UN MODÈLE DE REPRÉSENTATION DES CONNAISSANCES À TROIS NIVEAUX DE SÉMANTIQUE POUR LES SYSTÈMES TUTORIELS INTELLIGENTS

par

Philippe Fournier-Viger

mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, décembre 2005

Le 9 décembre 2005

le jury a accepté le mémoire de M. Philippe Fournier-Viger dans sa version finale.

Membres du jury

M. André Mayers
Directeur
Département d'informatique

M. Roger N'Kambou
Membre
Département d'informatique - Université du Québec à Montréal

Mme Hélène Pigot
Président-rapporteur
Département d'informatique

SOMMAIRE

Ce mémoire s'inscrit dans le cadre du projet *ASTUS* qui vise le développement d'un système tutoriel intelligent (*STI*). Les connaissances sont un élément crucial pour ces systèmes, car elles constituent le langage commun entre les différents modules. Le mémoire propose un modèle original de représentation des connaissances pour les *STI* qui tire profit de trois approches prometteuses : la représentation psychologique et didactique, la représentation logique et ontologique des logiques de description, et la notion d'objet d'apprentissage utilisée dans le domaine de la formation en ligne. De cette combinaison, résulte un modèle novateur avec des caractéristiques avantageuses, qui établira un fondement solide pour le système en développement.

REMERCIEMENTS

Je remercie tout d'abord André Mayers, mon directeur de recherche, pour sa grande disponibilité, ses précieux commentaires et pour toutes les belles opportunités qu'il m'a offertes.

Je suis également très reconnaissant envers l'ensemble des membres du groupe de recherche *ASTUS* et plus particulièrement envers Mehdi Najjar et Mohamed Hamadouche qui m'ont donné nombre de bons conseils, me faisant ainsi bénéficier de leur expérience et avec qui j'ai eu beaucoup de plaisir à collaborer.

J'adresse aussi des remerciements à Hélène Pigot et Roger Nkambou, membres du jury, pour la lecture de ce mémoire et leurs commentaires.

Je tiens également à remercier l'ensemble du personnel de l'Université de Sherbrooke et plus particulièrement le personnel du Département d'informatique pour leur travail quotidien essentiel.

Je souhaite finalement présenter des remerciements à mes parents Jean Viger et Marie Fournier pour leur immense soutien, ainsi qu'à ma compagne Chantal Gaignard.

TABLE DES MATIÈRES

Liste des abréviations

ACT-R : Adaptive Control of Thought - Rational

ASTUS : Apprentissage des Sciences par Tuteur intelligent de l'Université de Sherbrooke

CAI : Computer Aided Instruction

CKTN : Curriculum Knowledge Transition Network

CREAM : Curriculum REprensentation and Acquisition Model

CS : Contention Scheduling

CSS : Content Scramble System

DAML+OIL : Darpa Agent Markup Language + Ontology Inference Layer

DIG : Description logics Implementation Group

DOKGETT : DOmain Knowledge GEnerator auThoring Tool

DVD : Digital Versatile Disc

EAO : Enseignement Assisté par Ordinateur

GPD : Gestionnaire des Priorités de Développement

HNU : Hypothèse de Noms Uniques

HTTP : HyperText Transfer Protocol

IDC : International Data Corporation

IMS-CP : IMS Content Packaging

IMS-LD : IMS Learning Design

LD : Logiques de Description

LDE : Logiques de Description Expressives

LMS : Learning Management System

LOM : Learning Object Metadata

MERLOT : Multimedia Educational Ressources for Learning and Online Teaching

MGC : Module de Gestion des Connaissances

MIACE : Modèle Informatique de l'Acquisition des Connaissances par un Élève

MISA : Méthode d'Ingénierie des Systèmes d'Apprentissage

OIL : Ontology Inference Layer

OWL : Web Ontology Language

RDF : Resource Description Framework

SAS : Système Attentionnel Superviseur

SPARC : Scalable Processor ARChitecture

STI : Système Tutoriel Intelligent

SPK : Semantic and Procedural Knowledge

TML : Tutorial Markup Language

UML : Unified Modeling language

UV : Usager Virtuel

XML : eXtensible Markup Language

XSLT : eXtensible Stylesheet Language Tranformations

LISTE DES TABLEAUX

LISTE DES FIGURES

Introduction

Ce mémoire s'inscrit dans le cadre du projet *ASTUS* (Apprentissage des Sciences par Système Tutoriel intelligent de l'Université de Sherbrooke) [1], un système tutoriel intelligent (*STI*) qui propose aux apprenants d'effectuer des activités pédagogiques dans des laboratoires virtuels sous la supervision d'un tuteur virtuel intelligent. Ce système, développé à l'Université de Sherbrooke, a pour but de faciliter l'apprentissage des domaines scientifiques ou techniques. La clientèle visée de ce projet sont les étudiants en sciences et le personnel technique en formation. L'objectif de ce travail est de développer un modèle de représentation des connaissances, novateur et adapté aux spécificités d'*ASTUS*. Ce chapitre introductif adopte la structure suivante : il (1) décrit le concept de *STI*, (2) définit les caractéristiques d'*ASTUS*, (3) présente la problématique de la représentation des connaissances et l'objectif du mémoire puis (4) expose la méthodologie adoptée.

L'enseignement assisté par ordinateur (EAO)

Le terme EAO (en anglais : Computer Aided Instruction ou CAI) désigne la première génération de systèmes d'enseignement par ordinateur, apparue au début des années 1970. Ce chapitre présente cette classe de systèmes pour mieux mettre en relief par la suite les caractéristiques des STI. Les logiciels EAO proposent des activités éducatives simples

[1] *ASTUS* est simultanément le nom d'un groupe de recherche, de son principal projet de recherche et du système tutoriel intelligent qu'il développe.

sur ordinateur telles que des questions à choix multiples. Les concepteurs prédéfinissent la totalité des rétroactions : les décisions pédagogiques sont statiques et l'enseignement n'est pas individualisé. Par conséquent, tout apprenant qui commet une erreur dans une activité d'apprentissage recevra la même rétroaction préétablie du système. Ces systèmes, malgré leur simplicité, sont des outils pédagogiques très efficaces lorsqu'ils sont conçus adéquatement (voir par exemple, les logiciels de Dugdale (1992) et Culley (1992)). Toutefois, certaines limites sont réelles. La première limite est la non-personnalisation de l'enseignement. Une étude a montré que le tutorat améliore les résultats scolaires des apprenants jusqu'à deux écarts-types au-dessus de ceux instruits dans des classes traditionnelles (Bloom, 1984). En d'autres mots, pour atteindre une efficacité maximale, un système d'enseignement doit tenir compte des caractéristiques individuelles (bagage de connaissances, préférences, intentions, etc.). La seconde limite des logiciels EAO est intrinsèque au fait que tout est préétabli : puisque les concepteurs doivent prévoir chaque situation, l'interactivité est nécessairement restreinte.

Les systèmes tutoriels intelligents

Les STI pallient théoriquement les limites mentionnées des logiciels EAO. Évidemment, en pratique, la frontière est floue entre STI et EAO : la plupart des systèmes se retrouvent quelque part entre ces deux définitions. L'idée fondamentale des STI est d'offrir un enseignement personnalisé. Pour accomplir cet objectif, plutôt que d'encoder les décisions pédagogiques implicitement dans le contenu, les connaissances sont encodées dans une forme neutre et le système utilise des stratégies pédagogiques pour bâtir un enseignement dynamique (Wenger, 1987). Plusieurs architectures de STI ont été proposées (Payadachee, 2002). Cette section présente l'architecture classsique des systèmes tutoriels intelligents, formée des composantes suivantes (cf. figure 1) : un module expert, un module « modèle de l'apprenant », un module « interface de l'utilisateur » et un module tuteur (Wenger, 1987; Burns et Capps, 1988). La description de base des quatre modules qui suit, s'appuie

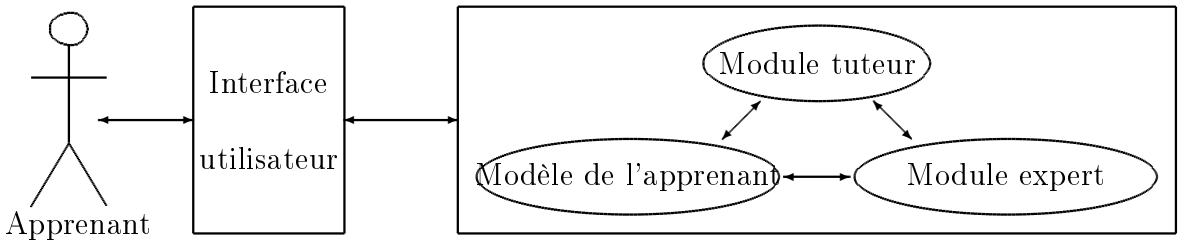

Figure 1 – Les composantes typiques d'un STI

sur le texte de Wenger quoiqu'enrichie par quelques textes plus récents.

Le module expert

Un STI doit, tout comme un enseignant humain, maîtriser les notions qu'il enseigne. C'est le rôle du module expert. Dans ce module, les connaissances du domaine sont encodées explicitement. Elles doivent être suffisamment précises pour que le système soit capable d'effectuer de l'inférence et de simuler le comportement d'un expert humain. La simulation d'un expert humain sur un problème du domaine, permet (1) de montrer les étapes de résolution du problème à l'apprenant et (2) de comparer les actions de l'apprenant pour déterminer si sa solution est correcte ou s'il est dans la bonne voie (Corbett et Anderson, 1992).

Le module « modèle de l'apprenant »

Le module « modèle de l'apprenant », indispensable à la personnalisation, est l'un des éléments fondamentaux des STI. Il contient la représentation par le système de l'apprenant. Bâtir ce portrait est une tâche difficile, étant donné les limites des ordinateurs et l'exiguïté du canal de communication (souvent un moniteur, un clavier et une souris). Le modèle construit ne couvre qu'un aspect sommaire de l'apprenant : le STI enregistre seulement les données pertinentes à la réalisation de l'activité pédagogique. Les données

exactes recueillies dépendent fortement des besoins du module tuteur. Un module « modèle de l'apprenant » pourrait contenir les informations suivantes : des degrés de certitude sur la maîtrise de chaque notion, les préférences de l'apprenant, les intentions perçues de l'apprenant, un historique des actions de l'apprenant et un historique des résultats de l'apprenant.

La relation avec les connaissances du module expert

Le module « modèle de l'apprenant » emploie généralement les mêmes structures de représentation des connaissances que le module expert afin de faciliter la mise en correspondance des acquis de ces deux modules. Deux approches prédominent pour mettre en relation les connaissances du module « modèle de l'apprenant » avec celles du module expert (VanLehn, 1988) :

- Le *modèle par recouvrement* représente le savoir de l'apprenant comme un sous-ensemble de celui de l'expert. Ce modèle attribue les erreurs de l'apprenant à l'ignorance d'éléments de connaissance nécessaires à l'accomplissement d'un but (Carr et Goldstein, 1977).
- Le *modèle par perturbation* exprime les connaissances de l'apprenant comme l'ensemble des connaissances de l'expert augmenté de connaissances erronées dont l'utilisation mène à l'erreur (Brown et Burton, 1978).

Dans les deux cas, l'encodage des connaissances doit être suffisamment précis pour représenter les conceptions erronées, sans toutefois modéliser des détails inutiles.

L'utilisation du module « modèle de l'apprenant »

Le principal bénéfice de l'inclusion d'un module « modèle de l'apprenant » réside dans la capacité à donner des explications et à générer des problèmes en fonction des connaissances estimées de l'apprenant. Cependant, un module « modèle de l'apprenant » ouvre d'autres possibilités. D'une part, un module « modèle de l'apprenant » qui possède des

fonctionnalités pour simuler le comportement de l'apprenant sur des problèmes du domaine, permettra à un système de tester sur cet apprenant virtuel des exercices avant de les donner à un apprenant réel (VanLehn et al., 1994). De cette façon, le système pourra valider ses choix pédagogiques. D'autre part, un module « modèle de l'apprenant » permet à un module tuteur d'intervenir aux moments propices tels que certaines approches éducationnelles le prescrivent et non pas seulement lorsque l'apprenant commet une erreur. En effet, juger les circonstances idéales d'intervention nécessite la connaissance de l'état des acquis de l'apprenant.

Le module tuteur

Le module tuteur prend les décisions pédagogiques. Ses trois tâches principales sont : (1) de planifier et d'adapter un programme d'études (une séquence appropriée d'activités), (2) de choisir la nature des interactions et (3) de déterminer les circonstances idéales pour intervenir (Wenger, 1987). Le module tuteur a recours au module expert et au module « modèle de l'apprenant » pour mener à bien ces tâches.

Le module « interface de l'utilisateur »

Le module « interface de l'utilisateur » gère la communication avec l'apprenant. Il offre une vue graphique sur le système et des outils pour le manipuler. La conception adéquate de ce module constitue un facteur critique de l'efficacité d'un *STI*, d'autant plus que l'apprenant ne peut pas recourir à la connaissance du domaine pour comprendre le fonctionnement de l'interface (Frye et Soloway, 1987). Une interface intuitive, bien conçue et adaptée au niveau de développement de l'apprenant, lui permettra d'allouer davantage de ressources cognitives au problème à résoudre (Bouchard, 2005; Frye et Soloway, 1987).

Le projet *ASTUS*

Cette section présente *ASTUS*, un STI spécialisé pour l'enseignement des domaines scientifiques et techniques qui sont formellement descriptibles. Les prochaines sous-sections décrivent (1) l'organisation macroscopique des modules logiciels d'*ASTUS*, (2) l'approche pédagogique du tuteur virtuel d'*ASTUS* et (3) la description individuelle des principaux modules d'*ASTUS*.

L'organisation macroscopique des modules logiciels d'*ASTUS*

ASTUS se veut un STI polyvalent, c'est-à-dire que son adaptation à divers domaines nécessite peu d'effort. Afin d'y arriver, *ASTUS* se base sur une architecture modulaire (Hamadouche, 2004) divisée en deux parties : le *cadre d'applications* et les *laboratoires virtuels*. La première est la partie inaltérable d'*ASTUS*, conçue entièrement en langage de programmation *Java* et qui encapsule les composantes de base d'*ASTUS* désignées par le terme *composantes immuables*. La seconde représente des configurations d'éléments logiciels spécifiques à l'enseignement d'un domaine : par exemple, un ensemble de composantes pour enseigner la chimie. Le terme *composantes substituables* désigne ces composantes. Le couplage du cadre d'applications à un laboratoire virtuel forme un *environnement d'apprentissage complet*. Jusqu'à présent, trois laboratoires virtuels ont été conçus :

- un laboratoire sur la réduction booléenne (Najjar et al., 2004b; Najjar et Mayers, 2004; Najjar et al., 2004a),
- un laboratoire de génie génétique (Zhao, 2003) et
- un laboratoire de conversion d'instructions en binaire, pour un cours sur l'architecture des ordinateurs (Hamadouche, 2004).

L'approche pédagogique du tuteur virtuel d'*ASTUS*

L'approche pédagogique de base retenue pour *ASTUS* est celle de Rieber (1992) qui propose une combinaison judicieuse de l'apprentissage par micromonde et de l'instructionnisme. Les paragraphes suivants décrivent brièvement (1) l'apprentissage par micromonde, (2) l'instructionnisme, (3) la combinaison proposée par Rieber et (4) l'approche pédagogique du tuteur virtuel d'*ASTUS*.

C'est Papert (1981) qui a introduit le concept de micromonde. Un micromonde est une simulation d'un monde réel, composé d'objets et d'outils, les outils permettant de manipuler les objets. L'acquisition des connaissances se fait par l'expérimentation : l'apprenant construit des hypothèses et les expérimente dans le micromonde. De cette façon, il assimile la dynamique du monde réel représenté par le micromonde. Dans la définition classique de micromonde (Papert, 1981), l'apprenant est le seul responsable de son apprentissage. Il guide son parcours dans le micromonde, au risque d'errer, d'autant plus qu'un apprenant est généralement un mauvais juge de son apprentissage (Rieber, 1994).

L'instructionnisme (Rieber, 1994) définit l'enseignement comme un processus impliquant deux acteurs : l'apprenant et le tuteur. Selon cette théorie, l'apprentissage résulte de l'acquisition de connaissances communiquées explicitement par le tuteur, dans le cadre de *leçons*. Seul le tuteur détermine la structure et le contenu des leçons. Cette contrainte a le désavantage de pouvoir susciter l'ennui ou le mécontentement de l'apprenant.

Rieber (1992) propose de combiner ces deux approches, en incorporant l'instruction par un tuteur à des séances d'exploration d'un micromonde par l'apprenant. Actuellement, une version rudimentaire du module tuteur d'*ASTUS* a été complétée. Cette première implantation offre quatre variations de l'approche de Rieber, décrite ci-haut (Hafsaoui, 2005) :

mode guidé : Le tuteur dicte une séquence d'exercices à réaliser dans le micromonde. Le tuteur intervient si l'apprenant se trompe pour un exercice ou ne répond pas dans les délais.

mode semi-guidé : L'apprenant explore le micromonde librement. Le tuteur infère les intentions de l'apprenant. Il intervient seulement s'il perçoit des buts incorrectement réalisés.

mode libre : L'apprenant explore le micromonde librement et demande l'assistance du tuteur si nécessaire.

mode sans intervention : L'apprenant explore le micromonde librement et sans aucune intervention du tuteur.

L'exploration de micromondes réels, conceptuels ou mixtes

Les laboratoires virtuels d'*ASTUS* (des micromondes) peuvent être conceptuels, réels ou mixtes. Un *laboratoire réel* reproduit la réalité (par exemple : la simulation d'une machine pour découper des pièces de métal), un *laboratoire conceptuel* permet la manipulation de concepts dans un environnement abstrait (par exemple : des outils pour effectuer le classement d'animaux par espèce), alors qu'un laboratoire mixte combine les deux approches. Les laboratoires d'*ASTUS* étendent la notion de micromonde parce qu'ils permettent d'assimiler la dynamique de domaine conceptuel ou de théorie sur le monde réel.

Les principaux modules d'*ASTUS*

La figure 2 illustre une vue détaillée de l'architecture d'*ASTUS*. Dans cette architecture, les modules ont les mêmes fonctionnalités que celles proposées par (Wenger, 1987) ; cette section ne décrira que les différences.

Le module « interface de l'utilisateur » présente une vue sur les objets et outils du micromonde. L'interface graphique peut-être développée au moyen de divers langages de programmation : *Flash*, *Java*, etc. Puisque le cadre d'applications est développé en *Java*, un adaptateur a pour fonction de réifier au système les actions enregistrées par le module

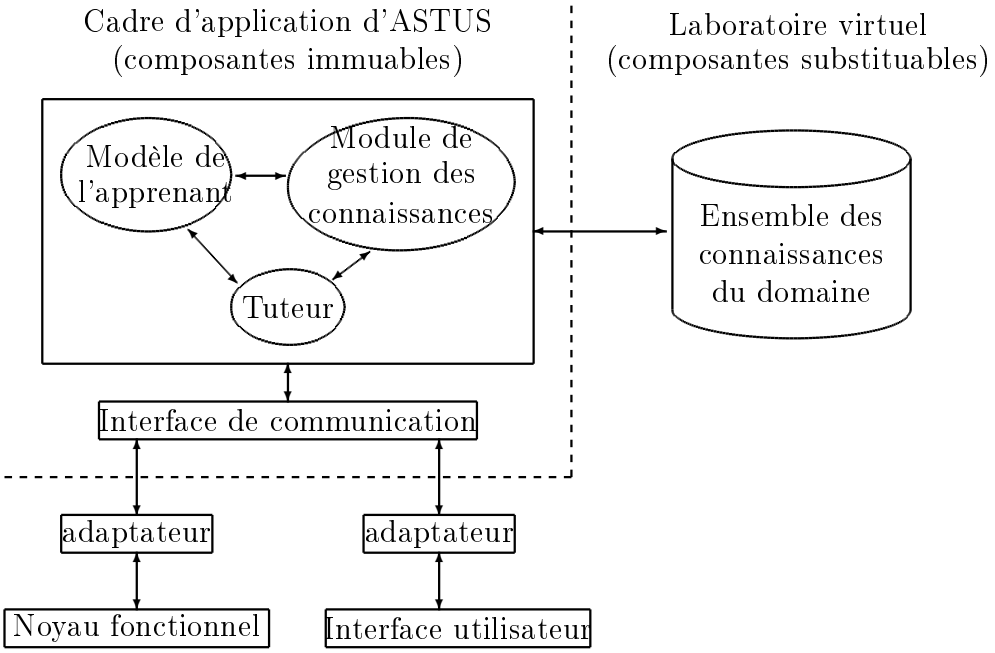

Figure 2 – L'architecture d'*ASTUS*

« interface de l'utilisateur ». Le lecteur intéressé par les adaptateurs est prié de consulter le mémoire de Hamadouche (2004).

Les outils et objets du micromonde visualisés par l'intermédiaire du module « interface de l'utilisateur » constituent une vue des objets et outils situés dans le noyau fonctionnel. Ce dernier contient l'ensemble du code informatique nécessaire pour effectuer les calculs complexes du micromonde. Par exemple, pour le laboratoire virtuel de conversion binaire développé pour le cours d'architecture des ordinateurs, le noyau fonctionnel est un émulateur de microprocesseur *SPARC*. De la même manière que pour le module « interface de l'utilisateur », ce module peut être conçu dans un autre langage de programmation que celui du cadre d'applications. Il est aussi possible que le noyau fonctionnel soit un logiciel boîte noire acheté ou un véritable laboratoire muni de capteurs et de robots. C'est pourquoi un adaptateur agit comme intermédiaire entre le noyau fonctionnel et l'interface de communication.

L'interface de communication a pour fonction de préserver la modularité par la gestion des messages entre les adaptateurs et les différents modules du cadre d'applications.

Le module de gestion des connaissances (*MGC*) contient l'ensemble des outils nécessaires à la manipulation des connaissances sous une forme abstraite propre aux composantes du cadre d'applications. Les connaissances manipulées sont chargées à partir d'un dépôt, nommé *ensemble des connaissances du domaine*.

Le module expert ne se retrouve pas explicitement dans l'architecture d'*ASTUS*. En fait, la fonctionnalité de ce module est répartie entre le *MGC*, l'ensemble des connaissances du domaine et le noyau fonctionnel.

Selon cette perspective architecturale d'*ASTUS*, ce mémoire vise à définir un langage de représentation des connaissances approprié pour l'ensemble des connaissances du domaine, et de définir le *MGC*.

Problématique et objectif

Plusieurs chercheurs offrent des modèles de représentation plus ou moins complexes pour les *STI*. Une infime part répondrait aux exigences du projet *ASTUS* : un modèle polyvalent qui permet de suivre finement les intentions de l'apprenant, un mode d'enseignement *semi-guidé*, etc. Un langage nommé *Usager virtuel* (Mayers, 2001) constitue une base intéressante pour la plupart de ces critères. Toutefois, certains aspects requièrent des améliorations, notamment au niveau de la réutilisation des connaissances et de la clarté de la sémantique. Ce mémoire réunit l'approche d'*Usager virtuel*, la notion d'objet d'apprentissage et les logiques de description, ceci afin d'obtenir un modèle novateur pour les *STI* qui tire profit des qualités de chacune. Ce mémoire prend pour hypothèse que cette fusion engendrera une combinaison gagnante.

Méthode

Les paragraphes qui suivent exposent à travers l'organisation du mémoire la méthode prise pour vérifier l'exactitude de l'hypothèse.

Le chapitre 1 décrit *CREAM* (Nkambou, 1996; Nkambou et al., 2003), un modèle de représentation de cours reconnu, polyvalent et mis en oeuvre dans plusieurs *STI*. La suite du mémoire prend *CREAM* comme point de référence pour donner une appréciation du travail effectué, ce qu'il reste à faire et illustrer certains points forts.

Le chapitre 2 présente *Usager Virtuel* (Mayers, 2001), un modèle pour *STI*, qui permet de modéliser finement les processus cognitifs humains et qui possède plusieurs caractéristiques originales intéressantes pour *ASTUS*. Cependant, ce modèle souffre de plusieurs lacunes : principalement, il n'offre aucun mécanisme pour faciliter la réutilisation et la distribution des connaissances, et il ne définit pas clairement la sémantique de certaines primitives inspirées de la logique de premier ordre.

Les deux chapitres qui suivent présentent chacun une approche susceptible de combler ces lacunes. Tout d'abord, le chapitre 3 aborde les objets d'apprentissage (Duncan, 2003), un concept populaire dans le domaine de la formation en ligne, qui dicte plusieurs principes pour favoriser la réutilisation et le partage du matériel pédagogique. Ensuite, le chapitre 4 survole les logiques de description (Baader et Nutt, 2003), une famille de langages de représentation de connaissances basée sur une approche ontologique et qui exploite, en général, des sous-ensembles de la logique de premier ordre. La principale qualité de ces logiques réside dans leurs algorithmes d'inférence dont la complexité est souvent inférieure aux complexités des démonstrateurs de preuve de la logique des prédicats (Tsarkov et Horrocks, 2003).

Le chapitre 5 propose un assemblage de la notion d'objet d'apprentissage, des logiques de description et des meilleurs aspects d'*Usager virtuel*. Le chapitre compare le modèle obtenu avec *CREAM* et présente les avantages du modèle.

Finalement, le chapitre 6 décrit l'expérimentation du modèle avec un domaine concret et avance des idées de travaux futurs.

CHAPITRE 1

La représentation des connaissances dans *CREAM*

Ce chapitre décrit le modèle de représentation des connaissances *CREAM*[1] (Nkambou, 1996; Nkambou et al., 2003) en vue d'en faire une comparaison au chapitre 5 avec le modèle proposé dans ce mémoire. La comparaison avec *CREAM* permettra de donner une appréciation du travail effectué, de ce qu'il reste à faire, et d'illustrer certains avantages du modèle proposé. Ce mémoire choisit *CREAM* puisque ce dernier est un modèle reconnu, indépendant des domaines et mis en oeuvre dans plusieurs *STI* (Nkambou et al., 2003). La section 1.1 décrit les caractéristiques principales de *CREAM* et la section 1.2 présente les critiques et conclusions.

[1] CREAM: Curriculum REprensentation and Acquisition Model

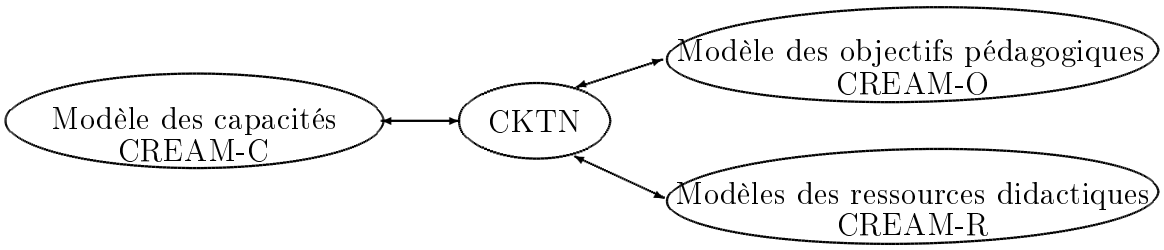

Figure 1.1 – Les composantes d'un curriculum *CREAM*

1.1 Le modèle *CREAM*

Le modèle *CREAM* se concentre sur la représentation des curriculums pour les *STI*. La figure 1.1 montre les composantes d'un curriculum *CREAM* : le modèle des capacités, le modèle des objectifs pédagogiques, le modèle des ressources didactiques et le *CKTN*. Les sous-sections 1.1.1, 1.1.2, 1.1.3 et 1.1.4 se consacrent respectivement à chacune de ces composantes et la section 1.1.5 décrit la notion de cours dans *CREAM*.

1.1.1 Le modèle des capacités

Le modèle des capacités (*CREAM-C*) structure la connaissance à enseigner en terme d'un réseau de *capacités*. Une *capacité* est une connaissance, acquise ou développée, permettant à une personne de réussir dans l'exercice d'une activité physique ou intellectuelle (Nkambou, 1996). La littérature propose plusieurs catégorisations de capacités. *CREAM* adopte la taxinomie de Gagné et al. (1992) qui définit des catégorie fines et qui indique les conditions d'apprentissage pour chaque catégorie. Le modèle des capacités (Nkambou, 1996) conserve trois des classes de capacités de Gagné :

Les informations verbales : Ce sont des connaissances déclaratives qu'un apprenant peut décrire à haute voix (par exemple, l'énoncé *Fernando Pessoa est un écrivain*).

Les habiletés intellectuelles : Ce sont les connaissances qui représentent un savoir-faire (par exemple, *servir du café*, *additionner* et *lancer une balle*). Elles se mani-

festent par des actions et doivent être réifiées sous forme d'informations verbales pour être expliquées.

Les stratégies cognitives : Les stratégies cognitives constituent des stratégies de haut niveau qu'une personne emploie dans divers contextes pour apprendre (stratégies d'apprentissage) ou pour résoudre des problèmes (stratégies de résolution de problèmes).

CREAM représente les capacités par un certain nombre d'attributs, propres à leur catégorie. Il inclut également des sous catégories que ce mémoire ne décrit pas.

Un réseau de capacités *CREAM-C* relie les capacités entre elles par cinq types de relations binaires. Une *relation d'analogie* groupe en paires des capacités qui partagent une analogie au niveau de leur fonctionnalité (analogie fonctionnelle), de leur résultat ou de leur définition (analogie structurelle). Une *relation de généralisation* lie une capacité c_1 à une capacité c_2 plus générale (c'est-à-dire que c_1 hérite de tous les attributs de c_2). Une *relation d'abstraction* lie une capacité c_1 à une capacité c_2 plus abstraite (c'est-à-dire que c_1 hérite de certains attributs de c_2). Une *relation d'agrégation* exprime qu'une capacité est une composante d'une autre capacité. Une *relation de déviation* indique qu'une capacité peut résulter en une autre capacité à la suite d'une transformation (pour décrire une interprétation erronée potentielle).

CREAM permet au concepteur d'un curriculum de choisir parmi plusieurs vocabulaires d'évaluation pour quantifier l'acquisition d'une capacité par un apprenant et spécifier un niveau minimal à atteindre pour juger qu'une capacité est maîtrisée (par exemple, le critère que l'apprenant doit pouvoir appliquer une règle dans plusieurs contextes, pour la maîtriser). Un vocabulaire d'évaluation consiste en un ensemble discret de valeurs, ordonnées de la maîtrise la plus faible à la plus élevée. À titre d'exemple, Nkambou (1996) mentionne l'alphabet suivant basé sur les recherches de Klausmeier (1990) pour évaluer l'acquisition d'un concept : {applique, transfère, reconnaît, classifie et généralise}.

1.1.2 Le modèle des objectifs pédagogiques

La seconde composante d'un curriculum *CREAM*, le modèle des objectifs pédagogiques (*CREAM-O*), structure dans un graphe les objectifs pédagogiques d'un curriculum. Un *objectif pédagogique* représente l'objectif pour le tuteur qu'un apprenant possède certaines connaissances après un apprentissage. Par exemple, un auteur de curriculum pourrait modéliser l'objectif que l'apprenant reconnaisse les principaux panneaux de signalisation routière. Un objectif ne spécifie pas comment enseigner le contenu des capacités qu'il vise. Ce sont les liens entre les objectifs et les capacités et les liens entre les objectifs et les ressources didactiques qui jouent ce rôle (la section 1.1.4 détaillera ces liens). Pour les objectifs pédagogiques, Nkambou (1996) reprend la taxinomie de Bloom qui définit six types d'objectifs pédagogiques : objectifs d'acquisition, objectifs de compréhension, objectifs d'application, objectifs d'analyse, objectifs de synthèse et objectifs d'évaluation. Chaque catégorie d'objectifs possède un ensemble d'attributs pour décrire un objectif.

Un réseau d'objectifs pédagogiques *CREAM-O* relie les objectifs pédagogiques entre eux par trois types de relations : la *relation de préalable* (un objectif est obligatoire ou souhaitable pour un autre objectif), la *relation de prétexte* (un objectif contribue à la maîtrise d'un second objectif) et la *relation de constitution* (un objectif est un sous objectif d'un autre objectif).

1.1.3 Le modèle des ressources didactiques

Le modèle des ressources didactiques (*CREAM-R*) organise les ressources didactiques d'un curriculum sous forme d'un graphe. Une *ressource didactique* est un moyen d'enseignement (une stratégie d'enseignement ou une activité pédagogique) pour évaluer un apprenant ou lui faire acquérir des capacités en vue d'atteindre un objectif pédagogique. Nkambou (1996) catégorise les ressources selon un ensemble de catégories et de sous catégories, dont trois catégories principales : les *ressources tutorielles* (des stratégies

d'enseignement générique pour gérer les interactions avec l'apprenant pendant des activités), les *ressources intelligentes* (des activités pédagogiques qui s'autogèrent totalement et qui possèdent leur propre connaissances du domaine, des connaissances didactiques, un module « modèle de l'apprenant », etc.) et les *ressources non intelligentes* (des activités pédagogiques passives utilisées généralement par les ressources tutorielles et les ressources intelligentes). Six types de relations relient les ressources didactiques entre elles : similarité, abstraction, cas particulier, utilisation, auxiliariat et équivalence. Les lecteurs intéressés pourront consulter la thèse de (Nkambou, 1996) pour les détails sur ces types de relations.

1.1.4 Le modèle pédagogique (*CKTN*)

Trois classes de liens mettent en relation les éléments provenant des différents modules de *CREAM*. Ce sont les liens entre *CREAM-O* et *CREAM-C*, les liens entre *CREAM-O* et *CREAM-R* et les liens entre *CREAM-C* et *CREAM-R*. Les paragraphes suivants décrivent respectivement ces trois classes de liens. Chaque classe distingue plusieurs sous-types de liens.

Les liens entre les objectifs et les capacités : Le modèle *CREAM* inclut deux types de liens entre objectifs et capacités : la *relation de préalable* qui dicte qu'une capacité est préalable à un objectif, et la *relation de contribution* qui affirme qu'un objectif contribue à la maîtrise d'une capacité.

Les liens entre les ressources didactiques et les objectifs : Ces liens lient des objectifs pédagogiques à des ressources didactiques qui permettent de les enseigner ou d'évaluer l'apprenant à l'égard de ces objectifs. La valeur *critique* ou *accessoire* accompagne chaque lien. Elle informe sur l'indispensabilité de la ressource pour l'objectif associé.

Les liens entre les ressources didactiques et les capacités : *CREAM* propose des liens qui mettent en relation des ressources didactiques à des capacités. Ces liens

présentent deux utilités : (1) ils permettent de spécifier des ressources didactiques pour renforcer une capacité (lien de renforcement) et (2) ils permettent d'associer des capacités à des ressources didactiques tutorielles génériques qui peuvent les enseigner (lien de ressource d'enseignement).

Une valeur entre 0 et 1 accompagne chaque lien, pour spécifier la force de la relation. De plus, *CREAM* considère les liens comme des connaissances. Par conséquent, des ressources didactiques peuvent être associées à certains types de liens pour les enseigner, les évaluer ou les corriger. Nkambou (1996) mentionne comme seul exemple l'association d'une ressource didactique à un lien de *déviation* entre deux capacités pour indiquer quels moyens prendre pour le rectifier.

De l'ajout des liens entre les modèles *CREAM-O*, *CREAM-C* et *CREAM-R*, résulte une structure nommée *CKTN* (Curriculum Knowledge Transition Network). Dans un *CKTN*, un objectif pédagogique couplé par des liens à une ou des ressources didactiques est considéré une *transition* qui permet d'acquérir de nouvelles capacités ou de consolider des capacités déjà acquises. La partie *a* de la figure 1.2 (Nkambou, 1996) illustre cette notion de transition. Pour que se réalise l'apprentissage représenté par une transition, un apprenant doit connaître un certain nombre de capacités préalablement (*capacités d'entrée*). Les liens de préalable entre les capacités et l'objectif pédagogique d'une transition, indiquent ses capacités d'entrée. Les ressources didactiques associées à l'objectif pédagogique de la transition par des liens constituent les moyens de réaliser l'apprentissage (des exemples, des activités, etc.). Ils permettent d'acquérir ou de renforcer une ou des capacités (*capacités de sortie*). Les liens de contribution entre l'objectif pédagogique de la transition et les capacités indiquent les capacités de sortie.

La partie *b* de la figure 1.2 illustre l'enchaînement de plusieurs transitions. Un *STI* peut, s'il connaît les capacités maîtrisées par un apprenant, planifier un ensemble de transitions pour permettre à l'apprenant d'atteindre des capacités visées.

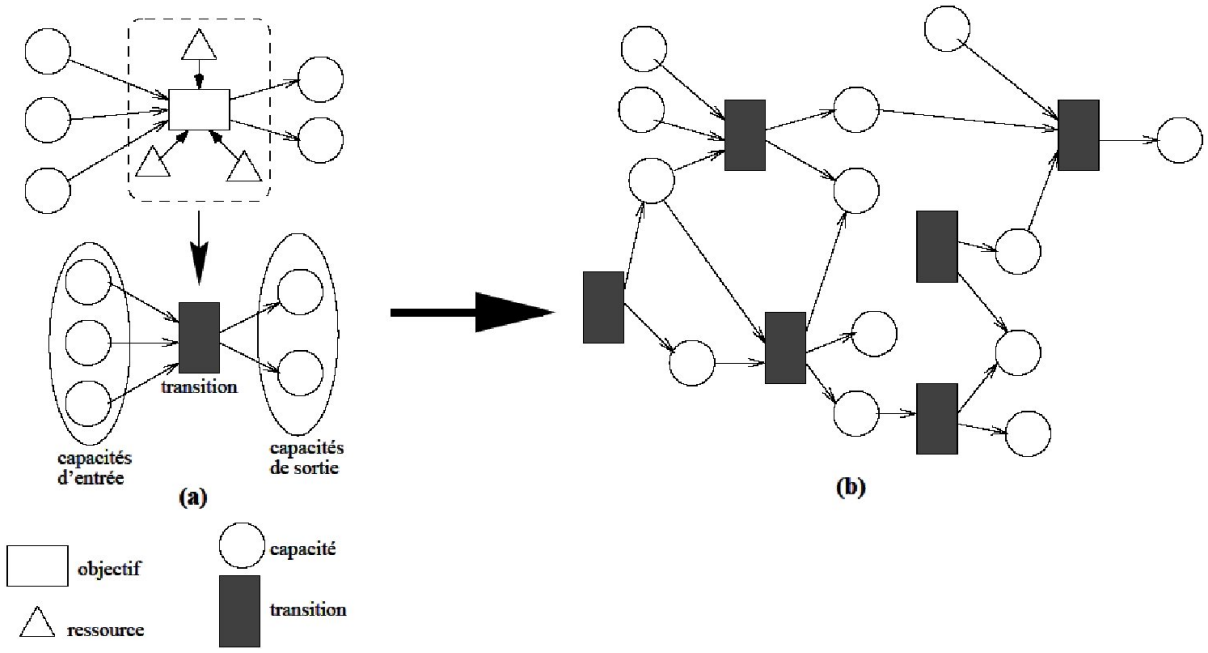

Figure 1.2 – La structure de *CKTN* (Curriculum Knowledge Transition Network)

1.1.5 La notion de *cours*

Cette section porte sur la notion de cours dans *CREAM* : elle formalise tout d'abord le concept de cours, présente les méthodes de génération automatique de cours à partir d'un curriculum et finalement expose le déroulement d'une séance d'enseignement avec les *STI* basés sur *CREAM*.

La définition d'un cours

Un *cours* dans *CREAM* se compose de trois parties : une partie déclarative, un graphe de cours et une partie structurelle. La partie déclarative regroupe les métadonnées générales sur le cours (description de la visée du cours, auteur, etc.) ainsi que les définitions des thèmes du cours. Les *thèmes* sont des regroupements d'objectifs pédagogiques qui abordent une même thématique. Les thèmes peuvent aussi contenir des sous-thèmes, dé-

finis de la même manière. Le *graphe de cours* (qui peut être généré selon un processus décrit plus bas) contient le sous-ensemble des objectifs pédagogiques de *CREAM-O* retenu pour le cours. Le graphe de cours conserve les relations entre ces objectifs, définies dans *CREAM-O*, pour en faire les arcs du graphe de cours. Finalement, la partie structurelle organise les objectifs en trois groupes : les objectifs globaux (les objectifs sans préalable), les objectifs intermédiaires (qui ont au moins un objectif préalabe et qui sont préalables d'au moins un objectif) et les objectifs terminaux (qui ne sont préalables d'aucun autre objectif). La partie structurelle peut être inférée à partir du graphe de cours.

La génération automatique du graphe de cours à partir d'un curriculum

Cette section décrit le processus implanté dans le logiciel *CREAM-TOOLS*[2] pour générer automatiquement un graphe de cours à partir d'un curriculum (Nkambou, 1996; Nkambou et al., 2003). Tout d'abord, un auteur doit décrire le public cible, les exigences de la formation et le curriculum *CREAM*. Le *public cible* se définit par l'association d'un niveau de maîtrise estimé à chacune des capacités dans le curriculum. Les *exigences de la formation* s'expriment soit en terme d'un ensemble de capacités à acquérir ou d'un ensemble d'objectifs pédagogiques à atteindre. Les prochaines phrases décrivent dans les grandes lignes les étapes que *CREAM-TOOLS* effectue pour générer un graphe de cours dans ces deux cas.

La génération d'un graphe de cours à partir d'un ensemble de capacités à maîtriser $\mathcal{D}$ se déroule comme suit :

1. Le logiciel assigne l'état des connaissances du public cible au curriculum, en associant à chaque capacité et à chaque lien de préalable (d'une capacité à un objectif) une valeur parmi *acquis*, *acquis partiellement* et *non acquis*.
2. Le logiciel retire de l'ensemble des capacités à maîtriser $\mathcal{D}$ les capacités jugées acquises par le public cible.

[2] *CREAM-TOOLS* est un système auteur qui supporte la conception visuelle de curriculums *CREAM* et la génération automatique de graphe de cours pour un public cible.

3. Le logiciel calcule un ensemble d'objectifs $\mathcal{E}$ suffisants pour enseigner les capacités dans $\mathcal{D}$ en fonction du public cible. Si plusieurs objectifs permettent d'acquérir une même capacité, le logiciel applique des heuristiques d'ingénierie pédagogique pour retenir un seul objectif.
4. Le logiciel construit le graphe de cours à partir des objectifs dans $\mathcal{E}$. Le graphe de cours constitue l'ensemble des objectifs dans $\mathcal{E}$ avec les mêmes relations que dans *CREAM-O*.

La génération d'un cours à partir d'un ensemble d'objectifs à maîtriser $\mathcal{D}$ se déroule selon les étapes suivantes :

1. Le logiciel assigne l'état des connaissances du public cible au curriculum, en associant à chaque capacité et à chaque lien de préalable (d'une capacité à un objectif) une valeur parmi *acquis*, *acquis partiellement* et *non acquis*.
2. Le logiciel retire de $\mathcal{D}$ les objectifs jugés acquis par le public cible.
3. Le logiciel calcule un ensemble d'objectifs $\mathcal{E}$ suffisants pour enseigner les objectifs dans $\mathcal{D}$ en fonction du public cible. Si plusieurs objectifs permettent d'acquérir une même capacité, le logiciel applique des heuristiques d'ingénierie pédagogique pour choisir un seul objectif.
4. Le logiciel construit le graphe de cours à partir des objectifs dans $\mathcal{E}$. Le graphe de cours constitue l'ensemble des objectifs dans $\mathcal{E}$ avec les mêmes relations que dans *CREAM-O*.

