Une architecture de connaissance pour les systèmes tutoriels intelligents

par

Jean-François Lebeau

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, octobre 2007

Your file Votre référence
ISBN: 978-0-494-42977-8
Our file Notre référence
ISBN: 978-0-494-42977-8

Le 25 avril 2008

le jury a accepté le mémoire de M. Jean-François Lebeau dans sa version finale.

Membres du jury

M. André Mayers
Directeur
Département d'informatique

M. Froduald Kabanza
Membre
Département d'informatique

Mme Hélène Pigot
Président-rapporteur
Département d'informatique

Sommaire

Ce mémoire s'inscrit dans le projet ASTUS qui vise à développer une plate-forme de système tutoriel intelligent (STI) basé sur une architecture de connaissance indépendante du domaine. L'intérêt d'une telle plate-forme est de réduire de façon importante l'effort d'encodage des connaissances d'un domaine particulier et de permettre le développement de modules tuteur, apprenant et communication capables de manipuler de façon générique les structures de cette architecture. Ce mémoire donne un nouveau souffle aux travaux passés du groupe ASTUS en prenant un virage pragmatique pour concevoir et implémenter une nouvelle architecture de connaissance qui s'inspire en grande partie des travaux antérieurs d'ASTUS, mais aussi de STI concurrents. En fin de compte, ce mémoire présente une image nette de l'approche d'ASTUS.

Pour en arriver à cette nouvelle architecture de connaissance, nous avons formulé des critères qui permettent de comparer les architectures de connaissance présentes dans les STI. Nous avons analysé des STI qui ont des objectifs similaires à ceux d'ASTUS à partir de ces critères. Cette analyse nous a guidés dans l'élaboration de lignes directrices qui ont fixé les caractéristiques recherchées dans une architecture de connaissance pour la plate-forme ASTUS. Nous avons donc proposé une nouvelle architecture de connaissance qui respecte ces lignes directrices. Finalement, nous avons testé cette dernière avec succès en utilisant des domaines simples, mais qui, considérés comme un tout, reflète les défis des domaines visés par le projet ASTUS.

Remerciements

Mes premiers remerciements vont à André Mayers, mon directeur de recherche, pour sa passion pour la recherche, son support moral et financier et toutes les heures de discussions fort enrichissantes que nous avons eus durant ce projet.

Je tiens également à remercier Medhi Najjar et Phillipe Fournier-Viger qui m'ont introduit à la recherche poursuivie par le groupe ASTUS lors d'un cours de «projet informatique» au bac.

Merci à mes collègues d'ASTUS, Amir Abdessemed, Mikaël Fortin et François Courtemanche, pour les discussions qui ont orienté mon travail et pour leurs contributions à l'implémentation du projet.

Finalement, je présente mes remerciements aux membres du jury, Froduald Kabanza et Hélène Pigot pour avoir accepté de lire et de commenter ce mémoire.

Table des matières

Sommaire ..... **ii**
Remerciements ..... **iii**
Table des matières ..... **iv**
Liste des abréviations ..... **viii**
Liste des tableaux ..... **ix**
Liste des figures ..... **x**
Introduction **1**
L'architecture des STI 3
Le pour et le contre des STI 5
Contexte 6
Objectifs 7
Méthodologie 8
Chapitre 1 - Le module expert **9**
Définitions préliminaires 9
Perspective historique 9
Notions de psychologie cognitive 10
1.1 - Les caractéristiques 11
1.2 - Évaluation des STI 14
1.2.1 - Cognitive Tutors 14
1.2.2 - Cognitive Tutors (TRE) 19
1.2.3 - Cognitive Tutors Authoring tools 20
1.2.4 - Andes 22
1.3 - ASTUS 27
1.3.1 - Historique 27
1.3.2 - Le niveau cognitif 30
1.3.3 - Le niveau logique 34

1.3.4 - Le niveau pédagogique 35
1.3.5 - Critique du modèle d'ASTUS 36
1.4 - Conclusion 41
Chapitre 2 - L'architecture de connaissance 42
2.1 - Aperçu du prototype 42
2.2 - Les lignes directrices 44
2.3 - L'agent expert 44
2.4 - Les structures de représentation de connaissance 46
2.4.1 - Les connaissances sémantiques 47
La représentation des connaissances factuelles 47
Le rôle des connaissances sémantiques 49
L'architecture des connaissances sémantiques 50
L'environnement 55
Examen des critères 57
2.4.2 - Les connaissances procédurales 59
Accès à l'environnement 60
Les buts 61
Les procédures 62
2.4.3 - Les connaissances épisodiques 64
L'unité épisodique courante 66
Les requêtes sur les connaissances épisodiques 67
2.5 - La résolution automatique 68
2.6 - Le suivi de l'apprenant 69
2.6.1 - Les ambiguïtés 70
2.6.2 - La comparaison des arguments 71
2.7 - Le noyau de simulation du laboratoire 72
2.8 - Conclusion 73

Chapitre 3 - Exemples de modèles de connaissance 74
3.1 - L'addition en colonnes 74
3.1.1 - Les connaissances sémantiques 75
3.1.2 - Les connaissances procédurales 76
3.1.3 - Conclusion sur l'addition en colonnes 77
3.2 - L'addition de fractions 78
3.2.1 - Les connaissances sémantiques 79
3.2.2 - Les connaissances procédurales 81
3.2.3 - Conclusion sur la modélisation de l'addition de fractions 84
3.3 - La machine 85
3.3.1 - Connaissances sémantiques. 86
3.3.2 - Les connaissances procédurales 87
3.3.3 - Conclusion sur la modélisation de la machine 89
3.4 - Discussion 90
3.4.1 - La sémantique des unités de connaissance est précise 90
3.4.2 - Il y a des unités boîtes noires et boîtes de verre 91
3.4.3 - Il est facile d'examiner les unités de connaissances 91
3.4.4 - Bilan 92
3.5 - Conclusion 92
Chapitre 4 - Le prototype 93
4.1 - Chargement des ressources 94
4.2 - Boucle externe et boucle interne 97
4.3 - L'agent expert 98
4.4 - L'agent interface 99
4.5 - L'agent apprenant 100
4.5.1 - Considérations pour travaux futurs 101
4.6 - L'agent pédagogue 102
4.6.1 - L'approche par situation 103
4.7 - Conclusion 104

Conclusion **105**

Annexe A : Les outils du prototype **107**

Annexe B : Le monde des blocs **114**

Bibliographie **116**

Liste des abréviations

ASTUS	Apprentissage par Système Tutoriel de l'Université de Sherbrooke
CAI	Computer assisted instruction
CBL	Computer-Based Learning
CBT	Computer-Based Trainig ou Constraints-Based Tutor
CLT	Cognitive Load Theory
CS/SAS	Contention Scheduling / Supervisory Attention System
CT	Cognitive Tutors
CTAT	Cognitive Tutors Authoring Tools
DOKGET	Domain Knowledge GEnerator auThoring Tool
EIAH	Environements informatiques pour l'apprentissage humain
ILE	Intelligent Learning Environment
ITS	Intelligent Tutoring System
LCMS	Learning Content Management System
LD	Logiques de description
LMS	Learning Management System
MGC	Module de gestion des connaissances
MTT	Model-Tracing Tutors
OWL	Web Ontology Language
SAI	Selection-Action-Input
SWRL	Semantic Web Rule Language Combining OWL and RuleML
STI	Système Tutoriel Intelligent
TDK	Tutorial Developpement Kit
TRE	Tutorial Runtime Engine
UV	Usager virtuel
VLE	Virtual Learning Environment,
WME	Working Memory Element

Liste des tableaux

Tableau 1 - Comparaison des lexiques dans les LD, OWL et ASTUS 34
Tableau 2 - Comparaison des architectures 43
Tableau 3 - Les états des occurrences de buts/procédures 66

Liste des figures

Figure 1 - Pseudo-code des boucles externe et interne. 5
Figure 2 - Un problème dans Algebra Tutor I. 15
Figure 3 - Un WME pour le problème de l'addition à une colonne. 16
Figure 4 - Une règle de production pour le problème de l'addition à une colonne. 16
Figure 5 - L'environnement graphique de CTAT. 21
Figure 6 - L'environnement graphique d'Andes. 22
Figure 7 - La méthode de résolution pour la 2e loi de Newton 23
Figure 8 - Un principe mineur sous forme de clause de Horn. 23
Figure 9 - Le graphe des solutions pour un problème de mécanique. 24
Figure 10 - L'architecture d'ASTUS utilisant le MGC. 30
Figure 11 - L'architecture de connaissance au sein du prototype. 43
Figure 12 - Les entrées/sorties de l'agent expert. 45
Figure 13 - Les algorithmes de résolution automatique et de suivi de l'apprenant. 46
Figure 14 - L'ontologie de haut-niveau du prototype. 50
Figure 15 - Une partie des chunks pour le laboratoire du monde des blocs. 51
Figure 16 - Le reste des chunks pour le laboratoire du monde des blocs. 54
Figure 17 - Une vue partielle de l'environnement du laboratoire du monde des blocs. 56
Figure 18 - Ontologie comprenant un exemple de l'utilisation des spanning objects. 57
Figure 19 - Des connaissances procédurales pour le laboratoire du monde des blocs. 62
Figure 20 - Épisode courant pour le problème «échanger les blocs sélectionnés». 65
Figure 21 - Un apperçu de la base des connaissances du prototype. 66
Figure 22 - Épisode courant pour le problème «dupliquer une pile». 68
Figure 23 - Une capture d'écran du laboratoire pour l'addition en colonnes. 75
Figure 24 - L'ontologie du laboratoire pour l'addition en colonnes. 76
Figure 25 - Les connaissances procédurales pour l'addition en colonnes. 77
Figure 26 - Une capture d'écran du laboratoire pour l'addition de fractions. 78

Figure 27 - La taxonomie des concepts dans le modèle pour l'addition de fractions. 79
Figure 28 - Le contexte principale du laboratoire pour l'addition de fractions. 80
Figure 29 - Les fonctions du modèle de connaissance pour l'addition de fractions. 81
Figure 30 - Les connaissances procédurales pour l'addition de fractions. 82
Figure 31 - Une capture d'écran du laboratoire machine (avec le contexte du rapport). . . . 85
Figure 32 - L'ontologie du laboratoire machine. 86
Figure 33 - Les connaissances procédurales pour la machine. 88
Figure 34 - Les connaissances procédurales pour le rapport. 89
Figure 35 - Un aperçu complet du prototype. 93
Figure 36 - La fenêtre du pédagogue. 101
Figure 37 - Boîte de dialogue pour le choix du laboratoire. 107
Figure 38 - Boîte de dialogue pour choisir le type de boucle externe. 107
Figure 44 - Boîte de dialogue affichée par l'agent interface lorsqu'il détecte une pause. . . 107
Figure 39 - Boîte de dialogue pour le choix de l'activité par l'apprenant. 108
Figure 40 - Fenêtre de contrôle de l'agent interface. 108
Figure 41 - Fenêtre de contrôle de l'agent expert. 109
Figure 42 - Fenêtre de contrôle de l'agent pédagogue. 109
Figure 43 - La fenêtre de contrôle de l'agent apprenant. 110
Figure 45 - Fenêtre du simulateur des actions de l'apprenant. 110
Figure 46 - Fenêtre qui permet de visualiser les connaissances procédurales. 111
Figure 47 - Fenêtre qui permet de visualiser l'environnement. 111
Figure 48 - Fenêtre qui permet de visualiser les connaissances sémantiques. 112
Figure 49 - Fenêtre qui permet de visualiser les connaissances procédurales. 112
Figure 50 - Fenêtre pour visualiser les ressources pédagogiques. 113
Figure 51 - Le monde des blocs configuré pour le problème des blocs à échanger. 114
Figure 52 - Le monde des blocs configuré pour le problème de la pile à dupliquer. 114
Figure 53 - Le contexte pour la création de blocs. 115
Figure 54 - Le contexte pour l'édition d'un bloc. 115

Introduction

Issu de la communauté scientifique française à la fin des années 90, le concept d'Environnement Informatique pour l'Apprentissage Humain (EIAH) englobe toutes les approches où l'informatique est utilisée pour favoriser le développement des connaissances chez un apprenant. Les EIAH ont en commun qu'ils sont de façon explicite ou non liée à un modèle de l'apprentissage et un modèle de l'enseignement [26]. Leur développement provient d'une part des expérimentations en psychologie cognitive ou dans les sciences de l'éducation et d'autre part des progrès de l'informatique, que ce soit les avancées de l'intelligence artificielle (IA) ou l'essor du web.

On peut classer en trois grandes familles les théories d'apprentissage et d'enseignement : le béhaviorisme, le cognitivisme et le constructivisme. Les béhavioristes[1] considèrent l'apprenant comme une boîte noire dont les entrées sont des stimuli et les sorties sont des comportements. Ce sont le renforcement et les punitions, sous forme positive ou négative qui motivent l'apprentissage, c'est-à-dire l'acquisition à long terme du comportement adéquat face à la situation présentée. Les exerciseurs à choix multiple ou à réponse courte mettent en œuvre ces idées. Ces derniers et leurs variantes sont regroupés sous le nom de *Computer-Based Training* (CBT) ou *Computer-Assisted Instruction* (CAI).

Les approches cognitivistes complètent les premières en s'intéressant aux processus qui, à partir des stimuli, produisent les comportements. Dans ces approches, on s'intéresse à représenter par un modèle de traitement de l'information, le raisonnement effectué par l'apprenant. Par exemple, la *théorie des conditions d'apprentissage* de Gagné [23] s'appuie à la fois sur le behaviorisme et le cognitivisme. Finalement, d'un côté, les architectures cognitives telles que ACT-R[6] et Soar[38] implémentent ces modèles afin de simuler l'apprentissage et de l'autre, les Systèmes Tutoriels Intelligents (STI) font de même pour offrir un enseignement basé sur la façon dont le cerveau humain traite les connaissances.

1.Du moins, ceux qui prennent en compte le concept de conditionnement opérant de Skinner.

Les approches constructivistes se différencient des dernières de deux façons importantes. Premièrement, elles ne considèrent pas que les connaissances ont une existence propre[1], mais qu'elles sont intimement liées à l'expérience d'apprentissage d'un apprenant. Deuxièmement, elles placent l'apprenant au centre de l'activité d'apprentissage, donnant à l'enseignant, qu'il soit humain ou logiciel, un rôle de soutien ou de support. Ces approches se basent sur l'hypothèse que les connaissances ne peuvent être acquises que par l'expérimentation qu'un apprenant peut réaliser dans un environnement le plaçant devant un problème à résoudre. Ces derniers peuvent être des simulateurs, des micromondes ou encore des *environnements d'apprentissage intelligents* (ILE pour Intelligent Learning Environment). L'ensemble de ces implémentations peut être regroupé sous le nom de *Computer-Based Learning* (CBL). On parle d'approche socioconstructiviste lorsqu'on ajoute la collaboration entre plusieurs apprenants et certaines plates-formes de type CBL le supportent.

Le cyberapprentissage (*eLearning*) est la catégorie d'EIAH qui s'est développée le plus dans les dernières années. Plusieurs plates-formes de cyberapprentissage (aussi VLE pour Virtual Learning Environment, LMS pour Learning Management System et par extension LCMS pour Learning Content Management System) ont vu le jour, qu'elles soient commerciales (par exemple, WebCT[2]) ou libres (par exemple, Moodle[3]). Ces plates-formes visent à présenter, à l'aide de documents hypermédia, le contenu d'un cours (au sens d'une unité d'apprentissage). Elles offrent aussi de nombreuses fonctionnalités liées à la gestion du cours telles l'inscription des apprenants ou la communication (courriels, forums, etc.), mais aussi à la gestion du contenu, de la création des documents à leur organisation au sein de standards (IMS[35, 33], LOM[31], SCORM[1], etc.). Certaines offrent également des mécanismes d'évaluation sous forme de questionnaire à choix multiple.

Les ILE combinent à la fois un micromonde dans lequel l'apprenant peut interagir sous la supervision possible d'un coach, mais aussi l'accès au contenu du cours sous forme de

1.L'objectivisme soutient l'hypothèse inverse.
2.WebCT.com http:// www.webct.com
3.Moodle A Free, Open Source CMS for Online Learning http://moodle.org

documents hypermédias. L'intelligence se retrouve dans leur capacité à s'adapter à l'apprenant à la fois au niveau de l'environnement et au niveau de la présentation du contenu [12].

Les STI (aussi ITS pour Intelligent Tutoring System) sont apparus dans les années 70 et certains d'entre eux ont maintenant franchi avec succès le passage des laboratoires de recherche au déploiement dans le milieu scolaire. Bien que plusieurs les considèrent comme dépassés, ils demeurent l'objet de nombreux travaux de recherche. Les STI sont issus de la combinaison des expérimentations en psychologie cognitive et des outils de l'IA, en particulier, des systèmes de production. Ils se distinguent des exerciseurs par la présence d'une modélisation de l'apprenant qui permet une aide à l'apprentissage personnalisée et des environnements d'apprentissage (de type CBL) par la présence d'un tuteur qui a le contrôle sur l'activité. Cependant, la plupart des STI offrent aussi un environnement de résolution de problème et pour la suite du mémoire, le terme STI désignera ceux-ci.

L'architecture des STI

L'architecture classique d'un STI comporte quatre modules [63] qui sont les suivants : le module communication, le module expert, le module «modèle de l'apprenant» et le module tuteur. Cependant, dans plusieurs implémentations, bien que l'on retrouve l'ensemble de leurs fonctionnalités, certains modules sont confondus en un seul.

Le module expert contient les connaissances du domaine. Son principal rôle est de pouvoir évaluer la solution de l'apprenant. Pour ce faire, dans un premier groupe de STI, les éléments de connaissance sont opérationnels, c'est-à-dire qu'à partir des données initiales du problème soumis à l'apprenant, ils sont capables de produire une solution. Un deuxième groupe, les *Constraints-Based Tutors* (CBT), est plutôt basé sur des contraintes que doit respecter une solution du problème [44]. Dans un cas ou l'autre, ces connaissances doivent être encodées et l'effort nécessaire pour y arriver représente généralement au moins 50% du travail nécessaire pour le développement d'un STI complet [5]. Le reste de ce mémoire s'intéresse à la

problématique des modules experts utilisant des éléments de connaissance opérationnelle et décrit le rôle des autres modules en conséquence.

Le module communication comprend d'une part l'interface entre l'apprenant et le STI et d'autre part l'interface entre les actions réalisées dans l'environnement et les autres modules qui doivent les analyser. Anderson et al. [7] recommande que ce module soit le premier à être conçu, car il précise les frontières du domaine dont les problèmes sont issus, par exemple en déterminant l'ensemble des actions possibles. De plus, Anderson et al. énonce trois contraintes minimales que doit pouvoir satisfaire le module communication. 1) Chaque action qui y est effectuée doit pouvoir être communiquée au module tuteur. 2) Le module tuteur doit avoir accès à une représentation de ce que l'apprenant perçoit. 3) Le module tuteur doit pouvoir y déclencher lui-même des actions.

Le module «modèle de l'apprenant» peut être considéré comme une interface entre le module communication et le module tuteur. Pour remplir ce rôle, il doit d'abord faire un diagnostic à partir des actions effectuées au niveau du module communication, c'est-à-dire déterminer quel(s) élément(s) de connaissance it l'apprenant à poser ces dernières. Puis il doit mettre à jour son évaluation du niveau de maîtrise de chacun de ces éléments de connaissance. Puisque l'apprenant peut faire des erreurs, la plupart des STI ajoutent pour la modélisation de l'apprenant, en plus des éléments de connaissance valide utilisés par le module expert, des éléments de connaissance erronés. Aussi, dans certains STI, ce module offre la possibilité de simuler un apprenant [58], c'est-à-dire de jouer le rôle d'une architecture cognitive en se basant entre autres sur l'évaluation courante des unités de connaissance. Il est alors par exemple possible de soumettre un problème à «l'apprenant virtuel» afin de vérifier s'il pose un niveau de difficulté raisonnable. De façon plus modeste, certains modules «modèle de l'apprenant» peuvent faire une prédiction d'une ou plusieurs actions à partir de l'état courant du problème [14]. Les résultats de ces différents calculs fournissent au module tuteur les informations nécessaires pour intervenir en aval ou en amont des actions de l'apprenant.

Le module tuteur, à partir des connaissances du module expert et de l'image de l'apprenant qu'il a au travers du module apprenant, doit produire des interventions pédagogiques. Que ce

soit d'afficher un message d'aide ou bien de déclencher une action à la place de l'apprenant, l'intelligence du tuteur réside dans sa capacité à choisir la bonne intervention, au moment le plus propice. Le tuteur peut être également responsable de sélectionner le problème que l'apprenant doit résoudre. Dans ce cas, il fera preuve d'intelligence s'il soumet à l'apprenant un problème qui maximisera l'apprentissage des éléments de connaissance pertinents par rapport aux objectifs pédagogiques poursuivis.

Bien que les STI décrits ci-haut puissent être basés sur des architectures différentes, ils ont tous un comportement similaire. En effet, Van Lehn a récemment formulé une synthèse des STI destinée tant aux néophytes qu'aux chercheurs du domaine en fonction de leur comportement plutôt que de leur structure [57]. Il les décrit comme l'exécution de deux boucles qu'il nomme boucle externe et boucle interne (figure 1).

```
1. Tant que les objectifs pédagogiques ne sont pas atteints :
   1.1 Soumettre un problème à l'apprenant.
   2 Tant que : le problème n'est pas résolu :
              2.1 Guider l'apprenant sur la prochaine action à faire.
              2.2 Attendre l'action faite par l'apprenant.
              2.3 Produire une rétroaction sur l'action effectuée.
```

Figure 1 - Pseudo-code des boucles externe et interne.

Le pour et le contre des STI

Du point de vue de la pédagogique, la motivation principale des STI se trouve dans les travaux de Bloom. Ceux-ci ont montré qu'un tuteur humain qui supervise individuellement un apprenant lui fera faire un gain de deux écarts-types vis-à-vis à la moyenne [10]. Or, certains STI, en particulier les Cognitive Tutors, ont montré, en situation réelle d'apprentissage des gains d'environ un écart-type [35]. Bien que ces résultats soient fort encourageants, nous proposons trois facteurs probables pour expliquer la perte de vitesse du développement des STI depuis la fin des années 80. Il s'agit du rejet par plusieurs chercheurs en IA des approches «symboliques» pour les approches «connexionnistes», de l'échec à réduire de façon substantielle les efforts d'encodage des connaissances du module expert et de l'influence grandissante des approches constructivistes en éducation. Face à ce constat, l'avenir des STI

pourrait sembler incertain, mais certains résultats sont plus encourageants. En effet, Sweller, reconnu pour avoir développé la théorie de la charge cognitive (CLT pour *Cognitive Load Theory*) [54], et ses collaborateurs ont récemment formulé une importante critique quant aux approches constructivistes. Celle-ci se base sur l'hypothèse que les approches constructivistes ne tiennent pas compte de la capacité et des mécanismes de la mémoire humaine. [34]. En effet, l'application des approches constructivistes ferait en sorte que l'apprenant se retrouve dans une situation où ses capacités cognitives sont dépassées. Plus précisément, en plongeant le novice dans une expérimentation pratique du domaine avant de lui en avoir fait la présentation théorique et en plus en le laissant à lui-même dans cette exploration, on crée une situation où la charge cognitive devient trop grande[1]. Or les STI peuvent justement jouer le rôle d'assistance à l'apprentissage complétant un enseignement des notions du domaine, que ce soit en classe ou par le biais du *eLearning*. De plus, le module tuteur peut, à partir des informations que lui fournissent les autres modules, s'assurer qu'un novice sera suffisamment guidé. Par exemple, le tuteur peut d'abord lui présenter la démonstration de la résolution d'un problème semblable à celui qu'il s'apprête à lui soumettre.

Contexte

Ce mémoire s'inscrit dans le projet ASTUS qui vise à développer une plate-forme de STI basée sur une architecture de connaissance qui est indépendante du domaine enseigné. L'intérêt d'une telle plate-forme est de réduire l'effort d'encodage des connaissances d'un domaine et de rendre possible l'élaboration de modules tuteur, «modèle de l'apprenant» et communication génériques. Les STI développés à partir de cette plate-forme viseront les domaines scientifiques ou techniques, en fait les domaines pour lesquels il est possible de modéliser l'expertise grâce aux formalismes de l'AI «symbolique». La plate-forme empruntera des éléments du eLearning comme la gestion des accès et la communication, mais aussi des ILE comme la juxtaposition à l'environnement d'une aide en ligne sous forme de documents hypermédia.

