation agent. Nous nous inspirons de cette norme et de ses implémentations pour proposer notre modèle. Il ne s'agit donc pas ici de proposer un modèle d'agent complètement novateur, mais plutôt une opérationnalisation de concepts existant dans d'autres modèles dans le cadre de la réalité virtuelle et de la modèlisation en Uml. Quels que soient les modèles les mêmes concepts sont présents, qu'ils soient plus ou moins formalisés. Ces concepts sont : l'agent, ses actions, ses comportements, ses moyens de communication et les organisations.

Dans le cadre de la simulation d'activités humaines collaboratives par les systèmes multi-agents, la notion de collaboration ou d'organisation structure les interactions entre les participants. L'organisation peut être décrite *a priori* ou déduite *a posteriori*. Elle peut être définie selon des règles strictes ou emmerger des comportements des agents. Dans le contexte de la simulation d'activités humaines dans un environnement virtuel, le modélisateur décrit explicitement la structure de l'organisation. Plusieurs modèles organisationnels existent [Montealegre Vazquez and Lopez y Lopez, 2007, Parunak and Odell, 2002, Omicini and Ricci, 2004, Ferber and Gutknecht, 2000], mais dans chacun le concept de groupe, organisation ou collaboration existe ainsi que le concept de rôle. En général, l'or-

1. http ://agent-software.com
2. http ://www.fipa.org

ganisation n'a pour but que de structurer les rôles, en revanche la formalisation du concept de rôle diffère selon les modèles. Il peut s'agir de la description de la responsabilité de l'agent jouant le rôle, le but, ou la liste des actions à réaliser par l'agent. Dans [Hubner et al., 2010], ce concept liste également les permissions ou les devoirs de l'agent. De la même manière que pour l'environnement ou les agents, l'organisation peut également être décrite au niveau de sa structure et au niveau des instances de cette structure. La structure organisationnelle définit les rôles qui la composent alors que l'entité organisationnelle affecte les rôles aux agents.

Dans le cadre de nos travaux, les activités humaines que nous abordons sont structurées sous forme d'un agencement *a priori* des actions réalisées par les participants. Il s'agit de procédures collaboratives car elle font intervenir plusieurs participants. La procédure est issue d'une expertise métier mais elle est décrite par un expert hors du contexte dans lequel elle peut s'executer. Le comportement des agents qui simuleront l'exécution de la procédure doit permettre son adaptation à l'environnement réel. De plus le formalisme utilisé pour décrire la procédure doit à la fois permettre son introspection en cours d'execution et servir au raisonnement d'agent de type pédagogique par exemple. Enfin le langage utilisé doit permettre de mêler l'éxecution par des agents autonomes et des utilisateurs humains. Pour un usage pédagogique il sera même important qu'un rôle puisse être inhibé ou rejoué par un agent en cours d'exécution de la procédure.

Plusieurs langages existent pour décrire les agencements d'actions. Dans ma thèse j'ai proposé un langage assez simple pour exprimer cet agencement. Toutefois ce langage, fondé sur des opérateurs de séquence et de parallélisme était justement trop simple et ne permettait pas de décrire l'ensemble des procédures complexes abordées. Un langage qui fait autorité pour répondre à ce problème est Hpts++ [Donikian, 2001] ou plus récemment son successeur Yalta. Dans le domaine des systèmes multi-agents, plusieurs auteurs ([Ehrler and Cranefield, 2004, Huget and Odell, 2004]) proposent de concevoir l'activité procédurale ou du moins la spécification du protocole d'interactions entre agents à l'aide des vues dynamiques d'Uml (séquences, activités, communications).

Nous proposons donc d'unifier l'ensemble des concepts abordés ici (environnement, agents, organisations et procédures) au sein d'un même métamodèle en utilisant un seul langage. L'objectif de Mascaret est de fournir un cadre de modélisation qui fasse la synthèse de ces travaux en les unifiant.

Dans ce chapitre, notre objectif est de démontrer qu'à l'aide de ce langage il est possible de couvrir les différentes facettes de la modélisation identifiées et de les rendre cohérentes.

Le métamodèle Mascaret s'appuie sur Uml 2.1 [Anonyme, 2007]. L'enjeu est d'éviter de redéfinir des concepts déjà éprouvés et de s'appuyer sur des standards en termes de langage de modélisation et de format d'échange de modèles (Xmi). Mascaret est un « profil » Uml qui est à la fois une restriction et une extension d'Uml dont tous les concepts ne sont ni nécessaires ni suffisants pour la modélisation envisagée ici. Le tableau 2.1 situe Mascaret dans le référentiel Mof qui distingue quatre « niveaux » de modélisation [Anonyme, 2006]. Comme Uml, il s'agit d'un métamodèle (M2) dont la sémantique est définie à l'aide de Mof (M3). S'appuyant sur ce métamodèle, Mascaret permet de construire un modèle d'environnement virtuel (M1) correspondant à un domaine métier particulier. Ce modèle sert ensuite à créer différents environnements virtuels (M0) exploités dans des applications de réalité virtuelle sans codage spécifique. A l'instar des langages d'ontologie, Mascaret

permet l'introspection des niveaux M0 et M1. La différence majeure avec ces langages est que Mascaret fournit une sémantique opérationnelle au travers ses modèles comportementaux.

M3	Mof (restriction d'Uml)				
M2	métamodèle Uml	métamodèle Mascaret			
M1	*user model* Uml	modèle EV_1		...	
M0	*user object*	EV_{1a}	EV_{1b}	...	...

EV : environnement virtuel

Table 2.1 – Couches de modélisation (Mi) : positionnement de Mascaret dans le référentiel Mof, parallèle avec Uml

2.2 Vue d'ensemble de Mascaret

Nous abordons ici la description de trois *packages* de Mascaret : Veha, Have et Behave. La figure 2.2 résume les concepts que nous abordons dans ce chapitre ainsi que les liens entre les métamodèles. Le métamodèle Veha définit les notions de concepts métiers, d'entités et de comportements de ces entités. Le métamodèle Behave aborde les notions d'agent, de comportement d'agents et d'organisation. Enfin le métamodèle Have définit le concepts d'humain virtuel.

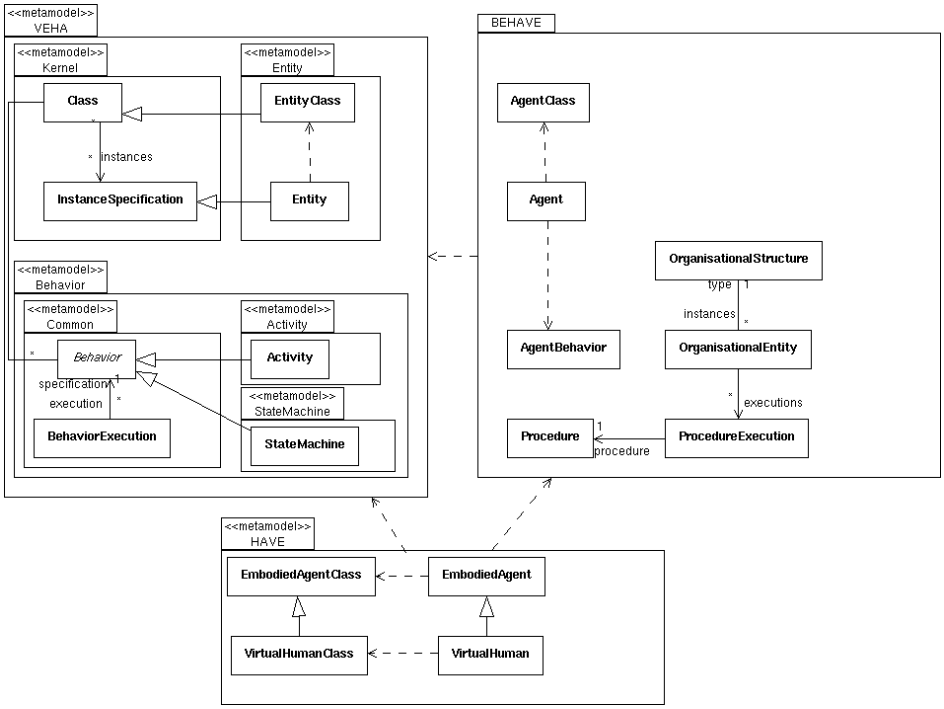


Figure 2.2 – Vue d'ensemble des métamodèles Mascaret.

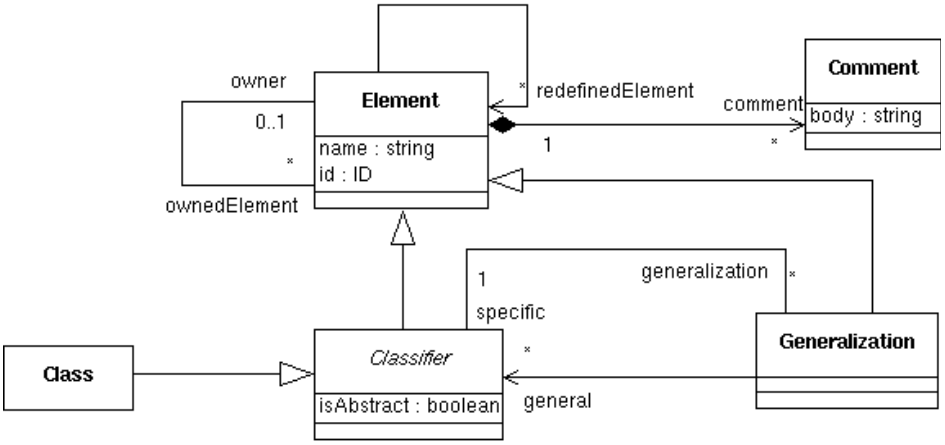


Figure 2.3 – Diagramme de classes du métamodèle Veha : la notion de classe

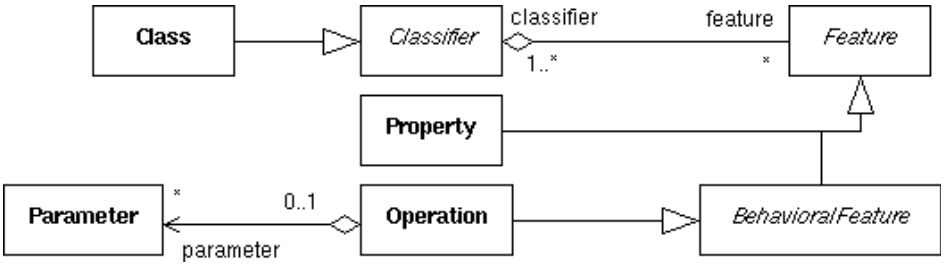


Figure 2.4 – Diagramme de classes du métamodèle Veha : propriétés d'un *Classifier*

2.3 Concepts du domaine

L'objectif ici est de formaliser la représentation des connaissances correspondant aux concepts manipulables par les utilisateurs dans la réalisation de leurs tâches. Comme dans toute description ontologique, on distingue les concepts et leurs instanciations et on s'attache à les nommer, à en donner les propriétés et à établir leurs relations. Ceci est supporté par le package `VEHA::Kernel`.

2.3.a. La notion de classe

Dans un environnement virtuel tous les objets n'ont pas nécessairement de représentation géométrique, c'est le cas lorsque l'ont fait référence à des concepts dans la description des actions des utilisateurs (p. ex. *mesurer la hauteur d'un objet*). Les connaissances sont organisées en classes qui constituent des types d'information. L'objectif du package `VEHA::Kernel` est de donner une sémantique aux différentes classes d'objets, qu'ils aient ou non une représentation tangible dans l'environnement virtuel. Les classes de ce métamodèle sont celles qui sont communes à Uml et Mof et qui supportent la description de métadonnées. Leur portée est suff-

isamment large pour couvrir la description des informations relatives aux objets composant un environnement virtuel. Cependant, nous n'avons conservé dans Veha que le sous-ensemble des classes strictement nécessaires à cette description.

À la racine du métamodèle, se trouve la classe `Element` (cf. 2.3). Elle permet de nommer chaque élément d'un modèle métier (attribut `name`). Une description textuelle peut également lui être attachée (`Comment`) ; ceci est utile pour fournir des explications à l'utilisateur sur la signification d'un objet. Les différents usages métiers des commentaires sont définis par des stéréotypes de `Comment` non représentés ici. Leur contenu n'est pas interprété. Un élément peut être redéfini. La sémantique de cette relation est précisée dans les sous-classes ; elle sert par exemple à exprimer des synonymies.

La notion de `Class` permet de donner une définition commune à des objets manipulés par les utilisateurs dans la réalisation de leurs tâches. Il se peut qu'aucun objet ne corresponde à cette définition car elle peut être incomplète ; la `Class` est alors abstraite. Dans Veha, `Class` est une sous-classe concrète de `Classifier` (cf. 2.3). Certaines classes d'autres packages héritent de `Classifier`, ce qui justifie que cette dernière soit distincte de `Class`. Les propriétés structurelles (`Property`) et comportementales (`BehavioralFeature`) des classes sont associées à `Classifier` *via* la classe `Feature` (cf. 2.4).

La classe `Generalization` permet de représenter des relations sémantiques entre `Classifiers` du type *sorte de* que l'on trouve dans les métamodèles d'ontologie. Elle modélise la mise en relation de deux classes, l'une étant d'une définition plus générale (`general:Classifier`) que l'autre (`specific:Classifier`).

Une manière de définir un concept ou un type d'objet est d'en énoncer les propriétés. Comme en Uml ou Owl, nous considérons qu'une propriété peut prendre plusieurs valeurs, ce qu'apporte `SetSpecification` en bornant le cardinal de l'ensemble (cf. 2.5). Nous avons ajouté la possibilité de définir un domaine de valeurs (`domain[0..1]:DomainSpecification`), éventuellement discrétisé. Ceci est utile en simulation pour contrôler les variations de certaines variables.

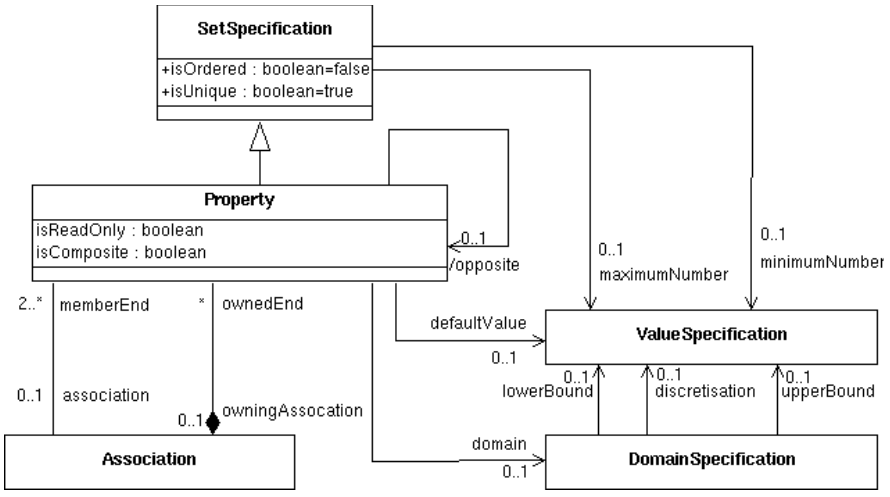


Figure 2.5 – Diagramme de classes du métamodèle Veha : propriétés d'une classe

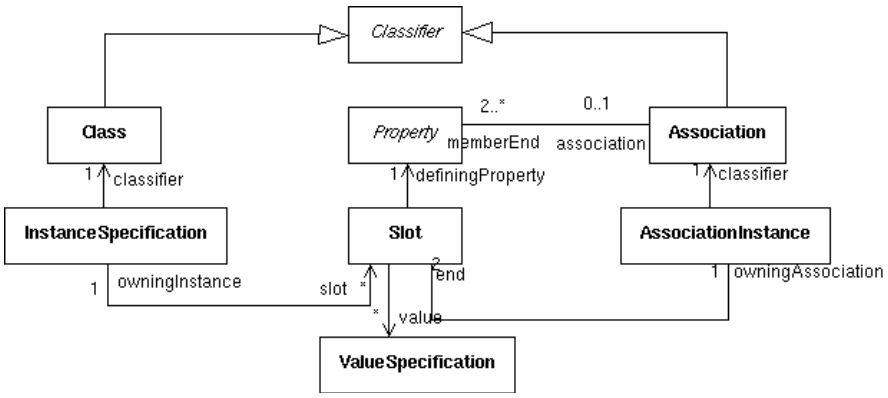


Figure 2.6 – Diagramme de classes du métamodèle Veha : instance d'une classe avec ses propriétés valuées et réalisation d'une association

La définition d'un type d'objet porte aussi sur des éléments comportementaux : opérations que l'utilisateur peut effectuer avec un objet (*on peut allumer la lumière*) ou comportements déclenchés par l'entité (*le chien aboie*). L'objectif ici est de donner une spécification du comportement de l'objet et non de la manière de le réaliser (méthode). La classe `Operation` permet d'exprimer ce qu'un objet ou un utilisateur peut effectuer sur un autre objet. Comme en Uml, il s'agit de la seule sous-classe concrète de la classe `BehavioralFeature` (cf. 2.4). Cette classe étant identique à celle d'Uml, ses attributs ainsi que les classes en association ne sont pas détaillés ici. La manière de modéliser les comportements associés à l'`Operation` est spécifiée à l'aide des modèles comportementaux (cf. section 2.5).

2.3.b. La notion d'instance

Après celle de `Class`, la notion d'`Instance`, synonyme d'objet, est le deuxième concept-clé de la métamodélisation, l'un permettant la définition – et l'introspection – de l'autre. Les classes `InstanceSpecification`, `Slot` et `AssociationInstance` représentent respectivement l'instanciation de `Class`, `Property` et `Association` (cf. 2.6). Le terme `InstanceSpecification` indique que l'on représente ici une entité du niveau M0 (cf. tableau 2.1), indépendamment de son implémentation.

2.4 Construction de l'environnement virtuel

Les classes du package `VEHA::Entity` permettent de modéliser les objets qui sont géométriquement représentés dans l'environnement virtuel et donc potentiellement manipulables par un utilisateur. Un environnement Veha est un ensemble d'objets situés ayant des représentations graphiques. Différentes propriétés topologiques et géométriques servent de support à la structuration de l'environnement virtuel et fournissent des informations sur les possibilités d'interaction. Les classes de ce package sont des spécialisations ou des stéréotypes des classes de `VEHA::Kernel`, ce qui établit le lien indispensable avec la modélisation des

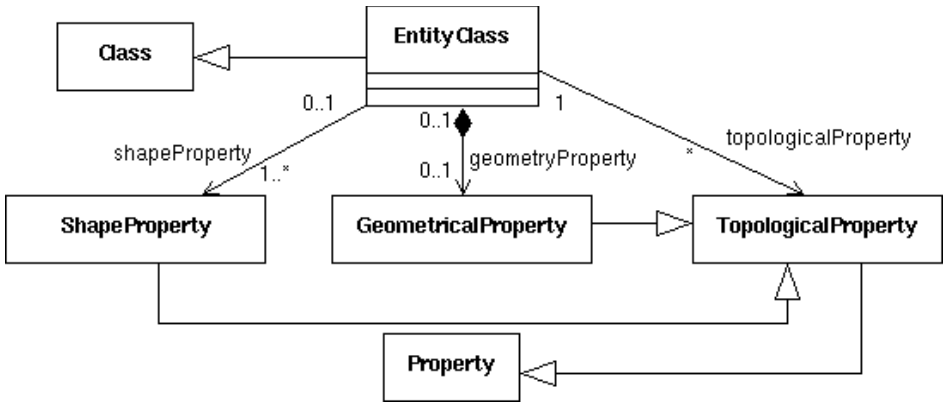


Figure 2.7 – Diagramme de classes du métamodèle Veha : entité de l'environnement virtuel

concepts du domaine.

2.4.a. Les entités de l'environnement virtuel

Les entités constituant l'environnement virtuel sont des objets qui ont toutes les propriétés des instances de `VEHA::Class` ainsi que des propriétés topologiques et géométriques (cf. 2.7). À un type d'entité, il est possible d'associer plusieurs propriétés topologiques dont le stéréotype permet de préciser l'usage. Pour des raisons de généricité nous n'avons pas contraint une entité à avoir des propriétés cinétiques (masse, vitesse, accélération). Elles peuvent être introduites dans un cadre métier particulier en tant qu'instances de la métaclasse `Property` ou comme stéréotypes de cette classe.

Le concept d'entité de Veha correspond aux notions de *smart object*, d'*artefact* ou d'*objet synoptique* (cf. section 2.1). En Veha, une entité de l'environnement virtuel est une instance de `EntityClass` : la classe `Entity` dérive de `Kernel::InstanceSpecification` (cf. 2.8). Chaque entité est localisée dans un repère global (`referentialPoint[1]:Point`) ; ce point peut être donné ou calculé. Un `Point` est une propriété topologique, classe dérivée de `InstanceSpecification`. Les valeurs des propriétés d'une entité sont définies par ses `slots`. Il s'agit donc des propriétés sémantiques (apportées par `InstanceSpecification`), morphologiques, géométriques et topologiques des objets de l'environnement virtuel.

Chaque entité a une ou plusieurs représentations graphiques dans l'environnement virtuel. Ces différentes formes peuvent servir à produire des animations ou à gérer des niveaux de détail (cf. 2.8). La forme (classe `ShapeSpecification`) d'une entité ne sert en Veha qu'au rendu graphique et à l'interaction « physique ». Elle n'a pas de sémantique et ne joue pas de rôle actif dans la simulation de la dynamique de l'environnement virtuel. Son rôle est similaire à celui d'une information textuelle dans un modèle de données. L'association d'une forme à une entité (`Entity.shape[1..*]:ShapeSpecification`) est le point d'intersection entre la modélisation sémantique de l'environnement virtuel (couverte par Veha) et la modélisation géométrique (utilisant des langages comme X3d) qui sont ainsi fortement découplées.

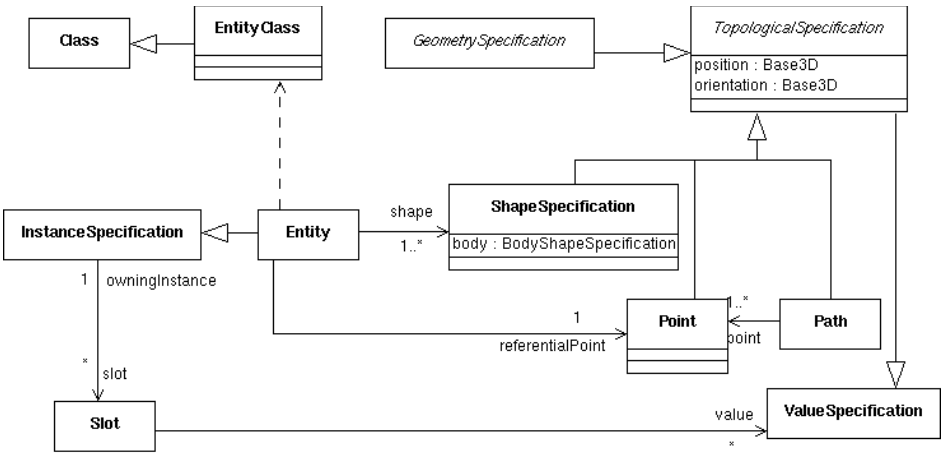


Figure 2.8 – Diagramme de classes du métamodèle Veha : propriétés d'une entité

2.4.b. La manipulation d'une entité

Les interactions que peuvent avoir les utilisateurs, ou les agents artificiels, avec une entité sont bien sûr localisées dans l'espace et plus précisément sur des régions des objets. Différentes parties d'un objet peuvent offrir des possibilités différentes d'interaction. Ces parties peuvent être des points (p. ex. pour guider un déplacement), des surfaces – simples ou complexes – (p. ex. pour saisir un objet), des volumes (p. ex. pour le calcul des collisions ou pour faciliter les interactions des utilisateurs en leur offrant des aides logiciels comportementales [Fuchs, 2006]). Ce concept généralise ceux de point informé [Kallmann and Thalmann, 1998] et de surface interactive [Badawi and Donikian, 2007] ; on peut le voir aussi comme une extension de la notion de *port* dans les architectures à base de composants [Briot et al., 2006].

En Veha ce concept a été fortement abstrait par la classe `GeometricalSpecification`. Les types d'usage qu'en fait le modélisateur sont identifiés à l'aide de TaggedValues. La figure 2.9 fait apparaître les trois familles de géométrie qui correspondent chacune à un moyen de la décrire. `PrimitiveGeometry` correspond aux types de base tels que Sphère, Cylindre... Ces types sont modélisés comme des stéréotypes. Ils sont en quelque sorte les équivalents des types de base Uml (`Boolean`, `Integer`...) : ils sont implémentables dans les plateformes de réalité virtuelle et permettent d'effectuer des opérations géométriques élémentaires dans un langage idoine. Une géométrie peut correspondre à une forme associée (ou non) à une entité (`ShapeGeometry`). Elle peut aussi être décrite dans un langage donné, ce qu'autorise l'attribut `body` de `SpecificGeometry`.

2.4.c. La structuration de l'environnement virtuel

L'objectif ici est de pouvoir exprimer des informations sur la localisation des objets, des agents artificiels et des utilisateurs, non pas par leurs coordonnées dans l'environnement virtuel, mais d'un point de vue logique qui fasse sens pour les utilisateurs. Ces connaissances

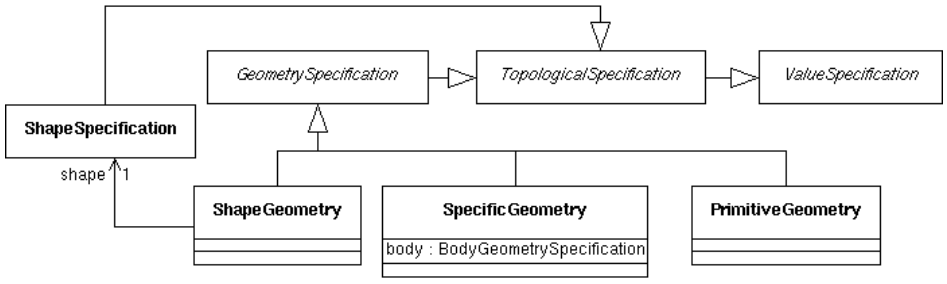


Figure 2.9 – Diagramme de classes du métamodèle Veha : géométries associées à une entité

sont utiles pour les tâches de déplacement d'objet, de navigation et de prise d'information. Dans ce contexte, à un certain niveau de description des actions des agents, on ne s'intéresse pas aux coordonnées précises des entités mais au fait que telle entité soit située dans telle zone. La réalisation de certaines tâches de manipulation et de navigation nécessite d'avoir des informations sur un chemin de façon à calculer une trajectoire, ce qu'apporte la classe `Path`.

La classe `TopologicalSpecification` supporte la notion de localisation (position et orientation) et permet d'exprimer des propriétés topologiques sur les éléments de l'environnement virtuel (cf. 2.7). Elle a comme sous-classes `GeometricalSpecification`, `ShapeSpecification` et `Point`. Ces trois éléments de modélisation permettent donc, dans un cadre métier particulier, de définir des points informés, des zones 2D ou 3D, des formes visibles.

2.5 Simulation comportementale d'une entité

Le package `VEHA::Behavior` a pour rôle de modéliser les comportements possibles des entités de l'environnement virtuel, l'objectif étant que le modèle soit interprétable en temps réel par un contrôleur comportemental et instrospectable en ligne. Comme pour les aspects structuraux, l'introspection porte à la fois sur le modèle comportemental (M1) et sur son « instanciation », c'est-à-dire son exécution (M0).

Sans cette formalisation du métamodèle, il n'est pas possible de définir la sémantique opérationnelle des interactions avec les entités de l'environnement virtuel. Sans lien avec les classes du métamodèle du domaine et de l'environnement, il ne serait pas possible de formaliser les effets des comportements en tant que modification des propriétés structurelles des entités. En l'absence d'un mécanisme d'introspection des modèles comportementaux des entités, les agents ne pourraient pas être dotés de règles d'inférence générales sur les effets potentiels de leurs actions, ce qui limiterait leurs capacités de planification ou nécessiterait un modèle *ad hoc*.

La modélisation comportementale repose sur les machines à états et le modèle d'activité d'Uml. Il s'agit donc de modèles réactifs asynchrones, bien adaptés à la modélisation de la

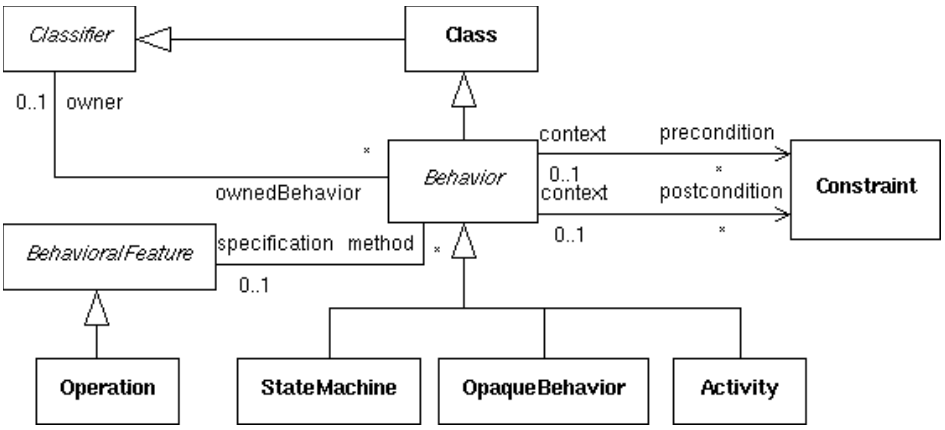


Figure 2.10 – Diagramme de classes du métamodèle Veha : spécification d'un comportement

réaction d'une entité à une interaction. Les modifications apportées par Veha par rapport à Uml sont mineures, c'est pourquoi ces modèles ne sont pas présentés exhaustivement. La différence majeure vient de l'introduction dans le métamodèle de la formalisation de l'exécution des comportements ; elle est justifiée par l'objectif d'introspection de la dynamique du modèle en cours d'exécution.

2.5.a. Les notions de comportement et exécution du comportement

Le package `Behavior::common` regroupe les classes qui définissent les notions de comportement (`Behavior`), d'événement (`Event`) et d'exécution de comportement (`BehaviorExecution`). Les entités Veha ont des comportements réactifs qui sont déclenchés sur occurrence d'un événement qui peut être produit par une action de l'utilisateur ou par une entité de l'environnement virtuel.

La classe `Behavior` (et ses spécialisations) fournit une spécification de la façon dont un objet d'une classe change (au sens large) au cours du temps (cf. 2.10). L'héritage de `Classifier` permet d'associer des propriétés à un `Behavior` et aussi d'autoriser l'héritage comportemental. Un `Behavior` a donc des propriétés structurelles (`Property`) et comportementales (`Operation`), ce qui introduit des possibilités de métacontrôle. Classiquement, on associe à un comportement des préconditions et postconditions.

Un comportement peut être la description de la méthode de réalisation d'une opération, plus généralement d'un `BehavioralFeature`, ce qui fait le lien avec la définition d'une classe (cf. 2.4). Un comportement peut être formalisé de différentes manières. Un comportement réactif est spécifié sous forme d'une machine à états Uml. Il peut aussi s'agir d'une fonction écrite dans un langage de programmation et invocable en ligne (`OpaqueBehavior`) ; la spécification est qualifiée d'opaque car non introspectable. Enfin il est possible d'utiliser le modèle d'activité d'Uml pour une formalisation procédurale introspectable.