La dynamique des *STI* bâtis sur les cours *CREAM*

La présente section aborde le déroulement des séances d'apprentissage dans les *STI* bâtis sur *CREAM*. Elle se base sur la description de Nkambou (1996) de deux systèmes tutoriels qui enseignent respectivement le code de la route, et la manipulation d'une pompe Baxter en domaine médical. Nkambou (1996) propose deux approches pour le déroulement des séances d'apprentissage.

La première approche attribue la tâche de la planification des activités pédagogiques au *STI*. Tout d'abord, un module *planificateur* de *CREAM* a pour tâche de déterminer quels objectifs pédagogiques du cours, enseigner en premier, en fonction des capacités à acquérir. Par la suite, le module tuteur sélectionne une ressource didactique qui couvre les premiers objectifs et qui est appropriée pour l'apprenant, d'après le module « modèle de l'apprenant ». Ensuite, le module tuteur lance l'exécution de la ressource didactique. Lorsque l'activité se termine, le système tutoriel met à jour le module « modèle de l'apprenant » avec le résultat de la ressource. Après, le cycle recommence : le planificateur établit un nouvel ordre de priorité des objectifs, le tuteur sélectionne une ressource didactique, etc.

Alternativement, (Nkambou, 1996) propose un second mode d'enseignement où le système offre à l'apprenant de choisir lui-même parmi les objectifs pédagogiques du cours et ensuite parmi les ressources didactiques associées. Cette méthode accorde à l'apprenant le contrôle de son cheminement. Le système continue à prêter assistance à l'apprenant dans les activités pédagogiques.

Le cours, une source riche d'information pour l'enseignement

Par son organisation, la structure des cours dans *CREAM* permet de répondre à une série de questions relatives à l'enseignement et utiles pour un *STI*. Voici ces questions, reprises de Nkambou (1996) :

- *L'enseignement effectué a-t-il permis d'atteindre les objectifs initiaux ?*
- *L'étudiant a-t-il acquis les connaissances ciblées par l'enseignement ?*
- *Que faut-il savoir si l'on veut apprendre tel ou tel aspect de la matière ?*
- *Quelle est la structure logique d'un thème que l'on veut aborder ?*
- *Quels sont les liens logiques entre une connaissance que l'on veut faire acquérir et les autres connaissances déjà acquises ou non, impliqués dans la matière ?*
- *Quel est l'objectif actuellement poursuivi ?* (perspective de l'étudiant)

– *Quel faut-il savoir ?* (perspective de l'étudiant)
– *De quels moyens dispose-t-on ?* (perspective de l'étudiant)

De plus, Nkambou (1996) souligne que *CREAM* est un *support de base à certains types de conseils tels que des conseils pertinents basés sur les liens didactiques ou logiques existant entre les différents éléments de la matière* (par exemple, un lien entre deux ressources didactiques permet de souligner à l'apprenant qu'un problème semblable a déjà été résolu).

1.2 Conclusion

Nkambou (1996) a défini un modèle de curriculum explicite pour les *STI* qui réunit les aspects didactiques, pédagogiques et les connaissances du domaine, une idée originale au moment de sa publication. Cette structure de curriculum comporte au moins trois intérêts : (1) elle permet la génération automatique de cours, (2) elle propose un fondement pour des *STI* qui enseignent plusieurs domaines et (3) sa structure constitue une source riche d'information pour l'enseignement. Néanmoins, le modèle *CREAM* est imparfait à certains égards. Le prochain paragraphe en critique un aspect.

Le module « modèle de l'apprenant » dans *CREAM* est un modèle par recouvrement, c'est-à-dire un module qui représente le savoir de l'apprenant comme un sous-ensemble de celui de l'expert. Le recouvrement se fait sur *CREAM-C*, *CREAM-O* et *CREAM-R*. Il indique quelles capacités, objectifs et ressources didactiques qu'un apprenant maîtrise, a atteints ou a vus. Une limite inhérente aux modèles par recouvrement est qu'ils ne peuvent pas représenter les connaissances fausses ; ils indiquent soit qu'un apprenant possède une connaissance ou soit qu'il ne la possède pas. Une connaissance fausse est soit une connaissance erronée, soit une connaissance valide qui est employée dans un contexte autre que celui auquel elle est destinée. Par exemple, l'habileté intellectuelle d'effectuer une soustraction multicolonne sans tenir compte des emprunts est une connaissance er-

ronée, alors que l'habileté intellectuelle d'effectuer une addition multicolonne est une connaissance valide, mais fausse lorsqu'employée dans une situation qui demande l'emploi de la soustraction. La notion de relation de déviation (cf. section 1.1.1) pallie partiellement le problème de représentation des connaissances fausses : cette relation qui lie des paires de capacités sujettes à des confusions, permet de modéliser l'erreur de confondre une capacité avec une autre capacité valide, mais incorrecte dans un contexte. Néanmoins, cette relation ne permet pas de lier une capacité à une capacité erronée, car ce concept n'existe pas dans *CREAM*. S'il est vrai que modéliser les capacités erronées n'est pas essentiel, il n'en demeure pas moins que leur prise en compte pourrait alimenter des stratégies de remédiation plus adaptées à l'apprenant.

La représentation des connaissances erronées constitue un objectif pour le projet *ASTUS*. *CREAM* ne répond clairement pas à cet objectif. Le prochain chapitre présente *usager-virtuel*, un second modèle de représentation des connaissances pour les *STI* qui inclut la représentation des connaissances erronées. Ce modèle servira de base pour la présentation d'un modèle hybride au chapitre 5. Le chapitre 5 dresse également un parallèle avec *CREAM* et le modèle proposé pour illustrer certains points forts et introduit des perspectives de travaux futurs.

CHAPITRE 2

L'approche *Usager Virtuel*

Usager Virtuel (*UV*) est un modèle de représentation des connaissances pour les *STI*, élaboré par Mayers (2001). Ce modèle repose sur le principe fondamental suivant : pour offrir un enseignement optimal, un *STI* doit être capable d'identifier les intentions, préférences, acquis et fausses croyances des apprenants (une simulation fine de l'apprenant telle que discutée au chapitre précédent). En ce sens, *UV* adopte une perspective différente de celle de *CREAM* : il s'intéresse davantage à modéliser finement l'état des acquis de l'apprenant qu'à formaliser une notion de haut niveau de curriculum. Afin d'atteindre cet objectif, *UV* représente les connaissances d'un domaine en terme des sous-systèmes de la mémoire humaine et de ses processus tels que décrits en psychologie cognitive. Puisque plusieurs recherches sur la cognition humaine s'avèrent inconciliables, des choix ont été faits. La théorie d'*UV* est fondée principalement sur *MIACE* (Mayers, 1997; Mayers et al., 2001)., auquel s'ajoutent des éléments de la logique des prédicats et d'autres propres aux *STI*. Ce chapitre présente tout d'abord ce modèle de représentation de connaissances, ensuite il décrit une mise en oeuvre réalisée dans le cadre de ce mémoire, et finalement, il dresse un bilan des points forts et des points faibles d'*UV* en guise de conclusion.

2.1 La mémoire long terme

La mémoire à long terme humaine comporte plusieurs sous-systèmes. Leur nombre et leur nature diffèrent selon les auteurs. *UV*, comme *MIACE* (Mayers, 1997) se base sur trois sous-systèmes : la mémoire sémantique, la mémoire procédurale et la mémoire épisodique. Les sections 2.1.1, 2.1.2 et 2.1.3 les décrivent respectivement.

2.1.1 Les connaissances sémantiques

Les connaissances sémantiques sont des connaissances descriptives sur le monde. L'unité de base de la mémoire sémantique est le *chunk*, une unité de stimuli perceptif ou cognitif ayant une signification familière (Fortin et Rousseau, 1992). La notion de chunk remonte à Miller (1956) qui associait une capacité de 7 ± 2 chunks à la mémoire à court terme. Quoique cette représentation de la mémoire court terme soit désuète, le concept de chunk ne l'est pas, en tant qu'élément d'information de taille variable. Deux catégories de chunks sont distinguées dans *UV* : les concepts et les buts.

Les concepts

Le terme *concept* réfère à la notion de concept au sens le plus large : le concept de voiture, d'être humain, le symbole 5, etc. Selon des recherches récentes (Halford et al., 2005; Cowan, 2001; Fisher, 1984), l'être humain ne peut considérer qu'environ quatre concepts à la fois (quatre dimensions) dans l'accomplissement d'une tâche. Cependant, l'architecture cognitive humaine possède la caractéristique de grouper plusieurs concepts pour les manipuler comme un seul sous forme d'un vecteur de concepts (Halford et al., 1993). Ces concepts syntaxiquement décomposables se nomment *concepts décrits*, par opposition aux *concepts primitifs*, syntaxiquement indécomposables (Mayers, 1997). Par exemple, le symbole 8 est un concept primitif alors que (+ 5 3) est un concept décrit avec la même sémantique. L'encodage sous forme de concepts décrits s'effectue par le processus

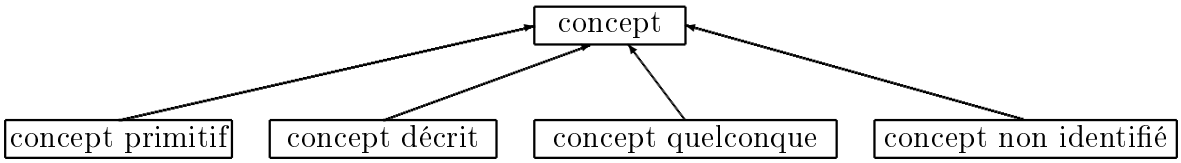

Figure 2.1 – Premier axe de catégorisation des concepts dans *UV*

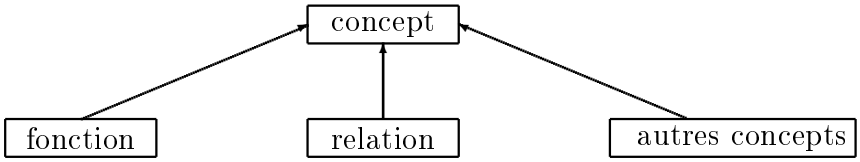

Figure 2.2 – Second axe de catégorisation des concepts dans *UV*

de *chunking*. Lorsqu'un humain manipule un concept décrit, il manipule un tout dont la sémantique est issue de ses composantes. Il ne peut plus accéder aux composantes individuelles. Un second processus, nommé *segmentation*, doit être utilisé pour extraire les parties d'un concept décrit (Halford et al., 1993).

Les buts

Les buts sont des intentions actuelles ou passées qu'un humain peut avoir, telles que le but de résoudre une équation mathématique, de dessiner un triangle ou d'additionner deux nombres. Un but s'accomplit au moyen d'une connaissance procédurale.

La classification des concepts selon *UV*

UV catégorise les concepts selon deux axes (figure 2.1 et figure 2.2 respectivement). Les notions, à la figure 2.1 et 2.2, de concepts quelconques, concepts non identifiés, fonction et relation sont des ajouts inspirés de la logique des prédicats. Bien qu'il soit faux de dire que les fonctions et les relations n'existent pas [1], il n'y a pas eu de recherches

[1] Par exemple, un humain peut dire « le père de Jacques » où pèreDe est une fonction s'appliquant à un individu Jacques.

en psychologie expérimentale qui visent à distinguer sur un plan psychologique (temps de latence, activation, etc.) les fonctions et les relations des autres concepts. Il n'y a donc pas nécessité d'en faire une distinction dans UV. Leur intrégration dans UV vise à faciliter la modélisation de domaines formellement descriptibles, et le raisonnement sur ces domaines.

Un concept décrit [2] dans UV se présente comme une fonction appliquée à des concepts. Dans l'exemple (+ 5 3), il s'agit de la fonction + et des concepts primitifs 5 et 3. Cette représentation des concepts décrits permet de faire une distinction importante entre la conceptualisation et l'évaluation d'une fonction par un apprenant. En effet, un apprenant peut très bien penser au concept décrit (+ 5 3) sans nécessairement vouloir ou savoir comment l'évaluer. L'utilisation des concepts décrits permet de créer des représentations syntaxiquement décomposables pour représenter des concepts primitifs (ici, le concept 8, qui peut être obtenu après l'évaluation correcte de la fonction + aux arguments 5 et 3)[3].

Un concept quelconque est un concept qui n'indique que la classe à laquelle il appartient. Ceci correspond à l'utilisation du quantificateur universel en logique. Des exemples de concepts quelconques sont : le concept de nombre entier quelconque, le concept d'expression booléenne quelconque et le concept d'humain quelconque.

Un concept non identifié est un concept avec une sémantique précise et inconnue. Un exemple est le concept non identifié x dans l'équation mathématique $x^2 + x = 2$. Dans cet exemple, le nombre 1 constitue la sémantique inconnue de x. Un concept non identifié correspond à l'utilisation du quantificateur existentiel en logique et plus spécifiquement aux constantes de skolem.

Les concepts quelconques et les concepts non identifiés remplacent le besoin d'introduire explicitement des quantificateurs dans le formalisme. Par exemple, quelqu'un peut avec ce modèle définir le concept père $PèreDe$ comme fonction, considérer x comme un individu

[2] La notion de concept décrit se rapproche du terme *fonctionnelle* en logique.

[3] En mathématiques, plusieurs concepts n'ont pas une représentation atomique, par exemple $\sqrt{2}$.

quelconque et mentionner son père qui est ($PèreDe\ x$).

2.1.2 Les connaissances procédurales

La mémoire procédurale est le lieu où sont stockées les connaissances dynamiques du domaine, c'est-à-dire les façons de manipuler les connaissances sémantiques pour accomplir les buts. Ces connaissances dynamiques se nomment *procédures*. Contrairement aux connaissances sémantiques où leur activation donne lieu à une prise de conscience de leur contenu, l'activation des connaissances procédurales ne nécessite pas l'attention du sujet. Par exemple, lorsqu'un humain maîtrisant une méthode de dactylographie tappe au clavier, il n'a pas besoin de réfléchir à l'emplacement des touches et à chacun des mouvements de ses doigts : cela se fait machinalement. Néanmoins, il se peut que la même personne, si quelqu'un lui demande l'emplacement d'une touche, ne puisse pas répondre. Ceci est, parce qu'elle n'a plus accès à la connaissance sémantique correspondante. Cet exemple d'Anderson (1993) illustre bien la distinction entre les connaissances sémantiques et les connaissances procédurales. Ces deux mémoires ne sont pas nécessairement mutuellement exclusives. Il est en effet possible qu'un étudiant ait à la fois une connaissance procédurale et sémantique de l'emplacement d'une touche. Ceci est une conséquence du fait que le cerveau humain n'organise pas les connaissances selon leur contenu mais plutôt selon la façon dont elles sont manipulées. Si un étudiant n'utilise plus une connaissance sémantique après l'avoir automatisée sous forme procédurale, celle-ci perdra graduellement de sa force et sera éventuellement oubliée. C'est ce qui arrive notamment dans l'exemple du dactylographe.

Mayers (2001) distingue deux types de procédures : les *procédures complexes* et les *procédures primitives*. Ces deux types de procédures diffèrent par leur degré d'automatisation. Les procédures primitives sont des procédures totalement automatisées. Elles ne nécessitent pas les ressources de l'attention pour être utilisées. Les procédures complexes sont des procédures qui ne sont pas totalement automatisées. Ces dernières sont spécifiées par

un script de sous-buts à accomplir. Ces sous-buts peuvent être réalisés à partir d'autres procédures complexes ou primitives, jusqu'à ce que tous les sous-buts soient accomplis par des procédures primitives. Une procédure complexe est une procédure automatisée, mais seulement au niveau de l'ensemble des sous-buts à accomplir. En effet, chaque sous-but peut être réalisé par possiblement plusieurs procédures différentes, mais l'ensemble des sous-buts immédiats est fixe.

Une justification de la décomposition hiérarchique des procédures

Le concept de procédures complexes présent dans *UV* se justifie partiellement en traçant un parallèle avec la théorie du contrôle de l'attention de Norman et Shallice (Cooper et Shallice, 2000). Selon cette théorie, deux sous-systèmes cognitifs humains supervisent la réalisation des tâches : le *gestionnaire des priorités de développement* (*GPD*) [4] et le *système attentionnel superviseur* (*SAS*). Le *GPD* contrôle l'exécution des tâches totalement automatisées, alors que le SAS gère l'exécution des tâches non automatisées.

Norman et Shallice nomment *schéma* le savoir-faire encodé pour réaliser une tâche. Les *schémas* spécifient des sous-buts, réalisables au moyen d'autres *schémas*, jusqu'aux *schémas* de dernier niveau opérés directement par le système moteur. Le processus d'exécution des *schémas* se déroule dans le *GPD*. Le *GPD* active le *schéma* qui dispose d'une activation suffisante, mais aussi supérieure à celle de tous les *schémas* en compétition pour le but actuel. Cinq facteurs déterminent la force d'activation d'un *schéma* : (1) les *schémas* qui engendrent le but courant, (2) les stimulis de l'environnement, (3) l'activation de base du *schéma*, (4) les *schémas* concurrents pour le but actuel et (5) un facteur de bruit. Dans le cas de tâches non routinières, le *SAS* entre en jeu : il active dans le *GPD* des *schémas* qui demeureraient normalement inhibés.

Des expérimentations montrent que le *GPD* de Norman et Shallice reproduit fidèlement plusieurs comportements d'humains normaux et de patients atteints de certaines lésions

[4] en anglais, *CS* pour Contention Scheduling.

neurologiques (Cooper et Shallice, 2000). La spécification du *SAS* reste néanmoins floue et peu validée (Glasspool, 2000). Un parallèle peut être tracé entre les procédures d'*UV* et les *schémas*. Les procédures primitives correspondent aux schémas de bas niveau, et les procédures complexes aux schémas de haut niveau qui spécifient les buts. Quoique certains chercheurs remettent en question la décomposition hiérarchique des tâches (Botvinick et Plaut, 2004), la théorie de Norman et Shallice procure sans contredit une certaine crédibilité à la distinction d'*UV* entre procédures complexes et procédures primitives.

2.1.3 La mémoire épisodique

La mémoire épisodique (Tulving, 1983) conserve une trace de tous les épisodes vécus. C'est grâce à cette mémoire, par exemple, qu'un individu se rappelle de son déjeuner le matin. Elle se compose d'épisodes. Chaque épisode correspond à une tentative d'accomplir un but avec une procédure. Lorsqu'un but est accompli par une procédure complexe (donc par la réalisation de plusieurs sous-buts), un sous-épisode est créé pour chacun des sous-buts mentionnés dans la procédure complexe. Les instances de concepts qu'un étudiant manipule dans un épisode se nomment *cognitions*.

La figure 2.3 montre la structure de la mémoire épisodique engendrée par le calcul de $(5-2)^2!$ par un apprenant. La notation graphique est une adaptation de celle proposée par Najjar et Mayers (2003). Les rectangles dénotent des buts, les cercles représentent des procédures et les losanges, des épisodes. La figure place entre parenthèses les cognitions manipulées par un but ou une procédure, et place les cognitions qui résultent de l'exécution d'une procédure à la droite d'une flèche. Le but principal B est accompli par la procédure complexe P. L'action de cette procédure donne lieu à un épisode E. Cette procédure complexe spécifie deux sous-buts $SB1$ et $SB2$ qui correspondent respectivement à l'objectif d'effectuer $(5-2)^2$ et de calculer 3!. La réalisation de $SB1$ est faite par une procédure complexe $P1$ qui engendre l'épisode $E1$, sous-épisode de E, et qui spécifie les sous-buts $SB3$ et $SB4$ qui consistent respectivement à calculer $5-2$ et 3^2. L'accom-

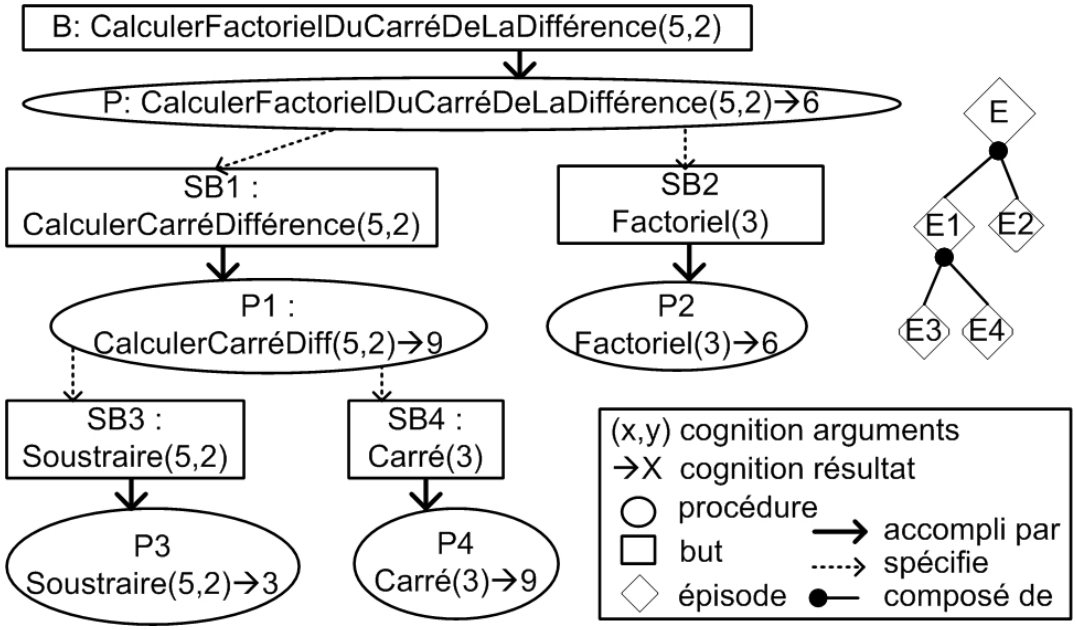

Figure 2.3 – Exemple d'exécution du calcul mathématique $(5-2)^2$!

plissement de ces buts, par les procédures $P3$ et $P4$, s'inscrit dans les épisodes $E3$ et $E4$, sous-épisodes de $E1$. Le but $SB2$ est atteint par la procédure primitive $P2$, et enregistré dans l'épisode $E2$, sous-épisode de E. Les sous-buts de dernier niveau ($SB3$, $SB4$, $SB2$) sont toujours atteints par des procédures primitives. L'exemple présente un enchaînement de procédures parmi plusieurs enchaînements envisageables, car plusieurs procédures correctes ou erronées peuvent accomplir un même but. Par exemple, un apprenant pourrait réaliser le but B avec une procédure complexe qui spécifie comme premier sous-but l'évaluation de l'exposant 2, c'est-à-dire le but d'effectuer $(5-2)^2 = (25+4-20)$).

2.2 Une représentation informatique de la mémoire long terme

Cette section présente la représentation informatique de la mémoire long terme selon *UV*. Chaque entité de connaissance (concepts, buts, procédures, cognitions et épisodes) est spécifiée par un schéma comportant un certain nombre de facettes. *UV* utilise la mémoire sémantique et procédurale pour représenter les connaissances du domaine, et conserve la mémoire épisodique pour les connaissances spécifiques à chaque apprenant. La mémoire épisodique est construite dynamiquement, soit en fonction des actions perçues de l'apprenant, soit en fonction des actions cachées qui expliquent les actions perçues. Puisque la mémoire sémantique et procédurale représente les connaissances du domaine, *UV* ajoute plusieurs facettes aux structures de la mémoire sémantique et procédurale de *MIACE* pour représenter des aspects didactiques et ontologiques, tandis que la représentation de la mémoire épisodique reste sensiblement la même.

UV est une architecture cognitive, car les connaissances procédurales peuvent être interprétées et donner lieu à un comportement cognitif où des cognitions sont générées dans une mémoire de travail et encodées dans des épisodes. *UV* mise sur l'utilisation de probabilités dans le module « modèle de l'apprenant » pour représenter le niveau d'acquisition des procédures et des concepts pour chaque apprenant. Le module « modèle de l'apprenant » traitera davantage des contraintes sur les capacités cognitives. Les prochaines sous-sections décrivent les facettes des schémas pour représenter les diverses entités de connaissances.

2.2.1 Les concepts

Le schéma pour encoder les concepts possède les facettes mentionnées dans le tableau 2.1, décrites ci-dessous.

Facettes de métadonnées	Facettes spécifiques aux concepts	Facettes didactiques	Facettes pour l'héritage
Identificateur	Type	Enseignement	Superconcepts
Nom	Buts		Sous-concepts
Description	Constructeurs		
Concepteur	Composantes		

Tableau 2.1 – Les facettes d'un concept selon *UV*

Les métadonnées

Les quatre premières facettes contiennent des métadonnées générales. *Identificateur* est une chaîne de caractères qui identifie le concept de façon unique. *Nom* contient le nom du concept tel qu'il est présenté à l'apprenant par le système. *Description* contient une description textuelle du concept. *Concepteur* identifie l'auteur du concept. Ces facettes sont utiles aux pédagogues et aux concepteurs de cours et faciliteront l'encapsulation de ces unités de connaissances en objets d'apprentissage. Ces derniers sont présentés au chapitre 3, et une méthodologie pour créer des objets d'apprentissage avec les connaissances est présentée au chapitre 5.

Les facettes spécifiques aux concepts

Les quatre facettes suivantes caractérisent les concepts. *Type* indique le type de concept : primitif, décrit, quelconque ou non identifié (cf. section 2.1.1). Cette facette occupe un double rôle : soit qu'elle spécifie le type de concept sur le plan psychologique (décrit, primitif) ou qu'elle octroie une sémantique logique au concept (quelconque, non identifié). Cette sémantique logique devait servir pour des mécanismes d'inférence (la section 2.4.2 revient sur ce point). La facette *Buts* contient la liste des buts dont la réalisation nécessite une occurrence de ce concept. Autrement dit, la facette indique les intentions qu'un apprenant peut manifester avec ce concept. Cette facette tirée de *MIACE* est d'une importance particulière, car elle permet de simuler le comportement d'un apprenant

qui cherche une solution par tâtonnement. *Constructeur* spécifie les identificateurs des principales procédures qui permettent d'obtenir une instance de ce concept. Dans une situation précise, le système pourrait choisir l'une de ces procédures en fonction de leur description (les facettes *Arguments*, *But*, *Contexte*, etc.) et de leur maîtrise estimée selon le module « modèle de l'apprenant ». La facette *Composantes* est spécifique aux concepts décrits. Elle indique pour chaque composante du concept décrit, le type de concept requis.

La facette *Enseignement*

La facette *Enseignement* suggère des outils didactiques pour enseigner le concept. Ce sont des pointeurs vers des exemples, que les stratégies didactiques génériques du module tuteur doivent utiliser pour aider l'apprenant.

La notion d'héritage entre les concepts

Les deux dernières facettes (*Superconcepts* et *Sous-concepts*) permettent de relier les concepts sous la relation d'héritage. *Superconcepts* contient la liste des concepts desquels ce concept hérite et *Sous-concepts* contient la liste des concepts qui héritent de ce concept. L'héritage implique que les buts, concepts décrits, fonctions, relations et procédures qui s'appliquent à un concept, s'appliquent également à tous les concepts qui héritent de celui-ci.

À titre d'exemple, pour une modélisation du domaine des mathématiques, le but général *ButIsolerVariableX* prend en paramètre un concept *ÉquationMathématique*. Ce but peut être utilisé avec toutes les occurrences des concepts fils d'*ÉquationMathématique* que pourraient être les concepts *ÉquationMathématiqueDuPremierDegré* et *ÉquationMathématiqueDuSecondDegré*.

La notion d'héritage d'*UV* constitue une notation simplifiée, un raccourci à la disposition des auteurs, plutôt qu'un mécanisme reflétant la structure réelle de la mémoire sémantique. En effet, la majorité des chercheurs appuie l'organisation de cette mémoire

Paramètres
Image
Association
But-fonction
Préconditions
Postconditions

Tableau 2.2 – Les facettes propres à une fonction ou relation selon *UV*

proposée dans le modèle de distribution d'activation de Collins et Loftus (1975). Cette théorie énonce que les liens d'héritage sont acquis comme toutes autres connaissances sémantiques et donc régulés par les mêmes règles : encodés sous forme de concepts, changeants, sujets à l'oubli, etc. Par conséquent, si un concepteur de connaissances souhaite représenter la manière humaine de gérer les relations d'appartenance, il ajoutera des buts et des procédures pour manipuler des concepts décrits d'une forme semblable à (*chat SousConceptDe animal*) qui encodent les relations d'appartenance. Plus précisément, au niveau de la modélisation des connaissances du domaine, le lien d'héritage entre deux concepts est soit présent ou absent tandis que dans le module « modèle de l'apprenant », ce lien est de nature probabiliste.

2.2.2 Les fonctions et les relations

UV définit les fonctions comme des concepts possédant des facettes supplémentaires (cf. tableau 2.2). La facette *Paramètres* spécifie les types de concepts pris en arguments par la fonction. *Image* indique le type de concept retourné par l'évaluation de la fonction. *Association* contient le prototype du concept décrit construit par cette fonction lorsque l'étudiant la conceptualise. *But-fonction* spécifie le prototype de but qui permet d'évaluer la fonction (pour obtenir le résultat). *Postconditions* et *Préconditions* dénotent respectivement un ensemble de contraintes sur les paramètres admissibles par la fonction et sur le résultat. Les contraintes se formulent à l'aide de relations.

Habileté
Paramètres
Procédures

Tableau 2.3 – Les facettes propres aux buts selon *UV*

Facettes de métadonnées	Facettes spécifiques aux procédures	Facettes didactiques
Identificateur	But	Diagnostique-solution
Nom	Arguments	Ressources didactiques
Description	Script	Exemples
Concepteur	Validité	Exercices
	Contexte	Test
	Utilisation	
	Méthode	

Tableau 2.4 – Les facettes des procédures selon *UV*

Les relations détiennent les mêmes facettes que les fonctions hormis *Préconditions* et *Postconditions*. De plus, leur image doit toujours être du type booléen.

2.2.3 Les buts

En plus des facettes des concepts, les buts détiennent trois facettes spécifiques (cf. tableau 2.3). *Habileté* contient une chaîne de caractères spécifiant l'habileté humaine requise pour accomplir le but. *Paramètres* indique le type des paramètres du but. *Procédures* spécifie les procédures utilisables pour tenter de réaliser le but.

2.2.4 Les procédures

Seize facettes décrivent les procédures (cf. tableau 2.4). Les quatre premières facettes occupent le même rôle que pour les concepts, et par conséquent, cette section ne les

Script : (ButAdditionnerDeuxEntiers x (ButAdditionnerDeuxEntiers y z))

Figure 2.4 – Le script d'une procédure complexe pour additionner trois entiers

décrit pas.

La facette *But* identifie le but associé à cette procédure. La facette *Arguments* spécifie le type de concepts pris en paramètres par la procédure. Cette facette ainsi que la facette *But* équivalent à la partie *condition* de règles de production, car la procédure ne peut s'activer qu'en présence d'un but et d'arguments compatibles avec cette description. Lors d'une simulation d'un comportement cognitif, le mécanisme de choix d'une procédure pour un but pourrait tenir compte de plusieurs autres facteurs. Par exemple, la théorie de Norman et Shallice (Cooper et Shallice, 2000) (cf. section 2.1.2) suggère de calculer l'activation d'un schéma (l'équivalent d'une procédure dans la théorie d'*UV*) en fonction (1) des schémas qui engendrent le but courant, (2) de l'activation de base du schéma, (4) des schémas concurrents pour le but actuel et (5) d'un facteur de bruit. Comme l'introduction de la section 2.2 le mentionne, l'élaboration de l'aspect dynamique de la mémoire de travail fera partie de futures recherches. Le module « modèle de l'apprenant » représentera les contraintes sur les capacités cognitives et le niveau d'acquisition de chaque unité de connaissance.

La facette *Script* ou *Méthode* décrit l'action qui résulte de l'application de la procédure. Pour les procédures complexes, *Script* énonce le script des sous-buts à accomplir, alors que pour les procédures primitives, la facette *Méthode* pointe vers une méthode *Java* qui accomplit la procédure (elle spécifie un nom de classe *Java*, une de ses méthodes et les arguments à lui passer). La figure 2.4 illustre un exemple de script pour une procédure complexe effectuant l'addition de trois entiers et qui spécifie deux sous-buts.

La facette *Validité* contient une paire de valeurs booléennes : la première indique si la procédure trouve à tout coup le bon résultat (pour le but associé) et la seconde, si la procédure termine toujours. Autant les procédures valides que non valides sont encodées, car

Facettes résultat	Facettes hiérarchie	Facettes identificateur	Facettes but
Résultat	Superépisode	Identificateur	But-épisode
Coût	Sous-épisodes	Temps	Procédure-épisode
État			

Tableau 2.5 – Les facettes des épisodes selon *UV*

l'apprenant peut les utiliser. *Contexte* définit des contraintes sur le contexte d'utilisation de la procédure. La facette *Utilisation* associe une valeur parmi (utile, inutile, probable) à chaque contexte afin d'indiquer la pertinence de la procédure à l'égard de ce dernier.

Diagnostique-solution contient une liste de paires de type « diagnostic, stratégie » indiquant pour une série de diagnostics, la stratégie pédagogique à adopter. Les dernières facettes proposent des pointeurs vers des ressources supplémentaires pour enseigner la procédure : des exemples, des tests, des exercices, etc.

2.2.5 Les épisodes

Le tableau 2.5 illustre les facettes des épisodes. La facette *Identificateur* contient un numéro unique attribué par le système. *Temps* indique le moment où l'épisode s'est déroulé. Cette facette permet de localiser un épisode de façon unique par son occurrence dans le temps. *But-épisode* spécifie le prototype du but de l'épisode suivi des identificateurs des cognitions prises en paramètres. *Procédure-épisode* contient l'identificateur de la procédure utilisée pour accomplir le but. La facette *Résultat* renferme l'identificateur de la cognition obtenue en résultat. La facette *coût* comporte un coût d'utilisation estimé de la procédure. À chaque but tenté par l'apprenant, correspond un épisode. Si l'apprenant utilise une procédure complexe, il doit accomplir une séquence de sous-buts auxquels correspondent des sous-épisodes. Les facettes *Superépisodes* et *Sous-épisodes* contiennent les identificateurs des sous-épisodes et des superépisodes, respectivement. Finalement, la facette *État* prend une des valeurs suivantes, selon l'état actuel de l'épisode :

Identificateur
Épisode
Id-concept
Composantes
Supercomposantes
Lieu
Statut
Précédent

Tableau 2.6 – Les facettes des cognitions selon *UV*

- **en attente** : l'état par défaut d'un épisode avant que le système lui assigne une des trois autres valeurs.
- **succès** : indique la réalisation avec succès de tous les sous-épisodes de cet épisode.
- **échec** : précise soit que (1) l'atteinte du but de cet épisode a échoué faute de procédures applicables aux cognitions prises en arguments ou bien (2) qu'un sous-épisode possède la valeur *échec* ou *abandonné*.
- **abandonné** : mentionne que l'apprenant a délibérément abandonné la réalisation du but de cet épisode.

2.2.6 Les cognitions

Les concepts sont des connaissances de la mémoire à long terme. Lorsqu'elles sont activées, elles laissent une trace dans la mémoire épisodique sous le nom de cognition[5]. Le tableau 2.6 présente les facettes des cognitions. *Identificateur* possède la même signification qu'antérieurement. La facette *Épisode* contient l'identificateur de l'épisode dans lequel la cognition fut employée. *Id-concept* indique de quel concept la cognition est une instance. Si la cognition est une cognition décrite, *Composantes* spécifie les identificateurs de ses parties (des cognitions). De façon similaire, si la cognition est composante d'une autre cognition, l'identificateur de cette dernière apparaîtra dans la facette *Supercom-*

[5] *UV* définit la mémoire épisodique comme un recouvrement sur la mémoire procédurale et sémantique.

posante. *Lieu* et *Statut* indiquent la position de la cognition sur l'écran et sa visibilité à l'apprenant, dans le cas où la cognition est associée à la perception (lecture ou écriture) d'un concept dans le laboratoire virtuel. Finalement, la facette *Précédent* contient un pointeur vers l'origine de la cognition (un identificateur de cognition). Si cette facette est vide, cela signifie que la représentation symbolique de la cognition n'est pas une copie de celle d'une autre cognition ; par exemple, si l'apprenant obtient la cognition par l'évaluation d'une fonction.

2.2.7 Les connaissances pédagogiques et didactiques

Dans *UV*, une distinction existe entre connaissances pédagogiques et didactiques [6]. Les premières sont communes à tous les laboratoires et encodées statiquement dans le module tuteur. Elles constituent des façons générales d'enseigner. Les secondes (connaissances didactiques) sont spécifiques à chaque domaine et sont encodées dans les facettes des éléments de connaissance du domaine. Elles [7] servent à instancier les stratégies génériques du module tuteur (Hafsaoui, 2005). Selon Mayers (2001), cette séparation entre didactique et pédagogique est importante parce que (1) traditionnellement ces connaissances étaient toujours entremêlées et que (2) l'association des connaissances du domaine aux connaissances didactiques en fait des objets pédagogiques réutilisables. Il est à noter que les connaissances didactiques sont encodées en fonction du domaine de connaissances, mais que l'encodage des connaissances du domaine est tout à fait indépendant des connaissances didactiques. Ce chapitre a mentionné les principales facettes didactiques sans présenter les détails de leur représentation, car ceci déborde du cadre de cet ouvrage. Les lecteurs intéressés pourront se référer à un mémoire consacré entièrement au sujet (Hafsaoui, 2005).

[6] Dans *CREAM*, la même distinction existe : des ressources didactiques tutorielles implémentent la pédagogie pour l'enseignement d'un type de capacités, alors que d'autres ressources sont associés directement à l'enseignement de capacités précises.

[7] Elles spécifient généralement une ou des stratégies génériques à utiliser du module tuteur ainsi que la façon de les instancier.

2.2.8 Les connaissances procédurales erronées

Le document original *UV* (Mayers, 2001) propose de spécifier l'ensemble des procédures qui peuvent conduire à des erreurs. Ainsi, grâce aux données didactiques associées à ces procédures, le tuteur dispose de stratégies spécifiques pour corriger chaque type d'erreurs. La section 2.2.10 abordera l'identification des erreurs et leur remédiation.

Modéliser un domaine possédant un large nombre de procédures erronées

Certains domaines renferment une vaste gamme de procédures erronées. La spécification de la totalité de celles-ci est tout simplement incommode ou irréalisable. Najjar, Mayers et Fournier-Viger (2004b) proposent une solution à cette problématique, que ce mémoire ne décrira pas.

Les procédures complexes : unique source d'échecs

Seules les procédures complexes peuvent mener à une erreur ; les procédures primitives, prises individuellement, ne peuvent causer d'erreurs. Pour affirmer ceci, Mayers (2001) présume que les concepteurs modéliseront strictement en procédures primitives, les procédures maîtrisées. Par conséquent, la base de connaissances d'un laboratoire virtuel différera selon la clientèle cible. Par exemple, pour un étudiant, l'habileté d'additionner deux nombres inférieurs à 10 sera primitive, alors que pour un élève du primaire, elle sera sans doute complexe. D'autres types d'erreurs sont possibles, mais *UV* ne les traite pas dans l'état actuel du projet. Par exemple, l'erreur d'utiliser une procédure complexe ou primitive associée à un autre but que le but actuel, ou pour des arguments qui ne correspondent pas à la procédure.

2.2.9 Les procédures et les buts systèmes

Le terme *but système* désigne les buts prédéfinis réalisables au moyen de procédures prédéfinies nommées *procédures systèmes*. *UV* comporte des procédures systèmes pour trois raisons :

- elles n'appartiennent à aucun domaine (généralité) ;
- elles ne s'expriment pas aisément autrement, car elles possèdent des capacités supérieures à celles des autres procédures (elles peuvent influer sur l'ordre de réalisation des buts, etc.) ;
- elles représentent des opérations importantes et les prédéfinir, permet aux autres modules du système de les reconnaître et de les traiter de façon particulière.

Le document *UV* (Mayers, 2001) spécifie deux buts systèmes : *INSTANCIE* et *ÉVALUE*. La mise en oeuvre réalisée pour ce mémoire ajoute d'autres buts systèmes jugés pertinents. Cette section les décrit conjointement à ceux d'*UV* puisqu'ils complètent la spécification. Le tableau 2.7 illustre la liste des buts systèmes. Ils se déclinent en trois catégories :

- **Catégorie 1 :** les buts qui constituent des primitives psychologiques fondées dans le sens où leur existence est nécessaire en soi ou découle logiquement d'une théorie psychologique reconnue.
- **Catégorie 2 :** les buts considérés comme des primitives psychologiques probables ;
- **Catégorie 3 :** les buts systèmes qui représentent des opérations dont aucune recherche en psychologie ne leur justifie un traitement particulier, mais qu'*UV* inclut parce que ce sont des buts généraux que tout le monde (le public cible) maîtrise et qu'il serait fastidieux ou compliqué d'inclure dans chaque domaine, et qu'*UV* souhaite leur donner un statut particulier.

Les concepteurs de connaissances peuvent modéliser des domaines finement en recourant aux buts systèmes de catégories 1 et 2, puisqu'ils donnent accès à des primitives psychologiques du niveau le plus bas.

But système	Catégorie	Arguments	Résultat
INSTANCIE	1	« idConceptDécrit », $a_1, a_2, ...a_n$	Une cognition décrite
INSTANCIE2	1	$fct, a_1, a_2, ...a_n$	Une cognition décrite
ÉVALUE	3	$Cognitionécrite1$	Une cognition
TROUVE	2	$entier1, entier2$	Une cognition
SÉQUENCE	1	$But_1, But_2, ...But_n$	Le résultat de But_n
REMPLACE	2	$CognitionDécrite1, AncComp,$ $NouvComp$	Une cognition décrite
COMPOSANTE	1	$CognitionDécrite1, entier1$	Une cognition

Tableau 2.7 – Les différents buts systèmes et leur syntaxe

Sur le plan psychologique, à la connaissance de l'auteur de ce mémoire, il n'y a pas de recherches sur l'existence des buts systèmes, et encore moins sur leurs caractéristiques. Peut-être qu'elles sont apprises comme tous les autres et que, fort probablement, les mécanismes cognitifs sont plus complexes que ceux proposés dans *UV*. Cependant, leur usage s'impose pour les raisons décrites ci-dessus.

Le but système *INSTANCIE*

Le but système *INSTANCIE* réside à créer une instance d'un concept décrit par le mécanisme cognitif de *chunking* (Halford et al., 1993) (c.f. section 2.1.1). Cette primitive est essentielle pour simuler la création des instances de concepts décrits. Les arguments d'*INSTANCIE* consistent en une chaîne de caractères qui spécifie l'identificateur du concept décrit, suivie des occurrences de concepts qui le composeront. Le résultat est l'instance de concept décrit créé.