1.Depuis l'écriture de ce mémoire, Sweller et al. a été accusé d'avoir présenté les approches constructivistes de manière réductrice, mais nous considérons que la critique originale est toujours pertinente.

L'approche pédagogique globale d'ASTUS se reflète dans une architecture comprenant un environnement d'apprentissage où différents types d'activités pourront être réalisés. On y retrouve, entre autres, la *résolution de problème* supervisée par un tuteur qui s'assure que le déroulement mène l'apprenant à la solution. En effet, bien que le comportement du module tuteur puisse être déterminé par différentes stratégies pédagogiques, il offre un comportement de base pour chaque type d'activité. Bien que la motivation principale du groupe réside dans la représentation des connaissances en IA, les défis à surmonter tant sur le plan informatique que sur le plan humain auprès des responsables de formation offrent un intérêt pour ASTUS. Initiés par la conception de l'architecture cognitive *Miace* [42], les travaux du groupe se sont concentrés sur la représentation des connaissances [21, 43, 45], mais ont aussi touché le module communication [11], le module tuteur [27] et la communication entre ses derniers [28].

Objectifs

La section précédente a décrit les ambitions du projet ASTUS. Cependant, les défis pour y arriver sont nombreux et le premier consiste à attirer, grâce à un prototype de la plate-forme, l'intérêt des enseignants, des gestionnaires de formation et des chercheurs en sciences de l'éducation pour le projet. Dans ce mémoire, nous proposons une architecture de connaissance qui possède certaines caractéristiques qui lui permettront de servir de base à ce prototype. L'étendue réelle d'une telle architecture dépasse évidemment les limites de ce qui peut être conçu et réalisé à l'intérieur d'un projet de maîtrise. Par conséquent, plusieurs limitations seront donc énoncées afin d'être éventuellement considérées dans un projet de doctorat. Ce mémoire ne cherche pas à démontrer l'efficacité au point de vue de l'aide à l'apprentissage d'un STI développé à partir de ce prototype, puisque nous pouvons nous appuyer sur les résultats que des systèmes semblables ont déjà démontrés. En résumé, les objectifs de ce mémoire sont :

- d'exposer les caractéristiques qu'une architecture de connaissance doit posséder pour atteindre les objectifs du prototype de la plate-forme d'ASTUS;

- de proposer une architecture de connaissance et d'illustrer ses caractéristiques à l'aide de la modélisation de différents domaines simples;

- de montrer, à l'aide d'une version minimale, mais complète et fonctionnelle du prototype, que cette architecture possède les caractéristiques recherchées.

Méthodologie

Nous avons adopté une méthodologie classique qui découle des objectifs énoncés plus haut. Premièrement, nous recensons et critiquons les solutions présentées dans la littérature. Deuxièmement, à partir de cette analyse, nous proposons notre solution originale. Troisièmement, nous procédons à une expérimentation pour valider notre solution. La structure de ce mémoire reflète les étapes de cette méthodologie.

Le chapitre 1 présente un état de l'art des architectures de connaissance au sein du module expert d'un STI. À partir d'une liste de caractéristiques, il évalue d'une part les principaux concurrents et d'autre part l'architecture expérimentale issue des travaux antérieurs du groupe.

Le chapitre 2 présente l'architecture de connaissance du prototype et la façon dont elle permet au module expert d'accomplir ses fonctions à l'aide d'exemples issus d'un domaine «jouet».

Le chapitre 3 présente des exemples issus de différents domaines qui illustrent les caractéristiques originales de l'approche tout en décrivant les compromis effectués et les limites rencontrées.

Le chapitre 4 présente une version minimale du prototype en indiquant clairement les composantes qui ont été ciblées de façon prioritaire et les limites de leur implémentation. En somme, il décrit comment le comportement générique des STI décrit en introduction s'applique dans le prototype.

Chapitre 1

Le module expert

Ce chapitre présente et critique différentes approches pour encoder les connaissances dans certains STI, ceux dont le module expert est basé sur des éléments de connaissance opérationnels et dont le comportement se décrit à l'aide d'une boucle externe et d'une boucle interne présentées en introduction. Dans ce but, ce chapitre présente d'abord un ensemble de caractéristiques qui seront utilisées comme critère de comparaison entre les différentes approches.

Définitions préliminaires

Avant tout, il faut définir quelques notions qui reviennent fréquemment dans le texte : les structures de représentation de connaissance, le moteur d'inférence, le modèle de connaissance, la solution, l'expertise et l'architecture de connaissance. Les **structures de représentation de connaissance** réfèrent aux structures de données pour encoder l'information. L'information nécessaire pour résoudre les problèmes d'un domaine et encodée dans les structures de représentation de connaissance forme le **modèle de connaissance du domaine**. Le **moteur d'inférence** est le processus qui utilise le modèle de connaissance pour résoudre un problème. La **solution** est la trace des étapes effectuées par le moteur d'inférence. L'**architecture de connaissance** désigne l'ensemble formé par les structures de représentation de connaissance et le moteur d'inférence. Finalement, l'**expertise** est constituée de l'ensemble des solutions possibles que l'architecture de connaissance peut produire à partir d'un modèle de connaissance du domaine.

Perspective historique

Il convient d'abord d'introduire la modélisation de l'expertise telle qu'elle a été historiquement présentée dans la littérature sur les STI. Dans cette perspective, une propriété

fondamentale de l'expertise est sa transparence. Dans le cas minimal, il s'agit d'une «boîte noire» qui, en entrée, reçoit les données du problème et qui fournit, en sortie, une solution. On peut imaginer un STI qui utiliserait Maple[1], un logiciel de calcul symbolique, pour obtenir l'intégrale qu'il demande à l'apprenant d'effectuer. À l'opposé, elle peut être totalement transparente, on parle alors d'une «boîte de verre» [63], les systèmes experts créés à partir de systèmes de production tels CLIPS[2] et Jess[3] en sont des exemples. Pour ces derniers, on obtient une séquence de règles qui justifient étape par étape la démarche qui a mené à la solution. Puis, peu importe sa transparence, la solution produite par le moteur d'inférence peut être plus ou moins similaire à celle produite par un expert humain du domaine. En effet, bien qu'un système expert fournisse une démarche complète, rien ne garantit que celle-ci soit psychologiquement plausible [63]. Or, lorsque l'expertise est transparente et que l'architecture de connaissance utilisée répond à des critères issus de la psychologie cognitive, on considère le modèle de connaissance comme un **modèle cognitif**.

Notions de psychologie cognitive

Étant donné l'importance des modèles cognitifs dans le développement des STI, il est important de les définir davantage. L'intérêt premier d'une expertise encodée sous forme d'un modèle cognitif est de faciliter les interactions, dans les deux sens, entre l'apprenant et le STI [5]. Pour concevoir une architecture de connaissance qui donne lieu à de tels modèles, il faut tenir compte de la façon dont les connaissances sont stockées et traitées au sein de la mémoire humaine. En psychologie cognitive [48], on considère que la mémoire peut d'abord être décomposée en fonction de sa capacité temporelle à retenir la connaissance, on obtient alors les mémoires sensorielles, à court terme et à long terme. Puisque c'est au sein de la mémoire à long terme que le stockage permanent des connaissances a lieu, la structure de cette dernière peut inspirer la façon d'encoder un modèle cognitif. Or, la mémoire à long terme peut elle-même être divisée en mémoire déclarative et mémoire procédurale. La mémoire déclarative

1.http://www.maplesoft.com
2.CLIPS: A Tool for Building Expert Systems http://www.ghg.net/clips/clips.html
3.Jess, the Rule Engine for the Java Platform http://herzberg.ca.sandia.gov

contient les faits, tandis que la mémoire procédurale contient les habiletés, autant celles faisant appel aux capacités de raisonnement que celles faisant appel aux capacités motrices. La mémoire procédurale est dite implicite, car son contenu n'est pas directement accessible. Par exemple, se rappeler l'emplacement des touches sur un clavier fait appel à la mémoire déclarative, tandis que l'habileté de frapper les touches en écrivant ce «mémoire de maîtrise» fait appel à la mémoire procédurale. La mémoire déclarative se décompose elle aussi en mémoire sémantique et en mémoire épisodique. La mémoire sémantique contient les concepts au sens général, indépendamment des expériences vécues tandis que la mémoire épisodique contient des occurrences particulières de ces concepts en fonction du lieu et du moment où elles ont été manipulées. Pour reprendre l'exemple précédent, la mémoire sémantique contient le concept de «mémoire de maîtrise» tandis que la mémoire épisodique de l'auteur contient tous les épisodes concernant l'écriture de celui-ci.

1.1 Les caractéristiques

Puisque l'intérêt fondamental du groupe ASTUS est de développer une architecture de connaissance indépendante du domaine, les approches de type «boîte noire», qui sont nécessairement dédiées à un domaine, ne sont pas considérées dans ce travail. Par conséquent, la modélisation de l'expertise dans les STI est un problème de *représentation des connaissances*, un champ d'intérêt partagé et influencé à la fois par les chercheurs en sciences cognitives et en IA. Les systèmes experts, la logique des prédicats, les *frames*, les réseaux sémantiques et plus récemment les logiques de description sont parmi les outils de représentation des connaissances les plus répandus. Ces derniers sont décrits dans plusieurs ouvrages traitants de l'IA [p. ex. 39]. Des chercheurs du MIT [49] ont tenté de définir ce qu'est la représentation des connaissances et pour ce faire, ils ont proposé cinq rôles clés qu'elle peut remplir dans un système à base de connaissances.

1) Elle agit comme un **substitut** du domaine réel, elle permet donc de raisonner sur le domaine sans agir directement sur celui-ci.
2) Elle définit une **vision** particulière du domaine.

3) Elle établit la **nature des inférences** qu'elle permet de produire.
4) Elle fournit un moyen **efficace sur le plan computationel** de manipuler les connaissances.
5) Elle constitue un **langage** pour communiquer à propos du domaine.

À partir de ces rôles, nous avons dégagé les caractéristiques suivantes pour l'architecture de connaissance d'un module expert :

1) La clarté de la sémantique du modèle de connaissance. D'abord, il faut considérer qu'il peut y avoir plus qu'un niveau de substitution dans la modélisation d'un domaine enseigné par un STI. En effet, le module communication constitue lui-même une représentation du domaine. De plus, la frontière entre l'expertise du domaine réel ou naturel et celle du domaine représenté par le module communication peut être ambiguë. Par exemple, dans un cas, il se peut que le module communication offre un formulaire de saisie pour la solution beaucoup plus structuré que la feuille de papier naturellement utilisée. Dans un autre cas, la représentation graphique d'un objet qui doit être analysé par l'apprenant peut ne pas refléter tous les détails de l'objet réel.

2) La vision qui teinte le modèle de connaissance. Un domaine peut être perçu ou décrit de différentes façons, il y a donc plus d'un modèle de connaissance possible pour un même domaine. Au cours de notre revue de la littérature, nous avons rencontré trois visions différentes : 1) la vision «cognitive» qui s'intéresse avant tout à l'apprentissage, 2) la vision «expert» qui correspond à l'approche classique de la résolution de problème en IA et 3) la vision «pédagogique» qui prend en compte au départ la fonction d'enseignement du STI lors de la modélisation des connaissances.

3) La nature des éléments opérationnels de connaissance. Bien que la plupart des STI utilisent les règles de production pour représenter les éléments opérationnels, d'autres techniques ont leurs avantages propres (des exemples seront présentés dans la suite de ce chapitre). Aussi, il peut être intéressant d'encoder différemment les connaissances en fonction

de la façon dont elles sont manipulées, notamment celles donnant lieu à des résultats mentaux de celles donnant lieu à des actions dans le module communication.

4) La capacité de l'architecture de connaissance à suivre le rythme des actions de l'apprenant. Les STI sont des systèmes à base de connaissances dont le moteur d'inférence doit pouvoir analyser les étapes produites par l'apprenant et déterminer si elles doivent être suivies d'une rétroaction. La tâche est computationnellement ardue, et pour contourner ce problème, plusieurs STI précalculent l'ensemble des solutions possibles pour un problème donné. Évidemment, cette stratégie n'est pas applicable dans les domaines où l'apprenant peut faire dévier la solution dans une direction inattendue ou quand le nombre de solutions possibles est grand, voire infinie.

5) Le modèle de connaissance comme un langage de communication entre les modules et avec l'apprenant. Plus ce langage est intelligible pour les différents modules, plus la communication est facilitée avec l'apprenant. Par exemple, si le module tuteur peut analyser la structure d'une solution fournie par le module expert ou simplement explorer la structure du modèle de connaissance, il sera en mesure d'offrir des rétroactions sans effort d'encodage propre au domaine. De même, le processus d'interprétation des actions effectuées par l'apprenant au niveau du module communication pour les comparer à la solution de l'expert sera simplifié.

Puisque c'est l'encodage des connaissances qui est reconnu comme étant le principal obstacle au développement des STI, nous le considérons comme une caractéristique.

6) L'effort nécessaire pour encoder le modèle de connaissance. Ce critère est souvent relié à la capacité de mettre en œuvre des outils auteurs pour réduire substantiellement l'effort. Pour les STI que nous considérons dans ce travail, il n'existe pas à notre connaissance de tels outils. Toutefois, un «environnement de développement intégré» (*Integrated Development Environement*, IDE) comprenant un éditeur, un environnement d'exécution et des outils de débogage à l'image de ceux offerts pour de nombreux langages de programmation, permet aux auteurs de concevoir plus rapidement et de tester plus rigoureusement le modèle.

Finalement, la mission pédagogique des ITS nous amène à considérer deux caractéristiques qui n'ont pas d'équivalents dans les systèmes à base de connaissances en général.

7) La possibilité de construire des unités pédagogiques. Il s'agit de regrouper en différentes unités l'ensemble du modèle de connaissance et de structurer ces dernières dans un curriculum. Dans certains cas, des objectifs pédagogiques sont définis en fonction des éléments de connaissance et reliés par des liens pédagogiques tels que préalables, conditions d'apprentissage et critères d'évaluation.

8) L'encodage des connaissances didactiques. Les connaissances didactiques sont des connaissances pédagogiques propres à une ou plusieurs unités de connaissance pour en faciliter l'acquisition. Ils peuvent être considérés comme des métadonnées par rapport au modèle de connaissance. La forme la plus répandue de ce type de connaissance est un message texte propre à élément de connaissance et contenant des variables qui sont initialisées en fonction des données du problème en cours.

1.2 Évaluation des STI

Les modules experts présentés ci-dessous sont issus de STI choisis autant pour leur réputation que pour leur parenté avec la plate-forme ASTUS. Ils seront d'abord présentés, puis critiqués en fonction des caractéristiques présentées ci-haut. Certaines caractéristiques ont été omises lorsqu'elles n'étaient pas pertinentes.

1.2.1 Cognitive Tutors

Les Cognitive Tutors (CT) de Carnegie-Mellon University sont les STI les plus souvent cités et ceux ayant fait l'objet du plus grand déploiement dans les classes [36]. Les CT ont été conçus pour tester la validité des principes sous-jacents à la famille d'architectures cognitives élaborée par J. R. Anderson et son équipe, la dernière étant ACT-R 6[1] [7]. Les résultats

1.Nous supposons par la suite que le lecteur est familier avec les principes de base de cette architecture.

impressionnants [35] obtenus en situation réelle d'apprentissage ont mené leurs concepteurs à fonder Carnegie Learning, une entreprise responsable du développement des CT, dont les plus connus sont Lisp Tutor, Algrebra Tutor et Geometry Tutor (voir figure 2).

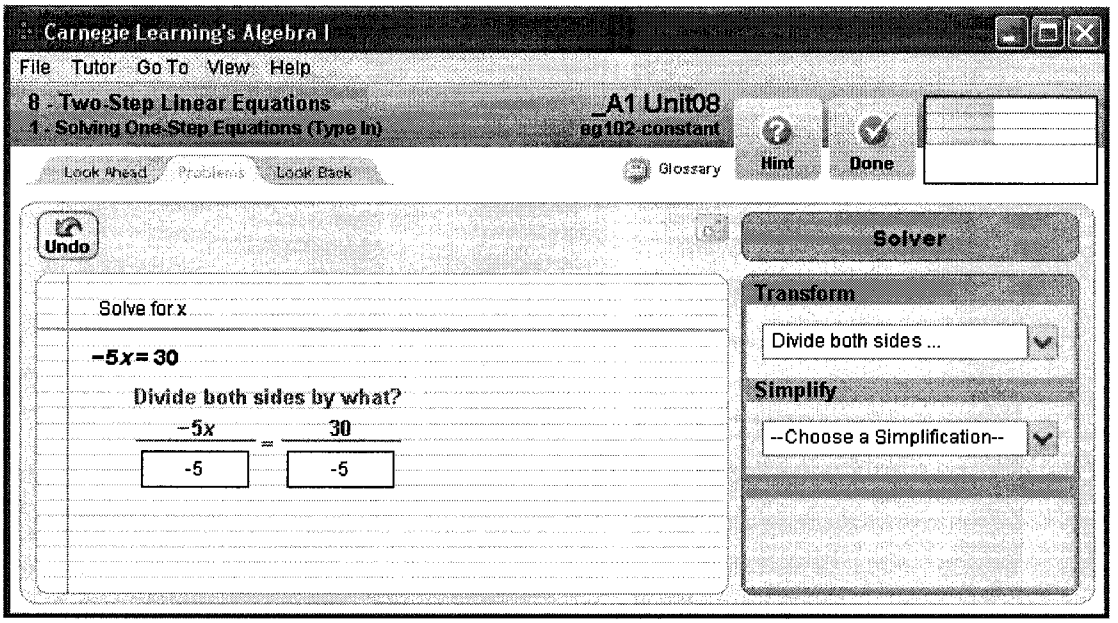

Figure 2 - Un problème dans Algebra Tutor I.

Les structures de représentation de connaissance utilisées pour obtenir les modèles cognitifs utilisés par les CT sont presque identiques à celles d'ACT-R. Il y a d'une part les «éléments de la mémoire de travail» (WME pour *Working Memory Element,* ou *chunk* dans ACT-R) pour représenter les connaissances déclaratives et d'autre part les règles de production pour représenter les connaissances procédurales. Ces deux structures ont un fondement provenant des nombreuses recherches en psychologie qui ont démontré la validité d'ACT-R. Cependant, l'architecture des CT n'intègre pas les processus de perception et de motricité, tandis que ceux de la mémoire sont réduits à leur plus simple expression. Puisque ce sont ces derniers qui ont fait d'ACT-R une architecture capable de simuler finement le comportement cognitif humain, il peut être intéressant d'étendre le modèle de connaissance d'un domaine utilisé avec les CT pour le valider avec ACT-R. Finalement, tout comme dans ACT-R, il n'y a pas de distinction parmi les connaissances déclaratives entre les connaissances sémantiques et les connaissances épisodiques.

Les WME représentent les buts, les données du problème et les valeurs entrées par l'apprenant. Un WME possède un type pouvant être dérivé d'un type plus général, qui définit ses attributs. Les valeurs de ces derniers peuvent être des données primitives (entier, chaîne de caractère, nombre à virgule flottante, etc.), un autre WME ou bien encore une liste contenant un des éléments précédents (voir figure 3).

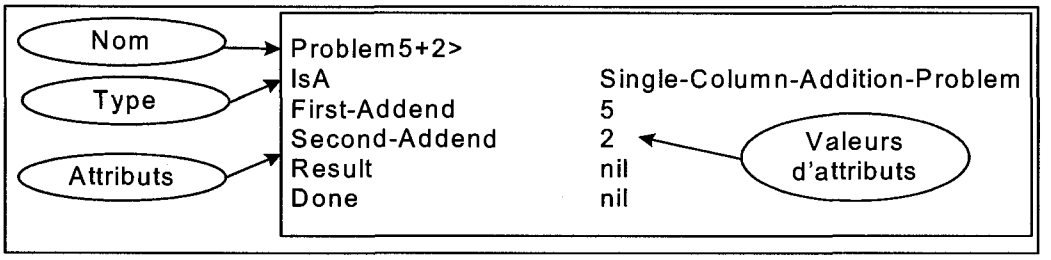

Figure 3 - Un WME pour le problème de l'addition à une colonne.

Les règles de production contiennent un ensemble de conditions et un ensemble d'actions déclenchées lorsque les conditions sont présentes (voir figure 4).

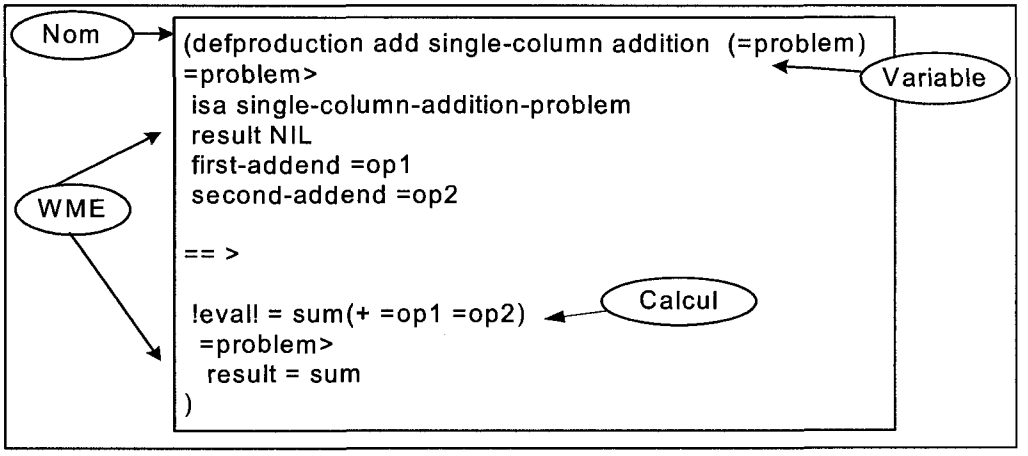

Figure 4 - Une règle de production pour le problème de l'addition à une colonne.

Les conditions sont des *patterns* de WME qui doivent être appariés avec des WME existants. Par exemple, le *pattern* présent dans la condition de la règle ci-dessus (voir figure 4) peut s'apparier avec le WME illustré plus haut (voir figure 3). Dans cet exemple, après l'unification, les variables op1 et op2 prennent respectivement les valeurs 5 et 2 partout où elles surviennent dans la règle.

Les actions expriment : 1) des calculs de valeur intermédiaire à l'aide du langage Lisp comme le montre la première action de la règle ci-haut (voir figure 4); 2) la mise à jour d'un sous-ensemble des WME présent dans la partie condition comme le montre la deuxième action (voir figure 4); 3) la création de nouveaux WME; 4) une requête au moteur d'inférence pour déclencher un nouveau cycle d'exécution des règles (chaînage). Le processus de chaînage ne se retrouve pas dans ACT-R, car au sein de cette architecture ce sont les *niveaux d'activation* des WME et l'*utilité* des règles de production qui déterminent le choix de la prochaine règle à déclencher.

Lors de la résolution d'un problème par un apprenant, un ou plusieurs WMEs expriment l'état actuel du problème. La mise à jour de ces WMEs par l'architecture en fonction des manipulations au sein du module communication est assurée par un algorithme de «traçage de modèle» (*Model Tracing)*. La manipulation faite par l'apprenant est représentée par une structure nommée SAI (pour *Selection-Action-Input*) et qui se divise en trois composantes : une **sélection**, une **action** et une **valeur**. La sélection indique quelle composante du module communication est manipulée, l'action précise la nature de cette manipulation et la valeur, qui peut être absente, contient la donnée saisie. Pour un SAI donné dans le problème de l'addition à plusieurs colonnes, la *sélection* indiquerait la colonne, l'*action* préciserait s'il s'agit de l'écriture de la retenue ou de la somme et la valeur contiendrait le chiffre saisi. Ensuite, toutes les règles (en incluant le chaînage) sont déclenchées à partir de l'état courant jusqu'à ce qu'il soit possible d'identifier une règle (ou une séquence) dont les actions correspondent au SAI. Si la recherche est fructueuse, une rétroaction positive (en vert) est affichée à l'apprenant. En plus des règles valides, des règles erronées sont aussi présentes pour reconnaître les erreurs les plus fréquentes des apprenants. Ces dernières sont associées à des messages d'erreur précis pour l'apprenant. Dans le cas où aucune règle ne peut être trouvée, un message d'erreur général est envoyé à l'apprenant. Pour ces deux cas, c'est une rétroaction négative (en rouge) qui est affichée.