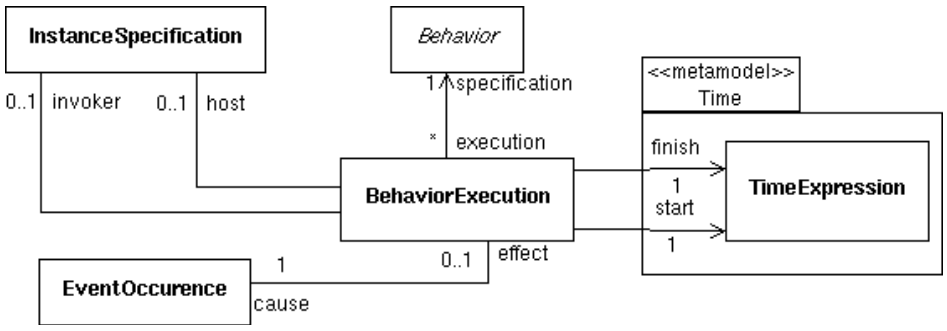


Figure 2.11 – Diagramme de classes du métamodèle Veha : exécution d'un comportement

Le comportement d'un `Classifier` peut donner lieu à plusieurs exécutions par une ou plusieurs `Entity`. (cf. 2.11). Chaque exécution est déclenchée par l'occurrence d'un événement. Elle peut être provoquée par un objet (`invoker[0..1]:InstanceSpecification`) et être exécutée par un `host[0..1]:InstanceSpecification`. Dans le cas d'un comportement autonome, l'`invoker` et le `host` sont la même entité. Dans le cas d'un comportement collectif, il n'y a pas nécessairement d'`invoker` et pas de `host`, mais des *participants*. Lorsque le comportement d'une entité est provoqué par une action de l'utilisateur, ce comportement est considéré comme l'`effect[0..1]:Behavior` d'un événement utilisateur (`Event`). Les démarrages et terminaisons d'exécution reçoivent une estampille temporelle exprimée sous forme d'une `TimeExpression`.

2.5.b. Les modèles comportementaux

Les modèles comportementaux de Veha sont de deux types : les machines à états qui sont plus centrées sur les interactions et les modèles d'activités qui décrivent la réalisation d'un comportement. Ces deux modélisations sont complémentaires.

Veha reprend le métamodèle des machines à états d'Uml (cf. 2.12) qui en définit deux interprétations : machines à états comportementale et de protocole. Nous conservons ici la première sémantique, l'objectif étant bien que la spécification comportementale soit exécutable. Le modèle comportemental est à événements discrets et est asynchrone : les événements produits sont placés dans un *pool* et traités un par un dans l'ordre de leur arrivée.

Les diagrammes d'activités Uml permettent de spécifier finement la manière dont un comportement est réalisé. Il s'agit de graphes qui contiennent, entre autres, des nœuds de type `ActionNode`. Ces derniers expriment l'exécution d'une action. La seule différence introduite dans Veha est l'association entre `ActionNode` et `BasicAction::Action` qui n'est pas explicite dans Uml car cette notion est redéfinie dans le modèle d'activité. Nous avons fait le choix de renforcer la cohérence entre les différents sous-modèles, contrairement à Uml qui se veut plus ouvert sur ce point.

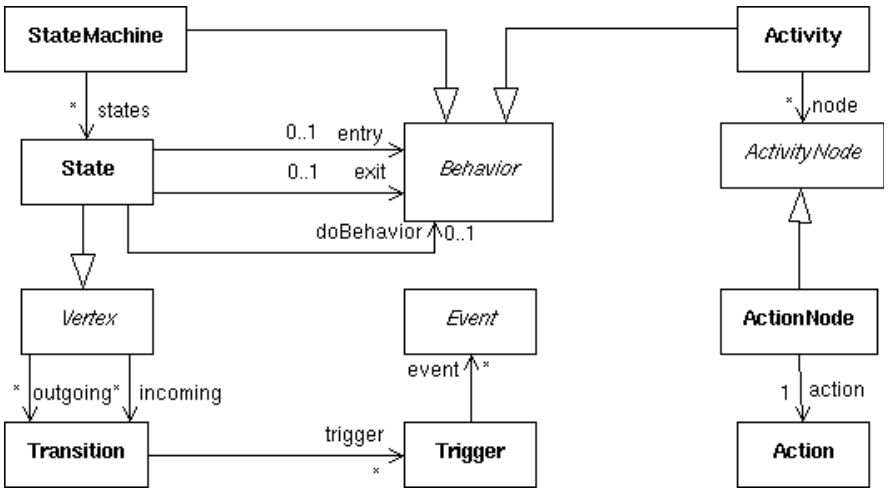


Figure 2.12 – Diagramme de classes du métamodèle Veha : machine à états comportementale

2.6 Le méta-modèle d'agent

2.6.a. L'agent

Le modèle d'agent que nous proposons s'inspire de la norme *FIPA* et de son implémentation dans *Jade*. Nous reprenons les concepts proposés et nous les implémentons en étendant Veha. Une vue globale du modèle est présentée sur la figure 2.13. Un agent réalise des comportements et peut communiquer avec d'autres agents par le biais de messages. Les agents sont hébergés par une plateforme, il n'est pas possible d'obtenir une référence (au sens informatique) sur un agent. Un agent est identifié par son nom et la plateforme qui l'héberge.

Dans le cadre de Veha un agent (`Agent`), est une instance (`VEHA::InstanceSpecification`) et possède un type (`AgentClass`) à l'instar de `VEHA::Entity` et `VEHA::EntityClass`. La classe `AgentClass` permet de décrire les différents types d'agents métiers. Ainsi il est possible de décrire les propriétés, états et actions des agents. Les actions sont en fait une spécialisation de `VEHA::Operation`. Ce modèle se veut suffissement générique pour adresser les différents types de métiers. Ceci signifie également que dans le modèle que nous proposons il n'est pas prévu de pouvoir dériver un autre type d'agents de la classe `Agent` ou `AgentClass`. Les spécificités que l'on souhaiterait faire apparaître en héritant de `Agent` se formulent dans notre modèle par des propriétes, opérations et comportements spécifiques (nouvelle instance de `AgentClass` décrite dans le modèle métier).

Un agent est identifié par un `AID`. Un `AID` contient le nom de l'agent, le nom de la plateforme sur laquelle il évolue ainsi que le port d'écoute de la plateforme. Ceci permet d'identifier les agents sur le réseau et de les faire communiquer. Les agents évoluent au sein d'une plateforme `AgentPlatform`. Plusieurs plateformes peuvent co-exister et interagir sur une même machine et interagir avec des plateformes distantes. La plateforme a également comme rôle de faciliter la communication entre les agents et d'informer sur les capacités des agents qu'elle héberge.

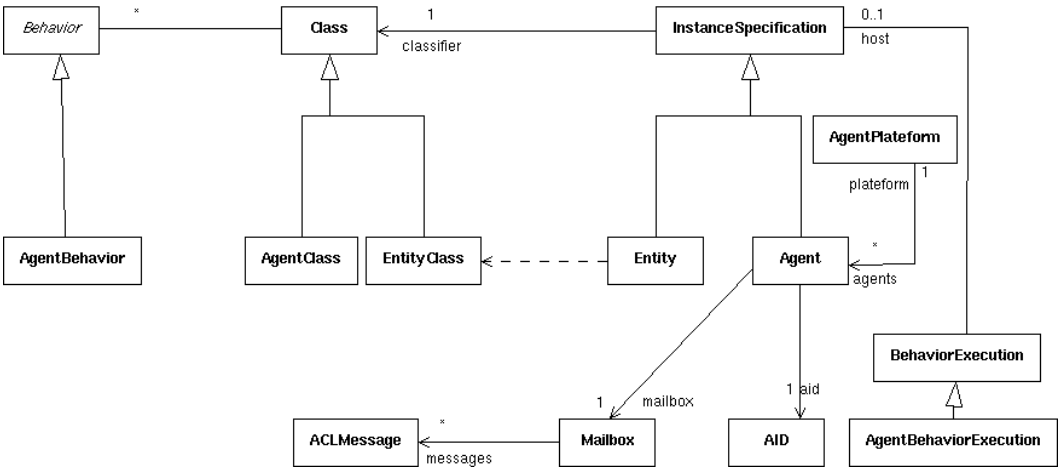


Figure 2.13 – Vue globale des classes agents, communications et comportements.

Dans le cadre des environnements virtuels, les agents considérés peuvent représenter des humains évoluant dans l'environnement. Ils sont donc incarnés, c'est à dire qu'ils possèdent une représentation géométrique (`Body`). Un `Body` possède toutes les propiétés d'`Entity`, mais pour un humain, il s'agit d'un corps polyarticulé. L'implémentation des corps polyarticulés est très dépendante de la plateforme de réalité virtuelle utilisée. Toutefois nous pensons pouvoir fournir une architecture abstraite suffisament générique pour supporter cette variété d'implémentation. Certaines normes telles que *HAnim* fournissent déjà des squelettes, nous pensons donc nous en inspirer. Le modèle proposé doit pouvoir gérer la cinématique directe (animation par positions-clés) et la cinématique inverse.

2.6.b. Les comportements

Pour décrire les comportements d'agents nous nous inspirons du modèle proposé dans *Jade* (figure 2.14). Un comportement réalise la méthode `action()` tant qu'une condition n'est pas remplie (évaluée par la méthode `done()`). Pour faciliter la conception des comportements *Jade* propose des `OneShotBehavior` executés une seule fois (`done()` renvoie toujours vrai) et des `CyclicBehavior`, exécutés en boucle (`done()` renvoie toujours faux). L'agent a alors un ensemble de comportement en cours d'exécution qui sont agencés en séquence. L'exécution des comportements (appel de la méthode `action`) est gérée par l'ordonnanceur proposé par Veha. Il est possible de proposer un agencement particulier. Ce type d'agencement est alors lui-même considéré comme un comportement qui manipule les autres comportements.

L'utilisateur fournit alors ses propres comportements en héritant de `OneShotBehavior` ou `CyclicBehavior` selon le cas et en surdéfinissant la méthode `action()`. Mascaret fournit déjà des comportements aux agents. Le comportement de communication est présenté plus loin dans cette section. Le comportement de suivi de procédures fait l'objet de la section suivante et les comportements pédagogiques celui des chapitres suivants. Leur fonctionnement n'a pas à être explicite lors de l'execution de la simulation, leur implémentation est donc réalisée sous forme d'`OpaqueBehavior` et non sous forme d'activité par exemple.

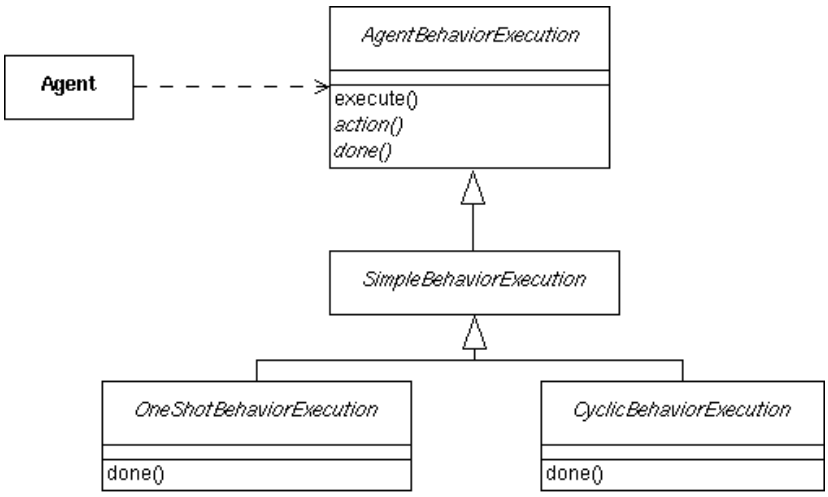


Figure 2.14 – Comportement d'agent.

Ces comportements ne sont pas spécifiques à un domaine métier particulier. Il est possible pour un programmeur de fournir ses propres comportements d'agent pour un domaine particulier en héritant des classes adéquates de Behave. Toutefois il pourrait être intéressant de pouvoir décrire un comportement métier spécifique à l'aide du même langage utilisé pour la modélisation de l'environnement. Pour l'instant ce n'est pas le cas, mais dans le cadre de Behave, il s'agirait de décrire une operation (*taggée* comme un comportement) et d'en fournir l'implémentation sous forme d'activité ou machine à états.

2.6.c. Les messages

Les agents communiquent entre eux par le biais des messages. Nous utilisons alors le modèle proposé par *FIPA* (figure 2.15) : *ACL* (Agent Communication Language [3]) . Un message a un but représenté par une performative. Le modèle *ACL* propose 23 performatives dont request qui permet à un agent de faire une requête à un autre. Il s'agit par exemple de demander la valeur d'une propriété ou de demander l'éxecution d'une action. Il existe également la performative inform qui au contraire permet à un agent d'informer un autre de la valeur d'une propriété ou de l'exécution d'une action en réponse à un request par exemple.

Les destinataires ou expéditeurs des messages sont identifiés par leur `AID`. Chaque message maintient un identifiant de conversation afin de faciliter le suivi du dialogue par les agents. Les messages sont exprimés dans un language et portent sur une onthologie. Plusieurs languages de contenu existent. Nous utilisons celui proposé par *FIPA* : *FIPA-SL*. Nous montrons dans la suite comment nous interprétons le contenu des messages exprimés en *FIPA-SL*.

3. Spécification FIPA SC00061

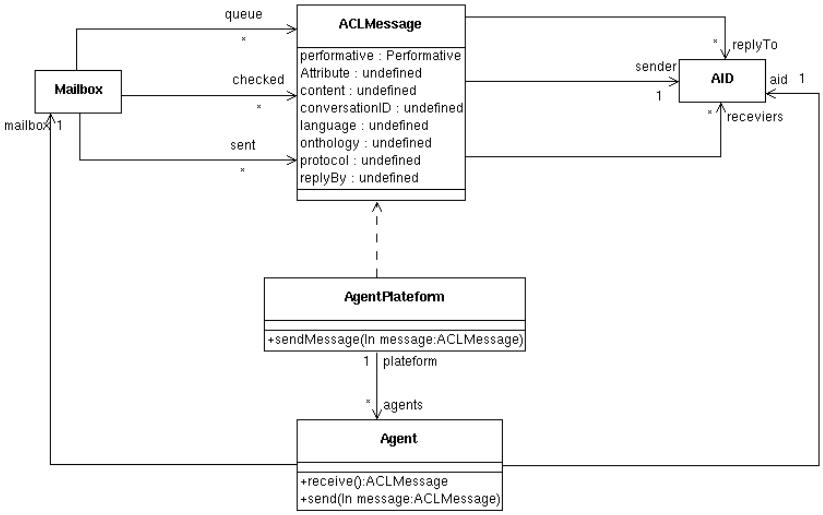


Figure 2.15 – Messages entre agents.

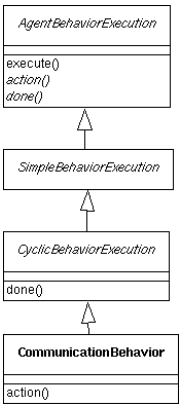


Figure 2.16 – Comportement de communication.

2.6.d. Le comportement de communication

Chaque agent de la plateforme a automatiquement un comportement de communication. Ce comportement de communication est un `cyclicBehavior`, (figure 2.16) il vérifie cycliquement si l'agent a reçu un nouveau message. L'objectif de ce comportement est d'analyser automatiquement le contenu du message grâce à la performative. Pour l'instant nous considérons que le contenu des messages est écrit en *FIPA-SL*. Nous ne gérons pour l'instant qu'une partie des performatives possibles (request et inform). Dans le langage *FIPA-SL* nous gérons tout ce qui se rapporte à la réalisation d'actions au sens large. Ainsi il est possible à un agent de demander la réalisation d'une action à un autre agent. Par exemple le message suivant est un message reçu par un agent lui demandant de réaliser l'action `ouvrirDeflecteur`.

```
ACLMessage : ((action (ia (ouvrirDeflecteur (deflecteur_lateral)))))
```

Par défaut le comportement de communication introspecte le contenu de la classe de l'agent. Si une opération portant le même nom que l'action demandée est trouvée alors l'agent exécute cette opération. L'exécution de l'opération peut nécessiter des paramètres. Ceux-ci sont affectés aux paramètres passés dans le message *ACL* en fonction de leur type. Si aucune opération n'est trouvée alors le comportement recherche une procédure portant ce nom dans les organisations dans lesquelles l'agent joue un rôle. Si elle existe alors l'agent déclenche l'exécution de cette procédure. Les paramètres représentent alors les ressources nécessaires à l'exécution de la procédure.

Si une action ou une procédure est réalisée sur l'occurence de ce message alors l'agent répond à l'expéditeur du message un message de performative AGREE. Si aucune opération et aucune procédure n'est trouvée ou n'est réalisable (en fonction de l'état de l'environnement) alors l'agent répond par un message de performative NOT UNDERSTOOD. Ce comportement est normalisé par la norme *FIPA*. L'algorithme de la figure 2.17 illustre le fonctionnement de ce comportement.

L'objectif final est de prendre en compte les autres aspects du langage de contenu de Fipa-sl. Ces autres aspects portent essentiellement sur les valeurs de propriétés. Le problème n'est pas d'analyser le contenu du message, mais de savoir quoi faire de manière générique. La norme *FIPA* propose que les agents aient un base de connaissances, mais sans donner de formalisme pour cette base de connaissances. En perspective de nos travaux, nous proposons que la base de connaissances de l'agent soit en fait un sous-ensemble de l'application (au sens de la classe `Application`), c'est à dire un sous-ensemble des valeurs du modèle métier et un sous-ensemble de l'environnement. Ainsi il serait possible au comportement de communication de lire ou écrire dans cette base de connaissances en fonction des messages reçus en *FIPA-SL* de manière générique. Les comportements spécifiquement développés pour l'application n'auraientt alors qu'a manipuler cette base de connaissances. Plusieurs problèmes subsistent toutefois. Comment déterminer les informations dont l'agent dispose en début de simulation ? Tous les comportements peuvent-ils réellement s'écrire en ces termes ? Comment synchroniser la modification de la base de connaissances et la réaction des comportements intéressés ?

2.7 Modélisation de la collaboration

Tenant compte du modèle de collaboration d'Uml et des différents modèles d'organisation d'agent, nous proposons le modèle résumé sur la figure 2.18.

2.7.a. Structure organisationelle

La structure organisationnelle (`OrganisationalStructure`) décrit la manière dont les organisations concrètes sont instanciées. Ce concept se rapproche de celui de *Collaboration* dans

```
1  début
2      msg ⟵ agt.receive();
3      si msg.performative == REQUEST alors
4          res ⟵ parseFIPASL(msg.content);
5          si res.succes alors
6              action ⟵ res.action;
7              classifieragt.classifier;
8              si action ∈ classifier.operations alors
                   //Faire l'action
9                  ACLMessage m(AGREE);
10                 m.content ⟵ msg.content;
11                 m.dest ⟵ msg.sender;
12                 agt.send(m);
13             finsi
14             sinon
15                 si action ∈ agt.equipes.procedures alors
                       //Declencher la procedure
16                     ACLMessage m(AGREE);
17                     m.content ⟵ msg.content;
18                     m.dest ⟵ msg.sender;
19                     agt.send(m);
20                 finsi
21                 sinon
22                     ACLMessage m(NOTUNDERSTOOD);
23                     m.content ⟵ msg.content;
24                     m.dest ⟵ msg.sender;
25                     agt.send(m);
26                 finsi
27             finsi
28         finsi
29     finsi
30     sinon si msg.performative == INFORM alors
           //Informe le comportement procedural qu'un agent a fait une
             action
31     finsi
32 fin
```

Figure 2.17 – Algorithme du comportement de communication.

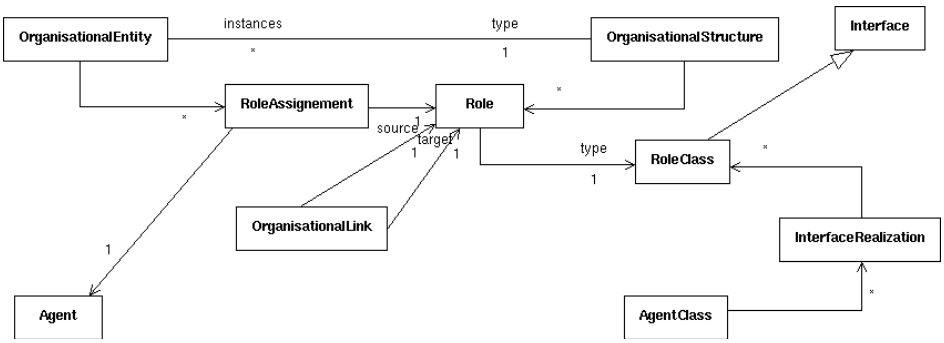


Figure 2.18 – Vue globale de la notion d'organisation.

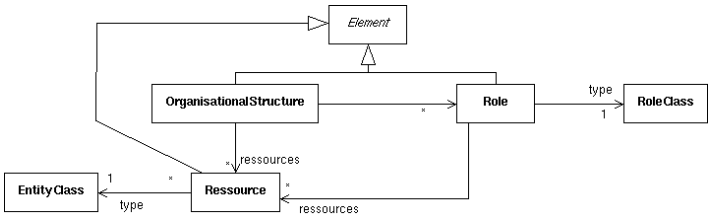


Figure 2.19 – Structure organisationnelle.

Uml. L'élément structurant important est le concept de rôle (`Role`). La structure organisationnelle et les rôles qui la constituent sont essentiellement caractérisés par un nom, c'est pourquoi ils héritent de `Element`. La définition d'un rôle diffère selon les modèles. Dans certains modèles un rôle définit un but qui est affecté à l'agent qui joue le rôle, il peut également s'agir de comportements à réaliser, etc. Cette dernière est la plus opérationnalisable et s'approche le mieux de ce dont nous avons besoin et du modèle de collaboration d'Uml. Nous représentons cela par la notion de procédure que nous exposons plus loin et par des contraintes (liste d'actions) sur les rôles que nous exprimons par le concept de `RoleClass`.

Les organisations et les rôles peuvent avoir la responsabilité de ressources `Ressource`. Ceci permet de faire le lien avec l'environnement dans lequel évolue l'organisation. Le concept de ressource peut être décrit indépendemment de l'objet concret qui jouera cette ressource lors de la simulation. Une ressource est définie par son nom et par la classe d'entité qui peut jouer le rôle de cette ressource.

La figure 2.20 illustre la manière dont l'expert du métier décrit une structure organisationnelle en utilisant un modeleur Uml, dans le même modèle que celui où il a décrit son environnement. La classe d'agent `Personnel` est spécialisée `VirtualHuman` (sorte d'`AgentClass`) et le type de rôle `Personnel` du *package* `Organisation` est spécialisée `RoleClass`. Une collaboration nommée `EquipeAviation` est créée, elle sera interprété par Mascaret comme une structure organisationnelle. Cette collaboration fait intervenir deux rôles `PEH` et `IA`, tous deux devant être joués par des agents sachant réaliser les actions proposées dans le type de rôle `Personnel`.

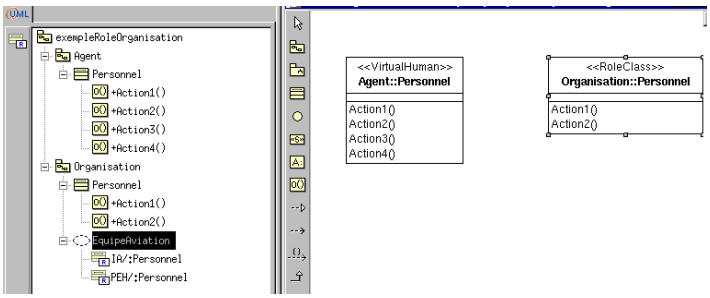


Figure 2.20 – Exemple de structure organisationelle.

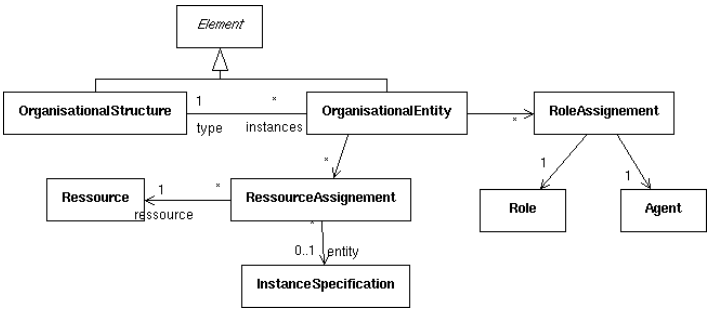


Figure 2.21 – Entité organisationnelle.

2.7.b. Entité organisationelle

Une entité organisationnelle `OrganisationEntity` est une instance d'une structure organisationnelle. Il s'agit de la même démarche que le principe de *CollaborationUse* dans Uml. Il s'agit essentiellement d'affecter les rôles aux agents (`RoleAssigment`) et les ressources aux entités (`RessourceAssignement`). Il existe plusieurs entités organisationnelles pour une même structure organisationnelle. Les rôles et ressources peuvent être fixés *a priori* mais également être dynamiques. Ceci pose alors un problème sur le principe même d'organisation puisque à un instant un rôle dans une organisation peut être joué par un agent et un instant plus tard être joué par un autre agent, cependant dans Mascaret, il s'agit de la même organisation. De la même manière, un rôle est joué par un et un seul agent à la fois. Or, dans plusieurs autres modèles, il est possible de faire jouer le même rôle par plusieurs agents. Pour que le comportement des agents soit cohérent, cela nécessite alors d'être capable de décrire des règles organisationnelles qui régissent l'activité des agents au sein du rôle, ce que Mascaret ne propose pas pour l'instant.

Dans Mascaret, un fichier de type *XML* permet d'instancier les entités de l'environnement, les agents et également les entités organisationnelles. Il s'agit alors de créer une entité organisationnelle et spécifiant la structure qui lui correspond et d'effectuer l'affectation des rôles et des ressources respectivement aux agents et aux entités. Bien sûr ceci peut se réaliser en dynamique lors de la simulation et également être modifié. Il aurait également été possible d'utiliser la définition des `CollaborationUse` du modeleur Uml pour faire cette

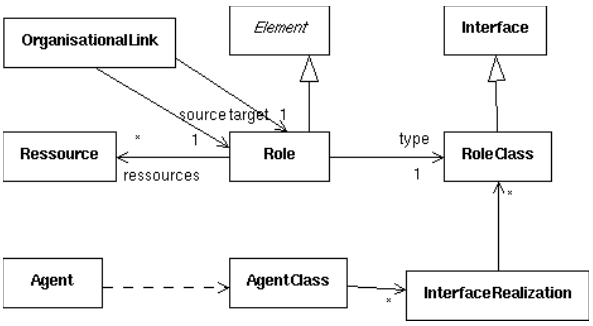


Figure 2.22 – Affectation des agents aux rôles.

instanciation, toutefois il semble raisonnable de séparer la description du modèle de celle de son environnement. En effet un même modèle peut servir à décrire plusieurs environnement et un modeleur Uml n'est pas l'outil idéal pour positionner des objets ou des agents dans un espace 3D.

2.7.c. Affectation

Dans la section précédente nous parlions de contraintes sur les rôles. Le principe est de donner des contraintes de manière à s'assurer que l'agent qui joue un rôle en a les compétences. Nous représentons ce principe par la notion de `RoleClass`. Une `RoleClass` est une sorte d'`Interface` (au même sens qu'Uml). Ceci permet de décrire l'ensemble des actions que doit savoir faire l'agent sans décrire la manière dont celles-ci sont réellement réalisées. Un agent (`Agent`) a une `AgentClass` qui décrit sa structure, ses états et les actions qu'il sait réaliser. L'`AgentClass` décrit (`InterfaceRealization`) alors également la manière dont elle implémente une `RoleClass` (qui hérite d'`Interface`). Il s'agit sensiblement du même principe que dans Uml. Ceci permet de proposer un mécanisme très riche sur la manière dont un service de l'interface est réalisé. Par exemple une action de l'interface peut être réalisée dans une `AgentClass` par une activité complexe agençant des actions de l'`AgentClass`. En pratique, comme dans beaucoup de langages informatiques, dans Mascaret il s'agit d'une correspondance entre les noms des actions de la `RoleClass` et de l'`AgentClass`.

Nous pouvons également noter que les rôles sont liés entre eux par des `OrganisationalLink`. Ceci correspond au modèle Moise+.

2.8 Modélisation des tâches procédurales

Nous décrivons dans cette section comment nous interprétons les diagrammes d'activités d'Uml pour l'exécution de procédures puis nous décrivons le comportement procédural que nous fournissons aux agents autonomes.

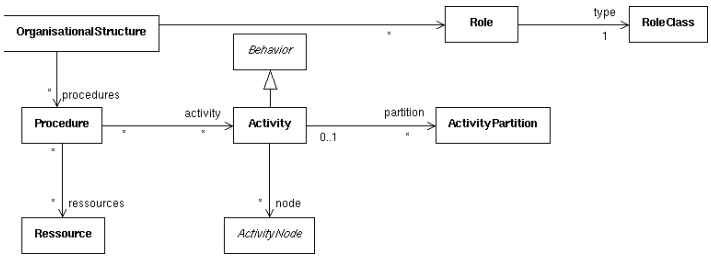


Figure 2.23 – Modèle de procédure.

2.8.a. Description de la procédure

Les procédures dans Mascaret sont de la responsabilité des organisations. Elles sont décrites dans le cadre des structures organisationnelles et éxecutées par des entités organisationnelles. Le modèle de procédure que nous proposons s'appuie sur le modèle des activtés du *package* Veha qui se fonde lui même sur les activités d'Uml. Nous ne décrivons donc pas ici les détails de ce modèle, mais nous décrivons le liens entre les organisations, les procédures, les activités et les rôles. La figure 2.23 explicite ces liens.

La procédure maintient des buts et préconditions (internes et externes), comme les actions qui la composent. Nous nous appuyons aussi pour cela sur les contraintes d'Uml.

Les `Partitions` des activités Uml sont interprétées comme référençant les rôles de l'organisation comme les `ObjectNode` référencent les ressources. Les actions utilisées pour décrire la procédure sont alors décrites dans les `RoleClass`.

La procédure doit pouvoir se faire par un agent ou par un humain de manière transparente d'ou la séparation de ce qu'il faut faire de comment cela est effectivement réalisé. Les `RoleClass` jouent également ce rôle.

Les procédures sont executées dans le cadre des entités organisationnelles. Une procédure étant décrite par une activité qui est un `Behavior`, l'exécution d'une procédure se fait par des `ProcédureExecution` qui sont des sortes de `BehaviorExecution`. Ceci permet de les horodater par exemple, mais le `ProcedureExecution` a aussi pour objectif de vérifier que les conditions de réalisation de la procédure sont remplies (affectations des ressources et des rôles par exemple). La figure 2.24 montre ce principe.

Si nous reprenons l'exemple de l'organisation de la figure 2.20 l'expert peut alors créer une procédure pour cette organisation. La figure 2.25 montre un exemple de description de ces procédures.

2.8.b. Comportement procédural

Selon le type de simulation, le concepteur peut doter les agents d'un comportement procédural. Ce comportement est déjà implémenté dans Mascaret. Nous décrivons ici son

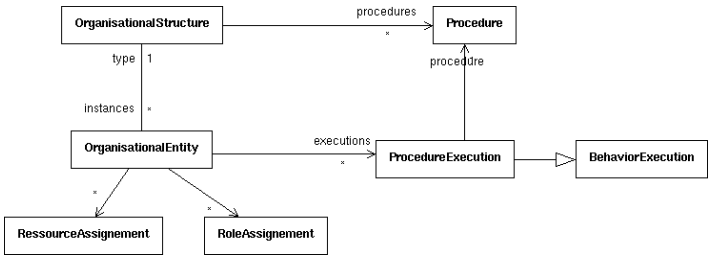


Figure 2.24 – Exécution d'une procédure par une organisation.

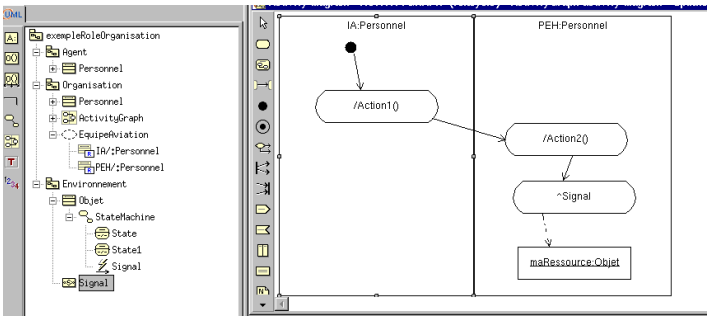


Figure 2.25 – Exemple de procédure

fonctionnement. Le comportement procédural doit prendre en compte plusieurs critères.