L'implantation d'*UV* comprend une seconde version d'*INSTANCIE* pour offrir aux concepteurs de connaissances une plus grande flexibilité dans la manière d'utiliser *INSTANCIE*. Les deux versions représentent le même mécanisme cognitif. La deuxième variante se nomme *INSTANCIE2*. Elle prend une fonction en paramètre plutôt que de prendre un identificateur de concept décrit. Le système détermine le type du concept décrit à ins-

tancier par le biais de la facette *association* de la fonction.

UV n'inclut pas de but système pour l'instanciation des concepts primitifs, pas plus que la mise en oeuvre d'*UV* réalisée pour ce mémoire, que la section 2.3 décrit. Dans cette mise en oeuvre, pour créer des cognitions primitives, un auteur doit modéliser une procédure primitive dont la facette *Méthode* pointe vers une méthode *Java* qui instancie une sous-classe de la classe *Java ConceptPrimitif.java*. La section 2.3.3 explique plus en détail le rôle de ces objets *Java* qui représentent les cognitions.

Le but système *ÉVALUE*

Tout concept décrit dans *UV* possède une fonction comme premier argument. *ÉVALUE* effectue l'évaluation de cette fonction sur les autres composantes du concept décrit *CognitionDécrite* et retourne le résultat. La facette *But-fonction* de la fonction indique le but à évaluer pour y parvenir.

Lorsqu'un apprenant évalue *(+ 3 5)*, il exécute le but *(ÉVALUE (+ 3 5))*. L'activation de ce but système provoque l'exécution d'une procédure système dont l'unique but consisterait à additionner 3 et 5. *UV* donne le statut de but système à *ÉVALUE* afin de simplifier la modélisation des connaissances, mais aussi pour permettre aux autres modules du système de reconnaître ce but et de le traiter de façon particulière, par exemple, pour savoir comment évaluer une fonction.

Le but système *SÉQUENCE*

Le but *SÉQUENCE* s'accomplit en effectuant plusieurs buts $(But_1, But_2, ...But_n)$ un à la suite de l'autre et en retournant le résultat du dernier. La justification de ce but s'appuie sur la théorie de Norman et Shallice, introduite à la section 2.1.2. Cooper et Shallice (2000) affirment qu'un ordre d'accomplissement n'est pas systématiquement présent entre les sous-buts d'une connaissance procédurale, mais que trois types de contraintes d'ordonnancement surviennent :

- les contraintes d'ordre physique qui dictent par exemple, qu'un individu doit ouvrir un sachet de sucre avant de verser son contenu ;
- les contraintes qui doivent être respectées pour garantir le succès d'une tâche, par exemple, un individu doit vider le contenu du sachet de sucre dans un café avant de jeter le sachet ;
- les contraintes établies par les préférences de l'individu qui imposent un ordre même quand celui-ci est arbitraire, par exemple, certains préfèrent ajouter le sucre avant le lait dans un café.

Le but *SÉQUENCE* offre une manière simple pour imposer un ordre parmi plusieurs sous-buts. L'autre façon de spécifier un ordre dans *UV* consiste à écrire un script tel que *(ButAdditionner x (But Additionner y z))*, utilisant des buts imbriqués. Quoiqu'*UV* n'offre pas de façon de spécifier un ordre arbitraire entre des sous-buts, ce problème se contourne en définissant une procédure pour chaque entrelacement de sous-buts possibles. Cependant, l'ajout d'un nouveau but système pour spécifier un ordre arbitraire serait préférable.

Le but système *TROUVE*

Le but *TROUVE* équivaut au but *episode* de *MIACE* (Mayers, 1997). Il consiste à parcourir la mémoire épisodique pour trouver le résultat d'un épisode antérieur en fonction de son contexte temporel. Ce but entre en action, par exemple, lorsqu'un individu tente de se rappeler la dernière tâche qu'il a entreprise avant le dîner.

L'exploration de la mémoire épisodique débute par une ascension vers le haut de *entier*1 épisodes parents par rapport à l'épisode actuel, se déplace ensuite de *entier*2 épisodes vers la gauche et finalement renvoie le résultat de l'épisode trouvé. Un épisode à gauche signifie l'épisode frère plus ancien immédiat, alors qu'un épisode en haut, le parent immédiat.

Fournier-Viger et Bouchard (2004); Najjar et al. (2005a) ont montré que l'architecture cognitive *ACT-R* se trouve limitée dans sa capacité à modéliser réalistement certains domaines puisqu'elle ne possède pas la capacité de se rappeler temporellement.

Le but système *REMPLACE*

Un humain ne peut considérer qu'un certain nombre de dimensions à la fois dans la résolution d'un problème (Halford et al., 2005; Cowan, 2001; Fisher, 1984). Chacune de ces dimensions correspond à une composante d'un concept décrit (cf. section 2.1.1) (Mayers, 1997). Par exemple, un étudiant pourrait manipuler une expression à simplifier (5 + (3 + 4)) composée de trois dimensions 5, + et (3 + 4).

Le but système *REMPLACE* modélise l'action de remplacer une dimension considérée, par une autre dimension ignorée actuellement. En d'autres mots, ce but consiste à substituer une composante *AncComp* par une composante *NouvComp* dans un concept décrit *CognitionDécrite*. Dans l'exemple mentionné, l'apprenant pourrait remplacer la composante (3 + 4) par 7 dans le concept décrit (5 + (3 + 4)). Il obtient en résultat la cognition décrite (5 + 7).

Halford et al. (1993) ne mentionnent pas explicitement ce mécanisme de remplacement. Toutefois, ce processus semble indispensable pour reproduire le comportement d'un humain qui au fur et à mesure de la résolution d'un problème abandonne certaines dimensions et en considère des nouvelles.

Le but système *COMPOSANTE*

Le but *COMPOSANTE* retourne la composante à l'emplacement *entier*1 d'un concept décrit *CognitionDécrite*. Il équivaut à l'utilisation du processus de *segmentation* en psychologie cognitive, qui permet d'extraire les parties d'un concept décrit (Halford et al., 1993) (c.f. section 2.1.1).

2.2.10 Le traitement explicite des buts

Les buts se distinguent des autres connaissances par une énergie qui leur est propre. L'introduction de ce concept n'est pas nouveau en soi. Il se retrouve par exemple dans des architectures cognitives telles qu'*ACT-R* (Anderson, 1993) et SOAR (Newell, 1990). Une originalité dans *UV* réside dans les bénéfices engendrés par la représentation particulière (facettes) des buts. Les avantages qui en découlent sont au nombre de deux (Mayers, 2001) : les buts (1) posent une base pour la formalisation des objectifs pédagogiques et (2) favorisent par leur nature l'identification des erreurs de l'apprenant et la construction d'une séquence de connaissances procédurales correctes pour y remédier. Les prochains paragraphes explicitent ces deux points.

Les objectifs pédagogiques en terme des buts

Ce mémoire considère deux types d'objectifs pédagogiques : (1) les objectifs portant sur une habileté à acquérir et (2) ceux qui portent sur une connaissance sémantique à maîtriser. Premièrement, vérifier l'acquisition d'une habileté équivaut à contrôler la maîtrise d'un but. L'importance réside dans l'aptitude de l'apprenant à atteindre le but, peu importe la procédure correcte qu'il emploie (plusieurs procédures correctes peuvent réaliser un même but). Si un apprenant réalise correctement un but un certain nombre de fois avec des problèmes de difficulté variée sans se tromper, un tuteur peut conclure que l'apprenant a intégré l'habileté correspondante. Par exemple, pour vérifier l'habileté de résoudre un système d'équations à trois équations et à trois inconnues, un tuteur pourrait donner une séquence d'exercices qui porteraient sur le but *ButRésoudreSystèmeÉquations3Inconnues3Variables*. Les procédures précises employées pour résoudre les problèmes n'intéressent pas le tuteur, dans la mesure où elles sont correctes. Pour contrôler l'acquisition d'une habileté plus spécifique telle qu'appliquer la méthode de Gauss-Jordan pour résoudre un système d'équations, le tuteur choisit simplement un but plus précis, par exemple, le but *ButRésoudreSystèmeÉquations3Inconnues3VariablesAvecGaussJordan*.

Deuxièmement, vérifier la maîtrise d'une connaissance sémantique requiert de s'assurer de la capacité à accomplir plusieurs buts qui l'utilisent. Par exemple, pour tester l'intégration du concept *ConstanteVrai* en réduction booléenne, un tuteur vérifie la maîtrise du but de faire la négation du vrai, la conjonction du vrai avec une proposition, etc.

En résumé, les objectifs pédagogiques qui portent sur une habileté s'expriment sous forme d'un but à maîtriser, alors que ceux qui portent sur un concept s'expriment en terme d'un ensemble de buts à maîtriser. En ce sens, *UV* reprend le point de vue d'Anderson et al. (1995) que l'apprentissage dans les *STI* doit être centré sur la maîtrise des connaissances procédurales.

Cette définition d'objectif pédagogique recouvre celle plus classique de chercheurs en pédagogie tel que Klausmeier (1990) qui indique que pour maîtriser un concept, un apprenant doit connaître l'ensemble des relations qui le caractérisent (par exemple, maîtriser le concept canari nécessite de connaître la relation couleur entre le concept *canari* et *jaune* et la relation *superconcept* entre *canari* et *oiseau*). Un concepteur peut encoder ces relations avec *UV* de deux façons (c.f. section 2.2.1) : (1) avec des concepts décrits semblables à la forme (*chat SousConceptDe animal*) et (*canari Couleur jaune*), ou (2) avec le mécanisme d'héritage intégré (pour les relations d'héritage seulement). Dans le premier cas, le concepteur ajoute des buts pour manipuler les concepts décrits qui encodent les relations. Les procédures associées à ces buts extraient l'information de ces concepts décrits. Le concepteur exprime les objectifs pédagogiques classiques en fonction de ces buts. Dans le second cas, il faudrait étendre la spécification d'*UV* pour ajouter de nouvelles procédures systèmes (voir section 2.2.9) qui permettraient d'extraire l'information des facettes qui décrivent un concept. Comme énoncé à la section 2.2.1, le mécanisme d'héritage d'*UV* est un raccourci qui ne reflète pas l'organisation majoritairement acceptée de la mémoire sémantique. Par conséquent, les buts et procédures systèmes ajoutés seraient également des raccourcis de modélisation.

Identification des erreurs et remédiation

Lorsque l'apprenant commet une erreur, *UV* considère que celle-ci résulte d'un mauvais choix de procédure complexe pour accomplir un but (cf. section 2.2.8). À partir des procédures primitives perçues par l'interface graphique, le système peut inférer les procédures complexes utilisées et par conséquent, la source de l'erreur. Puisque chaque procédure erronée contient dans sa facette *But* un pointeur vers le but associé, et que chaque but pointe par sa facette *Procédures* vers au moins une procédure correcte, le tuteur peut construire une séquence d'activités pédagogiques qui pallie l'erreur de l'apprenant. De plus, la présence d'informations didactiques dans la description des procédures permet une instanciation spécifique d'une stratégie didactique. Si la procédure est valide, ses facettes contiendront des façons de l'enseigner (ce qui peut être utile lorsque l'apprenant a bien commencé mais est bloqué), alors que si elle est erronée, ses facettes indiqueront des stratégies spécifiques de remédiation. Pour ces raisons, la représentation explicite des buts favorise l'identification des erreurs et la remédiation.

2.3 La mise en oeuvre d'*UV*

Pour ce travail, une implantation d'*UV* a été réalisée afin de mieux cerner ses qualités et ses faiblesses. Cette implantation a servi de base pour trois laboratoires virtuels : (1) un laboratoire de génie génétique (Zhao, 2003), (2) un laboratoire sur la réduction booléenne (Najjar et al., 2004b; Najjar et Mayers, 2004; Najjar et al., 2004a) et (3) un laboratoire de conversion d'instructions en binaire, pour un cours sur l'architecture des ordinateurs (Hamadouche, 2004). Cette section décrit les principales décisions prises pour réaliser l'implantation, de la représentation des connaissances en langage *XML*[8] au module *Java* qui interprète les procédures primitives et complexes et génère la mémoire épisodique. La section vise trois objectifs : (1) introduire certains enjeux discutés dans la

[8]XML : www.w3.org/XML/

conclusion de ce chapitre, (2) donner une meilleure compréhension d'*UV* par des exemples concrets d'utilisation et (3) permettre une comparaison plus approfondie avec la mise en oeuvre du modèle proposé au chapitre 5.

2.3.1 Le métalangage *XML* pour encoder les connaissances

XML est un métalangage, c'est-à-dire un langage pour définir d'autres langages. Les raisons suivantes justifient le choix d'*XML* comme base pour représenter nos fichiers de connaissances : (1) *XML* est un standard du *W3C*[9] gratuit et indépendant de toute plateforme, (2) il adopte un format textuel lisible par un humain, (3) des analyseurs syntaxiques et des outils, libres de droits, performants et faciles d'utilisation sont disponibles, enfin (4) de nombreuses spécifications utilisent *XML* (il facilite l'arrimage avec ces standards).

La technologie *XML Schema*[10] a permis de formaliser la structure des fichiers de cette mise-en-oeuvre. Deux formats de fichiers ont été établis : le premier pour les connaissances sémantiques et procédurales, communes à tous les usagers d'un laboratoire virtuel et le second, pour les connaissances épisodiques propres à chaque apprenant.

2.3.2 L'entrée des connaissances avec *XMLSpy* et *DOKGETT*

En premier lieu, un formulaire *XMLSpy*[11] a été créé pour saisir les connaissances des fichiers de connaissances sémantiques et procédurales. La figure 2.5 montre la vue par ce formulaire d'un fichier de connaissances pour la réduction booléenne. Ce formulaire facilite l'entrée du contenu et prévient les erreurs au niveau *XML* (placement des balises, etc.), mais il offre très peu pour s'assurer de la cohérence des connaissances. Pour outre-

[9] *W3C* : www.w3.org/
[10] XMLSchema : www.w3.org/XML/Schema
[11] XMLSpy est un logiciel d'édition *XML* : www.xmlspy.com

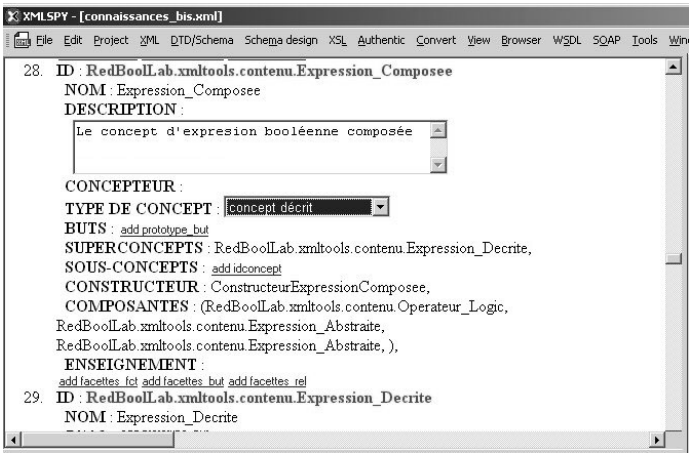

Figure 2.5 – La vue *XMLSpy* des fichiers de connaissances

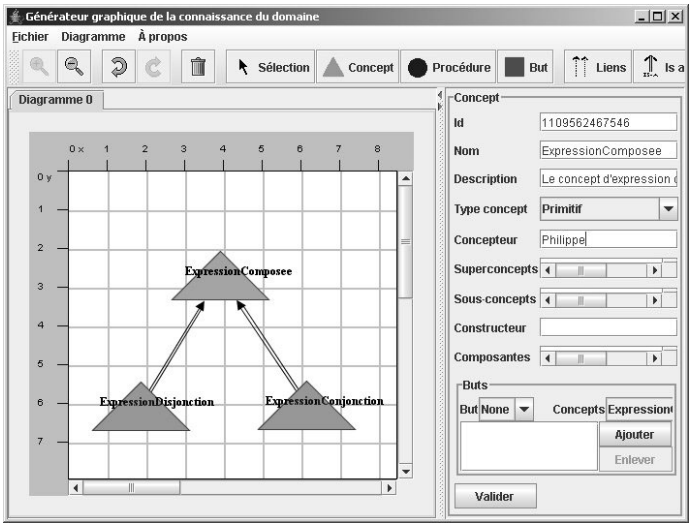

Figure 2.6 – Le système auteur *DOKGETT*

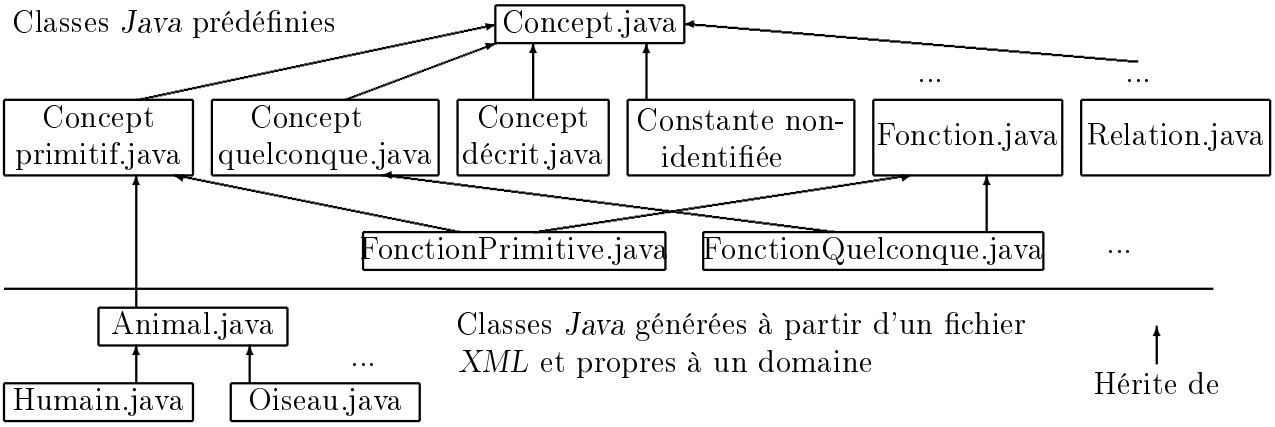

Figure 2.7 – La hiérarchie de classes *Java* générées par l'implantation d'*UV*

passer ces limites et apporter une interface plus conviviale, Najjar et al. (2005c,b) ont conçu un système auteur nommé *DOKGETT* (DOmain Knowledge GEnerator auThoring Tool) qui permet de modéliser graphiquement les connaissances au moyen de deux types de diagrammes : (1) les diagrammes conceptuels et (2) les diagrammes procéduraux. Les premiers représentent visuellement les relations d'héritage entre concepts, alors que les seconds établissent les liens entre les procédures, les buts et les concepts. La figure 2.6 illustre un diagramme conceptuel qui définit un concept *ExpressionComposée* et deux sous-concepts *ExpressionDisjonction* et *ExpressionConjonction*.

2.3.3 La génération des classes *Java*

Une fois les connaissances sémantiques et procédurales établies pour un laboratoire, le concepteur de connaissances génère des classes *Java* à partir du fichier *XML*, grâce à un logiciel conçu pour cette mise en oeuvre. Chacune des classes générées représente un des concepts présents dans le fichier *XML*. Elles renferment des squelettes de méthodes qui correspondent aux procédures primitives. Le concepteur remplit les squelettes avec le code désiré : ce code peut faire appel au noyau fonctionnel, à l'interface usager ou effectuer d'autres traitements. Pour déterminer quelles méthodes inclure dans quelles classes,

l'outil générateur de classes se base sur l'information contenue dans la facettes *Méthode*. Cette dernière indique, pour chaque procédure primitive, un nom de méthode *Java*, les arguments de la procédure à lui passer et l'argument de la procédure qui doit contenir la méthode. La facette *Méthode* peut également pointer vers un constructeur d'une classe *Java* pour la création des cognitions primitives. Les classes générées complètent la spécification *XML* par l'ajout du code compilé qui définit le comportement des procédures primitives.

L'héritage entre classes générées

Si la facette *Superconcepts* d'un concept A indique que A hérite de concepts B et C , la classe *Java* représentant le concept A héritera de celles des concepts B et C. L'héritage entre les classes *Java* assure que les procédures d'un concept s'appliquent aussi à ses sous-concepts. Pour réaliser l'héritage multiple - impossible en *Java* - ce travail a recours à l'*algorithme de linéarisation C3* (Barrett et al., 1996) utilisé notamment pour le langage de programmation *Python 2.3* (Simionato, 2003). Pour un ensemble de classes respectant des contraintes de base (la hiérarchie ne contient aucun cycle, etc.) et pour une classe donnée, un algorithme de linéarisation permet de calculer un ordre de parcours des superclasses. L'intérêt de l'algorithme *C3* est qu'il respecte plusieurs propriétés intéressantes telles que la monotonicité et la cohérence avec le *graphe de préséance étendu* (le lecteur intéressé est invité à se référer à l'article de Barrett et al. (1996) qui compare l'algorithme *C3* avec les principaux algorithmes de linéarisation).

Les classes *Java* prédéfinies

Les classes *Java*, selon leur type (concept primitif, concept décrit, fonction primitive, etc.), héritent de classes prédéfinies correspondantes (exemple : figure 2.7) qui regroupent des méthodes propres à chaque type de concept. Par exemple, la classe *ConceptDécrit.java* contient la méthode *évalue* qui correspond à la procédure système homonyme.

2.3.4 La gestion des concepts quelconques et non identifiés

Le document qui décrit *UV* (Mayers, 2001) reste vague sur l'utilisation des concepts quelconques. Aucune facette d'un concept quelconque ne permet de spécifier de quel concept un concept est quelconque (idem pour les concepts non identifiés). Par exemple, rien ne permet de spécifier que le concept *HumainQuelconque* est le concept quelconque du concept *Humain*. En conséquence, le travail d'implantation souscrit aux décisions suivantes :

1. tout concept spécifique (ex. : *Humain*) hérite de son concept quelconque (ex. : *HumainQuelconque*), s'il y en a un.
2. l'implantation autorise seulement la substitution d'une composante d'un concept décrit, par une plus spécifique (conséquence de la décision 1).

Pour les concepts non identifiés, rien n'est prévu ; ils sont inutilisables dans cette première implantation.

2.3.5 Les services du module de gestion des connaissances

Le *MGC* programmé pour cette implantation propose les services suivants : (1) simulation d'exécution de procédures avec génération de la mémoire épisodique correspondante, (2) accès aux connaissances sémantiques, procédurales et épisodiques, enfin (3) sauvegarde/chargement de la mémoire épisodique dans un fichier. Les prochaines sous-sections décrivent le fonctionnement du *MGC* par des exemples.

L'initialisation du *MGC*

Pendant la phase d'initialisation d'un laboratoire virtuel, le système *STI* demande au *MGC* de charger les fichiers *XML* de connaissances en mémoire (figure 2.8). La composante nommée *interpréteur* est le coeur du *MGC*; elle reçoit les requêtes, effectue les traitements et retourne les résultats.

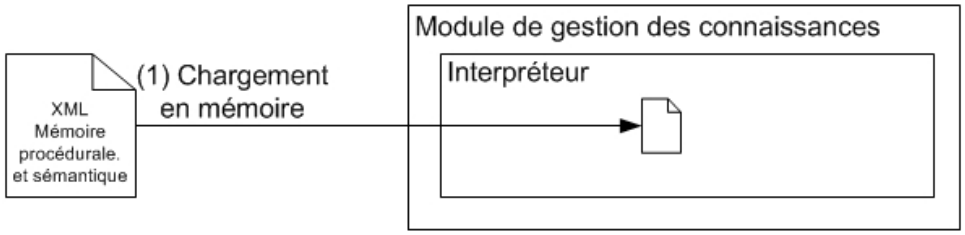

Figure 2.8 – Processus d'initialisation du *MGC*

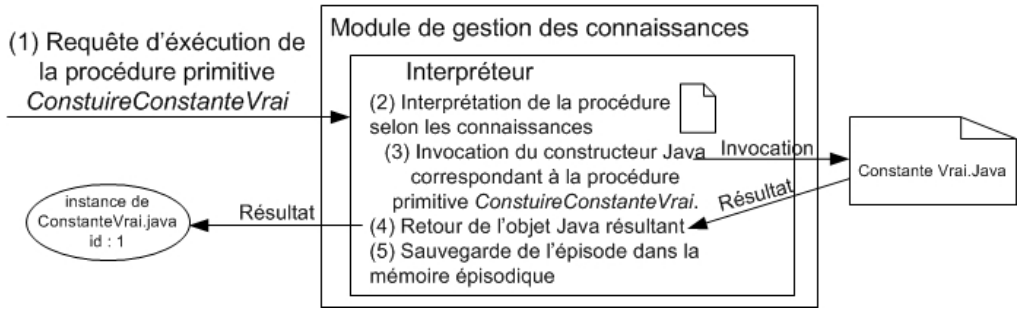

Figure 2.9 – Scénario d'exécution de la procédure primitive *ConstruireConstanteVrai*

Des scénarios d'exécution de procédures primitives

La figure 2.9 illustre la prise en charge par le *MGC* de l'exécution de la procédure primitive *ConstruireConstanteVrai*. Un tel scénario se produit notamment lorsque l'apprenant effectue la procédure avec l'interface graphique et que le système tutoriel désire générer les épisodes correspondants de la mémoire épisodique.

Tout d'abord, la requête d'exécution de la procédure *ConstruireConstanteVrai* parvient à l'interpréteur, qui fouillera dans le fichier *XML* chargé en mémoire afin de récupérer la définition de la procédure. Cet exemple suppose que la procédure primitive correspond à une méthode *Java* spécifique, le constructeur de la classe *ConstanteVrai.java*.

Par réflexivité, l'interpréteur invoquera la méthode et récupérera le résultat ; dans ce cas, une nouvelle instance de *ConstanteVrai.java*. La procédure primitive étant exécutée,

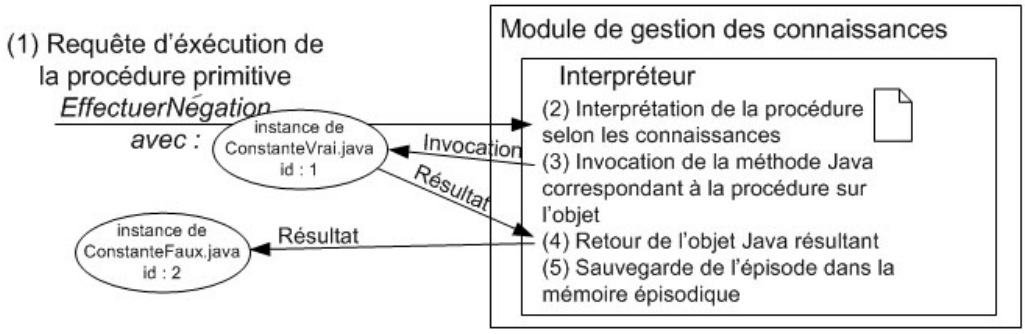

Figure 2.10 – Scénario d'exécution de la procédure primitive *EffectuerNégation*

l'interpréteur retournera le résultat (l'objet *Java* intitulé *instance 1* sur le diagramme) et sauvegardera l'épisode dans la mémoire épisodique.

Considérons la suite, où le *MGC* doit exécuter une seconde procédure, ce que la figure 2.10 décrit. La requête d'effectuer la procédure *EffectuerNégation* avec l'argument *instance 1* est envoyée au *MGC*. L'interpréteur décodera la procédure en utilisant le fichier de connaissances en mémoire, qui lui indiquera que la méthode *Java* n'est pas un constructeur, mais une méthode invocable à partir du premier argument : *instance 1*. L'interpréteur invoquera la méthode. En résultat, la machine virtuelle *Java* retournera une nouvelle instance de la classe *ConstanteFaux.java*, nommée *instance 2* dans le diagramme. L'interpréteur renverra cette instance à l'initiateur de la requête, et enregistrera l'épisode dans la mémoire épisodique.

2.3.6 L'algorithme général de l'exécution d'une procédure

La section précédente décrivait par des exemples le mécanisme d'exécution des procédures primitives par le *MGC*. La figure 2.11 illustre l'algorithme général de l'exécution d'une procédure primitive ou complexe. L'algorithme reçoit en entrée un identificateur de procédure et des cognitions auxquelles la procédure s'appliquera. Cet algorithme traite deux cas : l'exécution d'une procédure primitive et celle d'une procédure complexe. Advenant

AlgExecProc(Procedure Px, Arguments a_1, ... a_n)

1. Si Px est primitive,
   (a) Vérifier dans le fichier *XML* en mémoire quelle méthode *Java* correspond à la procédure Px.
   (b) Exécuter la méthode *Java* (soit un constructeur, ou une méthode invocable avec un ou plusieurs des arguments a_1, ... a_n).
   (c) Sauvegarder l'épisode de l'exécution de cette procédure.
   (d) Retourner le résultat obtenu en (b)
2. Sinon
   (a) Récupérer les k sous-buts de Px dans le fichier *XML* en mémoire.
   (b) Pour chaque sous-but Sb_1, Sb_2... Sb_k
      i. Si le but est un but système,
         A. Exécuter la procédure système correspondante.
         B. Sauvegarder l'épisode de l'exécution de cette procédure.
      ii. Sinon
         A. Demander au requérant de choisir une procédure Py.
         B. Selon la spécification de Px, déterminer les arguments b_1 , b_2, ... b_m requis parmi a_1, ... a_n et les résultats des sous-buts précédents.
         C. Effectuer AlgExecProc(Py, b_1 , b_2, ... b_m)
         D. Sauvegarder l'épisode de l'exécution de cette procédure.
   (c) Selon la spécification de Px, retourner le résultat d'un des buts Sb_1, Sb_2... Sb_k.

Figure 2.11 – L'algorithme général de l'exécution d'une procédure *AlgExecProc*

une procédure primitive, le *MGC* invoque la méthode *Java* spécifiée dans la facette *Méthode* de la procédure avec, comme paramètres, les cognitions. Le scénario de la section précédente illustre ce cas. Pour une procédure complexe, le *MGC* traite les sous-buts un à la suite de l'autre. Pour un but-système, le *MGC* exécute la procédure système correspondante et récupère le résultat. Pour un but ordinaire, le *MGC* exige que l'entité logicielle utilisant ses services, choisisse la procédure à utiliser pour accomplir le but, car un but se réalise possiblement par plusieurs procédures. Une fois la procédure choisie, le *MGC* appelle récursivement l'algorithme avec cette procédure afin de poursuivre le processus avec ses sous-buts, si elle est complexe, ou de l'appliquer, si elle est primitive. Pour chaque procédure exécutée, le *MGC* génère un épisode de la mémoire épisodique qui mémorise l'intention (le but), le moyen (la procédure), les paramètres de la procédure, le résultat, le moment, etc.

2.4 Conclusion

Cette section présente une critique de divers aspects d'*UV* : les sections 2.4.1, 2.4.2, 2.4.3, 2.4.4 et 2.4.5 s'intéressent respectivement à la mémoire épisodique d'*UV*, à la vraisemblance psychologique des ajouts inspirés de la logique, aux mécanismes d'inférence et au parallèle avec la logique des prédicats, à la complexité du langage, et finalement, au mécanisme d'héritage et à la réutilisation des connaissances. La section 2.4.6 dresse ensuite un bilan des qualités et des faiblesses d'*UV*.

2.4.1 La critique de la mémoire épisodique d'*UV*

La mémoire épisodique n'est pas présente explicitement dans plusieurs théories de la cognition telles qu'*ACT-R* (Anderson, 1993), mais cette mémoire peut être simulée d'une façon approximative par l'ajout de chunks dans la mémoire sémantique. L'usage de cet

artifice implique au moins trois inconvénients :

- *ACT-R* ne peut pas effectuer de recherche en mémoire long terme selon un contexte temporel, ni récupérer des éléments temporellement liés ;
- *ACT-R* ne possède pas la capacité autonoétique de la mémoire épisodique : lorsqu'*ACT-R* récupère un souvenir de sa mémoire long terme, *ACT-R* ne peut le percevoir comme un événement antérieur (Nuxoll et Laird, 2004) ;
- reproduire un comportement humain implique dans certains cas une perte de réalisme de la modélisation (Fournier-Viger et Bouchard, 2004; Najjar et al., 2005a), ce qui va à l'encontre des arguments pour l'usage d'un modèle cognitif (cette conséquence résulte des deux premières).

Quoique certaines évidences appuient la présence d'une mémoire épisodique sous forme d'une mémoire distincte (Tulving, 2002; Shastri, 2002, 2001; Garagnani et al., 2002), aucune étude n'en détermine la représentation précise. En effet, plusieurs chercheurs proposent des modèles informatiques (Nuxoll et Laird, 2004; Mayers, 1997) et des modèles neurologiques (Bond, 2005), mais aucun de ces modèles ne dispose de validations expérimentales exhaustives. Il est donc légitime de se questionner sur la validité psychologique de la représentation proposée dans *MIACE* (Mayers, 1997) et reprise dans *UV* (Mayers, 2001).

Nonobstant cette objection, la structure proposée s'avère intéressante, car elle propose un historique structuré en fonction des intentions et actions de l'apprenant ; une organisation qui semble naturelle. Cette représentation suffit pour recréer tous les actes d'un apprenant dans les moindres détails (intentions, procédures choisies et cognitions utilisées). Cette dernière caractéristique est particulièrement intéressante sur le plan pédagogique. En effet, la trace épisodique selon Weber (1996) permet au module tutoriel (1) de montrer à l'apprenant des exemples et des informations provenant de son historique et (2) d'inférer ce que l'apprenant essaye de faire à l'aide de l'analyse de son comportement antérieur.

2.4.2 La vraisemblance psychologique des ajouts inspirés de la logique

La seconde critique provient également du point de vue de la psychologie cognitive. *UV* ajoute plusieurs facettes prédéfinies pour décrire les fonctions, relations, concepts quelconques et concepts non identifiés, afin de faciliter le raisonnement dans un domaine de connaissances. Cet ajout laisse supposer qu'à partir d'un certain âge, tous les humains sont capables de manipuler de façon plus ou moins consciente les objets de ces catégories à partir de ces facettes. C'est une hypothèse qui n'est pas prouvée. Il faut remarquer qu'à cet égard les architectures cognitives alternatives sont silencieuses. Par exemple, dans *ACT-R* (Anderson, 1993) un auteur doit définir lui-même les facettes des fonctions et des relations s'il désire les utiliser dans une modélisation. En somme, même si les facettes supplémentaires d'*UV* sont utiles, elles doivent jusqu'à preuve du contraire, être considérées comme un raccourci informatique.

2.4.3 Les mécanismes d'inférence et le parallèle avec la logique des prédicats

D'autre part, Mayers (2001) prévoyait l'utilisation de mécanismes d'inférence pour raisonner sur les connaissances sémantiques décrites avec les primitives inspirées de la logique. Les principales tâches d'inférence auxquelles il est possible de s'attendre sont (1) de déterminer si une cognition décrite équivaut à une autre logiquement, (2) d'inférer des relations entre les concepts sur la base de leur définition logique et (3) de permettre la création d'un modèle expert et d'un module tuteur qui usent de ces relations pour diverses tâches telles que montrer à l'apprenant des liens logiques pertinents entre les concepts d'un problème et justifier certaines décisions de résolution de problèmes. Le rôle de l'inférence ne fut pas décrit dans le document *UV*, pas plus que les mécanismes d'inférence et le parallèle entre *UV* et la logique des prédicats. Par conséquent, aucune

étude sur l'expressivité et la complexité de l'inférence ne fut réalisée pour choisir les primitives logiques d'*UV*.

2.4.4 La complexité du langage

À l'écriture d'*UV*, aucune mise en oeuvre n'avait été réalisée. De plus, Mayers (2001) mentionnait des exemples brefs et restreints au domaine des équations mathématiques. En conséquence, plusieurs aspects de la sémantique du langage demeuraient flous. L'imprécision de la spécification et la quantité de facettes offertes - plus de 40 - en font un langage complexe. Cette complexité se répercute sur les concepteurs de contenu qui doivent maîtriser toutes ces facettes, mais aussi sur la complexité de l'élaboration de systèmes auteurs ou tout autre type d'outils conformes à cette spécification.

2.4.5 Le mécanisme d'héritage et la réutilisation des connaissances

L'emploi du mécanisme d'héritage rend la réutilisation des connaissances laborieuse pour deux raisons : (1) l'ajout d'un concept requiert son intégration dans une hiérarchie et (2) la réutilisation d'un concept pour un autre laboratoire demande l'extraction de ce concept d'une hiérarchie. Quoique l'utilisation du mécanisme d'héritage soit facultative, un auteur peut difficilement y renoncer parce que (1) l'usage des concepts quelconques dans l'implantation requiert l'héritage (cf. section 2.3.4) et (2) l'héritage réduit la taille des modélisations, car il permet de définir des buts, procédures, concepts décrits, fonctions et relations qui s'appliquent à plusieurs concepts.

2.4.6 Le bilan

Cette section clôt le chapitre avec un bilan des qualités et des faiblesses d'*UV*. *UV* affiche les points forts suivants :

- une fondation psychologique,
- une séparation entre connaissances didactiques et pédagogiques,
- le traitement explicite des buts,
- une mémoire épisodique sous forme d'historique structuré des actions et intentions.

Les aspects ci-après requièrent des changements :

- le mécanisme d'héritage qui gêne la réutilisation,
- l'intégration réalisée de la logique qui résulte en un très grand nombre de facettes, une complexité d'inférence inconnue, une expressivité non étudiée et qui accuse un manque de justifications psychologiques.

Les chapitres qui suivent présentent la notion d'objet d'apprentissage et les logiques de description, qui permettront par une combinaison adéquate avec *UV* de proposer un modèle qui pallie plusieurs de ces lacunes.

CHAPITRE 3

Les objets d'apprentissage

3.1 Introduction

Les objets d'apprentissage constituent une vision, plutôt qu'une technique précise pour représenter les connaissances. Il s'agit d'un ensemble de principes et de standards qui visent à maximiser la réutilisation et le partage du matériel pédagogique. Plusieurs institutions emploient ce concept très étudié dans le domaine de l'apprentissage en ligne. Par exemple, la Télé-université du Québec structure le matériel pédagogique de ses cours en objets d'apprentissage et les rend accessibles par l'intermédiaire du logiciel *EXPLOR@-2*[1]. Plusieurs initiatives visent également à partager les objets d'apprentissage entre institutions, tel que le dépôt canadien *MERLOT*[2]. Cependant, même si des standards explicitent comment décrire un objet d'apprentissage avec des métadonnées ou comment intégrer les objets dans des scénarios pédagogiques, il n'existe aucun consensus sur le format des objets d'apprentissage. Plus particulièrement, aucun *STI* ne propose d'encoder les connaissances basées sur un modèle cognitif sous forme d'objets d'apprentissage, afin

[1] http://explora2.licef.teluq.uquebec.ca/demo/fr/

[2] MERLOT (Multimedia Educational Ressources for Learning and Online Teaching) : http://www.merlot.org

de créer des objets qui peuvent servir de base à un enseignement hautement personnalisé.

Le chapitre se divise en trois sections : (1) la section 3.2 situe la théorie des objets d'apprentissage dans son contexte d'émergence, (2) la section 3.3 explicite la notion d'objet d'apprentissage et (3) la section 3.4 conclut par une présentation de quelques préoccupations actuelles des chercheurs qui souhaitent rendre commun le concept d'objet d'apprentissage et décrit une opportunité pour les *STI*.

3.2 Le contexte en éducation

Dans le contexte de l'économie du savoir, il est de plus en plus crucial d'actualiser ses connaissances et son savoir-faire. Puisque cette situation concerne de nombreux professionnels, une forte demande pour l'éducation asynchrone (en temps et lieu différés) se manifeste. De surcroît, la réalisation de trois objectifs bénéficieraient aux apprenants et aux établissements d'enseignement : la diminution des coûts de développement du matériel pédagogique et de l'enseignement, l'augmentation du partage de matériel pédagogique entre les auteurs et l'amélioration de l'accessibilité à l'éducation. Trois moyens d'adresser ces objectifs prédominent :

– (1) la formation à distance ;
– (2) la formation en ligne ;
– (3) l'utilisation des objets d'apprentissage.

Le troisième moyen, issu du domaine de la formation en ligne, est le sujet de ce chapitre. Cette section décrit le concept de formation en ligne afin de mettre en perspective la notion d'objet d'apprentissage.

La formation en ligne

Le terme *formation à distance* désigne l'ensemble des procédés d'enseignement grâce auxquels des apprenants peuvent effectuer des activités d'apprentissage sans avoir à se rendre

en un lieu physique. Une des formes les plus répandues sont les cours par correspondance, mais d'autres existent. Le sous-ensemble qui exploite les technologies informatiques se nomme *formation en ligne*. L'informatique procure certains avantages par rapport à la formation à distance traditionnelle :

- le format électronique favorise la réutilisation du matériel pédagogique et permet l'usage d'éléments multimédias et/ou interactifs ;
- l'enseignement ne se limite plus à la communication bidirectionnelle entre l'institution d'enseignement et un apprenant. Internet rend possible des modes de transmission tels que la vidéoconférence et les forums de discussion. Par conséquent, les activités peuvent être aussi collectives (Paquette, 2002).

Un nouveau paradigme de conception pédagogique

La formation en ligne amène un nouveau paradigme de conception pédagogique. Les équipes qui conçoivent les cours en ligne se composent souvent d'une vaste gamme de professionnels : infographistes, programmeurs, professeurs, etc. (Paquette, 2002). Ils utilisent des logiciels élaborés pour concevoir les cours, les administrer et permettre aux apprenants d'y accéder. Certains chercheurs proposent de nouvelles méthodologies pour la conception des formations en ligne, telle que la méthode MISA[3] (Paquette et al., 1997; Paquette, 2002).

Des chercheurs de l'Université d'Ottawa ont constaté lors d'une étude, qu'un cours d'autoformation multimédia d'une durée de quatre heures destiné à du personnel technique d'une grande entreprise, a demandé 1156 heures de travail de conception (un ratio de 1 pour 250) (Paquette, 2002). Quoique la conception initiale de matériel pédagogique sur support électronique demande beaucoup de travail et d'argent, sa durée de vie peut s'étaler par la suite sur plusieurs années.

[3]MISA (Méthode d'Ingénierie des Systèmes d'Apprentissage) est une méthode d'ingénierie qui décrit un ensemble de tâches à réaliser, de décisions à prendre et de documents à produire pour concevoir un système d'apprentissage.

Les paradigmes d'apprentissage de formation en ligne

Paquette (2002) dénombre cinq paradigmes pédagogiques de formation en ligne. Cette section énumère les trois qui utilisent des objets d'apprentissage :

La classe technologique répartie : une variation de la classe traditionnelle où les différents acteurs communiquent par un système de vidéoconférence et disposent d'outils auxiliaires (magnétoscope, lecteur de disques compacts, etc.).