Finalement, outre l'absence des processus d'accès aux éléments de la mémoire, il existe une autre différence significative entre le modèle de connaissance d'un CT et son équivalent pour

ACT-R. En effet, une partie de l'inférence réalisée à l'aide de règles de production dans ACT-R est réalisée directement à partir de code Lisp dans les CT. Par exemple, le calcul de la somme dans les actions de la règle ci-dessus (voir figure 4) pourrait être modélisé par une ou plusieurs règles de production et des *chunks* dans ACT-R. Anderson [5] a précisé qu'il était normal qu'un modèle de connaissance pour un STI traite le problème à un niveau d'abstraction plus élevé que le ferait une architecture cognitive.

Si la sémantique des *chunks* comme éléments de la «mémoire de travail» (au sens psychologique) est claire dans ACT-R, il n'en est pas toujours de même pour les WME dans les CT. En effet, il n'y a aucun formalisme, à notre connaissance, qui spécifie de façon explicite la nature des WME et des relations qui existent entre eux. Ils peuvent aussi bien représenter un objet cognitif comme le résultat d'un calcul mental, un objet tangible comme une cellule dans un tableur, ou encore un composant graphique comme une boîte de saisie ou un bouton. Par conséquent, nous jugeons difficile la prise en compte de la nature des WME de façon indépendante du domaine . Comme le chaînage des règles est encodé à l'intérieur de ces dernières, la stratégie de résolution modélisée est implicite. Nous jugeons donc difficile pour un processus computationnel de la rendre explicite, c.-à-d. sous une forme déclarative, à partir de l'inspection des règles. Par conséquent, il nous apparaît clair que le comportement du module tuteur des CT est forcément simple ou bien particulier au domaine (**Caractéristique #1 et #5**).

Les règles de production, qui sont des éléments de connaissance opérationnels des CT, sont certainement une représentation valide pour les connaissances procédurales. Toutefois, il ne nous apparaît pas évident que l'angle «cognitif» qu'elles injectent dans la modélisation du domaine soit le plus adapté aux besoins d'un STI (**Caractéristique #2 et #3**). Cette hypothèse sera détaillée davantage au chapitre 2.

L'efficacité du *Model Tracing* (**Caractéristique #4**) est garantie par l'utilisation d'algorithmes propres aux systèmes de production qui optimisent la recherche des règles pouvant être activées (p. ex. Rete [20]).

Étant donné que l'IDE des CT, le Tutorial Developpement Kit (TDK), n'est pas publiquement accessible, il est difficile pour nous d'évaluer quels outils ont été mis en place pour faciliter la modélisation des domaines (**Caractéristique #6**). Toutefois, la littérature permet de constater que ses utilisateurs ont remarqué certains problèmes qui découlent des critiques formulées plus haut. Entre autres, ils relèvent le fait qu'on ne puisse pas lier un message d'aide ou d'erreur à un déclenchement particulier d'une règle de production. Par exemple, il est impossible d'ajouter à la règle générale de l'addition un message pour le cas particulier où l'un des opérandes est zéro .

En ce qui concerne les connaissances didactiques, les Cognitive Tutors offrent un mécanisme simple et reconnu. Au sein de chaque règle de production, on peut fournir une séquence de messages d'aide, qui sont généralement, de plus en plus précis (**Caractéristique #7**). Les CT offrent des structures pédagogiques, mais la littérature donne peu de détails à leur sujet. Plusieurs règles de production similaires peuvent être regroupées sous une seule *habileté*. Ces habiletés peuvent à leur tour être regroupées dans une *leçon*. L'ensemble des leçons forme le curriculum (**Caractéristique #8**). L'évaluation de l'apprenant, présenté sous forme de diagramme à bande, est établie à l'aide d'un algorithme de traçage de connaissance (*Knowledge Tracing)*. Ce dernier se base sur le théorème de Bayes et des valeurs initiales qui doivent être déterminées à l'aide de données empiriques et propres à chaque domaine[17].

Deux variantes des CT sont présentées en fonction de leurs différences par rapport à l'architecture de base. Nous désignons par la suite l'architecture de base et ces variantes par l'acronyme MTT (*Model-Tracing Tutors*).

1.2.2 Cognitive Tutors (TRE)

Le Tutor Runtime Engine (TRE) est le successeur du TDK et il est présentement utilisé pour le développement des STI commercialisés par Carnergie Learning [50, 49]. À notre connaissance, peu de détails sur sa conception sont disponibles dans la littérature. Toutefois, l'objectif qu'il vise à atteindre est connu. Il s'agit, en conservant un comportement tutoriel identique, de gagner en temps de développement et en efficacité au moment de résoudre les

problèmes. Pour ce faire, ce nouveau modèle remplace d'abord les WME par une approche orientée-objet. Il est aussi indiqué que les buts ont leur propre représentation, mais la littérature n'en donne pas les détails. En particulier rien n'indique que leur usage soit systématique au niveau des règles de production (**Caractéristique #1, #3**). Autre nouveauté, le TRE fait un précalcule important pour rendre le *Model Tracing* plus efficace lors de la résolution du problème par l'apprenant. En fait, pour chaque problème, une cache contenant les règles pertinentes est conservée dans un fichier qui est généré au moment de l'écriture de ce dernier. Ceci permet d'éviter de fouiller toutes les règles quand certaines ne peuvent simplement pas être appliquées dans l'une ou l'autre des stratégies de résolution modélisées (**Caractéristique #4**). Finalement, bien que ces changements mis en œuvre dans une perspective d'ingénierie logicielle semblent fournir aux CT des structures où l'introspection serait plus facile; aucune volonté dans ce sens n'a été exprimée dans la littérature.

1.2.3 Cognitive Tutors Authoring tools

Les Cognitive Tutors Authoring tools (CTAT[1]) sont une version réduite, mais accessible sur le web de l'architecture des CT [2, 36]. Bien que cette architecture ne supporte pas encore toutes les fonctionnalités des TDK/TRE, comme la structure pédagogique et l'évaluation de l'apprenant, il permet de concevoir des STI comparables à ceux produits par ces dernières. L'objectif de CTAT est de permettre le développement d'un STI sans avoir recours à des informaticiens familiers avec la représentation des connaissances (**Caractéristique #6**). Pour atteindre cet objectif, CTAT propose : 1) des librairies de composantes graphiques (Java/ Flash) qui permettent d'implémenter facilement et rapidement un module communication; 2) un type de STI beaucoup plus limité nommé «tuteur par traçage d'exemple» (*Example-Tracing Tutors*) qui ne demande aucune modélisation sous forme de règles de production [3]; 3) un outil nommé *SimStudent*, encore au stade expérimental, qui permet la génération des règles de production à partir du suivi de la résolution d'un problème par un expert humain [40]. Le moteur d'inférence implémenté en Lisp pour les CT est remplacé par le système

1.Cognitive Tutors Authoring Tools Home http://ctat.pact.cs.cmu.edu/

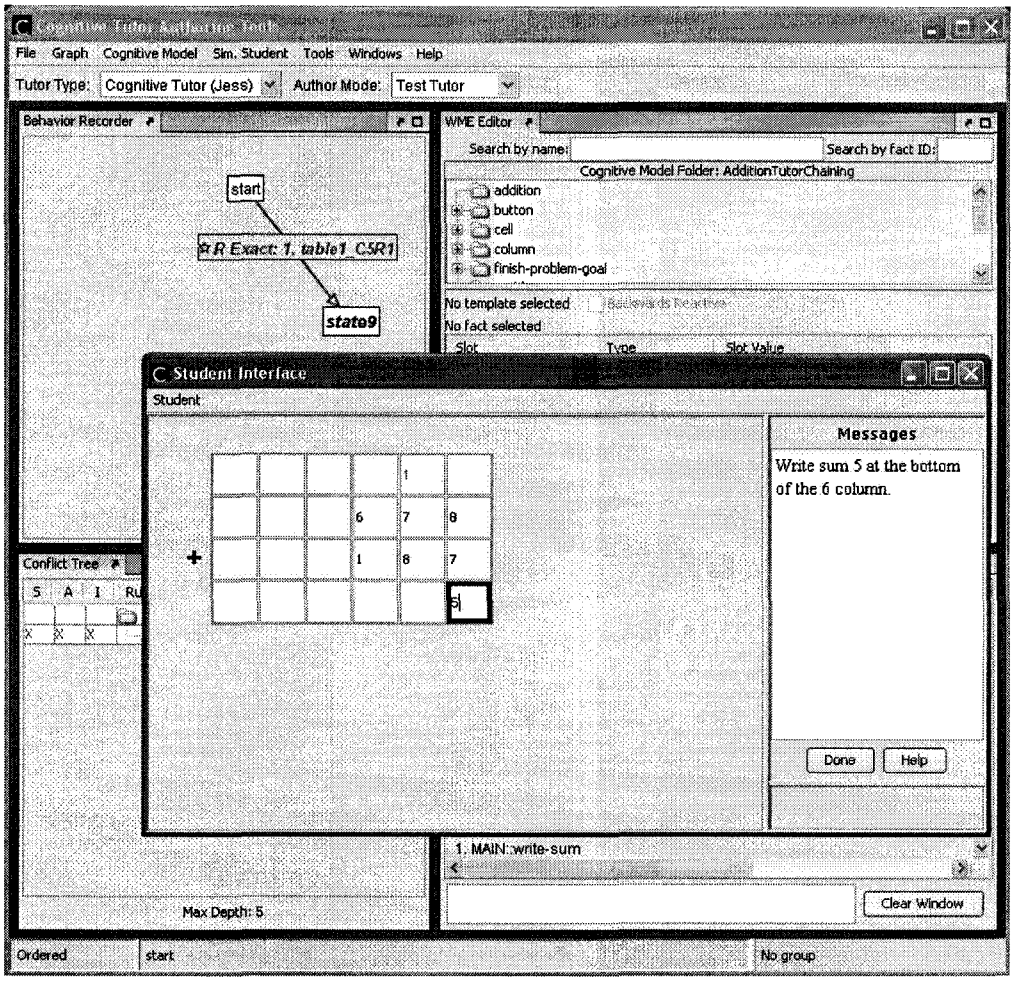

Figure 5 - L'environnement graphique de CTAT.

expert Jess, et l'édition des fichiers est facilitée par la présence d'un module d'extension (*plug-in*) pour l'environnement de développement *eclipse*[1]. La présence d'outils de débogage (*Conflict Tree, "Why not" Window, WME editor*) qui permettent à l'auteur de faire le suivi des valeurs des WME et du déclenchement des règles de production au fur et à mesure où il manipule le module communication est une originalité de CTAT.

1.Eclipse.org home http://www.eclipse.org/

1.2.4 Andes

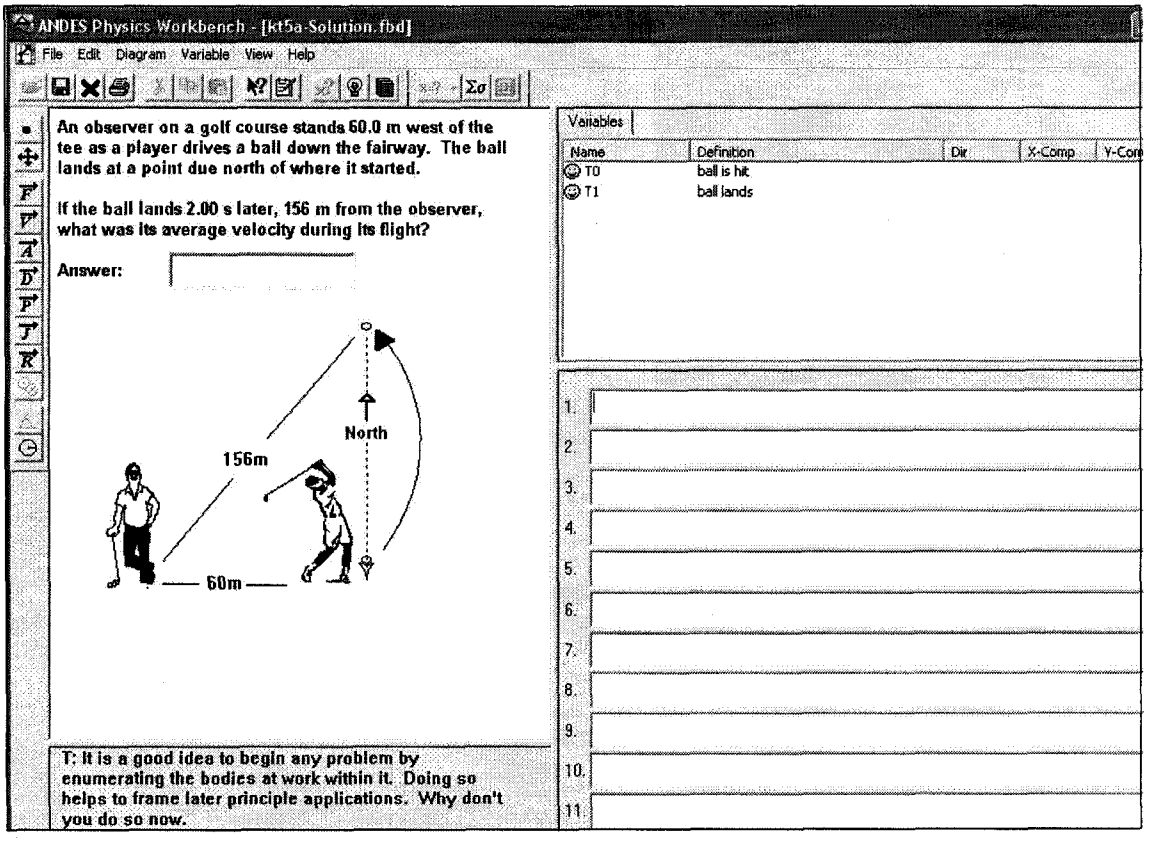

Figure 6 - L'environnement graphique d'Andes.

Andes (voir figure 6), développé par Kurt VanLehn et son équipe, vise l'enseignement de la physique de niveau collégial (mécanique, électricité et magnétisme). Il peut être téléchargé sur le web et en est maintenant à sa deuxième version majeure. Des étudiants de l'U.S. Naval Academy l'utilisent chaque année depuis 1999 [56].

L'objectif d'Andes est d'aider les étudiants à résoudre des problèmes. Dans ce but, pour chaque problème, Andes construit au préalable le graphe des solutions. Ce dernier contient toutes les étapes des démarches de solution possibles. Pour construire ce graphe, Andes a recours à des opérateurs, nommés *méthodes de résolution,* reflétant les principaux principes de la physique, par exemple la 2^{e} loi de Newton en mécanique (voir figure 7). De plus, certains opérateurs représentent des principes mineurs, des dérivations mathématiques ou encore un

```
Pour appliquer la 2e loi de Newton à un <corps> à <temps> sur l'<axe> :
1. Créer un diagramme de corps en chute libre pour le <corps> à <temps>
   sur l'<axe>.
   Créer un vecteur pour chaque <force> sur le <corps> à <temps>.
   Créer un vecteur pour l'accélération du <corps> à <temps>.
2. Écrire l'équation de la loi en fonction de la composante de l'<axe>.
3. Pour chaque <vecteur>, écrire l'équation de la projection sur chaque
   <axe>
4. Pour chaque principe mineur de la forme " <force> = <expression> " où
   la <force> est sur le <corps> à <temps>, écrire l'équation du principe
   mineur.
```

Figure 7 - La méthode de résolution pour la 2^{e} loi de Newton

```
Si l'<object1> tire sur l'<object2> alors il y a une force de tension sur
   l'<object2> dû à l'<object1>.
```

Figure 8 - Un principe mineur sous forme de clause de Horn.

certain «sens commun». Les méthodes de résolution sont modélisées de façon similaire à des méthodes en planification hiérarchique [25], tandis que les principes mineurs sont modélisés à l'aide de clauses de Horn (voir figure 8). Finalement, un algorithme de recherche fouille l'espace d'états formé par un ensemble de variables dont la valeur est soit connue ou inconnue, en utilisant les opérateurs pour ajouter de nouvelles variables, jusqu'à ce que la valeur de la variable qui est l'objet de la question soit connue.

Le modèle de connaissance des différents sous-domaines de la physique Andes est transparent et l'expertise reflète la façon dont des experts en physique résolvent les problèmes. Cependant, l'architecture de connaissance ne s'inspire pas d'une architecture cognitive d'aussi près que les MTT s'inspirent de ACT-R. Néanmoins, le modèle produit peut sans aucun doute être considéré comme un modèle cognitif. L'architecture de connaissance d'Andes se distingue de celle des MTT par le fait qu'elle est basée sur les caractéristiques de son domaine d'application. Si l'on peut comparer l'approche des MTT à un système expert qui réagit au fur et à mesure des changements déclenchés par l'apprenant; on peut comparer celle d'Andes à un planificateur produisant un plan hiérarchique faiblement ordonné (voir figure 9) [25]. En effet, en dehors des contraintes d'ordre qui sont inévitables, par exemple définir une variable avant de l'utiliser dans une équation, Andes n'induit pas une stratégie particulière de résolution du

problème. Le suivi de l'apprenant à l'aide de cette représentation des solutions consiste donc à noter les étapes effectuées afin de guider le module tuteur dans ses rétroactions. En effet, ce dernier n'enseigne pas une stratégie pour résoudre le problème, il ne fait qu'indiquer quelles sont les étapes valides et invalides en utilisant la couleur verte pour les premières et rouge pour les dernières. Si l'apprenant demande de l'aide, il suggère l'étape qui se rapproche le plus possible, dans le graphe des solutions, de la dernière réalisée par celui-ci. En fait, pour chacun des opérateurs reflétant les principaux principes de physique, l'auteur du modèle peut ajouter un contenu didactique qui est généralement constitué de trois messages : un premier pointe subtilement vers l'étape à faire; un deuxième décrit le principe de façon générale; un troisième applique le principe dans le problème en cours. Finalement, un outil auteur pour la création de problème est offert, mais il n'est pas décrit dans la littérature.

```
1. Créer le corps, ce qui définit une variable pour la masse «mc».
2. Définir les axes avec une rotation de 20 degré.
3. Définir les valeurs connues dans la donnée du problème
4. Créer le vecteur pour le déplacement «d»
5. Entrer la valeur du déplacement, «d = 20 m»
6. Entrer la valeur de la masse, «mc = 2000 kg»
7. Appliquer une première méthode de résolution [...]
8. Appliquer la 2e loi de Newton [...]
9. [...]
10. Résoudre le système d'équations obtenu et obtenir la valeur de «vf»
11. Entrer la valeur de vf (11.59 m/s) dans la zone de réponse.
```

Figure 9 - Le graphe des solutions pour un problème de mécanique.

Caractéristique #1. Deux facteurs importants font en sorte que la sémantique des modèles de connaissance d'Andes est moins ambiguë que celles des MTT. D'abord, le module communication représente explicitement le support naturel qui permet de résoudre ces problèmes, c'est-à-dire une feuille de papier sur laquelle on écrit des équations et où l'on trace des schémas (voir figure 6). De plus, le modèle établit clairement la différence entre une *étape* et une *inférence*.

Caractéristique #3. En effet, les étapes correspondent toujours à une action dans le module communication et pour chacune d'entre elles, une ou plusieurs *inférences*, c'est-à-dire l'application mentale d'un élément de connaissance, peuvent avoir été effectuées. Par

exemple, l'application de principes différents peut amener l'apprenant à écrire la même équation sans qu'il soit possible de discerner le principe utilisé.

L'architecture d'Andes est tout à fait adéquate pour aider à résoudre des problèmes dans la famille de domaines pour lesquels il est conçu. D'ailleurs, récemment, un STI très simple pour l'apprentissage des axiomes de probabilité a été réalisé à partir de l'architecture d'Andes. Cependant, de nombreux domaines, qui ne sont pas aussi formels que les mathématiques, mais dont l'expertise peut tout de même être enseignée ne se prêtent pas nécessairement bien à une modélisation sous forme de «système d'équations».

De plus, comme le graphe des solutions est statique et qu'il doit tenir compte de toutes les solutions potentielles, l'architecture d'Andes n'est pas applicable aux domaines dont les problèmes donnent lieu à une explosion combinatoire des solutions possibles. Or, cette difficulté survient justement dans le domaine de la physique puisque l'apprenant peut écrire une multitude d'équations qui sont des variantes valides de celles incluses dans le graphe des solutions. Pour contourner cette difficulté, Andes retrouve l'équation équivalente dans le graphe de solution grâce à un outil basé sur des techniques de calcul numérique qui vérifie l'indépendance du système d'équations [51].

Caractéristique #2. L'approche pédagogique d'Andes diffère de celle des MTT. Ces derniers forcent l'apprenant à suivre une stratégie particulière bien que celle-ci ne soit pas explicite dans le modèle de connaissance. Par exemple, dans Lisp Tutor, c'est le module communication qui force l'apprenant à programmer avec une approche descendante (*top down*) [16]. De son côté, Andes n'enseigne pas la stratégie globale de résolution, c'est-à-dire l'algorithme de fouille permettant de construire le graphe des solutions, explicitement. L'équipe d'Andes est consciente de ce fait et prétend que cette approche minimalement invasive est tout aussi pertinente pédagogiquement que celle utilisée dans les MTT [60]. Plus précisément, il faut savoir que VanLehn classifie les domaines en deux catégories, «procédural» et «non-procédural» [59]. Les critères sont le nombre d'étapes nécessaires, mais surtout le nombre d'étapes différentes possibles à chaque instant. Pour les domaines procéduraux, souvent illustrés par la manipulation d'un magnétoscope, la stratégie se réduit à

une marche à suivre entièrement prévisible qui est explicitée puisqu'elle est à la fois simple et essentielle à la résolution du problème. C'est dans le second cas, dont l'exemple le plus commun est la programmation, où les deux approches, MTT et Andes, privilégient des solutions différentes.

Toutefois, récemment, l'équipe de VanLehn s'est intéressée à l'enseignement explicite des stratégies. Ils ont comparé l'efficacité d'Andes à celle de Pyrenees, un STI enseignant la physique avec une approche similaire aux MTT, et les résultats obtenus ne leur permettent pas d'affirmer la supériorité de l'une ou l'autre de ces approches [59]. Notre hypothèse est qu'avec la perspective «expert» d'Andes, les apprenants acquièrent les mêmes stratégies de résolution de problème, mais sous forme de connaissance implicites, car ils ne peuvent pas les énoncer. À l'appui de cette hypothèse, la recherche de Chi [13] a montré que les stratégies ainsi apprises ne se transfèrent pas aisément à des domaines équivalents.

Caractéristique #4. D'un côté, Andes précalcule les graphes de solution et ne fait que suivre simplement les actions effectuées par l'apprenant. D'un autre côté, Andes doit utiliser des algorithmes d'analyse numérique pour interpréter les actions de l'apprenant et mettre à jour son «modèle de l'apprenant». Andes utilisait les réseaux bayésiens pour réaliser le *Model Tracing et le Knowledge Tracing*. Le «modèle de l'apprenant» issu des réseaux bayésiens aurait permis à Andes d'implémenter un comportement sophistiqué au niveau de la boucle externe, mais les enseignants impliqués dans le projet y étant opposés, il a été mis de côté pour l'instant.

Caractéristiques #6, #7 et #8. À quelques détails près, Andes offre les mêmes possibilités que les MTT. Encore une fois, la littérature n'en dit pas suffisamment pour que nous puissions juger de l'impact des outils offerts pour réduire les efforts d'encodage, en particulier lorsqu'un nouveau problème nécessite l'ajout d'opérateurs. Finalement, la littérature semble indiquer qu'il n'y a pas de structures pédagogiques pour regrouper les problèmes.

1.3 ASTUS

Pour la suite de ce chapitre, le module expert d'ASTUS, l'architecture d'ASTUS ou encore simplement «ASTUS» désignent la conception théorique, les différentes implémentations ou encore les expérimentations qui précèdent la réalisation de ce mémoire.

1.3.1 Historique

ASTUS est issu de *Miace : Une architecture théorique et computationnelle de la cognition humaine pour étudier l'apprentissage* [42], d'*Usager virtuel* (43), d'*Un modèle de représentation des connaissances à trois niveaux de sémantique pour les systèmes tutoriels intelligents* (21) et d'*Une approche cognitive computationnelle de représentation de la connaissance au sein des environnements informatiques d'apprentissage* (45).