Tout d'abord, la procédure est collaborative. Il existe donc des agencements d'actions qui font intervenir plusieurs rôles, par exemple l'agent jouant le rôle R1 réalise l'action A1 après que l'agent jouant le rôle R2 a réalisé l'action A2. Un agent a alors besoin de savoir ce que les autres participants sont en train de faire (début d'action, action en cours, fin d'action, résultat d'action). Dans la réalité, ceci peut se faire par observation et reconnaissance d'action. Nous n'avons pas implémenté ce type de fonctionnalité. Les participants peuvent être distribués sur plusieurs machines et être joués par des agents autonomes ou par des utilisateurs humains. Il n'est donc pas possible d'avoir un mécanisme centralisé vers lequel chaque participant aurait un *pointeur*. Nous avons donc pris le parti que chaque participant possède sa propre connaissance de l'évolution de la procédure et que dès qu'un agent débute ou termine une action il envoi le message approprié à chaque participant.

De plus, un rôle peut être inhibé ou *re-simulé* en ligne. Cela veut dire que au cours de la simulation, dans un but pédagogique ou autre, un utilisateur humain peut prendre le contrôle d'un rôle qui était joué par un agent autonome. Toutefois, ce même utilisateur peut rendre la main à l'agent autonome plus tard. Il faut alors que celui-ci ait la connaissance de l'évolution de la procédure. Ainsi même si le rôle est joué par un utilisateur humain, le comportement procédural est executé. Seul le choix de l'action à exécuter est laissé à l'utilisateur. L'envoi et la reception de messages liés à l'exécution de la procédure sont fait de manière automatique.

La procédure est décrite par un diagramme d'activité Uml. La manière d'interpréter les noeuds de ce diagramme est également un choix fort. Le diagramme d'activité agence des `ActionNode` qui peuvent être la réalisation d'une action, l'envoi d'un signal à un objet (ici

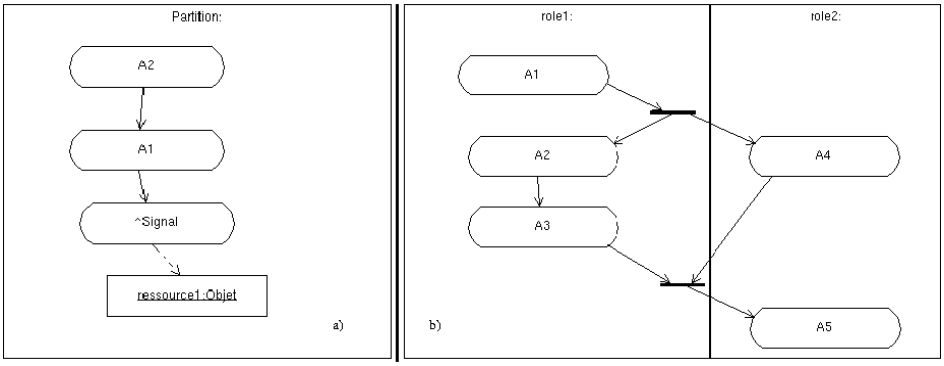


Figure 2.26 – Interprétation des `ControlNode`.

considéré comme une ressource)... Le choix le plus fort intervient lors de l'interprétation des `ControlNode`. Comme il s'agit en priorité de simuler les actions humaines, nous avons fait le choix d'occulter la contrainte du synchronisme. Ainsi lorsqu'une action A1 doit être réalisée après une action A2 (figure 2.26 (a)), A1 sera réalisée après A2 mais sans savoir si elle sera réalisée immédiatement aprés la fin de A2 ou plus tard. De même il n'est pas possible de dire que c'est l'exécution de A1 qui provoque la fin de A2 par exemple. Dans le cadre de la coordination entre plusieurs rôles, la même remarque s'applique. Ainsi selon l'exemple en figure 2.26 (b), les actions A2 et A4 débuteront après A1, mais sans avoir si elle débuteront en même temps. De même A5 débutera après que l'agent jouant le role1 ait finit A3 et que l'agent jouant le rôle 2 aura finit A4 mais sans qu'il y ait de synchronisation dure entre ces actions. Avec ce mécanisme il est assez difficile de décrire des actions du genre « porter une table à 2 personnes ». De la même manière si l'agent doit executer les actions A1, A2 et A3 en séquence, mais quelque soit l'ordre, il est difficile de l'exprimer avec les noeuds de contrôle prévus dans les activités d'Uml. Or plusieurs langages ont prévu ces alternatives. Cependant, comme nous en avons déjà parlé dans d'autres cas, Uml prévoit la possibilité d'étendre la sémantique de ces opérateurs. Ainsi pour un usage particulier il est possible de fournir des *tags* particulier sur un `ControlNode` pour en préciser la sémantique (contrainte de synchronisation, d'agencement indifférents, de probabilité...).

Dans le cadre des différents domaines d'application que nous avons pu aborder (procédures de sapeurs-pompiers, gestion aviation sur porte-avions...) il apparaît qu'il existe une certaine priorité entre les procédures. Plus précicément, afin d'optimiser l'organisation humaine du système simulé, un agent peut jouer plusieurs rôles dans plusieurs organisations et ainsi participer à l'exécution de plusieurs procédures. Ainsi il peut arriver qu'un agent engagé dans la réalisation d'une procédure n'ait rien à faire à un instant donné en fonction de l'evolution de l'éxécution de cette procédure. Il peut alors se permettre de réaliser une autre action d'une autre procédure. Cependant, il existe des situations qui font que même si l'agent n'a aucune action à faire à un instant dans la procédure, le fait qu'il fasse une action pour une autre, gène l'exécution de la première (actions nécessitant un déplacement par exemple...). Ainsi, nous avons intégré le fait qu'un procédure puisse être interruptible ou non. L'algorithme suivant donne le pseudo-code du comportement procédural.

```
1  début
2      pour chaque procedure proc dans procedures en cours faire
           //Sur la procedure il y a des pointeurs qui definisse l'etat de
             son execution
3          pour chaque pointeur pt dans pointeurs sur la procedure proc faire
4              noeud ⟵ proc.getNode(pt);
               //Suis-je concerne par ce pointeur (de mon role?)
5              si noeud.partition ∈ roles alors
                   //Le noeud est de ma responsabilite
6                  si noeud est de type ControlNode alors
7                      si noeud est de type JoinNode alors
                           //Si tout les noeuds avant le join sont finis alors
                             activer les noeuds apres
8                      finsi
                       //Autre type de ControlNode...
9                  finsi
10                 sinon
11                     etat ⟵ noeud.etat;
12                     si etat == ENCOURS alors
13                         si monRole.active alors
14                             si noeud.fini alors
15                                 noeud.etat ⟵ FIN;
16                             finsi
17                         finsi
18                     finsi
19                     si etat == FIN alors
                           //Mettre le pointeur sur les noeuds après
20                         ACLMessage m(INFORM);
21                         m.setSLContent((action(moi(noeud.name))));
22                         org.broadcast(m);
23                     finsi
24                     si etat == AFAIRE alors
25                         si pas d'action a faire dans une autre procedure prioritaire
                           alors
26                             si monRole.active alors
                                   //Faire l'action noeud
27                                 noeud.etat ⟵ ENCOURS;
28                             finsi
29                         finsi
30                     finsi
31                 finsi
32             finsi
33         finprch
34     finprch
35 fin
```

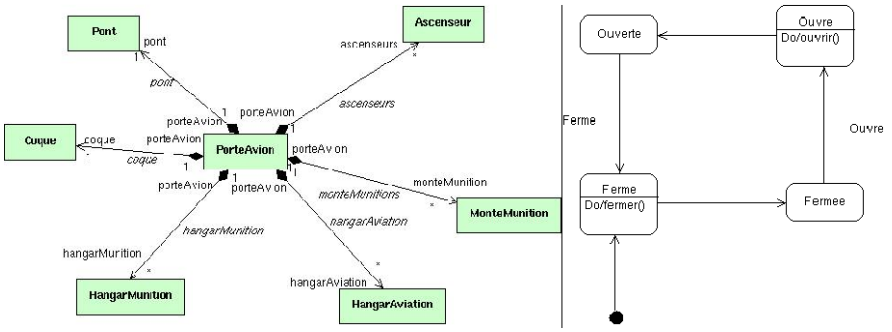


Figure 2.27 – Exemple de classes et de comportements dans Gaspar.

2.9 Application

Veha a été utilisé pour la réalisation de l'application Gaspar [Marion et al., 2007]. Ce projet a été mené pour Dcns de 2004 à 2008. L'objectif de Gaspar (gestion aviation sur porte-avions par la réalité virtuelle) est de réaliser des études de dimensionnement et d'ergonomie afin de démontrer à la marine nationale que le futur système couvre les besoins fonctionnels spécifiés. Les modèles géométriques des éléments du navire ont été réalisés en amont par une société d'infographie.

Le concepteur décrit dans un premier temps le modèle d'un porte-avions à l'aide d'un modeleur Uml supportant le métamodèle Veha. Dans le cas de Dcns, il s'agit de l'outil de génie logigiel Objecteering [4].

Les concepts de l'environnement

Le modélisateur décrit tout d'abord les concepts qui interviennent dans Gaspar sous forme de classes (notées en `EntityClass` grâce au profil Mascaret). Le diagramme de classes en figure 2.27 (gauche) fait apparaître qu'un porte-avions est composé, entre autres éléments, de monte-munitions et d'ascenseurs. Une instance de porte-avions dispose par exemple de trois monte-munitions et de deux ascenseurs (différentes configurations ont été testées). Chaque entité est décrite par ses propriétés (niveau de carburant d'un avion, angle maximun d'un déflecteur...) ainsi que ses opérations (ouverture du déflecteur, descente de l'ascenseur...). Le comportement de chaque entité du porte-avions est modélisé à l'aide d'une machine à états telle que sur la figure 2.27 (droite).

Les agents et les organisations

L'activité sur le porte-avions est effectuée par des équipes de personnels qui réalisent des procédures prédéfinies qu'il s'agit de simuler. Une procédure fait intervenir plusieurs rôles, réalisant des actions ordonnées (séquence, alternative, parallélisme) et manipulant des entités.

4. http://www.objecteering.com

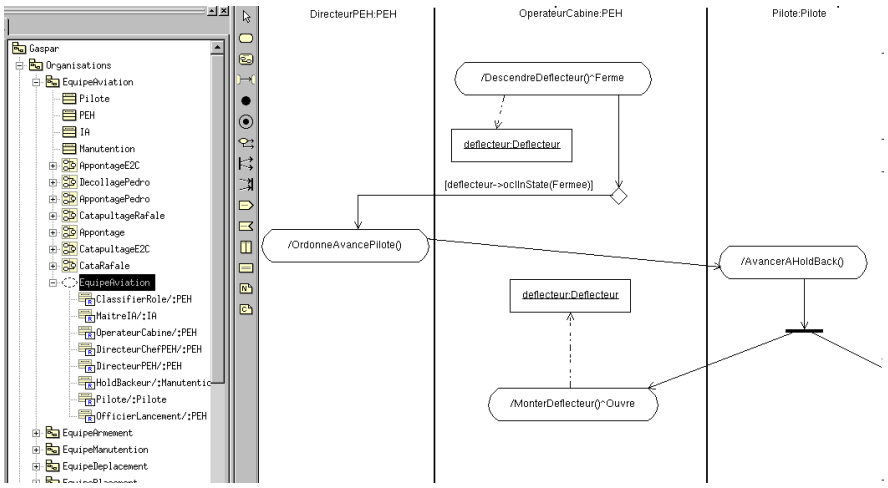


Figure 2.28 – Exemple d'équipe et de procédures dans Gaspar.

Les procédures sont décrites par un diagramme d'activités, selon le même modèle que celui de l'environnement (le porte-avions), ainsi le lien entre les procédures et l'environnement est possible. Une action d'un rôle peut déclencher le début d'une opération ou l'envoi d'un signal vers une entité de l'environnement (qui sera alors pris en compte dans la machine à état de ce dernier). Il est également possible d'utiliser l'état de l'environnement pour décrire le flot de contrôle (p. ex. déflecteur dans l'état fermé). La procédure est vue comme une connaissance de chaque agent qui décide d'effectuer les actions dont il a la responsabilité. Ceci montre que l'usage du métamodèle comportemental de Veha peut être étendu ; ici la sémantique opérationnelle correspondant à la réalisation d'une procédure par un collectif d'agents est spécifiée en stéréotypant la classe `Activity` en `<<procedure>>`.

Dans Gaspar, nous avons défini six structures organisationnelles, mais le client final en à défini d'autres liées à ses activités confidentielles. Ces structures organisent approximativement huit rôles (`OperateurCabine`, `Directeur`...) regroupés dans 4 types de rôles (`RoleClass` : `PEEH`, `IA`...). Ces équipes peuvent réaliser une dizaine de procédures dont les plus complexes (catapultage d'un Rafale par exemple) agencent environ cinquante actions (fixer le holdback, baisser la crosse...). Les équipes et procédures sont issues de documents formels produits par Dcns. Un exemple d'organisation et de procédures est donné sur la figure 2.28.

Instanciation de l'environnement et simulation

Les instances des classes définies dans le modèle Veha (Rafale1, déflecteur axial, ascenseur arrière...) sont décrites dans un fichier Xml. Toutefois, dans le cadre de Gaspar ce fichier à été généré automatiquement par un *plugin* que nous avons réalisé pour 3dsMax, l'outil utilisé par la société d'infographie pour concevoir la géométrie des objets. Il est plus facile d'utiliser un tel modeleur pour concevoir la scène globale (géométrie et position des objets). Ce fichier décrit chaque entité en lui affectant un nom, la classe du modèle Veha dont elle est une instance et la valeur initiale de chacune de ses propriétés (si différente de la valeur par

Figure 2.29 – Vues d'une scène sur porte-avions dans Gaspar

défaut de la classe). Les points informés servant aux interactions (mise en place du holdback par exemple) sont également déterminés dans ce fichier. L'environnement est structuré en plusieurs zones (ponts, hangars...). Le déplacement des aéronefs est guidé par des chemins précalculés (classes métiers, instances de la métaclasse `Path`).

La planification des flux aviation sur le navire a été calculée à l'aide d'un logiciel d'ordonnancement. Le résultat de ce calcul est un scénario nominal qui indique l'heure souhaitée pour déclenchement d'une procédure. Ce calcul ne se base que sur des estimations de temps de réalisation des actions, sans tenir compte des contraintes spatiales. La simulation à l'aide de Mascaret (figure 2.29) permet de valider ce scénario ainsi que les comportements des entités et les procédures réalisées par les équipes. Durant la simulation, l'utilisateur (le concepteur du scénario) peut naviguer librement dans l'environnement. Afin de tester les procédures, il lui est également possible de prendre la main sur un agent et de faire les actions à la place de ce dernier (par le biais d'un menu présentant les actions possibles) ou d'agir directement sur un objet en lui envoyant un signal particulier.

Dans Gaspar, une scène classique telle que présentée sur la figure 2.29 est constituée d'environ 1 000 entités ayant chacune une représentation 3D (VRML) pour un total d'environ 1 million de facettes au total. Dans cette scène, il y a environ 50 personnages, répartis en 10 équipes de 5 rôles en moyenne. Chacune de ces équipes a la responsabilité d'environ 5 procédures. Dans cette scène, à chaque instant, environ 50 comportements sont en cours de réalisation (suivi de procédures par les agents et réalisation d'opérations par les entités). Une telle scène est implémentée en utilisant ARéVi [Harrouet et al., 2006] et est simulée en temps réel (environ 40 images par secondes) sur un ordinateur personnel doté de 2 Go de mémoire vive, un processeur cadencé à 1.3 GHz et une carte de type GeForce avec 1 Go de mémoire vidéo. Initialement, l'application avait été réalisée entièrement en C++. Sa refonte en utilisant Veha a réduit le volume de code spécifique de 80 % sans dégradation préjudiciable des performances.

Tous les concepts « métiers » étant explicites lors de la simulation, il est facile de les horodater afin de fournir automatiquement un journal d'activité. Nous fournissons également un outil (Gedipom) s'appuyant sur Mascaret capable de lire ce journal et de présenter les résultats sous forme de graphes tels que présentés sur la figure 2.30. Dans le cadre de Gaspar, ce logiciel sert à analyser l'occupation des ressources et du personnel et à valider les temps d'exécution des procédures et des actions.

D'autres scénarios ont été implémentés sur d'autres navires (Bpc, Frégates) (figure 2.31

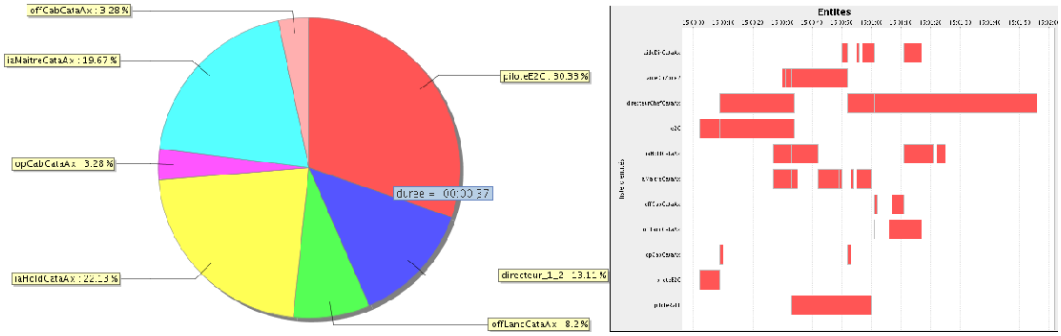


Figure 2.30 – Résultats de simulation à l'aide de Gedipom.

Figure 2.31 – Exemples d'autres applications de Mascaret.

gauche). Le même applicatif a été utilisé dans un autre cadre métier : la simulation de plans d'intervention pour la sécurité civile (SécuRévi). Ce projet a donné lieu à la création d'une *start-up* : Inovadys [5] et à une trentaine de conventions d'études avec des industriels. L'objectif est ici aussi de représenter des connaissances dans un environnement virtuel et de faire de la simulation comportementale interactive (figure 2.31 droite). Cette expérience illustre le caractère générique de la proposition et la capacité de réaliser des applications manipulant de gros volumes de données.

2.10 Conclusion

Dans ce chapitre, nous avons montré qu'Uml fournit un cadre de modélisation permettant de définir un métamodèle d'environnement virtuel structuré et informé autorisant a son tour la construction de modèles d'environnement ayant une sémantique métier. L'objectif est d'enrichir les environnements virtuels d'informations nécessaires à la réalisation des activités des utilisateurs et des agents artificiels tels que des personnages autonomes, des assistants ou

5. http://www.inovadys.com

des agents pédagogiques.

La nécessité d'unifier les concepts existant dans différents modèles ou langages utilisés pour construire des environnements virtuels informés et structurés dans lesquels évoluent des agents autonomes justifie notre proposition. Mascaret fait le lien entre la modélisation ontologique du domaine métier (classe et instance, propriété), la modélisation de l'environnement virtuel (entité, propriétés topologiques et géométriques), la modélisation comportementale (comportement, action, état, transition) et la modélisation de systèmes d'agents (comportement, organisation, plan d'actions). Ce résultat est acquis par une plus forte abstraction (formalisation d'un métamodèle et non de modèles « génériques ») et la caractérisation des éléments communs aux différentes facettes de la modélisation. Mascaret va un peu plus loin qu'Uml sur ce point.

Le recours à Uml évite l'inflation de concepts et facilite l'acceptabilité industrielle de la solution par l'utilisation de standards éprouvés. Ceci offre également la possibilité de réutiliser ou développer des outils fondés sur Mof pour construire et « embarquer » les modèles métiers. Les modèles d'environnement virtuel (M1) peuvent être construits à l'aide de modeleurs Uml. Pour cela le modeleur doit permettre d'utiliser un modèle M2 qui soit une extension formelle d'Uml et autoriser l'export des modèles au format Xmi. L'objectif est ensuite de développer des greffons qui permettent la construction des environnements virtuels (M0) à partir d'un modèle métier (M1) et l'exécution de modèles en temps réel tout en garantissant la sémantique opérationnelle, ce que nous avons fait avec ARéVi. Nous travaillons actuellement au développement de tels greffons pour Blender [6] pour la modélisation, ainsi que Ogre [7] et Virtools [8] pour la simulation.

Les applications que nous avons faites de Mascaret montrent la possibilité du passage à l'échelle avec comme exigences l'expressivité des modèles et leur exécution en temps réel. L'utilisation de ces applications met en évidence le gain apporté par l'introspection des niveaux M1 et M0.

En conclusion, nous considérons que Mascaret rend possible la simulation de la situation pédagogique tout en en fournissant une représentation explicite en ligne. Ceci permet d'analyser cette simulation tel que nous l'avons vu dans le projet Gaspar. Cela autorise surtout la définition de comportements d'agents à visée pédagogique. Dans le chapitre 3, nous définissons de tels comportements, s'appuyant sur Mascaret, pour agir sur la relation pédagogique du triangle didactique. Dans le chapitre 4 nous proposons un modèle de scénario pédagogique, s'appuyant également sur Mascaret, pour instrumenter la relation didactique.

A court terme, nous prévoyons d'enrichir Mascaret selon deux axes : 1. la formalisation d'une extension d'Ocl pour traiter de contraintes spatiales et temporelles, et la définition de primitives d'action spécifiques à Mascaret dont la sémantique sera formellement définie à l'aide des contraintes évoquées précédemment, Ces travaux font l'objet de la thèse de T.H. Trinh que je co-encadre avec P. Chevaillier (soutenance prévue en 2011), 2. la description de comportements d'agents capables d'interpréter d'autres modèles dynamiques d'Uml (séquences, collaborations, etc). Ceci nous conduira certainement à mieux formaliser le comportement de communication ainsi que la base de connaissances des agents. Ces travaux

6. http://www.blender.org
7. http://www.ogre3d.org
8. http://www.virtools.com

sont abordés dans le cadre de la thèse de F. Lecorre, encadrée par C. Buche (soutenance prévue en 2012).

Chapitre 3

Modélisation de la relation pédagogique

Dans le chapitre précédent, nous avons proposé un modèle qui permet de concevoir l'environnement virtuel tout en rendant explicite les connaissances et compétences à acquérir. Nous nous intéressons maintenant à l'utilisation de ce modèle pour représenter la relation pédagogique qui lie le formateur et l'apprenant (figure 3.1).

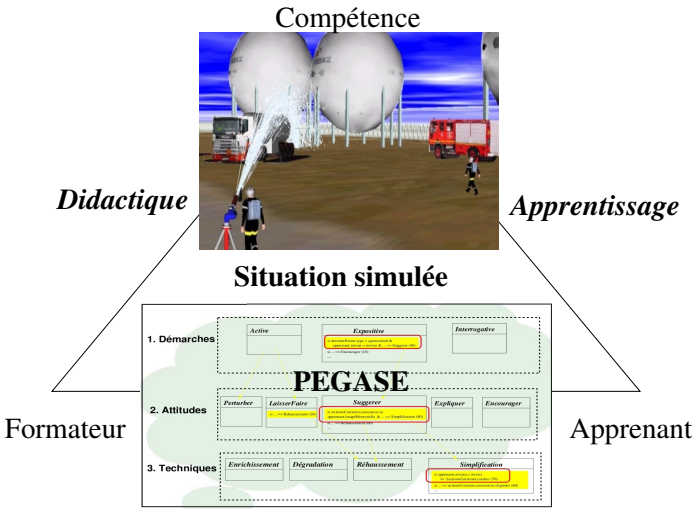


FIGURE 3.1 – Intégration d'un ITS dans la situation d'apprentissage.

Dans ce cadre, nous proposons l'utilisation de systèmes tutoriaux intelligents (en anglais ITS : *Intelligent Tutoring System*) qui utilisent les connaissances liées à la situation de formation. Ces connaissances seront manipulées, par exemple, pour interroger l'apprenant automatiquement. Nous proposons PEGASE, ([Buche et al., 2010, Buche et al., 2009]) un système tutoriel intelligent générique et adaptatif. L'objectif est de fournir des assistances pédagogiques à l'apprenant et une aide au formateur. PEGASE propose une représentation des modèles classiques qui constituent un ITS, mais ce sont les modèles métier et pédagogique qui sont les points originaux de PEGASE. Le modèle métier se fonde sur MASCARET. Ceci

permet à Pegase d'être indépendant d'un métier particulier. Nous détaillons dans ce chapitre l'architecture globale de Pegase puis nous détaillons notre proposition pour chacun des modèles d'un ITS. Nous détaillons plus particulièrement le modèle pédagogique. Ces travaux ont donné lieu à l'encadrement de la thèse de C. Buche [Buche, 2005], co-encadrée avec P. De Loor et J. Tisseau et soutenue en 2005.

3.1 Positionnement

Les ITS ont déjà été utilisés indépendamment de la réalité virtuelle. Comme le montre [Wenger, 1987], ils utilisent généralement quatre modèles. Le premier, appelé modèle du domaine, contient une représentation des connaissances liées à la compétence à acquérir. Les ITS utilisent également un modèle de l'apprenant qui précise ses caractéristiques personnelles et qui établit l'état de ses connaissances à un instant donné. Utilisant le modèle du domaine et de l'apprenant, l'ITS est capable d'évaluer les connaissances acquises par l'apprenant, en comparant ses activités et les informations sur le domaine. Mais l'objectif principal de l'ITS est de fournir une assistance adaptée à la situation pour l'apprenant ou le formateur (suivi d'activités ou propositions d'assistances). Dans ce cadre, le modèle pédagogique permet d'effectuer des choix concernant la formation et visant à favoriser l'apprentissage. Enfin, un modèle d'interface permet l'échange d'informations entre le système et l'utilisateur. Ce modèle n'est pas réifié pour l'instant dans les EV existants destinés à l'apprentissage.

Dans le cadre de nos EV, nous considérerons un ITS comme étant un système qui fait partie d'un environnement virtuel d'apprentissage humain (EVAH). Nous proposons d'évaluer le niveau d'intégration des ITS dans les EVAH existants que nous avons regroupés selon trois catégories :

1. *L'EVAH est un simulateur classique*
   Cette première catégorie regroupe les applications n'intégrant aucun des quatre modèles, comme dans l'application d'aide à la maintenance et à la conduite des ponts roulants présentée dans [Levesque, 2003]. Dans ce type d'EV, le système ne peut pas fournir des explications sur la tâche à réaliser, ce qui nécessiterait un modèle du domaine. Il ne peut pas s'adapter à un apprenant, ce qui nécessiterait un modèle de l'apprenant. Enfin, la pédagogie relève de la responsabilité du formateur. Le système ne peut pas prendre des décisions relatives à des interventions pédagogiques. Toutefois, un tel système permet le renforcement de compétences préalablement acquises.
2. *L'EVAH contient des modèles du domaine et/ou de l'apprenant*
   Cette deuxième catégorie d'EV est constituée des applications intégrant un modèle du domaine et/ou un modèle de l'apprenant. L'exemple le plus connu est Steve, un personnage qui a pour objectif la formation à des tâches procédurales [Rickel and Johnson, 1999]. Utilisant le modèle du domaine, il sait montrer la procédure, l'expliquer et surtout valider les actions de l'apprenant. Cependant, Steve intervient à la demande. Il est incapable de savoir quand, comment et pourquoi intervenir, ce qui nécessiterait un modèle pédagogique. Dans un tel système, l'acquisition de compétence est possible, mais les interventions pédagogiques restent du ressort du formateur.

3. *L'EVAH contient un modèle du domaine, un modèle de l'apprenant, et un modèle pédagogique*
Cette dernière catégorie rassemble les EV qui présentent non seulement les modèles du domaine et de l'apprenant, mais également un modèle pédagogique. Prenons pour illustration l'agent pédagogique Hal utilisé dans le système Fiacre [Lourdeaux, 2001]. L'application est destinée à la formation individuelle des agents de conduite de train à grande vitesse (TGV) à l'aide de la réalité virtuelle (intervention sur les voies ferrées). Outre les capacités de Steve, Hal aide les formateurs à construire un discours pédagogique. Concrètement, pour chaque comportement attendu, les assistances pédagogiques sont décrites (mise en transparence d'objet, ajout d'information...). Le formateur doit lister de manière exhaustive les erreurs pour chaque connaissance. De plus pour chacune de ces erreurs, il doit préciser la manière dont les stratégies pédagogiques sont réalisées par des assistances pédagogiques et cela pour chaque exercice. L'intérêt principal réside dans l'aide apportée au formateur dans la relation pédagogique qui le lie à l'apprenant et dans la relation didactique qui le lie à la compétence. Toutefois, le formateur doit expliciter l'ensemble des connaissances pour chaque exercice.

Ainsi, la plupart des EV n'incorpore que la représentation des connaissances sur le domaine. Quant aux systèmes qui proposent un module de diagnostic, ils ne fournissent que très rarement un mécanisme d'assistance pédagogique. Hal nous semble être le système le plus abouti. Néanmoins, le formateur doit lister les erreurs et préciser les stratégies pédagogiques pour chaque exercice. Bien plus, l'impact des assistances pédagogiques sur l'apprenant n'est pas considéré. Concrètement, une assistance pédagogique ne faisant pas progresser un apprenant sera remise en place à chaque fois que la situation en question apparaît.

Pour pallier ces limitations, notre proposition consiste à intégrer un système tutoriel intelligent au sein d'un EV. Ce système doit proposer un modèle pédagogique modulable, *i.e.* permettant d'intégrer, de modifier ou de supprimer des concepts pédagogiques facilement. De plus, il doit être *générique* dans le sens où le modèle pédagogique devra être utilisable indépendamment de l'exercice à réaliser. Enfin, les connaissances du modèle pédagogique et les expériences passées pourront être utilisées pour proposer automatiquement les interventions appropriées, en considérant l'apprenant et le contexte de simulation : le système devient *adaptatif*. Le modèle proposé se nomme Pegase (*PEdagogical Generic and Adaptive SystEm*).

Dans la section 3.2, nous décrivons l'architecture globale de Pegase. Le modèle du domaine est représenté par Veha, Behave et Have qui ont fait l'objet du chapitre précédent. Nous présentons par la suite les modèles que nous proposons. Nous discutons enfin de l'apport de notre proposition (cf. section 3.9). Notons que la proposition développée ici s'applique dans le cadre de l'*apprentissage de tâches procédurales et collaboratives* et n'est pas utilisable dans les situations d'apprentissage d'ordre général.

3.2 Vue d'ensemble de Pegase

Notre proposition consiste à réifier les quatre modèles classiques d'un ITS au sein d'un EV (domaine, apprenant, pédagogie, interface). Comme [Py, 1998], nous considérons les erreurs

comme des informations cruciales. Par conséquent nous avons décidé d'introduire un modèle appelé « modèle des erreurs ». C'est grâce à ce modèle que nous allons pouvoir généraliser (là où Hal ne le pouvait pas). De plus, nous ajoutons un « modèle du formateur » qui définit les connaissances sur l'exercice à réaliser précisées par le formateur. Il fixe les consignes, qui définissent la(les) procédure(s) à effectuer, et le(s) rôle(s) à jouer par tel apprenant (et par conséquent ceux qui seront joués automatiquement).

Ces modèles devront répondre aux limitations des systèmes existants que nous avons dégagées et par conséquent présenter deux propriétés : généricité et adaptativité. Pour que chaque modèle puisse partager ses informations et effectuer ses analyses en toute autonomie (indépendamment de la simulation et des autres modèles), une entité autonome (appelée agent) est associée à chaque modèle.

Les agents interagissent en s'échangeant des messages contenant des données (cf. figure 3.2). Celles-ci peuvent être extraites de la simulation ou déduites d'un raisonnement interne à l'agent à partir de ses connaissances (modèle auquel il est lié). La structure d'agent et le mode de communication utilisés ici s'appuient sur le *package* Behave et les messages en Fipa-Acl. Chaque agent possède ses propres comportements qui héritent des comportements définis dans Behave et sont implémentés par des `OpaqueBehavior`. Le comportement global du système d'agent est le suivant :

Étape 1. *Observer*
Utilisant le modèle d'interface, le système analyse les activités de l'apprenant. Les éléments pertinents pour la formation sont fournis au modèle de l'apprenant. Ces informations concernent les actions de l'apprenant, les éléments observables par l'apprenant, les déplacements de l'apprenant.

Étape 2. *Détecter et identifier une erreur*
Le système analyse les actions de l'apprenant (modèle de l'apprenant) et les compare aux actions à effectuer (modèle du domaine). Cette confrontation permet de détecter une éventuelle erreur. Si une erreur a été détectée, un mécanisme d'identification de l'erreur est mis en place (en utilisant le modèle des erreurs).