L'autoformation Web hypermédia : l'apprenant acquiert des connaissances de façon autonome en étudiant un contenu préfabriqué et accessible sur ordinateur.

L'enseignement en ligne : une variation de l'*autoformation Web hypermédia* où un formateur interagit de façon asynchrone pour coordonner les activités des apprenants et faire un suivi de leur cheminement, et où les activités peuvent être collaboratives. La Télé-université du Québec emploie actuellement ce modèle pour dispenser ses cours par le biais d'Internet.

La perspective de croissance de l'apprentissage en ligne

En conséquence de l'économie du savoir, une forte croissance du marché de l'apprentissage en ligne est appréhendée. Selon la firme de marketing américaine IDC, la taille du marché qui représentait 6.6 milliards de dollars US au début de 2002, devrait atteindre 23.7 milliards en 2006 (Haddad, 2003).

Les enjeux qui influencent l'efficacité des formations en ligne

Les auteurs de formations en ligne doivent relever un certain nombre de défis pour assurer une formation efficace. La littérature recense plusieurs facteurs qui influencent la réussite des apprenants. Selon Chen et Lin (2002), treize éléments exercent un rôle déterminant. Les six premiers sont, dans l'ordre : (1) l'occupation d'un emploi par l'apprenant, (2) la vitesse et la fiabilité de la connexion Internet, (3) la motivation de l'apprenant, (4) le

logiciel utilisé, (5) le contenu du cours et (6) l'interaction avec les autres participants. Au niveau de l'enseignement, le système d'apprentissage doit fournir à l'apprenant une séquence d'information et une approche pédagogique pertinente selon ses buts, ses préférences et ses habiletés. Les concepteurs de cours en ligne doivent également prendre garde de ne pas surestimer les capacités d'autonomie de l'apprenant (Hotte et Leroux, 2003).

3.3 Les objets d'apprentissage

Les objets d'apprentissage constituent une vision, plutôt qu'une technique précise pour représenter les connaissances. Elle repose sur un principe de base : celui de structurer le matériel pédagogique en unités réutilisables. La littérature contient des définitions plus ou moins restrictives du concept d'objet d'apprentissage. Par exemple, Duncan (2003) souligne que l'IEEE (2005) le définit comme « n'importe quelle entité numérique ou non, utilisable pour l'éducation, l'apprentissage ou la formation », Wiley (2000) le décrit comme « n'importe quelle ressource numérique réutilisable dans un cadre d'apprentissage » et Koper (2003) ajoute qu'« un objet d'apprentissage est autonome et réutilisable ». Pour résumer ces énoncés, un objet d'apprentissage est une entité numérique ou non, autonome et réutilisable dans des activités pédagogiques. Les sections suivantes détaillent les cinq étapes de leur cycle de vie afin de préciser le rôle et la nature des objets apprentissage : (1) la création d'unités d'information, (2) l'ajout de métadonnées, (3) l'agrégation d'objets d'apprentissage, (4) l'intégration des objets dans un scénario pédagogique et (5) la livraison des objets d'apprentissage aux apprenants.

3.3.1 Étape 1 : La création d'unités d'information

La première étape du cycle de vie d'un objet d'apprentissage consiste à créer une unité d'information ; un document de n'importe quel format (pages Web, images, applets *Java*, etc.) et de n'importe quel type (séquence vidéo, simulation interactive, figure, texte, etc.). De manière générale, un auteur d'unités d'information choisit le format en fonction de l'ensemble des logiciels avec lesquels il souhaite une compatibilité. Dans le domaine de la formation en ligne, les institutions préfèrent généralement les documents Web comme unités d'information afin de pouvoir les présenter par l'intermédiaire de fureteurs. Des exemples typiques d'objets d'information sont une page Web qui explique le processus de photosynthèse, une image qui représente une vue schématique du système solaire, un livre électronique sur l'algèbre linéaire, une image d'une peinture de Picasso et un enregistrement d'une oeuvre du violoniste Nicolo Paganini.

Les auteurs suggèrent différents principes pour la conception d'unités d'information. Ce paragraphe en relève quatre : construire des unités d'information (1) autonomes, (2) personnalisables, (3) de faible granularité et (4) des unités qui ne sont pas neutres pédagogiquement. Les trois premiers principes visent à maximiser la capacité de réutilisation des unités d'information, une idée fondamentale de l'approche des objets d'apprentissage. Premièrement, créer des unités autonomes signifie construire des unités libres de référence à des contextes extérieurs. À titre d'exemple, un auteur qui conçoit une unité qui explique le fonctionnement d'un moteur doit idéalement éviter de faire référence à d'autres unités d'information, car elles pourraient être utilisées séparément. Les auteurs doivent néanmoins prendre garde dans le processus de décontextualisation, car comme Wiley et al. (2000) le souligne, les éléments très décontextualisés sont plus difficiles à indexer et possèdent une sémantique moins claire pour une machine. Le deuxième principe dicte de créer des unités personnalisables pour faciliter l'intégration à des contextes particuliers en permettant l'adaptation des unités (Moodie et Kunz, 2003). Le troisième principe stipule qu'une granularité trop grande réduit le nombre d'unités avec lesquelles une unité pourra

s'assembler (Wiley et al., 2000). Par exemple, un livre se réutilise moins facilement qu'un de ses chapitres ou que l'un de ses paragraphes. Le quatrième principe dicte qu'un auteur d'unités d'information ne doit pas viser la neutralité pédagogique, car cela amène souvent une dégradation de la pertinence pédagogique des unités d'information (Friesen, 2004).

3.3.2 Étape 2 : L'ajout de métadonnées

La deuxième étape du cycle de vie des objets d'apprentissage consiste à ajouter des métadonnées aux objets d'information. Les métadonnées sont des données structurées qui décrivent d'autres données. Actuellement, le standard de métadonnées dans le domaine des objets d'apprentissage est *LOM* (IEEE, 2005). Ce standard propose environ 80 éléments regroupés en neuf catégories pour décrire une unité d'information. Voici les cinq catégories principales et leur description :

- *général* : contient les éléments pour énoncer les caractéristiques générales de l'unité comme la langue utilisée, le titre, la description, les mots-clés, etc. ;
- *état actuel et historique* : regroupe les entrées relatives à l'état actuel de l'unité d'information et à son historique, par exemple les contributeurs, la version, le stade de développement de l'unité (brouillon, en révision, version finale, etc), etc. ;
- *caractéristiques pédagogiques* : permet de décrire l'aspect éducationnel des unités d'information, ce qui comprend le type d'interactivité, le degré d'interactivité, la densité sémantique, la difficulté, l'âge de la clientèle typique, le contexte pédagogique suggéré, etc. ;
- *description technique* : contient les éléments pour décrire les caractéristiques de l'unité d'information d'un point de vue technique, parmi ceux-ci, il y a la taille, le format, la durée, les préalables, etc. ;
- *conditions d'utilisation* : comprend les éléments qui décrivent les conditions d'utilisation telles que les coûts, les droits d'auteur, les restrictions, etc.

Les métadonnées présentent une utilité triple : elles facilitent la localisation des unités d'information dans des dépôts, elles renseignent sur la façon d'utiliser les unités (modalités d'utilisation, droits d'auteur, etc.) et elles ouvrent la voie à un assemblage automatique des unités d'information par une machine (Rehak et Mason, 2003; Wiley, 2000; Campbell, 2003). La présence de métadonnées distingue aussi les unités d'information des objets d'apprentissage selon Duncan (2003). Plus précisément, ajouter un objectif pédagogique sous forme de métadonnées transforme une unité d'information en objet d'apprentissage. L'idée sous-jacente à cette définition est que l'ajout d'un objectif pédagogique indique clairement que l'unité d'information se destine à une vocation pédagogique, donc qu'elle est un objet d'apprentissage.

Par ailleurs, remplir correctement les 80 champs de métadonnées de *LOM* s'avère parfois difficile (Rehak et Mason, 2003). Par exemple, un auteur éprouvera peut-être de la difficulté à spécifier pour une photographie de la Muraille de Chine, la densité sémantique, la durée d'utilisation, le public cible, la difficulté et l'activité d'apprentissage. Pour pallier ce problème, diverses organisations ont établi des *profils d'application* de *LOM* tels que *Cancore* (Friesen, 2003; Campbell, 2003). *Cancore* réduit le nombre d'éléments de *LOM* à 39 éléments. De plus, il propose un guide et une terminologie commune pour remplir les métadonnées. *Cancore* n'est pas le seul profil d'application de *LOM*. D'autres existent tels que *ManUeL*, *Normetic*, etc. (Passardière et Jarraud, 2004). Malgré l'utilisation de profils d'application, les concepteurs de matériel pédagogique font souvent appel à un ou plusieurs spécialistes pour remplir les différentes métadonnées (Passardière et Jarraud, 2004).

3.3.3 Étape 3 : L'agrégation d'objets d'apprentissage

La troisième étape, l'agrégation d'objets d'apprentissage, est optionnelle dans le cycle de vie des objets d'apprentissage. Cette étape consiste à réunir plusieurs objets en un paquetage pour en faciliter la distribution et la réutilisation(Duncan, 2003). À titre d'exemple,

un professeur pourrait grouper l'ensemble des objets nécessaires à une activité pédagogique en un paquetage afin de les distribuer. Puisqu'un paquetage est aussi une unité d'information, si un concepteur ajoute les métadonnées appropriées, l'agrégat sera un objet d'apprentissage. Une spécification d'agrégation populaire est *IMS-CP* (IMS Global Learning Consortium, 2005a) pour laquelle un paquetage est un fichier au format *zip* qui contient tous les objets en plus d'un fichier *imsmanifest.xml* qui agit comme une table des matières.

3.3.4 Étape 4 : L'intégration des objets dans un scénario pédagogique

La quatrième étape est aussi optionnelle. Elle consiste à déterminer l'ordre de présentation (*sequencing*) des objets d'apprentissage. Quoiqu'un auteur puisse établir un enchaînement statique d'objets d'apprentissage, des spécifications permettent de modéliser des scénarios pédagogiques complexes. D'une façon générale, un scénario pédagogique définit une activité d'apprentissage par un ensemble de contraintes qui régissent l'ordre de présentation des objets d'apprentissage en fonction de résultats et d'événements intermédiaires, et précise les rôles des différents intervenants. La figure 3.1 présente un exemple adapté d' IMS Global Learning Consortium (2005b) pour une activité pédagogique de simulation du traité de Versailles. En réalité, les concepteurs pédagogiques ne modélisent pas les scénarios sous forme textuelle, comme dans cet exemple, mais avec des *langages de modélisation éducationnels*. Rawlings et al. (2002) propose une comparaison approfondie des principaux langages de modélisation éducationnels. La plupart des langages cités dans cette étude se limitent à des domaines particuliers (par exemple, *TML* se concentre sur la modélisation de scénarios avec des banques de questions). Le langage le plus général est *IMS-LD* (IMS Global Learning Consortium, 2005b). Sa description suit.

1. Le professeur expose le déroulement de l'activité pédagogique aux élèves.
2. Le professeur présente brièvement les pays signataires du traité de Versailles.
3. Chaque élève choisit un pays.
4. Les élèves, ayant choisi le même pays, forment un groupe.
5. Une fois tous les groupes formés, le professeur distribue à chacun des élèves des livres décrivant la situation de leur pays en 1919.
6. Chaque équipe développe une stratégie de négociation.
7. Chaque équipe élit un chef.
8. Le chef de chaque équipe rencontre les chefs des autres équipes pour établir un calendrier de rencontres de négociation.
9. Les équipes se rencontrent pour les différentes négociations. Celles-ci peuvent être bilatérales ou multilatérales.
10. Pendant les négociations, le professeur se promène d'un emplacement à l'autre pour donner des conseils.
11. Une fois la phase de négociation terminée, chaque équipe rédige un rapport et le remet au professeur.

Figure 3.1 – Exemple de scénario pédagogique : la simulation du traité de Versailles

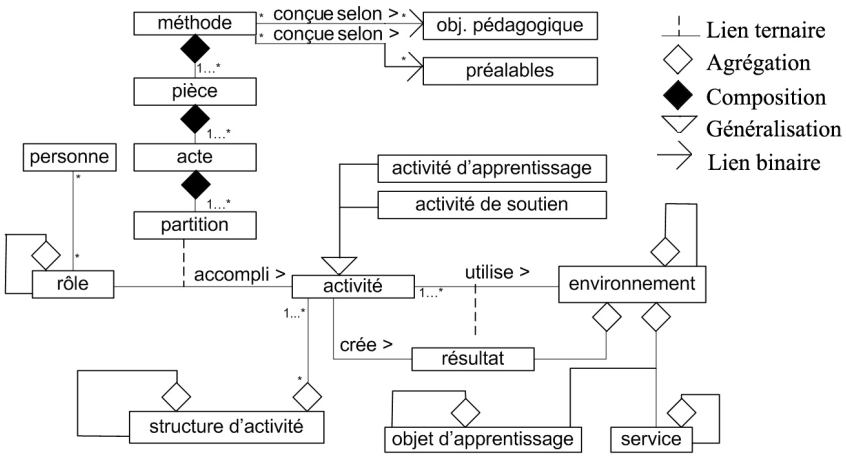

Figure 3.2 – Modèle conceptuel d'un scénario pédagogique selon *IMS-LD*

Le langage de modélisation éducationnel *IMS-LD*

IMS-LD recourt à une métaphore de théâtre : un scénario spécifie les acteurs, les pièces, les actes, le jeu associé à chaque rôle dans un acte, les services et les objets d'apprentissage. Pour reprendre l'exemple du traité de Versailles, mais adapté à la formation en ligne, les acteurs sont les élèves et les professeurs, les rôles sont ceux d'*élève*, d"*élève chef d'équipe* et de '*professeur*, les objets d'apprentissage utilisés sont des livres électroniques et les services utilisés sont un système de courriels et un logiciel de vidéoconférence.

Un scénario se compose de plusieurs pièces qui se déroulent en parallèle, une pièce contient des actes exécutés en séquences, et un acte contient des partitions qui correspondent à la tâche qu'un rôle doit effectuer pendant un acte.

Afin de mieux expliquer les relations entre les différentes notions, la figure 3.2 présente le modèle conceptuel simplifié d'*IMS-LD* (traduction d'une figure de l'IMS Global Learning Consortium (2005b)). Le schéma emploie la notation *UML*. Tel qu'illustré sur cette figure, une personne peut occuper plusieurs rôles et plusieurs personnes tiennent parfois un même rôle. Par exemple, un apprenant peut assurer le rôle de celui qui distribue les livres et de celui qui réserve les locaux, tout en partageant ces rôles avec un second apprenant.

Une *méthode* correspond grossièrement à un scénario, conçu selon un objectif pédagogique et des préalables. Une *méthode* comporte des pièces, composées d'actes, lesquels se décomposent à leur tour en partitions. Une partition définit une paire (rôle, activité) qui assigne à un rôle une tâche à effectuer pour cette partition. Les activités se divisent en deux familles : les *activités de soutien*, qui visent à faciliter l'apprentissage des apprenants et les *activités d'apprentissage*, celles-là mêmes des apprenants. Au cours d'une activité, les acteurs interagissent avec un environnement formé d'objets d'apprentissage ou de services. Le terme *services* dénote, comme énoncé plus tôt, des outils tels qu'un forum de discussion, un site Web de clavardage, un outil de vidéoconférence et un service de courriels.

Ceci complète la description de la spécification de base d'*IMS-LD* (le niveau A). *IMS-LD* comprend deux autres niveaux (niveau B et C) qui ajoutent la prise en compte du profil de l'apprenant et la gestion des évènements, respectivement. Le niveau B ajoute les éléments *propriété* et *condition*. Le premier est un attribut d'acteur, tel que son niveau de connaissance d'une matière, son âge, etc. Le second est une condition pour effectuer un acte ou un scénario. Les conditions et les propriétés permettent de définir, par exemple, que si un apprenant possède moins de 18 ans, alors il devra rejoindre le groupe des enfants, plutôt que celui des adultes. Le niveau C introduit le concept de messages entre les rôles et les composants du système. Cela permet la création de parcours pédagogiques dirigés par des événements. Il joint l'élément *notification* qui permet à un résultat d'activité de déclencher une activité d'une autre partition. Par exemple, dès que la moitié d'une classe a terminé un exercice, le professeur pourrait donner la solution sur le tableau noir.

En résumé, la force d'*IMS-LD* repose dans son indépendance à l'égard de tous les domaines : il peut représenter toutes les approches d'acquisition des connaissances (constructivisme, instructionnisme, cognitivisme, etc.) et autant pour ce qui est des activités individuelles ou collaboratives qu'à distance ou non.

3.3.5 Étape 5 : La livraison des objets d'apprentissage aux apprenants

La dernière étape est la livraison des objets d'apprentissage. Cette section n'aborde pas les façons triviales d'utiliser les objets d'apprentissage, comme inclure des objets dans une simple page Web. Les acteurs dans le domaine de la formation en ligne emploient généralement le terme *Learning Management System* (*LMS*) pour désigner les systèmes qui présentent les objets d'apprentissage aux apprenants. Un *LMS* se compose d'un ensemble de logiciels ou d'un environnement Web avec lequel se déroulent des activités d'apprentissage incorporant des objets d'apprentissage (Simic et al., 2004). Chaque *LMS* se conforme à un certain nombre de standards. Habituellement les *LMS* proposent aussi une gamme complète de logiciels de gestion de comptes, de développement de cours, d'administration, etc. et offrent en plus une compatibilité avec des langages de modélisation pédagogique. De nombreux *LMS* commerciaux et non commerciaux existent : *WebCT*[4], *Lotus Learning Management System*[5], *Blackboard Learning Management System*[6], *Stellar*[7], etc.

Les principales différences entre les *LMS* et les *STI*

Les *LMS* se distinguent des *STI* de plusieurs façons. Les *LMS* ne possèdent pas de module « modèle de l'apprenant » ou seulement un module très superficiel, souvent établi à partir de tests ou d'une comptabilisation du temps passé sur chaque page (Simic et al., 2004). Ceci contraste avec les modèles cognitifs élaborés de certains *STI* (Anderson et al., 1995). Puisque les *LMS* adoptent des modèles de l'usager restreints, ces systèmes s'adaptent peu à chaque apprenant (Simic et al., 2004; Morales et Aguera, 2002). En général, l'adaptation la plus considérable d'un *LMS* consiste à construire des séquences d'objets d'apprentissage dynamiquement (Morales et Aguera, 2002), ce qui reste très limité comparativement

[4] *WebCT*, http://www.webct.com/
[5] *Lotus Learning Management System*, http://lotus.com/lotus/offering6.nsf/wdocs/homepage
[6] *Blackboard Learning Management System*, http://www.blackboard.com/products/academic/ls/index.htm
[7] *Stellar*, http://stellar.mit.edu

aux tuteurs virtuels des *STI* qui tentent de suppléer de véritables enseignants. En effet, un tuteur virtuel doit (1) planifier et adapter un programme d'études (une séquence appropriée d'activités), (2) répondre aux questions de l'apprenant portant sur les objectifs des activités et sur le contenu pédagogique, (3) déterminer le meilleur moment pour intervenir et (4) donner les conseils les plus appropriés en fonction de l'apprenant et du contexte (Burns et Capps, 1988; Wenger, 1987). La faible personnalisation des *LMS* s'explique par le fait que les *LMS* enseignent souvent des cours complets, alors que les *STI* ne proposent d'ordinaire que de courtes activités qui correspondent davantage à la mise en pratique d'une théorie acquise en classe (Reiser et al., 1992; Lesgold et al., 1992; Corbett et Anderson, 1992). La portée restreinte des *STI* leur permet d'offrir des tuteurs intelligents dispensant une instruction hautement individualisée. C'est pourquoi les *LMS* accordent généralement une forte importance à l'intervention de formateurs humains (Hotte et Leroux, 2003), à l'antipode des *STI* qui ne proposent généralement, que des activités entre un apprenant et le logiciel (Wenger, 1987).

3.4 L'universalisation du concept d'objet d'apprentissage et une opportunité pour les *STI*

Cette section conclut le chapitre en présentant quelques préoccupations actuelles des chercheurs qui souhaitent rendre commune la notion d'objet d'apprentissage, pour se terminer en décrivant une opportunité pour les *STI*.

3.4.1 Les facteurs critiques à l'universalisation du concept d'objet d'apprentissage

Wiley (2004) et Friesen (2004) recensent quatre facteurs critiques à l'universalisation de la notion d'objet d'apprentissage : (1) la facilité d'adaptation du contenu existant,

(2) la prise en compte des facteurs d'adoption d'une nouvelle technologie, (3) la gestion efficace des droits d'auteur et (4) le rôle de l'industrie militaire. Premièrement, Wiley (2004) énonce qu'il importe de simplifier les processus d'adaptation du contenu existant, puisqu'une quantité considérable de documents pédagogiques nécessitent une adaptation pour être conforme aux standards d'objets d'apprentissage. Deuxièmement, trois facteurs favorisent l'adoption des nouvelles technologies : la simplicité, la compatibilité avec les méthodes actuelles et l'avantage relatif par rapport à celles-ci. Or, dans le cas des objets d'apprentissage, ces caractéristiques ne semblent pas toujours évidentes pour les utilisateurs potentiels (Friesen, 2004). Les nouvelles techniques et outils nécessitent un effort de familiarisation. Par exemple, comme il a été mentionné, la difficulté de trouver les métadonnées adéquates peut décourager certains utilisateurs. Troisièmement, l'inclusion d'un système efficace de gestion des droits d'auteurs pour empêcher la copie illégale des objets d'apprentissage s'avère essentielle, puisque tout document en format électronique se reproduit facilement (Wiley, 2004). Actuellement, nombre de chansons, de logiciels et de films piratés circulent illégalement sur Internet. D'ailleurs, des pirates ont déjoué plusieurs systèmes élaborés de protection tels que la protection *CSS*[8] qui protège la plupart des disques *DVD*[9]. Quatrièmement, Friesen (2004) exprime la crainte que l'industrie militaire impose sa vision, puisqu'elle investit une énorme quantité d'argent dans l'apprentissage en ligne pour le domaine militaire.

3.4.2 Une opportunité pour les *STI*

Le concept d'objet d'apprentissage a émergé du domaine de la formation en ligne, dans lequel les *LMS* l'emploient. Toutefois, il s'applique théoriquement à d'autres types de systèmes manipulant des connaissances à vocation pédagogique. Une idée novatrice et originale serait d'en appliquer les principes au domaine des *STI*. Les *STI* génériques (tel qu'*ASTUS*) gagneraient beaucoup à accroître la réutilisabilité des connaissances entre

[8] CSS (Contents Scramble System) : http://www.dvdcca.org/
[9] DVD (Digital Versatile Disc)

plusieurs laboratoires virtuels. Par exemple, un concepteur pédagogique profiterait à réutiliser des connaissances de mathématiques, dans un laboratoire virtuel sur la synthèse d'image.

Les *STI* avec des modèles cognitifs utilisent des éléments de connaissance bien différents de ceux des *LMS*. Les *LMS* manipulent comme unités d'information des documents électroniques (pages Web, images, etc.), alors que les *STI* cognitifs travaillent avec des connaissances procédurales et des connaissances sémantiques reliées intimement entre elles. Adapter la notion d'objets d'apprentissage à un *STI* cognitif, nécessite de définir une façon de spécifier des objectifs pédagogiques en terme d'ensembles de ces connaissances.

Le modèle *UV* propose une façon de formaliser les objectifs pédagogiques en terme des buts (cf. section 2.2.10). Néanmoins, la notion d'héritage nuit à la formation d'ensembles de connaissances réutilisables (cf. section 2.4.5) car (1) l'ajout d'un concept requiert son intégration dans une hiérarchie et (2) la réutilisation d'un concept pour un autre laboratoire virtuel demande l'extraction de ce concept d'une hiérarchie.

Le chapitre 5 décrira un modèle hybride original combinant les objets d'apprentissage, *UV* et les logiques de description (présentées au chapitre 4). Le remplacement de l'intégration de la logique d'*UV* par l'usage de la logique de description permettra de substituer la relation d'héritage explicite d'*UV* par une taxinomie qui peut être inférée dynamiquement à partir d'un ensemble d'axiomes logiques expressifs qui se retrouvent avec les connaissances. Le modèle proposera également une façon différente de structurer les connaissances en plusieurs fichiers, etc. La combinaison de ces améliorations facilitera le groupement des connaissances pour former des objets d'apprentissage. Le chapitre 5 en donne les raisons.

*LOM*pour les métadonnées.

CHAPITRE 4

Les logiques de description

Ce chapitre présente les *logiques de description* (*LD*), une famille de langages de représentation de connaissances qui exploitent, en général, des sous-ensembles décidables[1] de la logique de premier ordre. Ils ont été largement étudiés et utilisés dans plusieurs systèmes à base de connaissances (Nardi et Brachman, 2003).

Ce chapitre aborde les *LD* en vue de la proposition d'un modèle hybride de représentation des connaissances décrit au chapitre suivant, qui combine à la fois les *LD*, *UV* et la notion d'objet d'apprentissage. Le modèle proposé remplace l'intégration de la logique d'*UV* par un usage des *LD*, duquel s'ensuivent plusieurs avantages, exposés dans le chapitre 5.

Cinq sections forment ce chapitre : la section 4.1 met en contexte les *LD*, la section 4.2 présente les notions de base des *LD* avec un exemple, la section 4.3 décrit la sémantique formelle de la logique minimale $\mathcal{AL}$ et de ses extensions les plus courantes, la section 4.4 aborde les services d'inférence usuels pour les *LD* et compare les principaux moteurs d'inférence alors que la section 4.5 clôt le chapitre avec une discussion sur l'applicabilité aux systèmes tutoriels intelligents.

[1] Pour une logique, un problème de raisonnement est *décidable* si une machine de Turing peut le résoudre en un nombre fini d'étapes.

4.1 La mise en contexte

L'accent au sein des *LD* est mis sur les services de raisonnement et plus particulièrement sur ceux pour la prise de décision : l'objectif principal des *LD* consiste à pouvoir raisonner efficacement pour minimiser les temps de réponse. Par conséquent, la communauté scientifique a publié de nombreuses recherches qui portent sur l'étude du rapport expressivité/performance des différentes *LD* (Nardi et Brachman, 2003). La principale qualité des *LD* réside dans leurs algorithmes d'inférence dont la complexité est souvent inférieure aux complexités des démonstrateurs de preuves de la logique de premier ordre (Tsarkov et Horrocks, 2003).

4.1.1 Une approche ontologique

Les *LD* utilisent une approche ontologique, c'est-à-dire que pour décrire les *individus* d'un domaine, elles requièrent tout d'abord la définition des catégories générales d'individus et les relations logiques que les individus ou catégories peuvent entretenir entre eux. Cette approche ontologique est naturelle pour le raisonnement puisque même si la majorité des interactions se déroulent au niveau des individus, la plus grande partie du raisonnement se produit au niveau des catégories (Russell et Norvig, 2002).

4.1.2 Un historique des *LD*

Le développement des *LD* fut fortement influencé par les travaux sur la logique des prédicats, les schémas (*frames*) (Minsky, 1981) et les réseaux sémantiques. Des correspondances existent entre les *LD* et ces formalismes (Sattler et al., 2003; Baader et Nutt, 2003). La présence de catégories générales d'objets et de relations fait d'ailleurs partie de l'héritage conceptuel des schémas et des réseaux sémantiques.

La première génération de *LD* (1980 - 1990)

Les premiers travaux sur les *LD* commencèrent au début des années 1980 avec des systèmes à base de connaissances tels que KL-ONE, BACK et LOOM (Baader et al., 2003; Nardi et Brachman, 2003). Ces premières implantations résolvent des problèmes d'inférence en temps souvent polynomial, par le biais d'une catégorie d'algorithmes de vérification de subsomption de type normalisation/comparaison (*structural subsumption algorithms*). Ces algorithmes ne s'appliquent qu'à des *LD* peu expressives, sans quoi ils sont incomplets, c'est-à-dire qu'ils sont incapables de prouver certaines formules vraies.

La deuxième génération de *LD* (1990 - aujourd'hui)

Dans les années 1990, une nouvelle classe d'algorithmes est apparue : les algorithmes de vérification de satisfiabilité à base de tableaux (*tableau-based algorithms*). Ces derniers raisonnent sur des *LD* dites *expressives* ou *très expressives*, mais en temps exponentiel. Cependant, en pratique, le comportement des algorithmes est souvent acceptable (Baader et al., 2003). L'expressivité accrue a ouvert la porte à de nouvelles applications telles que le Web sémantique (Baader et al., 2003; Zou et al., 2004; Horrocks et al., 2003). Le terme *logiques de description expressives* (*LDE*) désigne l'ensemble des *LD* qui ont émergé pendant cette période.

4.2 Les deux niveaux de description

La modélisation des connaissances d'un domaine avec les *LD* se réalise en deux niveaux. Le premier, le niveau terminologique ou *TBox*, décrit les connaissances générales d'un domaine alors que le second, le niveau factuel ou *ABox*, représente une configuration précise. Une *TBox* comprend la définition des concepts et des rôles, alors qu'une *ABox* décrit les individus en les nommant et en spécifiant en terme de concepts et de rôles, des asser-

TBox	*ABox*
$Femelle \sqsubseteq \top \sqcap \neg M\hat{a}le$	$Humain(Anne)$
$M\hat{a}le \sqsubseteq \top \sqcap \neg Femelle$	$Femelle(Anne)$
$Animal \equiv M\hat{a}le \sqcup Femelle$	$Femme(Sophie)$
$Humain \sqsubseteq Animal$	$Humain(Robert)$
$Femme \equiv Humain \sqcap Femelle$	$\neg Femelle(Robert)$
$Homme \equiv Humain \sqcap \neg Femelle$	$Homme(David)$
$M\grave{e}re \equiv Femme \sqcap \exists relationParentEnfant$	$relationParentEnfant(Sophie, Anne)$
$P\grave{e}re \equiv Homme \sqcap \exists relationParentEnfant$	$relationParentEnfant(Robert, David)$
$M\grave{e}reSansFille \equiv M\grave{e}re \sqcap$	
$\forall relationParentEnfant.\neg Femme$	
$relationParentEnfant \sqsubseteq \top_R$	

Tableau 4.1 – Une base de connaissances composée d'une *TBox* et d'une *ABox*

tions qui portent sur ces individus nommés. Plusieurs *ABox* peuvent être associés à une même *TBox* ; chacune représente une configuration constituée d'individus, et utilise les concepts et rôles de la *TBox* pour l'exprimer.

4.2.1 Le niveau terminologique (*TBox*)

Le coté gauche du tableau 4.1 présente un exemple de *TBox*. Les prochains paragraphes explicitent divers aspects des *TBox* en se référant à cet exemple.

Les entités atomiques

Les *concepts atomiques* et *rôles atomiques* constituent les entités élémentaires d'une *TBox*. Les noms débutant par une lettre majuscule désignent les concepts, alors que ceux débutant par une lettre minuscule dénomment les rôles (par exemple : les concepts $Femelle$, $M\hat{a}le$, $Homme$ et $Femme$, et le rôle $relationParentEnfant$).

Les concepts et rôles atomiques prédéfinis

Les *LD* prédéfinissent minimalement quatre concepts atomiques : le concept $\top$ et le rôle $\top_R$, les plus généraux de leur catégorie respective, et le concept $\bot$ ainsi que le rôle $\bot_R$, les plus spécifiques (c'est-à-dire l'ensemble vide).

Les entités composées

Les concepts et rôles atomiques peuvent être combinés au moyen de *constructeurs* pour former respectivement des *concepts* et des *rôles composés*. Par exemple, le concept composé $Mâle \sqcap Femelle$ résulte de l'application du constructeur $\sqcap$ aux concepts atomiques $Mâle$ et $Femelle$. Le concept $Mâle \sqcap Femelle$ s'interprête comme l'ensemble des individus qui appartiennent aux concepts $Mâle$ et $Femelle$. Les différentes *LD* se distinguent par les constructeurs qu'elles proposent. Plus les *LD* sont expressives, plus les chances sont grandes que les problèmes d'inférence soient non décidables ou de complexité très élevée. Par contre, les *LD* trop peu expressives démontrent une inaptitude à représenter des domaines complexes.

La notation

La suite de ce mémoire adopte la notation suivante : R dénote un rôle, C, D des concepts composés et A, B des concepts atomiques.

La notion d'interprétation

Expliciter formellement la sémantique d'une *TBox*, requiert de définir préalablement la notion d'*interprétation*. Une interprétation $\mathcal{I}$ se compose d'un *domaine d'interprétation* $\triangle^{\mathcal{I}}$ et d'une *fonction d'interprétation* $\cdot^{\mathcal{I}}$. Le domaine d'interprétation consiste en un ensemble d'individus. La fonction d'interprétation assigne à chaque concept atomique A un ensemble tel que $A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$, et à chaque rôle atomique R une relation binaire

$R^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$.

La définition formelle de *TBox*

Une *TBox* contient des *axiomes terminologiques* de la forme $C \equiv D$ ou $C \sqsubseteq D$. La première sert à énoncer des relations d'équivalence entre concepts, alors que la seconde permet d'exprimer des relations d'inclusion. Une interprétation $\mathcal{I}$ satisfait un axiome $C \equiv D$ si et seulement si $C^{\mathcal{I}} = D^{\mathcal{I}}$. Une interprétation $\mathcal{I}$ satisfait un axiome $C \sqsubseteq D$ si et seulement si $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

Une interprétation satisfait une *TBox* (est un *modèle* de *TBox*) si et seulement si l'interprétation satisfait tous les axiomes de la *TBox*.

La section 4.3 décrit la sémantique des différents constructeurs utilisés dans la *TBox* de l'exemple.

4.2.2 Le niveau factuel (*ABox*)

Une *ABox* contient un ensemble d'assertions sur les individus : (1) des *assertions d'appartenance* et (2) des *assertions de rôle*. Chaque *ABox* doit être associée à une *TBox*, car les assertions s'expriment en terme des concepts et des rôles de la *TBox*. La partie droite du tableau 4.1 illustre un exemple d'*ABox*.

Une *ABox* désigne des individus dans ses assertions par des noms qu'elle leur donne. Ce mémoire utilise le terme *individu nommé* (*nominal* ou *individual name*, en anglais) pour référer à ces noms. L'exemple du tableau 4.1 comprend les individus nommés suivants : *Anne*, *David*, *Robert* et *Sophie*. La suite de ce mémoire représente par les lettres a, b les individus nommés. Une fonction d'interprétation assigne à chacun de ces noms a, un individu $a^{\mathcal{I}}$ tel que $a^{\mathcal{I}} \in \triangle^{\mathcal{I}}$. Les moteurs d'inférence pour *LD* adoptent généralement l'*hypothèse de noms uniques* (HNU), c'est-à-dire que pour tout individu nommé a et b, $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ (Baader et Nutt, 2003).

$$
\begin{array}{lll}
C, D \longrightarrow & A & \text{(concept atomique)} \\
& |\top & \text{(le concept universel)} \\
& |\ \bot & \text{(le concept le plus spécifique)} \\
& |\neg A & \text{(la négation atomique)} \\
& |C \sqcap D & \text{(l'intersection)} \\
& |\exists R.\top & \text{(quantification existentielle limitée)} \\
& |\forall R.C & \text{(quantification universelle complète)}
\end{array}
$$

Figure 4.1 – La grammaire des expressions conceptuelles selon $\mathcal{AL}$

Chaque assertion d'appartenance d'une *ABox* (notée $C(a)$), déclare que pour cette *ABox*, il existe un individu nommé a, membre du concept C de la *TBox* associée. Une interprétation satisfait une assertion d'appartenance $C(a)$ si et seulement si $a^{\mathcal{I}} \in C^{\mathcal{I}}$.

Une assertion de rôle, de la forme $R(a, b)$ indique que pour cette *ABox*, il existe un individu nommé a qui est en relation avec un individu nommé b par le rôle R (défini dans la *TBox* associée), tel que a fait partie du domaine de R et b fait partie de l'image. Une interprétation satisfait une assertion de rôle $R(a, b)$ si et seulement si $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

Finalement, une interprétation $\mathcal{I}$ satisfait une *ABox* $\mathcal{A}$ (est un *modèle* d'*ABox*) si et seulement si $\mathcal{I}$ satisfait toutes les assertions de $\mathcal{A}$.

4.3 La logique minimale $\mathcal{AL}$

Pour des fins de simplicité, ce chapitre décrit en premier lieu une logique minimale nommée $\mathcal{AL}$ qui a été introduite par Schmidt-Schaub et Smolka (1991) et qui revêt d'une grande importance dans le domaine. Cette logique est minimale, dans le sens où une logique moins expressive représente peu d'intérêt (Baader et Nutt, 2003).

4.3.1 Les constructeurs d'$\mathcal{AL}$

La figure 4.1 illustre les constructeurs offerts par $\mathcal{AL}$ pour l'édification de concepts composés.

Le constructeur $C \sqcap D$ permet de faire la conjonction de deux concepts composés, ce qui représente l'ensemble des individus, membres à la fois du concept C et du concept D pour une interprétation. Le constructeur $\neg A$ est utilisé pour évoquer la négation d'un concept atomique, c'est-à-dire les individus pour une interprétation qui n'appartiennent pas au concept atomique A.

Le quantificateur existentiel non typé $\exists R.\top$ désigne l'ensemble des individus, membres du domaine d'un rôle R pour une interprétation donnée. Par exemple, pour une interprétation $\mathcal{I}$ qui est un modèle de l'*ABox* et de la *TBox* du tableau 4.1, $\exists relationParentEnfant.\top$ équivaut l'ensemble des individus $\{Sophie^{\mathcal{I}}, Robert^{\mathcal{I}}\}$.

Le quantificateur universel $\forall R.C$ évoque l'ensemble des individus du domaine d'un rôle R qui sont en relation, par le biais de R, avec un individu du concept C, pour une interprétation donnée. Par exemple, pour une interprétation $\mathcal{I}$ qui est un modèle de l'*ABox* et de la *TBox* du tableau 4.1, $\forall relationParentEnfant.Homme$ équivaut l'ensemble des individus $\{Robert^{\mathcal{I}}\}$.

$\mathcal{AL}$ ne permet pas la spécification de rôles à l'aide de constructeurs (rôles composés). La sous-section qui suit décrit la sémantique formelle d'$\mathcal{AL}$.

4.3.2 La sémantique formelle d'$\mathcal{AL}$

Afin de supporter la notion de *concepts composés*, la fonction d'interprétation est étendue par les règles décrites à la figure 4.2. Deux concepts C et D d'une *TBox* $\mathcal{T}$ s'*équivalent* si et seulement si $C^{\mathcal{I}} = D^{\mathcal{I}}$ pour toute interprétation $\mathcal{I}$ modèle de $\mathcal{T}$.

$$
\begin{array}{lcl}
\top^{\mathcal{I}} & = & \triangle^{\mathcal{I}} \\
\bot^{\mathcal{I}} & = & \emptyset \\
(\neg A)^{\mathcal{I}} & = & \triangle^{\mathcal{I}} \backslash A^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} & = & \{a \in \triangle^{\mathcal{I}} | \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\exists R.\top)^{\mathcal{I}} & = & \{a \in \triangle^{\mathcal{I}} | \exists b.(a,b) \in R^{\mathcal{I}}\}
\end{array}
$$

Figure 4.2 – La sémantique formelle d'$\mathcal{AL}$

L'interprétation selon l'hypothèse du monde ouvert

Dans les *LD*, l'interprétation des connaissances respecte l'*hypothèse du monde ouvert*. Contrairement à la sémantique des bases de données traditionnelles, l'absence d'information représente l'ignorance plutôt qu'une information négative. Pour enrichir la comparaison avec les bases de données, répondre à une requête pour un système bâti sur les *LD* nécessite d'effectuer un raisonnement logique souvent plus complexe qu'une simple recherche pour vérifier la présence d'une information, car le système doit souvent considérer plusieurs interprétations possibles, une conséquence de l'hypothèse du monde ouvert (Baader et al., 2003).

La correspondance entre $\mathcal{AL}$ et la logique des prédicats

Une correspondance existe entre la logique $\mathcal{AL}$ et la logique de premier ordre (Baader et Nutt, 2003). Un concept atomique A correspond à un prédicat unaire $\phi_A(x)$, un rôle R à un prédicat binaire $\phi_R(x,y)$, un individu correspond à une constante et un concept composé à une formule à une variable libre $\phi_C(x)$. La figure 4.3 illustre la façon de traduire les constructeurs d'$\mathcal{AL}$ en leur équivalent en logique des prédicats.

$$
\begin{array}{lcl}
\phi_{\neg C}(x) & = & \neg\phi_C(x) \\
\phi_{C\sqcap D}(x) & = & \phi_C(x) \wedge \phi_D(x) \\
\phi_{C\sqcup D}(x) & = & \phi_C(x) \vee \phi_D(x) \\
\phi_{\exists R.C}(y) & = & \exists x.R(y,x) \wedge \phi_C(x) \\
\phi_{\forall R.C}(y) & = & \forall x.R(y,x) \rightarrow \phi_C(x)
\end{array}
$$

Figure 4.3 – Correspondance entre les constructeurs d'$\mathcal{AL}$ et la logique des prédicats

4.3.3 Les extensions d'$\mathcal{AL}$

Il existe trois façons proéminentes d'étendre $\mathcal{AL}$: (1) ajouter des constructeurs de concepts, (2) ajouter des constructeurs de rôles et (3) énoncer des contraintes sur l'interprétation des rôles (Baader, 2003).

L'extension par ajout de constructeurs de concepts ou de rôles

Le tableau 4.2 illustre des exemples de constructeurs pour augmenter $\mathcal{AL}$ (Baader, 2003). La première colonne contient la lettre qui désigne le constructeur, la deuxième sa syntaxe d'utilisation et la dernière sa sémantique. La nomenclature des *LD* dicte que pour chaque constructeur ajouté, il faut agglutiner la lettre correspondante au nom de la logique originale. Par exemple, la logique $\mathcal{AL}$, enrichie de l'union ($\mathcal{U}$) et de la quantification existentielle complète ($\mathcal{E}$), se nomme $\mathcal{ALUE}$. Il faut noter que l'appelation $\mathcal{ALC}$ équivaut à $\mathcal{ALUE}$, puisque l'union et la quantification existentielle complète s'expriment par la négation complète et inversement, car $C \sqcup D \equiv \neg(\neg C \sqcap \neg D)$ et $\exists R.C \equiv \neg\forall R.\neg C$ (Baader et Nutt, 2003).

Le constructeur $\mathcal{O}$ permet la description de concepts par l'énumération d'individus nommés, $\mathcal{U}$ désigne l'union de concepts arbitraires, $\mathcal{E}$ la quantification existentielle complète, $\mathcal{C}$ la négation complète, $\mathcal{I}$ les rôles inverses et $\mathcal{H}$ l'inclusion entre rôles. Les constructeurs $\mathcal{F}$, $\mathcal{Q}$ et $\mathcal{N}$ sont trois variantes de la contrainte de cardinalité sur rôle.