Miace est une architecture cognitive qui emprunte de nombreuses caractéristiques à ACT-R, mais se distingue en intégrant dans une seule architecture des recherches effectuées dans des domaines distincts : mémoire de travail, mémoire épisodique, mémoire sémantique, mémoire déclarative, mémoire procédurale, attention visuelle, etc. Les idées principales proposées dans *Miace* subsistent toujours dans ASTUS. Parmi celles-ci on retrouve : la représentation des connaissances sémantiques sous forme de concepts primitifs ou décrits, un traitement explicite des buts comme un cas particulier des connaissances sémantiques et la représentation des connaissances procédurales sous forme de procédures complexes décrites par un script de sous-buts et de procédures primitives faisant appel à du code compilé. Aussi, *Miace* inclut une mémoire épisodique qui représente la trace temporelle des buts satisfaits par les procédures qui manipulent des instances des concepts nommés cognitions. Finalement, avant même qu'ACT-R incorpore ses modules perception/motricité *Miace* simulait un environnement graphique constitué d'un cahier et d'un tableau noir virtuel.

«*Usager virtuel*» (UV) propose de façon succincte une piste de recherche inspirée de *Miace* pour développer des STI pour des domaines tels que les mathématiques (algèbre booléenne) et la biologie (génie génétique) au niveau du baccalauréat. D'un côté, UV abandonne de

nombreux détails de *Miace* qui sont plus pertinents pour une architecture cognitive qu'un STI, mais introduit des idées qui demeurent importantes pour ASTUS, par exemple de proposer que les connaissances sémantiques et procédurales sont communes à tous les apprenants tandis que les connaissances épisodiques sont individuelles. Finalement, UV ajoute aussi à *Miace* une ontologie de haut niveau (concept, fonction, relation, ensemble, etc.) et des mécanismes d'inférence inspirés de la logique des prédicats.

Ces idées ont fait l'objet d'une implémentation lors de l'élaboration d'«*Un modèle de représentation des connaissances à trois niveaux de sémantique pour les systèmes tutoriels intelligents*». Le modèle conceptuel d'UV est mis en œuvre dans le niveau cognitif (2) qui en élimine certains éléments jugés flous ou superflus. Ce niveau repose sur le niveau logique (1) qui est basé sur les logiques de description (LD). Au-dessus du niveau cognitif se retrouve le niveau pédagogique (3) qui encapsule les éléments des niveaux précédents au sein d'objets d'apprentissage compatibles avec les standards du *eLearning*. L'apport principal de Fournier-Viger est le **module de gestion des connaissances** (MGC), un moteur d'inférence qui utilise les données encodées dans les trois niveaux pour simuler le comportement d'un apprenant ou d'un expert humain.

« *Une approche cognitive computationnelle de représentation de la connaissance au sein des environnements informatiques d'apprentissage* » propose un modèle qui précise et épure le modèle conceptuel d'UV. Tandis que Fournier-Viger travaillait sur la conception des structures de représentation de connaissance en trois niveaux et sur le MGC; Najjar se concentrait à démontrer la validité du niveau cognitif pour construire des activités pédagogiques et aider l'apprenant au cours de la résolution de problème. La thèse de Najjar décrit donc plusieurs expérimentations qui abondent dans ce sens. Ces expérimentations utilisent une architecture de connaissance basée essentiellement sur le niveau cognitif qui a été implémenté précédemment par Fournier-Viger. Une première expérimentation porte sur la préparation d'un gâteau tiramisu; elle montre que la mémoire épisodique permet une simulation plus proche du comportement humain que l'architecture cognitive ACT-R lors d'un rappel suivant une interruption de la tâche en cours. Une deuxième expérimentation

touche le domaine de la réduction d'expressions booléennes; elle montre comment l'architecture de connaissance peut aider à générer une rétroaction personnalisée en fonction, entre autres, de la mémoire épisodique de l'apprenant. Finalement, la thèse présente un outil auteur nommé *DOKGET* qui permet de modéliser les connaissances du domaine à l'aide d'une interface graphique. Puisque cet outil a été peu utilisé, son utilisabilité hors d'un contexte expérimental n'a pas pu être démontrée. La notation graphique utilisée dans ce mémoire s'inspire de celle de *DOKGET*.

Les travaux de Najjar et Fournier-Viger dans le cadre d'ASTUS [p. ex. 22, 46] reflètent le début d'une période de transition entre une architecture de connaissance la plus fidèle possible aux contraintes de la psychologie cognitive et une architecture qui tout en gardant une certaine validité cognitive facilite l'enseignement. Le MGC est en quelque sorte l'équivalent du moteur d'inférence dans les MTT ou bien du générateur du graphe de solution dans Andes. Cependant, il cherche autant à simuler les processus cognitifs de manipulations des connaissances qu'à former le module expert d'un STI. Or, pour être cognitivement valide, il lui manque certains mécanismes nécessaires pour réaliser une simulation, par exemple un algorithme de calcul de l'activation des éléments de connaissance. De plus, le MGC n'a pas non plus montré expérimentalement qu'il peut remplir le rôle de module expert, parce que le domaine qui a été modélisé, la réduction d'expressions booléenne, posait des défis importants. Plus précisément, une seule stratégie de résolution simple a été mise en œuvre pour l'ensemble des problèmes. Par conséquent, la solution fournie par le MGC n'était pas toujours optimale et ne pouvait donc pas nécessairement se comparer à celle produite par l'apprenant. Évidemment, les expérimentations décrites dans la thèse de Najjar ont nécessité certains ajustements propres à ce domaine; il n'en demeure pas moins qu'elles ont permis de valider davantage l'approche dans son ensemble.

Le MGC s'imbrique dans l'architecture conceptuelle d'ASTUS (voir figure 10). Dans cette architecture, le module communication se divise en trois parties : 1) l'interface graphique du laboratoire qui est propre au domaine enseigné, 2) l'interface de communication qui gère les entrées/sorties de façon indépendante du domaine et 3) l'adaptateur qui fait le lien entre les

deux. Le module expert est composé de deux sous-ensembles, d'une part, le MGC et le modèle de connaissance du domaine, et d'autre part le noyau de simulation et son adaptateur. Le noyau de simulation permet d'obtenir le résultat de processus physiques comme une réaction chimique dans un laboratoire ou de manipulations cognitives nécessaires à la résolution du problème, mais qui ne correspondent pas à des objectifs pédagogiques.

Le MGC est un programme Java qui instancie d'abord les unités de connaissance du domaine à partir de leurs définitions encodées dans le format XML, puis qui permet l'exécution d'un interpréteur. Ce dernier est un cadre d'exécution qui produit les connaissances épisodiques, aussi sous format XML, correspondantes aux connaissances sémantiques et procédurales qui ont été sollicitées pour résoudre un problème. Finalement, ni le module de l'apprenant ni le

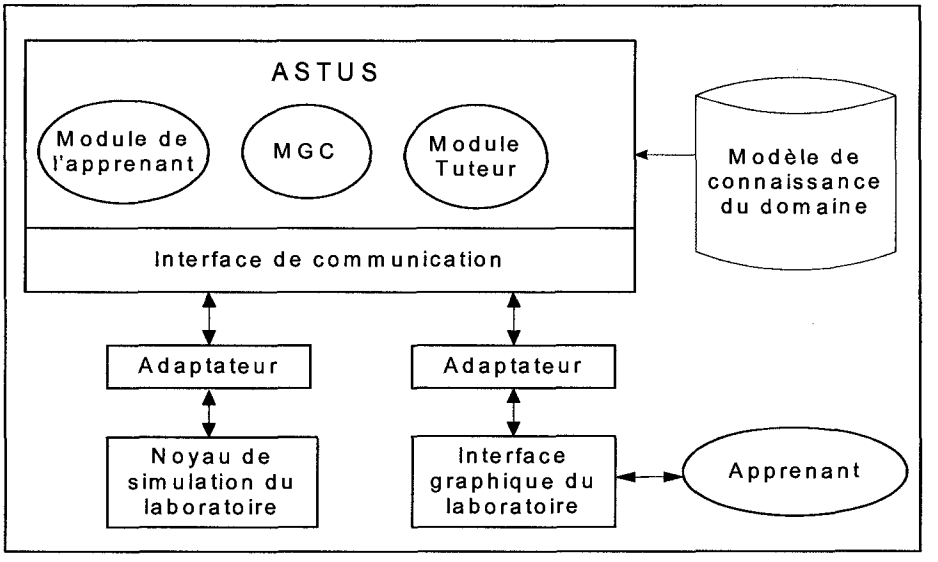

Figure 10 - L'architecture conceptuelle d'ASTUS utilisant le MGC.

module tuteur n'ont été suffisamment développés en fonction du MGC pour qu'il soit présenté plus en détail dans ce mémoire.

1.3.2 Le niveau cognitif

Le niveau cognitif est le cœur du modèle de connaissance d'un domaine. Il contient les connaissances sémantiques, les connaissances procédurales et les connaissances épisodiques.

Les connaissances sémantiques distinguent les concepts et les buts. Les concepts sont la représentation des différentes notions d'un domaine tel qu'un nombre, une équation à résoudre ou l'opérateur « = » de cette même équation. Ils sont dits primitifs lorsqu'ils sont syntaxiquement indécomposables (le nombre 3, l'opérateur « + ») et décrits lorsqu'ils sont construits à partir d'un ensemble de concepts eux-mêmes primitifs ou décrits. Par exemple « (+ 2 3) » est un concept décrit de la forme (opérateur nombre nombre) qui a la même sémantique que le concept primitif « 5 ».

Les buts représentent des intentions, par exemple celle de résoudre une équation. En effet, contrairement à plusieurs approches de l'IA où les buts décrivent des états à atteindre, ce n'est pas le cas dans le modèle d'ASTUS, puisqu'ils sont représentés uniquement par un identificateur et des paramètres sous forme de concept. Par exemple, un but représentant l'intention d'additionner deux nombres aurait la forme suivante : B_Additionner (nombre n1, nombre n2). Dans ce modèle, le traitement des buts est dit explicite, car contrairement à l'approche des MTT, ce sont toujours ces derniers qui motivent l'action, c'est-à-dire l'exécution d'une connaissance procédurale auquel il transmet ses arguments. De plus, en conformité avec certaines recherches en sciences cognitives [4], le MGC ne se limite pas à gérer la mémoire des buts à l'aide d'une pile.

Les connaissances procédurales sont représentées sous forme de procédures complexes, primitives et systèmes. Une procédure représente le(s) action(s) à effectuer pour satisfaire un but. Elles sont dites génériques puisqu'elles représentent un comportement général qui peut être appliqué à des arguments différents qui proviennent du but. Par exemple, la même procédure peut effectuer l'addition correcte de tous les couples possibles de nombres entiers compris entre zéro et dix. Les procédures primitives représentent des actions atomiques, qu'elles soient perceptibles ou non dans l'environnement d'apprentissage, par conséquent, elles ne constituent jamais une source d'erreur. Les procédures primitives sont encodées à l'aide de code Java. Les procédures complexes sont définies dans le modèle par un script de sous-but. L'enchaînement des procédures qui satisfont ces sous-buts est comparable au chaînage dans les MTT. Il y a par contre deux différences importantes. Premièrement, la

forme déclarative du script permet de connaître les sous-buts qui sont requis pour compléter cette procédure. Deuxièmement, une procédure complexe peut correspondre, contrairement à un chaînage de règle de production, à plusieurs séquences différentes de manipulations par l'apprenant dans l'environnement. Ceci entraîne évidemment l'utilisation d'un algorithme différent que celui du *Model Tracing*, dont il en sera question au chapitre 2. À l'exécution de l'interpréteur, le passage des arguments, des instances de concept nommées cognitions, est effectué des buts vers les procédures. Les procédures peuvent également retourner une cognition qui sera transmise récursivement au(x) but(s) supérieur(s). Il peut y avoir plusieurs procédures conçues pour satisfaire le même but. Cependant, il est possible que certaines de ces procédures n'aient pas des types de paramètres formels compatibles aux arguments que le but fourni. L'interpréteur utilise un algorithme basé sur la subsomption, tel que mentionné dans la section sur le niveau logique, pour éliminer les procédures incompatibles. Lors d'une simulation, s'il reste encore un choix de plusieurs procédures, le MGC, devait théoriquement choisir celle ayant la plus forte probabilité de correspondre avec le choix d'un apprenant ou d'un expert. Cependant comme le module apprenant n'a pas été mis en œuvre et que le MGC n'offre pas une façon d'établir le choix d'un expert, il présente à l'utilisateur une interface lui permettant de choisir la procédure.

Les structures de représentation de connaissance pour les buts et les procédures complexes sont compatibles avec la décomposition hiérarchique des tâches proposées par le modèle CS/ SAS de Norman et Shallice [15, 47], elles sont à cet égard cognitivement plausibles. La syntaxe des scripts pour décrire l'enchaînement des buts d'une procédure complexe offre deux avantages par rapport aux règles de production. Une règle de production associe dans une seule structure l'état d'un problème et une partie de la séquence d'actions à faire pour atteindre l'état final du problème. La séquence d'action est partielle parce que chacune de ces actions ne doit pas dépendre d'effets non prévisibles issus des précédentes à partir des conditions de la règle. Le script d'une procédure complexe, parce qu'il décrit des sous-buts (plutôt que des actions) pouvant se réaliser par des procédures différentes est moins sujet à cette contrainte. Le module tuteur en examinant la structure de la procédure possède une vision globale du processus pour atteindre le but. La vision du module tuteur est encore plus

globale si le module tuteur examine la structure du but parce qu'il peut y avoir plus d'une procédure complexe pour un but et ses paramètres. De plus, la syntaxe des scripts fournit une description d'une tâche complexe à un apprenant proche de celle qu'utiliserait un tuteur humain. Par exemple, pour le problème de l'addition à plusieurs colonnes, le script peut être une itération sur toutes les colonnes tandis que les règles de productions équivalentes réagissent à la présence du résultat dans une colonne pour déclencher la chaîne de règles qui traitera la colonne suivante [29]. En résumé, l'utilisation des scripts rend possible la réification de la stratégie de résolution du problème sous une forme déclarative, permettant ainsi au module tuteur de l'enseigner autrement qu'en offrant une rétroaction sur chacune des actions individuellement.

Le script d'une procédure complexe peut contenir l'appel à un but système. Ces derniers, définis au sein du MGC, sont satisfaits par des procédures systèmes qui sont implémentées par du code Java à même celui-ci. Les buts systèmes ont deux rôles : 1) ils représentent des mécanismes cognitifs de base et 2) ils permettent la construction de script plus complexe en introduisant des structures de contrôles.

Certaines procédures complexes sont dites «non valides», elles représentent l'unique forme d'erreur dans le modèle d'ASTUS. Lorsqu'un but est satisfait par une procédure valide, on dit qu'il est *atteint*, dans le cas contraire on dit qu'il a été *échoué*. Lorsqu'une procédure complexe n'est pas *terminée* avant qu'une autre ne soit commencée, le but que satisfait la première est *abandonné*. L'état des buts et des procédures fait partie des connaissances épisodiques.

Les connaissances épisodiques contiennent la trace hiérarchisée intégrale des éléments de connaissance qui ont été traités par le MGC, elles permettent ainsi entre autres de reproduire leur exécution à nouveau. Elles sont constituées d'épisodes. Un épisode est le regroupement d'un but, de la procédure choisie et dans le cas d'une procédure complexe, des sous épisodes engendrés. Essentiellement, les connaissances épisodiques forment une banque d'information précieuse qui peut être utilisée par les modules apprenants et tuteur.

1.3.3 Le niveau logique

Le niveau logique représente les connaissances sémantiques du domaine à l'aide des logiques de description (LD) [9]. Puisque le Web Ontology Language (OWL[1]) [61] a été choisi pour réaliser l'implémentation et qu'il possède certaines particularités, la suite de ce mémoire utilisera le lexique de ce dernier plutôt que celui, plus théorique, des LD. La modélisation du domaine est réalisée grâce à l'éditeur Protégé [37] et le moteur d'inférence utilisé est Pellet[2] [52]. Le choix de ce dernier a été motivé par les facteurs suivants : il offre une API Java, il est libre de droit et il supporte le langage OWL-DL, ou $\mathcal{SHOIN}(\mathcal{D})$.

Tableau 1 - Comparaison des lexiques dans les LD, OWL et ASTUS

LD	OWL	ASTUS
Concept	Classe	Concept
Rôle (*role*)	Propriété (*property*)	Composante
Individu (*individual*)	Individu (*individual*)	Cognition

Dans le modèle d'ASTUS (voir tableau 1), les concepts, dont l'ensemble forme l'ontologie, sont représentés à l'aide des classes et les composantes des concepts décrits sont représentées à l'aide des propriétés; les individus eux, représentent les cognitions. Grâce au niveau logique, il devient inutile de définir des liens *is-a* (ou d'héritage) dans le niveau cognitif. En effet, le MGC fait appel au moteur d'inférence du niveau logique pour obtenir la taxonomie d'un concept ou d'une cognition. De plus, l'ajout de restrictions dans la définition des classes OWL permet de définir plus précisément les concepts du niveau cognitif.

La présence d'un niveau logique au sein du MGC a de nombreux avantages concernant la modélisation de l'expertise, le plus important étant celui d'avoir une représentation cohérente des connaissances sémantiques. En effet, le moteur Pellet, à l'aide des «algorithmes de tableau» [19], vérifie que l'ontologie est consistante, c'est-à-dire qu'elle ne contient pas de classes pour lesquelles il serait logiquement impossible de créer un individu. De plus, l'élaboration d'une telle ontologie se fait hors des contraintes associées à l'héritage en

1.Version 1.0
2.Version 1.3

programmation orientée-objet, que ce soit celles de l'héritage simple ou de l'héritage multiple. La relation de *subsomption* n'est pas limitée à la définition d'un concept comme étant une spécialisation d'un autre. En effet, une classe peut être définie par une expression (Parent := Père || Mère) ou un ensemble de restrictions. Par exemple, une Mère est une Femme, pour qui il existe un individu qui satisfait la propriété «aEnfant». La subsomption s'applique tant aux classes qu'aux individus, bien que pour ces derniers, l'inférence soit restreinte parce qu'OWL, comme les LD, adopte une sémantique de «monde ouvert» [30].

Parmi les avantages du niveau logique, on peut ajouter qu'OWL permet l'extension d'une ontologie de façon plus souple que les approches classiques d'héritage. Cependant, sa principale utilisation dans ASTUS est le mécanisme d'unification entre les arguments du but et les paramètres des procédures potentiellement applicables. Ce mécanisme est simplifié puisque le MGC n'a qu'à faire une requête au moteur d'inférence pour obtenir dynamiquement l'ensemble des classes auxquelles un individu appartient.

1.3.4 Le niveau pédagogique

Le niveau pédagogique est le résultat d'un premier pas vers la création d'unités de connaissance autonomes et réutilisables. D'abord, il introduit un formalisme simple pour définir des objectifs pédagogiques à partir des buts ou des procédures. Il permet par exemple de spécifier qu'un objectif est rempli seulement si la maîtrise conjointe de deux buts est atteinte. Il permet aussi de spécifier que pour un objectif, deux procédures sont équivalentes. Ce niveau permet d'identifier à l'aide de métadonnées et de regrouper en un seul fichier un groupe de fichiers (de format XML et OWL) qui définit les connaissances du domaine, ceci en respectant des standards de l'apprentissage en ligne (IMS-CP[32] et IMS-METADATA[33]). L'utilité réelle des objets d'apprentissage ainsi formés reste cependant à démontrer. En effet, bien qu'il pourrait être intéressant, dans une plateforme comme celle visée par le projet ASTUS, de former une agrégation de ces objets; il faudra montrer que les standards propres à l'apprentissage en ligne sont bien adaptés pour le faire. Par exemple, il serait intéressant qu'un

STI qui enseigne un sous-domaine de la physique puisse faire appel à un STI qui enseigne une habileté mathématique requise et qui n'est pas maîtrisée par l'apprenant.

1.3.5 Critique du modèle d'ASTUS

Cette section décrit cinq faiblesses ou lacunes que nous avons détectées dans l'architecture d'ASTUS. Elles constituent la piste de départ pour l'élaboration de la nouvelle architecture de connaissance du prototype d'ASTUS présenté dans le chapitre suivant.

Premièrement, il faut discuter d'un niveau du modèle qui n'est pas présenté explicitement par Fournier-Viger. Ce niveau, que l'on pourrait nommer «niveau 0» se situe dans le module communication. Par exemple, dans le cas de la réduction booléenne, les expressions qui sont affichées à l'apprenant ou entrées par ce dernier, sont stockées à l'aide de structures de données dans le programme Java qui produit l'interface graphique. Ceci est contraire autant à l'approche des MTT et d'Andes qu'à *Miace,* toutefois c'est davantage le fruit d'un compromis d'implémentation que d'une réelle intention de désincarner les processus cognitifs par rapport à l'interface graphique du laboratoire.

Dans cette implémentation du modèle d'ASTUS, les cognitions représentent des éléments mentaux indépendants de l'interface du laboratoire. Pour joindre le niveau 0 à l'architecture de connaissance, il faudrait ajouter des concepts à au modèle de connaissance pour représenter les données propres à un problème. Ainsi, dans le modèle de la réduction booléenne, il existe déjà des concepts comme «VariableA, VariableB, ... VariableF». Cependant, on peut se demander si ceux-ci ne devraient pas plutôt être modélisés en tant qu'individus. Toutefois, si ces concepts constants et très précis sont représentés comme des individus, la gestion des cognitions serait reléguée au niveau du MGC. De cette façon, la distinction entre les éléments généraux du domaine et les éléments appartenant au problème serait établie clairement (**Caractéristique #1**).

De plus, cette décision de conception entraîne de nombreux problèmes pour établir des liens entre le module communication et les autres modules (**Caractéristique #5**). Par exemple,

même pour les modèles des domaines formels comme la réduction booléenne, il faut fournir un adaptateur pour faire correspondre les éléments présentés par le module communication et les éléments correspondants dans le modèle de connaissance. Ce constat n'a probablement pas été établi plus tôt parce que l'implémentation du MGC et les travaux faits sur la communication entre les modules [28] ont été faits de façon indépendante.

Deuxièmement, la représentation des procédures complexes sous forme de script ne permet pas d'extraire aisément les liens qui existent entre les sous-buts, que ce soit des contraintes d'ordre, de séquence ou d'exclusivité mutuelle. Ceci s'explique de deux façons. D'une part, un script peut avoir une complexité arbitraire, par exemple il peut faire appel à une imbrication de buts qui sont à leur tour des buts systèmes. D'autre part, le choix arbitraire du «pouvoir de calcul» permis par les structures de contrôles des buts systèmes. En effet, les buts systèmes proposent des structures de contrôle ayant été récupérées de certaines fonctions Lisp, langage utilisé pour l'implémentation de *Miace*. En somme, ces scripts de forme libre contenant des appels imbriqués (autre héritage du Lisp) à des sous-buts demandent un interpréteur Java inutilement complexe (**Caractéristiques #3 et #4**).

Troisièmement, la granularité des procédures qui sont modélisées doit être révisée. En effet, en observant le modèle de la réduction booléenne, on peut remarquer que sur la cinquantaine de procédures qu'il contient, plusieurs correspondent à des manipulations cognitives fines comme l'instanciation de concept primitif/décrit, et l'accès aux composantes des concepts décrits. «PConstruireExpressionNégation» qui retourne la cognition représentant la négation de l'expression reçue en argument et «PObtenirOpérandeGauche» qui produit une cognition sur l'opérateur de l'expression reçue en sont des exemples. Bien que ces éléments de connaissance opérationnels soient en partie nécessaires, leur attribuer le rôle de procédure n'est probablement pas approprié. En effet, puisqu'elles correspondent, dans la terminologie d'Andes, davantage à des *inférences* qu'à des *étapes*, , nous jugeons qu'elles devraient être modélisées de façon plus succincte. (**Caractéristique #2, #3**).

Quatrièmement, d'une part la redondance des connaissances sémantiques au niveau logique et au niveau cognitif et d'autre part le choix de OWL pour améliorer la représentation de ces

dernières peuvent être critiqués. En premier lieu, bien que Fournier-Viger suggère que les deux niveaux pourraient contenir des différences, aucun exemple en ce sens n'est présenté. Par conséquent, le niveau cognitif ajoute simplement des métadonnées qui auraient pu être contenues à l'aide d'annotations dans le niveau logique. En second lieu, à l'exception du mécanisme de vérification des arguments décrit plus haut, Fournier-Viger ne montre pas que les deux principaux mécanismes de OWL, la vérification de la consistance de l'ontologie et l'inférence de la subsomption au niveau des classes, ont été particulièrement utiles pour modéliser le domaine de la réduction booléenne. De plus, il faut savoir que ce sont les requêtes au moteur Pellet qui constituent la principale source de l'effort computationel au sein du MGC. **(Caractéristique #4)**.