Étape 3. *Proposer des assistances pédagogiques*
Utilisant le modèle de l'apprenant (caractéristiques, activités, erreurs, etc.) et le modèle du domaine (connaissances sur les structures organisationnelles), un mécanisme simulant un raisonnement pédagogique préconise les assistances pédagogiques adaptées à la situation. Notons que cette étape n'est pas conditionnelle, elle intervient même si aucune erreur n'est détectée.

Étape 4. *Sélectionner une assistance pédagogique*
Le formateur peut sélectionner une assistance pédagogique parmi les propositions.

Étape 5. *Représenter l'assistance pédagogique*
L'assistance pédagogique sélectionnée est présentée dans l'environnement virtuel.

Ce comportement d'agent pédagogique s'appuie sur les connaissances sur l'environnement exprimées à l'aide de Mascaret. Ces connaissances sont complétées par des informations supplémentaires contenues dans les 6 modèles de l'ITS. Ces données forment une base de connaissances pour le modèle pédagogique que nous appelons *situation pédagogique*. Ces

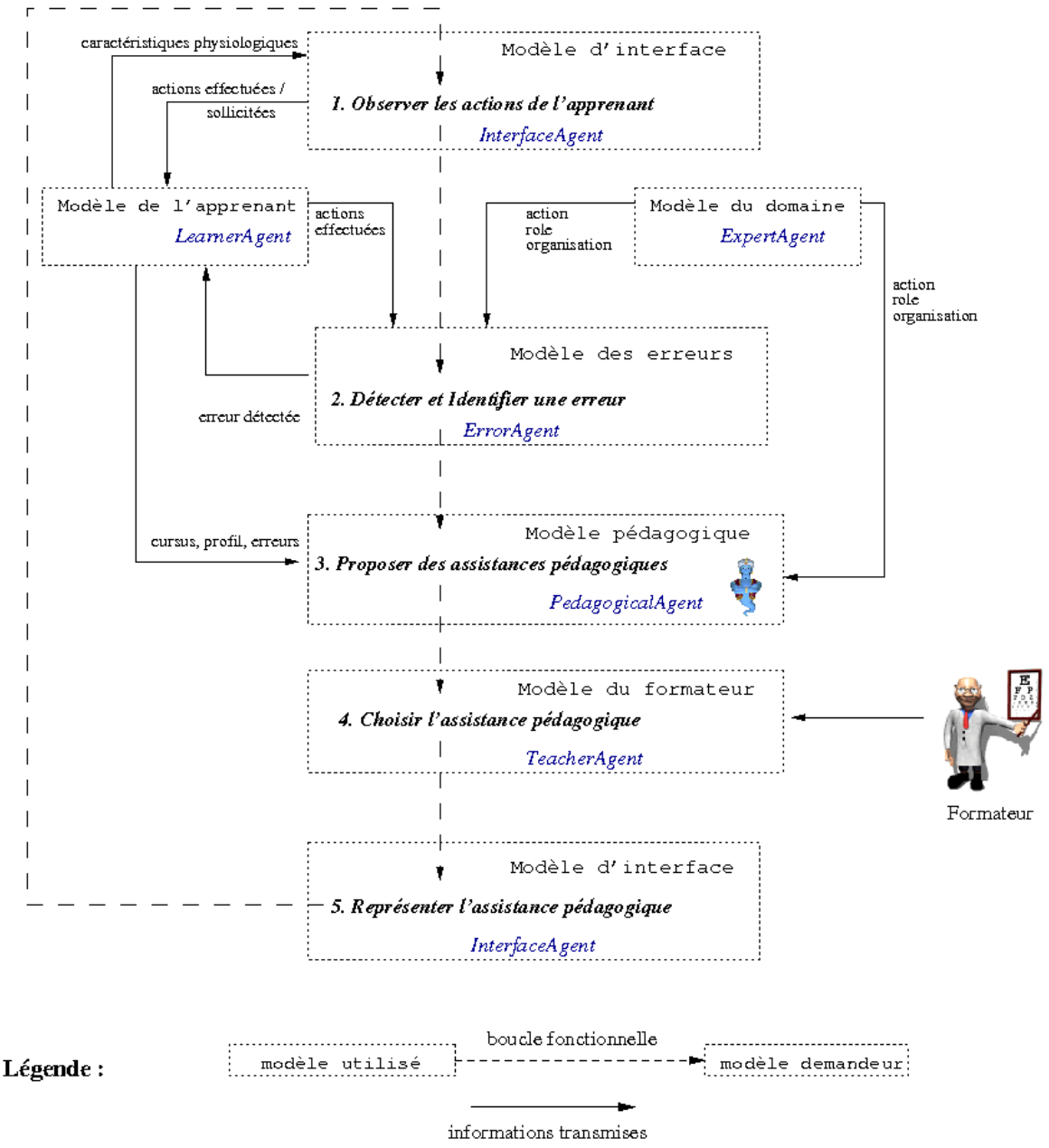


Figure 3.2 – Processus pédagogique de notre système constitué de cinq étapes

connaissances alimentent le moteur de prise de *décision pédagogique* de l'ITS (cf. section 3.8). Nous détaillons par la suite les différents modèles de l'Its.

3.3 Modèle du domaine

Le modèle du domaine est une représentation de la connaissance que l'apprenant doit acquérir, il représente l'ensemble des connaissances expertes sur le domaine d'apprentissage. Il contient également des informations permettant d'interpréter ces connaissances, afin de permettre l'exécution des actions pour « *faire à la place* » de l'apprenant. Le modèle du domaine contient alors des connaissances qui sont comparables aux connaissances déclaratives (savoir) et procédurales (savoir-faire) liées à la compétence définie précédemment. Le modèle du domaine est en fait le modèle qui est décrit par l'expert à l'aide de Veha, Behave et Have. Il s'agit alors de spécifier les équipes, les procédures, les actions et les règles d'usage général.

Nous décrivons ici les formalismes utilisés pour réifier ces informations pour chaque type de connaissance :

1. Travail procédural en équipe.
   La représentation des connaissances sur les équipes utilise la structure organisationnelle définie dans Behave qui précise les responsabilités. L'ordonnancement des actions au sein de la procédure est représenté par la structure du diagramme d'activités. Les actions sont définies par un but et des conditions évaluables. Ainsi, le modèle du domaine utilise directement les informations issues du modèle d'organisation que nous avons précisé pour le travail procédural en équipe.
   Plus précisément, le modèle du domaine maintient des connaissances générales indépendantes du contexte dans lequel elles sont utilisées. Par conséquent, le modèle utilisé s'appuie sur les organisations, mais ne référence pas les instances de ces organisations. Ainsi, un agent sollicitant l'`ExpertAgent`, ne demande pas des informations spécifiques (*e.g.* Est ce que l'agent Y doit faire l'action « marcher vers l'objet X » ?), mais pose une question générique : « quelles actions B doivent être effectuées, après l'action A, dans la mission M, considérant le rôle R_1 et dans l'organisation O ? ».
   De plus, le modèle du domaine contient les liens entre actions qui ne sont pas exprimés dans la procédure. Les actions sont liées entre elles en considérant les relations de dépendance « logique », indépendamment de l'agencement de la procédure. Ces liens sont spécifiés par l'expert du domaine.
   Exemple : L'action « démarrer la voiture » est liée à l'action « prendre les clefs de la voiture ».
   Chaque action connaît les actions qui sont dépendantes d'elles et celles dont elle-même est dépendante. Ainsi, il est possible d'expliquer qu'il faut faire l'action A puisqu'elle est nécessaire pour effectuer l'action B.
2. Les règles d'usage général.
   La représentation des règles d'usage utilise un formalisme fondé sur des règles logiques. Nous souhaitons obtenir des explications (telle condition est non remplie). Nous proposons alors qu'une règle soit constituée de composantes qui correspondent à des termes logiques évaluables. L'évaluation des termes permet de générer une explication le cas échéant.

Exemple de règle : un véhicule peut ne pas s'arrêter à un stop si il est un véhicule prioritaire (composante logique 1) et qu'il a actionné sa sirène (composante logique 2).

Pour résumer, l'expert du domaine doit préciser la structure organisationnelle, les procédures, les actions, les liens entre actions et les règles d'usage général. Ces informations constituent la base de connaissances de l'agent expert.

Comportements

L'agent expert (`ExpertAgent`) n'a pas un comportement réactif opérationnalisé. Il a un rôle d'informateur. En effet, l'agent expert renseigne tout modèle qui souhaite l'utiliser. Il peut raisonner sur le domaine concernant les éléments suivants :

1. *L'environnement social* : `OrganisationBehavior`.
   Il possède la capacité de raisonnement sur les agents et les rôles. L'`ExpertAgent` est capable de renseigner sur les responsabilités de chaque intervenant. En considérant la connaissance sur l'environnement social, il répond à la question « est ce que l'action A est de la responsabilité du rôle R ? ». Il peut également renseigner des rôles affectés à chaque agent par organisation.
2. *Les procédures* : `ProcedureBehavior`.
   Il peut fournir l'ordonnancement des actions ainsi que les buts de chaque étape (procédure / action). Comme Steve [Rickel and Johnson, 1999], l'`ExpertAgent` est capable de fournir des explications. Il répond à la question « quelles sont les actions à réaliser après l'action A ? ». Il précise le type de relation entre les actions. Exemple : « Après l'action A, il faut faire l'action B ou (l'action C et l'action D) ».
3. *Les actions* : `ActionBehavior`.
   Le modèle organisationnel, défini dans Behave, propose un modèle d'actions soumis à des pré/post conditions. Ces dernières sont représentées sous la forme de règles. Comme Steve, l'`ExpertAgent` est capable de fournir des explications et de faire à la place de l'apprenant. Il peut fournir les pré/post conditions et expliquer pourquoi elles sont (ou ne sont pas) satisfaites. Il peut également donner les liens entre actions indépendants des procédures.
4. *Les règles du domaine* : `RuleBehavior`.
   L'`ExpertAgent` a la possibilité de fournir des explications sur les « règles d'usage » (indépendantes de la procédure) (`Rule`) en s'appuyant sur les réifications des termes de la règle. En effet, déterminer si une règle est transgressée revient à une évaluation individualisée des termes de la règle. Il est alors possible de déterminer quel terme est impliqué dans le non respect de la règle.

3.4 Modèle des erreurs

L'apprenant doit réaliser une procédure dans laquelle il joue un ou plusieurs rôles. Lorsqu'il évolue dans l'environnement, il peut effectuer des comportements qui sont en désaccord avec le travail à réaliser ou avec des règles d'usage général. Le modèle des erreurs offre alors une caractérisation générique des erreurs permettant de les détecter et de fournir à l'agent pédagogique des moyens d'y réagir.

Le modèle des erreurs permet de détecter et de typer des erreurs effectuées par l'apprenant indépendamment de l'exercice. Outre cette capacité à caractériser une erreur d'une manière générique, il contient une base de connaissances sur des erreurs classiques dans un domaine particulier permettant d'expliquer pourquoi il y a erreur, et d'associer des éléments de l'environnement impliqués pouvant être utilisés pour faciliter leur compréhension. Ces connaissances sont spécifiées par le formateur. Les erreurs détectées et éventuellement associées à des erreurs classiques portent sur la compétence à acquérir.

Base de connaissances

Le premier objectif du modèle des erreurs est de caractériser les erreurs de manière générique. Pour cela, il faut représenter les informations qui explicitent la distinction entre telle ou telle erreur, sans considérer l'exercice particulier à traiter. Nous proposons alors de distinguer différents types d'erreurs caractéristiques :

1. Les règles d'usage (`RuleError`).
   Cette erreur précise les termes impliqués dans la mise en échec de la règle. Considérons la règle d'usage suivante : « en cas de fuite de gaz, il faut arroser la source de la fuite seulement si la chaleur est supérieure à trente degrés et que le vent est inférieur à 70km/h. » Une fuite est survenue, l'apprenant donne l'ordre d'arroser la source. La température est de vingt degrés et le vent est de 20 km/h. C'est une erreur portant sur une règle d'usage, le terme impliqué dans la mise en échec est « la chaleur est supérieure à trente degrés ».
2. Sur les actions (`ActionError` : pré-condition).
   Elle détermine les parties de la pré-condition d'une action qui sont en échec. Considérons l'action A_1 « faire une mesure d'explosimétrie » qui a pour pré-condition « possède un explosimètre ». L'apprenant sollicite l'action A_1 sur le pompier chef, ce dernier ne possède pas un explosimètre. C'est une erreur portant sur une action. La pré-condition en échec est « possède un explosimètre ».
3. Sur les rôles (`TeamError` : rôle dans l'équipe).
   Elle précise la différence de responsabilité entre des rôles à jouer et le rôle lié à une action particulière. Cette erreur intervient lorsque l'action sollicitée fait partie des actions à réaliser, mais n'est pas de la responsabilité de l'apprenant.
   Exemple : Dans la procédure courante P_1, la dernière action correcte effectuée est l'action A_0, les actions possibles sont $\{A_1, A_2$ et $A_3\}$. Les rôles joués par l'apprenant contiennent les actions $\{A_0, A_1$ et $A_7\}$. L'action réalisée est l'action A_3. Bien que cette action fasse partie des actions à réaliser, elle n'est pas contenue dans l'ensemble des actions possibles pour les rôles joués, il s'agit donc d'une erreur.
4. Sur la procédure (`ProceduralError` : ordonnancement).
   Elle souligne la différence entre une action et les actions possibles pour une procédure donnée. Cette erreur intervient lorsque l'action sollicitée ne fait pas partie des actions à réaliser.
   Exemple : Dans la procédure courante P_1, la dernière action correcte effectuée est l'action A_0, les actions possibles sont $\{A_1, A_2$ et $A_3\}$. L'action réalisée est l'action A_8. Cette action n'est pas contenue dans l'ensemble des actions possibles, il s'agit donc d'une erreur.

5. Sur les rôles dans la procédure (`TeamProceduralError` : rôle dans l'ordonnancement). Il s'agit d'une spécialisation de `ProceduralError`. Cette erreur intervient lorsque l'action sollicitée ne fait pas partie des actions à réaliser et en plus lorsqu'elle n'est pas de la responsabilité de l'apprenant.
Exemple : Dans la procédure courante P_1, la dernière action correcte effectuée est l'action A_0, les actions possibles sont $\{A_1, A_2$ et $A_3\}$. Les rôles joués par l'apprenant contiennent les actions $\{A_0, A_1$ et $A_7\}$. L'action réalisée est l'action A_8. Cette action ne fait pas partie des actions à réaliser, en plus cette action ne fait pas partie d'un des rôles joués, il s'agit donc d'une erreur.

Pour déterminer le type de l'erreur, il suffit de connaître la procédure courante, les rôles joués par l'apprenant et les règles d'usage général.

Comme nous l'avons souligné précédemment, le modèle des erreurs offre bien plus qu'une caractérisation générique des erreurs. En effet, la base de connaissances de l'agent erreur est constituée d'une liste d'erreurs « classiques » du domaine faisant référence à des cas particuliers des exercices à réaliser. Ces erreurs sont classées par type et associées à des informations supplémentaires. Ces dernières prennent la forme d'une règle qui est spécifiée par le formateur. Une règle est composée de deux parties. La première précise les conditions qui permettent de détecter l'erreur en question. Les conditions portent sur le type générique de l'erreur (`RuleError`, `TeamError`, `TeamProceduralError`, `ProceduralError` ou `ActionError`) et sur les informations particulières de l'erreur.
Exemple : « Si l'erreur est de type ProceduralError et que l'apprenant a effectué l'action A au lieu de l'action B ».

La connaissance est alors représentée en spécifiant les caractéristiques de ces erreurs classiques (partie gauche de la règle) que nous augmentons par des informations supplémentaires associées (partie droite de la règle). Plus précisément ces informations, constituant la deuxième partie de la règle, sont :
– un texte explicatif précisant en quoi c'est une erreur,
– des éléments de l'environnement pouvant être utilisés pédagogiquement en relation à cette erreur (objet 3D),
– des éléments externes à l'environnement (vidéo, documents écrits, *etc.*) en relation à cette erreur.
Nous appelons les règles représentant les erreurs classiques du domaine des concepts sur erreurs (`Concept_on_Error`).

Comportements

L'agent erreur (`ErrorAgent`) a pour principal objectif de détecter une erreur et d'en informer les autres modèles. Ce renseignement est primordial, notamment pour la prise de décision pédagogique. De plus, comme nous l'avons vu, l'agent erreur peut éventuellement reconnaître une erreur classique et dans ce cas fournir des informations complémentaires qui constituent des ressources pédagogiques adaptées à la situation. Ainsi, l'`ErrorAgent` propose les comportements suivants :

1. *Détecter et typer les erreurs :* `ErrorDetectionBehavior`.
   Il détecte une erreur potentielle effectuée par l'étudiant :

   (a) Il vérifie que les pré-conditions de l'action sollicitée par l'apprenant sont satisfaites (`ActionError`).

   (b) Il compare les actions possibles pour avancer d'un pas dans la procédure (fournies par le modèle du domaine) et l'action sollicitée par l'apprenant (fournie par le modèle de l'apprenant). Les informations sur ces actions sont récupérées en dialoguant respectivement avec l'`ExpertAgent` et le `LearnerAgent` représentant l'apprenant. Il s'agit de vérifier si l'action sollicitée fait partie des actions potentiellement possibles en terme de procédure, *i.e.* qu'elle est l'une des actions qui doivent être effectuées en considérant la dernière action correcte et l'ordonnancement de la procédure (`ProceduralError`). Il s'assure également que l'apprenant a bien la responsabilité (définie par les rôles qu'il doit jouer) d'effectuer l'action sollicitée (`TeamError` ou `TeamProceduralError`).

   (c) Il vérifie les termes des règles d'usage général du modèle du domaine (`RuleError`).

2. *Reconnaître une erreur classique du domaine :* `ErrorAssociationBehavior`.
   Notre modèle des erreurs contient une base de connaissances sur les erreurs classiques effectuées par un étudiant dans le domaine concerné. Il peut être comparé au "*Buggy model*" présent dans d'autres ITS [Brown and Burton, 1978]. L'`ErrorAgent` est capable de reconnaître une telle erreur en comparant l'erreur à la partie gauche des règles correspondant aux erreurs connues de la base de connaissances. Il enrichit alors l'erreur (qui a été préalablement typée) avec des éléments supplémentaires (`Concept_on_Error`).
   Lorsque l'`ErrorAgent` a détecté, typé une erreur (et éventuellement reconnu une erreur classique du domaine), il transmet ses informations à l'agent pédagogique.

3. *S'auto enrichir :* `EnrichBehavior`.
   L'`ErrorAgent` enregistre des informations sur la fréquence des erreurs afin d'en rendre compte au formateur. Un mécanisme, sous le contrôle du formateur, utilise cette information pour enrichir la connaissance sur les erreurs classiques du domaine.

3.5 Modèle de l'apprenant

Il existe à ce jour de nombreux modèles qui proposent de modéliser un utilisateur [Rouse, 1982, Rasmussen, 1986, Anderson, 1993]. Contrairement à ces approches, nous n'avons pas l'ambition de modéliser un humain virtuel, mais plutôt de récolter des informations dans un cadre de formation. Le modèle de l'apprenant s'intéresse aux caractéristiques et aux activités de l'apprenant. Il doit représenter les informations caractérisant l'étudiant tant d'un point de vue prototypique (profil de l'étudiant) que d'un point de vue cursus (progression de l'étudiant). Il correspond alors à la représentation de l'apprenant dans la situation d'apprentissage.

Base de connaissances

Le modèle de l'apprenant représente les connaissances qui caractérisent l'apprenant. Dans notre cas, les informations liées à l'acquisition de compétences sont particulièrement importantes. De plus, nous devons considérer les particularités de l'apprenant qui ne sont pas liées à son cursus (*e.g.* capacité auditive).

Pour représenter ces connaissances, nous avons utilisé la distinction proposée par [Delestre, 2000], séparant dans la base de connaissances la partie *épistémique* et la partie *non épistémique* :

1. La partie *épistémique* (`EpistemicKB`) fournit les informations sur l'état des connaissances de l'apprenant pour les notions présentes dans le domaine. Elle contient une représentation des actions et des erreurs effectuées par l'apprenant. Les actions et les erreurs de l'apprenant sont stockées en suivant le déroulement de la simulation, nous obtenons alors son cursus lors de l'exercice.

Comme dans la proposition de [Lourdeaux, 2001], notre modèle de l'apprenant contient un sous-modèle du domaine en représentant les actions effectuées, reprenant la méthode de l'*overlay* [Stansfield et al., 1976]. Nous faisons alors l'hypothèse simplificatrice considérant que les actions reflètent les acquis de l'apprenant.

2. La partie *non épistémique* (`NonEpistemicKB`) propose une représentation de trois concepts issus de l'ingénierie cognitive :

(a) un profil *psychologique* (par rapport à la tâche) : cette connaissance provient d'une analyse d'un questionnaire rempli préalablement à l'exercice. Il permet de définir notamment la stratégie d'apprentissage privilégiée de l'apprenant [Kermarrec, 2002] ;

(b) des caractéristiques *physiologiques* : elles expriment les capacités auditives et visuelles de l'étudiant. Ces informations sont utilisées par le `PCVObservation` (*cf.* modèle d'interface 3.6) ;

(c) une image *mémorielle* : elle reflète des éléments potentiellement observés par l'apprenant, de manière instantanée (`sensorialTimeMemory`) ou à court terme sur les dernières secondes (`shortTimeMemory`). Un mécanisme d'oubli efface les éléments de la partie à court terme qui n'ont pu être observés récemment [1].

L'apprenant évolue dans un environnement virtuel. Nous devons alors considérer les activités de l'apprenant et notamment ses déplacements dans l'univers 3D. Par conséquent, la partie non épistémique possède des informations relatives aux déplacements de l'apprenant (zones, directions, vitesses, *etc*). Ces informations sont fournies par le modèle d'interface.

1. Il s'agit simplement d'un conteneur d'éléments 3D, il ne faut donc pas faire de rapprochement abusif avec la notion de mémoire à court terme utilisée en psychologie cognitive. En effet, nous n'avons pas réifié les connaissances qui pourraient permettre la représentation d'un tel concept. Il en est de même pour le concept de mémoire à long terme.

Comportements

L'agent apprenant met à jour les informations associées à l'apprenant cible. De plus, il informe les autres modèles qui souhaitent connaître les activités et particularités de l'apprenant.

1. Il met à jour les informations dynamiques concernant les activités de l'apprenant :
   (a) *Mise à jour du curriculum* : `CurriculumBehavior`
   Il stocke les nouvelles actions et les nouvelles erreurs effectuées par l'apprenant.
   (b) *Mise à jour de la mémoire* : `MemoryBehavior`
   Il stocke les objets qui ont pu être observés par l'apprenant en considérant ses caractéristiques physiologiques.
   (c) *Mise à jour des informations liées aux déplacements* : `MotionBehavior`
   Il stocke la position de l'apprenant (les applications sont structurées en zones), les directions et vitesses moyennes et instantanées.
2. Il répond aux requêtes d'autres agents (`InformBehavior`) concernant les activités (exemple : dernière action effectuée) ou particularités de l'apprenant (exemple : stratégie d'apprentissage privilégiée).

3.6 Modèle d'interface

Le modèle d'interface a la charge de la communication bidirectionnelle entre l'apprenant et le système. Dans un cadre de formation, il s'agit d'offrir à l'apprenant la possibilité d'interagir avec son environnement et d'analyser ses activités. Le modèle d'interface est particulièrement important lorsqu'on étudie une application de réalité virtuelle. Le modèle d'interface (`MI`) représente et analyse les interactions entre l'apprenant et le système. Plus précisément, son rôle est de régir la communication et les interactions avec l'environnement. De plus, il doit détecter les activités de l'apprenant. Pour tout cela, le modèle de l'interface doit considérer les caractéristiques particulières de l'apprenant. Il se place comme un médiateur entre le système et l'apprenant dans la situation d'apprentissage.

Deux acteurs humain jouent un rôle dans la spécification du modèle de l'interface : le formateur et le concepteur de l'EVF. En effet, lorsque le formateur précise l'exercice, il définit les éléments physiques à considérer dans l'analyse des observations de l'apprenant. De plus, le concepteur de l'EVF implémente les moyens d'interaction entre l'apprenant et le système.

Le modèle d'interface a pour objectif principal l'analyse des comportements de l'apprenant dans l'univers virtuel. Par conséquent, pour représenter le modèle d'interface, nous nous inspirons des Primitives Comportementales Virtuelles (PCV) [Fuchs and Moreau, 2003]. En effet, selon [Fuchs and Burkhardt, 2003], les activités d'un sujet dans une application de réalité virtuelle sont toujours décomposables en quelques comportements de base correspondants (les PCV). Les auteurs les regroupent en quatre catégories :
– observer le monde virtuel ;
– se déplacer dans le monde virtuel ;
– agir sur le monde virtuel ;
– communiquer avec autrui.

Base de connaissances

Afin de prendre en compte les caractéristiques personnelles de chaque apprenant pour analyser ses comportements, notre modèle d'interface contient une base de connaissances sur l'apprenant (`LearnerInformationKB` : sous partie de la base de connaissances du modèle de l'apprenant). Nous avons, par exemple, considéré ses particularités visuelles pour déterminer les éléments qu'il peut potentiellement observer dans le monde. De la même manière, le niveau de l'apprenant est utilisé pour régir ses possibilités d'actions.

Comportements

L'`InterfaceAgent` possède trois comportements qui correspondent à trois des quatre PCV (*cf.* figure **??**). En effet, nous n'avons pas traité la PCV communication, car les connaissances sur ce point semblent, pour l'instant, délicates à exploiter pédagogiquement.

PCVAction

Le `PCVAction` offre d'une part des moyens d'interaction entre l'apprenant et l'environnement et d'autre part détecte les actions sollicitées :

1. *Les moyens d'interaction*
   Dans un premier temps, il s'agit de proposer à l'apprenant des moyens d'interaction avec les objets de l'environnement virtuel correspondant à ses prises de décision. Ces interactions peuvent être régulées en s'adaptant à l'apprenant (en utilisant la base de connaissances sur l'apprenant et en sollicitant l'agent pédagogique). En effet, on peut penser qu'il ne faut pas surcharger un apprenant novice en lui proposant des décisions irréalisables sur le moment, alors que cela peut être intéressant pour un apprenant expérimenté.

   Pour prendre une décision, l'apprenant va solliciter des objets physiques en leur demandant l'ensemble des actions qu'ils proposent. Les actions sur les objets d'un environnement virtuel peuvent être variées (rotation, translation, déformation, *etc.*). Néanmoins, notre cadre de travail n'est pas la formation au geste technique, mais à la prise de décision. Les objets physiques possèdent des connaissances sur les agents qu'ils représentent et par conséquent sur les actions qu'ils peuvent effectuer [2]. Lorsque l'apprenant sollicite une action sur un de ces objets :

   (a) la `PCVAction` récupère les informations intrinsèques à l'objet (ensemble des actions possibles que peut effectuer l'entité en considérant le rôle à jouer ou en considérant tous les rôles),

   (b) elle va ensuite demander à l'agent pédagogique quelles sont les actions à proposer, à partir de telles informations et des caractéristiques de l'apprenant,

   (c) la `PCVAction` affiche les actions proposées.

   La `PCVAction` permet de faciliter la sélection des diverses tâches à effectuer. Nous mettons en place une « substitution sensori-motrice » [Lourdeaux, 2001] sous la forme

2. Nous envisageons d'intégrer une autre solution qui consiste à inspecter les opérations de l'objet et d'en déduire les actions dans la procédure.

d'un menu OSD [3]. Le `PCVAction` va afficher les diverses actions proposées à l'utilisateur.

2. *Les actions sollicitées*
Dans un second temps, il s'agit de détecter les actions sollicitées et effectuées par l'apprenant, afin de mettre à jour le modèle de l'apprenant. Ce dernier peut solliciter une action en sélectionnant une icône du menu par la souris. Il prend alors sa décision et sollicite une action. Le `PCVAction` récupère cette information et va la transmettre au reste du système. Le `LearnerAgent` récupère l'information de sollicitation et met à jour son curriculum. Après une analyse de la décision par l'`ErrorAgent`, l'agent pédagogique choisit d'autoriser ou non l'action. Retenons qu'une action erronée ne sera pas forcément interdite selon la stratégie de l'agent pédagogique. En effet, on peut penser qu'il peut être intéressant d'effectuer une action, même s'il y a une erreur, pour certains apprenants [4]. Le `PCVAction` repère l'information relative à l'exécution de l'action (l'action a été réalisée) et informe le `LearnerAgent` le cas échéant.

PCVObservation

Il est intéressant de connaître les éléments qui sont, ou ont été, dans le champ de vision de l'apprenant pour prendre une décision pédagogique. En effet, considérons par exemple que la pré-condition d'une action A soit la présence de l'objet B. Nous pouvons penser que l'assistance pédagogique peut prendre en compte le fait que l'apprenant ait eu la possibilité d'observer cet objet ou non.

La PCV d'observation, telle que nous l'avons définie, va permettre d'obtenir une liste des objets qui apparaissent dans le champ de vision de l'utilisateur, ainsi que leurs distances et leurs orientations par rapport à l'utilisateur. Bien qu'il soit réducteur de penser qu'un objet visible soit forcement perçu par l'apprenant, nous effectuons cette simplification pour notre modèle.

PCVMotion

Il n'est pas nécessaire de connaître en permanence la position exacte de l'apprenant dans l'environnement. En revanche, lors d'une tâche précise, les informations permettant de savoir si l'apprenant se déplace vers ou en dehors du lieu où se déroulera l'action, si l'apprenant s'approche d'un objet précis et à quelle allure, sont des données intéressantes pour la prise de décision pédagogique.

La PCV de déplacement répond à ces besoins en permettant :
– d'obtenir l'orientation moyenne de l'utilisateur depuis un point de départ donné ;
– de savoir quel est l'objet vers lequel l'utilisateur se dirige (moyenne relative aux déplacements) ;
– de connaître la vitesse instantanée de l'utilisateur ;

3. OSD : On Screen Display. Expression anglaise se référant à un appareil qui utilise l'écran de télévision pour afficher ses différents menus, afin de faciliter les réglages ou son utilisation courante. Concrètement l'élément en OSD est au premier plan en deux dimensions.

4. Nous nous intéressons plus loin au problème de la « récupération » sur erreur dans un EVF (mécanisme de retour arrière).

– de savoir les distances (sous forme de variables floues : au dehors, loin, près, au dedans) entre des zones et l'utilisateur.

3.7 Modèle du formateur

Le modèle du formateur fournit au formateur des possibilités en terme de pédagogie et définit l'exercice à réaliser. Un exercice est caractérisé par un environnement simulé faisant intervenir une ou plusieurs équipes qui effectuent une procédure collaborative.

Le modèle du formateur (`MF`) est la représentation du formateur dans la situation simulée. Il fixe « les règles du jeu », *i.e.* quelle est la ou quelles sont les procédures à effectuer, et le ou les rôles à jouer par tel apprenant (et par conséquent ceux qui seront joués automatiquement). De plus, le modèle du formateur offre également la possibilité d'intervenir par des moyens d'interaction privilégiés dans l'environnement au travers de l'agent formateur. Aussi, il offre une vue d'ensemble de la procédure à réaliser (qui a fait quoi) permettant un suivi de l'exercice courant. Enfin, il récolte les propositions d'assistances pédagogiques résultant du raisonnement de l'agent pédagogique.

Base de connaissances

Pour qu'un exercice ait lieu, il faut que le formateur détermine les consignes, *i.e.* les règles à respecter. La base de connaissances de l'agent formateur précise alors, pour chaque exercice à effectuer, les informations suivantes :
– l'équipe dans laquelle les apprenants évoluent,
– la procédure à réaliser,
– le(s) rôle(s) à jouer.

L'exercice étant précisé, il s'agit d'assister le formateur dans le suivi des activités au sein de la procédure à réaliser. Pour cela, le modèle maintient un suivi de la procédure, en considérant l'exercice courant, qui centralise les informations concernant les actions et les erreurs effectuées automatiquement par le système, ou réalisées par les apprenants.