$[\mathcal{O}]$	$\{a_1, a_2, ..., a_n\}$	$\{a_1^{\mathcal{I}}, a_2^{\mathcal{I}}, ..., a_n^{\mathcal{I}}\}$
$[\mathcal{U}]$	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$[\mathcal{E}]$	$\exists R.C$	$\{a \in \triangle^{\mathcal{I}} \mid \{\exists b.(a,b) \in R^{\mathcal{I}}\} \wedge b \in C^{\mathcal{I}}\}$
$[\mathcal{C}]$	$\neg C$	$\triangle^{\mathcal{I}} \backslash C^{\mathcal{I}}$
$[\mathcal{I}]$	R_1^{-1}	$\{(y,x) \mid (x,y) \in R_1^{\mathcal{I}}\}$
$[\mathcal{H}]$	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
$[\mathcal{F}]$	$= 1R$	$\{x \in \triangle^{\mathcal{I}} \mid \mid \{y \in \triangle^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}\}\mid = 1\}$
	$\geq 2R$	$\{x \in \triangle^{\mathcal{I}} \mid \mid \{y \in \triangle^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}\}\mid \geq 2\}$
$[\mathcal{N}]$	$\geq nR$	$\{a,b \in \triangle^{\mathcal{I}} \mid \mid (a,b) \in R^{\mathcal{I}}\}\mid \geq n\}$
	$\leq nR$	$\{a,b \in \triangle^{\mathcal{I}} \mid \mid (a,b) \in R^{\mathcal{I}}\}\mid \leq n\}$
	$= nR$	$\{a,b \in \triangle^{\mathcal{I}} \mid \mid (a,b) \in R^{\mathcal{I}}\}\mid = n\}$
$[\mathcal{Q}]$	$\geq nR.C$	$\{a,b \in \triangle^{\mathcal{I}} \mid \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\mid \geq n\}$
	$\leq nR.C$	$\{a,b \in \triangle^{\mathcal{I}} \mid \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\mid \leq n\}$
	$= nR.C$	$\{a,b \in \triangle^{\mathcal{I}} \mid \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}\mid = n\}$

Tableau 4.2 – Exemple de constructeurs de rôles et concepts pour étendre $\mathcal{AL}$

L'extension par ajout de contraintes sur l'interprétation des rôles

La spécification d'un ensemble de rôles transitifs $\mathcal{N}_{\mathcal{R}+}$, constitue une extension par ajout de contraintes sur l'interprétation des rôles (désignée par la lettre $_{\mathcal{R}+}$), qui permet l'expression de rôles transitifs tels qu'*ancêtreDe* ou *frèreDe* (Baader, 2003). La lettre $\mathcal{S}$ désigne la logique $\mathcal{ALC}$ additionnée de $_{\mathcal{R}+}$.

L'extension des types primitifs $(\mathcal{D})$

Une dernière extension, symbolisée par la lettre $(\mathcal{D})$, ajoute le support des types primitifs (Horrocks et al., 2003). Cette extension augmente $\mathcal{AL}$ d'un second domaine d'interprétation $\triangle_D^{\mathcal{I}}$ disjoint avec $\triangle^{\mathcal{I}}$ et qui représente l'ensemble des valeurs de type primitif. Le domaine $\triangle_D^{\mathcal{I}}$ définit plusieurs sous-domaines tels que les entiers, les chaînes de caractères, les entiers positifs, etc. Les éléments de ces domaines se nomment *individus primitifs*.

De plus, l'extension ajoute un nouveau type de rôle, défini comme une relation binaire sur $\triangle^{\mathcal{I}} \times \triangle_D^{\mathcal{I}}$ et appelé *rôles à valeurs primitives*. La lettre U représente l'ensemble de ces

rôles. Cet ajout autorise la spécification d'assertions de rôle telles que $u(a, 205006007)$ et $v(a, "Jean\ Jacques")$ où $u, v \in U$.

Quoiqu'il soit possible de définir un constructeur de hiérarchies pour ces rôles (par exemple : $u \sqsubseteq v$ avec la sémantique $u^{\mathcal{I}} \subseteq v^{\mathcal{I}}$), plusieurs constructeurs pour rôle classique tels que le constructeur de rôle transitif et le constructeur de rôle inverse n'ont pas leur équivalent pour les rôles à valeurs primitives.

4.4 L'inférence

L'inférence s'effectue au niveau terminologique ou factuel. Les sections 4.4.1 et 4.4.2 abordent le raisonnement au niveau terminologique et factuel, respectivement. Pour terminer, la section 4.4.3 présente un tableau comparatif des différents moteurs d'inférence.

4.4.1 L'inférence au niveau terminologique

Quatre principaux problèmes d'inférence se présentent au niveau terminologique (Baader et Nutt, 2003) :

- *Satisfiabilité* : Un concept C d'une terminologie $\mathcal{T}$ est satisfiable si et seulement s'il existe un modèle $\mathcal{I}$ de $\mathcal{T}$ tel que $C^{\mathcal{I}} \neq \emptyset$.
- *Subsomption* : Un concept C est subsumé par un concept D pour une terminologie $\mathcal{T}$ si et seulement si $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ pour tout modèle $\mathcal{I}$ de $\mathcal{T}$.
- *Équivalence* : Un concept C est équivalent à un concept D pour une terminologie $\mathcal{T}$ si et seulement si $C^{\mathcal{I}} = D^{\mathcal{I}}$ pour chaque modèle $\mathcal{I}$ de $\mathcal{T}$.
- *Disjonction (disjointness)* : Des concepts C et D sont disjoints par rapport à une terminologie $\mathcal{T}$ si et seulement si $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ pour chaque modèle $\mathcal{I}$ de $\mathcal{T}$.

C est insatisfiable	$\Longleftrightarrow$	C est subsumé par $\bot$
C et D sont équivalents	$\Longleftrightarrow$	C est subsumé par D, et D par C
C et D sont disjoints	$\Longleftrightarrow$	$C \sqcap D$ est subsumé par $\bot$

Figure 4.4 – Réduction des problèmes d'inférence d'une *TBox* à des problèmes de subsomption

C est subsumé par D	$\Longleftrightarrow$	$C \sqcap \neg D$ est insatisfiable
C et D sont équivalents	$\Longleftrightarrow$	$C \sqcap \neg D$ et $\neg C \sqcap D$ sont insatisfiables
C et D sont disjoints	$\Longleftrightarrow$	$C \sqcap D$ est insatisfiable

Figure 4.5 – Réduction des problèmes d'inférence d'une *TBox* à des problèmes de satisfiabilité

Les moteurs d'inférence actuels tirent généralement profit du fait que les quatre types de problèmes d'inférence peuvent être réduits à des problèmes de subsomption ou à des problèmes de satisfiabilité. Les figures 4.4 et 4.5 illustrent cette propriété qui implique, que les moteurs d'inférence des *LD* ne nécessitent souvent qu'un seul algorithme d'inférence pour raisonner au niveau terminologique (Baader et Nutt, 2003). D'ailleurs, les deux grandes classes d'algorithmes de raisonnement pour les *LD* (*algorithmes de subsomption de type normalisation/comparaison* et *algorithmes de vérification de satisfiabilité à base de tableaux*) correspondent aux façons de réduire respectivement des problèmes d'inférence à des problèmes de subsomption et de satisfiabilité (Baader et al., 2003).

Complexité	Logiques de description
P	$\mathcal{AL}$, $\mathcal{ALN}$
NP	$\mathcal{ALE}$
PSpace	$\mathcal{ALC}$, $\mathcal{ALEN}$
ExpTime	$\mathcal{SHIQ}$, $\mathcal{SHOQ}$
NExpTime	...

Tableau 4.3 – Complexité de la vérification de la subsomption et de la satisfiabilité en fonction de l'expressivité des *LD*

Le tableau 4.3 présente un aperçu non exhaustif de la complexité du raisonnement au niveau terminologique en fonction de l'expressivité (Donini, 2003; Baader et al., 2003; Horrocks et Sattler, 2005). Ce tableau met en évidence la connaissance pointue des *LD* que la communauté scientifique détient. Les classes de complexité énumérées dans le tableau proviennent de la théorie de la complexité informatique. Voici une définition de ces classes (Padadimitriou, 1994) :

P : la classe des problèmes de décision [2] qui requièrent un temps polynomial par rapport à la taille du problème pour obtenir une solution avec une machine de Turing déterministe.

NP : la classe des problèmes qui nécessitent un temps polynomial pour trouver une solution avec une machine de Turing non déterministe.

PSpace : la classe des problèmes de décision qui requièrent une quantité de mémoire polynomiale pour résoudre un problème avec une machine de Turing déterministe ou non déterministe.

ExpTime : la classe des problèmes de décision solvables par une machine de Turing déterministe en un temps $\Theta(2^{p(n)})$ où $p(n)$ est une fonction polynomiale de n, la taille du problème.

NExpTime : la classe des problèmes de décision solvables par une machine de Turing non-déterministe en un temps $\Theta(2^{p(n)})$ où $p(n)$ est une fonction polynomiale de n, la taille du problème.

Il est connu que P $\subseteq$ NP $\subseteq$ PSpace $\subseteq$ ExpTime $\subseteq$ NExpTime (Padadimitriou, 1994).

4.4.2 L'inférence au niveau factuel

Le niveau factuel comprend quatre principaux problèmes d'inférence (Baader et Nutt, 2003) :

[2] Un *problème de décision* prend en entrée un énoncé de problème et produit en sortie une réponse positive ou négative (*oui* ou *non*).

Moteur	*Racer*	*FaCT*	*Pellet*	*FaCT++*
LD	$\mathcal{SHIQ}(\mathcal{D})-$	$\mathcal{SHIQ}$, $\mathcal{SHF}$	$\mathcal{SHIN}(\mathcal{D})$, $\mathcal{SHON}(\mathcal{D})$	$\mathcal{SHIF}(\mathcal{D})$
Implantation	*C++*	*Common Lisp*	*Java*	*C++*
Inférence	*TBox*/*ABox*	*TBox*	*TBox*/*ABox*	*TBox*
API *Java*	oui	oui	natif	oui
Mise-à-échelle	bonne	bonne	bonne	bonne
OWL	*OWL-DL*$\sim^{\dagger}$	*OWL-DL*$\sim^{\dagger}$	*OWL-DL*$\sim^{\dagger}$	*OWL-LITE*
Décidabilité	oui (*OWL-LITE*)	oui	oui (*OWL-LITE*)	oui
DIG	oui	oui	non	?

Moteur	*Surnia*	*Hoolet*	*F-OWL*	
LD	logique prédicats	logique prédicats	$\mathcal{SHIQ}(\mathcal{D})$ et *RDF*	
Implantation	*Python*	*Java*	*Java*	
Inférence	*TBox*/*ABox*	*TBox*/*ABox*	*TBox*/*ABox*	
API *Java*	?	oui	oui	
Mise-à-échelle	médiocre	médiocre	médiocre	
OWL	*OWL-FULL*$\sim^{\dagger}$	*OWL-DL*$\sim^{\dagger}$	*OWL-FULL*$\sim^{\dagger}$	
Décidabilité	non	non	non	
DIG	non	non	non	

† : Le symbole $\sim$ préfixé d'un nom de niveau *OWL*, signifie que le moteur d'inférence supporte approximativement ce niveau. La discussion sur *OWL* se trouve au chapitre 5.

Tableau 4.4 – Tableau comparatif des principaux moteurs d'inférence pour *LD*

- *Cohérence* : Une *ABox* $\mathcal{A}$ est cohérente par rapport à une *TBox* $\mathcal{T}$ si et seulement s'il existe un modèle $\mathcal{I}$ de $\mathcal{A}$ et $\mathcal{T}$).
- *Vérification d'instance* : Vérifier par inférence si une assertion $C(a)$ est vraie pour tout modèle $\mathcal{I}$ d'une *ABox* $\mathcal{A}$ et d'une *TBox* $\mathcal{T}$.
- *Vérification de rôle* : Vérifier par inférence si une assertion $R(a, b)$ est vraie pour tout modèle $\mathcal{I}$ d'une *ABox* $\mathcal{A}$ et d'une *TBox* $\mathcal{T}$.
- *Problème de récupération* : Pour une *ABox* $\mathcal{A}$, un concept C d'une terminologie $\mathcal{T}$, inférer les individus $a_1^{\mathcal{I}}...a_n^{\mathcal{I}} \in C^{\mathcal{I}}$ pour tout modèle $\mathcal{I}$ de $\mathcal{T}$.

4.4.3 Les moteurs d'inférence

Le tableau 4.4 dresse une comparaison des principaux moteurs d'inférence pour les *LDE*[3] : *FaCT* (Horrocks, 1998), *Racer* (Haarslev et Möller, 2001), *Pellet* (Sirin et Parsia, 2004), *FaCT++* (Tsarkov et Horrocks, 2004), *F-OWL* (Zou et al., 2004), *Surnia*, et *Hoolet*. Le critère *Mise-à-échelle* mesure la capacité à demeurer efficace proportionnellement à la complexification des ontologies. Le tableau reprend les données de Zou et al. (2004) pour ce critère, celui de décidabilité et pour les caractéristiques de *Hoolet*, *Surnia* et *F-OWL*. Ce chapitre ne traitera pas du critère *OWL*, car le chapitre 5 le présentera.

Le raisonnement sur *TBox* ou *ABox*

Tous ces moteurs raisonnent autant sur des *ABox* que sur des *TBox*, mis à part *FaCT* et *FaCT++* qui se spécialisent en raisonnant sur des *TBox* seulement. *FaCT++* est une variante de *FaCT* implantée en *C++* pour une efficacité accrue, et avec quelques ajouts et différences tels que le tableau l'illustre. Le raisonnement sur les *TBox* et *ABox* constitue un préalable pour le modèle proposé au chapitre 5.

L'expressivité et l'efficacité des moteurs d'inférence

Les moteurs *F-OWL*, *Hoolet* et *Surnia* raisonnent sur des logiques de description très expressives. *Hoolet* et *Surnia* raisonnent sur l'expressivité totale de la logique des prédicats, alors que *F-OWL* infère sur la logique $\mathcal{SHIQ}(\mathcal{D})$ et sur l'expressivité totale de *RDF*[4] Ces trois moteurs se basent sur des méthodes expérimentales de raisonnement qui exhibent des performances intéressantes pour des problèmes simples, mais insuffisantes pour une utilisation industrielle, en raison de leur faible efficacité et de la non-décidabilité de leurs algorithmes. Ce mémoire les mentionne à titre informatif.

[3] LDE : logiques de description expressives (cf. section 4.1.2)

[4] RDF (Ressource Description Framework) : http ://www.w3.org/RDF/. Un modèle RDF représente un domaine par un ensemble de triplets *sujet, prédicat, valeur* qui lient par des propriétés (prédicat) des ressources entre elles (sujet, valeur).

Comme l'indique le critère *Mise-à-échelle* dans le tableau, les moteurs les plus performants actuellement sont *Racer*, *FaCT*, *FaCT++* et *Pellet*. *FaCT*, *FaCT++* et *Racer* disposent probablement de la plus grande notoriété actuellement, chacun d'eux utilisé dans de nombreux projets. Le moteur *Pellet* est beaucoup plus récent que les autres et encore en développement. Il échoue à atteindre le niveau de performance de *FaCT* et *Racer* (Sirin et Parsia, 2004).

Le fonctionnement de *Racer* en grappe de calcul

Racer se présente comme un logiciel serveur avec lequel l'interaction a lieu par communication réseau. *Racer* offre également de fonctionner sous forme de grappe de calcul grâce à *RacerProxy*[5]. Cet aspect confère un avantage pour *Racer*, puisqu'un objectif d'*ASTUS* est de rendre certains laboratoires virtuels accessibles par le Web. En effet, que le serveur de raisonnement soit distant ou en grappe de calcul facilite la gestion de fortes demandes.

Les services offerts

D'après des tests réalisés pour ce mémoire avec la version 1.7.23 de *Racer*, ce dernier accuse un manque de fonctionnalité pour modifier une base de connaissances en mémoire et un manque de certains services de raisonnement. Par exemple, *Racer* supporte les rôles à valeur primitive, mais ne propose pas de méthode pour en ajouter de nouveaux ou en supprimer dans une base de connaissances chargée en mémoire. *Pellet*, quoique moins rapide, offre davantage de services pour raisonner et modifier une base de connaissances, selon des essais avec la version 1.1.0. L'auteur de ce mémoire a omis volontairement de tester les moteurs *FaCT* et *FaCT++*, car ils ne raisonnent pas au niveau *ABox*, un aspect essentiel pour le modèle proposé dans ce mémoire au chapitre 5.

[5] voir : http ://www.sts.tu-harburg.de/ r.f.moeller/racer/

L'interface de communication *DIG*

Racer, *FaCT* et *FaCT++* se conforment à *DIG* (Bechhofer et al., 2003), un protocole standard pour interroger un moteur d'inférence par des requêtes *HTTP* [6]. *DIG* vise à faciliter la substitution d'un moteur d'inférence par un autre. Ce protocole se base sur le langage $\mathcal{SHOIQ}(\mathcal{D}_{\mathcal{N}}^{-})$. Certains auteurs proposent un *api Java* pour communiquer par *DIG*[7]. Toutefois, *DIG* comporte un lot considérable de problèmes (Dickinson, 2004) : (1) une documentation pauvre, incomplète pour certains aspects tels que les codes d'erreurs et les réponses aux requêtes, (2) un manque de régularité dans la nomination des entités, (3) des tests de conformité inexistants, (4) un non support de certains services de raisonnement tels que vérifier l'égalité d'individus nommés dans le cas où l'hypothèse de nom unique (*HNU*) ne tient pas et (5) peu de moteurs d'inférence se conformant à *DIG* actuellement.

Les interfaces de programmation *Java*

La plupart des moteurs mentionnés dans cette section procurent une interface de programmation (*API*) (autre qu'une interface *DIG*) pour faciliter l'accessibilité par un programme *Java* (voir le tableau). Ce critère importe particulièrement puisqu'*ASTUS* est réalisé en *Java*, et que *DIG* comporte plusieurs problèmes.

4.5 Discussion et conclusion

Les *LD* et ses moteurs d'inférence constituent d'excellents outils pour représenter les connaissances de domaines formels. Elles offrent une fondation logique et des algorithmes efficaces de raisonnement tout en profitant des bénéfices d'une approche ontologique.

[6] HTTP (HyperText Transfert Protocol) : http://www.w3.org/Protocols/
[7] voir http://dig.sourceforge.net/

4.5.1 L'applicabilité des *LD* aux systèmes tutoriels intelligents

À la connaissance de l'auteur de ce mémoire, seulement trois équipes de chercheurs proposent d'utiliser les *LD* dans les systèmes tutoriels intelligents (Teege, 1994; Bergamaschi et al., 1994; Krdzavac et al., 2004). En particulier, aucun n'évoque l'emploi des *LD* dans un *STI* cognitif. Les prochaines sous-sections résument les idées de Teege, Bergamaschi et al., et Krdzavac et al.

Les idées de Teege

Teege (1994) envisagea de représenter les connaissances sémantiques du domaine d'un *STI* avec les *LD*. Quoiqu'il ne valida pas expérimentalement ses propos, il énonça quelques idées intéressantes :

Abstraction sémantique : Les *LD* permettent de représenter le niveau de la sémantique (*TBox*) comme une abstraction de celui du langage naturel (*ABox*). Cette abstraction est nécessaire pour distinguer les réponses de l'apprenant et les explications/exemples en langage naturel du tuteur, des concepts généraux enseignés.

Évaluation de l'apprenant : Un *STI* pourrait extraire des concepts des réponses en langage naturel de l'apprenant pour former une *TBox*. Ensuite, le *STI* pourrait comparer les concepts de l'apprenant à ceux du module de l'expert. Par exemple, si le concept *ordinateur* du système subsume celui de l'apprenant, cela indique que l'apprenant ne maîtrise pas encore certaines caractéristiques du concept *ordinateur*.

Inférence des relations entre les concepts : La relation de subsomption calculée dynamiquement se montre pratique pour diverses tâches. Par exemple, le système peut y recourir afin de trouver un concept, subsumant un concept incompris, pour mieux expliquer ce qui caractérise le concept incompris ou pour trouver des concepts équivalents à un autre concept, etc.

Cette intégration des *LD* vise strictement la représentation des connaissances sémantiques. Que Teege exclut les connaissances procédurales n'est certainement pas un ha-

sard : les *LD* manquent de moyens simples pour les représenter et ce n'est pas le but des *LD*. Le commentaire de Teege le plus intéressant est probablement le suivant : tout ne doit pas forcément être représenté en *LD*. Teege suggère d'ailleurs d'ajouter des métadonnées pour complémenter les concepts : par exemple, une définition textuelle du concept ou une indication sur l'usage typique.

Les idées de Bergamaschi et al.

Bergamaschi et al. (1994) recourent aux *LD* au sein d'un *STI* non pas pour modéliser la connaissance sémantique, mais pour modéliser un profil d'apprenant et le classer dans un certain nombre de catégories. Par la suite, le système compare la catégorie aux préalables d'une leçon pour déterminer si elle est appropriée à l'étudiant. L'article, très bref, ne détaille pas la représentation du profil de l'apprenant ou des catégories. L'idée de Bergamaschi et al. (1994) est intéressante, mais elle ne vise pas la modélisation des connaissances du domaine, l'intérêt principal de ce mémoire.

Les idées de Krdzavac et al.

Krdzavac et al. (2004) récupèrent plusieurs idées de Teege dans leur article et en apportent quelques nouvelles :

Détecter les erreurs de modélisation : L'inférence permet de détecter les erreurs dans la modélisation, en inférant les connaissances implicites et en détectant les inconsistances.

Faciliter l'ajout de nouveau matériel pédagogique : Si quelqu'un désire ajouter une leçon (représentée par un concept en logique de description) à un ensemble de leçons constituant un cours, un moteur d'inférence peut inférer dynamiquement la position de la leçon par rapport aux autres leçons du cours, de par sa définition en *LD*.

Analyse des réponses de l'apprenant : Le système d'apprentissage pourrait consi-

dérer une réponse de l'étudiant comme une *ABox* et vérifier si elle respecte la *TBox* du domaine, pour en déterminer sa justesse. Le système pourrait déterminer pourquoi elle est incorrecte, etc.

4.5.2 Une approche combinée

L'utilisation des *LD* seule ne remplit pas les objectifs du projet *ASTUS* en terme de modélisation des connaissances du fait que les *LD* n'ont pas été conçues précisément pour les *STI*. Il serait donc requis d'élaborer des mécanismes complémentaires pour les connaissances procédurales (dynamisme) et les connaissances pédagogiques afin de bénéficier ainsi d'une représentation cognitive.

Le chapitre 2 a présenté *UV* dont une des principales faiblesses est la mauvaise intégration de la logique. Un assemblage judicieux avec les *LD* pourrait procurer une spécification solide pour asseoir les connaissances d'*ASTUS*.

L'assemblage semble facile à accomplir puisqu'*UV* possède lui aussi deux niveaux d'abstraction pour les connaissances sémantiques : concepts et cognitions[8]. Si une *TBox* représente les concepts d'*UV*, une *ABox* pourrait décrire des individus qui représentent les cognitions, et évoluer de concert avec la simulation de la mémoire de travail.

D'autre part, associer à chaque concept (au sens d'*UV*) une définition en *LD* permettrait de substituer la relation de subsomption à la relation d'héritage d'*UV*. L'avantage résiderait dans la possibilité de définir d'une façon plus riche les relations logiques entre les concepts. En effet, la relation d'héritage d'*UV* équivaut l'utilisation d'un seul constructeur parmi ceux offerts par les *LD* (le constructeur $C \sqsubseteq D$).

Le prochain chapitre (chapitre 5) propose un nouveau modèle de représentation des connaissances qui explore cette avenue. Ce modèle est conçu pour intégrer la notion d'objet d'apprentissage. Plusieurs avantages découlent de ce modèle.

[8]Une cognition est une instance de concept que la mémoire de travail humaine manipule.

CHAPITRE 5

Le modèle de représentation des connaissances d'*ASTUS*

Le modèle que ce mémoire propose combine : (1) l'approche des objets d'apprentissage, (2) le modèle *UV* et (3) les *LD*. Ce mémoire prend pour hypothèse que la combinaison de ces trois approches engendrera une combinaison gagnante. L'organisation des connaissances au sein du modèle se fait selon trois couches (cf. figure 5.1), spécifiées l'une après l'autre, chacune ajoutant des informations spécifiques et complémentaires aux précédentes et chacune décrivant les connaissances selon une perspective différente.

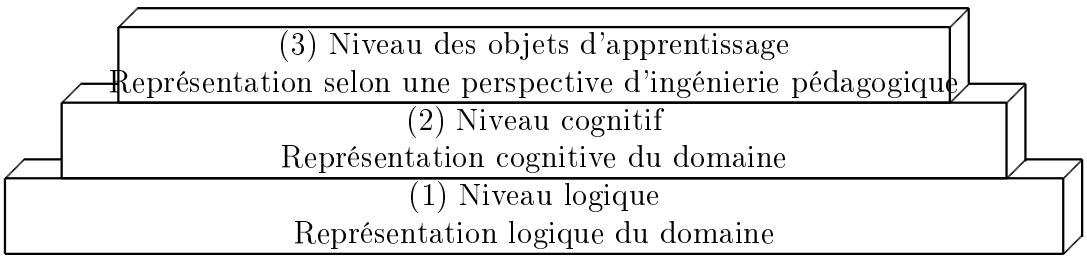

Figure 5.1 – Les trois niveaux descriptifs du modèle

La section 5.1 présente le niveau 1. Ce niveau représente les concepts d'un domaine dans une *TBox* avec des concepts au sens des *LD*, de manière analogue à Teege (1994) et Krdzavac et al. (2004). Cette représentation ne suffit pas pour effectuer une représentation cognitive d'un domaine puisque les *LD* ne sont pas conçues à cette fin.

Le niveau 2, présenté à la section 5.2, comble cette lacune. Il augmente les concepts dans la *TBox* du niveau 1 de facettes qui proviennent pour la plupart d'*UV*, afin d'en faire des concepts au sens psychologique. Le niveau 2 ajoute également la notion de procédures et de buts afin de modéliser les moyens cognitifs pour manipuler les instances des concepts (cognitions) et inclut des facettes pour représenter les connaissances didactiques. Les cognitions[1] sont représentées par des individus nommés ajoutés dans une *ABox* associée à la *TBox* de niveau 1. L'*ABox* contient l'ensemble des cognitions que la mémoire de travail simulée par le *MGC* (module de gestion des connaissances) manipule. L'*ABox* évolue dynamiquement, au fur et à mesure de l'exécution des procédures, pour refléter l'ensemble des cognitions manipulées par l'apprenant. La mise à jour de l'*ABox* et la génération de la mémoire épisodique se fait par le *MGC*. Le modèle reprend la représentation de la mémoire épisodique d'*UV*. La section 5.2 décrit la structure informatique pour encoder les connaissances de niveau 2, ainsi que le le fonctionnement du *MGC* pour l'interpréter.

Le modèle proposé dans ce chapitre est aussi une architecture cognitive, car les connaissances procédurales peuvent être interprétées et donner lieu à un comportement cognitif où des cognitions sont générées dans une mémoire de travail et encodées dans des épisodes. Cependant ce mémoire, tout comme *UV*, met davantage l'accent sur les structures de représentation des connaissances que sur l'aspect dynamique de la mémoire de travail. Le module « modèle de l'apprenant » représentera les contraintes sur les capacités cognitives et le niveau d'acquisition de chaque connaissance. Des recherches ultérieures élaboreront ces aspects.

[1] Une cognition est une instance de concept manipulée par la mémoire de travail pour l'accomplissement d'un épisode. Un épisode est la tentative d'accomplir un but avec une procédure à un temps t.

Avec les structures du niveau 2, le modèle suffit aux *STI*. La section 5.3 présente le niveau 3. Il ajoute des structures de données supplémentaires afin d'organiser les connaissances des niveaux 1 et 2 en objets d'apprentissage, pour faciliter la réutilisation des connaissances entre laboratoires virtuels.

La section 5.4 effectue une comparaison avec les structures de *CREAM* et propose une méthodologie inspirée de *CREAM* pour structurer les objets d'apprentissage en cours.

Finalement, la section 5.5 conclut le chapitre en résumant les bénéfices du modèle. Le chapitre suivant décrit une utilisation concrète du modèle comme preuve du concept, en relate les points positifs et les améliorations nécessaires.

5.1 Le niveau 1 : le niveau logique

Le niveau 1 représente les concepts d'un domaine dans une *TBox* avec des concepts au sens des *LD*, de manière analogue à Teege (1994) et Krdzavac et al. (2004). Dans chaque *TBox* définie pour le niveau 1, les concepts symbolisent des catégories d'objets manipulables dans un laboratoire virtuel (conceptuel ou réel) et les rôles, à savoir les relations logiques entre ces objets.

La figure 5.2 montre un ensemble de connaissances non exhaustif de niveau 1 pour un laboratoire virtuel de réduction booléenne (le chapitre 6 présente la totalité de cet exemple). Le premier axiome modélise par le concept *ExprAbstraite*, la notion d'expression booléenne valide au sens le plus large. Les axiomes qui suivent énoncent que les constantes de vérité, les variables et les expressions décrites, sont des expressions booléennes et qu'il y a deux sous-catégories de constantes (les constantes vrai et les constantes faux). De plus, l'exemple spécifie que chaque expression décrite possède un opérateur parmi trois types : les opérateurs de conjonction, les opérateurs de disjonction et les opérateurs de négation. La sous-section suivante motive le choix du langage concret pour représenter les connaissances de niveau 1.

Concepts
$ExprAbstraite \sqsubseteq \top$
$ConstanteDeVérité \sqsubseteq ExprAbstraite$
$ConstanteFaux \sqsubseteq ConstanteDeVérité$
$ConstanteVrai \sqsubseteq ConstanteDeVérité$
$Variable \sqsubseteq ExprAbstraite$
$ExprDécrite \sqsubseteq \Big(ExprAbstraite \sqcap (= n.opérateur\ 1) \sqcap \forall opérateur.Opérateur\Big)$
$Opérateur \sqsubseteq \top$
$OpérateurNégation \sqsubseteq Opérateur$
$OpérateurDisjonction \sqsubseteq Opérateur$
$OpérateurConjonction \sqsubseteq Opérateur$
...
Rôles
$opérateur \sqsubset \top_R$
...

Figure 5.2 – Exemple non exhaustif de connaissances de niveau 1 pour la réduction booléenne

5.1.1 Le langage de représentation

Les *LD* telles que le chapitre 4 les présente, constituent un formalisme dépourvu de représentation informatique. Plusieurs institutions proposent des formats de fichiers standardisés pour représenter des bases de connaissances en *LD*. Cette section attaque le problème de choisir l'une de ces spécifications pour représenter les connaissances de niveau 1. Les critères pour le choix d'un langage étaient les suivants :

- Le langage doit se baser sur une ou des *LD* suffisamment expressives pour accepter des modèles complexes avec *TBox* et *ABox* ;
- Un moteur d'inférence gratuit, capable de raisonner autant sur des *ABox* que des *TBox*, efficace et exploitable par des programmes *Java* (le langage d'implantation d'*ASTUS*) doit être disponible ;
- Un système auteur graphique doit préférablement être disponible ;
- Le langage doit être idéalement répandu afin d'assurer une certaine compatibilité avec d'autres systèmes.

Ce travail a considéré plusieurs langages adoptés massivement par la communauté scientifique tels qu'*OIL* (Fensel et al., 2001), *DAML+OIL*[2] et *OWL*[3] (Horrocks et al., 2003). Le choix fut facile puisqu'un seul d'entre eux se démarque : *OWL*. Il est d'ailleurs le successeur de son plus proche compétiteur *DAML+OIL*[4] qui est le sucesseur d'*OIL*. *OWL* est probablement le langage basé sur les *LD* qui bénéficie des plus grands efforts de conception et de mise en oeuvre actuellement. Les raisons suivantes motivent plus particulièrement le choix du langage *OWL*.

Une recommandation du *W3C* : le *W3C* recommande *OWL* pour la description des ontologies pour le Web sémantique, ce qui a permis de recueillir de nombreux commentaires de sources externes et d'attirer des efforts considérables.

Trois ensembles de primitives : *OWL* offre trois ensembles de primitives, pour différents compromis entre expressivité et performance ; du moins expressif au plus expressif, il y a *OWL-LITE*, *OWL-DL* et *OWL-FULL*. Les deux premiers correspondent étroitement à des *LDE* largement étudiées : $\mathcal{SHIF}(\mathcal{D})$ et $\mathcal{SHION}(\mathcal{D})$ respectivement.

Pour fin de rappel, $\mathcal{SHI}(\mathcal{D})$ comprend la négation complète ($\neg C$), l'intersection ($C \sqcap D$), la quantification existentielle limitée ($\exists R.\top$), la quantification universelle complète ($\forall R.C$), les rôles transitifs ($trans(R)$) , les rôles inverses (R_1^{-1}), les hiérarchies de rôles ($R_1 \sqsubseteq R_2$), et l'extension des types primitifs. Le constructeur $\mathcal{O}$ autorise la description de concepts par énumération d'individus nommés ($\{a_1, a_2, ..., a_n\}$). Les constructeurs $\mathcal{F}$ ($= 1R, \geq 2R$) et $\mathcal{N}$ ($\geq nR, \leq nR, = nR$) constituent deux variantes de la contrainte de cardinalité sur rôle.

La complexité des problèmes d'inférence avec *OWL-LITE* et *OWL-DL* est bien connue et des algorithmes efficaces pour les résoudre ont été proposés. Pour *OWL-FULL*, qui correspond vaguement à l'expressivité complète de *RDF* combinée à

[2] *DAML+OIL* : http://www.daml.org/2001/03/daml+oil-index.html
[3] *OWL* : http://www.w3.org/2004/OWL/
[4] Un convertisseur *DAML+OIL* vers *OWL* est disponible : *OWL Converter* : http://www.mindswap.org/ golbeck/code.shtml

$\mathcal{SHIQ}(\mathcal{D})$), des moteurs d'inférence tels que *F-OWL* et *Surnia* sont disponibles, mais ils sont indécidables et incomplets (Zou et al., 2004).

Les systèmes auteurs d'*OWL* **:** Plusieurs outils d'édition pour *OWL* sont disponibles. Ce mémoire suggère *Protégé* (Knublauch et al., 2004) pour sa convivialité et son support de la presque totalité de la spécification *OWL*. Toutefois, chaque concepteur demeure libre d'utiliser l'outil qu'il souhaite, en autant que l'outil génère des fichiers conformes.

Les moteurs d'inférence pour *OWL* **:** Plusieurs chercheurs offrent des moteurs d'inférence efficaces gratuits (cf. section 4.4.3). Actuellement, les principaux moteurs d'inférence acceptent en entrée des fichiers *OWL*.

La syntaxe *XML* **:** *OWL* utilise la syntaxe *XML*, ce qui permet une manipulation facile des données (avec *XSLT*[5], etc.) et a l'avantage de bien s'intégrer avec d'autres spécifications.

Les mécanismes d'extension d'ontologies de *OWL* **:** *OWL* repose sur un principe de développement distribué et incrémental d'ontologies ; des mécanismes en place facilitent l'extension d'ontologies *OWL* existantes. En effet, un fichier *OWL* peut inclure et étendre des définitions de fichiers *OWL* existants sans modifier directement les fichiers originaux. Afin de préserver l'intégrité de ces ontologies distribuées, *OWL* propose des balises de versionnage et la notion d'espace de nom.

Les connaissances de niveau 1 pour un laboratoire virtuel peuvent être réparties dans plusieurs fichiers *OWL*. Ces fichiers peuvent avoir été créés pour un laboratoire spécifique ou être conçus pour un autre, mais réutilisés.

[5] *XSLT* (eXtensible Stylesheet Language Transformations) : www.w3.org/TR/XSLT. *XSLT* est un langage de spécification de transformations sur des fichiers *XML* pour produire en sortie des fichiers texte de toutes formes : *XML*, etc.)

5.2 Le niveau 2 : le niveau cognitif

Les structures de niveau 1 ne permettent pas d'effectuer simplement une représentation cognitive d'un domaine puisque les *LD* ne sont pas conçues pour cet usage. Cette section présente le niveau 2 qui comble cette lacune.

Au second niveau, le concepteur représente les connaissances manipulées par l'architecture cognitive des apprenants et pertinentes à un laboratoire virtuel. Ce niveau augmente les concepts et rôles dans la *TBox* du niveau 1, de facettes qui proviennent pour la plupart d'*UV*, afin d'en faire des concepts au sens psychologique : chaque définition de concept primitif pointe vers un concept *OWL* et chaque définition de concept décrit pointe aussi vers un concept *OWL* ainsi que vers plusieurs rôles *OWL* pour lier les instances du concept décrit à ses composantes. Le niveau 2 ajoute également la notion de procédures (actions) et de buts (intentions), afin de modéliser les moyens cognitifs pour manipuler les instances des concepts (cognitions). Le modèle incorpore ainsi l'aspect interactif des micromondes.

Contrairement à *UV* (cf. chapitre 2), ce modèle décrit les buts avec des facettes différentes de celles des concepts parce qu'il ne permet pas de les utiliser comme paramètres d'une procédure. Ce choix de représentation présent dans *MIACE* (Mayers, 1997) ne se justifie pas par une différence de nature[6], mais par l'utilisation cognitive particulière des buts. Les buts constituent des intentions qui amènent l'action, alors que les concepts représentent des connaissances déclaratives manipulées par les procédures pour atteindre les buts. L'utilité d'appliquer une procédure sur un but, d'inclure un but comme composante d'un concept décrit et de créer un but décrit[7] n'est pas un sujet étudié dans la littérature. Ce mémoire préfère donc s'en tenir au minimum et par conséquent, il reprend la position de

[6] Les études récentes tendent à montrer que les buts sont sujets aux mêmes contraintes que les autres connaissances déclaratives : ils sont enclins à l'oubli, en compétition pour le rappel et ils possèdent une activation qui s'accroît en fonction du contexte (Altmann et Trafton, 2002; Anderson et Douglass, 2001; Trafton et al., 2003).

[7] Ce mémoire ne considère pas les arguments d'un but comme ses composantes, au sens d'un concept décrit.

MIACE (Mayers, 1997).

Le modèle décrit dans ce chapitre n'associe pas de descriptions logiques de niveau 1 aux buts, puisque (1) les procédures complexes lient déjà les buts à des sous-buts et que (2) hormis cette relation de tâche/sous-tâches, un lien logique entre des intentions n'a pas de signification contrairement aux liens entre les autres connaissances sémantiques (par exemple, la relation *capitale* peut lier le concept *Paris* au concept *France*).

Le modèle ne retient pas la notion de concepts quelconques, concepts non identifiés, fonctions et relations d'*UV*, car le niveau 1 satisfait le besoin d'une représentation logique.

Des fichiers nommés *SPK*[8], de format *XML*, encapsulent les connaissances de niveau 2. Les auteurs établissent des liens entre les concepts primitifs, les concepts décrits (dans les fichiers *SPK*) et les concepts et les rôles (dans les fichiers *OWL*). De cette façon, le second niveau étend le premier.

La section 5.2.1 présente tout d'abord la structure des fichiers *SPK*, fortement inspirée d'*UV* et la section 5.2.2, le *MGC* élaboré pour fonctionner avec ce modèle. La description du *MGC*, appuyée de quelques exemples, permettra de mieux cerner la relation entre les niveaux 1 et 2.

5.2.1 La structure d'un fichier *SPK*

Un fichier *SPK* (noté $f_1, f_2, ...$) se compose d'un en-tête et d'un ensemble de définitions d'entités. Les quatre types d'entités sont : les concepts primitifs, les concepts décrits, les buts et les procédures. Les prochaines sous-sections décrivent les facettes de ces entités. Certaines facettes sont présentées, mais leur utilité dans la pratique ne pourra être comprise qu'ultérieurement avec les exemples de la section sur le *MGC* (cf. section 5.2.2).

D'autre part, la présentation omet la description des facettes didactiques puisque le

[8] *SPK* (Semantic and Procedural Knowledge) :

modèle reprend tel quel les facettes définies par Hafsaoui (2005). Leur amélioration ne relève pas des objectifs de ce travail et elle sera le sujet d'autres recherches.

L'en-tête des fichiers *SPK*

L'en-tête d'un fichier *SPK* comprend un ensemble de facettes de métadonnées :
– **FichiersOWL** : la liste des ontologies *OWL* requises et leur version ;
– **FichiersSPK** : la liste des fichiers *SPK* requis et leur version ;
– **Métadonnées générales** : des métadonnées telles que l'auteur, la version, etc.
Pour chaque fichier *SPK*, la facette *FichiersOWL* met en liste les fichiers *OWL* qui contiennent des descriptions *OWL* référencées dans les définitions des entités de ce fichier *SPK*.

Le modèle autorise la répartition des connaissances de niveau 2 dans plusieurs fichiers *SPK*, pour favoriser la réutilisation. À cet effet, la facette *FichiersSPK* énumère un ensemble de fichiers *SPK* à charger conjointement. Par exemple, un fichier *SPK* $f1$ qui définit la procédure $PAppliquerDeMorgan$ pourrait indiquer, par sa facette *FichiersSPK*, de précharger un fichier $f2$ qui définit le concept d'expression booléenne, afin d'éviter d'avoir à le redéfinir.

Finalement, un ensemble de métadonnées générales complète l'en-tête en énonçant par exemple, la version, l'auteur, etc. Le versionnage revêt d'une importance particulière, pour assurer la cohérence de l'importation de définitions d'autres fichiers *SPK*.

Les facettes des concepts primitifs

Cinq facettes décrivent les concepts primitifs (cf. tableau 5.1). *IDconcept* est une chaîne de caractères qui identifie le concept de façon unique. *RéférenceOWL* contient une référence vers un concept *OWL* qui spécifie une description logique de ce concept primitif. Cette référence sert lors de l'instanciation du concept primitif par une procédure. À ce moment, le *MGC* ajoute dans une *ABox*, un individu nommé *OWL* associé au concept

Facettes de métadonnées	Facettes instanciation/ utilisation	Facettes OWL
IDConcept	Buts	RéférenceOWL
Métadonnées	Constructeurs	

Tableau 5.1 – Les facettes des concepts primitifs

Facettes de métadonnées	Facettes instanciation/ utilisation	Facettes OWL	Facettes évaluation
IDConcept	Buts	RéférenceOWL	But-évaluation
Métadonnées	Constructeurs	Composantes	

Tableau 5.2 – Les facettes des concepts décrits

OWL mentionné dans la facette *RéférenceOWL*. La section 5.2.2 explicitera les détails. *Métadonnées* contient un ensemble de métadonnées générales sur le concept telles que le nom du concept présenté à l'apprenant par le système, une description textuelle du concept, l'auteur du concept, etc. La facette *Buts* contient la liste des buts dont la réalisation nécessite une occurrence de ce concept. Autrement dit, la facette indique les intentions qu'un apprenant peut avoir avec ce concept. *Constructeurs* spécifie les identificateurs des principales procédures qui permettent d'obtenir une instance de ce concept.

Les facettes des concepts décrits

Le modèle représente les concepts décrits avec sept facettes (cf. tableau 5.2). *IDconcept*, *RéférenceOWL*, *Métadonnées*, *Buts* et *Constructeurs* possèdent la même signification qu'antérieurement.