Un avantage potentiel du niveau logique est d'offrir une représentation plus formelle des connaissances sémantiques, et ce, d'une façon éventuellement intelligible pour le module tuteur **(Caractéristique #5)**. Cependant, l'inférence, basée sur un sous-ensemble décidable de la logique des prédicats, implique de fortes contraintes sur l'expressivité du langage. Voici celles qui sont le plus souvent mentionnées dans la littérature :

1) L'impossibilité de définir une classe à partir de restrictions sur les propriétés primitives (*datatype property*). Par exemple, si on veut définir la classe «Adulte» à partir d'une restriction sur la propriété «âge», de type entier, qui doit être «>= 18». Bien que certains mécanismes[1] soient proposés pour contourner ce problème aucun n'apporte pour le moment une solution définitive au problème, qui a son origine dans les algorithmes de tableaux utilisés pour réaliser l'inférence [9].
2) Le fait que l'inférence au niveau des individus soit limitée par la perspective du «monde ouvert». En effet, contrairement au cas des bases de données qui sont un exemple de «monde fermé», avec OWL il faut prendre en compte l'ensemble des interprétations possibles du monde. Par exemple, on peut créer une classe «Enfant unique» subsumée par la classe «Personne» et définie à l'aide d'une restriction spécifiant que les propriétés «aFrère» et «aSœur» doivent avoir une cardinalité zéro.

1.Voir les changements apportés dans la version 1.1 de OWL qui supporte le language $\mathcal{SHROIQ}(\mathcal{D})$.

Cependant, Pellet ne peut inférer qu'un individu, sauf s'il a été défini comme tel, est inclus dans cette classe. En effet, le fait qu'aucun individu ne soit explicitement lié par ces propriétés à un individu de la classe «Personne» ne permet pas de conclure que toutes les interprétations du monde n'en contiennent pas. Autrement dit, dans un «monde ouvert», la négation ne peut être déduite d'une absence d'information.

3) L'impossibilité d'ajouter à une classe une restriction qui s'applique sur les valeurs de plus d'une propriété. Dans le jargon des LD, les *role-value-map* définissent une façon d'établir de telles restrictions. Ils sont par contre non décidables, sauf dans certains cas, quand on a recours aux algorithmes de tableaux [9]. Un exemple simple consiste à vouloir définir la classe «Gens qui portent le nom de famille de leur mère». Pour ce faire, il faut comparer les valeurs des propriétés «nom» et «aMère» pour deux individus. En plus d'être simple, cet exemple utilise des propriétés *fonctionnelles*, c'est-à-dire qui peuvent prendre au plus une valeur. En effet, chaque personne a un seul nom et une seule mère. Même si dans ces conditions l'inférence est décidable, aucun moteur d'inférence ne le supporte à notre connaissance.

Un cas particulièrement intéressant de *role-value-map* a été rencontré lors de la modélisation de la réduction booléenne. Il s'agit de définir la classe des expressions booléennes qui sont des *contradictions*, c'est-à-dire des expressions de la forme $P \wedge \neg P$ où P est une proposition. En effet, il faudrait dans ce cas comparer la valeur de la propriété «opérande-gauche» et de la propriété «opérande-droite» pour déterminer si elles sont équivalentes.

Puisqu'une représentation à la fois plus riche et plus formelle des connaissances sémantiques est un avantage important, nous avons étudié une piste de solution pour contourner ces limites. Il s'agit du Semantic Web Rule Language (SWRL[1]), un langage de règles qui étend de façon importante l'inférence possible au niveau des individus. Cependant, le moteur Pellet ne supportait pas cette extension au moment de réaliser ce projet[2]. Même si le moteur d'inférence Kaon2[3] le supporte, nous avons jugé que faire cohabiter l'inférence résultant (puisque ce

1.A Semantic Web Rule Language http://www.w3.org/Submission/SWRL/
2.Un sous-ensemble de SWRL est maintenant supporté par Pellet 1.5.

dernier n'offre pas toutes les fonctions offertes par Pellet) de deux moteurs différents posait des problèmes d'implémentation sérieux, du moins au niveau des performances. Même si nous avons rejeté cette piste, il est quand même intéressant d'observer les forces et les faiblesses de SWRL pour déterminer s'il aurait pu théoriquement être une solution satisfaisante.

Les règles SWRL sont des clauses de Horn qui ont pour conséquent l'affirmation soit de l'appartenance d'un individu à une classe, soit de l'existence d'une propriété liant deux individus. Un exemple classique est de déduire la propriété «oncleDe» à partir des propriétés «pèreDe» et «frèreDe». De plus, SWRL propose plusieurs prédicats prédéfinis et la possibilité d'en ajouter à l'aide de code Java. Bien que SWRL permette ainsi de contourner les limitations mentionnées plus haut, il n'y a rien qui lie formellement les classes aux règles SWRL qui les manipulent. Dans l'exemple de la classe «Adulte», bien qu'on puisse utiliser SWRL pour inférer que certains individus (ayant plus de 18 ans) en font partie, cette dernière demeure une classe atomique, c'est-à-dire sans définition. Dans ce cas, il devient difficile, même impossible pour le module tuteur de présenter, de façon générique, une définition de la classe.

Cinquièmement et dernièrement, les différentes publications qui décrivent l'architecture d'ASTUS ne mettent pas suffisamment en évidence sa principale limite (**Caractéristiques #2, #3**). D'abord, nous considérons que les procédures complexes sont une représentation hybride entre les *méthodes de résolution* d'Andes et les règles de production des MTT. En effet, le MGC construit un arbre de solution dynamiquement, c'est-à-dire en le développant après chaque étape effectuée par l'apprenant. De plus, le MGC ne possède pas d'algorithme de fouille tel qu'Andes qui lui permettrait de sélectionner plusieurs coups à l'avance, une séquence de procédures complexes qui mènent à une solution. En effet, le MGC ne tient pas compte de l'effet de bord produit par l'exécution des procédures, mais seulement du changement d'état qu'elles provoquent au niveau du but. Autrement dit, le MGC sait qu'un problème est résolu parce qu'une procédure a satisfait un but qui représente l'intention globale de résoudre le problème.

3.KAON2 : Ontology Management for the Semantic Web http://kaon2.semanticweb.org/

Par conséquent, pour les domaines «non procéduraux», le MGC requiert une approche semblable à celles des MTT, c'est-à-dire de forcer une stratégie de résolution particulière de résolution au sein du module communication. Or, parce que les procédures complexes n'ont pas la souplesse des règles de production, **chaque problème de ces domaines risque de nécessiter la modélisation d'une procédure complexe particulière**. Par exemple, dans le domaine de la réduction booléenne, qui est «non procédural», il n'a pas été possible de modéliser une procédure complexe qui représentait une stratégie de résolution satisfaisante pour la majorité des problèmes.

Nous n'avons pas pour objectif d'offrir une solution à cette limite importante de l'approche d'ASTUS dans ce travail. Par conséquent, nous nous limiterons à des modèles de connaissance pour des domaines «procéduraux», ou bien à des domaines où il est possible de modéliser un petit nombre de procédures complexes pour représenter la stratégie globale de résolution.

1.4 Conclusion

Nous avons présenté un ensemble de caractéristiques et nous l'avons appliqué comme critère de comparaison pour faire une revue des approches les plus pertinentes face à notre travail. De plus, nous avons analysé en détail certaines lacunes de l'architecture d'ASTUS en fonction de ces mêmes critères. En somme, il ressort de ce chapitre des pistes intéressantes à suivre pour concevoir l'architecture de connaissance du prototype, qu'elles proviennent des travaux antérieurs du groupe ASTUS ou bien des approches des MTT ou d'Andes.

Chapitre 2

L'architecture de connaissance

Ce chapitre présente d'abord les lignes directrices qui ont mené à l'élaboration de l'architecture de connaissance, puis la conception de cette dernière à l'aide d'exemples tirés d'un modèle de connaissance «jouet», le *monde des blocs*[1]. Avant tout, ce chapitre introduit le contexte dans lequel l'architecture de connaissance prend son sens, le prototype de la plateforme ASTUS.

2.1 Aperçu du prototype

L'architecture du prototype est semblable à celle qui a été conçue théoriquement dans les projets de recherche antérieurs d'ASTUS, mais elle se distingue sur certains points : 1) l'interface de communication est maintenant considérée comme un agent à part entière qui absorbe aussi l'adaptateur pour l'interface graphique; 2) le modèle des connaissances du domaine est intégré dans les ressources du laboratoire; 3) les ressources propres à un apprenant sont explicitement représentées et 4) une base de connaissances qui regroupe les éléments dynamiques produits par les agents est le coeur du prototype (voir tableau 2 et figure 11).

Bien que le prototype dans son ensemble soit présenté au chapitre 4, le rôle de l'agent expert doit être défini ici, parce qu'il est fortement dépendant de l'architecture de connaissance. Essentiellement, l'agent expert interprète le modèle de connaissance du domaine en fonction d'un problème et produit, dans la base de connaissance, une solution sous la forme d'une unité de connaissance épisodique. Ce mécanisme de base, inspiré directement du MGC, permet à l'agent expert de fournir au prototype deux fonctions essentielles :

1.Ce domaine est souvent donné en exemple dans la littérature sur la représentation des connaissances, voir l'annexe B pour des captures d'écran du laboratoire qui implémente ce modèle.

1) la résolution d'un problème tel que le ferait l'expert humain du domaine;
2) le suivi de l'apprenant tout au long de sa démarche pour résoudre le problème.

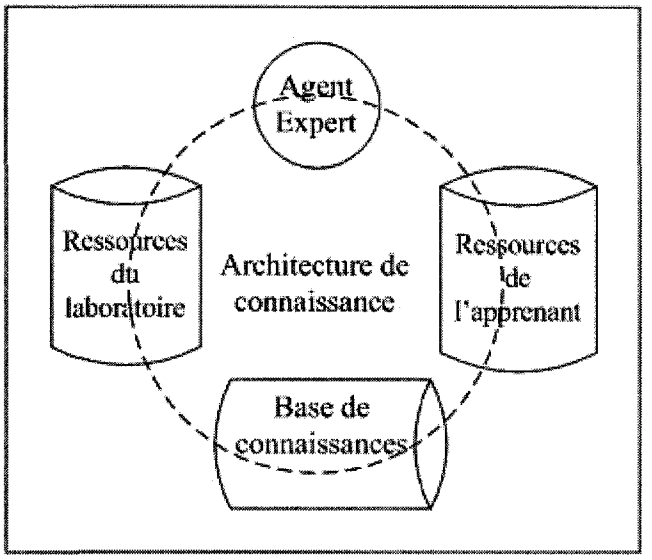

Figure 11 - L'architecture de connaissance au sein du prototype.

Tableau 2 - Comparaison des architectures

Architecture classique	**Architecture conceptuelle d'ASTUS**	**Prototype de la plateforme ASTUS[a]**
Module expert	MGC Noyau de simulation Modèle de connaissance	Agent expert Noyau de simulation Ressources du laboratoire
Module communication	Interface de communication, Interface graphique du laboratoire, Adaptateur	Agent interface Interface graphique du laboratoire
Module «modèle de l'apprenant»	Module «modèle de l'apprenant»	Agent «modèle de l'apprenant», Ressources de l'apprenant
Module tuteur	Module tuteur	Agent pédagogue

a.La base de connaissance peut être considérée comme une composante de chacun des modules.

2.2 Les lignes directrices

À partir des caractéristiques comparatives et des problèmes de l'architecture d'ASTUS que nous avons exposés au chapitre précédent, nous avons fixé trois lignes directrices pour guider la conception de l'architecture de connaissance du prototype :

1) Tous les agents doivent être capables d'examiner les éléments de connaissance manipulés et produits par l'architecture de connaissance.
2) La sémantique des éléments de connaissance doit être précise pour les auteurs afin de standardiser la modélisation des domaines et pour les agents afin que leur comportement soit cohérent.
3) La représentation des éléments de connaissance doit être le fruit d'un compromis judicieux entre a) des éléments transparents parce qu'ils ont un intérêt pédagogique; b) des éléments «boîtes noires» efficaces sur le plan informatique.

Afin d'atteindre les objectifs de ce travail en suivant ces lignes directrices, une nouvelle architecture de connaissance a été conçue et réalisée. D'abord en revenant aux sources originales (*Miace*, UV), puis en considérant les forces et les faiblesses qui ont été identifiées par Fournier-Viger et Najjar et finalement en s'inspirant d'autres STI et différents outils de représentation des connaissances.

2.3 L'agent expert

Le rôle joué par le MGC dans l'architecture d'ASTUS est principalement partagé par l'agent expert et la base de connaissance[1]. Autrement dit, le moteur d'inférence dans le prototype est à la fois situé dans l'agent expert et dans la base de connaissance. Dans la figure 10, on retrouve un sous-ensemble de l'architecture du prototype dans laquelle on met en relief les entrées/ sorties propres à l'agent expert. Dans cette même figure, on peut aussi remarquer les deux processus principaux de l'agent expert : 1) la résolution automatique et 2) le suivi de

1.Il faut voir cette dernière simplement comme une mémoire partagée entre les agents.

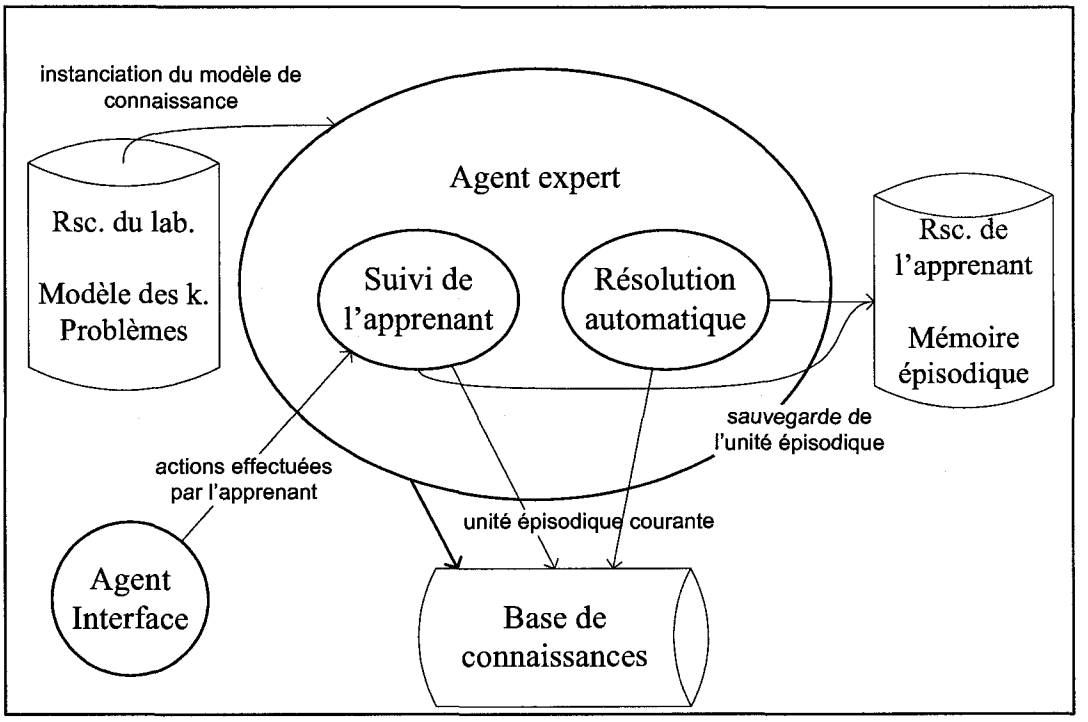

Figure 12 - Les entrées/sorties de l'agent expert.

l'apprenant (voir figure 13 pour une synthèse de leur algorithme). Avant d'accomplir l'une ou l'autre de ces fonctions, l'agent expert doit instancier dans la base de connaissance certains éléments de connaissance provenant du modèle de connaissance. En particulier, il doit ajouter dans la base de connaissance l'unité épisodique courante qui représente l'état courant de la résolution du problème et les éléments de connaissance sémantiques propres au problème (ces éléments sont présentés dans la section suivante). Bien que pour la résolution automatique l'agent expert puisse travailler en vase clos, pour le suivi, il doit recevoir des évènements provenant de l'agent interface (voir chapitre 4). Une fois le suivi de l'apprenant ou la résolution automatique terminée, l'agent expert ajoute l'unité épisodique créée aux ressources de l'apprenant ou, dans le cas de la résolution automatique, aux ressources de l'expert. Finalement, bien que sur la figure 12 les flèches vers la base de connaissance sont toutes entrantes, elles illustrent le flux de données principal, l'agent expert doit aussi y faire de nombreux accès en lecture pour jouer son rôle dans le prototype.

Résolution automatique (But initial b1)

1. Créer une liste de buts à satisfaire : «buts» et une variable «but courant»

2. Ajouter b1 dans «buts».

3. Tant que b1 n'est pas satisfait :

   3.1 «but courant» = premier élément de «buts».

   3.2 Trouver les procédures applicables pour «but courant».

   3.3. Choisir la procédure «p» à appliquer

   3.4 Si «p» est une procédure complexe :
   Ajouter les sous-buts de «p» à «buts».

   Si «p» est une procédure primitive :
   a) Exécuter la procédure pour mettre à jour la base de connaissance.
   b) Mettre à jour l'état des buts / procédures dans l'unité épisodique sur le chemin entre «p» et «b1»

Suivi de l'apprenant (But initial b1)

1. Créer une liste de buts à satisfaire : «buts» et une variable «but courant»

2. Ajouter b1 dans «buts».

3. Pour chaque but «b» actif dans «buts» :

   3.1 Trouver les procédures applicables : «procs».

   3.2 Pour chaque «p» dans «procs» :
   3.2.1 Atteindre récursivement la première procédure primitive accessible depuis «p» : «pp possibles».

   3.3 Attendre l'action de l'apprenant : «pa».

   3.4 Trouver «pa» dans «pp possibles».

   3.5 Mettre à jour l'état des buts / procédures dans l'unité épisodique sur le chemin entre «pa» et «b»

Figure 13 - Les algorithmes de résolution automatique et de suivi de l'apprenant.

2.4 Les structures de représentation de connaissance

Les structures de représentation de connaissance qui encodent les connaissances sémantiques, procédurales et épisodiques, forment la base de l'architecture de connaissance. Étant le centre d'intérêt de ce mémoire, elles seront présentées en détail et illustrées dans ce chapitre, mais davantage justifiées par rapport aux lignes directrices à la fin du chapitre 3, lorsque des exemples de domaines complets auront été présentés.

2.4.1 Les connaissances sémantiques

Les structures de représentation de connaissance sémantiques décritent dans ce travail sont similaires à celles du modèle d'ASTUS, aux ontologies OWL-LD ou encore à une hiérarchie de classe dans un langage orienté objet. Les nuances choisies découlent de l'objectif de produire un modèle certes cognitif, mais surtout qui s'harmonise avec le caractère pédagogique d'un STI.

La représentation des connaissances factuelles

Bien que les outils de l'IA comptent depuis longtemps de nombreux formalismes pour encoder des connaissances factuelles, l'engouement récent pour le web sémantique[1] a suscité le développement de langages ontologiques (p. ex. OWL). La définition classique d'une ontologie est une spécification explicite et formelle de la représentation d'un domaine qui est obtenu par consensus [55]. Cette définition s'applique dans notre cas, en considérant que : 1) la représentation exprime une vision pédagogique du domaine, 2) le consensus est celui des enseignants. Par rapport au débat entourant la différence entre une ontologie et une base de connaissance[2] classique, nous dirons simplement que l'inférence produite par une ontologie est plutôt de nature descriptive, tandis que l'inférence produite par une base de connaissance déclenche généralement des actions.

On retrouve dans la littérature qui traite de ces langages de nombreux critères de comparaison qui peuvent s'appliquer à la représentation des connaissances sémantiques dans le prototype [24]. Ces critères touchent la nature des structures, leur composition, leur organisation et finalement, la capacité des moteurs d'inférences qui les manipulent. Nous retenons certains critères que nous allons examiner dans ce mémoire, parce que leur application sur l'architecture de connaissance du prototype montre les choix qui ont été faits pour suivre les lignes directrices.

1.http://www.w3.org/2001/sw/

2.Au sens usuel et non de la base de connaissance que se partage les agents du prototype.

Avant d'examiner ces critères, il faut d'abord présenter les éléments les plus souvent rencontrés dans une ontologie et sur lesquels ces derniers s'appliquent :

- Les classes qui désignent des concepts au sens large (comme les concepts dans le modèle d'ASTUS).

- Les mécanismes taxinomiques qui permettent de définir des liens de généralisation/ spécialisation (ou encore de type *is-a*) entre les classes.

- Les relations, qui définissent les interactions entre les concepts (classes). Par exemple, la composition (lien de type *has-a*) est une relation binaire entre le tout et une de ses parties.

- Les instances, les individus ou encore les faits sont l'équivalent des objets dans les langages orientés objet. Elles représentent des éléments du domaine qui appartiennent (au moins) à une classe, mais qui n'en forment pas une.

- Les axiomes sont des assertions qui restreignent l'ontologie, permettant (entre autres) ainsi de valider les instances ajoutées à l'ontologie.

- Les règles, par exemple les énoncés SWRL et les règles de production, qui permettent de déclencher des effets en fonction des autres éléments de l'ontologie.

Les critères retenus sont fonction des questions suivantes :

1) En plus des classes, peut-on définir des métaclasses, c'est-à-dire des classes dont les instances sont elles-même des classes?
2) Les attributs ou propriétés[1] sont-ils représentés distinctement des relations binaires? Si c'est le cas, est-il possible de partager les attributs entre des concepts n'appartenant pas à la même hiérarchie?

1.Au sens usuel et non au sens de OWL.

3) Est-il possible de donner une définition pour une classe (comme la classe «Parent» dans OWL au chapitre 1). Dans ce cas, peut-on inférer la taxonomie résultante? Par exemple, déduire qu'une Mère *est une* Femme [9].
4) Peut-on, à l'aide de règles, rendre opérationnelles les relations? (C'est-à-dire instancier les relations qui sont vraies étant donné les instances présentes dans l'ontologie.)
5) Est-il possible d'inférer qu'une instance appartient à une autre classe que celle dont elle origine? Si oui, est-ce en fonction de classes définies ou bien de règles de classification?
6) Distingue-t-on les fonctions des relations?
7) Quelles actions les règles peuvent-elles déclencher, en particulier peuvent-elles créer de nouvelles instances?
8) Reconnaît-on comme particulières les relations de composition (*has-a*), en particulier de façon à supporter leur nature transitive?
9) Est-ce que l'arité des relations est limitée?
10) Quelles restrictions les axiomes permettent-ils de définir sur les relations/attributs?

En somme, bien que certains langages ontologiques (p. ex. OWL+SWRL et Loom[1]), qui combinent *frames*, règles et logique, auraient pu être théoriquement utilisés comme tel pour la représentation des connaissances sémantiques, aucun d'entre eux n'est spécialement adapté aux besoins d'un STI. L'architecture de connaissance proposée, certes plus simple et limitée, a été spécialement conçue en fonction des grandes lignes dressées au début de ce chapitre. Nous présenterons l'examen des critères après avoir présenté l'architecture des connaissances sémantiques.

Le rôle des connaissances sémantiques

Dans le cadre de la résolution de problèmes dans lequel s'inscrit le prototype, les connaissances sémantiques sont avant tout considérées comme des paramètres manipulés par

1.Loom http://www.isi.edu/isd/LOOM/

les connaissances procédurales. Par exemple, dans la modélisation de la réduction d'expressions booléennes de Fournier-Viger, on ne retrouve pas le concept «Loi de DeMorgan», parce qu'aucune connaissance procédurale n'a été mise en œuvre pour interpréter (ou manipuler) ce concept. Par contre, il existe une procédure «Appliquer Loi de DeMorgan» qui manipule le concept «Expression booléenne». Anderson et al. décrit cette modélisation comme une «compilation» des connaissances sémantiques sous forme procédurale. Ceci reflète le fait que l'objectif du prototype, comme celui des MTT, est d'enseigner un savoir-faire [7].