Comportements

L'agent formateur (`TeacherAgent`) dispose de trois comportements qui offrent au formateur les possibilités suivantes :

1. *Intervenir dans l'environnement :*
   Le formateur est un utilisateur particulier et par conséquent, son statut doit pouvoir lui permettre d'intervenir dans l'environnement. Dans cette optique, l'agent formateur fournit des moyens d'interaction privilégiés avec l'application :
   – il offre la possibilité de prendre le contrôle d'agents, le formateur peut alors jouer un rôle dans l'équipe ;

– il permet de solliciter des opérations.
Exemples : Sélectionner un objet physique et le faire clignoter. Afficher sur l'ordinateur de l'apprenant un message, *etc.*

2. *Choisir parmi les propositions d'assistances pédagogiques :*
L'agent formateur propose les assistances pédagogiques qui sont élues par l'agent pédagogique. Le formateur fait son choix *via* l'agent formateur.
3. *Accéder au raisonnement pédagogique simulé :*
L'agent formateur donne accès au raisonnement pédagogique simulé *via* l'agent pédagogique. Le formateur peut alors essayer de comprendre pourquoi l'agent pédagogique a proposé telle ou telle assistance pédagogique.

Notre modèle du formateur reste actuellement très limité. En effet, nous avons borné notre étude aux besoins minimums. Néanmoins, des travaux futurs pourront exploiter ce modèle pour envisager un environnement qui tienne compte des préférences du formateur en terme de formation ou qui considère plusieurs formateurs.

3.8 Modèle pédagogique

Le modèle pédagogique représente l'apport majeur de Pegase. Il s'articule autour de la construction de la situation pédagogique qui sert de base de connaissance à l'agent pédagogique. Le comportement de cet agent se fonde sur ces connaissances et sur des connaissance d'experts de la pédagogie pour proposer des actions pédagogiques au formateur.

3.8.a. Situation pédagogique

Nous nous plaçons dans le cadre d'une formation au travail procédural. L'objectif de l'ITS est d'aider l'apprenant pour qu'il progresse dans la procédure. La compétence à acquérir porte donc sur la réalisation de la procédure dans un environnement dynamique.

Dans un premier temps, nous pouvons considérer la procédure comme un enchaînement d'actions défini par un expert du domaine. Les éléments à considérer sont donc sujets à un *agencement* qui n'est pas discutable et parfois non explicable. Dans un second temps, nous pouvons penser que mémoriser l'enchaînement des actions pourrait être facilité par sa compréhension. Dans ce cadre, [Richard, 1990] propose de se rattacher à la notion de sous-buts dans la procédure. Pour aboutir à l'objectif, c'est-à-dire la réalisation de la procédure, il faut effectuer un ensemble de sous-buts liés causalement. Il s'agit alors d'étudier la procédure en considérant la distance au but de la procédure d'un point de vue *causal*, et non plus d'un point de vue chronologique.

L'analyse précédente fait donc apparaître deux manières d'aborder l'apprentissage de procédures : l'étude des liens d'ordonnancement métier qui sont fortement liés aux rôles dans la procédure et l'étude des liens causaux entre sous-buts.

1. *Les liens d'ordonnancement*
Ils effectuent les relations entre les actions en utilisant la description stricte de la

procédure. Ils sont la conséquence-même de l'ordonnancement des actions qui est défini par l'expert. Nous considérons les informations liées aux actions qui sont au plus proches de l'action sollicitée par l'apprenant. Plus précisément, il s'agit de :
- la dernière action correcte avant celle qu'il vient juste de solliciter ;
- l'action qui vient d'être sollicitée par l'apprenant ;
- les actions correctes à effectuer en considérant le(s) rôle(s) à jouer (actions potentiellement différentes de l'action sollicitée) ;
- les actions correctes à réaliser en considérant que tous les rôles sont joués par l'apprenant ;
- les actions qui suivent toutes les actions correctes.

2. *Les liens causaux entre sous-buts*
La procédure peut être considérée comme un graphe représentant l'enchaînement des sous-buts causaux. Nous considérons alors toutes les actions liées à l'action effectuée par l'apprenant. Concrètement, il s'agit des actions nécessitant l'effet de l'action correcte (condition d'utilisation, état d'une ressource...). Il faut bien distinguer ces liens qui correspondent à une logique individuelle alors que les liens d'ordonnancement correspondent à l'organisation d'une procédure collective. Techniquement, il s'agit des liens entre postconditions et préconditions évoqués en section 2.5.

Ces informations sont résumées sur le tableau 3.1. Le « contexte d'action » est constitué des connaissances liées directement à l'`Action` (description, ressources...), des connaissances liées à l'`Operation` cible de l'`Action`, ainsi que les connaissances liées à l'agent qui a sollicité l'action puisqu'il est le protagoniste. Ainsi, nous utilisons les *contextes d'action* pour représenter les connaissances associées aux actions (sous-ensemble de l'environnement, constitué d'entités et d'agents, considéré comme pertinent dans le contexte de l'action).

Connaissances	**Nature**	**Description**
① Contexte d'action précédente	Ordonnancement	La dernière action correcte à avoir été réalisée. Cette action fait référence et permet de se positionner dans la procédure.
② Contexte d'action sollicitée	Ordonnancement	Il s'agit de l'action sollicitée. Cette action peut être correcte, comme erronée. Elle n'est pas forcément réalisée, conformément au modèle pédagogique.
③ Contexte d'action(s) correcte(s) sans considération sur les rôles	Ordonnancement	En considérant la dernière action correcte, nous déterminons les actions à effectuer selon la procédure courante.
④ Contexte d'action(s) correcte(s)	Ordonnancement	Il s'agit d'un sous-ensemble de l'item précédent, qui ne considère que les rôles joués par l'apprenant.
⑤ Contexte d'action(s) suivante(s)	Ordonnancement	Pour chaque action correcte, nous déterminons les actions qui suivent selon la procédure courante.
⑥ Contexte d'action(s) liée(s)	Causale	En considérant les actions à réaliser à la suite de la dernière action correcte, nous récupérons les liens « causaux » entre actions indépendamment de la procédure. Nous obtenons les actions qui sont liées.

Table 3.1 – La situation pédagogique : connaissances sur la tâche à réaliser

La construction de cet ensemble de connaissances est à la charge de l'agent pédagogique. L'agent pédagogique récupère ou construit les connaissances sur le travail à réaliser lorsqu'il

reçoit un message de l'agent d'interface qui précise qu'une action vient d'être sollicitée. Ce choix peut être discuté. En effet, une autre solution est de mettre à jour les connaissances sur erreur. Nous avons préféré reconstruire ces connaissances sur action pour offrir la possibilité d'intervenir, même si le comportement de l'apprenant est correct. Cela permet d'envisager des assistances pédagogiques qui le confortent dans ses décisions, qui l'encouragent ou qui tentent de semer un doute (exemple : affirmer des règles fausses qui entrent en contradiction avec ses choix).

La situation pédagogique contient également des informations sur l'apprenant. Notons que les erreurs effectuées par l'apprenant sont stockées, c'est-à-dire le type et éventuellement d'autres connaissances sur l'erreur. Ces données offrent, par exemple, la possibilité de vérifier la fréquence de telle ou telle erreur, ou de se rendre compte que l'apprenant effectue la plupart du temps des erreurs de type agencement. En outre, il est nécessaire de montrer pourquoi l'apprenant a fait une erreur. Pour ce faire, nous avons modélisé la méthode CREAM (*Cognitive Reliability and Error Analysis Method*). Nous avons intégré le mécanisme d'analyse rétrospective de CREAM pour que le système puisse indiquer les liens causaux les plus probables expliquant l'occurrence des erreurs. Pour plus de détails sur le modèle des erreurs le lecteur pourra se référer à [Trinh et al., 2009].

De la même manière, les contextes associés aux actions sont conservés. Ces informations permettent, par exemple, de savoir si l'apprenant a déjà utilisé telle ou telle ressource.

Concrètement, nous venons de définir les informations disponibles en entrée et les éléments définis comme pertinents sur lesquels on peut agir en sortie pour une décision pédagogique.

3.8.b. Agent pédagogique

La situation pédagogique offre la possibilité de déclencher des assistances pédagogiques sur les éléments qu'elle contient, elle fournit alors les sorties possibles de la prise de décision pédagogique. Il s'agit à présent de définir un modèle simulant le comportement décisionnel de l'agent pédagogique qui fournit les propositions d'assistance pédagogique, c'est-à-dire un modèle qui fait le lien entre les connaissances et les propositions d'assistances. Rappelons que nous nous plaçons dans le cadre de l'apprentissage de tâches procédurales et collaboratives. Nous devons considérer :

- l'hétérogénéité et la nature des connaissances impliquées (connaissances issues de la pédagogie fondamentale jusqu'à la réalité virtuelle) ;
- les capacités d'adaptation (le raisonnement doit s'automodifier pour prendre en compte les expériences passées) ;
- le raisonnement doit pouvoir être spécifié *a priori* (la spécification initiale peut alors être réalisée par un pédagogue).

Les critères qui en découlent sont : expressivité, hiérarchisation, modularité, réactivité et adaptabilité.

Après nous être intéressés aux familles d'architectures comportementales existantes (connexionistes, à base d'automates, à base de règles), nous avons opté pour celles qui utilisent des règles qui répondent au mieux aux critères précédents. Plus précisément, nous avons choisi les

systèmes de classeurs [Buche et al., 2006]. Il s'agit d'une architecture réactive et adaptative, fondée sur des règles conditionnelles.

Nous proposons un modèle basé sur un système de classeurs hiérarchiques. Il structure les connaissances sur trois niveaux, allant des règles fondées sur des connaissances abstraites de la pédagogie (les démarches pédagogiques) jusqu'aux règles fondées sur des connaissances concrètes de la réalité virtuelle (les techniques pédagogiques), en passant par un niveau intermédiaire (les attitudes pédagogiques).

Chaque niveau d'abstraction contient des ensembles qui regroupent plusieurs règles. Un ensemble représente une façon d'aborder une démarche, une attitude ou une technique pédagogique. Les règles sont conditionnées par les éléments de la situation pédagogique et ont pour effet de favoriser des ensembles du niveau inférieur. Le système utilise alors un mécanisme de diffusion dans les trois niveaux qui considère les règles appariées par la situation pédagogique. Il aboutit à une liste qui ordonne les propositions d'assistance pédagogique.

La figure 3.3 illustre la structure et la dynamique du modèle pédagogique contrôlant le comportement de l'agent pédagogique. Les informations prises en considération dans les parties conditions des règles sont procurées par notre ITS (situation pédagogique). Ces "entrées" sont disponibles sur les trois niveaux d'abstraction des données (démarches, attitudes et techniques pédagogiques). Les règles dont la partie condition est satisfaite par rapport aux entrées, favorisent certains ensembles de règles pédagogiques du niveau inférieur. Le dernier niveau (techniques) favorise directement des assistances pédagogiques applicables dans l'environnement. Ces dernières sont proposées au formateur qui choisit celle qui est, selon lui, la plus appropriée.

Pour mettre en œuvre le modèle pédagogique, le travail du pédagogue est de spécifier :

1. les ensembles de règles pour les trois niveaux d'abstraction,
2. les règles pédagogiques pour chaque ensemble de règles.

Pour spécifier les ensembles de règles pédagogiques, nous nous basons sur les références [Lourdeaux, 2001, Burkhardt et al., 2003]. Nous obtenons les tableaux 3.2, 3.3 et 3.4 correspondant respectivement aux trois niveaux (démarches, attitudes et techniques).

Une règle est représentée par une chaîne de caractères. Les parties *condition* et *effet*, portent sur les éléments de la situation pédagogique. Dans l'exemple suivant, nous nous plaçons au niveau d'abstraction `Démarches Pédagogiques`, nous avons un ensemble de règles appelé `Active`. Une première règle de cet ensemble est satisfaite si l'apprenant est novice (`Apprenant.Niveau==novice`), si il vient d'effectuer une erreur de type agencement (`Apprenant.Erreur.type==procédural`) et si l'action effectuée est différente de l'action correcte (`! Travail.ActionSollicitée in Travail.ActionCorrectes`). Dans ce cas, la règle favorise l'ensemble `Expliquer` du niveau qui suit.

```
if (Apprenant.Niveau == novice &&
    Apprenant.Erreur.type==procédural &&
    ! Travail.ActionSollicitée in Travail.ActionCorrectes)
then (Expliquer)
```

Démarches pédagogiques	Description
Active/Constructiviste	Les démarches actives sont centrées sur l'apprenant, considérant qu'il est acteur principal de son apprentissage. Elles lui proposent des techniques au travers desquelles il est amené à produire, à créer, à chercher. Le savoir est dans l'environnement.
Expositive/Affirmative	C'est la démarche la plus traditionnelle qui utilise la technique de l'exposé. Elle repose sur une démarche de transmission de contenus. Le savoir est externe.
Interrogative	Elle préconise de diriger l'apprenant vers les solutions recherchées. L'apprenant peut avoir l'impression de découvrir quelque chose, mais c'est toujours le formateur qui conduit la réflexion. Le savoir est interne.

Table 3.2 – Exemples de définitions des ensembles pour le niveau d'abstraction « Démarches pédagogiques », en se basant sur [Lourdeaux, 2001, Burkhardt et al., 2003]

Attitudes pédagogiques	Description
Réaliser	Réaliser à la place des formés. Cette attitude permet au formateur de montrer, par exemple, le bon geste ou la bonne technique.
Perturber	Certains formateurs perturbent les formés, en donnant de mauvaises informations ou des solutions potentiellement fausses, afin de tester la confiance des formés dans leurs raisonnements.
Suggérer	Montrer où les formés peuvent trouver les connaissances théoriques ou bien, où trouver les connaissances sur le terrain. Ces attitudes permettent au formateur de montrer aux formés qu'ils peuvent trouver les connaissances par eux-mêmes, et donc, traiter la situation calmement.
Laisser faire	Cette attitude propose au formateur de ne pas intervenir, mais plutôt de rester en tâche de fond, en tant qu'observateur.
Expliquer	Les informations ont aussi pour but de d'expliquer le fonctionnement de certains appareils, les règles de raisonnement, les règles de sécurité, etc.
Encourager	Encourager les formés lorsqu'ils réalisent correctement la tâche.

Table 3.3 – Exemples de définition des ensembles pour le niveau d'abstraction « Attitudes pédagogiques », en se basant sur [Lourdeaux, 2001]

Techniques	Description pédagogiques
Enrichissement	Ajout de symboles visuels, sonores ou de films d'animation.
Dégradation	Détérioration du réalisme (repères effacés, *feed-back* proprioceptifs dégradés, couleurs atténuées, flous à l'arrière-plan et/ou sur les côtés, réduction d'objets, icônisation, etc.).
Rehaussement	Exagération de la réalité (représentation d'objets à plus grande échelle, objets surréalistes, plus lumineux, plus brillants, etc.).
Simplification	Allègement de la scène virtuelle (une foule peut être représentée par des personnes aux mouvements simplifiés, objets simplifiés, systèmes kinesthésiques simplifiés, représentation en fil de fer, etc.), représentation schématique de certains appareils.
Restriction	Limitation de certains déplacements ou manipulations (limitations du périmètre dans lequel l'utilisateur peut se déplacer, etc.)
Animation	Séquence animée (positionnement automatique, clef qui tourne automatiquement une fois mise en place, etc.).
Décentrage	Changement du point de vue habituellement attaché à l'œil du formé immergé (vue de derrière, au-dessus, etc.).
Modification	Modification d'aspect, de texture (changement de couleurs, clignotement d'objets, etc.).
Modélisation	Représentation de concepts abstraits, de phénomènes physiques invisibles à l'œil nu, de type de pannes, etc.
Visualisation	Mécanismes cachés (intérieur d'un moteur, engrenages, etc.).

Table 3.4 – Exemples de définitions des ensembles pour le niveau d'abstraction « Techniques pédagogiques », en se basant sur [Lourdeaux, 2001]

3.8.c. Apprentissage

L'apprentissage va permettre de raffiner les poids des règles pour s'adapter aux habitudes du formateur et imiter ainsi son expertise.

L'algorithme d'apprentissage distribue les rétributions aux règles qui ont permis de les obtenir. Il est initialement adapté aux systèmes de classeurs [Buche et al., 2006] possédant une file de règles dont l'enchaînement produit une action. Dans notre cas, l'enchaînement correspond au passage entre les niveaux hiérarchiques. La rétribution est simplement reflétée par le choix du formateur : la technique pédagogique qu'il sélectionne définit les règles du niveau trois qui vont être rétribuées. Par chaînage arrière, les règles du niveaux 2 et 1 sont également rétribuées. Les règles qui s'apparient avec la *situation pédagogique*, mais qui participent à l'activation d'une autre technique pédagogique que celle choisie par le formateur, voient leur poids diminuer. L'algorithme effectue en fait un partage de la rétribution, incluant une taxe permettant de ne pas désavantager les règles rarement appariées et pénalisant plus intensément les règles fortes pour préserver l'adaptativité du système.

Ainsi, au fur et à mesure des exercices, l'agent pédagogique doit proposer des choix de plus en plus en accord avec les décisions du formateur. L'agent pédagogique pourrait alors, dans le cadre d'un apprentissage multi-apprenant, prendre le relais temporairement et appliquer directement les assistances qu'il a choisies.

3.9 Conclusion

Nous avons montré que, même dans les systèmes existants les plus aboutis, le modèle pédagogique est en partie dépendant de l'exercice. Plus précisément, les erreurs et les stratégies pédagogiques doivent être redéfinies pour chaque exercice et domaine d'application. De plus, les stratégies pédagogiques sont imposées au formateur ou tout du moins le choix est très limité.

Sans revenir sur chaque point de nos travaux, nous pouvons montrer en quoi notre proposition répond aux difficultés des modèles existants. Les connaissances utilisées dans le raisonnement pédagogique ne portent pas sur une particularité de l'exercice à réaliser. Ainsi, une règle pédagogique ne considère pas des informations spécifiques, « si l'apprenant peut observer l'avion 2 alors... », mais utilisera plutôt des connaissances génériques indépendantes de l'exercice (« si les ressources des actions correctes sont visibles alors... »). De la même manière, les assistances pédagogiques, bien que proposant des solutions concrètes au formateur, « faire clignoter le pompier », manipulent également des connaissances génériques indépendantes de l'exercice (« Faire clignoter les protagonistes des actions suivantes »). Ainsi, la généricité de notre proposition est une caractéristique forte qui est illustrée par l'intégration de notre ITS au sein de plusieurs applications : apprentissage de procédures collaboratives sur porte-avions (Gaspar) [Marion et al., 2007] et sur site SEVESO pour les pompiers (SecuReVi) [Querrec et al., 2004]. Ceci est obtenu en s'appuyant sur le métamodèle Mascaret. Bien plus, le modèle pédagogique de notre ITS présente une forte modularité, puisqu'il offre la possibilité d'ajouter, de supprimer ou de modifier chacune des composantes

du modèle pédagogique participant à la prise de décision pédagogique (règle ou ensemble de règles). De surcroît, le mécanisme d'apprentissage artificiel adapte les propositions d'assistances pédagogiques au couple apprenant-formateur, c'est à dire que Pegase est capable d'apprendre les assistances pédagogiques qu'un formateur donné aurait choisit pour un apprenant donné.

Toutefois, nous pouvons nous interroger sur les possibilités liées à l'utilisation de notre ITS dans le cadre d'un apprentissage non procédural. Envisager ce type de formation impose de repenser les éléments qui sont étroitement liés à la notion de procédure, *i.e.* les connaissances de la situation pédagogique. Un autre point sur lequel nous comptons travailler pour améliorer Pegase est de lui fournir un comportement plus délibératif. En effet Pegase agit en réaction d'une action (ou inaction) d'un apprenant. Il serait souhaitable qu'il puisse élaborer un plan plus complexe et s'inscrire dans un scénario pédagogique structuré. Ce scénario pédagogique est un des instruments de la relation didactique du triangle pédagogique. Notre proposition de scénario est détaillé dans le chapitre suivant.

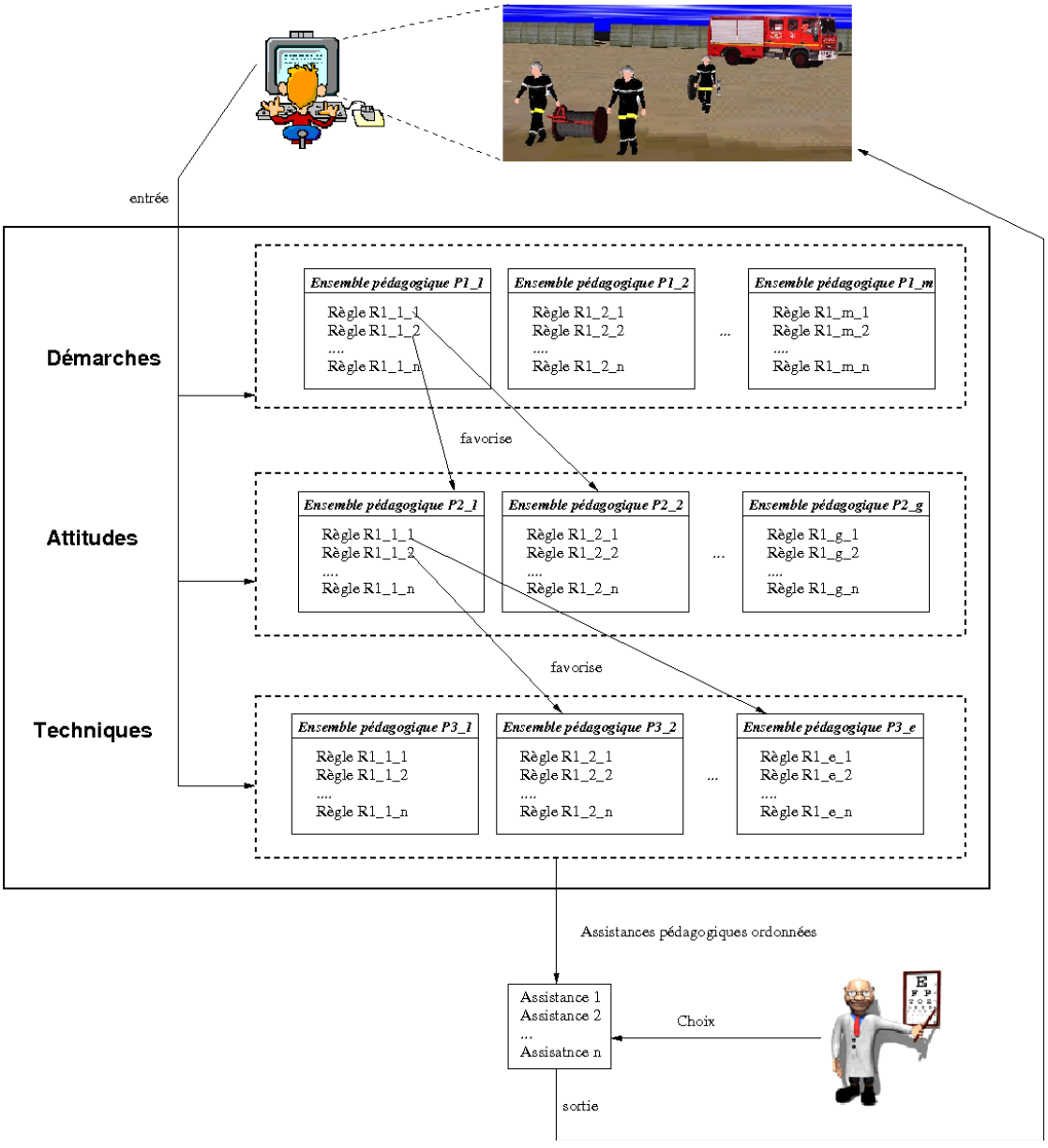


Figure 3.3 – Représentation complète du modèle pédagogique

Chapitre 4

Modélisation de la relation didactique

Nous nous intéressons ici à l'activité du formateur qui souhaite concevoir et mettre en place des sessions d'apprentissage dont les activités se déroulent en environnement virtuel. Nous proposons un modèle pour instrumenter la relation didactique entre le formateur et la compétence (figure 4.1). Ce travail a donné lieu à la thèse de Nicolas Marion [Marion, 2010, Marion et al., 2009] co-encadrée avec Pierre Chevaillier. Le résultat de ce travail peut enrichir les connaissances de Pegase, l'Its que nous avons proposé au chapitre pécédent. Dans ce cadre, le rôle du formateur consiste à préparer et exploiter les interactions entre l'apprenant et le dispositif d'apprentissage [Henri et al., 2007]. Pour cela, il écrit un scénario pédagogique dont l'exécution doit permettre la construction des connaissances visées chez les apprenants. Un scénario pédagogique contient généralement [Koper and Manderveld, 2004] :

- les objectifs pédagogiques qui décrivent les connaissances ou compétences que les apprenants sont supposés avoir acquies en fin de scénario ;
- les prérequis qui définissent les connaissances ou les compétences que les apprenants doivent posséder pour tirer parti du scénario pédagogique ;
- les activités pédagogiques et leur enchaînement ;
- les rôles qui décrivent la participation des utilisateurs aux activités pédagogiques ;
- les environnements, qui décrivent les outils et ressources nécessaires à la réalisation des activités pédagogiques par les utilisateurs.

4.1 Positionnement

Dans le domaine des EIAH, il existe de nombreux modèles de scénario pédagogique. Une des plus importantes propositions de la communauté est la publication de la norme IMS Learning Design, ou IMS-LD [1]. Dans IMS-LD, un scénario est considéré comme un enchaînement d'activités pédagogiques. Chacune de ces activités est décrite par un texte ou un ensemble de documents expliquant le but de l'activité, la tâche à réaliser, les consignes à respecter, etc. Du fait de cette description textuelle, le déroulement des activités pédagogiques est considéré du point de vue de la plateforme exécutant le scénario comme une « boîte noire » dont on ne connaît que les entrées (ressources) et sorties (productions). Si ce type de description peut

1. http://www.imsglobal.org/learningdesign/

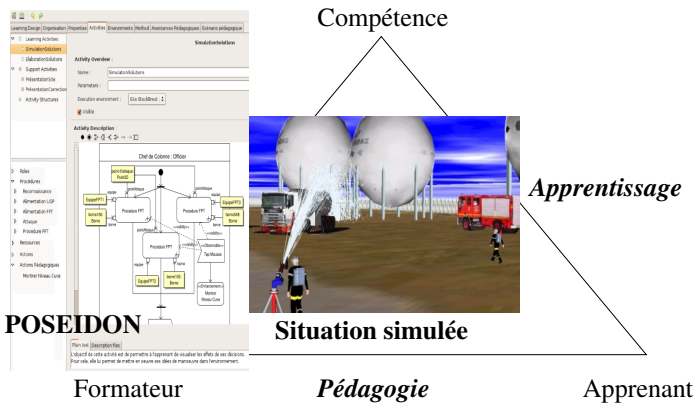


Figure 4.1 – Outil de scénarisation.

être suffisant pour la réalisation de tâches ne nécessitant pas le contrôle de l'activité des apprenants (par exemple remplir des questionnaires ou réaliser des calculs), cela ne convient pas pour décrire des activités pendant lesquelles les apprenants sont en forte interaction avec l'environnement [Guéraud and Cagnat, 2006] comme c'est le cas avec les EVAH.

Toutefois il existe des modèles utilisant la réalité virtuelle ou du moins des simulations informatiques dans un cadre de pédagogie. Cependant, en général, le langage pour décrire le scénario pédagogique est soit très limité soit très dépendant de l'application dans lequel il est implanté. Ainsi le langage utilisé dans GVT [Gerbaud et al., 2008, Cazeaux et al., 2005] pour décrire le scénario porte certes sur un domaine assez vaste que sont les activités procédurales et collaboratives. Cependant les assistances pédagogiques disponibles sont très limitées et ce modèle de scénario n'offre aucune extension possible en l'état. D'autres modèles tels que Formid [Guéraud et al., 2004, Guéraud and Cagnat, 2006] ou Fiacre [Lourdeaux, 2001] offrent des capacités intéressantes (dont nous nous inspirons ici) mais le langage mêle la description de l'environnement de simulation et la description du scénario pédagogique. Ceci empêche d'utiliser un même scénario dans le contexte d'un autre environnement.

Dans le domaine de la simulation, des modèles tels que SimQuest décrivant le système par un ensemble d'équations [Van Joolingen and de Jong, 2003], proposent de construire le scénario pédagogique à partir du modèle du domaine. C'est la démarche que nous adoptons ici, mais dans un autre domaine que la simulation numérique. Dans Rides [Munro et al., 1997], le formateur conçoit le scénario en réalisant la procédure qui devra être apprise par l'apprenant. Ceci facilite la conception du scénario, mais conduit à des scénarios complètement linéaires et sans alternatives.

Nous proposons Poseidon (PedagOgical ScEnario for vIrtual and informeD envirONment), un modèle de scénario pédagogique générique pour les EVAH. Poseidon rend possible la description d'activités pédagogiques permettant le contrôle de l'activité d'apprenants en interaction au sein d'un environnement virtuel, de manière générique dans le sens où il n'est pas spécifique à un domaine d'étude, à une stratégie pédagogique ou à une plateforme d'exécution. Pour atteindre ce niveau de généricité, Poseidon s'appuie sur une représentation abstraite des environnements virtuels : Mascaret.

4.2 Vue d'ensemble de Poseidon

La figure 4.2 montre le processus de création d'un scénario pédagogique à l'aide de Poseidon. À partir de la description du modèle, le formateur instancie un environnement spécifique destiné à simuler la situation d'apprentissage. L'objectif de Poseidon est de fournir au formateur un langage lui permettant de décrire un scénario pédagogique tel que défini précédemment. Ce langage doit être indépendant du modèle métier abordé. Poseidon s'appuie alors sur les méta-modèles proposés dans Veha, Have et Behave.

La figure 4.3 illustre les différents niveaux d'abstraction utilisés. Si l'on reprend les trois niveaux de modélisation identifiés précédemment, Mascaret joue le rôle de méta-modèle d'EVAH de niveau M2. Ce méta-modèle définit le langage de modélisation utilisé pour créer différents modèles M1 correspondant aux différents domaines. Une instanciation de ce modèle résulte en un environnement virtuel (niveau M0) contenant des entités, instances des concepts définis en M1. Pour les niveaux M1 et M0 nous prenons un exemple issue des travaux de [Baudouin et al., 2008] sur la « paillasse virtuelle » menés au CERV à l'aide de Mascaret. Sur la droite de la figure sont représentés les différents niveaux du modèle de scénario pédagogique que nous proposons. Le modèle de scénario s'appuie sur Veha, Have et Behave et se place donc au même niveau d'abstraction. Ceci lui permet d'avoir accès aux concepts manipulés dans un environnement virtuel. Le modèle de scénario permet la création d'un scénario abstrait qui utilise le modèle métier spécifique et référence un environnement virtuel particulier. A ce stade ce scénario est abstrait (M1) (ou « scénario type » [Pernin and Lejeune, 2004]), car il peut conduire à plusieurs exécutions (scénarios concrets de niveau M0 ou scénario contextualisé [Pernin and Lejeune, 2004]) selon les attributions des rôles et la plateforme d'exécution choisie.

Le diagramme de classe de la figure 4.4 présente une vue synthétique du contenu d'un scénario pédagogique Poseidon.

Le concept central du modèle est la **session d'apprentissage** (`LearningSession`). C'est l'entité qui agrège les différents composants du scénario pédagogique. Elle est comparable à l'unité d'apprentissage d'IMS-LD. Comme nous pouvons le voir sur ce diagramme, une session contient :

- des prérequis. Il peut s'agir d'un ensemble de sessions que l'apprenant doit avoir réalisé au préalable (relation `prerequisite`), ou d'une description textuelle de ce qu'il doit savoir ou savoir faire. Dans le premier cas, le prérequis peut être vérifié par la plateforme à partir du cursus de l'apprenant. Dans le second cas, c'est à un formateur humain de vérifier le prérequis.
- des objectifs pédagogiques. Il s'agit ici d'un texte décrivant l'objectif de compétence visé par la session d'apprentissage. Les objectifs de réalisation sont décrits pour chaque activité pédagogique, au niveau du scénario (`EducationalScenario`).
- des environnements d'apprentissage (`EducationalEnvironment`), qui contiennent les ressources nécessaires à la réalisation des activités pédagogiques. Le modèle d'environnement est présenté dans la section 4.3.
- une organisation pédagogique (`EducationalOrganisation`), qui définit les différents rôles qui devront être joués dans le scénario pédagogique par les acteurs de la formation (apprenants ou formateurs). Le modèle d'organisation pédagogique est décrit dans la section 4.4.