La facette *Composantes* contient une liste de références vers des rôles *OWL*. Pour chaque instance de concept primitif ou décrit, le *MGC* associe dans une *ABox* un individu nommé membre du concept *OWL* cité dans la facette *RéférenceOWL*. Chaque rôle *OWL*, mentionné dans la facette *Composantes*, permet de lier les individus nommés qui représentent

Facettes de métadonnées	Facettes spécifiques aux buts
IdBut	Habileté
Métadonnées	Paramètres
	Procédures

Tableau 5.3 – Les facettes des buts

Facettes de métadonnées	Facettes préconditions	Facettes type	Facettes comportement	Facettes utilisation
IDProcédure	Arguments	Validité	Appel externe	Contexte
Métadonnées	But-associé	Type	Script	Utilisation

Tableau 5.4 – Les facettes des procédures

des instances de ce concept décrit à des individus nommés qui représentent une composante. Les détails seront présentés à la section 5.2.2.

La facette *But-évaluation* contient une référence vers le but qui permet de procéder à l'évaluation de ce concept décrit. Par exemple, pour un concept décrit (+ 4 5), cette facette pourrait indiquer le but d'additionner les deux nombres composantes. La facette *But-évaluation* peut aussi être laissée vide.

Les facettes des buts

Cinq facettes décrivent les buts (cf. tableau 5.3). La facette *IDBut* contient une chaîne de caractères qui identifie le but de façon unique. *Métadonnées* possède la même signification que pour les autres entités. *Habileté* contient une chaîne de caractères qui spécifie l'habileté humaine requise pour accomplir le but. La facette *Paramètres* contient des références vers les concepts nécessaires à la réalisation de ce but. *Procédures* spécifie les procédures disponibles pour tenter de réaliser le but.

Les facettes des procédures

Le tableau 5.4 présente les facettes des procédures. La facette *IDProcédure* contient un identificateur unique pour la procédure. La facette *Métadonnées* détient la même signification que pour les autres entités. La facette *But-associé* pointe vers le but associé à cette procédure. *Arguments* spécifie le type de concepts pris en paramètre par la procédure au moyen d'une liste de paires. Chaque paire décrit un argument et se compose d'un identificateur de concept (le type de l'argument) et d'une chaîne de caractères donnant un nom arbitraire à l'argument. Les noms arbitraires sont utilisés dans la facette *Script* ou *Appel externe* pour désigner les arguments.

La facette *Arguments* et la facette *But-associé* équivalent à la partie *condition* de règles de productions, car la procédure ne peut s'activer qu'en présence d'un but et d'arguments compatibles avec cette description (voir section 2.2.4 pour plus de détails).

La facette *Validité* contient une paire de valeurs booléennes : la première indique si la procédure trouve à tout coup le bon résultat (pour le but associé) et la seconde, si la procédure termine toujours. Autant les procédures valides que non valides sont encodées, car l'apprenant peut les utiliser. La facette *Type* indique le type de la méthode : primitive ou complexe. Spécifier le type explicitement permet de créer des procédures primitives avec script. L'action résultant de l'application d'une procédure se décrit par la facette *Script* ou *Appel externe*. Pour les procédures complexes, *Script* énonce le script des sous-buts à accomplir, alors que pour les procédures primitives, deux possibilités s'offrent. Premièrement, un auteur peut décrire l'action d'une procédure primitive en indiquant une méthode *Java* qui accomplit l'action dans la facette *Appel externe*. Deuxièmement, une procédure primitive peut être spécifiée par la facette *Script* à la condition qu'elle ne spécifie qu'un seul but système. À ce moment, le *MGC* interprétera le comportement de la procédure comme étant le même que la procédure associée au but système mentionné dans la facette *Script*. Le bénéfice de cette capacité est de permettre à un auteur de modéliser une procédure primitive qui reproduit le comportement d'une procédure système, mais

qui associe des valeurs différentes aux autres facettes. Par exemple, un auteur peut définir une procédure primitive *PRemplacerComposanteExpressionAddition* qui spécifie le script *(REMPLACE ConceptDécrit* X_{ancien} $X_{nouveau}$*)* et qui prend en paramètre un concept de type *ExpressionAddition* (une expression mathématique de la forme $(x + y)$) et deux concepts de type *ExpressionMathématique*. Cette procédure effectue la même action que la procédure système associée au but *REMPLACE*, mais elle prend en argument une expression de type *ExpressionAddition* ; elle se spécialise dans le remplacement d'une composante d'un concept décrit de type *ExpressionAddition*. Il ne fait aucun doute qu'une telle action spécialisée existe et qu'un auteur doit la représenter comme une procédure primitive, plutôt que sous la forme d'une procédure complexe qui spécifie le but système *REMPLACE* comme sous-but. C'est ce qui justifie la nécessité de pouvoir définir des procédures primitives qui reproduisent le comportement de buts systèmes.

Les expérimentations avec l'implantation du chapitre 2 n'ont pas fait usage des facettes *Contexte* et *Utilisation*. Ce modèle les reprend tel quel, mais des recherches ultérieures confirmeront ou infirmeront leur utilité.

Le code compilé des procédures primitives

Dans la mise en oeuvre d'*UV* (chapitre 2), le code compilé de chaque procédure primitive se trouve dans une classe *Java* qui correspond à un concept. Ce modèle place plutôt chaque méthode *Java* dans une classe *Java* individuelle. Cette approche facilite la réutilisation. En effet, pour ajouter une procédure primitive avec code compilé, il suffit d'ajouter une classe *Java* qui correspond à la procédure ; nul besoin de modifier une classe existante représentant un concept et possiblement définie par d'autres auteurs.

D'autre part, plutôt que de définir une classe pour chaque concept, le modèle définit une classe générique nommée *Cognition.java* pour représenter les instances de concept manipulées par le *MGC*. Ce changement devient possible puisque les méthodes *Java* des procédures primitives ne se retrouvent plus dans des classes qui représentent les concepts

comme dans la mise en oeuvre d'*UV*. Les instances de *Cognition.java* possèdent une variable membre *RefIndividuOWL* qui pointe vers l'individu nommé *OWL* correspondant dans l'*ABox* du moteur d'inférence (la section 5.2.2 dévoilera les détails avec des exemples).

5.2.2 Le fonctionnement du *MGC*

Cette section décrit le fonctionnement du *MGC* pour le modèle de connaissances. Le *MGC* conserve l'algorithme général de l'exécution d'une procédure d'*UV* (décrit à la section 2.3.6) ainsi que la représentation de la mémoire épisodique d'*UV*. Cette section s'organise comme suit. La première sous-section décrit le processus d'initialisation du *MGC* avec les connaissances d'un laboratoire. Les deux sous-sections suivantes expliquent les procédés utilisés par le *MGC* pour créer des cognitions primitives et des cognitions décrites. Ensuite, des sous-sections énoncent respectivement les buts systèmes autorisés avec ce modèle et les algorithmes du *MGC* pour accomplir l'action de chaque procédure système. Après, une sous-section justifie le choix de moteurs d'inférence pour l'implantation du modèle. Finalement, une sous-section discute d'une stratégie adoptée par le modèle et implantée dans le *MGC* pour faciliter l'extension des connaissances existantes.

L'initialisation du *MGC*

Le processus d'initialisation se déroule comme suit (cf. figure 5.3) :

1. Un module logiciel qui désire utiliser le *MGC* envoie une requête au *MGC* pour demander l'initialisation avec les fichiers *SPK* $f_1, f_2, ...f_n$ d'un laboratoire.
2. Le *MGC* transmet la requête à son noyau (l'interpréteur). Ce dernier charge les fichiers $f_1, f_2, ...f_n$ en mémoire. Si ces fichiers demandent dans leur en-tête le chargement d'autres fichiers *SPK* (cf. section 5.2.1), le *MGC* les chargera également.
3. L'interpréteur repère dans les en-têtes de ces documents le nom des fichiers *OWL* né-

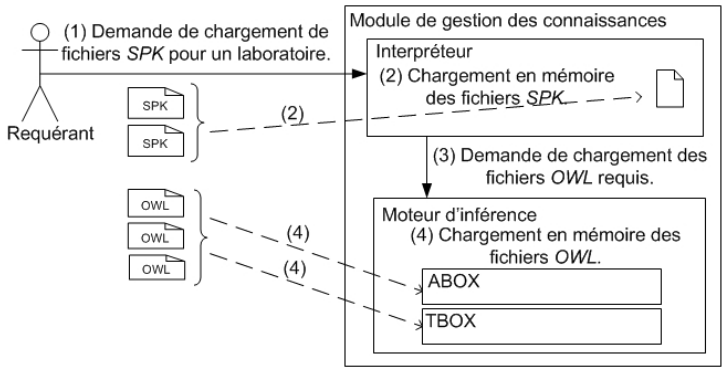

Figure 5.3 – Processus d'initialisation du *MGC* d'*ASTUS*

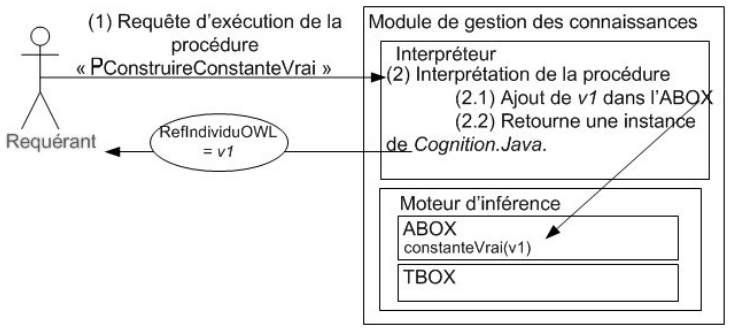

Figure 5.4 – Scénario d'instanciation d'un concept primitif

cessaires et commande leur chargement par le moteur d'inférence.

4. Le moteur d'inférence charge les fichiers *OWL* dans une *TBox* et une *ABox*.

Le processus d'instanciation d'un concept primitif

La création d'une instance d'un concept primitif constitue un mécanisme fondamental et essentiel de la mémoire de travail. En effet, chaque occurrence de concept doit être créée avant d'être prise en paramètre par une procédure pour accomplir un but. Pour expliquer le processus d'instanciation des concepts primitifs par le *MGC*, considérons le scénario où la requête d'exécution d'un constructeur de concept primitif, nommé *PConstruire-ConstanteVrai*, parvient au *MGC* (cf. figure 5.4). En premier lieu, l'interpréteur scrute

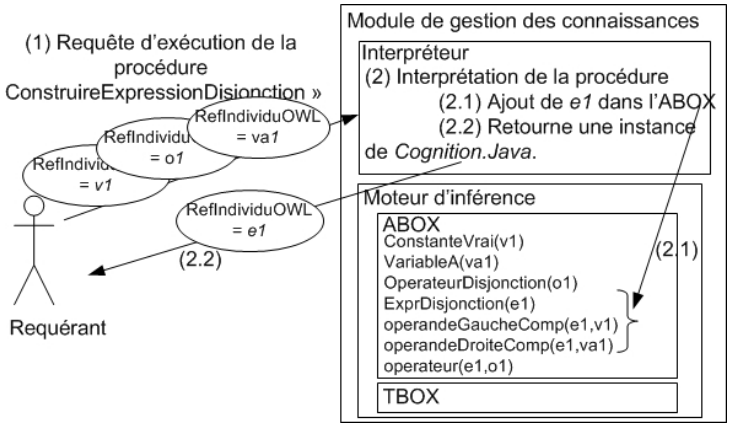

Figure 5.5 – Scénario d'instanciation d'un concept décrit

les fichiers *SPK* en mémoire et découvre que la procédure est primitive et exécutable par le script *(CONSTRUIT-PRIMITIF "CConstanteVrai")*.

Le but système *CONSTRUIT-PRIMITIF* ordonne la création d'une nouvelle instance d'un concept primitif (ici, le concept *CConstanteVrai*). L'interpréteur récupère la valeur de la facette *RéférenceOWL* qui indique pour ce concept, un concept *OWL* C_1 correspondant. Ensuite, l'interpréteur crée un nouvel individu nommé a dans l'*ABox* par l'ajout d'une assertion d'appartenance $C_1(a)$.

Finalement, l'interpréteur retourne une nouvelle instance de la classe *Java Cognition.java* avec sa variable membre *RefIndividuOWL* initialisée avec la chaîne de caractères a, et la retourne à l'initiateur de la requête. Les diagrammes de ce chapitre représentent par une ellipse les instances de *Cognition.java*.

À la figure 5.4, $ConstanteVrai$ et $v1$ jouent respectivement le rôle de C_1 et de a .

Le processus d'instanciation d'un concept décrit

Les cognitions manipulées par la mémoire de travail peuvent être également décrites (instances de concepts décrits). Cette section présente un second scénario qui expose le

mécanisme de création des cognitions décrites (cf. figure 5.5). Tout d'abord, la requête d'exécution de la procédure *PConstruireExpressionDisjonction* parvient au *MGC* avec des instances de *Cognition.java* dont leur facette *RefIndividuOWL* pointe respectivement vers des individus nommés $a_1, a_2, ...a_n$ dans l'*ABox* (dans l'exemple, $v1$, $o1$ et $va1$). Ces individus nommés formeront les composantes de la cognition décrite qui sera créée.

Premièrement, l'interpréteur vérifie le type de la procédure (ici, primitive avec un script). Il décode le script (dans l'exemple : *(INSTANCIE "CExpressionDisjonction" argument1 argument2)*). Le but système *INSTANCIE* ordonne la construction d'une nouvelle instance d'un concept décrit CC_1 (ici, le concept *CExpressionDisjonction*). L'interpréteur récupère la valeur de la facette *RéférenceOWL* qui indique pour ce concept, un concept *OWL* C_1 correspondant. Ensuite l'interpréteur crée un nouvel individu nommé a dans l'*ABox* par l'ajout d'une assertion d'appartenance $C_1(a)$ ($ExprDisjonction(e1)$ dans l'exemple).

Puis, l'interpréteur crée dans l'*ABox* pour chaque nom de rôle $r_1, r_2, ...r_n$ énoncé dans la facette *Composantes* du concept CC_1 (*CExpressionDisjonction*), des assertions de rôle afin de spécifier les composantes du concept décrit nouvellement créé : $r_1(a, a_1), r_1(a, a_2), ...r_1(a, a_n)$ (dans l'exemple, les assertions $opérandeGaucheComp(e1, v1)$, $opérandeDroiteComp(e1, va1)$ et $opérateur(e1, o1)$).

Le *MGC* retourne une nouvelle instance de la classe *Cognition.java* avec la valeur a ($e1$ dans l'exemple) affectée à son attribut *RefIndividuOWL*.

Finalement, dans une table associative nommée *Table des concepts décrits*, le *MGC* ajoute l'association (a, CC_1). Cette table permet de connaître en tout temps le type des cognitions décrites et est nécessaire pour l'accomplissement des buts systèmes *REMPLACE*, *ÉVALUE* et *ÉQUIVALENT-OWL*.

La vérification de l'applicabilité d'une procédure ou d'un but

La vérification de l'applicabilité d'une procédure à une instance I_1 d'un concept CX se réalisait dans UV en deux étapes. Tout d'abord, par l'obtention du nom du concept CY demandé en argument par la procédure, puis par la vérification que $CX = CY$ ou que CY est parent de CX par le biais de liens de généralisation.

Le modèle que propose ce mémoire n'incorpore pas de liens d'héritage explicites. Pour déterminer si une instance de concept I_1 appartient au concept CY, l'interpréteur récupérera l'individu nommé $OWL\ a$ correspondant à I_1 (inscrit dans sa facette *RefIndividuOWL*) et le concept $OWL\ C_1$ correspondant à CY. Puis il demandera au moteur d'inférence de vérifier si a est membre de C_1. En d'autres mots, la relation d'héritage est remplacée par la relation de subsomption des LD, calculée dynamiquement, selon les caractéristiques (participation à des rôles et appartenance à un concept) d'un individu nommé. Ceci permet, par exemple, d'inférer par les composantes d'une instance de concept décrit son appartenance à un concept précis ; l'appartenance pouvant varier en fonction des composantes. Ainsi, une cognition décrite initialisée comme instance de rationnel avec les composantes (5 , 0), pourrait être perçue comme une instance d'entier et pourrait donc être employée avec une procédure *PCalculerFactoriel*, contrairement à une autre cognition décrite initialisée comme instance de rationnel avec les composantes (5 , 1).

Les buts systèmes

Les buts systèmes encodent les fonctionnalités de base de l'architecture cognitive et doivent se justifier par leur nécessité logique ou par des fondements psychologiques. Cependant dans le modèle, certains buts systèmes bien identifiés échappent à cette règle pour simplifier les connaissances à encoder ou parce que le modèle comme architecture cognitive est une approximation exacte de la réalité et que les recherches futures corrigeront la situation (cf. section 2.2.9). Les buts systèmes *INSTANCIE*, *ÉVALUE*, *TROUVE*,

But système	Arguments	Résultat
CONSTRUIT-PRIMITIF	$idConceptPrimitif$	Une instance de concept primitif.
INSTANCIE	« idConceptDécrit », $arg_1, arg_2, ...arg_n$	Une cognition décrite
ÉVALUE	$CognitionDécrite1$	Une cognition
TROUVE	$entier1, entier2$	Une cognition
SÉQUENCE	$But_1, But_2, ...But_n$	Le résultat de But_n
REMPLACE	$CognitionDécrite1$, $AncComp$, $NouvComp$	Une cognition décrite
COMPOSANTE	$CognitionDécrite1$, « nomRôle »	Une cognition
ÉQUIVALENT-OWL	$Cognition1, Cognition2$	Un booléen $vrai$ ou $faux$
IF	$valeur, B_1, B_2$	Le résultat de B_1 ou B_2

Tableau 5.5 – Les différents buts systèmes et leur syntaxe dans *ASTUS*

SÉQUENCE, *REMPLACE*, et *COMPOSANTE* conservent la même sémantique et les mêmes justifications que les buts systèmes décrits au chapitre 2. À cela s'ajoute le but système *CONSTRUIT-PRIMITIF* qui demande la construction d'une instance d'un concept primitif, le but *ÉQUIVALENT-OWL* qui vérifie que deux cognitions s'équivalent logiquement, et le but *IF* qui effectue un choix. Les prochaines sections décrivent les algorithmes implantés dans le *MGC* pour simuler la réalisation de chaque but système.

Le but système *CONSTRUIT-PRIMITIF*

Le but *CONSTRUIT-PRIMITIF* se réalise en trois étapes :

1. L'interpréteur récupère la valeur de la facette *RéférenceOWL* pour le concept $idConceptPrimitif$ qui indique le concept *OWL* C_1 correspondant.
2. L'interpréteur crée un nouvel individu nommé a dans l'*ABox* par l'ajout d'une assertion d'appartenance $C_1(a)$.
3. Le *MGC* retourne une nouvelle instance de la classe *Cognition.java* et affecte la valeur a à son attribut *RefIndividuOWL*.

Le but système *INSTANCIE*

Le but système *INSTANCIE* consiste à créer une instance d'un concept décrit. Les arguments sont l'identificateur du concept décrit, et les occurrences de concept qui formeront ses composantes. Le résultat est une instance du concept décrit. Voici le déroulement de son exécution :

1. L'interpréteur récupère la valeur de la facette *RéférenceOWL* du concept *idConceptDécrit* qui indique un concept *OWL* C_1 correspondant.
2. L'interpréteur récupère les noms des individus nommés *OWL* $b_1, b_2, ...b_n$ correspondant aux paramètres $arg_1, arg_2, ...arg_n$ par l'inspection de leur attribut *RefIndividuOWL*.
3. L'interpréteur crée un nouvel individu nommé a dans l'*ABox* par l'ajout d'une assertion d'appartenance $C_1(a)$.
4. L'interpréteur crée dans l'*ABox* pour tous noms des rôles $r_1, r_2, ...r_n$ énoncés dans la facette *Composantes* du concept *idConceptDécrit*, des assertions de rôle afin de spécifier les composantes du concept décrit nouvellement créé : $r_1(a, b_1), r_2(a, b_2), ...r_n(a, b_n)$.
5. Finalement, le *MGC* retourne une nouvelle instance de la classe *Cognition.java* avec la valeur a affectée à son attribut *RefIndividuOWL*.

Le but système *ÉVALUE*

ÉVALUE effectue l'évaluation d'une cognition décrite *CognitionDécrite*1 et retourne le résultat. Par exemple, un apprenant pourrait évaluer le concept décrit (+ 5 6) pour obtenir le résultat 11. Formellement, ce but se réalise ainsi :

1. L'interpréteur obtient la valeur CC_1 associée à *CognitionDécrite*1 dans la *Table des concepts décrits* (cf. page 117).
2. L'interpréteur obtient le nom a de l'individu nommé *OWL* inscrit dans l'attribut *RefIndividuOWL* de *CognitionDécrite*1.

3. L'interpréteur récupère les noms des rôles $r_1, r_2, ...r_n$ inscrits dans la facette *Composantes* du concept décrit CC_1.
4. L'interpréteur récupère le nom des individus nommés *OWL* $a_1, a_2, ...a_n$ tel que $r_1(a, a_1)$, $r_2(a, a_2)$, ... $r_n(a, a_n)$ tiennent dans l'*ABox*.
5. L'interpréteur récupère la valeur de la facette *But-évaluation* pour le concept décrit CC_1. Cette valeur indique un but B_1 permettant de procéder à l'évaluation du concept décrit.
6. L'interpréteur demande au module logiciel qui l'utilise de choisir une procédure pour réaliser B_1. Ce choix peut être fait en fonction de ce que l'apprenant a fait dans le laboratoire virtuel ou bien en fonction d'autres informations si le module tuteur simule un expert, par exemple.
7. L'interpréteur exécute la procédure avec les arguments $a_1, a_2, ...a_n$ et retourne le résultat.

Les buts systèmes *SÉQUENCE* et *TROUVE*

Ces buts systèmes sont accomplis exactement de la même manière qu'auparavant (cf. section 2.2.9).

Le but système *REMPLACE*

Le but *REPLACECOMP* se réalise par le remplacement d'une composante *AncComp* par une nouvelle composante *NouvComp* dans une instance de concept décrit *CognitionDécrite*1. Voici comment le *MGC* réalise ce but système :

1. L'interpréteur récupère les noms des individus nommés *OWL* a, b_{ancien} et $b_{nouveau}$ correspondant aux trois paramètres, en regardant dans leur attribut *RefIndividuOWL* respectif.
2. L'interpréteur obtient la valeur CC_1 associée à *CognitionDécrite*1 dans la *Table des concepts décrits* (cf. page 117).

3. L'interpréteur récupère les noms des rôles $r_1, r_2, ...r_n$ inscrits dans la facette *Composantes* du concept décrit CC_1.
4. Pour chaque rôle r tel que $r \in \{r_1, r_2, ...r_n\}$ et que $r(a, b_{ancien})$ tient dans l'*ABox*, l'interpréteur remplace l'assertion $r(a, b_{ancien})$ par l'assertion $r(a, b_{nouveau})$.

Le but système *COMPOSANTE*

Le but *COMPOSANTE* retourne la cognition qui correspond à l'individu nommé, lié par le rôle *nomRôle*, à l'individu nommé qui représente le concept décrit *CognitionDécrite*1. Les étapes pour accomplir ce but sont :

1. L'interpréteur obtient le nom a de l'individu nommé *OWL* inscrit dans l'attribut *RefIndividuOWL* de *CognitionDécrite*1.
2. L'interpréteur demande au moteur d'inférence le nom de l'individu nommé b tel que l'assertion $nomRôle(a, b)$ tient dans l'abox.
3. L'interpréteur retourne une instance de *Cognition.java* avec l'attribut *RefIndividuOWL* initialisé à la valeur b.

Le but système *ÉQUIVALENT-OWL*

Le but *ÉQUIVALENT-OWL* vérifie si deux cognitions s'équivalent à l'égard de leur description *OWL* :

1. L'interpréteur obtient les valeurs CC_1 et CC_2 associées à *Cognition*1 et *Cognition*2 dans la *Table des concepts décrits* (cf. page 117). Si une seule valeur est retournée, l'interpréteur retourne $faux$, car cela indique qu'une des cognitions est décrite et l'autre primitive.
2. Si les deux cognitions sont primitives (aucune valeur n'est retournée à l'étape précédente), l'interpréteur obtient le nom a et b des individus nommés *OWL* inscrits dans l'attribut *RefIndividuOWL* de *Cognition*1 et *Cognition*2, respectivement. L'interpréteur demande au moteur d'inférence *OWL* le concept *OWL* le plus spécifique

pour a et pour b (nommés C_1 et C_2). Si C_1 et C_2 s'équivalent, l'interpréteur retourne $vrai$, sinon il retourne $faux$.

3. Si les deux cognitions sont décrites, l'interpréteur récupère le nom des rôles $r_1, r_2, ...r_m$ et $s_1, s_2, ...s_n$ inscrits dans la facette *Composantes* des concepts décrits CC_1 et CC_2. Si $m \neq n$, l'interpréteur retourne $faux$. L'interpréteur obtient le nom a et b des individus nommés *OWL* inscrits dans l'attribut *RefIndividuOWL* de $Cognition1$ et $Cognition2$, respectivement. L'interpréteur obtient les individus nommés $a_1, a_2, ...a_n$ et $b_1, b_2, ...b_n$ tels que $r_1(a, a_1), r_2(a, a_2), ...r_m(a, a_n)$ et $s_1(b, b_1), s_2(b, b_2), ...s_n(b, b_n)$ tiennent dans l'*ABox*. Pour chaque paire $(a_i, b_i)|i \in [1, n]$ l'interpréteur recommence à la première étape avec a_i et b_i. Si toutes ces vérifications retournent $vrai$, l'interpréteur retourne $vrai$, sinon il retourne $faux$. De cette manière, l'interpréteur s'assure récursivement que toutes les composantes des concepts décrits s'équivalent.

Seul le but système IF peut utiliser les valeurs $vrai$ et $faux$ retournées. Le but système *ÉQUIVALENT-OWL* est plausible psychologiquement à première vue, car vérifier que deux éléments sont du même type semble être une opération élémentaire indispensable, par exemple, pour savoir appliquer l'idempotence en réduction booléenne $((p \,\&\, (\sim p)) \rightarrow p)$. Ce but est nécessaire, d'autant plus que l'expressivité de *OWL-DL* ne permet pas de représenter logiquement la forme d'expression $(p \,\&\, (\sim p))$. La section 6.2.2 du chapitre 6 élaborera cet exemple.

Le but système *IF*

Le but *IF* effectue la tâche suivante :

1. L'interpréteur récupère la valeur booléenne *valeur* (qui ne peut venir que du but *ÉQUIVALENT-OWL*).

2. Si la valeur *valeur* équivaut $vrai$, le prochain but à exécuter est B_1, sinon le prochain but est B_2. Optionnellement, un auteur peut ne pas spécifier de but B_2. Le résultat du but choisi est retourné.

Ce but semble plausible psychologiquement. Il incarne l'opération élémentaire de faire un choix en fonction d'un résultat d'un épisode antérieur. Ce modèle le propose présentement pour une utilisation conjointe avec *ÉQUIVALENT-OWL*. La section 6.2.2 présentera une utilisation concrète de ce but.

Le choix d'un moteur d'inférence

Le choix d'un moteur d'inférence a nécessité une comparaison des solutions disponibles pour *OWL*. Cette section se base sur la comparaison présentée au chapitre 4.4.3. Ce travail rejette d'emblée les moteurs d'inférence *Hoolet*, *Surnia* et *F-OWL* puisqu'ils offrent des performances médiocres pour des ontologies complexes. Les moteurs *FaCT* et *FaCT++* ont été éliminés puisqu'ils ne supportent pas le raisonnement pour *ABox*, une composante importante pour le modèle (entre autres, pour certains buts systèmes). Les deux candidats restants sont *Pellet* et *Racer*. Chacun détient certains avantages par rapport à l'autre. *Pellet* offre plus de services d'inférence et de modification d'ontologies en mémoire, alors que *Racer* dispose de meilleures performances d'exécution. Concernant un domaine simple, certains préféreront *Pellet* pour ses fonctionnalités, alors qu'ayant trait à des domaines complexes, *Racer* pourrait être préféré pour sa performance. D'autre part, *Pellet* et *Racer* possèdent une expressivité similaire : *Racer* se conforme à $\mathcal{SHIQ}(\mathcal{D})-$ et *Pellet* à $\mathcal{SHIN}(\mathcal{D})$ ou $\mathcal{SHON}(\mathcal{D})$. Les deux supportent approximativement *OWL-DL* le sous-ensemble de *OWL* le plus complet avant le non décidable *OWL-FULL*. L'implantation du modèle proposé pour ce mémoire assure une compatibilité avec *Racer* et *Pellet* afin d'accorder une plus grande liberté aux concepteurs des laboratoires virtuels. Il est important de souligner que le modèle que ce chapitre présente ne dépend pas de la *LD* et du moteur d'inférence choisi, en autant que le moteur d'inférence supporte les principaux services de raisonnement pour *ABox* et *TBox*.

L'indépendance des buts envers les procédures

Puisque que le modèle proposé dans ce mémoire permet la définition des procédures et des buts dans plusieurs fichiers *SPK*, il serait bénéfique, afin de favoriser l'extension et la réutilisation des connaissances, de pouvoir définir dans un fichier f_2 de nouvelles procédures pour un but défini dans un fichier f_1, sans devoir modifier le fichier f_1. Ce type d'extension était impossible avec la mise en oeuvre d'*UV* à cause du fort couplage entre les procédures et les buts : chaque but pointait vers ses procédures par la facette *Procédures* et chaque procédure pointait vers son but par la facette *But*. Pour pallier cette lacune, ce modèle apporte les changements suivants :

- La facette *Procédures* est supprimée des fichiers *SPK*.
- Au chargement en mémoire des fichiers, le *MGC* ajoute dynamiquement une facette *Procédures* à chaque but et y inscrit l'ensemble des procédures en mémoire qui pointent vers ce but par leur facette *But*.

Cette génération dynamique de liens et le déplacement des méthodes *Java* dans des classes représentant les procédures (cf. section 5.2.1) éliminent le couplage des buts vers les procédures (et non inversement) et permettent ainsi d'atteindre l'objectif d'ajouter des procédures à un but sans le modifier.

5.3 Le niveau 3 : les objets d'apprentissage

Le niveau 3 organise les connaissances selon une perspective d'ingénierie pédagogique et de réutilisation des connaissances. Il donne les moyens pour encoder les connaissances de niveau 1 et 2 en objets d'apprentissage. Cette section précise la stratégie utilisée pour intégrer la notion d'objet d'apprentissage au modèle. Les sous-sections 5.3.1, 5.3.2, 5.3.3, 5.3.4 et 5.3.5 décrivent respectivement la nature des objets d'apprentissage, la description des objectifs pédagogiques, la problématique de la distribution, la capacité d'étendre ces objets d'apprentissage sans les modifier et enfin, l'entreposage et la localisation des objets

d'apprentissage dans un dépôt.

5.3.1 La nature des objets d'apprentissage

Un objet d'apprentissage regroupe des connaissances qui permettent d'enseigner un ou plusieurs objectifs pédagogiques (cf. chapitre 3). Puisque les buts, les procédures et les concepts sont très liés par leurs spécifications, le niveau 3 considère comme objets d'apprentissage des ensembles de buts, de procédures et de concepts. Il prend donc comme structure de base pour les objets d'apprentissage les fichiers *SPK* auxquels il apporte quelques ajouts.

5.3.2 La description des objectifs pédagogiques

Selon (Duncan, 2003), un objet d'apprentissage doit comprendre des métadonnées qui spécifient quel(s) objectif(s) pédagogique(s) il permet l'atteinte (cf. chapitre 3). Le niveau 3 ajoute la définition d'objectifs pédagogiques dans l'en-tête des fichiers *SPK*. Dans *UV*, les buts servent à l'élaboration des objectifs pédagogiques (cf. section 2.2.10). Ce modèle reprend cette idée, mais permet également la définition d'objectifs pédagogiques en terme de procédures à maîtriser, car plusieurs procédures peuvent être associées à un but et un auteur peut souhaiter que le système ne se concentre que sur l'enseignement de quelques-unes de ces procédures.

Six facettes décrivent les objectifs pédagogiques :
- **IDObjectifPédagogique** : un identificateur unique ;
- **Métadonnées** : un ensemble de métadonnées (nom, description, auteur, etc.) ;
- **ButsNécessaires** : une liste de buts dont la maîtrise[9] conjointe est nécessaire à l'at-

[9]Formellement, un tuteur peut conclure qu'un apprenant *maîtrise* un but ou une procédure si l'apprenant réalise correctement le but ou applique correctement la procédure un certain nombre de fois avec des problèmes de difficulté variée sans avoir recours à des procédures de type *non valide*.

teinte de l'objectif pédagogique ;

- **ButsÉquivalents** : une liste d'équivalences de buts dont chacune spécifie un but suivi d'un ensemble de buts considérés équivalents par rapport à l'objectif pédagogique, s'ils sont maîtrisés conjointement.
- **ProcéduresNécessaires** : une liste de procédures dont la maîtrise conjointe est nécessaire à l'atteinte de l'objectif pédagogique.
- **ProcéduresÉquivalentes** : une liste d'équivalences de procédures dont chacune spécifie une procédure suivie d'un ensemble de procédures considérées équivalentes par rapport à l'objectif pédagogique, si elles sont maîtrisées conjointement.

Le chapitre 6 présentera des exemples concrets d'objectifs pédagogiques.

5.3.3 La distribution commune des fichiers *OWL*, *SPK* et des classes *Java*

L'organisation des connaissances en paquetages constitue la troisième étape (optionnelle) du cycle de vie des objets d'apprentissage (cf. section 3.3.3). Pour qu'un fichier *SPK* soit utilisable, il faut distribuer conjointement les classes *Java*[10], les fichiers *OWL* et les fichiers *SPK* requis. Pour faciliter la distribution, ce mémoire propose de grouper tous ces fichiers dans des paquetages au format *IMS-CP*, un standard dans le domaine de la formation en ligne (cf. section 3.3.3). Un paquetage *IMS-CP* est un fichier *zip* qui contient un certain nombre de fichiers en plus d'un fichier *imsmanifest.xml* qui agit comme une table des matières et qui peut contenir des métadonnées sur le paquetage ainsi que sur chaque fichier contenu. Les métadonnées d'un paquetage *IMS-CP* peuvent être encodées selon plusieurs standards. Ce mémoire suggère d'utiliser *IMS-METADATA*, une implantation du standard de métadonnées *LOM* (cf. section 3.3.2), car la communauté de la formation en ligne l'emploie largement. Ce standard propose environ 80 éléments pour décrire un document. Pour créer des paquetages *IMS-CP* et les métadonnées qui y sont incluses,

[10]Les fichiers *Java* peuvent être aussi distribués sous leur forme précompilée (les fichiers *CLASS*).

l'auteur de ce mémoire propose d'utiliser le gratuiticiel *RELOAD*[11] pour sa simplicité d'utilisation.

L'implantation du *MGC* réalisée pour ce mémoire prend en entrée des fichiers *IMS-CP* et peut extraire les fichiers *OWL* et *SPK* contenus. Il est important de noter qu'un auteur doit intégrer dans un laboratoire virtuel (un programme *Java*) les classes *Java* référées par les fichiers *SPK* avant de les utiliser avec le *MGC*, car elles contiennent des méthodes *Java* qui représentent des outils que l'apprenant peut manipuler dans le laboratoire. Ces classes y sont donc étroitement liées. Le *MGC* implanté gère également les fichiers *IMS-CP* qui contiennent d'autres fichiers *IMS-CP*, pour faciliter la réutilisation des connaissances. Dans cette éventualité, le *MGC* charge tous les fichiers *SPK* et *OWL* inclus directement dans le paquetage *IMS-CP* principal. Ensuite, le *MGC* chargera les fichiers *SPK* et *OWL* supplémentaires nécessaires dans les sous-paquetages *IMS-CP* (mentionnés dans les facettes des fichiers *SPK* ou *OWL* du paquetage principal). De cette façon, si un fichier *IMS-CP* inclut un second fichier *IMS-CP* pour utiliser seulement quelques-unes des connaissances qui y sont définies, le *MGC* ne chargera pas tous les fichiers *SPK* et *OWL* du second paquetage, mais seulement les fichiers réellement nécessaires. Le processus de chargement est récursif et permet plusieurs niveaux d'inclusion de fichiers *IMS-CP*.

Certaines métadonnées de ce modèle sont indescriptibles avec les standards tels que *IMS-METADATA*. Par exemple, il est impossible de spécifier des objectifs pédagogiques en terme de buts ou de procédures avec *IMS-METADATA* (une particularité du modèle décrit dans ce mémoire). Par conséquent, ce mémoire suggère d'inscrire ces métadonnées particulières dans les facettes *Métadonnées* des fichiers *SPK* (cf. section 5.2.1) et d'utiliser le fichier *imsmanifest.xml* d'un paquetage *IMS-CP* pour décrire les fichiers *SPK* et *OWL* avec des métadonnées standards. Un objet d'apprentissage dans ce modèle est donc un fichier *IMS-CP* qui contient ou demande l'inclusion de fichiers *OWL* et *SPK*, qui inclut possiblement des fichiers *Java* et qui définit au moins un objectif pédagogique.

[11] *RELOAD* : http://www.reload.ac.uk

5.3.4 La propriété d'extension sans altération

Une capacité intéressante pour l'application du concept d'objet d'apprentissage est celle de pouvoir étendre les définitions d'un fichier $f1$ dans un fichier $f2$ sans devoir modifier $f1$. La théorie des objets d'apprentissage ne requiert pas cette propriété, mais elle demeure utile. Cette section explique pourquoi *UV* ne respecte pas cette propriété et comment le modèle proposé s'y conforme.

L'extension par ajout de procédures à un but

Premièrement, puisqu'une procédure est liée à son but et un but à ses procédures, un auteur ne peut pas, avec *UV*, ajouter une procédure sans devoir modifier le but. Le modèle de ce mémoire corrige ce problème avec la solution qui a été présentée à la section 5.2.2, page 125.

L'extension par ajout de concepts à une hiérarchie

Deuxièmement, la relation d'héritage d'*UV* nuit à l'ajout de concepts pour étendre une hiérarchie existante. Puisque chaque concept d'*UV* possède une facette *Superconcepts* et *Sous-concepts* pour définir les liens d'héritage, il est impossible d'ajouter un concept dans une hiérarchie existante, sans devoir modifier des concepts existants. Par exemple, pour une hiérarchie où un concept $ConceptB$ est un sous-concept d'un concept $ConceptC$, un concepteur de contenu ne peut pas ajouter un concept $ConceptD$ tel que $ConceptD$ est plus général que $ConceptB$ mais plus spécifique que $ConceptA$, sans modifier $ConceptA$ ou $ConceptB$.

Ce paragraphe explique comment relever ce deuxième défi avec le modèle proposé dans ce mémoire. Soit un fichier *SPK* f_{spk1} qui définit $ConceptB$ et $ConceptC$, des concepts qui correspondent aux concepts *OWL* B et C, respectivement. Supposons également que le fichier *OWL* f_{owl1} qui définit B et C spécifie l'axiome *OWL* $B \sqsubseteq C$. Soit un fichier

SPK f_{spk2} qui définit un concept $ConceptD$. Le fichier f_{spk2} importe les définitions de f_{spk1} et un nouveau fichier *OWL* f_{owl2}. f_{owl2} importe f_{owl1} par les mécanismes d'extension d'*OWL*. f_{owl2} définit un concept *OWL* D par les axiomes $D \sqsubseteq C$ et $B \sqsubseteq D$. Les fichiers f_{owl1} et f_{spk1} demeurent ainsi intacts.

Cet exemple démontre qu'un auteur peut facilement ajouter un nouveau concept à une hiérarchie existante sans modifier les concepts existants. Il suffit d'ajouter quelques axiomes supplémentaires dans un second fichier *OWL* pour enrichir (et non altérer) la description logique du premier fichier *OWL*. Cette capacité provient non seulement des mécanismes d'*OWL* pour créer des ontologies distribuées, mais découle aussi du fait que le modèle n'inclut pas de classes *Java* générées, représentant les concepts et unies par des liens héritage, comme dans l'implantation d'*UV* (cf. section 5.2.1).

Il importe de noter que tous les buts et procédures applicables à $ConceptC$ définies dans le fichier f_{spk1} deviennent automatiquement applicables à $ConceptD$ sans apporter le moindre changement à leur définition.

L'extension par ajout d'objectifs pédagogiques

Finalement, le modèle permet de définir plusieurs ensembles d'objectifs pédagogiques pour des mêmes connaissances de niveau 2, puisqu'un fichier *SPK* peut demander le chargement d'autres fichiers *SPK*. Le modèle permet aussi de définir des fichiers *SPK* sans objectifs pédagogiques, ou des fichiers avec plusieurs objectifs. Un paquetage *IMS-CP* doit toutefois contenir au moins un fichier *SPK* contenant un objectif pédagogique pour être considéré comme un objet d'appregentissage.

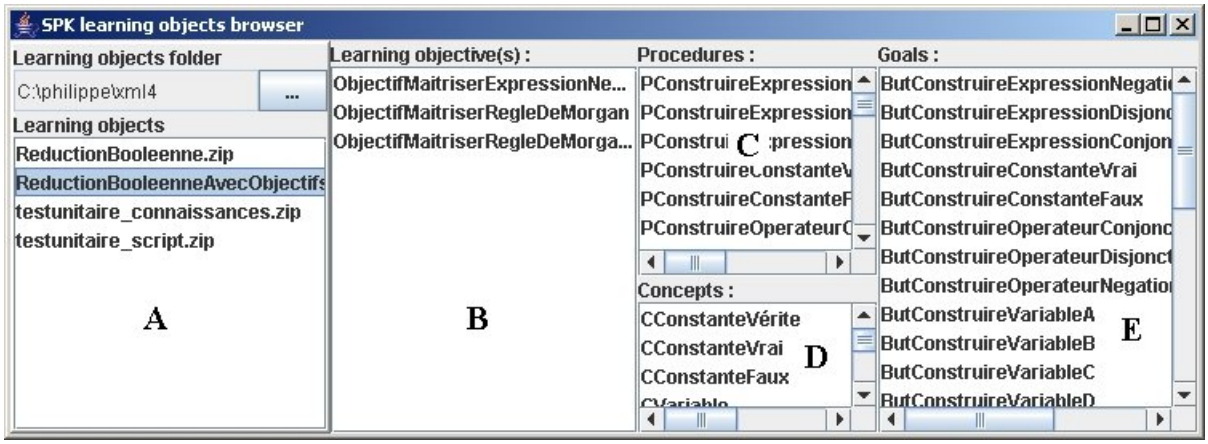

Figure 5.6 – L'explorateur d'objets d'apprentissage *SPK*

5.3.5 L'entreposage et la localisation des objets d'apprentissage *SPK* dans un dépôt

Utiliser des standards de métadonnées et de paquetage (*IMS-METADATA* et *IMS-CP*) pour les objets d'apprentissage permet de stocker les objets d'apprentissage créés selon ce modèle dans des dépôts d'objets d'apprentissage classiques. Néanmoins, pour ce mémoire, un logiciel spécialisé très sommaire nommé *SPK Learning Objects Browser* (cf. figure 5.6) a été conçu pour pouvoir organiser les objets d'apprentissage en collections, les localiser et consulter leurs caractéristiques principales. Dans la partie supérieure gauche de son interface, un bouton permet de sélectionner un répertoire de travail qui contient des fichiers *IMS-CP*. Une liste, située au-dessous du bouton, énumère les fichiers *IMS-CP* dans le répertoire (zone A sur la figure). Pour ajouter un nouvel objet d'apprentissage au dépôt, un concepteur ajoute simplement un fichier *IMS-CP* dans le répertoire. Sur la figure, un utilisateur a sélectionné l'objet d'apprentissage *RéductionBooléenneAvecObjectifs.zip*. Les parties B,C,D,E de l'interface affichent respectivement les objectifs pédagogiques, les procédures, les concepts et les buts compris dans le paquetage *IMS-CP* sélectionné.