L'architecture des connaissances sémantiques

L'**ontologie** du domaine est constituée des éléments qui représentent les connaissances sémantiques, sauf ceux qui sont propres à un problème donné. Ces éléments sont définis à partir de l'ontologie de «haut niveau» (voir figure 14) qui est indépendante du domaine.

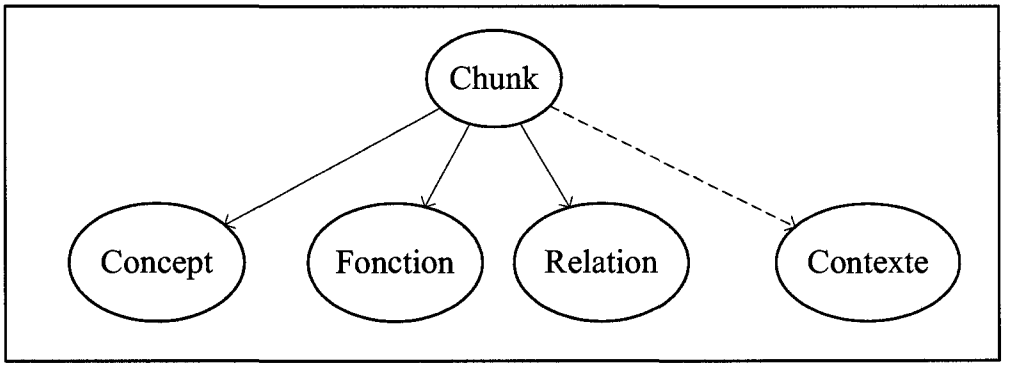

Figure 14 - L'ontologie de haut-niveau du prototype.

Dans cette ontologie, on retrouve d'abord la classe abstraite[1] des **chunks** qui se spécialise selon la nature du contenu manipulé en quatre classes : concept, relation , fonction et contexte. La classe contexte, à cause de ses caractéristiques, sera décrite avec une attention particulière plus loin dans ce chapitre. L'appellation chunk est empruntée à la psychologie cognitive où elle désigne les structures pour le stockage des connaissances sémantiques dans la mémoire à long terme.

1.Au sens classique des langages orientés objet, c.-à-d. une pour laquelle on ne peut créer d'instances.

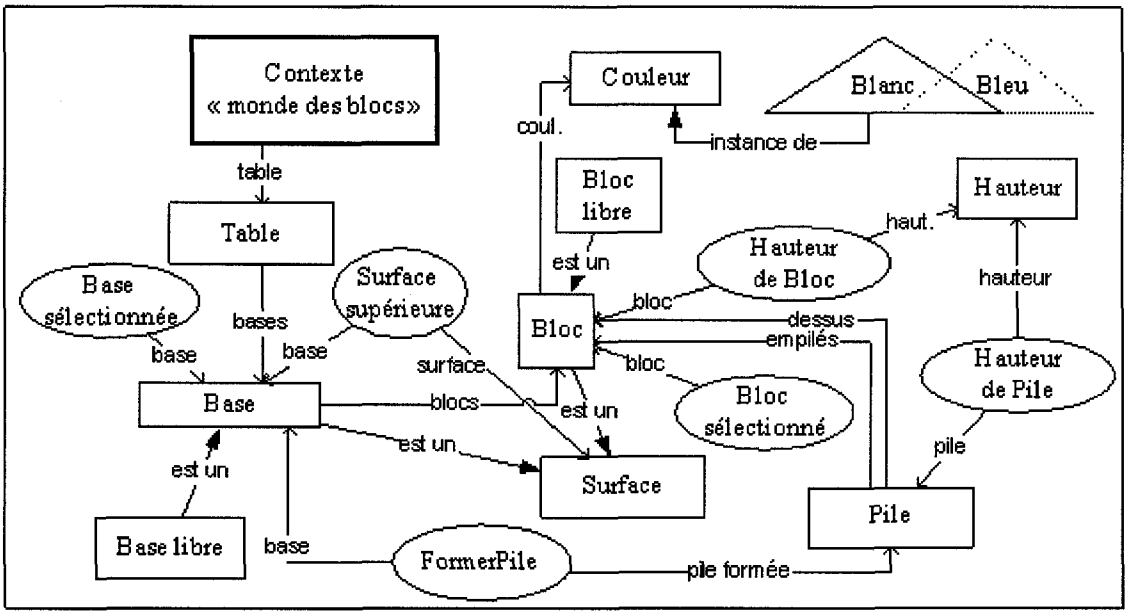

Figure 15 - Une partie des chunks pour le laboratoire du monde des blocs.
Les rectangles sont des concepts, les ellipses des fonctions/relations et les triangles des instances statiques. Les flèches à tête vide sont des attributs de type lien, les attributs ayant des valeurs primitives ne sont pas illustrés, par exemple une base a une *position* (entier).

Les instances de ces classes, simplement nommées **instances** dans la suite de ce chapitre, servent à représenter un problème à résoudre ainsi que tous les éléments qui doivent être créés pour représenter sa solution. Ces dernières ne font pas partie de l'ontologie et sont stockées dans la base de connaissance (voir figure 12) pendant la résolution du problème. Or, lors de la résolution d'un problème, une instance peut être modifiée, et ce, à plusieurs reprises; on retrouvera donc au sein des connaissances épisodiques, comme dans le modèle d'ASTUS, des cognitions qui représentent l'état ponctuel d'une instance (voir section 4.3).

Puisqu'on doit être capable de conserver l'état d'une instance à un moment précis, la structure d'une instance précise quelles données sont sauvegardées. Les caractéristiques «définissantes» d'un chunk sont représentées à l'aide d'attributs, tandis que les autres sont définies à l'aide des relations et fonctions (voir plus loin dans cette section).

Les **attributs** sont divisés en fonction du fait qu'ils acceptent comme valeur un **lien** vers une instance de chunk ou bien une **donnée primitive**. Par exemple, dans le monde des blocs, un

bloc a un attribut *nom* (chaîne de caractère) et un attribut *couleur* (lien vers une instance du chunk «Couleur», voir figure 15). La sémantique d'un attribut est donnée, comme un chunk, par son nom, mais aussi précisée par son domaine et sa portée. Le **domaine** restreint l'ensemble des chunks qui peuvent inclure cet attribut, tandis que la **portée** spécifie soit, pour les liens, les chunks qui peuvent être liés; soit pour les données primitives, le type de celle-ci (entiers, chaînes de caractère, valeur booléenne, etc.). Les liens sont soit des **liens faibles** (par défaut) ou bien des **liens forts**. Pour les liens forts, le retrait de la base de connaissance d'une instance du chunk incluant l'attribut entraînera le retrait de l'instance liée (ceci peut entraîner plusieurs retraits en cascade). Finalement, la sémantique des attributs est précisée par le rôle qu'il joue au sein d'un chunk. Ces rôles, qui sont propres à chaque classe de chunks, sont décrits dans les paragraphes qui suivent.

Les **concepts** décrivent des notions abstraites ou tangibles, atomiques ou composites[1]. Les concepts peuvent être soit constitués d'un ensemble d'attributs, soit être associés à une **règle de classification**. Par exemple, un «bloc» a des attributs, tandis qu'un «bloc libre» est associé à une règle de classification (voir figure 15). Dans les deux cas, ils peuvent être la spécialisation d'un ou plusieurs autres concepts; un concept spécialisé hérite de tous les attributs du concept plus général. Lorsqu'on associe un attribut à un concept, on doit préciser s'il joue le rôle de **propriété** ou bien celui de **composante**. Les concepts peuvent être désignés comme étant **abstraits**, et dans ce cas, ils ne peuvent pas être utilisés pour créer une instance.

Les **relations** et les **fonctions** représentent les caractéristiques des chunks et leurs interactions qui ne sont pas essentielles pour les définir. Elles sont construites à l'aide de liens jouant le rôle de **tuples** pour les relations, et d'**arguments** ou d'**image** pour les fonctions. Une **règle d'instanciation** peut leur être associée pour les instancier en fonction des autres instances présentes dans la base de connaissance. Pour les fonctions, la règle peut en plus créer l'instance qui sera associée à l'image. Dans ce cas, l'image est nécessairement un lien fort. Par exemple, la fonction «former pile» a un lien fort vers son image «pile formée» qui lie cette dernière à une instance de «pile» (voir figure 15).

1.Ici, il faut prendre tous ces termes (abstraits, tangibles, atomique, composite) au sens large.

Bien que la majorité des instances soient créées au moment de résoudre un problème donné, il existe aussi des instances dites statiques et stockées, comme les chunks, au niveau de l'ontologie. **Les instances statiques** représentent généralement des constantes du domaine. Par exemple, «Vrai» et «Faux» peuvent être modélisés comme des instances du concept «Constante de vérité» dans le domaine de la réduction d'expressions booléennes. De même, si l'on a les instances statiques «3» et «-3», on peut définir l'instance de fonction statique «InverseAdditif(«3», «-3»). Pour chaque instance statique, un chunk est généré par l'architecture de connaissance. Dans le monde des blocs, les couleurs que peuvent prendre ceux-ci sont des instances statiques (voir figure 15).

La **taxonomie d'une instance** correspond à celle du chunk à partir duquel elle a été créée, c'est-à-dire l'ensemble des chunks plus généraux que ce dernier, incluant ceux de l'ontologie de «haut niveau». Pour une instance de concepts, d'autres chunks (nécessairement des concepts), suite au déclenchement d'une règle de classification, peuvent s'ajouter à sa taxonomie.

Bien que dans la majorité des cas un lien réfère à une seule instance, il peut aussi référer à un **ensemble d'instances**. Par exemple, la valeur de l'attribut «empilés» du concept «Pile» est un lien vers un ensemble d'instances du concept «Bloc». La sémantique d'un ensemble est donnée par le chunk commun («Bloc» dans cet exemple) mais surtout par l'attribut qui le lie à une instance. En effet, ces ensembles doivent nécessairement constituer la valeur d'un attribut d'une autre instance.

Un lien peut référer à un ensemble d'instances vide, mais il peut aussi référer à la **valeur inconnue**. En effet, une valeur «nulle» n'a pas de sens pour un attribut, puisque ces derniers représentent des aspects définissants d'un concept. Toutefois, plusieurs liens sont initialisés avec la valeur inconnue puisque le problème est justement de découvrir les valeurs de ceux-ci. Par exemple, dans le problème de l'addition en colonne, la somme réfère à la valeur inconnue au départ.

Certaines **restrictions** peuvent être appliquées à un attribut lorsqu'il est associé à un chunk. Les restrictions disponibles sont : 1) rendre la portée des liens plus précise, 2) préciser la cardinalité (minimum/maximum), 3) forcer la valeur à une valeur précise, 4) marquer l'attribut comme étant immuable (c'est-à-dire pour lequel la valeur ne peut changer). Une restriction de cardinalité permet de spécifier qu'un lien référera à une instance unique et non un ensemble d'instances.

Les **contextes**, bien qu'ils soient inclus dans l'ontologie, ont un statut particulier puisque leurs instances ne sont pas explicitement manipulées par l'apprenant. Autrement dit, elles n'ont pas de correspondance dans les connaissances épisodiques. Les contextes définis pour un domaine représentent les différents regroupements d'instances qui sont d'une façon ou d'une autre regroupées dans l'interface du laboratoire. En règle générale, on retrouve un contexte pour chaque type de «fenêtre de l'interface» où l'apprenant exécute une une étape significative de la résolution du problème.

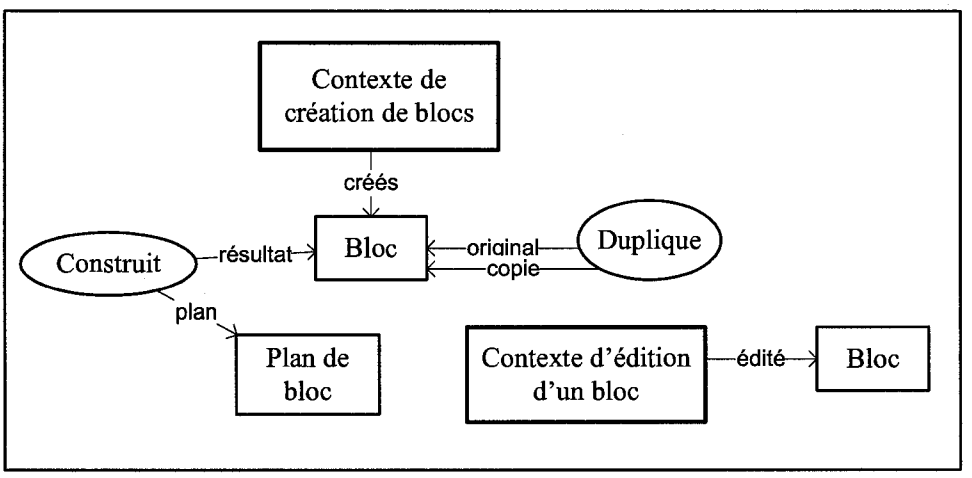

Figure 16 - Le reste des chunks pour le laboratoire du monde des blocs.

Les contextes sont définis par un ensemble de «**points d'ancrage**», c'est à dire des liens vers les chunks (généralement des concepts) qui constituent les «parties importantes du contexte». Par exemple, un contexte peut être composé d'une partie pour la donnée du problème (un schéma) et d'une pour solution (des lignes pour écrire des équations). Les interfaces des laboratoires simples ont généralement un seul contexte; dans les plus complexes, il y aura non

seulement des contextes différents, mais aussi plusieurs instances d'un même contexte. Les contextes contiennent aussi les **transitions de contextes** qu'ils offrent vers d'autres contextes. Dans certains cas, pour faire une transition, il faut spécifier une instance qui fait le pont entre le contexte de départ et celui d'arrivée. Le **pont**, comme les points d'ancrage, est un lien faible. Pour chaque contexte, il est possible de définir une **règle de création** qui déclenche la création d'une instance du contexte dans la base de connaissance. Le contexte initial doit être associé à une telle règle. Par exemple, le «contexte de création de blocs» a comme point d'ancrage «créés» qui lie ce dernier à un ensemble de blocs qui est vide au départ et le «contexte d'édition d'un bloc» a un pont «édité» qui est un lien vers le bloc qui fait l'objet de l'édition (voir figure 16).

L'environnement

Nous désignons par «environnement» l'ensemble des instances et les règles qui interagissent avec elles. L'environnement est une façade sur un système de production (Jess), les instances sont donc représentées par des *faits* et les règles de classification et d'instanciation mentionnées précédemment sont implémentées à l'aide de *règles de production*. Une instance est toujours ajoutée à l'environnement en association avec une instance de contexte. Pour la mémoire de travail du système de production, toutes les instances sont au même niveau, cependant pour l'environnement, il existe une hiérarchie assurée par trois contextes indépendants du domaine : 1) le **contexte ontologique** qui contient les instances statiques; 2) le **contexte de raisonnement** qui contient la plupart des instances créées par des règles et 3) le **méta-contexte** qui ferme la boucle en contenant les instances de contextes. De cette façon, à partir de l'instance du métacontexte, il est possible d'obtenir toutes les instances de contexte puis en suivant les liens récursivement, toutes les instances présentes dans l'environnement.

Le contexte par lequel une instance est introduite dans l'environnement précise sa sémantique. En effet, les instances des contextes propres au domaine *représentent les éléments qui sont visibles ou manipulables par l'apprenant dans l'interface du laboratoire*, elles forment donc un «modèle» au sens du *design pattern* Model-View-Controller [53]. Les instances qui sont créées dans le contexte de raisonnement représentent les inférences considérées «évidentes»

pour l'apprenant ou plus précisément, qui sont le résultat d'unités de connaissances qui ne sont pas enseignées. Les instances dans le contexte ontologique représentent des faits qui sont supposés connus par l'apprenant.

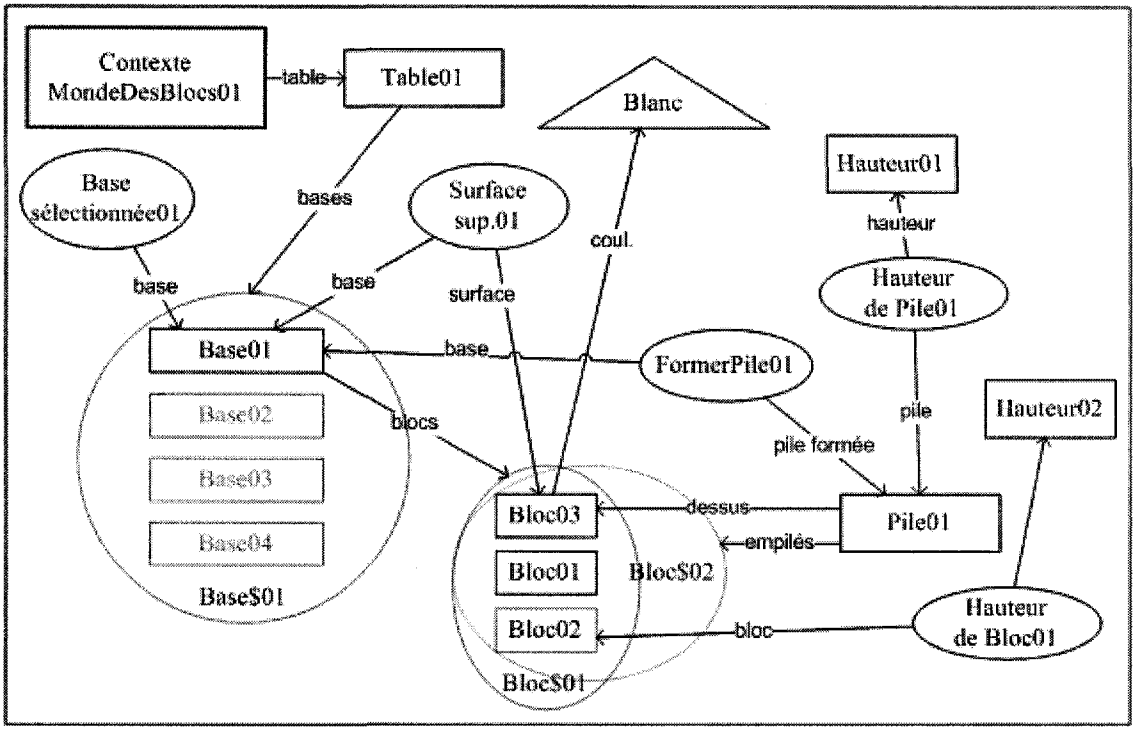

Figure 17 - Une vue partielle de l'environnement du laboratoire du monde des blocs.

L'environnement est doté d'un ramasse-miettes (*garbage collector*). En effet, puisqu'un «**lien inverse**» est généré lorsqu'un lien est créé entre deux instances, il est possible de détecter qu'une instance n'est plus liée par aucune autre et qu'elle peut donc être retirée de l'environnement. Finalement, l'environnement est également responsable de gérer le déclenchement approprié des règles lorsque des instances ou des ensembles d'instances sont modifiés (voir section 4.2.1). Par exemple, lorsque l'argument d'une fonction instanciée par une règle est modifié, l'instance de la fonction est retirée de l'environnement pour être remplacée par une nouvelle (si une image existe pour le nouvel argument). Par exemple, si un des blocs contenus dans l'ensemble de blocs lié à une base est retiré, l'instance correspondante

de la fonction «former pile» sera elle aussi retirée de l'environnement. Elle sera remplacée par une nouvelle s'il y a encore au moins un bloc «sur la base» (voir figure 17).

Examen des critères

Il n'est pas possible de définir des métaclasses au sens large (**critère 1**), toutefois, un mécanisme basé sur l'approche des *spanning objects* [62] a été conçu. Ce mécanisme consiste à définir un «*spanning* chunk» associé à une instance statique. Ce chunk peut alors être utilisé pour créer des instances qui ne seront pas dans la taxonomie de l'instance statique originale. Dans la figure 18, nous reprenons l'exemple de la littérature que nous comparons à une taxonomie très simple du domaine de l'électronique. Il y a une instance statique et un chunk

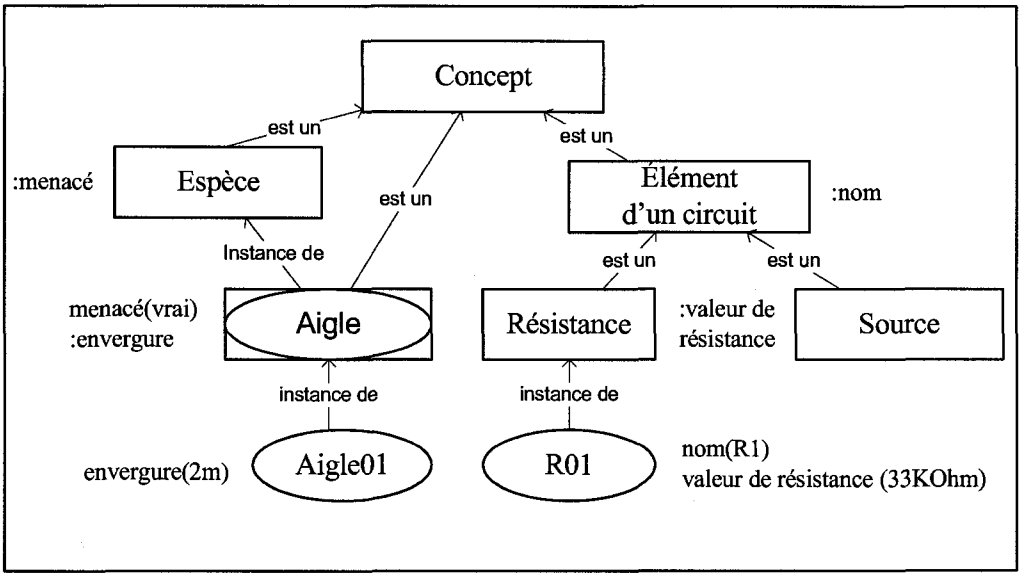

Figure 18 - Ontologie comprenant un exemple de l'utilisation des *spanning objects*.
Les rectangles représentent des concepts, les ellipses des instances. Certains attributs sont représentés.

pour «Aigle». L'instance statique donnera une valeur aux attributs propres à «Espèce» tandis que le chunk définira les attributs propres à un «Aigle». Finalement, puisque ce mécanisme n'a pas encore été utilisé pour modéliser un domaine, il n'est ni présenté ni justifié davantage.

Nous avons déjà justifié le fait de distinguer les attributs et les relations (ou fonctions). Nous avons choisi de permettre l'utilisation des mêmes attributs pour, entre autres, éviter les redondances lorsque l'auteur d'un domaine ajoute des connaissances didactiques (**critère 2**).

Il n'est pas possible d'avoir des concepts définis, puisque la conception et l'implémentation de tels formalismes dépassaient largement les objectifs de ce travail (**critère 3**). Toutefois, un mécanisme pour générer une ontologie OWL à partir des chunks est en cours de conception. Cette ontologie serait utilisée seulement lors du développement d'un nouveau domaine, de façon à : 1) pouvoir bâtir une taxonomie qui prend en compte les relations de subsomptions qui sont habituellement implicites, 2) découvrir plus facilement les incohérences.

Il est possible de rendre les relations et les fonctions opérationnelles à l'aide de règles (**critère 4**). En plus de représenter de façon «boîte noire» des connaissances qui n'ont pas d'intérêt pédagogique en elles-mêmes, cela permet de réduire le nombre d'instances qui doivent être spécialement créées pour un problème. L'architecture distingue les relations des fonctions (**critère 6**) pour deux raisons : les règles associées aux fonctions peuvent créer l'instance associée à l'image (**critère 7**); il est plus facile pour l'agent pédagogue de décrire les instances de façon générique ainsi.

La classification des instances à partir de règles de classification (**critère 5**) réduit également la quantité d'information à fournir pour un problème donné. De plus, elle rend le processus de suivi plus puissant (voir section 2.6.2).

Bien qu'on distingue les propriétés des composantes au niveau des concepts (**critère 8**), c'est davantage le type du lien (fort ou faible) qui influence le comportement de l'environnement, par exemple pour le retrait en cascade.

L'architecture ne limite pas le nombre de tuples dans les relations, ni le nombre d'arguments dans les fonctions; ceci évite que les relations ou les fonctions soient modélisées comme des concepts avec de nombreuses composantes (**critère 9**). Toutefois, puisque les tuples et les arguments sont eux-mêmes des attributs, il n'y a pas à proprement parler de «relations» n-aires dans l'architecture de connaissance.

Les restrictions sur les attributs qui peuvent être spécifiées au niveau des chunks (**critère 10**) servent 1) à découvrir les erreurs de modélisations plus facilement, 2) à préciser les descriptions génériques produites par l'agent pédagogue.