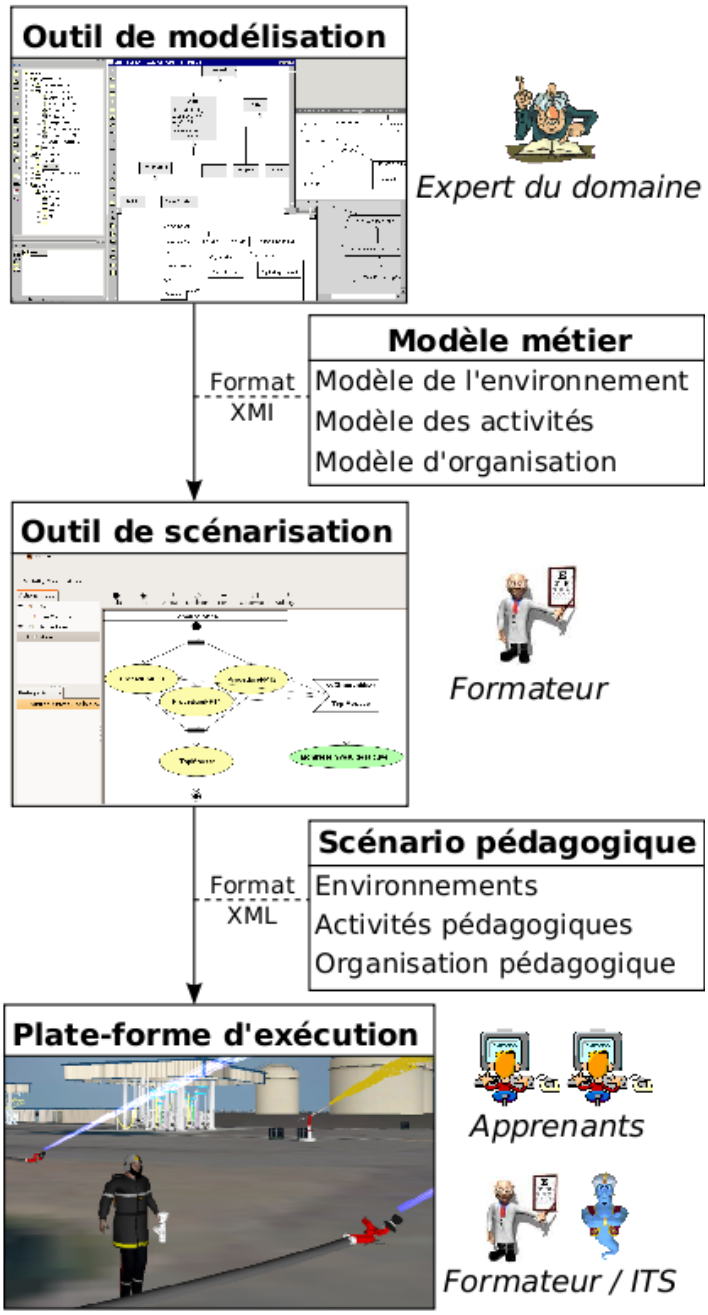


Figure 4.2 – Processus de création d'un scénario pédagogique Poseidon.

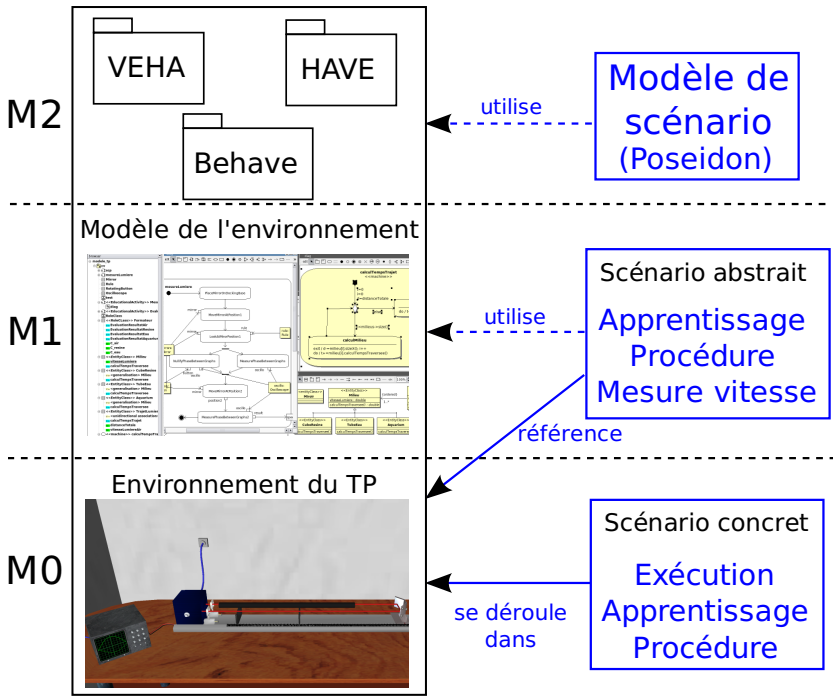


Figure 4.3 – Les trois niveaux de modélisation : un exemple.

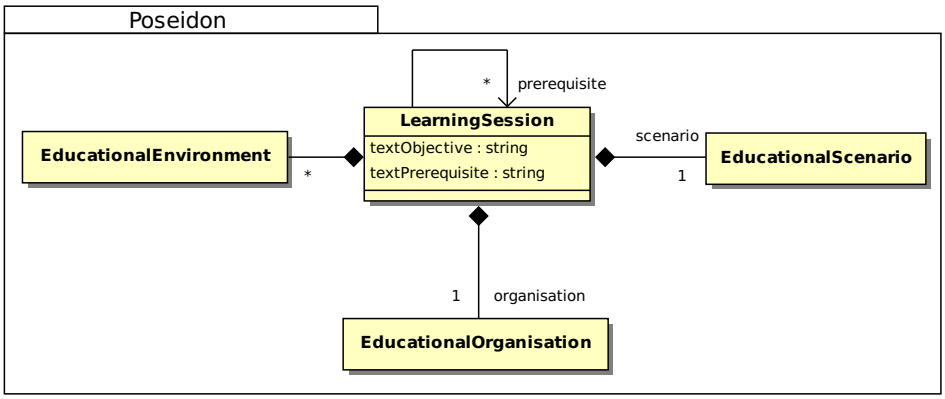


Figure 4.4 – Diagramme de classes synthétique des principaux éléments du modèle.

– un scénario pédagogique (`EducationalScenario`), qui décrit les activités pédagogiques ainsi que leur enchaînement. Les activités pédagogiques représentent les activités d'apprentissage que devront réaliser les apprenants, ou encore les activités de soutien qui pourront être réalisées par un membre de l'équipe pédagogique. Cette partie du modèle est détaillée dans la section 4.5.

La figure 4.5 présente un diagramme de classes (niveau M1) illustrant une partie d'un exemple de scénario abstrait.

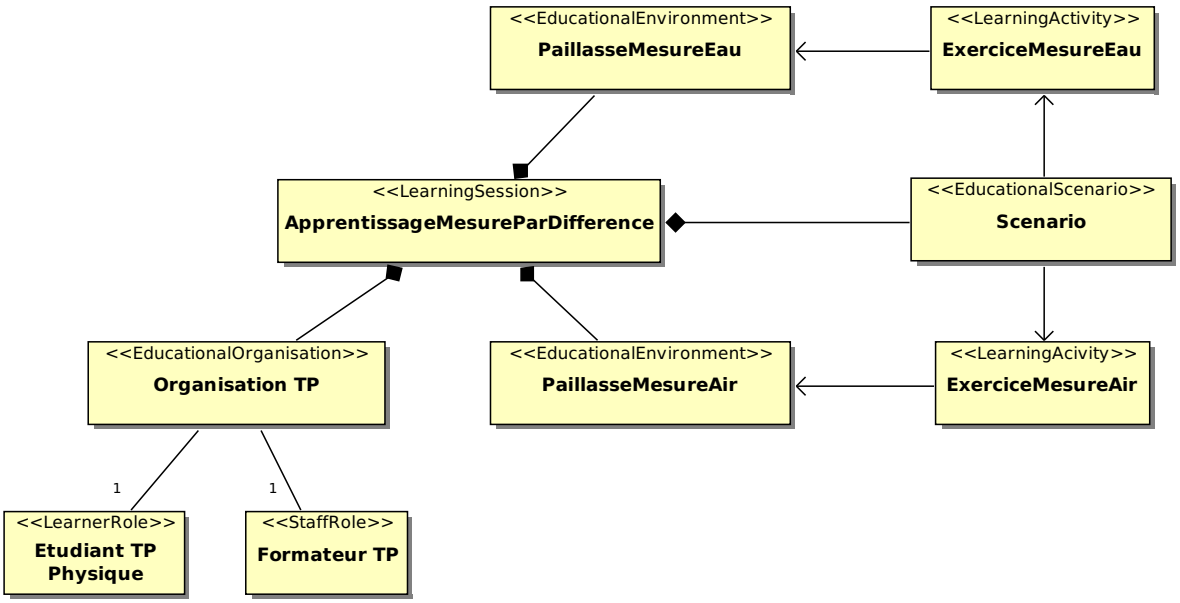


Figure 4.5 – Diagramme d'objets d'un exemple de scénario abstrait.

Sur ce diagramme, on voit que les concepts définis au niveau du modèle ont été instanciés pour créer un exemple. Ainsi, `ApprentissageMesureParDifference` représente la session d'apprentissage, qui contient une organisation pédagogique composée de deux rôles : un apprenant (`LearnerRole`) et un formateur (`StaffRole`). La définition de ces deux types de rôles s'appuie sur IMS-LD et est décrite dans la section 4.4. La session d'apprentissage est également composée d'un scénario pédagogique (`EducationalScenario`) qui contient deux activités pédagogiques (`LearningActivity`, *cf* section 4.5) se déroulant chacune dans un environnement (`EducationalEnvironment`).

Notons que la figure 4.5 décrit la représentation interne du scénario par le système. Ce n'est pas cette représentation peu synoptique que les utilisateurs du modèle seront amenés à manipuler. Pour créer un scénario pédagogique Poseidon, un outil-auteur est proposé. Cette remarque est valable pour la plupart des composantes du scénario présentées dans ce chapitre.

4.3 Description des environnements

Dans les modèles de scénario pédagogique utilisés en EIAH (dont IMS-LD), un environnement d'apprentissage consiste en un ensemble de ressources pédagogiques (fichiers, services, *etc*). Un scénario décrivant des activités se déroulant en environnement virtuel doit prendre en compte les caractéristiques de cet environnement pour que le formateur puisse par exemple :

- décrire la situation initiale des entités de l'environnement virtuel lors du démarrage de l'activité pédagogique ;
- définir l'objectif de réalisation de l'activité (état final de l'environnement) ;
- spécifier des observables portant sur les entités de l'environnement, afin d'exprimer des situations particulières et éventuellement déclencher les actions adéquates.

Dans le modèle Poseidon, un environnement d'apprentissage tient compte de deux types de composants : l'**environnement virtuel** et les **ressources pédagogiques**, et assure le lien entre les deux. Un environnement d'apprentissage référence une instance de la classe `VEHAApplication`, qui contient un ensemble d'entités définissant l'environnement virtuel (`InstanceSpecification`) et le modèle du domaine de cet environnement (`Model`). La figure 4.6 représente le diagramme de classe de cette partie du modèle.

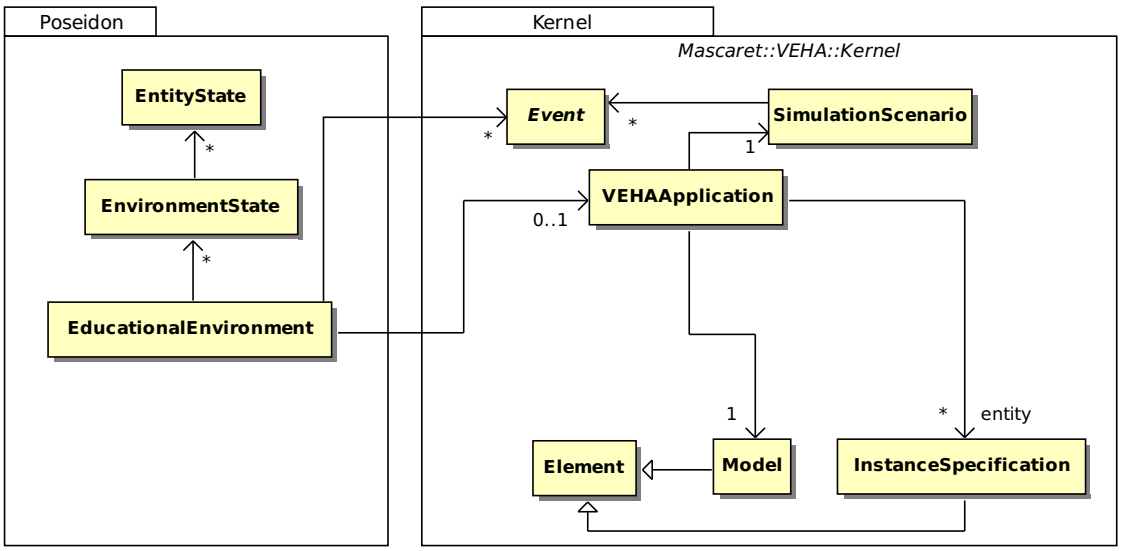


Figure 4.6 – Lien entre l'environnement d'apprentissage et l'environnement virtuel.

Cette référence à l'environnement dans le scénario pédagogique est utilisée pour **inspecter** les propriétés des entités de l'environnement pendant la simulation ou **modifier** ces propriétés. Par exemple, il est possible, lors de la conception du scénario, d'exprimer des situations particulières, sous forme d'événements (`Event`) ou sous forme d'états (`EnvironmentState`), à la manière de FORMID [Guéraud et al., 2004]. Ces états peuvent ensuite être utilisés dans le scénario pour décrire une situation initiale, un objectif de réalisation (situation finale), une erreur classique de l'apprenant, ou simplement un état dont le formateur veut être averti de l'occurrence.

Les **états de l'environnement**, quant à eux, contiennent les valeurs des propriétés de toutes les entités de l'environnement à un instant t. Pour chaque entité, l'état de l'environnement `EnvironmentState` précise si l'état de celle-ci est **pertinent**. En effet, s'il est important de connaître les valeurs des propriétés de toutes les entités pour pouvoir appliquer l'état global à un environnement (situation initiale ou retour en arrière par exemple), le formateur peut vouloir considérer cet état comme étant atteint par un apprenant en ne prenant en compte que l'état de certaines entités (un état global étant difficile à reproduire et souvent peu pertinent). De la même manière, l'état d'une entité `EntityState` décrit la valeur de toutes ses propriétés (attributs, machines à états, relations). Pour les mêmes raisons que pour l'état de l'environnement, chacune de ces propriétés peut être considérée comme pertinente ou non.

Un environnement d'apprentissage contient également des ressources pédagogiques, qui peuvent être des ressources nécessaires à la réalisation de l'activité, ou des ressources utiles à la compréhension de l'apprenant. Ces ressources peuvent prendre une multitude de formes : document de cours, questionnaire, calculatrice, etc. À la manière d'IMS-LD, le modèle Poseidon gère l'intégration de ces ressources dans l'environnement d'apprentissage. La réification au sein du scénario pédagogique de l'environnement virtuel et de son modèle nous permet de créer un lien entre une ressource pédagogique et des éléments Veha. Ce lien est représenté sur la figure 4.7.

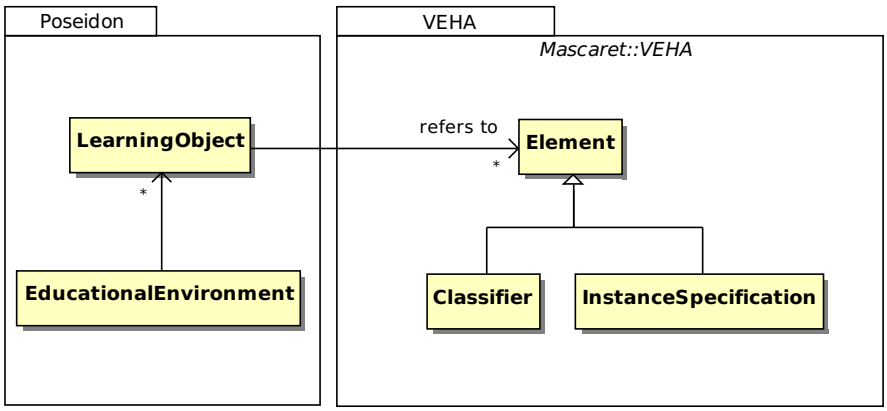


Figure 4.7 – Lien entre les ressources pédagogiques et les éléments de l'environnement virtuel (modèle Veha simplifié).

Sur cette figure, nous voyons qu'une ressource pédagogique (`LearningObject`) peut référencer un ou plusieurs éléments (`Element`) Veha. Ces éléments peuvent être un élément du modèle (la ressource contient des informations qui portent sur un concept, une classe d'objets ou encore une procédure par exemple) ou une entité de l'environnement. À l'heure actuelle, des méta-données de type LOM [2] prennent en charge ces informations.

2. Learning Object Metadata : http ://ltsc.ieee.org/wg12/

4.4 L'organisation pédagogique

Au sein d'une session d'apprentissage, différents acteurs (humains ou agents artificiels) interagissent. Ces acteurs jouent des **rôles** distincts : apprenant, formateur, accompagnateur, perturbateur, *etc* [Chou et al., 2003, Payr, 2003]. Un des principaux buts du scénario pédagogique est d'organiser les activités des différents acteurs qui y prennent part, et par conséquent, d'organiser les activités des rôles pédagogiques que ces acteurs vont jouer au sein d'une organisation pédagogique. Notre proposition est basée sur le modèle d'organisation de Mascaret : `Behave`.

Le but d'un rôle pédagogique est de faire le lien entre les acteurs du scénario et les activités pédagogiques qu'ils doivent réaliser. Comme dans IMS-LD, nous distinguons deux types de rôles : apprenant (`LearnerRole`) et encadrement (`StaffRole`). Nous nous appuyons sur `Behave` pour définir la notion de rôle pédagogique. Un `EducationalRole` est donc une sorte de `Role` (figure 4.8). Un rôle étant défini par son type (`RoleClass`), nous avons créé deux nouveaux types, `LearnerRole` et `StaffRole`, sous-classes de `RoleClass`. La principale raison

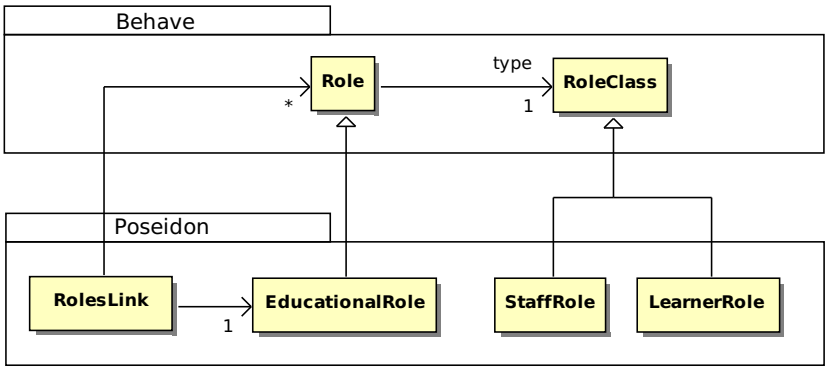


FIGURE 4.8 – Le modèle de rôle pédagogique.

justifiant la création de la classe `EducationalRole` concerne la différenciation entre les rôles et les rôles pédagogiques par la classe `RolesLink`. Cette classe exprime le fait que dans le cadre d'une activité pédagogique en environnement virtuel, un acteur du scénario pédagogique peut jouer des rôles (définis dans le modèle métier) au sein de cet environnement.

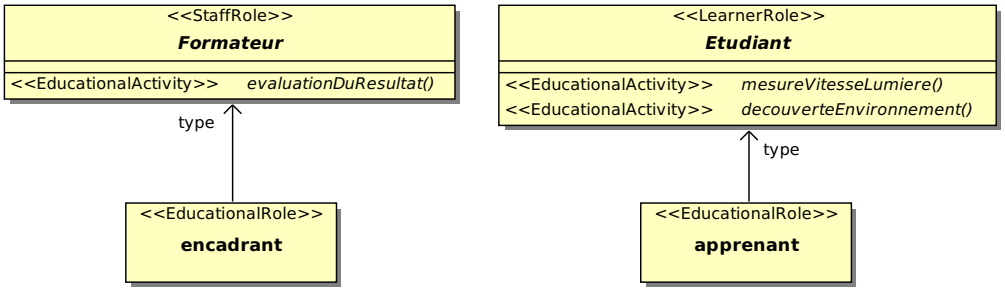


FIGURE 4.9 – Un exemple de définition de rôles pédagogiques.

La figure 4.9 montre un exemple de rôles pédagogiques créés au niveau M1. Dans cet exemple, deux types de rôles (instances de `RoleClass`) ont été créés :

– un rôle `Formateur`, de type `StaffRole`. Pour pouvoir jouer ce rôle, il faut être capable de réaliser l'activité pédagogique *evaluationDuResultat*.
– un rôle `Etudiant`, de type `LearnerRole`. Pour pouvoir jouer ce rôle, il faut être capable de réaliser les activités pédagogiques *decouverteEnvironnement* et *mesureVitesseLumiere*.

Lors d'une session d'apprentissage, les rôles pédagogiques sont organisés au sein d'une organisation pédagogique. Comme nous l'avons expliqué précédemment, Mascaret distingue les concepts de **structure organisationnelle** et d'**entité organisationnelle**. Nous définissons une organisation pédagogique comme étant une structure organisationnelle dont les rôles sont des rôles pédagogiques. Une entité organisationnelle représente une instanciation de cette structure. C'est donc une organisation concrète dans laquelle les rôles sont affectés (à des apprenants, des formateurs, des agents, *etc*). La figure 4.10 présente le modèle correspondant. Une organisation pédagogique (`EducationalOrganisation`) hérite des propriétés

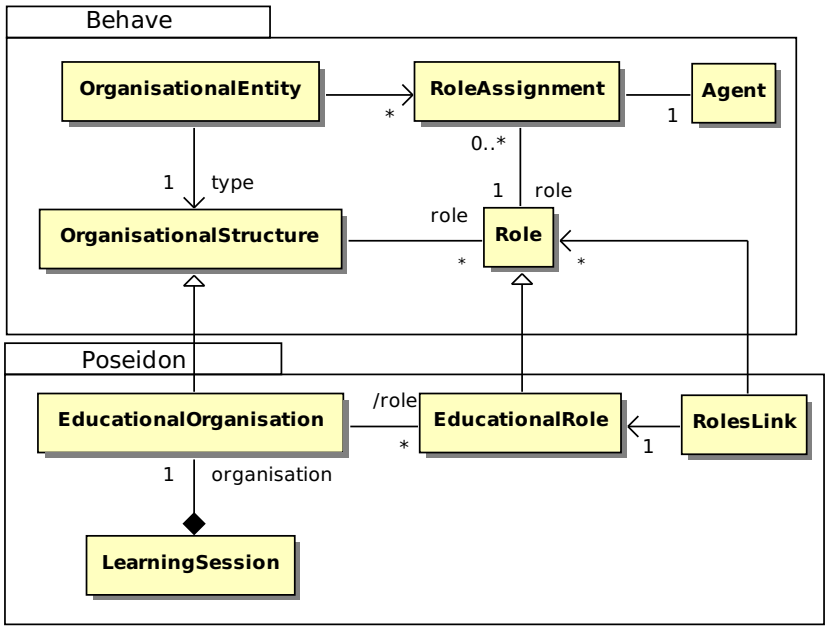


Figure 4.10 – Le modèle d'organisation pédagogique.

d'une organisation Mascaret. Sa spécificité est qu'elle contraint les rôles à être de type `EducationalRole`.

La figure 4.11 montre un exemple de structure organisationnelle pédagogique. Elle décrit un patron d'organisation contenant trois rôles : encadrant et apprenant, définis dans l'exemple de la figure 4.9, ainsi qu'un rôle accompagnateur. L'organisation pédagogique se représente sous la forme d'une collaboration, de la même manière qu'une organisation Mascaret. Les rôles sont définis par leur nom, leur multiplicté et leur type (`RoleClass`).

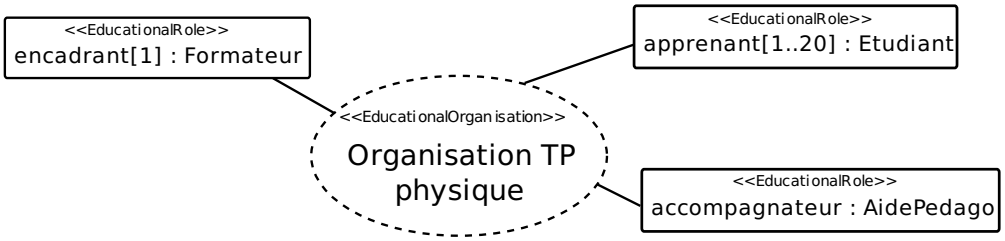


Figure 4.11 – Un exemple de structure organisationnelle pédagogique.

4.5 Les activités pédagogiques et leur enchaînement

Dans cette section, nous décrivons le point central de la scénarisation pédagogique, à savoir les activités pédagogiques et la manière dont elles s'enchaînent. Le rôle d'une activité pédagogique telle que nous la concevons n'est pas seulement de décrire les actions que l'apprenant doit faire pour réaliser une tâche, mais également d'exprimer les **spécificités** de cette tâche d'un point de vue **pédagogique** (erreurs classiques, situations de danger, *etc*) ainsi que les solutions généralement mises en œuvre pour assister l'apprenant dans des situations particulières. Le scénario pédagogique permet de définir l'organisation de ces activités et de les lier aux autres éléments définis précédemment (environnements, ressources, organisations pédagogiques, *etc*). De la même manière que dans IMS-LD, chaque activité pédagogique se déroule dans un environnement (`EducationalEnvironment`). Cet environnement peut contenir ou non un environnement virtuel. La figure 4.12 présente un schéma résumant les liens entre les éléments de la session d'apprentissage.

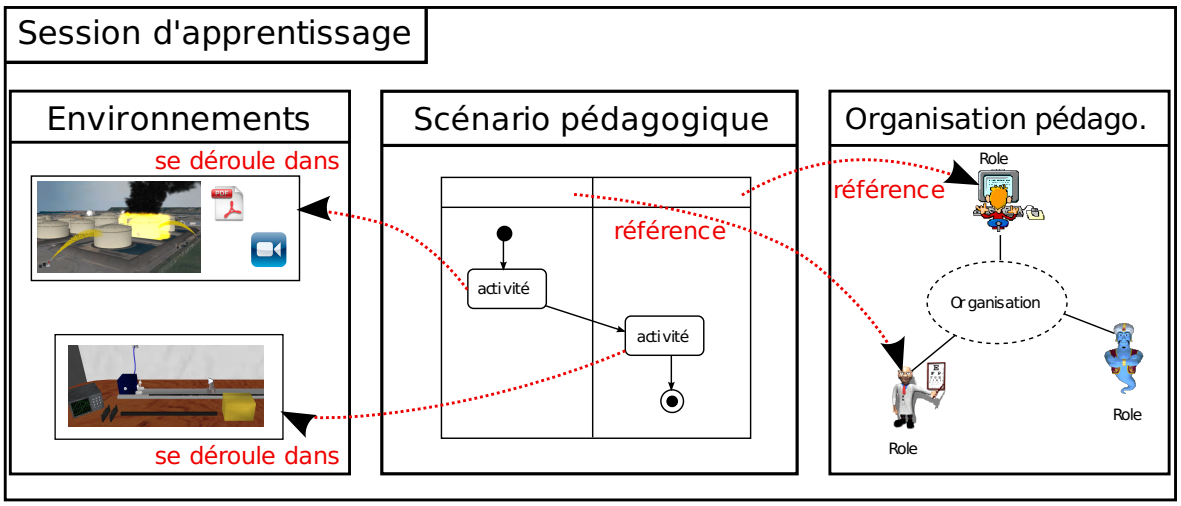


Figure 4.12 – Schéma présentant les liens entre les différents éléments de la session d'apprentissage.

Dans un premier temps, nous commençons par décrire la manière dont Poseidon décrit l'enchaînement des activités pédagogiques (section 4.5.a.), avant de détailler la nature de ces

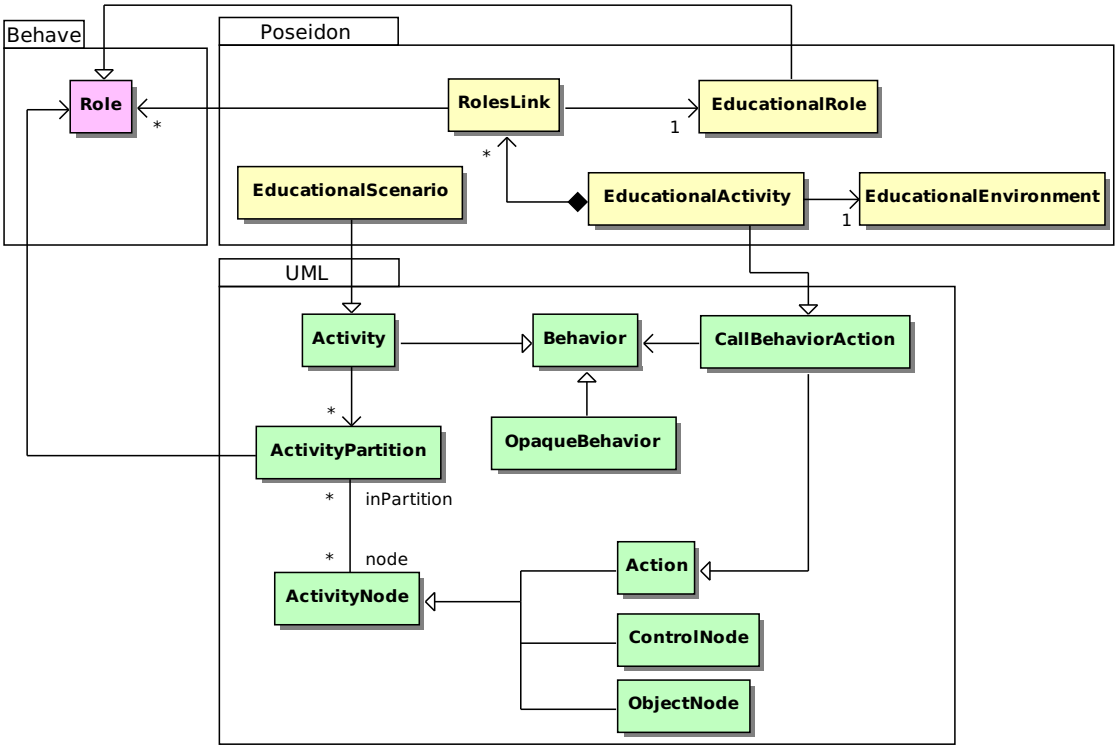


Figure 4.13 – Enchaînement d'activités pédagogiques.

activités pédagogiques (section 4.5.b.).

4.5.a. Modèle d'enchaînement des activités pédagogiques

Le modèle d'enchaînement des activités pédagogiques que nous proposons s'appuie sur le modèle des activités défini dans Have, qui enrichit les activités UML. La figure 4.13 présente une vue simplifiée du modèle d'enchaînement des activités pédagogiques que nous proposons. Un scénario pédagogique (`EducationalScenario`) est une sorte d'`Activity`. Une activité contient des couloirs d'activités également appelés partitions (`ActivityPartition`). Behave considère qu'un couloir d'activité regroupe les actions qu'un acteur jouant un rôle particulier doit réaliser.