5.4 La structuration des objets d'apprentissage en curriculums

La présente section a pour rôle de montrer que le modèle pourrait servir d'assise au développement de cours au sens de *CREAM*. Elle ne vise pas à entreprendre une recherche complète sur cet aspect, mais à prouver que le modèle peut s'étendre et reproduire des structures similaires à celles de *CREAM* (cf. chapitre 1), dont certains éléments peuvent être inférés, alors que dans *CREAM*, les auteurs devaient les spécifier à la main. La section 5.4.1 présente les correspondances qui peuvent être établies avec les entités *CREAM* et la section 5.4.2 propose une méthode pour bâtir des cours, d'une façon similaire à *CREAM*, mais à partir des objets d'apprentissage dynamiques de ce modèle.

5.4.1 La correspondance entre les notions du modèle et celles de *CREAM*

Plusieurs correspondances peuvent être établies entre les principales notions de *CREAM* et celles du modèle proposé. En premier lieu, *CREAM* recourt à la notion de capacité pour asseoir les connaissances d'un domaine. Les capacités peuvent être associées à des outils et objets d'un simulateur à la manière des procédures et concepts d'*ASTUS*.

En second lieu, *CREAM* incorpore la notion d'objectif pédagogique qui trouve son pendant sous la même dénomination dans le modèle décrit dans ce chapitre.

Les ressources didactiques dans le modèle proposé (reprises d'*UV* et des travaux d'Hafsaoui (2005)) associent des méthodes d'enseignement ou de remédiation spécifiques à des connaissances. Elles spécifient généralement une ou des stratégies génériques à utiliser du module tuteur ainsi que la façon de les instancier. Dans *CREAM*, les ressources didactiques constituent le seul moyen d'acquérir des connaissances. Avec le modèle proposé ici, le groupe *ASTUS* souhaite plutôt que les ressources didactiques jouent un rôle de support tutoriel.

En effet, le projet *ASTUS* espère traiter des situations d'apprentissage selon des modes d'enseignement moins contraignants où l'apprenant explore et expérimente ses propres théories, et où l'apprentissage se trouve par conséquent moins structuré.

Finalement, les prochains paragraphes expliquent comment extraire des liens semblables à ceux de *CREAM* à partir d'une base de connaissances qui se conforme au modèle proposé. Puisque Nkambou (1996) ne définit pas formellement la sémantique de la plupart des relations de *CREAM* et que le type de connaissances ainsi que le contexte diffèrent substantiellement, les liens donnés ici peuvent ne pas réfléter sa pensée. L'objectif ici est seulement de montrer que les structures de représentation de connaissances sont suffisamment expressives pour décrire un cours ou un curriculum, en prenant comme exemple, les construits proposé par *CREAM* qui sont souvent cités dans la littérature à cette fin. La démonstration se fait en donnant la formule à évaluer ou l'algorithme à exécuter pour tester un lien. Des recherches ultérieures, dans le groupe *ASTUS*, analyseront la pertinence de ces construits pour encoder un cours ou un curriculum et pourront, s'il y a lieu, en proposer d'autres. Certains pourraient être inférés et d'autres spécifiés explicitement par les auteurs. En définitive, les liens conservés dépendront principalement des besoins du module tuteur.

Les liens entre les capacités (concepts)

Le modèle proposé permet l'extraction des relations suivantes entre des concepts X et Y d'une base de connaissances :

relation d'équivalence : X équivaut Y si les concepts *OWL* associés B et C vérifient la relation $B \equiv C$.

relation d'analogie structurelle : Une analogie structurelle (au niveau de la définition) existe entre X et Y si pour les concepts *OWL* associés B et C, $B \sqcap C$ est satisfiable.

relation d'analogie fonctionnelle : X et Y présentent une *analogie fonctionelle* (du

point de vue de la fonctionnalité) si leur facette *Buts* pointe vers des mêmes buts (dans le cas de concepts primitifs) ou si leur facette *But-évaluation* pointe vers le même but (dans le cas de concepts décrits). La force de l'analogie entre deux concepts primitifs se calcule par le rapport du nombre de buts en commun, sur le nombre total de buts inscrits dans les facettes *Buts* de X et Y.

relation de généralisation : X est plus général que Y si les concepts *OWL* associés B et C vérifient la relation $C \sqsubseteq B$.

relation d'abstraction : Nkambou (1996) définit que « une capacité k_1 est dite plus abstraite qu'une capacité k_2, si k_2 hérite de certains attributs de k_1 ». Pour respecter cette définition, ce mémoire établit que X est plus abstrait que Y si pour les concepts *OWL* associés B et C, $B \sqcap C$ est satisfiable. La définition donnée respecte celle de Nkambou (1996) bien qu'elle n'apparaisse pas intuitive. La relation d'abstraction est définie de la même façon que la relation d'analogie.

relation de déviation : La relation de déviation dans *CREAM* spécifie qu'une capacité pourrait être confondue avec une capacité, qui dans le contexte de la première, mène à l'erreur. Le modèle actuel ne permet pas l'inférence de relations de déviation entre des concepts, ni leur spécification, car il considère tout comme *UV* que seul le choix d'une procédure erronée mène à l'erreur (cf. section 2.2.8). Cet aspect fera l'objet d'améliorations futures.

relation d'agrégation : Un lien d'agrégation existe de X à Y, si X est un concept décrit et que le modèle permet d'utiliser une instance de Y comme composante de X. Formellement, ceci signifie qu'il existe un rôle *OWL* r dans la facette *Composantes* de X, tel que le concept $C \sqcap \exists r.B$ est satisfiable pour les concepts *OWL* B et C associés à X et Y.

Les liens entre les capacités (procédures)

Ce chapitre propose d'inférer les relations suivantes entre des procédures P_1 et P_2 :

relation d'équivalence : P_1 équivaut à P_2 s'ils possèdent les mêmes paramètres et le même comportement (réfèrent à la même méthode *Java* avec les mêmes arguments ou possèdent le même script)[12].

relation d'analogie structurelle : P_1 et P_2 présentent une *analogie structurelle* (au niveau de leur définition) s'ils possèdent des scripts avec des sous-buts communs. La force de l'analogie se calcule par le rapport du nombre de sous-buts avec les mêmes paramètres dans les scripts de P_1 et P_2 sur le nombre total de combinaisons de sous-buts et de paramètres présents dans les scripts de P_1 et P_2.

relation d'analogie fonctionnelle : Une *analogie fonctionnelle* (du point de vue de la fonctionnalité) existe entre P_1 et P_2 si leur facette *But-associé* réfère au même but. Cette relation indique que P_1 et P_2 s'emploient dans le même but.

relation de généralisation : P_1 est plus général que P_2 si les concepts *OWL* qui correspondent aux types de concepts pris en paramètres par P_1 subsument ceux de P_2, et que P_1 et P_2 s'appliquent pareillement (réfèrent à la même méthode *Java* avec les mêmes arguments ou possèdent le même script).

relation d'abstraction : Ce mémoire ne définit pas d'équivalent de la relation d'abstraction pour les procédures.

relation de déviation : Une relation de déviation unit P_1 et P_2 si une analogie fonctionnelle existe entre P_1 et P_2 alors que P_1 est de type *valide*, et P_2 une procédure de type *non valide* (cf. section 2.2.4).

relation d'agrégation forte : P_1 compose une procédure complexe P_2 si chaque séquence de procédures qui réalise les sous-buts de P_2 inclut P_1.

relation d'agrégation faible : P_1 compose faiblement une procédure complexe P_2 s'il n'existe pas de relation d'agrégation forte entre P_1 et P_2 et qu'il existe une séquence de procédures qui réalise les sous-buts de P_2 et inclut P_1.

[12] Cette définition d'équivalence est trop restrictive, mais respecte celle donnée par (Nkambou, 1996). Le lecteur est invité à comparer avec la relation d'analogie fonctionnelle.

Quelques définitions préalables

Voici quelques définitions préalables à la définition des liens entre objectifs pédagogiques de même qu'entre objectifs pédagogiques et capacités :

Réaliser un but s'accomplit par l'application d'une procédure primitive ou complexe dont la facette *But-associé* pointe vers le but en question et pour laquelle les paramètres du but correspondent.

Appliquer une procédure primitive se fait par l'exécution de son action. *Appliquer* une procédure complexe se fait par la réalisation de chacun de ses sous-buts.

Appliquer une procédure complexe P_1 *nécessite d'appliquer* une procédure P_2 si chaque séquence de procédures qui réalise les sous-buts de P_1 inclut P_2.

Réaliser un but B *nécessite d'appliquer* une procédure P si chaque séquence de procédures qui réalise B inclut P.

Appliquer une procédure complexe P *nécessite de réaliser* un but B si chaque séquence de procédures qui réalise les sous-buts de P inclut une procédure qui réalise le but B.

Réaliser un but B_1 *nécessite de réaliser* un but B_2 si chaque séquence de procédures qui réalise B_1 inclut une procédure qui réalise le but B_2.

Un tuteur peut conclure qu'un apprenant *maîtrise* un but ou une procédure si l'apprenant réalise correctement le but ou applique correctement la procédure, un certain nombre de fois avec des problèmes de difficulté variée, sans avoir recours à des procédures de type *non valide*.

Un *ensemble réalisant* d'un objectif pédagogique O est un ensemble de buts et de procédures mentionnés dans les facettes de O dont la maîtrise conjointe permet d'atteindre O (cf. section 5.3.2). Un objectif pédagogique peut avoir plusieurs ensembles réalisants.

Un apprenant *atteint* un objectif pédagogique O s'il maîtrise tous les buts et toutes les procédures présentes dans un des ensembles réalisants de O.

Un ensemble réalisant E *nécessite directement de maîtriser* une procédure P si E inclut P.

Un ensemble réalisant E *nécessite directement de maîtriser* un but B si E inclut B.

Un ensemble réalisant E *nécessite indirectement de maîtriser* une procédure P si réaliser un but ou appliquer une procédure de E nécessite d'appliquer P et que E n'inclut pas P.

Un ensemble réalisant E *nécessite indirectement de maîtriser* un but B si réaliser un but ou appliquer une procédure de E nécessite de réaliser B et que E n'inclut pas B.

Atteindre un objectif pédagogique O *nécessite directement de maîtriser* une procédure P si chaque ensemble réalisant de O nécessite directement de maîtriser P.

Atteindre un objectif pédagogique O *nécessite directement de maîtriser* un but B si chaque ensemble réalisant de O nécessite directement de maîtriser B.

Atteindre un objectif pédagogique O *nécessite indirectement de maîtriser* une procédure P s'il existe un ensemble réalisant de O qui nécessite indirectement de maîtriser P et que chaque autre ensemble réalisant de O nécessite directement ou indirectement de maîtriser P.

Atteindre un objectif pédagogique O *nécessite indirectement de maîtriser* un but B s'il existe un ensemble réalisant de O qui nécessite indirectement de maîtriser B et que chaque autre ensemble réalisant de O nécessite directement ou indirectement de maîtriser B.

Atteindre un objectif pédagogique O *nécessite parfois de maîtriser* une procédure P si atteindre O ne nécessite pas directement ou indirectement de maîtriser P, et qu'au moins un ensemble réalisant de O nécessite indirectement de maîtriser P.

Atteindre un objectif pédagogique O *nécessite parfois de maîtriser* un but B si atteindre O ne nécessite pas de maîtriser B directement ou indirectement, et qu'au moins un ensemble réalisant de O nécessite indirectement de maîtriser B.

Les liens entre les objectifs pédagogiques

Différentes relations se déduisent pour deux objectifs pédagogiques O_1 et O_2 :

relation de préalable obligatoire : O_1 constitue un préalable obligatoire à O_2, si atteindre O_2 nécessite indirectement de maîtriser une procédure ou un but mentionné seulement dans les facettes de l'objectif pédagogique O_1.

relation de préalable souhaitable : O_1 constitue un préalable souhaitable à O_2 si atteindre O_1 n'est pas un préalable obligatoire de O_2 et qu'atteindre O_2 nécessite indirectement de maîtriser une procédure ou un but mentionné dans les facettes de l'objectif pédagogique O_1. Alternativement, O_1 peut être considéré comme un préalable souhaitable de O_2 si atteindre O_2 nécessite parfois de maîtriser une procédure ou un but mentionné dans les facettes de l'objectif pédagogique O_1.

relation de constitution : O_1 constitue un sous-objectif de O_2 si atteindre O_2 nécessite directement ou indirectement de maîtriser tous les éléments d'au moins un des ensembles réalisant de O_1.

relation de prétexte : Ce mémoire ne définit pas d'équivalent pour cette relation, car la définition de Nkambou (1996) se montre trop imprécise.

Les liens entre les ressources didactiques

CREAM définit un ensemble de liens entre les ressources didactiques : similarité, abstraction, cas particulier, utilisation, auxiliariat et équivalence. Ils servent principalement à des fins tutorielles telles que signaler à l'apprenant qu'il a déjà réussi un problème similaire, etc. Ce mémoire n'entrera pas dans les détails de ces relations puisque l'aspect tutoriel ne constitue pas le centre d'intérêt de ce mémoire. Hafsaoui (2005) a élaboré la représentation des ressources didactiques pour *UV*. Il n'inclut pas de liens explicites entre les ressources, mais certains pourraient être déduits ou ajoutés similairement à ce que ce mémoire présente pour les concepts, procédures et objectifs pédagogiques.

Les liens entre les capacités et les objectifs pédagogiques

Le modèle proposé permet l'extraction des relations suivantes entre un concept X ou une procédure P, et un objectif O :

relation de préalable obligatoire (concept) : X est un préalable obligatoire à O si chaque ensemble réalisant de O nécessite indirectement de maîtriser un but ou une procédure qui prend un concept Y en paramètre, tel que $C \sqsubseteq D$ est satisfiable pour C et D, les concepts OWL qui correspondent respectivement à X et Y.

relation de préalable obligatoire (procédure) : P est un préalable obligatoire de O si atteindre O nécessite indirectement de maîtriser P.

relation de préalable souhaitable (concept) : X est un préalable souhaitable à O si atteindre O nécessite parfois de maîtriser une procédure qui prend un concept Y en paramètre, tel que $C \sqsubseteq D$ est satisfiable pour C et D, les concepts OWL associés respectivement à X et Y.

relation de préalable souhaitable (procédure) : P est un préalable souhaitable à O si atteindre O nécessite parfois de maîtriser P.

relation de contribution (concept) : O contribue à l'acquisition de X si chaque ensemble réalisant de O nécessite directement ou indirectement de maîtriser une procédure ou un but qui prend un concept Y en paramètre, tel que $C \sqsubseteq D$ est satisfiable pour C et D, les concepts OWL associés respectivement à X et Y.

relation de contribution (procédure) : O contribue à l'acquisition de P si O nécessite directement ou indirectement de maîtriser P.

Les liens entre les capacités et les ressources didactiques

Diverses relations se déduisent entre un concept X ou une procédure P et une ressource didactique R :

relation de renforcement (concept) : R permet de renforcer X, si X mentionne R dans sa facette *enseignement*.

relation de renforcement (procédure) : R permet de renforcer P si P mentionne R dans l'une de ses facettes parmi *Ressources didactiques*, *Exemples*, *Exercices* et *Test*.

relation de ressource d'enseignement : Cette relation constitue la relation inverse de la relation de renforcement : une relation de ressource d'enseignement existe entre R et un concept ou une procédure, si une relation de renforcement existe du concept ou de la procédure vers R.

Les liens entre les ressources didactiques et les objectifs pédagogiques

Dans *CREAM*, une ressource didactique est soit critique, soit accessoire à l'atteinte d'un objectif pédagogique. Le projet *ASTUS* considère qu'aucune ressource n'est critique parce qu'il accorde un rôle différent aux ressources didactiques. Une relation de contribution lie une ressource didactique R à un objectif pédagogique O si O est en relation avec un concept ou une procédure (par une relation de préalable ou de contribution), et que R en permet le renforcement.

Les liens entre les liens et les ressources didactiques

Les liens entre des liens et des ressources didactiques permettent dans *CREAM* d'associer des ressources didactiques à certains liens afin d'apporter une remédiation (par exemple, pour des liens de type déviation) ou de les enseigner. Ce type de lien n'est pas présent actuellement, mais pourrait être ajouté a posteriori aux liens inférés selon les définitions ci-haut.

5.4.2 La création de curriculums basés sur les objets d'apprentissage d'*ASTUS*

Cette section présente une méthodologie pour spécifier un cours d'une façon analogue à *CREAM* (cf. section 1.1.5). Un curriculum se compose à la base d'un ensemble d'objets d'apprentissage. À partir de cet ensemble, les liens définis pour le modèle peuvent être calculés à l'avance selon les principes dictés par les paragraphes précédents, ou spécifiés explicitement. Ensuite, l'auteur d'un cours doit préciser les *exigences de la formation* et le *public cible*. Cette étape s'inspire directement de *CREAM*. Le public cible se spécifie par l'attribution d'un niveau de maîtrise estimé à chaque connaissance, alors que les exigences de la formation se spécifient par l'association de niveaux de maîtrise exigés à des connaissances procédurales ou sémantiques, ou à des objectifs pédagogiques. La tâche de définir le vocabulaire pour expliciter les niveaux de maîtrise déborde du cadre de ce mémoire et fera partie des recherches qui porteront sur le module tuteur et le module « modèle de l'apprenant ». Avec la formalisation des exigences de la formation et du public cible, un agent tuteur pourrait facilement déduire les connaissances à enseigner par un procédé similaire à *CREAM*.

Puisque le modèle proposé laisse les facettes didactiques d'*UV* inchangées, le module tuteur d'Hafsaoui (2005) demeure compatible. Des adaptations pourraient toutefois être apportées au module tuteur pour tirer profit d'une structure de cours telle que décrite ci-haut, pour déterminer les connaissances à enseigner et afin d'exploiter la présence de liens entre les connaissances de façon à améliorer l'enseignement.

Cette section a montré comment bâtir une structure de cours pour l'enseignement basé sur les objets d'apprentissage développés dans ce mémoire. Des recherches pourraient élaborer davantage cette structure inspirée de *CREAM* , mais son raffinement surpasse les limites de ce travail.

5.5 Discussion et conclusion

Cette section conclut le chapitre par la discussion de quelques points importants et un rappel des avantages les plus considérables du modèle. Les suggestions d'améliorations figurent à la fin du chapitre suivant qui présente une modélisation d'un domaine concret.

5.5.1 L'utilisation de *OWL* pour les niveau 2 ou 3

Ce mémoire a évalué la possibilité d'employer *OWL* pour représenter les niveaux 2 et 3. Cette idée permettrait d'uniformiser les types de fichiers et de bénéficier d'inférence. Le mémoire rejette l'idée pour deux raisons.

Premièrement, *OWL* alourdirait considérablement les fichiers. Présentement, la syntaxe des fichiers *SPK* est compacte, basée sur *XML* et ne contient que les balises nécessaires. Selon un essai effectué pour ce travail, reproduire les mêmes descriptions avec *OWL* demanderait beaucoup de syntaxe superflue, doublant peut-être même la taille des fichiers. Cela complexifierait aussi la création d'un système auteur visuel.

Deuxièmement, les bénéfices de raisonner sur les niveaux 2 et 3 seraient pratiquement inexistants puisque toutes les informations de niveau 2 et 3 peuvent être vérifiées rapidement par une recherche linéaire dans le fichier. Par exemple, obtenir tous les concepts dont la facette *auteur* contient la chaîne de caractères « Philippe Fournier-Viger », ne nécessite pas un raisonnement logique, mais seulement de parcourir linéairement le fichier et de récupérer les concepts qui correspondent.

5.5.2 Les avantages de notre modèle

Les avantages suivants découlent du modèle présenté dans ce chapitre :
– **Abstraction** : La séparation en trois niveaux augmente la capacité de réutilisation,

les fichiers *SPK* et *OWL* se réutilisent indépendamment. Ils peuvent être conçus par des personnes différentes qui ne possèdent pas une compréhension de l'ensemble des niveaux ; le modèle permet de considérer une seule perspective à la fois (logique, psychologique, pédagogique). De plus, une fois les connaissances de niveaux 1 définies, moins d'information doit être prise en compte pour la réalisation des connaissances de niveau 2.

- **Objets d'apprentissage** : L'encodage en objets d'apprentissage favorise la structuration des connaissances en unités qui, par leur forme, facilitent la réutilisation des connaissances dans plusieurs laboratoires virtuels. Elles peuvent être organisées en dépôt pour rendre plus facile leur entreposage et leur localisation (cf. section 5.3.5). L'intégration au modèle du standard de métadonnée *IMS-METADATA* et du standard de paquetage *IMS-CP* assure une compatibilité avec de nombreux outils et des dépôts d'objets d'apprentissage existants.
- **L'extensibilité :** Le modèle favorise l'extension de fichiers *SPK* sans modifier les fichiers originaux, un apport positif pour la réutilisation des connaissances (cf. section 5.3.4).
- **Compatibilité** *OWL* : Les auteurs de laboratoires virtuels peuvent réutiliser des ontologies *OWL* existantes ou partager les leurs.
- **Systèmes auteurs** : Plusieurs outils d'édition visuels pour *OWL* sont disponibles. Un auteur peut utiliser ces outils pour modéliser les connaissances de niveau 1.
- **Ingénierie ontologique** : Pour l'ingénierie pédagogique, l'inférence permet de détecter les inconsistances dans les fichiers *OWL* et d'inférer les connaissances implicites pour s'assurer que le sens donné est bien celui voulu. Par exemple, le système auteur *Protégé* (Knublauch et al., 2004) offre cette fonction.
- **Fondation logique solide** : Graçe aux *LD*, le modèle dispose d'une fondation logique solide. Le modèle élimine les aspects flous de la sémantique d'*UV*, tout en permettant la définition de liens logiques plus riches entre les concepts, à l'aide des nombreux constructeurs des *LD*.

- **Sémantique précise et succincte** : Le modèle réduit considérablement le nombre de facettes de la spécification, tout en conservant les points forts d'*UV*.
- **Moteur d'inférence efficace** : Le modèle dispose de moteurs d'inférence efficaces pour raisonner sur l'aspect logique. Présentement seul le *MGC* nécessite l'inférence, mais d'autres modules pourraient éventuellement exploiter cette capacité.
- **Extraction de liens entre les connaissances** : Ce mémoire a décrit un procédé pour extraire de nombreux liens similaire à *CREAM* (cf. section 5.4) qui peuvent être utilisés à des fins tutorielles. L'extraction de ces liens se fait par inférence, alors que dans *CREAM* un auteur doit spécifier la plupart de ces liens manuellement.
- **Structuration de cours et de curriculum** : Le modèle constitue une base intéressante pour la structuration de cours en terme d'objets d'apprentissage dans les systèmes tutoriels intelligents (cf. section 5.4).

CHAPITRE 6

L'expérimentation

Ce chapitre décrit la validation expérimentale du modèle proposé pour un domaine concret. Le chapitre s'organise comme suit : section 6.1 présente le laboratoire de réduction booléenne, la section 6.2 décrit brièvement la modélisation réalisée et la section 6.3 dresse les conclusions de cette expérimentation.

6.1 Le laboratoire virtuel de la réduction booléenne

La figure 6.1 illustre l'interface du laboratoire virtuel de réduction booléenne. Bouchard (2005) a construit cette interface, se basant sur une première version du laboratoire développée par Najjar et Mayers (2004).

Une séance d'exercice consiste à résoudre une série de problèmes de réduction booléenne. La figure 6.1 montre un exemple de problème : l'expression $((a \mid V) \,\&\, (b \,\&\, V))$. Les lettres minuscules représentent les variables. Les symboles V, F, $\&$, $\mid$ et $\sim$ représentent respectivement la constante vrai, la constante faux, l'opérateur de conjonction, l'opérateur de disjonction et l'opérateur de négation.

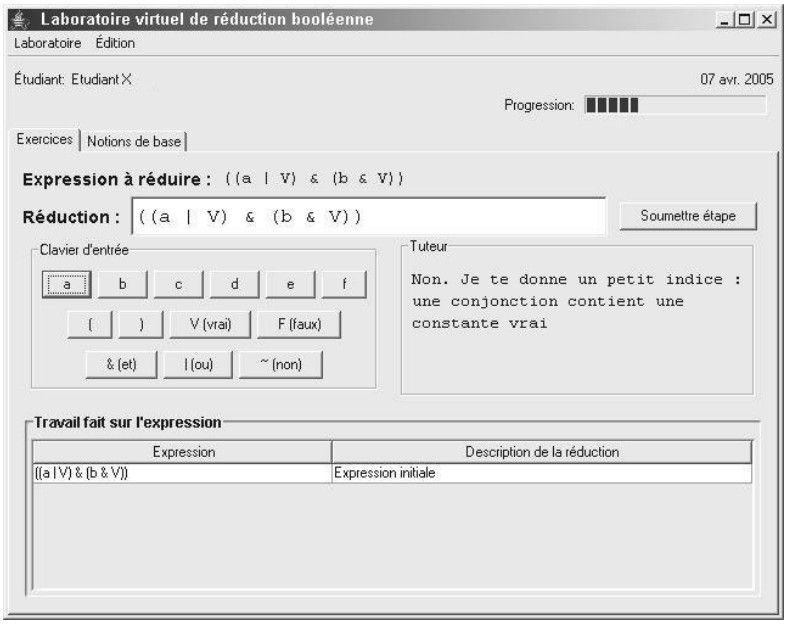

Figure 6.1 – L'interface du laboratoire virtuel de réduction booléenne

Un apprenant peut sélectionner une partie de l'expression dans le champ *Réduction* et la modifier au moyen d'un clavier réel ou du clavier virtuel proposé. Une fois l'expression modifiée à son goût, l'apprenant doit cliquer sur le bouton *Soumettre étape* pour valider le ou les changements. Dans la partie du bas de la fenêtre, l'apprenant peut voir l'historique des règles qu'il a appliquées dans l'exercice courant. Dans le coin supérieur droit, un indicateur affiche la progression globale de la séance d'exercice. L'interface réserve une zone à droite du clavier virtuel pour l'interaction avec le tuteur. Présentement, le module tuteur fournit une rétroaction limitée, mais des recherches se poursuivent pour améliorer cet aspect. Un second onglet nommé *Notions de base* permet d'accéder aux notions théoriques du laboratoire en tout temps. L'apprenant peut consulter la définition de chacune des 13 règles de réduction par le biais d'un menu déroulant.

6.1.1 La détection des procédures primitives et complexes

L'interface permet à l'apprenant d'appliquer plusieurs règles avant de soumettre une étape. D'autre part, plusieurs règles donnent parfois le même résultat. Par exemple, si l'apprenant donne la réponse (V) pour le problème $(V \ \& \ V)$, il peut avoir appliqué la règle d'idempotence $((a \ \& \ a) \equiv (V)$, la règle de réduction d'une conjonction avec le vrai à gauche $((V \ \& \ a) \equiv (V))$, la règle de réduction d'une conjonction avec le vrai à droite $((a \ \& \ V) \equiv (V))$ ou toutes autres règles erronées qui donnent ce résultat.

Le mandat de déterminer les procédures primitives et complexes que l'apprenant a bel et bien exécutées revient au module « modèle de l'apprenant », un module en développement. Il répondra à ces besoins avec une certaine certitude, en se basant sur les habitudes, l'historique (mémoire épisodique) et les préférences de l'apprenant.

6.2 La modélisation des connaissances

Le travail de modélisation des connaissances pour ce laboratoire consiste à représenter : (1) les concepts des expressions booléennes, (2) les outils offerts dans le laboratoire pour manipuler ces expressions booléennes (procédures) et (3) modéliser les connaissances didactiques. Puisque Hafsaoui (2005) a déjà représenté le troisième point (les connaissances didactiques de la réduction booléenne) et que l'amélioration des facettes pédagogiques et didactiques ne relève pas des objectifs de ce mémoire, ce travail de validation ne vise que les deux premiers points. Contrairement à la modélisation des connaissances de réduction booléenne réalisée par Najjar et al. (Najjar et al., 2004b; Najjar et Mayers, 2004; Najjar et al., 2004a) avec la première implantation d'*UV*, la modélisation décrite dans ce chapitre inclut la totalité des 13 règles de réduction booléenne (loi de De Morgan, idempotence, commutativité, distributivité, etc.). Les sections 6.2.1, 6.2.2 et 6.2.3 décrivent respectivement les connaissances de niveau 1, 2 et 3 pour le laboratoire de réduction booléenne.

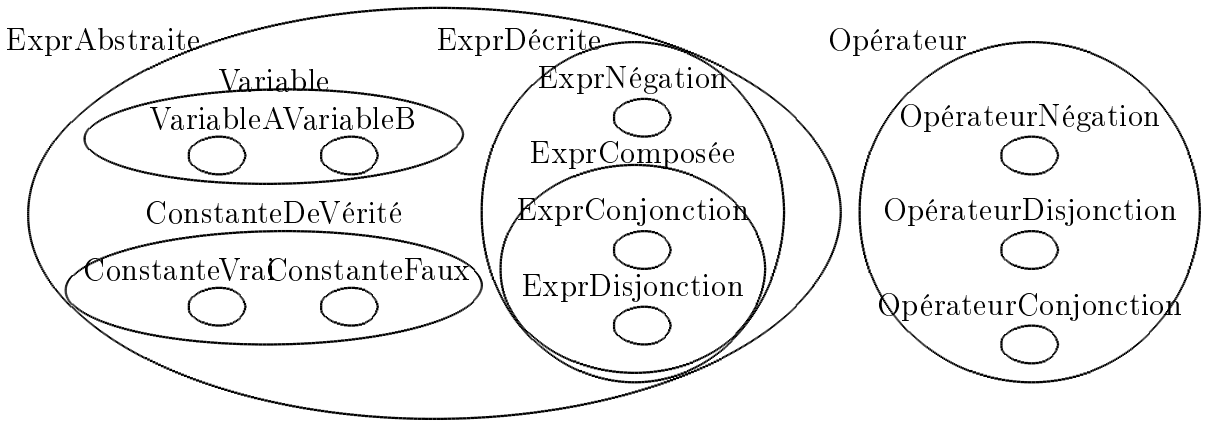

Figure 6.2 – Les principaux concepts *OWL* de niveau 1 pour le laboratoire de réduction booléenne

6.2.1 Les connaissances de niveau 1

Le niveau 1 représente les connaissances du laboratoire d'un point de vue logique, dans un fichier nommé *redbool.owl*. Ces connaissances pourraient être réparties dans plusieurs fichiers *OWL* en utilisant le mécanisme d'extension des fichiers *OWL*, proposé dans *OWL*, ou en utilisant la facette *FichiersOWL* des fichiers *SPK*, pour demander le chargement de plusieurs fichiers *OWL*.

La figure 6.2 montre au moyen de la notation des ensembles les principaux concepts *OWL* de la modélisation. La notion d'*expression abstraite* symbolise l'ensemble des expressions booléennes valides, c'est-à-dire les variables, les constantes de vérité et les expressions décrites. Le terme *expressions décrites* dénomme les expressions booléennes à un opérateur (les expressions de négation et les expressions composées). Le terme *expressions composées* dénote les expressions à deux opérandes (les expressions de conjonction et les expressions de disjonction). Le concept d'opérateur constitue un concept *OWL* distinct de celui d'expression booléenne. Il regroupe les opérateurs de négation, les opérateurs de disjonction et les opérateurs de négation.

Les prochaines sous-sections présentent les axiomes qui définissent ces concepts ainsi

que quelques autres. Ce chapitre emploie la syntaxe des *LD* plutôt que celle d'*OWL* pour faciliter la compréhension. Le modèle se conforme à *OWL-DL*, le sous-ensemble d'*OWL* le plus complet avant le non décidable *OWL-FULL*.

Les variables

Les axiomes suivants définissent les variables :

Les variables sont des expressions abstraites.
$Variable \sqsubseteq ExprAbstraite$
Les variables nommées a sont des variables.
$VariableA \sqsubseteq Variable$
Les variables nommées b sont des variables.
$VariableB \sqsubseteq Variable$
Les variables nommées a ne peuvent être nommées b.
$(VariableA \sqcap VariableB) \sqsubseteq \bot$
Les variables ne sont pas des constantes de vérité.
$(Variable \sqcap ConstanteDeVérité) \sqsubseteq \bot$
Les variables ne sont pas des expressions décrites.
$(Variable \sqcap ExprDécrite) \sqsubseteq \bot$

La modélisation définit un concept *OWL* pour chaque type de variable ($VariableA$ et $VariableB$), puisque le type d'une variable importe parfois dans la réduction des expressions booléennes (par exemple, $a \wedge a \equiv a \not\equiv a \wedge b$). La présentation se limite à deux variables pour éviter les répétitions.

Les constantes de vérité

Les axiomes suivants définissent les constantes de vérité :

Les constantes de vérité sont des expressions abstraites.
$ConstanteDeVérité \sqsubseteq ExprAbstraite$

Les constantes faux sont des constantes de vérité.

$ConstanteFaux \sqsubseteq ConstanteDeVérité$

Les constantes vrai sont des constantes de vérité.

$ConstanteVrai \sqsubseteq ConstanteDeVérité$

Les constantes faux ne sont pas des constantes vrai.

$(ConstanteFaux \sqcap ConstanteVrai) \sqsubseteq \bot$

Les constantes de vérité ne sont pas des expressions décrites.

$(ConstanteDeVérité \sqcap ExprDécrite) \sqsubseteq \bot$

Les opérateurs

Les axiomes suivants définissent les opérateurs :

Les opérateurs de négation sont des opérateurs.

$OpérateurNégation \sqsubseteq Opérateur$

Les opérateurs de disjonction sont des opérateurs.

$OpérateurDisjonction \sqsubseteq Opérateur$

Les opérateurs de conjonction sont des opérateurs.

$OpérateurConjonction \sqsubseteq Opérateur$

Les opérateurs de conjonction ne sont pas des opérateurs de disjonction.

$(OpérateurConjonction \sqcap OpérateurDisjonction) \sqsubseteq \bot$

Les opérateurs de conjonction ne sont pas des opérateurs de négation.

$(OpérateurConjonction \sqcap OpérateurNégation) \sqsubseteq \bot$

Les opérateurs de disjonction ne sont pas des opérateurs de négation.

$(OpérateurDisjonction \sqcap OpérateurNégation) \sqsubseteq \bot$

Les opérateurs ne sont pas des expressions booléennes valides.

$(Opérateur \sqcap ExprAbstraite) \sqsubseteq \bot$

Les opérateurs entrent en relation avec les expressions décrites par les rôles suivants :

Le rôle qui relie les expressions composées à leur opérande de droite.

$opérandeDroiteComp \sqsubseteq \top_R$

Le rôle qui relie les expressions composées à leur opérande de gauche.

$opérandeGaucheComp \sqsubseteq \top_R$

Le rôle qui relie les expressions composées à leur opérateur.

$opérateur \sqsubseteq \top_R$

Le rôle qui relie les expressions de négation à leur opérande.

$opérandeNégation \sqsubseteq \top_R$

Les expressions décrites

Les axiomes suivants définissent les expressions décrites :

Les expressions décrites sont des expressions booléennes à un opérateur.

$ExprDécrite \sqsubseteq \Big(ExprAbstraite \sqcap (= n.opérateur\ 1) \sqcap \forall opérateur.Opérateur\Big)$

Les expressions décrites sont soit des expressions de négation, soit des expressions composées.

$ExprDécrite \sqsubseteq (ExprAbstraite \sqcup ExprComposée)$

Les expressions de négation

L'axiome suivant définit les expressions de négation :

Les expressions de négation sont des expressions décrites qui possèdent un opérateur de négation, une expression abstraite sous la portée de l'opérateur de négation et aucun opérande de gauche, ni de droite au sens des expressions composées.

$ExprNégation \sqsubseteq \Big(ExprDécrite \sqcap (= n.opérandeNégation\ 1) \sqcap (= n.opérandeGaucheComp\ 0) \sqcap$
$(= n.opérandeDroiteComp\ 0) \sqcap (\forall opérateur.OpérateurNégation)$
$\sqcap (\forall opérandeNégation.ExprAbstraite)\Big)$

Les expressions composées

L'axiome suivant définit les expressions composées :

> Les expressions composées possèdent un opérande droite et un opérande gauche, aucun opérande sous la portée d'un opérateur de négation et les opérandes sont des expressions abstraites.

$$ExprComposée \equiv \Big(ExprDécrite \sqcap (= n.opérandeDroite\ 1) \sqcap (= n.opérandeGauche\ 1) \sqcap (= n.opérandeNégation\ 0) \sqcap (\forall opérandeDroiteComp.ExprAbstraite) \sqcap (\forall opérandeGaucheComp.ExprAbstraite)\Big)$$

Les expressions de disjonction

L'axiome suivant définit les expressions de disjonction :

> Les expressions de disjonction sont des expressions composées pour lesquelles l'opérateur est un opérateur de disjonction.

$$ExprDisjonction \equiv (ExprComposée \sqcap \forall opérateur.OpérateurDisjonction)$$

Les expressions de conjonction

L'axiome suivant définit les expressions de conjonction :

> Les expressions de conjonction sont des expressions composées pour lesquelles l'opérateur est un opérateur de conjonction.

$$ExprConjonction \equiv (ExprComposée \sqcap \forall opérateur.OpérateurConjonction)$$

Les autres concepts

Le fichier *redbool.owl* contient plusieurs autres concepts pour modéliser les concepts pris en paramètres par certaines règles de réduction booléenne. Les prochaines lignes présentent les axiomes qui définissent quelques-uns de ces concepts.

Les expressions de négation qui portent sur une conjonction sont des expressions de négation dont l'opérande est une expression de conjonction.

$ExprNégationConjonction \equiv (ExprNégation \sqcap \forall opérandeNégation.ExprConjonction)$

Les doubles négations sont des expressions de négation qui portent sur une expression de négation.

$ExprDoubleNégation \equiv (ExprNégation \sqcap \forall opérandeNégation.ExprNégation)$

Les expresions de négation du vrai sont des expressions de négation qui portent sur la constante de vérité vrai.

$ExprNégationVrai \equiv (ExprNégation \sqcap \forall opérandeNégation.ConstanteVrai)$

Les expressions composées avec une négation comme opérande de droite.

$ExprComposéeNégation2 \equiv (ExprComposée \sqcap \forall opérandeDroite.ExprNégation)$

6.2.2 Les connaissances de niveau 2

Le niveau 2 représente les concepts selon une perspective psychologique et ajoute les procédures et les buts. Les prochaines sous-sections décrivent l'en-tête, les concepts primitifs, les concepts décrits, les buts et les procédures définies pour le laboratoire de réduction booléenne. Ces derniers éléments sont encodés dans le fichier *RéductionBooléenne.spk*.

L'en-tête du fichier *RéductionBooléenne.spk*

L'en-tête du fichier *RéductionBooléenne.spk* (figure 6.3) précise que le niveau 2 se base sur le fichier *OWL redbool.owl* et incorpore un ensemble de métadonnées telles que le nom de l'auteur du fichier, etc.

Les concepts primitifs

La modélisation contient 11 concepts primitifs : *CConstanteVérite*, *CConstanteVrai*, *CConstanteFaux*, *CVariable*, *CVariableA*, *CVariableB*, *COpérateurConjonction*, *COpérateur-*

FichiersOWL	redbool.owl : version 1.0
FichiersSPK	
Métadonnées générales	Auteur : Philippe Fournier-Viger Version : 1.0 Organisation : Université de Sherbrooke Description : La modélisation des concepts et des règles de la réduction booléenne.

Figure 6.3 – L'en-tête du fichier *RéductionBooléenne.spk*

IDconcept	CConstanteVrai
RéférenceOWL	*ConstanteVrai*
Métadonnées	Auteur : Philippe Fournier-Viger Description : Le concept de constante vrai.
Buts	
Constructeurs	PConstruireConstanteVrai
IDconcept	CVariableA
RéférenceOWL	*VariableA*
Métadonnées	Auteur : Philippe Fournier-Viger Description : Le concept de variable nommée *a*.
Buts	
Constructeurs	PConstruireVariableA

Figure 6.4 – La définitions des concept *CConstanteVrai* et *CVariableA*

Disjonction, *COpérateurNégation*, *CExpressionAbstraite* et *CExpressionDécrite*. À titre d'exemple, la figure 6.4 présente *CConstanteVrai* et *CVariableA*. Leur définition est triviale.

Les concepts décrits

La modélisation définit 44 concepts décrits tels que *CExpressionComposée*, *CExpressionConjonction*, *CExpressionDisjonction*, *CExpressionNégation*, *CExpressionNégationFaux*, *CExpressionNégationVrai*, *CExpressionNégationConjonction*, *CExpressionDoubleNégation*. La description de *CExpressionNégationVrai*, *CExpressionDoubleNégation*, *CExpressionNégationConjonction* et *CExpressionComposéeNégation2* suit à la figure 6.5.