2.4.2 Les connaissances procédurales

Les caractéristiques fondamentales de la représentation des connaissances procédurales présentées au chapitre 1 continuent de s'appliquer pour l'architecture de connaissance du prototype. Par conséquent, les paragraphes qui suivent mettront l'emphase sur les changements qui ont été apportés pour suivre les lignes directrices définies au début de ce chapitre.

En plus de réitérer que la représentation des connaissances procédurales dans ASTUS est une approche intermédiaire entre celle des MTT et celle d'Andes, il est intéressant de la comparer aux formalismes de planification hiérarchique [25]. En effet, on peut comparer les buts satisfaits par des procédures complexes aux *tâches non primitives* et les buts satisfaits par des procédures primitives aux *tâches primitives*. De la même façon, les procédures complexes peuvent être comparées aux *méthodes*, les requêtes aux *préconditions*, et les procédures primitives aux *opérateurs* ou *actions*. Dans les faits, cette comparaison n'est pas réellement applicable, puisque les connaissances procédurales d'un domaine correspondent davantage aux sorties (les *plans*) d'un planificateur qu'à ses entrées (les éléments mentionnés plus haut). En effet, bien que les connaissances procédurales soient dynamiques, il n'en demeure pas moins qu'elles ne sont pas accompagnées d'un mécanisme décisionnel puissant, tel qu'un planificateur, comme nous l'avons mentionné à la fin du chapitre précédent.

Bien qu'il aurait été possible d'ajouter un planificateur à l'architecture de connaissance afin de rendre plus puissantes les connaissances procédurales, cela irait à l'encontre des lignes directrices. En effet, pour les domaines où d'importantes décisions doivent être prises en fonction des données particulières d'un domaine, il serait fort peu intéressant d'avoir recours à un processus *boîte noire*[1] pour les prendre.

Accès à l'environnement

Puisque les connaissances procédurales puisent leurs arguments et appliquent leurs effets dans l'environnement, il est nécessaire avant de les décrire, de présenter les mécanismes d'accès à cette dernière.

Les connaissances procédurales ont accès à l'environnement en lecture, grâce aux requêtes, et en écriture, grâce aux scripts d'actions. Les **requêtes** permettent d'obtenir une instance ou un ensemble d'instances à partir de l'environnement. Même s'il est possible de créer des requêtes propres à un domaine, les requêtes prédéfinies sont préférées afin de standardiser les modèles et ainsi faciliter le travail de l'agent pédagogue. Pour chaque requête, différentes données doivent être fournies par la connaissance procédurale, au minimum il s'agit de : 1) un chunk représentant le type du résultat, 2) l'instance de contexte dans lequel faire la requête et 3) un nom local pour le résultat. Voici les requêtes prédéfinies qui sont les plus fréquemment utilisées :

- obtenir l'unique, ou toutes les instances représentant un chunk,
- obtenir une instance (ou un ensemble) à partir d'un lien d'une autre instance,
- obtenir l'image d'une fonction (on peut contraindre les arguments),
- obtenir une instance statique à partir de son nom,
- obtenir l'ensemble des instances qui satisfont une relation (unaire ou binaire) avec une instance donnée.
- les sous-requêtes qui à partir d'un ensemble d'instances retournent un résultat plus précis : la différence, l'union et l'intersection avec un autre ensemble; le «sous-ensemble» qui retourne seulement les instances représentant un chunk plus précis et

1.Boîte noire dans le sens où demander à l'agent pédagogue de décrire les décisions de l'agent expert à l'apprenant à partir de la trace d'éxécution d'un algorithme tel qu'A* n'est pas une solution enviable.

«l'application», qui à partir d'un ensemble et d'une fonction, retourne l'ensemble des images.

- Les requêtes sur les connaissances épisodiques (voir section 4.3.2).

La plupart des requêtes sont encodées sous forme de règles de production, mais certaines sont directement implémentées à l'aide de code Java.

Les **scripts d'actions** (en Java) permettent la manipulation des instances contenues dans l'environnement. En fait, les scripts peuvent :

1) créer une nouvelle instance ou un nouvel ensemble d'instances,
2) modifier la valeur d'un attribut d'une instance de concept,
3) ajouter ou retirer une instance dans un ensemble.
4) ajouter ou retirer une instance de fonction ou de relation
5) obtenir le résultat d'une requête.

L'environnement conserve une trace de l'exécution de chaque action contenue dans le script, de façon à rendre possible l'annulation (*undo*) de ce dernier.

Les buts

Comme dans le modèle précédent d'ASTUS, les **buts** représentent une intention qui est précisée à l'aide de paramètres et qui peut être satisfaite par une procédure (voir section suivante). Bien que les buts soient en réalité des connaissances sémantiques, leur usage, présentement limité, fait en sorte qu'il convient de les présenter avec les connaissances procédurales. Un **paramètre** est spécifié par un nom local et un chunk que devra représenter l'argument (de façon semblable à ce qu'on retrouve dans une définition de méthode dans un langage statiquement typé). Un but contient une **requête de contexte** qui servira à vérifier que le contexte dans lequel le but peut être satisfait est le contexte courant de l'environnement, lorsqu'une occurrence du but est créée dans l'unité épisodique courante (voir section 4.3.1).

Les procédures

Comme les buts, les **procédures** sont définies à l'aide d'un ensemble de paramètres et de requêtes. Comme dans le modèle précédent d'ASTUS, une procédure est créée pour satisfaire un but particulier, les paramètres de celle-ci doivent donc être compatibles avec ceux du but. Puisque l'unification des paramètres est établie par les noms locaux (des paramètres/requêtes), le chunk peut être plus spécialisé au niveau de la procédure.

Les procédures sont, comme dans le modèle précédent d'ASTUS, divisée en deux catégories, les **procédures complexes** et les **procédures primitives**. Cependant, le «pouvoir de calcul» des procédures complexes est défini clairement et la sémantique des procédures primitives est fortement restreinte.

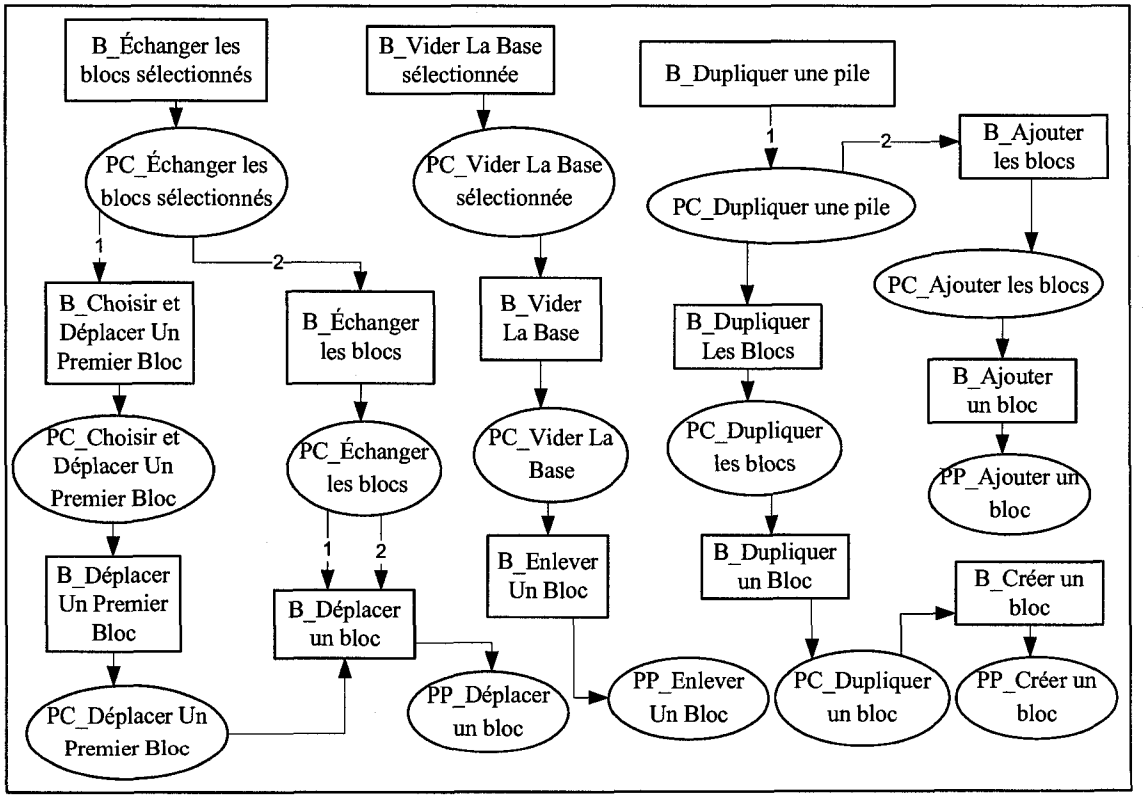

Figure 19 - Des connaissances procédurales pour le laboratoire du monde des blocs.

Les procédures complexes représentent des façons de faire (méthode, recette, algorithme, etc.) qui ont un intérêt pédagogique. Puisque les structures de base des algorithmes sont la séquence, la sélection et l'itération [7], nous avons créé trois catégories de procédures complexes qui produisent lors de leur exécution une ou plusieurs occurrences de chacun de leurs sous-buts. Un **sous-but** est un adaptateur entre la procédure complexe et le but qu'elle veut solliciter. Les sous-buts sont nécessaires puisque différentes procédures complexes peuvent solliciter un même but. La principale fonction d'adaptation est d'établir une correspondance entre les noms locaux de la procédure et ceux des paramètres du but.

La **procédure-séquence** spécifie un ensemble de sous-buts qui doivent être satisfaits et elle contraint, de façon partielle ou totale, l'ordre dans lequel ces sous-buts doivent être satisfaits (voir *PC_DupliquerUnePile* sur la figure 19).

La **procédure-sélection** se spécialise en la **procédure-conditionelle** et la **procédure-choix**. Pour la procédure-conditionelle, on retrouve un ensemble de conditions auquel un sous-but est lié. Une condition est une expression booléenne implémentée par une règle de production qui inclut généralement des tests sur la valeur des arguments. Les conditions sont évaluées dans l'ordre et dès que l'une entre elles est remplie, le sous-but correspondant est retenu. **La procédure-choix** offre un même sous-but pour chaque instance d'un ensemble d'instances (voir *PC_DéplacerUnPremierBloc* sur la figure 19). Un seul de ces sous-buts doit être satisfait pour compléter la procédure. Le sous-but doit évidemment avoir un paramètre qui peut accueillir une des instances issues de l'ensemble.

La **procédure-itération** spécifie un sous-but qui doit être répété et un itérateur qui détermine le nombre d'itérations nécessaire. Plutôt que d'offrir uniquement un itérateur universel basé sur une «condition», quatre itérateurs, issus des expériences de modélisation, ont été implémentés. La **répétition**, qui fournit un nombre fixe d'itérations, le **pour** qui produit un nombre d'itérations dépendant de la valeur d'une propriété d'une instance; le **pour-chaque** qui crée une itération pour chaque instance appartenant à un ensemble d'instances (voir *PC_AjouterLesBlocs* sur la figure 19); et le **tant que** qui produit des itérations jusqu'à ce qu'une condition se vérifie.

Par rapport au modèle d'ASTUS, on peut toujours spécifier qu'une procédure complexe est valide ou invalide, mais il n'est plus possible d'indiquer qu'elle ne satisfait pas toujours le but. En effet, même si cette option était offerte dans le passé, elle n'a jamais vraiment été prise en compte. Finalement, on peut aussi préciser une priorité qui sera utilisée, entre autres, lors de la résolution automatique.

Les **procédures primitives** représentent des actions atomiques qui correspondent pour l'apprenant à une manipulation de l'interface du laboratoire. De *Miace* au MGC, une procédure primitive pouvait retourner une valeur à la procédure complexe qui l'avait déclenchée, ceci permettait à un but d'obtenir le «résultat» d'un but précédent. Cette façon de faire, hérité du langage Lisp et cognitivement discutable était nécessaire pour passer le «résultat» d'une procédure. Puisque dans le prototype les procédures primitives agissent sur l'environnement, ce sont les requêtes qui permettent à un but de retrouver un «résultat» antérieur. À chaque procédure primitive, on associe un script d'action qui est déclenché automatiquement lors de la résolution automatique et lorsque l'agent apprenant reconnaît que l'apprenant a accompli cette dernière lors du suivi.

Les scripts d'actions sont aussi associés aux transitions de contexte. Dans ce cas, le rôle du script est plus précis : il doit fournir, en la créant au besoin, l'instance du contexte de destination. Finalement, un script d'action peut aussi être associé à l'initialisation d'un contexte, c'est d'ailleurs ces derniers qui créent généralement la plupart des instances.

2.4.3 Les connaissances épisodiques

La représentation hiérarchique et exhaustive des connaissances épisodiques est une autre originalité du modèle d'ASTUS. Comme pour les connaissances procédurales, les modifications apportées pour l'architecture de connaissance du prototype découlent davantage de l'expérimentation que de la comparaison avec d'autres STI.

Un épisode est défini simplement comme une occurrence de but avec un ensemble d'occurrences de procédures qui ont été considérées ou utilisées pour satisfaire ce dernier. La

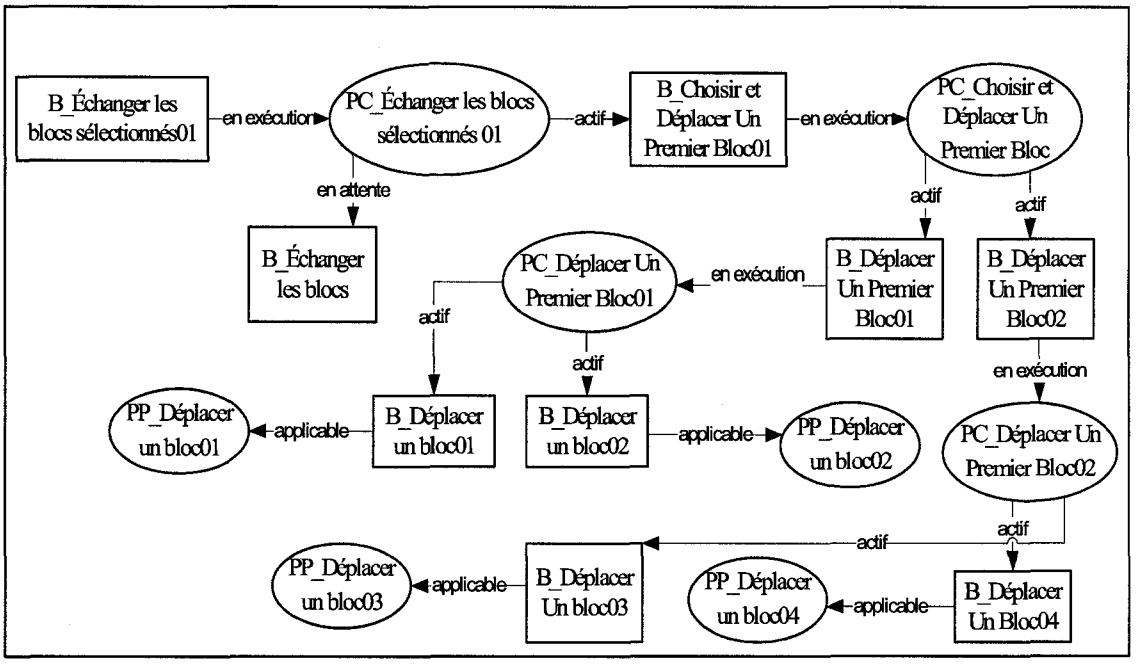

Figure 20 - Épisode courant pour le problème «échanger les blocs sélectionnés».

mémoire épisodique contient un ensemble d'unités épisodiques qui contiennent à leur tour tous les épisodes concernés par la résolution d'un même problème. Les occurrences de buts et de procédures sont définies en grande partie par leurs arguments, c'est-à-dire l'unification des noms locaux de leurs paramètres/requêtes et de leurs valeurs. Ces valeurs sont des sauvegardes d'une instance à un moment précis que nous nommons **cognition**. Autrement dit, une cognition contient une copie des valeurs d'attributs d'une instance ainsi qu'une estampille temporelle. L'implémente évite de copier ces valeurs lorsqu'une instance est statique ou immuable. Il est possible de créer une cognition d'un ensemble d'instances, qui au moment de sa création, crée une cognition de chacune des instances contenues dans l'ensemble. À chaque occurrence de but/procédure, on associe un état (voir tableau 3 et figure 20). De plus, à chaque occurrence de procédure est associé un résultat. Dans le cas des procédures complexes, il s'agit d'un ensemble de sous-épisodes, tandis que pour les procédures primitives, il s'agit d'un identificateur qui permet de retrouver des détails supplémentaires sur son déclenchement, si celle-ci a été faite par l'apprenant (voir le journal des actions au chapitre 4).

Tableau 3 - Les états des occurrences de buts/procédures

États des occurrences de buts	**États des occurrences de procédures**
Inactif : Une procédure a sollicité le but, mais ne l'a pas encore activé.	*Applicable* : Une procédure dont les paramètres correspondent aux arguments du but.
Prêt : Un but dont l'ensemble des requêtes a obtenu leur résultat.	*Non applicable* : voir état précédent.
En attente : Un but qui attend qu'un autre but soit satisfait ou qui n'est pas prêt.	*En exécution* : Une procédure, dont au moins un des sous-buts n'est toujours pas satisfait (le but est atteint ou échoué).
Actif : Un but pour lequel on a déterminé l'ensemble de procédures applicables.	*Annulée :* une procédure en exécution ou terminée qui a été annulée par le pédagogue ou l'apprenant.
Abandonné : Un but pour lequel aucune procédure n'a été terminée.	*Terminée :* Une procédure dont tous les sous-buts sont satisfaits (atteint ou échouée).
Atteint : Un but pour lequel une procédure valide s'est terminée.	*Rejetée* : Une procédure applicable qui n'a pas été mise en exécution.
Échoué : Un but pour lequel une procédure invalide s'est terminée.	*Avortée* : Une procédure dont l'un des sous-buts est abandonné.

L'unité épisodique courante

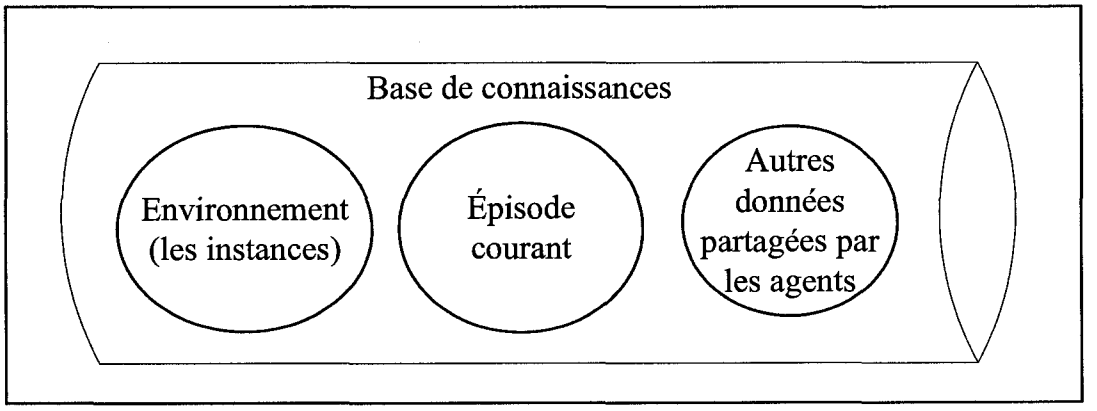

Figure 21 - Un aperçu de la base des connaissances du prototype.

L'unité épisodique courante, comme l'environnement, se trouve dans la base de connaissance (voir figure 21). Qu'elle soit créée par la résolution automatique ou le suivi de l'apprenant, elle contient à la fois les épisodes en cours, mais aussi les épisodes à venir à court terme. Par

exemple, une occurrence de procédure-séquence peut ajouter à l'unité épisodique courante plusieurs occurrences de but qui sont en attente. La création d'une occurrence de procédure donne lieu à l'instanciation d'un interpréteur spécialisé (il y en a un pour chacune des variantes de procédure). Celui-ci conserve un état interne permettant de suivre le progrès de la procédure au fur à mesure où des sous-buts sont satisfaits. Il permet par exemple, dans le cas de la procédure-séquence, de mettre en attente les sous-buts en fonction des contraintes d'ordre et de les activer lorsque le but requis est satisfait. Par exemple, *B_AjouterLesBlocs* est en attente dans la figure 22.

Dans le modèle précédent d'ASTUS, les connaissances épisodiques sont plutôt de nature passive, c'est le MGC qui les crée au fur et à mesure. Dans l'architecture de connaissance du prototype, l'intégration de ces dernières aux processus de résolution/suivi les rend plus dynamiques. Toutefois, ceci relève plus du design que de la théorie sous-jacente.

Les requêtes sur les connaissances épisodiques

La modélisation de certaines procédures complexes nécessite la capacité de tenir compte des choix procéduraux effectués précédemment. Pour cette raison, il est possible de : 1) faire une requête pour retrouver une occurrence de but; 2) à partir d'une occurrence de but retrouvée, de faire une requête pour obtenir la valeur d'un argument de cette dernière. Une requête pour retrouver un but peut être contrainte en donnant la valeur d'un des arguments ou bien en précisant la procédure utilisée. Si plusieurs occurrences de but correspondent aux critères de la requête, la plus récente est retournée. Par exemple, dans le monde des blocs, une requête pour trouver quel bloc a été déplacé en premier est possible en retrouvant la dernière occurrence du but «déplacer bloc» (voir figure 20).

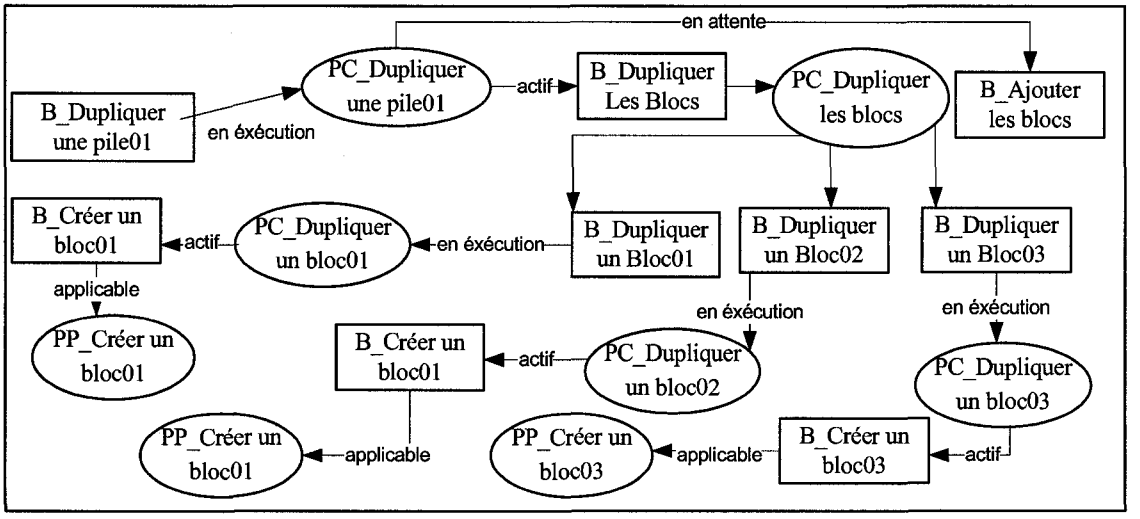

Figure 22 - Épisode courant pour le problème «dupliquer une pile».

2.5 La résolution automatique

La résolution automatique est principalement un outil pour l'auteur du domaine. En effet, elle est d'abord conçue pour aider la modélisation des connaissances du domaine et la validation des problèmes. On peut aussi envisager de faire du forage de données (*data mining*) sur les unités épisodiques obtenues à partir de la résolution automatique pour découvrir des propriétés du modèle de connaissance ou bien des problèmes offerts.

Il y a trois modes de résolution automatique : 1) le mode passif où l'agent expert sélectionne toujours le but le plus récemment ajouté à liste des buts actifs et la procédure valide applicable ayant la plus haute priorité; 2) le mode actif où l'auteur doit choisir lui-même l'ordre des buts et les procédures pour les satisfaire à l'aide d'une boîte de dialogue; 3) le mode « en lot » où l'on doit spécifier le nombre de résolutions qui seront faites en mode passif. Ce dernier mode sera particulièrement utile pour faire du forage de données ou bien pour explorer l'utilisation d'algorithmes d'apprentissage automatique (*machine learning*). Dans les deux premiers modes, l'interface du laboratoire est mise à jour pour faciliter le travail de l'auteur. Dans tous

les modes, le processus de résolution automatique déclenche automatiquement les transitions de contexte qui sont nécessaires.