Dans une `Activity` d'UML, les actions à réaliser par les acteurs sont représentées par des nœuds de type `Action`. Dans le cadre du scénario pédagogique, ces actions sont des activités pédagogiques, représentées par la classe `EducationalActivity`. Une activité pédagogique hérite (indirectement) de la classe `Action`, ce qui permet de l'agencer dans un diagramme d'activités représentant le scénario pédagogique. Dans le scénario pédagogique, les nœuds de contrôle (`ControleNode`) gardent la sémantique décrite par UML.

Dans un scénario pédagogique, un nœud d'objet (`ObjectNode`) peut représenter une ressource fournie pour la réalisation d'une activité (s'il est en entrée de l'activité), ou un résultat produit

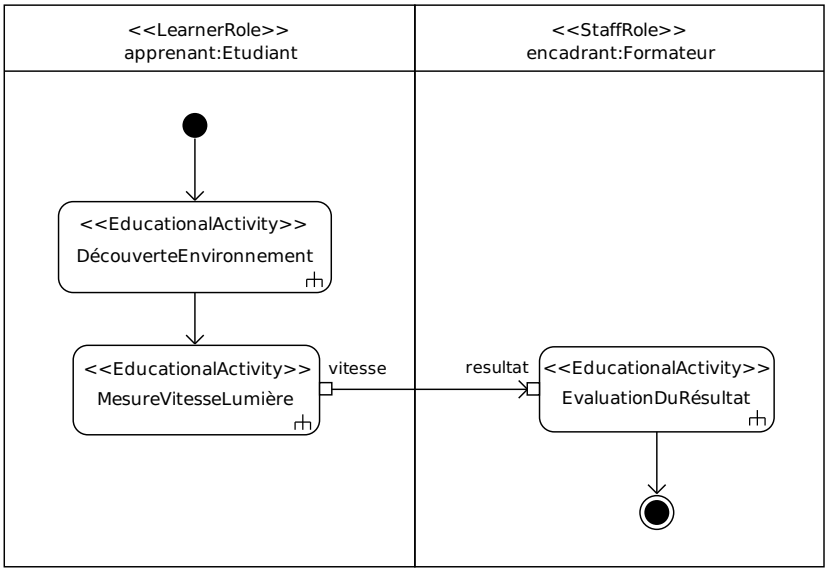


Figure 4.14 – Exemple de diagramme d'activités représentant un scénario pédagogique.

par cette activité (s'il est en sortie). Ces objets n'ont aucun lien avec un environnement virtuel (puisque le scénario ne se déroule pas dans un environnement particulier) ; il peut s'agir en revanche de ressources pédagogiques (donnée audio, vidéo, LOM, SCORM, *etc*) ou une variable créée par le formateur pour contrôler l'évolution du scénario. Les nœuds d'objets peuvent aussi être représentés sous formes de broches d'entrée-sortie (`Pin`). La figure 4.14 présente un exemple de scénario pédagogique représenté sous cette forme.

4.5.b. Description des activités pédagogiques

La principale différence entre notre proposition et les modèles de scénarios décontextualisés tels qu'IMS-LD réside dans la prise en compte du déroulement des activités pédagogiques. Notre démarche consiste à permettre au formateur d'exprimer, au niveau du scénario pédagogique, le contrôle de l'activité de l'apprenant qu'il veut mettre en place en vue d'instrumenter cette activité d'un point de vue pédagogique (aides contextuelles, consignes particulières, *etc.*). Pour cela, les activités pédagogiques durant lesquelles les apprenants ou le formateur manipulent un environnement de réalité virtuelle ne sont pas considérées comme des *boîtes noires*. La description d'une activité pédagogique avec Poseidon se déroule en deux phases.

1. Dans un premier temps, le formateur décrit la tâche qui doit être réalisée dans l'environnement virtuel. Cette description s'appuie sur les actions ou procédures définies dans le modèle métier (M1).
2. Dans un deuxième temps, le formateur peut enrichir cette première description en ajoutant des informations pédagogiques concernant l'activité (traitement des erreurs classiques, consignes particulières, *etc*). Ces informations peuvent avoir pour conséquence de modifier la tâche décrite précédemment, afin de l'adapter aux car-

actéristiques de l'exercice (niveau des apprenants, points importants, difficultés particulières, etc).

La description de la tâche attendue dans l'environnement s'appuie sur le modèle métier des activités humaines qu'il est possible d'y réaliser. Cette modélisation s'appuie sur le méta-modèle Have.

Lorsque le formateur crée un scénario pédagogique, il a accès de manière explicite au modèle des tâches métier. À partir de ce modèle, il décrit la tâche à réaliser par l'apprenant dans l'environnement virtuel de deux manières.
– soit il utilise directement une procédure déjà définie dans le modèle,
– soit il crée lui-même le diagramme d'activités, en sélectionnant dans le modèle les actions à réaliser qui l'intéressent. De cette manière, le formateur peut concevoir une activité adaptée à l'objectif pédagogique et au niveau de l'apprenant.

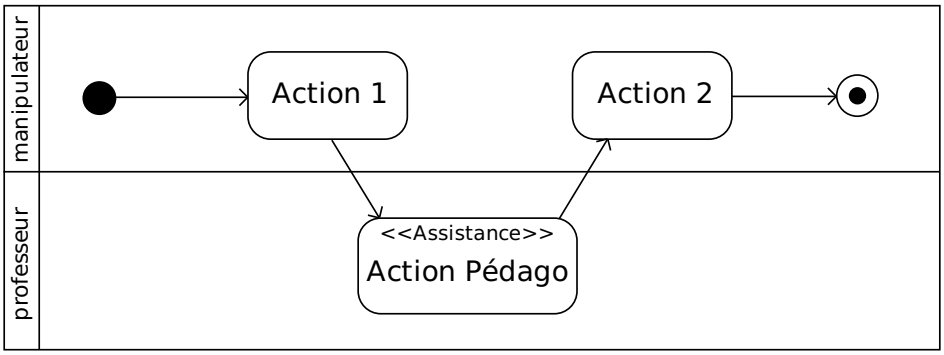
Dans les deux cas, le résultat est un diagramme d'activités décrivant l'activité attendue dans l'envionnement virtuel. Ce formalisme est capable de gérer des enchaînements complexes d'actions et permet au formateur de décrire l'activité attendue de l'apprenant de manière plus précise que les modèles utilisant le principe de l'*authoring by doing* comme RIDES ou FORMID. De plus, l'utilisation d'un modèle d'activité métier explicite permet aux outils de scénarisation d'utiliser le vocabulaire du domaine, qui est maîtrisé par la personne qui crée le scénario pédagogique.

Nous proposons maintenant d'aller plus loin que la simple description de la tâche à réaliser, en permettant au formateur d'enrichir cette description afin de l'adapter au cadre pédagogique de la formation. En effet, la description d'activité effectuée par le formateur lors de la première étape peut servir de base à la création d'exercices différents, ayant pour but d'appréhender différents aspects d'une même tâche, chacune ayant ses difficultés propres. Pour instrumenter la tâche à réaliser, nous fournissons au formateur un ensemble d'outils appelés **actions pédagogiques**. Une action pédagogique peut avoir pour effet de modifier l'environnement virtuel ou le scénario pédagogique. Ces actions permettent au formateur d'exprimer dans le scénario pédagogique, des informations concernant la manière de traiter une erreur classique ou de recadrer un élève en difficulté, dans des situations particulières, identifiées au préalable.

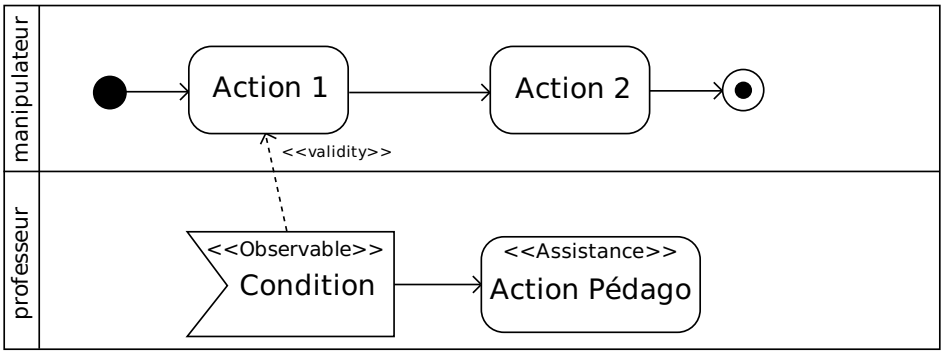
Dans le scénario pédagogique, le traitement des situations particulières par des actions pédagogiques peut être décrit de deux manières.

1. Il peut être décrit de manière **statique** ; il est possible d'agencer une action pédagogique parmi les actions d'un diagramme d'activité décrivant le déroulement d'une activité pédagogique. Dans ce cas, on connaît l'enchaînement particulier d'actions qui mène à la situation particulière à traiter.
2. Il peut également être décrit de manière **événementielle** ; la réalisation d'une action pédagogique peut être associée à la vérification d'un observable défini dans le scénario. Cet observable peut porter sur l'état de l'environnement d'apprentissage, sur l'activité de l'apprenant, *etc.* Dans ce cas-ci, la situation à traiter est caractérisée par un événement plutôt que par un enchaînement d'actions.

La figure 4.15 présente les deux notations correspondant à l'agencement statique ou événementiel des actions pédagogiques. Sur la figure 4.15(a), l'action pédagogique est agencée dans le déroulement de l'activité pédagogique, entre les actions `Action1` et `Action2`. Elle



(a)



(b)

Figure 4.15 – Notation des actions pédagogiques dans un diagramme d'activités.

se déclenchera systématiquement entre les exécutions de ces deux actions. Sur la figure 4.15(b), l'action pédagogique se déclenchera uniquement lorsque l'observable associé sera vérifié. De plus, le domaine de validité de l'observable permet de fixer l'espace de temps durant lequel l'occurrence de l'observable déclenche l'action pédagogique. Sur cet exemple, l'action pédagogique sera déclenchée uniquement si l'observable est vérifié durant l'exécution de `Action1`.

4.5.c. Modèle des actions pédagogiques

Dans la section précédente, nous avons expliqué à quoi sert une action pédagogique et comment elle peut être mise en œuvre dans le scénario pédagogique. Il s'agit des mêmes actions pédagogiques qui sont manipulées pas Pegase (cf chapitre 3). Une action pédagogique permet au formateur d'apporter des modifications dans l'environnement ou dans le déroulement du scénario, dans le but d'aider, de guider ou de perturber l'apprenant dans la réalisation de sa tâche.

Pour permettre au formateur de créer les actions pédagogiques qu'il souhaite, nous avons identifié un ensemble de **primitives** fournissant un moyen de manipuler les différents éléments du scénario. Ces primitives peuvent alors être **instanciées** et **combinées** pour créer des actions pédagogiques. La figure 4.16 présente le modèle des primitives et des actions pédagogiques. La classe `EducationalAction` hérite de la classe `Action` d'UML. On peut ainsi placer les actions pédagogiques dans un diagramme d'activités. Une action pédagogique est un ensemble de primitives (`AssistancePrimitive`), et s'applique à un environnement (`EducationalEnvironment`). Chaque sous-classe de primitive doit définir les méthodes permettant de l'appliquer (`act`) dans l'environnement, de l'annuler (`reset`), d'en avoir un aperçu (`preview`) ainsi qu'une description en langue naturelle (`description`). L'appel d'une de ces méthodes sur une action pédagogique provoque l'appel de cette méthode sur chacune de ses primitives. Le tableau de la figure 4.17 résume les primitives fournies.

Une action pédagogique est un ensemble d'instanciations des primitives que nous venons de voir. Pour permettre au formateur de pouvoir créer de manière simple ce genre d'actions, nous avons développé un éditeur. La figure 4.18 montre l'utilisation de l'interface pour créer l'action pédagogique « Montrer les zones de sécurité du Rafale 12 », pour l'application Gaspar [Marion et al., 2007]. Cette assistance consiste à colorer en orange les zones de sécurité avant et arrière du Rafale, puis à fixer leur transparence à 50%. De plus, elle stoppe la procédure de catapultage du Rafale. Elle est donc composée de cinq primitives.

Cette action pédagogique est généralement déclenchée lorsque l'apprenant ne respecte pas les zones de sécurité lors du catapultage du Rafale. La figure 4.19 montre le résultat du déclenchement de cette action pédagogique.

4.5.d. Exécution des activités pédagogiques

Dans cette section, nous expliquons la manière dont sont traitées les informations contenues dans le scénario pédagogique concernant les activités pédagogiques. Nous abordons donc

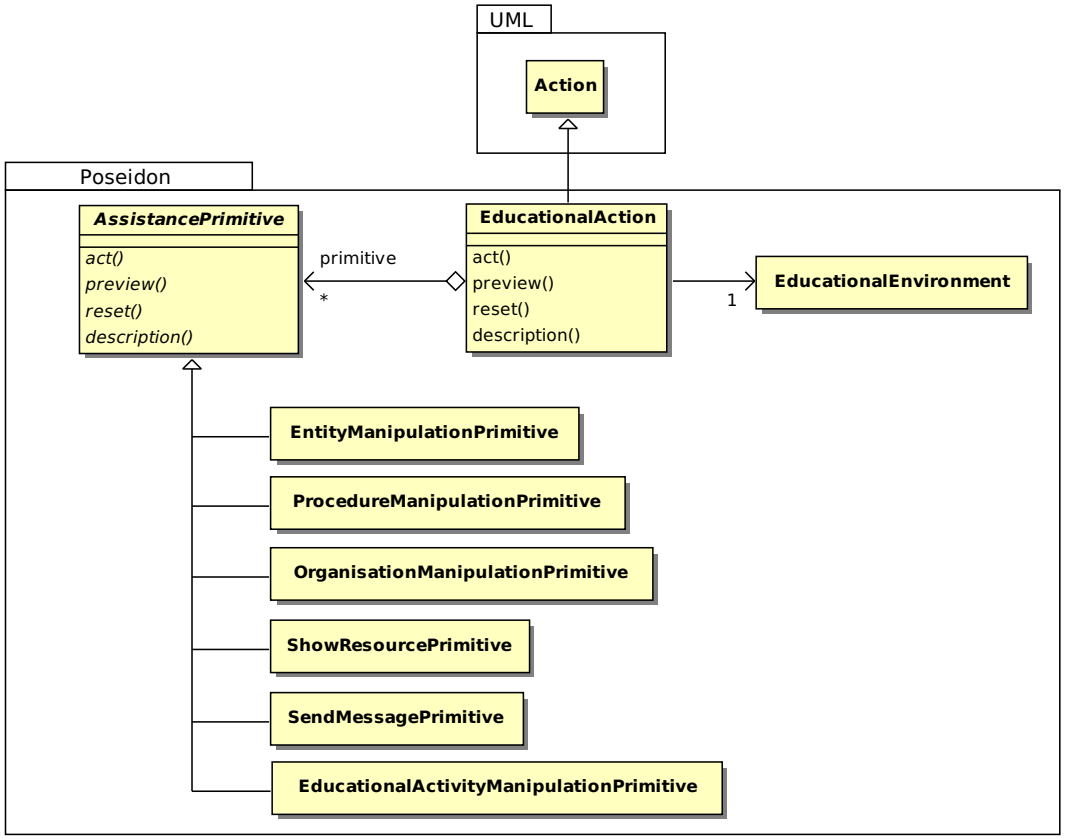


Figure 4.16 – Modèle des primitives et des actions pédagogiques.

la question de l'exécution des scénarios Poseidon, et plus particulièrement des activités pédagogiques se déroulant en environnement virtuel.

Pour qu'un scénario pédagogique puisse s'exécuter, il faut que tous les rôles de l'organisation pédagogique soient affectés. Les acteurs jouant ces rôles peuvent être humains ou virtuels. La modélisation des informations concernant les apprenants n'étant pas au cœur de nos travaux, Poseidon utilise le modèle de l'apprenant défini par l'ITS Pegase [Buche et al., 2010] que nous décrivons plus loin. Une fois les acteurs connectés et les rôles affectés, l'exécution du scénario pédagogique peut commencer. La plateforme se charge de proposer chaque activité pédagogique aux acteurs concernés, et de leur fournir les ressources nécessaires à la réalisation de ces activités. Au début de chaque activité pédagogique, les éléments constituant l'environnement virtuel sont instanciés sur la plateforme [3], les rôles sont attribués aux utilisateurs dans l'environnement virtuel, puis le scénario décrivant les événements à survenir dans l'environnement (`SimulationScenario`) est démarré. À partir de ce moment, les utilisateurs peuvent agir dans l'environnement virtuel.

Une fois l'activité pédagogique démarrée, le rôle de l'agent « scénario pédagogique » est de suivre l'avancement de cette activité en fonction de l'activité des différents acteurs dans

3. Nous n'abordons pas dans ce manuscrit les questions liées à la distribution des utilisateurs sur le réseau. Poseidon s'appuie sur une implémentation de Mascaret qui permet une distribution de l'environnement virtuel *via* une architecture clients-serveur.

Primitive	Description
Manipulation d'entités	Cette primitive permet de créer, modifier ou supprimer une entité. Elle permet par exemple de modifier la valeur d'une propriété ou rendre transparent l'entité.
Manipulation d'entités organisationnelle	Cette primitive permet de créer, modifier ou supprimer une entité organisationnelle. Elle permet par exemple d'assigner ou révoquer un rôle. Cela permet par exemple au formateur de prendre la place de l'apprenant.
Manipulation d'une procédure métier	Cette primitive permet de démarrer ou stopper l'exécution d'une procédure métier.
Modification des capacités de l'apprenant	Cette primitive permet de modifier les capacités d'action et de perception de l'apprenant. Il s'agit par exemple de modifier le point de vue de l'apprenant ou modifier les paramètres du périphérique d'interaction utilisé.
Décharger un utilisateur	Cette primitive permet de faire faire une action de la responsabilité de l'apprenant à un autre agent.
Montrer la réalisation d'une action	Cette primitive permet de faire faire une action par un agent sans que l'action est réellement un effet dans la simulation.
Envoyer un message	Cette primitive permet au formateur d'envoyer un message aux autres agents (humains ou virtuels).
Fournir une ressource pédagogique	Cette primitive permet d'afficher une ressource pédagogique (film, document *etc*).

Figure 4.17 – Liste d'actions pédagogiques.

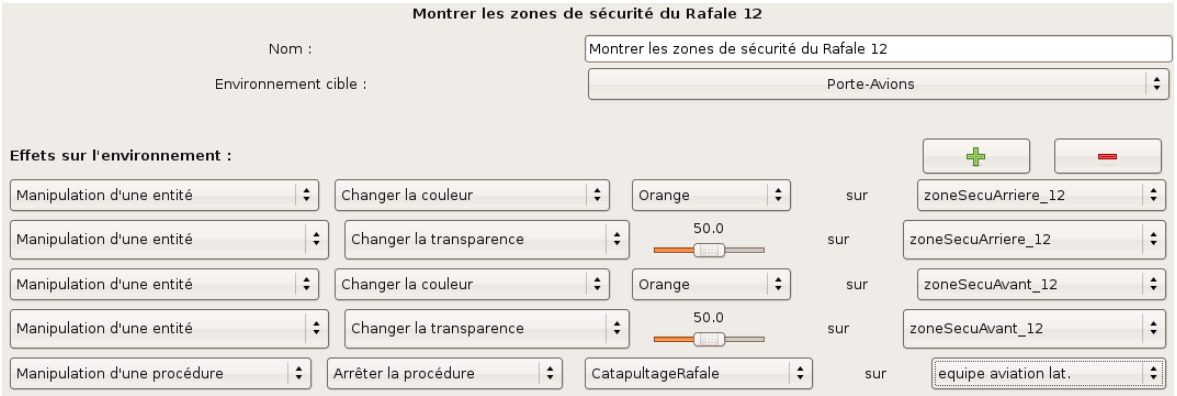


Figure 4.18 – Création d'une action pédagogique.

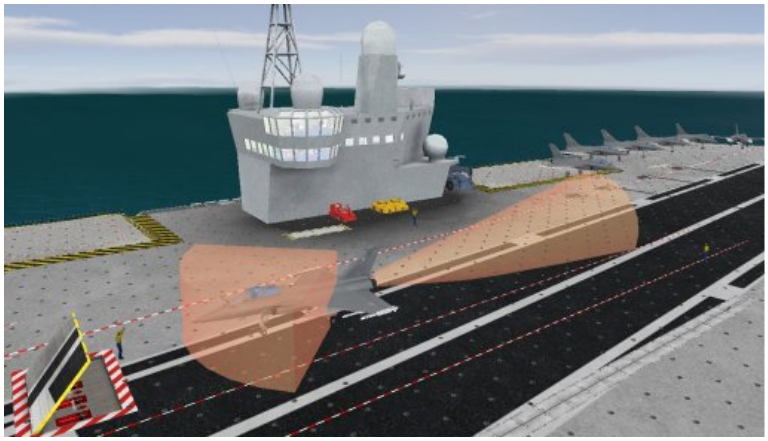


Figure 4.19 – Un exemple d'enrichissement de l'environnement : affichage des zones de sécurité autour d'un Rafale.

l'environnement virtuel. L'algorithme de la figure 4.20 présente ce comportement.

Chaque acteur du scénario pédagogique peut être joué par un humain ou un agent artificiel. Nous décrivons ici les spécificités liées à ces différents cas. Même si aucun obstacle technique ne s'y oppose, nous ne nous intéressons pas au cas où les apprenants sont joués par des agents artificiels, cette éventualité étant peu pertinente d'un point de vue pédagogique. Dans le cas où le rôle du formateur est joué par un agent artificiel, nous distinguons deux cas différents : soit aucun agent n'est proposé, auquel cas c'est un agent au comportement réactif, implémenté dans Poseidon qui joue le rôle du formateur ; soit un agent pédagogique intelligent est fourni. Il y a donc trois cas distincts.

Cas n°1 : apprenant(s) humain(s) – formateur joué par un agent réactif

Dans ce cas de figure, le comportement du formateur est uniquement basé sur les informations contenues dans le scénario pédagogique. L'agent réactif est informé de l'évolution du scénario par l'agent « scénario pédagogique » (dont nous venons de décrire le comportement), qui lui envoie des messages de type FIPA-ACL. Lorsque le scénario préconise le déclenchement d'une action pédagogique (sur occurrence d'un observable par exemple), l'agent réactif déclenche systématiquement l'action pédagogique.

Si ce cas de figure évite d'avoir à mobiliser un formateur humain, la réussite de l'activité pédagogique repose sur la qualité du scénario fourni, puisque le comportement de l'agent ne permet aucune adaptation à des situations non prévues.

Cas n°2 : apprenant(s) humain(s) – formateur humain

Dans le cas où le rôle du formateur est joué par un humain, le scénario pédagogique lui fournit des informations qu'il peut choisir de suivre, ou pas. Dans ce cas de figure, l'agent « scénario pédagogique » envoie toujours les informations concernant l'évolution

```
1  fini ⟵ false;
2  ajouter etatInitial dans la liste des noeuds à atteindre;
3  tant que fini = false faire
4      pour chaque message m dans la messagerie faire
5          si m est de type ActionExecutionMessage alors
               //Un utilisateur a réalisé une action
6              si m.action ∈ liste des nœuds à atteindre alors
7                  si l'action est réalisée par le bon utilisateur alors
8                      enlever m.action de la liste des nœuds à atteindre;
9                      pour chaque nœud n dans m.action.nœudsSuivants faire
10                         ajouter n dans la liste des nœuds à atteindre;
11                     finprch
12                 finsi
13             finsi
14         finsi
15         sinon si m est de type ChangeEventMessage alors
               //Un élément de l'environnement a été modifié
16             pour chaque observable obs faire
17                 si l'élément qui a changé intervient dans la condition de obs alors
18                     si obs est entièrement vérifié alors Signaler l'occurrence de obs;
19                 finsi
20             finprch
21             pour chaque état à atteindre e faire
22                 si l'élément qui a changé intervient dans la condition de e alors
23                     si e est entièrement vérifié alors
24                         enlever e de la liste des nœuds à atteindre;
25                         pour chaque nœud n dans e.nœudsSuivants faire
26                             ajouter n dans la liste des nœuds à atteindre;
27                         finprch
28                     finsi
29                 finsi
30             finprch
31         finsi
32         si m est de type EducationalActionMessage alors
               //Une action pédagogique doit être exécutée
33             pour chaque primitive p de l'action pédagogique faire
34                 Transmettre p à la plateforme;
35             finprch
36         finsi
37         Mettre à jour avancement activité;
38         si étatFinal ∈ liste des actions à réaliser alors
39             fini ⟵ true;
40         finsi
41     finprch
42 fintq
```

Figure 4.20 – Algorithme de suivi de l'exécution d'une activité pédagogique

de la situation pédagogique sous forme de messages FIPA-ACL. Poseidon fournit un agent « interface » qui reçoit et interprète ces messages, et qui retranscrit ces informations dans une interface. Cette interface permet non seulement au formateur de suivre le déroulement de l'activité, mais également d'agir dans l'environnement, par le biais des actions pédagogiques. Lorsque le scénario préconise le déclenchement d'une action pédagogique, c'est le formateur qui choisit s'il souhaite la déclencher ou non. De la même manière, l'interface permet au formateur de déclencher des actions pédagogiques non prévues dans le scénario.

Cas n°3 : apprenant(s) humain(s) – formateur joué par un agent intelligent

Ce dernier cas est très similaire au précédent, dans le sens où ici encore, le comportement du formateur ne se limite pas aux informations contenues dans le scénario. La principale différence est que dans ce cas, les messages émanant de l'agent « scénario pédagogique » doivent être traités par l'agent pédagogique intelligent lui-même, et non par l'agent « interface » fourni par Poseidon. Pour pouvoir être utilisé dans l'exécution d'un scénario Poseidon, l'agent pédagogique doit donc être capable d'interpréter les informations contenues dans les messages de la plateforme, pour les ajouter à sa base de connaissances. Actuellement, nous travaillons sur le tuteur pédagogique intelligent Pegase [Buche et al., 2010], afin de le rendre compatible avec Poseidon.

4.6 Compatibilité avec les standards éducatifs

Même s'il n'a pas été initialement conçu dans cet objectif, le modèle Poseidon peut être vu comme une extension d'IMS-LD. En effet, Poseidon conserve les concepts de base d'IMS-LD (activités pédagogiques, rôles, environnement d'apprentissage, *etc*), auxquels il ajoute des informations concernant l'environnement virtuel.

Classiquement les scénarios pédagogiques sont exprimés dans un formalisme appelé langage de modélisation pédagogique (EML). Ce langage permet de décrire les différents éléments qui composent le scénario (activités, rôles, ressources, *etc*), sous une forme permettant l'exécution de ce scénario sur une plateforme pédagogique. Les scénarios IMS-LD par exemple sont exprimés dans un format XML, conforme à un schéma [4] particulier.
Pour définir le langage de modélisation pédagogique des scénarios Poseidon, nous nous sommes appuyés sur le formalisme IMS-LD. Les balises représentant les concepts communs aux deux modèles ont été conservées, et des balises spécifiques ont été ajoutées. Le *listing* de la figure 4.21 présente une partie d'exemple de scénario pédagogique Poseidon. L'utilisation de balises communes fait qu'il est possible d'exécuter des scénarios IMS-LD sur une plateforme Poseidon.

La compatibilité inverse quant à elle, n'est pas assurée. Dans un premier temps, on pourrait penser qu'un scénario Poseidon peut être exécuté sur une plateforme IMS-LD, simplement en omettant les balises spécifiques. Cependant, certaines particularités du modèle font que les scénarios ne peuvent être interprétés par IMS-LD. Par exemple, la description de

4. Un schéma XML spécifie quelles balises sont autorisées dans le document, et à quels endroits. Les schémas IMS-LD peuvent être téléchargés sur le site : http ://www.imsglobal.org/learningdesign/

```
<environments>
  <environment identifier="env-9c9780378cfa1776a9f30fc2a2ffa7bc">
    <title>Paillasse</title>
    <class-model type="XMI" url="XML/Paillasse.world_model.xmi"/>
    <virtual-environment type="Mascaret" url="Paillasse.world.xml"/>
  </environment>
</environments>
<activities>
  <learning-activity identifier="act-07469e194ffaa07b4fcabddca98fd618" isvisible="true">
    <title>Mesure Vitesse Lumiere</title>
    <environment-ref ref="env-9c9780378cfa1776a9f30fc2a2ffa7bc"/>
    ...
  </learning-activity>
</activities>
```

Figure 4.21 – Un exemple de scénario Poseidon au format XML

l'enchaînement des activités sous forme de diagramme d'activité UML rend possible la description d'enchaînements complexes d'actions, qu'il est impossible de représenter en IMS-LD. L'interprétation d'un scénario Poseidon par IMS-LD n'est donc possible que si le formateur n'utilise pas d'enchaînement complexe pour les activités pédagogiques.

4.7 Conclusion

Dans ce chapitre, notre objectif était de formaliser un modèle de scénario pédagogique rendant possible la prise en compte de la description d'activités pédagogiques se déroulant en environnement virtuel. Le modèle que nous avons présenté s'intègre dans le projet Mascaret, ce qui lui permet de s'appuyer sur une représentation abstraite des environnements virtuels pour l'apprentissage.

Comme la plupart des modèles de scénarios décontextualisés existants, Poseidon décrit un scénario pédagogique comme un enchaînement d'activités pédagogiques. La description de l'enchaînement des activités est basée sur un formalisme existant et standardisé : les diagrammes d'activités UML. Ce formalisme permet de décrire des enchaînements complexes qu'il est très difficile (voire impossible) de mettre en œuvre avec un langage comme IMS-LD.

Poseidon se place dans la catégorie des modèles produisant des scénarios contextualisés, en permettant la description du déroulement d'activités pédagogiques situées en environnement virtuel. La description d'une activité pédagogique s'effectue en deux phases. Tout d'abord, le formateur décrit la tâche à réaliser dans l'environnement par les acteurs impliqués dans l'activité. La description de cette tâche s'appuie à la fois sur la description de l'environnement et de son modèle, conforme au méta-modèle Veha, et sur la description des actions métier qu'il est possible de réaliser dans cet environnement, conforme au méta-modèle Have.

Puis, le formateur superpose à la description de cette tâche des informations pédagogiques. Ces informations peuvent avoir plusieurs buts : adapter la difficulté de la tâche au niveau des apprenants, traiter des erreurs classiques, etc. Pour cela, Poseidon met à la disposition du formateur un ensemble de primitives lui permettant de créer des actions pédagogiques. Ces actions pédagogiques peuvent être agencées dans les actions attendues de l'apprenant ou sur

occurrence d'observables. À l'exécution, ces informations sont utilisées par l'entité jouant le rôle de formateur (qu'il soit réel ou virtuel) pour prendre ses décisions.

Enfin, Poseidon fournit un modèle organisationnel permettant de décrire les rôles joués par les différents acteurs du scénario. Ce modèle s'appuie sur les concepts d'apprenant (`Learner`) et d'encadrant (`Staff`) définis par IMS-LD. Le modèle décrit l'implication des acteurs dans les activités pédagogiques. Nous avons montré également comment Pegase peut jouer un rôle dans une tel organisation. Pour chaque activité pédagogique se déroulant en environnement virtuel, Poseidon rend possible l'expression des liens entre les acteurs jouant un rôle dans le scénario pédagogique, et le rôle qu'ils doivent jouer dans une organisation métier de l'environnement. La formalisation de ces liens est une chose que nous n'avons trouvée dans aucun modèle existant.

Nous résumons maintenant les points forts de notre proposition.
- Poseidon n'induit pas de stratégie pédagogique particulière, contrairement à des modèles comme Gvt ou SimQuest. Il peut donc être utilisé dans des contextes plus variés.
- La description de la tâche à réaliser et celle des informations pédagogiques concernant cette tâche sont décrites indépendamment, contrairement à des modèles comme Formid ou Fiacre. Ainsi, la même description de tâche à réaliser peut être réutilisée pour différents exercices.
- Les activités se déroulent dans des environnements informés, basés chacun sur un modèle du domaine introspectable. Lors de la conception du scénario, le formateur peut donc manipuler des concepts métiers qui lui sont familiers, contrairement à des modèles comme Gvt ou Vts.
- Le méta-modèle décrivant les environnement virtuels est défini de manière explicite, contrairement à des modèles comme Rides, SimQuest ou Sam. Il est donc possible de fournir des outils génériques pour la conception et l'exploitation des scénarios, indépendamment du domaine étudié.