IDconcept	CExpressionNégationVrai
RéférenceOWL	*ExprNégationVrai*
Composante	*opérateur*, *opérandeNégation*
But-évaluation	ButRéduireExpressionAbstraite
Métadonnées	Auteur : Philippe Fournier-Viger
	Description : Le concept d'expression de négation du vrai ($\sim V$).
Buts	ButRéduireExpressionAbstraite
Constructeurs	
IDconcept	CExpressionDoubleNégation
RéférenceOWL	*ExprDoubleNégation*
Composante	*opérateur*, *opérandeNégation*
But-évaluation	ButRéduireExpressionAbstraite
Métadonnées	Auteur : Philippe Fournier-Viger
	Description : Le concept de la négation d'une expression de négation ($\sim (\sim p)$).
Buts	ButRéduireExpressionAbstraite
Constructeurs	
IDconcept	CExpressionNégationConjonction
RéférenceOWL	*ExprNégationConjonction*
Composante	*opérateur*, *opérandeNégation*
But-évaluation	ButRéduireExpressionAbstraite
Métadonnées	Auteur : Philippe Fournier-Viger
	Description : Le concept de la négation d'une expression de conjonction ($\sim (p \,\&\, q)$).
Buts	ButRéduireExpressionAbstraite
Constructeurs	
IDconcept	CExpressionComposéeNégation2
RéférenceOWL	*ExprComposéeNégation*2
Composante	*opérateur*, *opérandeGauche*, *opérandeDroite*
But-évaluation	ButRéduireExpressionAbstraite
Métadonnées	Auteur : Philippe Fournier-Viger
	Description : Le concept d'expression de la forme ($p \,\&\, (\sim q)$).
Buts	ButRéduireExpressionAbstraite
Constructeurs	

Figure 6.5 – La description de *CExpressionNégationVrai*, *CExpressionDoubleNégation*, *CExpressionNégationConjonction* et *CExpressionComposéeNégation2*

Les procédures

Les procédures se déclinent en deux catégories : les procédures valides et les procédures parfois non valides. Les procédures parfois non valides se modélisent fondamentalement de la même manière, à la différence de quelques facettes remplies différemment (par exemple, la facette *Validité*). Ce chapitre met l'accent sur les procédures valides puisque le but ne consiste pas à avoir une modélisation exhaustive, mais à valider l'idée générale du modèle. Il est à noter qu'une stratégie décrite dans (Najjar et al., 2004b) et non présentée dans ce mémoire permet d'éviter de définir explicitement toutes les procédures erronées d'un domaine.

La modélisation contient une cinquantaine de procédures : des procédures pour construire les concepts (par exemple, *PConstruireExpressionNégation*), des procédures pour manipuler les expressions décrites (par exemple, *PObtenirOpérandeNégation*) et des procédures pour les différentes règles de réduction (par exemple, *PRéduireExpressionNégationVrai*). Cette section présente les procédures *PConstruireConstanteVrai*, *PRéduireExpressionNégationVrai*, *PAppliquerDeMorganConjonction* et *PVérifierApplicabilitéComplémentarité* (voir figure 6.6).

La modélisation associe le but *ButRéduireExpressionAbstraite* à chacune des procédures qui correspondent à une règle de réduction (*PRéduireExpressionNégationVrai*, *PAppliquerDeMorganConjonction*, etc.). Le but *ButRéduireExpressionAbstraite* constitue le but général de faire une étape de résolution.

La modélisation ignore les procédures liées aux modules perceptuels (lire l'expression booléenne sur l'écran) ainsi que les procédures de prise de décision (décider de la prochaine réduction), car l'interface usager ne permet pas de les percevoir. Élaborer une stratégie pour représenter ces processus et les simuler d'une façon réaliste, demandent un travail de recherche considérable. De futurs travaux porteront sur ces aspects, qui constituent une extension de ce travail. Dans l'immédiat, ce mémoire adopte la simplification que chaque concept décrit à réduire est créé préalablement avec les procédures disponibles,

IDProc	PConstruireConstanteVrai
Métadonnées	Auteur : Philippe Fournier-Viger
	Description : La procédure pour construire une instance de V.
But	ButConstruireConstanteVrai
Arguments	
Type	Primitive
Script	(CONSTRUIRE-PRIMITIF "CConstanteVrai")
IDProc	PRéduireExpressionNégationVrai
Métadonnées	Auteur : Philippe Fournier-Viger
	Description : La procédure pour réduire la négation du vrai $(\sim V) \rightarrow (F)$.
But	ButRéduireExpressionAbstraite
Arguments	(CExpressionNégationVrai exp)
Type	Primitive
Script	(CONSTRUIRE-PRIMITIF "CConstanteFaux")
IDProc	PAppliquerDeMorganConjonction
Métadonnées	Auteur : Philippe Fournier-Viger
	Description : La procédure pour appliquer la règle de DeMorgan
	pour la forme qui comprend une conjonction $(\sim (p \ \& \ q)) \rightarrow (\sim p \mid \ \sim q)$.
But	ButRéduireExpressionAbstraite
Arguments	(CExpressionNégationConjonction exp)
Type	Complexe
Script	(ButConstruireExpressionDisjonction (ButConstruireOpérateurDisjonction) (ButConstruireExpressionNégation (ButConstruireOpérateurNégation) (ButObtenirOpérandeGaucheExpressionComposée (ButObtenirOpérandeExpressionNégation exp))) (ButConstruireExpressionNégation (ButConstruireOpérateurNégation) (ButObtenirOpérandeDroiteExpressionComposée (ButObtenirOpérandeExpressionNégation exp))))

Figure 6.6 – La description des procédures *PConstruireConstanteVrai*, *PRéduireExpressionNégationVrai* et *PAppliquerDeMorganConjonction*

ne se souciant pas des contraintes et spécificités des mécanismes de perception.

Les procédures spécifient le type des paramètres qu'elles nécessitent avec un concept pour chaque argument. Par exemple, la procédure *PRéduireExpressionNégationVrai* prend en paramètre une cognition de type *CExpressionNégationVrai*. Pour chaque argument correspond une description logique précise du paramètre demandé. Toutefois, l'expressivité de *OWL-DL*[1] ne permet pas de représenter logiquement certaines formes d'expression telles que $(p \ \& \ (\sim \ p))$. Ce qui rend ainsi inexpressible cette forme d'expression est que le p de gauche et le p de droite peuvent être des expressions booléennes complexes à plusieurs niveaux d'imbrication telles que $(a \mid \sim (b \ \& \ c))$. Pour modéliser le concept $(p \ \& \ (\sim \ p))$, il faudrait pouvoir exprimer que le p de gauche et le p de droite appartiennent aux mêmes concepts, que s'ils sont en relation avec des individus, ces individus appartiennent aux mêmes concepts, et ceci récursivement, pour les individus en relation avec ces individus, etc. (car une expression peut contenir des sous-expressions). En fait, la seule façon de modéliser le concept $(p \ \& \ (\sim \ p))$ avec *OWL-DL* serait d'énumérer l'infinité de cas possibles $((V \ \& \ (\sim \ V))$, $(F \ \& \ (\sim \ F))$, $(a \ \& \ (\sim \ a))$, $((a \mid \sim (b \ \& \ c)) \ \& \ (\sim \ (a \mid \sim (b \ \& \ c))))$, etc.). Pour pallier cette limite, la modélisation ajoute des procédures telles que *PVérifierApplicabilitéComplémentarité* qui effectuent la vérification de l'applicabilité de la règle avant d'activer un autre sous-but pour l'appliquer (par exemple, *BAppliquerComplémentarité*). Pour cet exemple, *PVérifierApplicabilitéComplémentarité* fait appel au but système *ÉQUIVALENT-OWL* (cf. section 5.2.2) pour vérifier que la composante p et la composante q s'équivalent à l'égard de la logique pour une expression de la forme $(p \ \& \ (\sim \ q))$. La figure 6.7 illustre les principales facettes de *PVérifierApplicabilité-Complémentarité*. Le concept *CExpressionComposéeNégation2* (cf. figure 6.5) pris en paramètre par cette procédure représente un concept de la forme $(p \ \& \ (\sim \ q))$. Ainsi le niveau 2, peut compenser les limites d'expressivité du niveau 1.

[1] Ce travail se prive de l'expressivité supplémentaire de *OWL-FULL* car il peut rendre les problèmes d'inférence indécidables.

IDProc	PVérifierApplicabilitéComplémentarité
Métadonnées	Auteur : Philippe Fournier-Viger Description : Cette procédure vérifie l'applicabilité de la règle de la complémentarité.
But	ButRéduireExpressionAbstraite
Arguments	(CExprComposéeNégation2 exp)
Type	Complexe
Script	(IF (ÉQUIVALENT-OWL (ButObtenirOpérandeGaucheExpressionComposée exp) (ButObtenirOpérandeExpressionNégation (ButObtenirOpérandeDroiteExpressionComposée exp))) (BAppliquerComplémentarité)

Figure 6.7 – La description de la procédure *PVérifierApplicabilitéComplémentarité*

Les buts

Ce travail modélise 16 buts : *ButConstruireExpressionNégation*, *ButConstruireExpressionDisjonction*, *ButConstruireExpressionConjonction*, *ButConstruireConstanteVrai*, *ButConstruireConstanteFaux*, *ButConstruireOpérateurConjonction*, *ButConstruireOpérateurDisjonction*, *ButConstruireOpérateurNégation*, *ButConstruireVariableA*, *ButConstruireVariableB*, *ButRemplacerComposanteExpressionComposée*, *ButRemplacerComposanteExpressionNégation*, *ButObtenirOpérandeGaucheExpressionComposée*, *ButObtenirOpérandeDroiteExpressionComposée*, *ButRéduireExpressionAbstraite*, *ButObtenirOpérandeExpressionNégation*.

Cette section décrit les buts *ButConstruireExpressionNégation* et *ButRéduireExpressionAbstraite* (voir figure 6.8). Le but *ButConstruireExpressionNégation* signifie l'intention de construire une expression de négation à partir d'un opérateur de négation et d'une expression booléenne, alors que le but *ButRéduireExpressionAbstraite* représente le but d'appliquer une étape de réduction.

IDBut	ButConstruireExpressionNégation
Métadonnées	Auteur : Philippe Fournier-Viger
	Description : Le but de construire une expression de négation.
Paramètres	COpérateurNégation, CExpressionAbstraite
IDBut	ButRéduireExpressionAbstraite
Métadonnées	Auteur : Philippe Fournier-Viger
	Description : Le but de réduire une expression booléenne.
Paramètres	CExpressionAbstraite

Figure 6.8 – La description des buts *ButConstruireExpressionNégation* et *ButRéduireExpressionAbstraite*

6.2.3 Les connaissances de niveau 3

Au niveau 3, le fichier *SPK RéductionBooléenne.spk* définit plusieurs objectifs pédagogiques. La figure 6.9 en présente deux. L'objectif *ObjectifMaîtriserExpressionNégation* porte sur la maîtrise d'une connaissance sémantique. Il énonce les critères pour affirmer qu'un étudiant maîtrise le concept *CExpressionNégation*. Pour ceci, l'apprenant doit pouvoir appliquer un certain nombre de procédures qui manipulent ce concept telles que *PAppliquerDeMorganConjonction*. Le second objectif (*ObjectifMaîtriserRègleDeMorgan*) porte sur l'acquisition d'un savoir-faire. Il énumère un certain nombre de procédures à maîtriser pour savoir appliquer la règle de De Morgan, une connaissance strictement procédurale.

Un paquetage de format *IMS-CP* nommé *RéductionBooléenne.zip* regroupe les connaissances de niveau 1, 2 et 3 pour le laboratoire de réduction booléenne. Il contient les fichiers *RéductionBooléenne.spk*, *Redbool.owl* et *imsmanifest.xml*. Ce dernier fichier renferme les métadonnées globales sur le paquetage et des métadonnées spécifiques pour chaque fichier contenu (cf. section 5.3.3).

Le fichier *RéductionBooléenne.zip* constitue un objet d'apprentissage dynamique ; un ensemble de connaissances réutilisable et extensible, qui porte sur des connaissances sémantiques et procédurales, et qui décrit les objectifs pédagogiques dont il permet l'atteinte.

IDObjectifPédagogique	ObjectifMaîtriserExpressionNégation
Métadonnées	Auteur : Philippe Fournier-Viger Description : L'objectif pédagogique de maîtriser la connaissance sémantique d'expression de négation.
ButsNécessaires	
ButsÉquivalents	
ProcéduresNécessaires	PAppliquerDeMorganConjonction, PRéduireExpressionNégationVrai, PRéduireExpressionDoubleNégation
ProcéduresÉquivalentes	PRéduireExpressionNégationVrai, PRéduireExpressionNégationFaux
IDObjectifPédagogique	ObjectifMaîtriserRègleDeMorgan
Métadonnées	Auteur : Philippe Fournier-Viger Description : L'objectif pédagogique de maîtriser la règle de De Morgan.
ButsNécessaires	
ButsÉquivalents	
ProcéduresNécessaires	PAppliquerDeMorganConjonction, PAppliquerDeMorganDisjonction
ProcéduresÉquivalentes	

Figure 6.9 – La description des objectifs pédagogiques *ObjectifMaîtriserExpressionNégation* et *ObjectifMaîtriserRègleDeMorgan*

Quoique l'objet n'inclut pas actuellement de connaissances didactiques, elles pourraient cependant être ajoutées dans les facettes reprises d'Hafsaoui (2005) pour indiquer également comment l'objet doit être enseigné.

La mise en application

Ce travail a expérimenté les différentes procédures de réduction booléenne définies sur des exercices au moyen du *MGC*. Pour ces tests, le choix des procédures s'est fait au moyen de touches du clavier. En temps normal, l'interface du laboratoire avec l'aide du module « modèle de l'apprenant » devrait dicter les procédures effectuées par l'apprenant, mais le module « modèle de l'apprenant » est en développement. Le *MGC* a généré avec succès la mémoire épisodique, ce qui démontre le bon fonctionnement du logiciel

développé. La conception du *MGC* est indépendante du domaine : il peut charger des objets d'apprentissage autres que celui de la réduction booléenne.

6.3 Conclusion

Ce chapitre a décrit la représentation d'un domaine concret avec le modèle. La modélisation s'est complétée avec succès et l'expérience semble confirmer les avantages anticipés. Dans le futur, le développement d'autres modules d'*ASTUS* et la modélisation d'autres domaines contribueront sans aucun doute à l'amélioration du modèle. Cette dernière section porte sur quelques observations réalisées pendant l'expérimentation et sur des extensions envisagées au modèle .

La performance du *MGC*

L'emploi d'un moteur d'inférence pour *LD* ne semble pas ralentir considérablement l'exécution du *MGC* par rapport à l'implantation d'*UV* présentée au chapitre 2. En outre, une fois le fichier *OWL* chargé en mémoire, le moteur d'inférence infère une seule fois la hiérarchie des concepts dans la *TBox*, car elle ne varie pas. Le raisonnement sur *ABox* demande plus de traitement, car des individus nommés peuvent être ajoutés par le *MGC* et par conséquent, le moteur d'inférence ne peut pas tout calculer à l'avance.

La validation du modèle psychologique

L'équipe du groupe de recherche *ASTUS* a représenté les connaissances psychologiques avec l'implantation d'*UV* dans plusieurs domaines : la réduction booléenne (Najjar et al., 2004b; Najjar et Mayers, 2004; Najjar et al., 2004a), la réalisation d'un tiramisu (une recherche qui sera publiée éventuellement), le génie génétique (Zhao, 2003) et des exercises sommaires sur la conversion d'instructions machines en binaire (Hamadouche, 2004). Ces expérimentations et celle décrite dans ce chapitre tendent à montrer que le modèle

possède des primitives psychologiques qui lui assurent une certaine polyvalence pour représenter divers domaines. Néanmoins, il serait intéressant de développer le modèle pour en faire une architecture cognitive complète comme *MIACE* ou *ACT-R* afin de valider la plausibilité des différents construits psychologiques.

L'adaptation de l'éditeur *DOKGETT* au modèle

Des travaux futurs adapteront l'éditeur visuel *DOKGETT* (voir section 2.3.2) au modèle décrit dans ce mémoire. Cet éditeur intégrera l'interface de *SPK Learning Objects Browser* (cf. section 5.3.5) ou une variante, afin que les concepteurs de contenu puissent naviguer parmi des collections d'objets d'apprentissage *SPK*. Une autre amélioration à l'éditeur *DOKGETT* envisagée consiste à ajouter un nouveau type de diagramme pour lier visuellement les objectifs pédagogiques à des buts et à des procédures.

Conclusion

Ce travail présenté est le fruit de beaucoup d'expérimentation, de tâtonnement et de réflexion. Il a débuté à l'hiver 2003 par la spécification d'*UV* (Mayers, 2001). Puisque cette description contenait plusieurs aspects flous, des efforts considérables ont été déployés pour la clarifier et en faire par la suite une implantation informatique. Ensuite, l'auteur de ce mémoire a modélisé, en collaboration avec d'autres chercheurs, plusieurs domaines avec l'implantation d'*UV* (Zhao, 2003; Najjar et al., 2004b; Hamadouche, 2004). Ces expérimentations ont mis en évidence plusieurs éléments à améliorer. Finalement, les recherches ont porté sur la préparation, l'implantation et la validation d'un nouveau modèle qui pallie ces points faibles.

Un objectif crucial de cette recherche était de fournir une fondation solide et facilement adaptable, pour asseoir les connaissances d'*ASTUS*. Le travail visait donc à définir des structures minimales et bien justifiées, plutôt qu'un vaste ensemble de primitives, peu justifiées. Pour cette raison, ce mémoire a analysé les résultats en provenance de la littérature sur les *STI*, mais aussi sur la psychologie cognitive et l'intelligence artificielle. Plusieurs détails ont été décrits dans ce mémoire, puisqu'à la demande du directeur du projet *ASTUS*, ce mémoire constitue aussi un outil didactique d'introduction à la représentation des connaissances pour *ASTUS*.

Un autre objectif de cette recherche était de proposer un modèle original qui offre certains avantages par rapport aux modèles existants. Combiner les logiques de description et la

notion d'objet d'apprentissage à la modélisation cognitive pour *STI* constitue une idée originale. Par ce travail, le défi de trouver un assemblage judicieux de ces trois approches a été relevé. L'intégration du concept d'objet d'apprentissage est avantageuse, car elle favorise la réutilisation, l'extension, la description, l'entreposage et la localisation des connaissances. En outre, il est intéressant et profitable que le groupe *ASTUS* se joigne à la communauté internationale qui travaille sur les objets d'apprentissage et recourt aux mêmes standards et aux mêmes outils. Les objets d'apprentissage décrits dans ce mémoire surpassent les objets d'apprentissage traditionnels, en ce sens qu'ils décrivent à la fois les connaissances sémantiques d'un domaine, les connaissances procédurales (moyens pour les manipuler) et les connaissances didactiques spécifiques pour les enseigner. D'autre part, l'intégration des *LD* (1) permet de décrire avec des primitives logiques riches les liens entre les concepts, (2) facilite l'organisation des connaissances en unités réutilisables extensibles et (3) permet de profiter de moteurs d'inférence efficaces. Le mémoire a exposé l'utilité des mécanismes d'inférence et l'expressivité du modèle en donnant une méthodologie pour extraire une structure de cours similaire à celle de *CREAM*. Des recherches restent toutefois à accomplir pour déterminer la façon dont un module tuteur pourrait tirer profit de ces construits dans *ASTUS*.

La modélisation de la réduction booléenne s'est complétée avec succès et l'expérience semble confirmer les avantages anticipés. Actuellement, des étudiants préparent la validation avec d'autres domaines.

Bibliographie

Altmann, E. et Trafton, J., 2002. Memory for goals : An architectural perspective. Cognitive Science 26, 39–83. (Cité p. 107)

Anderson, J. R., 1993. Rules of the mind. Lawrence Erlbaum Associates, Hillsdale. (Cité p. 29, 48, 59 et 61)

Anderson, J. R., Corbett, A. T., Koedinger, K. R. et Pelletier, R., 1995. Cognitive tutors : Lessons learned. Journal of Learning Science 4 (2), 167–207. (Cité p. 49 et 76)

Anderson, J. R. et Douglass, S., 2001. Tower of hanoi : Evidence for the cost of goal retrieval. Journal of Experimental Psychology : Learning, Memory and Cognition 27 (6), 1331–1346. (Cité p. 107)

Baader, F., 2003. Appendix 1 : Description logic terminology. Dans Baader, F., Calvanese, D., McGuinness, D., Nardi, D. et Patel-Schneider, P. (éditeurs), The Description Logic Handbook : Theory, Implementation and Applications. Cambridge University Press, pp. 495–505. (Cité p. 89 et 90)

Baader, F., Horrocks, I. et Sattler, U., 2003. Description logics as ontology languages for the semantic web. Dans Hutter, D. et Stephan, W. (éditeurs), Festschrift in honor of Jörg Siekmann. Lecture Notes in Artificial Intelligence. Springer-Verlag. (Cité p. 82, 88, 92 et 93)

Baader, F. et Nutt, W., 2003. Basic description logics. Dans Baader, F., Calvanese, D., McGuinness, D., Nardi, D. et Patel-Schneider, P. (éditeurs), The Description Logic Handbook : Theory, Implementation and Applications. Cambridge University Press, pp. 47–100. (Cité p. 11, 81, 85, 86, 88, 89, 91, 92 et 93)

Barrett, K., Cassels, B., Haahr, P., Moon, D. A., Playford, K. et Withington, P. T., 1996. A monotonic superclass linearization for dylan. Dans Conference on Object-Oriented. pp. 69–82. (Cité p. 54)

Bechhofer, S., Möller, R. et Crowther, P., 2003. The dig description logic interface. Dans Proc. of the 2003 Description Logic Workshop (DL 2003). (Cité p. 97)

Bergamaschi, S., Lodi, S. et Sartori, C., 1994. Description logics as core of a tutoring system. Dans Proc. of the 1994 Description Logic Workshop (DL 94). pp. 73–74. (Cité p. 98 et 99)

Bloom, B., 1984. The 2 sigma problem : The search for methods of group instruction as effective as one-to-one tutoring. Educational Researcher 13 (6), 4–16. (Cité p. 2)

Bond, A. H., 2005. Representing episodic memory in a system-level model of the brain. Dans Neurocomputing volume. pp. 261–273. (Cité p. 60)

Botvinick, M. et Plaut, D. C., 2004. Doing without schema hierarchies : A recurent connectionist approach to normal and impaired routine sequential action. Psychological Review 111 (2), 395–429. (Cité p. 31)

Bouchard, F., 2005. Guides de conception d'interfaces pour les systèmes tutoriels intelligents. Mémoire, Université de Sherbrooke. (Cité p. 5 et 145)

Brown, J. S. et Burton, R. R., 1978. Diagnostic models for procedural bugs in basic mathematical skills. Cognitive Science 2 (2), 155–192. (Cité p. 4)

Burns, H. L. et Capps, C. G., 1988. Foundations of intelligent tutoring systems : An introduction. Dans Polson, M. C. et Richardson, J. J. (éditeurs), Foundations of Intelligent Tutoring Systems. Erlbaum, Hillsdale, NJ, pp. 1–19. (Cité p. 2 et 77)

Campbell, L. M., 2003. Engaging with the learning object economy. Dans Littlejohn, A. (éditeur), Reusing Online Resources : A Sustainable Approcach to e-learning. Kogan Page : London, Chap. 5, pp. 35–45. (Cité p. 71)

Carr, B. et Goldstein, I. P., Février 1977. Overlays : a theory of modeling for computer-aided instruction. AI Memo 406, MIT, Cambridge, Massachusetts. (Cité p. 4)

Chen, N.-C. et Lin, K.-M., 2002. Factors affecting e-learning for achievement. Dans Proceedings of the 2nd IEEE International Conference on Advanced Learning Technologies (ICALT 02). pp. 200–205, 9–12 août. (Cité p. 67)

Collins, A. M. et Loftus, E. F., 1975. A spreading-activation theory of semantic processing. Psychological Review 82 (6), 407–428. (Cité p. 36)

Cooper, R. et Shallice, T., 2000. Contention scheduling and the control of routine activities. Cognitive Neuropsychology 17 (4), 297–338. (Cité p. 30, 31, 38 et 45)

Corbett, A. T. et Anderson, J. R., 1992. Lisp intelligent tutoring system : Research in skill acquisition. Dans Larkin, J. H. et Chabay, R. W. (éditeurs), Computer-Assisted Instruction and Intelligent Tutoring Systems : Shared Goals and Complementary Approaches. Lawrence Earlbaum, Chap. 3, pp. 73–110. (Cité p. 3 et 77)

Cowan, N., 2001. The magical number 4 in short-term memory : A reconsideration of mental storage capacity. Behavioral and Brain Sciences 24 (1), 87–114. (Cité p. 26 et 47)

Culley, G. R., 1992. From syntax to semantics in foreign language cai. Dans Larkin, J. H. et Chabay, R. W. (éditeurs), Computer-Assisted Instruction and Intelligent Tutoring

Systems : Shared Goals and Complementary Approaches. Lawrence Earlbaum, Chap. 2, pp. 47–72. (Cité p. 2)

Dickinson, I., 2004. Implementation experience with the dig 1.1 specification. Rapport technique HPL-2004-85, Digital Media Systems Laboratory, Hewlett-Packard, Bristol. (Cité p. 97)

Donini, F. M., 2003. Complexity of reasoning. Dans Baader, F., Calvanese, D., McGuinness, D., Nardi, D. et Patel-Schneider, P. (éditeurs), The Description Logic Handbook : Theory, Implementation and Applications. Cambridge University Press, pp. 101–141. (Cité p. 93)

Dugdale, S., 1992. The design of computer-based mathematics education. Dans Larkin, J. H. et Chabay, R. W. (éditeurs), Computer-Assisted Instruction and Intelligent Tutoring Systems : Shared Goals and Complementary Approaches. Lawrence Earlbaum, Chap. 1, pp. 11–45. (Cité p. 2)

Duncan, C., 2003. Granularization. Dans Littlejohn, A. (éditeur), Reusing Online Resources : A Sustainable Approcach to e-learning. Kogan Page : London, pp. 12–19. (Cité p. 11, 68, 71 et 126)

Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D. L. et Patel-Schneider, P. F., 2001. Oil : An ontology infrastructure for the semantic web. IEEE Intelligent Systems 16 (2), 38–45. (Cité p. 105)

Fisher, D. L., 1984. Central capacity limits in consistent mapping, visual search tasks : four channels or more ? Cognitive Psychology 16 (4), 449–484. (Cité p. 26 et 47)

Fortin, C. et Rousseau, R., 1992. Psychologie cognitive : Une approche du traitement de l'information. Presses de l'Université du Québec. (Cité p. 26)

Fournier-Viger, P. et Bouchard, F., 2004. Comparaison d'act-r et miace, expérience : Confection d'un tiramisu. Rapport interne. (Cité p. 46 et 60)

Friesen, N., 2003. Cancore : Interoperability for learning object metadata. Dans Hillman, D. (éditeur), Metadata in Practice volume. ALA Editions, Chicago. (Cité p. 71)

Friesen, N., 2004. Three objections to learning objects. Dans McGreal, R. (éditeur), Online Education Using Learning Objects. Routledge, Londre, pp. 59–70. (Cité p. 70, 77 et 78)

Frye, D. et Soloway, E., 1987. Interface design : a neglected issue in educational software. Dans CHI '87 : Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface. ACM Press, pp. 93–97. (Cité p. 5)

Gagné, R. M., Briggs, L. et Wager, W., 1992. Principles of instructional design. Harcourt Brace Jovanovich, Orlando, FL, quatrième édition. (Cité p. 14)

Garagnani, M., Shastri, L. et Wendelken, C., 2002. A connectionist model of planning as back-chaining search. Dans Proceedings of the 24th Annual Conference of the Cognitive Science Society (CogSci 2002). Fairfax (Virgina), pp. 345–350. (Cité p. 60)

Glasspool, D. W., 2000. The integration and control of behaviour : Insights from neuroscience and ai. Dans article pour le symposium "how to design a functional mind" de AISB Convention 2000. (Cité p. 31)

Haarslev, V. et Möller, R., Août 2001. Description of the racer system and its applications. Dans Proceedings of the International Workshop on Description Logics (DL-2001). Stanford, Californie, pp. 132–141. (Cité p. 95)

Haddad, J., 2003. International market insight 117658 : E-learning market, public programs for teacher training. US and Foreign Commercial Service, US Department of Commerce. (Cité p. 67)

Hafsaoui, M., 2005. Élaboration d'une stratégie tutorielle dans un environnement intelligent d'apprentissage des sciences. Mémoire, Université de Sherbrooke. (Cité p. 7, 41, 109, 132, 138, 141, 147 et 161)

Halford, G. S., Baker, R., McCredden, J. E. et Bain, J. D., 2005. How many variables can humans process ? Psychological Science 16 (1), 70–76. (Cité p. 26 et 47)

Halford, G. S., Wilson, W. H., J. Guo, R. G., Wiles, J. et Stewart, J. E. M., 1993. Connectionist implications for processing capacity limitations in analogies. Dans Holyoak, K. J. et Barnden, J. (éditeurs), Advances in connectionist and neural computation theory : Vol. 2. Analogical connections. Norwood, pp. 363–415. (Cité p. 26, 27, 44 et 47)

Hamadouche, M., 2004. Modèle de communication entre un laboratoire virtuel réflexif et un tuteur dans un système tutoriel intelligent. Mémoire, Université de Sherbrooke. (Cité p. 6, 9, 50, 162 et 164)

Horrocks, I., 1998. The FaCT system. Dans de Swart, H. (éditeur), Automated Reasoning with Analytic Tableaux and Related Methods : International Conference Tableaux'98, number 1397 in Lecture Notes in Artificial Intelligence. Springer-Verlag, pp. 307–312. (Cité p. 95)

Horrocks, I., Patel-Schneider, P. F. et van Harmelen, F., 2003. From shiq and rdf to owl : The making of a web ontology language. J. of Web Semantics 1 (1), 7–26. (Cité p. 82, 90 et 105)

Horrocks, I. et Sattler, U., 2005. A tableaux decision procedure for SHOIQ. Dans Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005). (Cité p. 93)

Hotte, R. et Leroux, P., 2003. Technologies et formation à distance. revue STICEF 10, numéro spécial : Technologies et Formation à distance. (Cité p. 68 et 77)

IEEE, 2005. Learning object metadata standard (lom) of ieee. http ://ltsc.ieee.org/wg12/index.html, consulté en mai 2005. (Cité p. 68 et 70)

IMS Global Learning Consortium, 2005a. Content packaging information model et best practice and implementation guide, version 1.1.4 final specification.

http ://www.imsproject.org/content/packaging/index.cfm, consulté en mai 2005. (Cité p. 72)

IMS Global Learning Consortium, 2005b. Learning design information model et best practice and implementation guide, version 1.0 final specification. http ://www.imsglobal.org/learningdesign/index.cfm, consulté en mai 2005. (Cité p. 72 et 74)

Klausmeier, H. J., 1990. Conceptualizing. Dans Jones, B. F. et Idol, L. (éditeurs), Dimensions of Thinking and Cognitive Instruction. Lawrence Erlbaum Associates, Chap. 3, pp. 93–138. (Cité p. 15)

Knublauch, H., Musen, M. A. et Rector, A. L., 2004. Editing description logic ontologies with the protégé owl plugin. Dans Haarslev, V. et Möller, R. (éditeurs), Proceedings of the International Workshop on Description Logics (DL2004). (Cité p. 106 et 143)

Koper, R., 2003. Combining reusable learning resources and services with pedagogical purposeful units of learning. Dans Littlejohn, A. (éditeur), Reusing Online Resources : A Sustainable Approcach to e-learning. Kogan Page : London, Chap. 5, pp. 46–59. (Cité p. 68)

Krdzavac, N., Gasevic, D. et Devedzic, V., 2004. Description logics reasoning in web-based education environments. Dans Proceedings of the Adaptive Hypermedia and Collaborative Web-based Systems (AHCWŠ04). Munich, Allemagne. (Cité p. 98, 99, 102 et 103)

Lesgold, A., Lajoie, S., Bunzo, M. et Eggan, G., 1992. Sherlock : A coached practice environment for an electronics troubleshooting job. Dans Larkin, J. H. et Chabay, R. W. (éditeurs), Computer-Assisted Instruction and Intelligent Tutoring Systems : Shared Goals and Complementary Approaches. Lawrence Earlbaum, Chap. 7, pp. 201–238. (Cité p. 77)

Mayers, A., 1997. Miace : Une architecture théorique et computationnelle de la cognition humaine pour étudier l'apprentissage. Thèse de doctorat, Université de Montréal. (Cité p. 25, 26, 46, 47, 60, 107 et 108)

Mayers, A., 2001. Usager virtuel, document non publié, Département d'informatique, Université de Sherbrooke. (Cité p. 10, 11, 25, 29, 41, 42, 43, 48, 55, 60, 61, 62 et 164)

Mayers, A., Lefebvre, B. et Frasson, C., 2001. Miace, a human cognitive architecture. SIGCUE Outlook 27 (2), 61–77. (Cité p. 25)

Miller, G. A., 1956. The magical number seven, plus or minus two : Some limits on our capacity for processing information. Psychological Review 1 (63), 81–97. (Cité p. 26)

Minsky, M., 1981. A framework for representing knowledge. Dans Haugeland, J. (éditeur), Mind Design. The MIT Press, pp. 95–128. (Cité p. 81)

Moodie, P. et Kunz, P., 2003. Recipe for an intelligent learning management system (ilms). Dans Hoppe, U., Aleven, V., Kay, J., Mizoguchi, R., Pain, H., Verdejo, F. et Yacef, K. (éditeurs), Supplemental Proceedings of the 11th International Conference On Artificial Intelligence In Education (aied). Sydney, Australie, pp. 132–139. (Cité p. 69)

Morales, R. et Aguera, A. S., 2002. Dynamic sequencing of learning objects. Dans Proceedings of ICALT-2002. pp. 502–506. (Cité p. 76)

Najjar, M., Fournier-Viger, P., Mayers, A. et bouchard, F., 2005a. Memorising remembrances in computational modelling of interrupted activities. Dans Proceedings of the 8th Joint Conference on Information Sciences (JCIS 2005). Salt Lake City, Utah, États-unis, à paraître. (Cité p. 46 et 60)

Najjar, M., Fournier-Viger, P., Mayers, A. et Hallé, J., 2005b. DOKGETT - an authoring tool for cognitive model-based generation of the knowledge. Dans Proceedings of the

5th IEEE International Conference on Advanced Learning and Technologies (ICALT 05). Kaohsiung, Taiwan, à paraître. (Cité p. 53)

Najjar, M., Fournier-Viger, P., Mayers, A. et Hallé, J., 2005c. Tools and structures for modelling domain/user knowledge in virtual learning. Dans Proceedings of the 16th AACE World Conference on Educational Multimedia, Hypermedia and Telecommunications (ED-MEDIA 05). Montréal, Québec, Canada, pp. 4023–4028, à paraître. (Cité p. 53)

Najjar, M. et Mayers, A., 9-11 juillet 2003. A computational cognitive-based approach to represent knowledge within intelligent tutoring systems. Dans Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies (ICALT 03). Athène, Grèce., pp. 66–71. (Cité p. 31)

Najjar, M. et Mayers, A., 2004. Using human memory structures to model knowledge within algebra virtual laboratory. Dans Proceedings of the 2nd IEEE International Conference on Information Technology in Research and Education (ITRE 04),. London, UK, pp. 155–159. (Cité p. 6, 50, 145, 147 et 162)

Najjar, M., Mayers, A. et Fournier-Viger, P., 2004a. Goal-based modelling of the learner behaviour for scaffolding individualised learning instructions. Dans Proceedings of the 7th International Conference on Computer and Information Technology. Dhaka, Bangladesh, pp. 236–242, decembre 26-28. (Cité p. 6, 50, 147 et 162)

Najjar, M., Mayers, A. et Fournier-Viger, P., 2004b. Representing correct/erroneous knowledge during learning : A framework for a novel cognitive-based student model. Dans Proceedings of the 2004 International Conference on Intelligent Knowledge Systems (IKS). Assos, Troy, Turkey, pp. 236–242. (Cité p. 6, 42, 50, 147, 156, 162 et 164)

Nardi, D. et Brachman, R. J., 2003. An introduction to description logics. Dans Baader, F., Calvanese, D., McGuinness, D., Nardi, D. et Patel-Schneider, P. (éditeurs), The

Description Logic Handbook : Theory, Implementation and Applications. Cambridge University Press, pp. 5–44. (Cité p. 80, 81 et 82)

Newell, A., 1990. Unified theories of cognition. Harvard University Press. (Cité p. 48)

Nkambou, R., 1996. Modélisation des connaissances de la matière dans un système tutoriel intelligent : modèles, outils et applications. Thèse de doctorat, Université de Montréal, Montréal, Canada. (Cité p. 11, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 133, 134, 135 et 138)

Nkambou, R., Frasson, C. et Gauthier, G., 2003. CREAM-tools : An authoring environment for knowledge engineering in intelligent tutoring systems. Dans Murray, T., Blessing, S. et Ainsworth, S. (éditeurs), Authoring Tools for Advanced Technology Learning Environments. Kluwer Academic Publishers, pp. 269–308. (Cité p. 11, 13 et 20)

Nuxoll, A. et Laird, J., 2004. A cognitive model of episodic memory integrated with a general cognitive architecture. Dans International Conference on Cognitive Modeling 2004. (Cité p. 60)

Padadimitriou, C. H., 1994. Computational complexity. Addison-Wesley Publishing Company, Massachusetts, É.-U. (Cité p. 93)

Papert, S., 1981. Mindstorms : Computers, Children, and Powerful Ideas. The Harvester Press Limited., New York. (Cité p. 7)

Paquette, G., 2002. L'ingénierie pédagogique. Presses de l'Université du Québec, Sainte-Foy (Québec), Canada. (Cité p. 66 et 67)

Paquette, G., Crevier, F. et Aubin., C., 1997. Méthode d'ingénierie d'un système d'apprentissage (misa). Revue Informations In Cognito (8). (Cité p. 66)

Passardière, B. D. L. et Jarraud, P., 2004. Manuel, un profil d'application du lom pour c@mpusciences. Revue STICEF 11, iSSN : 1764-7223. (Cité p. 71)

Payadachee, I., 2002. Intelligent tutoring systems : Architecture and characteristics. Dans Wells, G. et McNeill, J. (éditeurs), Proceedings of the 32nd Annual SACLA Conference. (Cité p. 2)

Rawlings, A., Rosmalen, P. V., Koper, R., Rodriguez-Artacho, M. et Lefrere, P., Septembre 2002. Survey of educational modelling languages (emls) version 1, ces/isss. Http ://sensei.lsi.uned.es/palo/eml-version1.pdf. (Cité p. 72)

Rehak, D. R. et Mason, R., 2003. Keeping the learning in learning objects. Dans Littlejohn, A. (éditeur), Reusing Online Resources : A Sustainable Approcach to e-learning. Kogan Page : London, Chap. 5, pp. 20–34. (Cité p. 71)

Reiser, B. J., Kimberg, D. Y. et Lovett, M. C., 1992. Knowledge reprsentation and explanation in gil, an intelligent tutor for programming. Dans Larkin, J. H. et Chabay, R. W. (éditeurs), Computer-Assisted Instruction and Intelligent Tutoring Systems : Shared Goals and Complementary Approaches. Lawrence Earlbaum, Chap. 4, pp. 111–150. (Cité p. 77)

Rieber, L. P., 1992. Computer-based microworlds : A bridge between constructivism and direct instruction. Educational Technology Research and Development 40 (1), 93–106. (Cité p. 7)

Rieber, L. P., 1994. Computers, Graphics and learning. Brown and Benchmark, Madison, Wisconsin. (Cité p. 7)

Russell, S. J. et Norvig, P., 2002. Artificial Intelligence : A Modern Approach, 2ième édition. Prentice Hall. (Cité p. 81)

Sattler, U., Calvanese, D. et Molitor, R., 2003. Relationships with other formalisms. Dans Baader, F., Calvanese, D., McGuinness, D., Nardi, D. et Patel-Schneider, P. (éditeurs), The Description Logic Handbook : Theory, Implementation and Applications. Cambridge University Press, pp. 142–183. (Cité p. 81)

Schmidt-Schaub, M. et Smolka, G., 1991. Attributive concept descriptions with complements. Artificial Intelligence 48 (1), 1–26. (Cité p. 86)

Shastri, L., 2001. A computational model of episodic memory formation in the hippocampal system. Neurocomputing 38 (40), 889–897. (Cité p. 60)

Shastri, L., apr 2002. Episodicmemory and cortico-hippocampal interactions. TRENDS in Cognitive Sciences 6 (4), 162–168. (Cité p. 60)

Simic, G., Gasevic, D. et Devedzic, V., 2004. Semantic web and intelligent learning management systems. 2nd International Workshop on Applications of Semantic Web Technologies for E-Learning (at the 7th International Conference on Intelligent Tutoring Systems). (Cité p. 76)

Simionato, M., 2003. The python 2.3 method resolution order. http ://www.python.org/2.3/mro.html, accédé le 10-02-2003. (Cité p. 54)

Sirin, E. et Parsia, B., 2004. Pellet : An owl dl reasoner. Dans Haarslev, V. et Möller, R. (éditeurs), Proceedings of the International Workshop on Description Logics (DL2004). (Cité p. 95 et 96)

Teege, G., 1994. Using description logics in intelligent tutoring systems. Dans Proc. of the 1994 Description Logic Workshop (DL 94). pp. 75–78. (Cité p. 98, 102 et 103)

Trafton, J. G., Altmann, E. M., Brock, D. P. et Mintz, F. E., 2003. Preparing to resume an interrupted task : effects of prospective goal encoding and retrospective rehearsal. International Journal of Human-Computer Studies 58, 583–603. (Cité p. 107)

Tsarkov, D. et Horrocks, I., 2003. DL reasoner vs. first-order prover. Dans Proc. of the 2003 Description Logic Workshop (DL 2003) volume. pp. 152–159. (Cité p. 11 et 81)

Tsarkov, D. et Horrocks, I., 2004. Efficient reasoning with range and domain constraints. Dans Proc. of the 2004 Description Logic Workshop (DL 2004). pp. 41–50. (Cité p. 95)

Tulving, E., 1983. Element of Episodic Memory. Clarendon Press : Oxford. (Cité p. 31)

Tulving, E., 2002. Episodic memory : From mind to brain. Annual Review of Psychology (53), 1–25. (Cité p. 60)

VanLehn, K., 1988. Student modeling. Dans Polson, M. C. et Richardson, J. J. (éditeurs), Foundations of Intelligent Tutoring Systems. Erlbaum, Hillsdale, NJ, pp. 55–78. (Cité p. 4)

VanLehn, K., Ohlsson, S. et Nason, R., 1994. Applications of simulated students : An exploration. Journal of Artificial Intelligence in Education 5 (2), 135–175. (Cité p. 5)

Weber, G., 1996. Episodic learner modeling. Cognitive Science 20 (2), 195–236. (Cité p. 60)

Wenger, E., 1987. Artificial intelligence and tutoring systems : computational and cognitive approaches to the communication of knowledge. Morgan Kaufmann Publishers Inc. (Cité p. 2, 5, 8 et 77)

Wiley, D. A., 2000. The instructional use of learning objects. http ://reusability.org/read, livre en ligne, consulté en septembre 2004. (Cité p. 68 et 71)

Wiley, D. A., 2004. Learning objects : Difficulties and opportunities., http ://wiley.ed.usu.edu/docs/lo_do.pdf. (Cité p. 77 et 78)

Wiley, D. A., Recker, M. et Gibbons, A., 2000. The reusability paradox. http ://rclt.usu.edu/whitepapers/paradox.html. (Cité p. 69 et 70)

Zhao, X., 2003. Knowledge reprensentation for restriction digestion and reconstructing dna in a genetic lab. mémoire, Université de Sherbrooke. (Cité p. 6, 50, 162 et 164)

Zou, Y., Finin, T. et Chen, H., 2004. F-owl : an inference engine for semantic web. Dans Third NASA-Goddard/IEEE Workshop on Formal Approaches to Agent-Based Systems. Greenbelt, Maryland. (Cité p. 82, 95 et 106)