Chapitre 5

Bilan et perspectives

Nous avons proposé des modèles permettant l'explicitation des connaissances pour les agents peuplant des environnements virtuels. Ces connaissances portent sur la structure et la dynamique de l'environnement ainsi que sur les procédures que peuvent réaliser des équipes de personnels dans cet environnement. Elles représentent la compétence qui constitue le premier sommet du triangle pédagogique. Le langage utilisé pour décrire cette compétence métier est UML ; MASCARET peut alors être considéré comme un profil spécifique à la conception d'environnements virtuels. Ainsi, un modèle métier écrit à l'aide de MASCARET est une instance du méta-modèle MASCARET et est donc une donnée manipulable en temps réel dans la simulation. L'avantage d'UML dans ce cadre est qu'il s'agit d'un langage normalisé et graphique, ce qui le rend utilisable par des experts d'un métier spécifique mais non experts de la conception informatique. De plus, UML est un langage unifié dans le sens où le même langage, et donc le même outil, permet de décrire l'aspect statique et dynamique du système abordé. GASPAR est un exemple d'application de type *serious game* que nous avons conçue pour DCNS. L'intérêt de MASCARET dans cette application est d'une part de pouvoir tester rapidement de nouvelles configurations (géométrie du porte-avions, positions des avions, nouvelles procédures) mais également de pouvoir générer automatiquement un bilan du résultat de la simulation (durée d'une procédure, temps d'occupation d'une ressource, etc).

Afin de réifier les relations pédagogiques et didactiques du triangle pédagogique nous avons proposé deux modèles :
- PEGASE qui s'appuie sur un modèle pédagogique pour proposer des assistances pédagogiques adaptées,
- POSEIDON, permet au formateur de décrire l'exercice et les actions qui devront être menés dans l'environnement virtuel.

Ce deux modèles s'appuient sur MASCARET, ce qui les rend indépendants d'un domaine métier particulier. Toutefois, POSEIDON et PEGASE manipulent les connaissances métier car elles sont explicitées grâce à MASCARET.

La dernière relation du triangle pédagogique est la relation d'apprentissage. Nous n'avons pas proposé de modèle informatique permettant d'intervenir en ligne pour réifier cette relation. Toutefois, nous considérons que les outils que nous avons proposés pour réifier

les autres parties du triangle favorisent cet apprentissage. Toutefois, il serait nécessaire d'évaluer cet apprentissage afin de répondre aux questions posées en introduction. Pour cela nous avons développé un logiciel pour concevoir et conduire des expérimentations sur l'apprentissage et l'utilisation de documents procéduraux en environnement virtuel (TIP-EXE [Ganier and Querrec, 2008]). Grâce à ce logiciel nous avons conçu et réalisé une première expérimentation sur l'apprentissage de procédures. Il s'agissait d'évaluer le transfert d'un apprentissage dans un environnement virtuel vers la même situation dans le réel.

Le disposif étudié était un programmateur éléctronique. Les tâches à apprendre étaient la mise à l'heure de l'appareil et la création de programmes de démarrage et d'arrêt d'un appareil électrique relié au programmateur. Un environnement virtuel représentant l'appareil et son fonctionnement a été réalisé en utilisant Mascaret. Un programmateur réel a été équipé afin de le relier à un ordinateur pour enregistrer les actions de l'utilisateur. Pour apprendre la procédure, un groupe de sujets a utilisé le dispositif virtuel et un autre groupe a utilisé le dispositif réel. Les instructions liées à la tâche et à la procédure étaient accessibles *via* Tip-exe. Toutes les actions et les consultations réalisées par l'utilisateur étaient automatiquement enregistrées dans un fichier journal. Tenant compte des résultats de la première expérimentation citée en introduction, lors de la phase d'apprentissage trois essais seulement ont été réalisés par les utilisateurs. Une semaine après cette phase d'apprentissage, les deux groupes ont réalisé un essai sur le dispositif réel. Les premiers résultats montrent qu'il n'y a aucune différence significative entre les deux groupes. Tous les participants qu'ils aient appris sur le dispositif virtuel ou réel ont réussit à réaliser la procédure demandée sans avoir recours à la documentation. Cela signifie qu'un apprentissage réalisé en environnement virtuel est susceptible d'être transféré avec succès dans la réalité. La figure 5.1 illustre le dispositif utilisé.

Figure 5.1 – Dispositif d'expérimentation sur le transfert d'apprentissage.

En introduction, nous nous posions deux questions sur la conception des situations d'apprentissage en environnement virtuel et sur l'évaluation de l'efficacité de ces situations. Notre réponse à ces questions est notre proposition du modèle Mascaret. L'ensemble des modèles constituant Mascaret, qui ont été décrits dans ce mémoire ou qui sont en cours, sont résumés sur le triangle pédagogique de la figure 5.2. Nous considérons que le fait que la compétance soit explicite (Veha, Have et Behave), que le formateur puisse ainsi structurer les scénarios (Poseidon) et qu'il soit assister dans sa relation pédagogique avec l'apprenant

(Pegase) garantie la conception de situations pédagogiques efficaces. Afin de valider cette affirmation, nous proposons de réaliser des expérimentations. Pour concevoir et pour mener ces expérimentations nous avons conçu Tip-Exe et nous collaborons avec des chercheurs en psychologie et en sciences de l'éducation pour réaliser ces expérimentations.

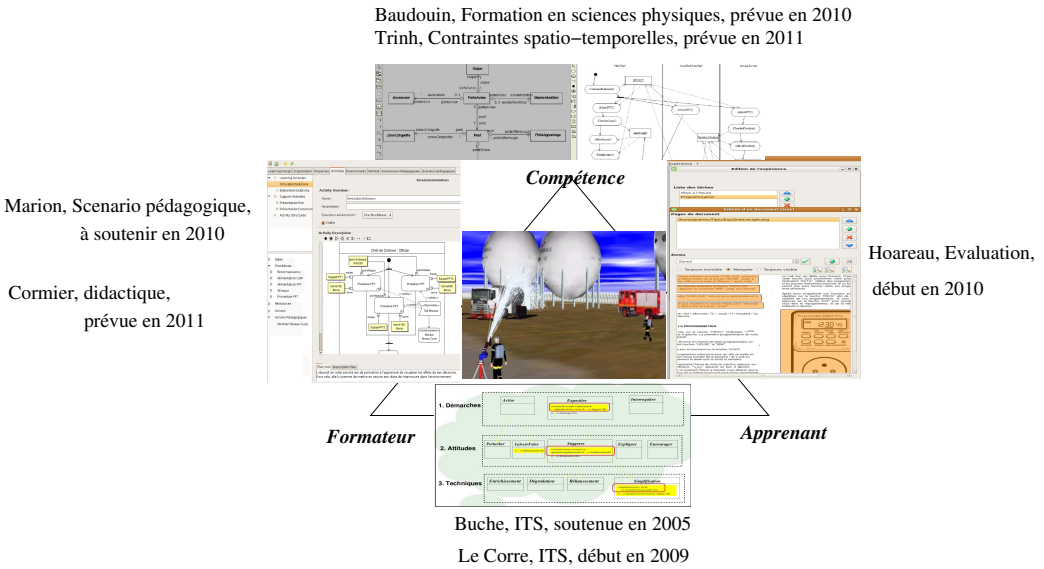


Figure 5.2 – Bilan des activités de recherche.

Les perspectives de nos travaux portent tout d'abord sur la modélisation de la compétence. Actuellement, les compétences modélisées grâce à Mascaret concernent la notion de travail procédural. De ce fait nous abordons plus particulièrement la formation professionnelle. Or dans ces domaines métier, les connaissances des experts sont déjà capitalisées grâce à des logiciels de CAO pour l'aspect géométriques ou des bases de données pour d'autres aspects techniques. Une des premières perspectives que nous souhaitons conduire est la faisabilité pour Mascaret de se connecter à ce type de produit (que l'on nomme PLM) [1].

Cependant, de nombreuses activités humaines ou domaines de formation peuvent s'exprimer sans procédure explicite. C'est le cas, par exemple, dans le domaine des sports collectifs [2] ou celui de l'apprentissage scolaire en particulier en travaux pratiques [3]. Dans ces domaines, les situations sont très dynamiques, et les décrire *a priori* à l'aide de procédures n'est pas réaliste. Pour aborder ce type d'activité, il faut intégrer des mécanismes adaptatifs et donc associer de l'autonomie aux comportements simulés. Cela aura pour conséquence d'ajouter des éléments sémantiques aux modèles afin que les agents pédagogiques puissent non pas raisonner sur le respect ou non d'un enchaînement temporel d'actions, mais sur les effets des actions, sur les causes d'erreurs, sur l'état de l'environnement et sur les contraintes à respecter. De plus, actuellement, les agents pédagogiques ne réagissent que sur action (ou

1. Ceci pourrait être mené dans le cadre d'un projet en cours de dépôt en collaboration avec de nombreux partenaires industriels.

2. Ces travaux font l'objet de la thèse de C. De Keukelaere encadrée par P. De Loor, soutenance prévue en 2011.

3. Ces travaux font l'objet de la thèse de C. Baudouin encadrée par P. Chevaillier, soutenance prévue en 2011.

inaction) de l'apprenant. Il est donc important que l'agent lui-même puisse avoir l'initiative d'une interaction avec l'apprenant de manière, par exemple, à lui poser une question afin de vérifier qu'un concept a été acquis [4].

Le fait d'élargir ces domaines d'application aux activités moins procédurales pose également le problème de l'immersion et de l'interaction naturelle de l'apprenant avec le système. Nous supposons que plus l'interaction avec le système est naturelle, meilleurs sont l'apprentissage et le transfert de l'apprentissage à l'environnement réel. Toutefois, plus l'interaction est naturelle, moins il est facile pour nos modèles de reconnaître les actions que l'apprenant a réalisé ou souhaité réaliser. De plus, certaines situations pédagogiques pourraient bénéficier de la complémentarité entre les situations réelle et virtuelle. Il s'agirait ici d'utiliser les techniques de réalité augmentée. Cela nécessitera de trouver des solutions pour garder la cohérence entre la simulation en MASCARET et l'environnement réel.

D'un point de vue didactique, deux problèmes restent à aborder. D'une part, POSEIDON permet d'exprimer les objectifs de réalisation, c'est-à-dire l'état du monde souhaité, mais pas les objectifs pédagogiques en termes de compétences à acquérir. Des modèles de ces objectifs pédagogiques existent. Il s'agit de voir dans quelle mesure il serait possible d'en tenir compte dans POSEIDON. D'autre part, nous avons montré que l'activité du didacticien peut être généralisée, ce qui nous a permis de proposer des outils de scénarisation pédagogique. Toutefois, le scénario que le formateur construit dépend fortement de l'activité spécifique et il préfère alors utiliser un outil complètement adapté au métier auquel il veut former [5]. Nous devons donc proposer des moyens de traduire les concepts généraux que nous avons identifiés vers des outils spécifiques à chaque domaine.

La modélisation d'un système complexe est une tâche ardue, coûteuse en temps et en ressources financières et humaines, et source d'erreurs. La contrainte que nous nous imposons dans nos travaux est de ne faire intervenir l'expert métier qu'une seule fois. Le même modèle métier doit donc servir aux différents cas d'utilisation. Nous avons pour l'instant abordé l'utilisation du modèle dans le cadre de la simulation en réalité virtuelle et la formation. Nous souhaitons démontrer que MASCARET permet également de modéliser le système afin de générer automatiquement la documentation et le prototype réel [6].

4. Ces travaux font l'objet de la thèse de F. Lecorre encadrée par C. Buche, soutenance prévue en 2012.

5. Une thèse sur la formation des chirurgiens dentistes grâce à la réalité virtuelle est menée par J. Cormier, encadrée par D. Pasco et P. Chevaillier

6. Un projet dans ce sens à été déposé en collaboration avec F. Ganier (Psycho-ergonome au CERV) auprès du GDR « Production écrite : apprentissage et expertise ».

Bibliographie

[Abaci et al., 2006] Abaci, T., Cíger, J., and Thalmann, D. (2006). Planning with smart objects. In *Proceedings of WSCG 2005 (short paper)*, pages 25–28.

[Anderson, 1995] Anderson, J. (1995). *Learning and memory : An integred approach.* John Wiley and Sons.

[Anderson, 1993] Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ : Erlbaum.

[Anonyme, 2006] Anonyme (2006). Meta object facility (mof) 2.0 core specification. Technical report, Object Management Group. formal/06-01-01.

[Anonyme, 2007] Anonyme (2007). Unified modeling language : Superstructure, version 2.1.1. Technical report, Object Management Group. formal/2007-02-05.

[Aubry et al., 2007] Aubry, S., Thouvenin, I., Lenne, D., and Okawa, S. (2007). Knowledge integration for annoting in virtual environments. *International Journal of Product Development*, 5(6) :533–546.

[Aylett and Luck, 2000] Aylett, R. and Luck, M. (2000). Applying artificial intelligence to virtual reality : Intelligent virtual environnements. *Applied Artificial Intelligence*, 14(1) :3–23.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuiness, D. L., Nardi, D., and Patel-Schneider, P. F. (2003). *The Description Logic Handbook : Theory, Implementation, Applications*. Cambridge University Press, Cambridge, UK.

[Badawi and Donikian, 2004] Badawi, M. and Donikian, S. (2004). Autonomous agents interacting with their virtual environment through synoptic objects. In *Proceedings of CASA'04*, pages 179–187. The Computer Graphic Society.

[Badawi and Donikian, 2007] Badawi, M. and Donikian, S. (2007). The generic description and management of interaction between autonomous agents and objects in an informed virtual environment. *Computer Animation and Virtual Worlds*, 18(4-5) :559–569.

[Baudouin et al., 2008] Baudouin, C., Chevaillier, P., Le Pallec, A., and Beney, M. (2008). Feedback on design and use of a virtual environment for practical lab work. In *Proceedings of the Virtual Reality International Conference, VRIC 2008*, pages 117–125.

[Bauer et al., 2001] Bauer, B., Muller, J., and Odell, J. (2001). Agent uml : A formalism for specifying multiagent software systems. In *Agent-Oriented Sofware Engineering*, volume 1957 of *Lecture Notes in Computer Science*, pages 109–120. Springer Berlin / Heidelberg.

[Beydoun et al., 2009] Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J., Pavon, J., and Gonzales-Perez, C. (2009). Faml : A generic metamodel for mas development. *IEEE Transactions on Software Engineering*, 35(6) :841–863.

[Briot et al., 2006] Briot, J.-P., Meurisse, T., and Peschanski, F. (2006). Une expérience de conception et de composition de comportements d'agents à l'aide de composants. *Revue des sciences et technologies de l'information, série L'Objet*, 12(4) :11–41. numéro spécial "Composants et systèmes multi-agents", sous la direction d'Olivier Boissier.

[Brown and Burton, 1978] Brown, J. S. and Burton, R. R. (1978). A paradigmatic example of an artificially intelligent instructional system. *International Journal of Man-Machine Studies*, 10 :323–339.

[Buche, 2005] Buche, C. (2005). *Un système tutoriel intelligent et adaptatif pour l'apprentissage de compétences en environnement virtuel de formation.* Thèse de doctorat, Université de Bretagne Occidentale.

[Buche et al., 2010] Buche, C., Bossard, C., Querrec, R., and Chevaillier, P. (2010). Pegase : A generic and adaptable intelligent system for virtual reality learning environments. *Internation Journal of Virtual Reality*, 9(2) :1–13.

[Buche et al., 2005] Buche, C., Querrec, R., Chevaillier, P., and Kermarrec, G. (2005). Apports des systèmes tutoriaux intelligents et de la réalité virtuelle à l'apprentissage de compétences. *Cahiers Romans de Sciences Cognitives, In Cognito – ISSN 1267–8015*, 2(2) :53–87.

[Buche et al., 2009] Buche, C., Querrec, R., Loor, P. D., Chevaillier, P., and Tisseau, J. (2009). PEGASE : un système tutoriel intelligent générique et adaptatif en environnement virtuel. *Revue des sciences et technologies de l'information, série Techniques et Sciences Informatiques*, 28(8) :1051–1076.

[Buche et al., 2006] Buche, C., Septseault, C., and De Loor, P. (2006). Les systèmes de classeurs. une présentation générale. *Revue des Sciences et Technologies de l'Information, série Techniques et Sciences Informatiques (RSTI–TSI)*, 25 :963–990.

[Burkhardt et al., 2003] Burkhardt, J. M., Lourdeaux, D., and Mellet d'Huart, D. (2003). *Le traité de la réalité virtuelle*, chapter La conception des environnements virtuels pour l'apprentissage. Les Presses de l'Ecole des Mines, deuxiÃ¨me edition.

[Cazeaux et al., 2005] Cazeaux, E., Devillers, F., Saint-Romas, C., Arnaldi, B., Maffre, E., Mollet, N., and Tisseau, J. (2005). Giat virtual training : Formation Ã la maintenance. In *Laval Virtual*.

[Chevaillier et al., 2009] Chevaillier, P., Querrec, R., and Septseault, C. (2009). VEHA : Un métamodèle d'environnement virtuel informé et structuré. *Revue des sciences et tecnologies de l'information, série Techniques et Sciences Informatiques*, 28(6/7) :63–88.

[Chou et al., 2003] Chou, C.-Y., Chan, T.-W., and Lin, C.-J. (2003). Redefining the learning companion : the past, present, and future of educational agents. *Computers & Education*, 40(3) :255–569.

[Delestre, 2000] Delestre, N. (2000). *METADYNE, un Hypermèdia Adaptatif Dynamique pour l'Enseignement.* PhD thesis, Université de Rouen, France. Informatique.

[Donikian, 2001] Donikian, S. (2001). HPTS : a behaviour modelling language for autonomous agents. In *Proc. of the Fith International Conference on Autonomous Agents*, Montréal (Canada).

[Doyle, 2002] Doyle, P. (2002). Believability through context using "knowledge in the world"to create intelligent characters. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 342–349. ACM Press.

[Doyle and Hayes-Roth, 1998] Doyle, P. and Hayes-Roth, B. (1998). Agents in annotated worlds. In Sycara, K. P. and Wooldridge, M., editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 173–180, New York. ACM Press.

[Ehrler and Cranefield, 2004] Ehrler, L. and Cranefield, S. (2004). Executing agent uml diagrams. In *Autonomous Agent and Multi-Agent System 2004*, pages 906–913, New York, USA.

[Ferber and Gutknecht, 2000] Ferber, J. and Gutknecht, O. (2000). Operational semantics of multi-agent organizations. In Jennings, N. and LespÃ©rance, Y., editors, *Intelligent Agents VI. Agent Theories Architectures, and Languages*, volume 1757 of *Lecture Notes in Computer Science*, pages 205–217. Springer Berlin / Heidelberg.

[Fuchs, 2006] Fuchs, P. (2006). Approche théorique et pragmatique de la réalité virtuelle. In *Le traité de la réalité virtuelle*, volume 2, chapter 2, pages 19–59. Les Presses de l'École des Mines de Paris, 3 edition.

[Fuchs and Burkhardt, 2003] Fuchs, P. and Burkhardt, J. M. (2003). *Le traité de la réalité virtuelle*, chapter Approche théorique et pragmatique de la réalité virtuelle. Les Presses de l'Ecole des Mines, deuxiÃ¨me edition.

[Fuchs and Moreau, 2003] Fuchs, P. and Moreau, G. (2003). *Le traité de la réalité virtuelle*. Presses de l'école des mines, Paris, seconde edition.

[Ganier and Querrec, 2008] Ganier, F. and Querrec, R. (2008). Tip-exe : Editeur d'-expériences et d'évaluations de documents procéduraux. In *De la France au Québec : l'écriture dans tous ses états*, pages 40–42, Poitiers (France).

[Gao et al., 2005] Gao, Z., Ren, L., Qu, Y., and Ishida, T. (2005). Virtual space ontologies for scripting agents. In *Massively Multi-Agent Systems I*, volume 3446 of *Lecture Notes in Computer Science*, pages 70–85. Springer Berlin / Heidelberg.

[Gerbaud et al., 2008] Gerbaud, S., Mollet, N., Ganier, F., Arnaldi, B., and Tisseau, J. (2008). Gvt : a platform to create virtual environments for procedural training. *IEEE Virtual Reality*, 1(1) :225–232.

[Gonçalves et al., 2001] Gonçalves, L. M. G., Kallmann, M., and Thalmann, D. (2001). Programming behaviors with local perception and smart objects : An approach to solve autonomous agents tasks. In *Proceedings of the XIV Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'01)*, pages 184–191. IEEE Computer Society.

[Guéraud et al., 2004] Guéraud, V., Adam, J.-M., Pernin, J.-P., Calvary, G., and David, J.-P. (2004). L'exploitation d'objets pédagogiques interactifs à distance : le projet FORMID. *STICEF*, 11 :103–163.

[Guéraud and Cagnat, 2006] Guéraud, V. and Cagnat, J.-M. (2006). Automatic semantic activity monitoring of distance learners guided by pedagogical scenarios. In Nejdl, W. and Tochtermann, K., editors, *Innovative Approaches for Learning and Knowledge Sharing*, volume 4227 of *Lecture Notes in Computer Science*, pages 476–481. Springer Berlin / Heidelberg.

[Hahn et al., 2009] Hahn, C., Madrigal-Mora, C., and Fisher, K. (2009). A plateform-independent metamodel for multiagent systems. *Autonomous Agent and Multi-Agent System*, 18(2) :239–266.

[Harrouet et al., 2006] Harrouet, F., Cazeaux, E., and Jourdan, T. (2006). ARéVi. In Fuchs, P., Moreau, G., and Tisseau, J., editors, *Le traité de la Réalité Virtuelle*, volume 3, pages 369–392. Les Presses de l'École des Mines, 3e edition.

[Henri et al., 2007] Henri, F., Compte, C., and Charlier, B. (2007). La scénarisation pédagogique dans tous ses débats. *Revue internationale des technologies en pÃ©dagogie universitaire*, 4(2) :14–24.

[Houssaye, 1988] Houssaye, J. (1988). *Théorie et pratiques de l'éducation scolaire I : Le triangle pédagogique*. Peter Lang.

[Hubner et al., 2010] Hubner, J., Boissier, O., Kitio, R., and Ricci, A. (2010). Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agent and Multi-Agent System*, 20(3) :369–400.

[Huget and Odell, 2004] Huget, M.-P. and Odell, J. (2004). Representing agent interaction protocols with agent uml. In *Autonomous Agent and Multi-Agent System 2004*, pages 1244–1245, New York, USA.

[Joab et al., 2002] Joab, M., Auzende, O., Futtersack, M., Bonnet, B., and Le Leydour, P. (2002). Computer aided evaluation of trainee skills on a simulator network. In Cerri, S., GouardÃ¨res, G., and ParaguaÃ§u, F., editors, *Intelligent Tutoring Systems*, volume 2363 of *Lecture Notes in Computer Science*, pages 521–530. Springer Berlin / Heidelberg.

[Kallmann and Thalmann, 1998] Kallmann, M. and Thalmann, D. (1998). Modeling objects for interaction tasks. In *Proceedings of Computer Animation and Simulation'98*, pages 73–86.

[Kermarrec, 2002] Kermarrec, G. (2002). *Stratégies d'apprentissage et autorégulation en EPS, une recherche descriptive en contexte scolaire*. PhD thesis, Université de Rennes 2.

[Koper and Manderveld, 2004] Koper, R. and Manderveld, J. (2004). Educational modelling language : modelling reusable, interoperable, rich and personalised units of learning. *British Journal of Educational Technology*, 35 :537–551.

[Lamarche and Donikian, 2004] Lamarche, F. and Donikian, S. (2004). Crowd of virtual humans : a new approach for real time navigation in complex and structured environments. *Compututer Graphics Forum*, 23(3) :509–518.

[Leroux, 1999] Leroux, P. (1999). Roboteach - a pedagogical environment in technology learning. In *Workshop on Educational Robotic, International Conference on Artificial Intelligence in Education (AI-ED'99)*, pages 57–66, Le Mans (France).

[Levesque, 2003] Levesque, P. (2003). Creation and use of 3d as-built models at EDF. In *FIG Working Week 2003*.

[Lourdeaux, 2001] Lourdeaux, D. (2001). *Réalité virtuelle et formation : conception d'environnements virtuels pédagogiques*. Thèse de doctorat, Ecole des mines de Paris.

[Lugrin and Cavazza, 2007] Lugrin, J.-L. and Cavazza, M. (2007). Making sense of virtual environments : Action representation, grounding and common sense. In *ACM Intelligent User Interfaces*, pages 647–652, Honolulu, Hawaii.

[Marilleau et al., 2008] Marilleau, N., Cambier, C., Drogoul, A., Chotte, L.-L., Perrier, E., and Blanchart, E. (2008). Environnement multi-echelle Ã base de fractales pour la modélisation agent d'écosystèmes. In Mandiau, R. and Chevaillier, P., editors, *Journées Francophones sur les Systèmes Multi-Agents*, pages 23–32, Brest, France.

[Marion, 2010] Marion, N. (2010). *Modélisation de scénarios pédagogiques pour les environnements de réalité virtuelle d'apprentissage humain.* Thèse de doctorat, Université de Bretagne occidentale.

[Marion et al., 2009] Marion, N., Querrec, R., and Chevaillier, P. (2009). Integrating knowledge from virtual reality environments to learning scenario models. a meta-modeling approach. In *International conference of computer supported education*, pages 254–259, Lisbonne (Portugal).

[Marion et al., 2007] Marion, N., Septseault, C., Boudinot, A., and Querrec, R. (2007). GASPAR : Aviation management on aircraft carrier using virtual reality. In *Proceedings of Cyberworlds*, Hanovre (Allemagne).

[Mathieu and Picault, 2005] Mathieu, P. and Picault, S. (2005). Towards an interaction–based design of behaviors. In *Proceedings of the The Third European workshop on Multi-Agent Systems (EUMAS'05)*, Brussel (Belgium).

[Montealegre Vazquez and Lopez y Lopez, 2007] Montealegre Vazquez, L. and Lopez y Lopez, F. (2007). An agent-based model for hierarchical organizations. In Noriega, P., Vazquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., and Matson, E., editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, volume 4386 of *Lecture Notes in Computer Science*, pages 194–211. Springer Berlin / Heidelberg.

[Munro et al., 1997] Munro, A., Johnson, M., Pizzini, Q., Surmon, D., Towne, D., and Wogulis, J. (1997). Authoring simulation-centered tutors with RIDES. *International Journal of Artificial Intelligence in Education*, 8(3-4) :284–316.

[Ochs et al., 2008] Ochs, M., Sabouret, N., and Corruble, V. (2008). Un modèle de la dynamqiue des relations sociales pour des agents virtuels. In Mandiau, R. and Chevaillier, P., editors, *Journées Francophones sur les Systèmes Multi-Agents*, pages 87–96, Brest, France.

[Omicini and Ricci, 2004] Omicini, A. and Ricci, A. (2004). Mas organization within a coordination infrastructure : Experiments in tucson. In Omicini, A., Petta, P., and Pitt, J., editors, *Engineering Societies in the Agents World*, volume 3071 of *Lecture Notes in Computer Science*, pages 520–520. Springer Berlin / Heidelberg.

[Paris et al., 2006] Paris, S., Donikian, S., and Bonvalet, N. (2006). Environmental abstraction and path planning techniques for realistic crowd simulation : Research articles. *Computer Animation and Virtual Worlds*, 17(3-4) :325–335.

[Parunak and Odell, 2002] Parunak, H. V. D. and Odell, J. (2002). Representing social structures in UML. In Wooldridge, M. J., Weiss, G., and Ciancarini, P., editors, *Agent Oriented Software Engineering Workshop (AOSE 2001), International Conference on Autonomous Agents*, volume 2222 of *Lecture Notes in Computer Science*, pages 1–16. Springer–Verlag (Berlin).

[Patel-Schneider et al., 2004] Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2004). Owl web ontology language semantics and abstract syntax. W3C Recommandation REC-owl-semantics-20040210, W3C.

[Payr, 2003] Payr, S. (2003). The virtual university's faculty : An overview of educational agents. *Applied Artificial Intelligence*, 17(1) :1–19.

[Pellens et al., 2006] Pellens, B., Kleinermann, F., and Troyer, O. D. (2006). Intuitively specifying object dynamics in virtual environments using vr-wise. In *ACM Symposium on*

Virtual Reality Software and Technology, pages 334–337, Limassol, Cyprus. ACM Press, ISBN 1-59593-321-2.

[Pernin and Lejeune, 2004] Pernin, J.-P. and Lejeune, A. (2004). Dispositifs d'apprentissage instrumentés par les technologies : vers une ingénierie centrée sur les scénarios. *Colloque Technologies de l'Information et de la Connaissance dans l'Enseignement supérieur et l'industrie (TICE'04), Compiègne, France.*

[Py, 1998] Py, D. (1998). Quelques méthodes d'intelligence artificielle pour la modèlisation de l'élève. *Sciences et Techniques Educatives*, 5(2) :123–140.

[Querrec, 2001] Querrec, R. (2001). *Les systèmes multi-agents pour les environnements virtuels de formation. Application à la sécurité civile.* Thèse de doctorat, Université de Bretagne Occidentale.

[Querrec et al., 2004] Querrec, R., Buche, C., Maffre, E., and Chevaillier, P. (2004). Multi-agents systems for virtual environment for training. application to fire-fighting. *Advanced Technology for Learning*, 1(1) :25–34.

[Rasmussen, 1986] Rasmussen, J. (1986). *Information Processing and Human and Machine Interaction : An approch to cognitive engineering.* Elsevier Science Publishers, New York, NY.

[Richard, 1990] Richard, J. F. (1990). *Les activitÃ©s mentales : Comprendre, Raisonner, Trouver des solutions.* Armand Colin, Paris. ISBN 2-200-2187-0.

[Rickel and Johnson, 1999] Rickel, J. and Johnson, W. L. (1999). Animated agents for procedural training in virtual reality : Perception, cognition, and motor control. *Applied Artificial Intelligence*, 13 :343–382.

[Rimassa, 2003] Rimassa, G. (2003). *Runtime Support for Distributed Multi-Agent Systems.* PhD thesis, University of Parma.

[Rouse, 1982] Rouse, W. B. (1982). Models of human problem solving : Detection, diagnosis and compensation for system failures. *Automatica*, 19 :613–625.

[Sequeira et al., 2007] Sequeira, P., Vala, M., and Paiva, A. (2007). What can I do with this ? : finding possible interactions between characters and objects. In *AAMAS '07 : Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–7, New York, NY, USA. ACM.

[Stansfield et al., 1976] Stansfield, J., Carr, B., and Goldstein, I. P. (1976). Wumpus advisor 1 : a first implementation of a program that tutors logical and probabilistic reasoning skills. Technical Report AIM-381, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

[Thouvenin, 2009] Thouvenin, I. M. (2009). *Interaction et connaissance : construction d'une expérience dans le monde virtuel.* Habilitation à Diriger des Recherches, Université de Technologie de Compiègne.

[Torres DaSilva et al., 2004] Torres DaSilva, V., Choren, R., and J.P. De Lucena, C. (2004). A uml based approach for modeling and implementing multi-agent systems. In *Autonomous Agent and Multi-Agent System 2004*, pages 914–921, New York, USA.

[Trinh et al., 2009] Trinh, T., Buche, C., Querrec, R., and Tisseau, J. (2009). Modeling of errors realized by a human learner in virtual environment for training. *International Journal of Computers, Communications and Control*, IV(1) :73–81.

[Trinh et al., 2010] Trinh, T.-H., Querrec, R., Loor, P. D., and Chevaillier, P. (2010). Ensuring semantic spatial constraints in virtual environments using UML/OCL. In *VRST'2010 : Proceedings of the 2010 ACM symposium on Virtual reality software and technology*, pages ? ? – ? ? ACM.

[Van Joolingen and de Jong, 2003] Van Joolingen, W. and de Jong, T. (2003). SimQuest, authoring educational simulations. *Authoring tools for advanced technology educational software : Toward cost-effective production of adaptive, interactive, and intelligent educational software*, pages 1–31.

[Wenger, 1987] Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems*. Morgan Kaufmann, Los Altos, California.

[Wooldridge et al., 2000] Wooldridge, M., Jennings, N., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Autonomous Agent and Multi-Agent System*, 3(3) :285–312.