2.6 Le suivi de l'apprenant

Non seulement le modèle de connaissance doit permettre la résolution automatique, mais aussi le suivi d'un apprenant qui fait des erreurs. Le suivi opéré par l'agent expert tient cependant pour acquis que l'apprenant tente bel et bien de résoudre le problème et que s'il commet une erreur, c'est parce qu'il a des lacunes. De plus, le suivi se fait en grande partie sans avoir recours à un modèle de l'apprenant, c'est seulement lorsqu'une ambiguïté persistante est détectée qu'une requête est envoyée à l'agent apprenant (voir chapitre 4).

Puisque d'une part, pour chaque but actif toutes les procédures applicables sont mises en exécution, et d'autre part, parce que c'est l'apprenant qui déclenche les transitions de contexte, des états supplémentaires doivent être ajoutés pour les occurrences de buts/procédures lors du suivi :

- «Non-tracé» (par défaut);
- «Tracé» (lorsqu'une ou plusieurs procédures primitives sous-jacentes ont été effectuées par l'apprenant);
- «Traçage ambigu» (lorsqu'il y a compétition entre plusieurs chemins pour expliquer la procédure primitive signalée par l'agent interface);
- «Transition de contexte requise» (pour les occurrences de buts seulement).

Le suivi est mis à jour suite au signal d'une procédure primitive ou d'une transition de contexte. Le processus de suivi produit un résultat que l'agent expert transmet à aux autres agents. Ce résultat peut être :

1) «Traçage mis à jour» avec l'ensemble des procédures primitives possibles;
2) «Traçage bloqué» (transition de contexte requis, but échoué, etc.);

3) «Fin du suivi» (le but racine est satisfait);
4) «Erreur» (le traçage est bloqué, aucune transition n'est possible et le but racine n'est pas satisfait, le problème se situe dans la modélisation du domaine).

2.6.1 Les ambiguïtés

Normalement, lorsqu'une procédure primitive est effectuée est qu'elle est reconnue comme une des procédures primitives possibles, l'état des occurrences de but/procédure est modifié en remontant le chemin jusqu'à l'occurrence du but racine. Or, dans certains cas, par exemple lorsqu'une procédure complexe invalide ressemble de près à une procédure complexe valide, deux occurrences d'une même procédure primitive, avec les mêmes arguments, peuvent se retrouver dans l'ensemble des procédures primitives possibles. Dans cet exemple, les occurrences de but/procédure sur les deux chemins partant des occurrences de procédures primitives vers la racine se verront attribuer l'état «traçage ambigu ».

Ce qui permet de lever ces ambiguïtés, c'est que, généralement, deux procédures complexes différentes engendrent au moins une occurrence d'une procédure primitive différente dans son exécution. L'ambiguïté est donc levée au moment où une occurrence de procédure primitive appartenant seulement à l'un des chemins est reconnue.

Cependant, il peut arriver que l'agent expert se trouve face à une ambiguïté persistante, c'est-à-dire qu'au moins deux procédures complexes différentes ont produit les mêmes occurrences de procédures primitives. Ceci est fréquent lorsque plusieurs procédures satisfaisant un même but ont les mêmes sous-buts, mais des requêtes différentes. En effet, pour certains problèmes, des requêtes différentes peuvent produire des résultats identiques. Plus rarement, l'ambiguïté survient lorsque deux procédures qui peuvent satisfaire le but sont de types différents (séquence vs répétition) et que la donnée du problème fait en sorte que les occurences de buts engendrés sont les mêmes. Par exemple, une répétition qui fait une seule itération et une séquence qui a un seul sous-but (le même que celui associé à l'itération).

2.6.2 La comparaison des arguments

Lorsqu'on doit comparer la procédure primitive déclenchée par l'apprenant à celles prévues par le suivi, la difficulté réside dans la comparaison des arguments. Plus précisément, il faut voir si les instances que l'agent interface transmet à l'agent expert lorsqu'il signale qu'une procédure primitive a été effectuée peuvent être unifiées aux cognitions prévues par le processus de suivi. Les instances envoyées par l'agent l'interface peuvent être issues de deux origines différentes. D'une part, il peut s'agir d'instances représentées dans l'interface du laboratoire que l'apprenant a manipulée, par exemple en faisant une sélection dans un menu déroulant. D'autre part, il peut s'agir d'une instance qui a été créée à la suite d'une saisie effectuée par l'apprenant, par exemple écrire un chiffre dans une boîte de texte (la récolte des instances par l'agent interface sera présentée plus en détail au chapitre 4).

L'unification des premières, nommées **sélections**, est sans équivoque. Il suffit de vérifier dans l'environnement que l'instance à l'origine de la cognition est bien celle fournie par l'agent interface. Pour les deuxièmes, nommées **entrées**, il y a deux mécanismes possibles pour vérifier la correspondance, la **classification** et la **comparaison**.

Avec le mécanisme de classification, le processus de suivi considère qu'une instance entrée sera reconnue équivalente à l'argument si sa taxonomie contient l'ensemble des chunks associés aux requêtes/paramètres qui ont porté cette dernière sur le chemin entre l'occurrence du but racine et l'occurrence de la procédure primitive. Pour ce faire, à chaque fois qu'un argument est passé d'un but à une procédure ou d'une procédure à un sous-but, le chunk associé est conservé pour qu'au niveau des occurrences de procédures primitives on retrouve l'ensemble des chunks que l'argument représente. La pertinence de l'approche par classification ressort lorsqu'on veut accepter des saisies inexactes ou équivalentes, puisque la règle de classification associée au concept permet une grande flexibilité. Par exemple, si on considère le concept «Nombre» qui se spécialise en «Nombre pair» et «Nombre impair» et qu'on demande à l'apprenant de faire une saisie, il est plus intéressant de classifier l'instance de «Nombre» qu'il a crée, que de la comparer à une foule d'instances préexistantes qu'on connaît être paires ou impaires.

Toutefois, cette approche est inutilement lourde quand la saisie doit être exacte. En effet, dans ce cas, il vaut mieux comparer l'instance crée par l'apprenant à celles créées par le processus de suivi. Pour ce faire, un **comparateur** peut être associé à un concept; il peut s'agir d'une simple comparaison sur la valeur d'un attribut «clé», ou bien d'un appel à un algorithme de comparaison complexe. Par exemple, si l'on demande à l'apprenant de saisir la «Somme» de deux «Nombre», la comparaison de la somme produite par ce dernier avec celle produite par le suivi est suffisante, tandis que des concepts tels que «Somme juste» et «Somme erronée» seraient superflus.

2.7 Le noyau de simulation du laboratoire

Dans certains domaines, les instances et les scripts d'actions ne sont pas suffisants pour gérer l'ensemble des manipulations possibles dans l'interface du laboratoire. En effet, en dehors des domaines formels comme les mathématiques, l'interface du laboratoire peut agir comme un simulateur. Dans ce cas, un noyau de simulation doit être implémenté (minimalement par une API Java) pour effectuer les différents calculs nécessaires, par exemple le résultat d'une réaction chimique. Tout ce qui est dans le noyau est inconnu au niveau des agents et de la base de connaissance, par contre les éléments visibles ou manipulables sont associés à des instances. Les scripts d'actions (associés aux procédures primitives ou aux transitions de contexte) peuvent faire appel aux méthodes du noyau afin de modifier l'environnement.

Lorsque le noyau possède un état interne, l'annulation des scripts d'actions ne peut plus être faite automatiquement, il faut alors que l'auteur du domaine fournisse un script d'annulation complémentaire. Finalement, même dans les cas où un noyau de simulation est nécessaire, le comportement des scripts d'actions doit demeurer déterministe pour que toutes les fonctionnalités du prototype soient disponibles, par exemple, la possibilité d'arrêter et de reprendre la résolution d'un problème (voir chapitre 4).

2.8 Conclusion

Ce chapitre a présenté une architecture de connaissance conçue à partir de lignes directrices qui découlent des lacunes de l'architecture d'ASTUS et de l'exploration d'approches alternatives comparables. De plus, les exemples tirés du *monde des blocs* ont permis d'illustrer comment l'architecture de connaissance permet à l'agent expert de remplir ses fonctions, en particulier la résolution automatique et le suivi de l'apprenant, au sein du prototype. Une fois que nous aurons présenté des modèles de domaines plus consistants au chapitre 3, nous discuterons de comment l'architecture de connaissance atteint les objectifs ciblés.

Chapitre 3

Exemples de modèles de connaissance

Ce chapitre, décrit trois modèles de connaissance pour des domaines simples[1] qui ont été conçus soit pour valider l'architecture de connaissance ou la conception de l'agent interface : l'addition en colonnes, l'addition de fractions et la «machine». Les modèles ont été faits sans porter directement attention à leur potentiel pédagogique. Toutefois, ils reflètent un niveau de complexité semblable à celui des exemples fournis par d'autres STI dotés d'une architecture de connaissance indépendante du domaine (p. ex. CTAT). En fin de chapitre, nous discutons de l'architecture de connaissance en fonction des lignes directrices établies au chapitre 2.

3.1 L'addition en colonnes

Ce laboratoire réifie la technique classique de l'addition de nombres naturels en colonnes. Pour effectuer une addition, il faut additionner mentalement les deux opérandes et la retenue d'une colonne, en extraire les unités et les dizaines et écrire le résultat (les unités) et le report. Ce dernier sera la retenue de la colonne suivante. Le report de la dernière colonne, s'il existe, se fait à gauche de la dernière somme. La modélisation du domaine décrit dans ce chapitre se veut minimaliste et vise seulement à illustrer la différence principale entre une modélisation à l'aide de procédures complexes et de règles de production. De plus, pour simplifier le travail de l'agent apprenant, la somme et le report sont choisis à l'aide de menus déroulants plutôt que d'être entrés dans des boîtes de texte (voir figure 23). Autrement dit, il s'agit de *sélections* plutôt que d'*entrées*. Aucune procédure invalide n'a été modélisée, puisque nous désirions avant tout tester la capacité de l'agent pédagogue à fournir une aide précise sans avoir recours à des connaissances didactiques.

1. Voir aussi les exemples tirés du monde des blocs au chapitre 2.

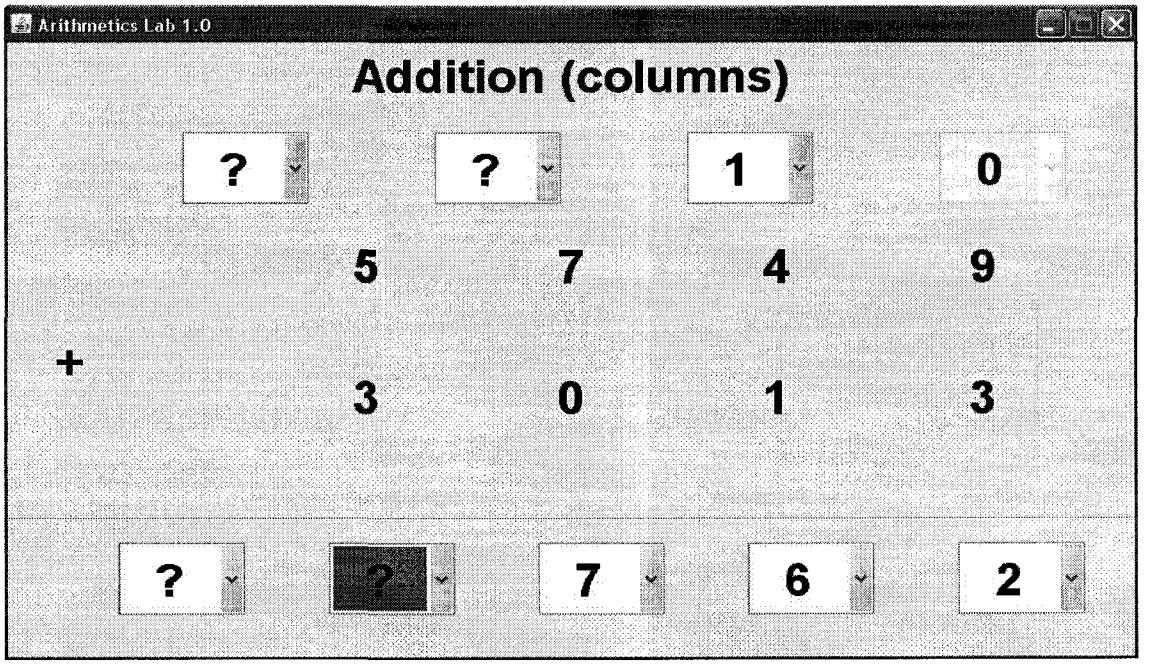

Figure 23 - Une capture d'écran du laboratoire pour l'addition en colonnes.

3.1.1 Les connaissances sémantiques

Puisqu'aucune instance n'est directement créée par l'apprenant pour représenter ses saisies, on retrouve dans l'ontologie dix instances statiques qui représentent les chiffres de 0 à 9 (voir figure 24). Un chiffre a un attribut *valeur* de type entier, cette donnée primitive est utilisée pour les calculs de somme et de report. Les concepts de base de l'ontologie sont l'addition et la colonne. Il y a un seul contexte, le contexte principal, dont le seul point d'ancrage est l'addition dont l'instance est créée par le script d'initialisation. Ce dernier crée aussi les instances de colonnes et l'ensemble d'instances qui les contient et qui est lié à l'addition. Une colonne a six attributs : une *position* (un entier), les deux *opérandes*, son *report* sur la colonne suivante, sa propre *retenue* et son *résultat*. Les trois premiers sont initialisés au départ, tandis que les autres réfèrent à la valeur inconnue (sauf la première colonne dont la retenue est zéro). Des règles de classification identifient la première et la dernière colonne, puis les colonnes «indépendantes» et «sans report». Les colonnes «sans report» ont un résultat inférieur à dix, tandis que les colonnes «indépendantes» sont les colonnes qui suivent les colonnes «sans

report».(p. ex la colonne 3, "7+0", est une «colonne indépendante» puisque 1+4+1=6<10). Des règles d'instanciation créent les instances des fonctions «colonne suivante» et «colonne précédente» à partir de la position des colonnes. Les instances des fonctions «somme»[1] et «report» sont également créées à l'aide de règles d'instanciation. Ces deux dernières n'ont pas besoin de créer l'instance correspondant à leur image, puisqu'il s'agit d'un chiffre.

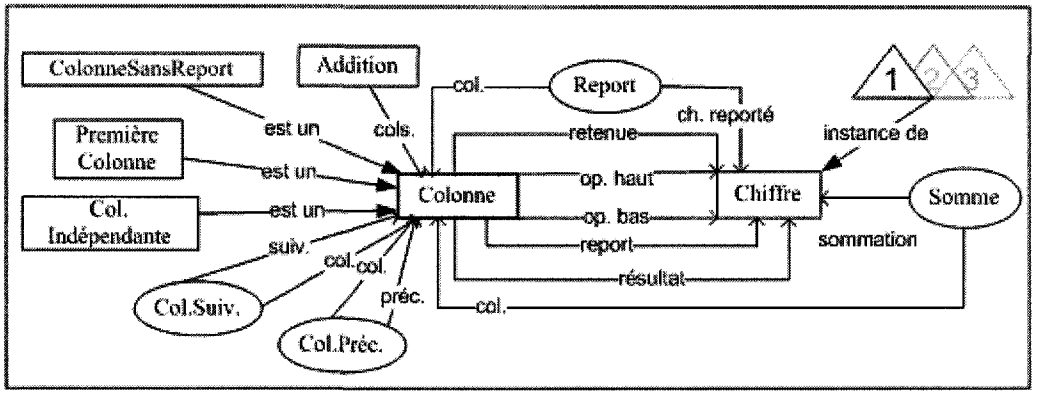

Figure 24 - L'ontologie du laboratoire pour l'addition en colonnes.

3.1.2 Les connaissances procédurales

Le but «additionner colonnes» (voir figure 25) fait une requête pour obtenir l'addition (elle est unique), ce résultat est passé à la procédure «additionner colonnes». Cette procédure complexe est une itération de type *pour-chaque* qui itère sur l'ensemble des colonnes qu'elle obtient par requête sur l'addition reçue en paramètre. Puisque c'est un pour-chaque *ordonné* (il faut additioner les colonnes de droite à gauche), un ensemble de sous-buts «additionner colonne» est créé, mais un seul est actif (les autres sont en attente). L'ordre de création des sous-buts est identique à l'ordre de l'ajout des instances de colonnes dans l'ensemble associé à l'addition, ordre qui correspond à la position des colonnes de droite à gauche. La procédure complexe de type séquence «additioner colonne» a deux sous-buts sans contrainte d'ordre : «écrire somme» et «écrire report». Ces deux sous-buts ont comme paramètre une colonne et l'image d'une fonction, celle de la fonction «somme» pour le premier et celle de la fonction «report» pour le deuxième. Ces deux requêtes sont exécutées par la procédure «aditionner colonne». La

1.Il s'agit en fait de la somme en arithmétique modulo 10, puisqu'on retient que les unités.

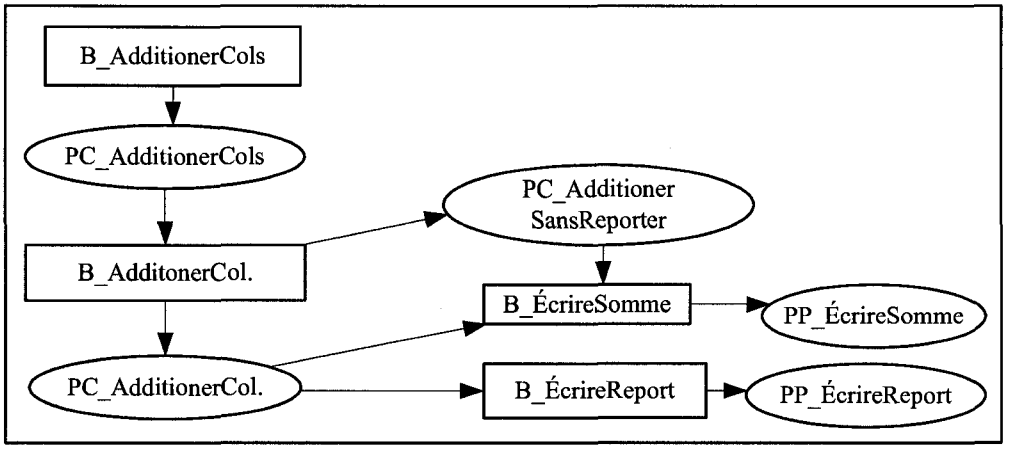

Figure 25 - Les connaissances procédurales pour l'addition en colonnes.

procédure alternative «additionner sans reporter» est applicable lorsque la colonne est «sans report». Cette procédure n'a que le sous-but «écrire somme», autrement dit, lorsque le report est zéro, il n'est pas nécessaire de l'écrire (voir figure 23). Les procédures primitives «écrire somme » et «écrire report» modifient toutes deux la colonne qu'elles reçoivent en paramètre, mais en plus, la deuxième modifie également la colonne suivante (que l'on retrouve grâce à la relation «colonne suivante»). En effet, le report d'une colonne et la retenue de la colonne suivante (si elle existe) réfèrent toujours au même chiffre. Dans le cas de la «dernière colonne» seul le report de la dernière colonne est modifié par la procédure primitive.

3.1.3 Conclusion sur l'addition en colonnes

Bien qu'il s'agit du laboratoire le plus simple que nous avons modélisé, l'*addition en colonnes* a permis de tester et de montrer dans leur plus simple expression la plupart des originalités de l'architecture de connaissance (sauf la gestion des *entrées*). En effet, il illustre le fait que : 1) les connaissances procédurales hiérarchiques permettent à l'agent pédagogue de connaître facilement les prochaines étapes que l'apprenant doit faire; 2) les fonctions/relations associées à des règles d'instanciation représentent élégamment les inférences; 3) la classification permet d'établir des critères d'applicabilité pour les procédures; 4) le rôle des instances statiques dans l'ontologie; 5) la distinction entre les attributs et les fonctions/relations.

3.2 L'addition de fractions

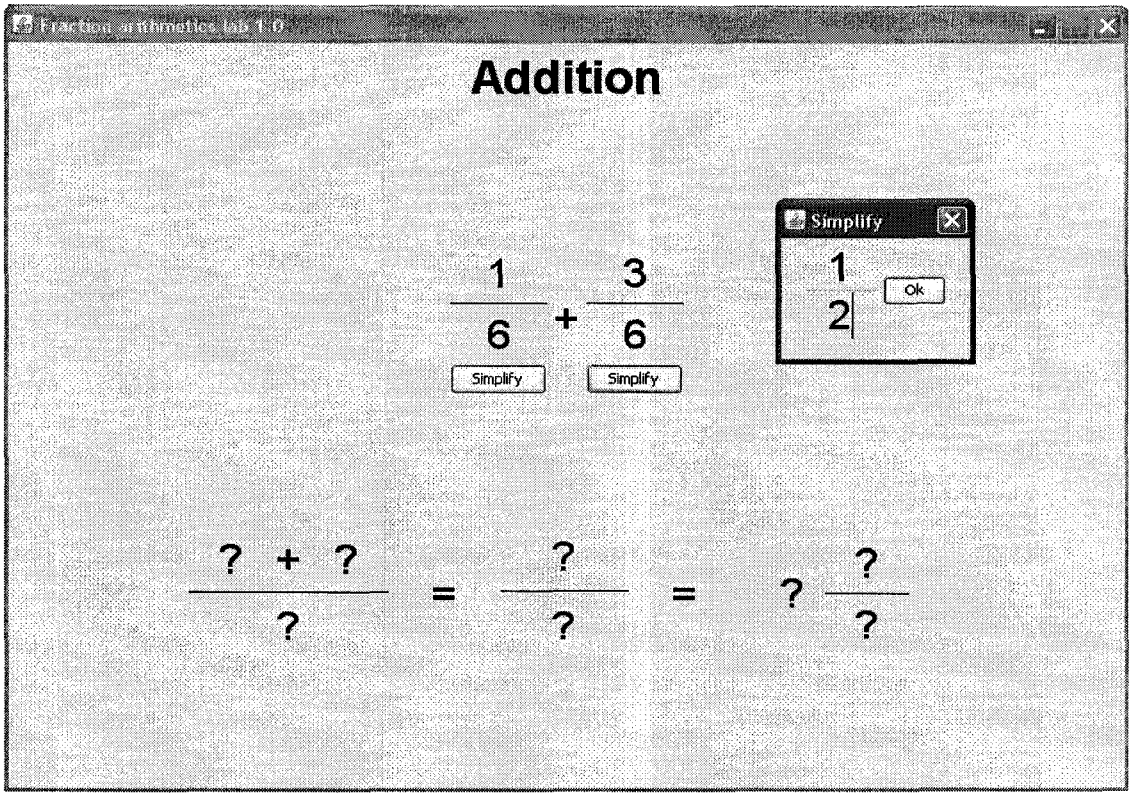

Figure 26 - Une capture d'écran du laboratoire pour l'addition de fractions.

Le laboratoire pour l'addition de fractions (voir figure 26) a été développé premièrement pour valider le fonctionnement de l'association entre les instances et leur représentation graphique dans l'interface du laboratoire grâce à l'agent interface (voir chapitre 4). Comme l'addition en colonnes, l'addition de fractions est un domaine de nature *procédurale* au sens de VanLehn (voir chapitre 1). De ce sens, l'interface du laboratoire supporte une stratégie de résolution particulière où l'apprenant doit remplir, dans le bon ordre, trois formulaires qui reflètent la technique classique pour additionner des fractions. Comme dans tous les domaines présentés dans ce chapitre, nous considérons que les notions théoriques ont été présentées aux apprenants et que ceux-ci ont une idée générale de la marche à suivre. Par exemple, dans le cas de l'addition des fractions, nous supposons que l'enseignant a précisé que les formulaires doivent être remplis dans un ordre particulier. Nous supposons aussi qu'une démonstration

précède les activités de résolution de problèmes (voir chapitre 4), ce qui assure aux apprenants une familiarité minimale avec l'interface. Par rapport à l'addition en colonnes, ce laboratoire permet de valider notre approche pour traiter les saisies de l'apprenant à l'aide des instances de type *entrée* (voir chapitre 2).

3.2.1 Les connaissances sémantiques

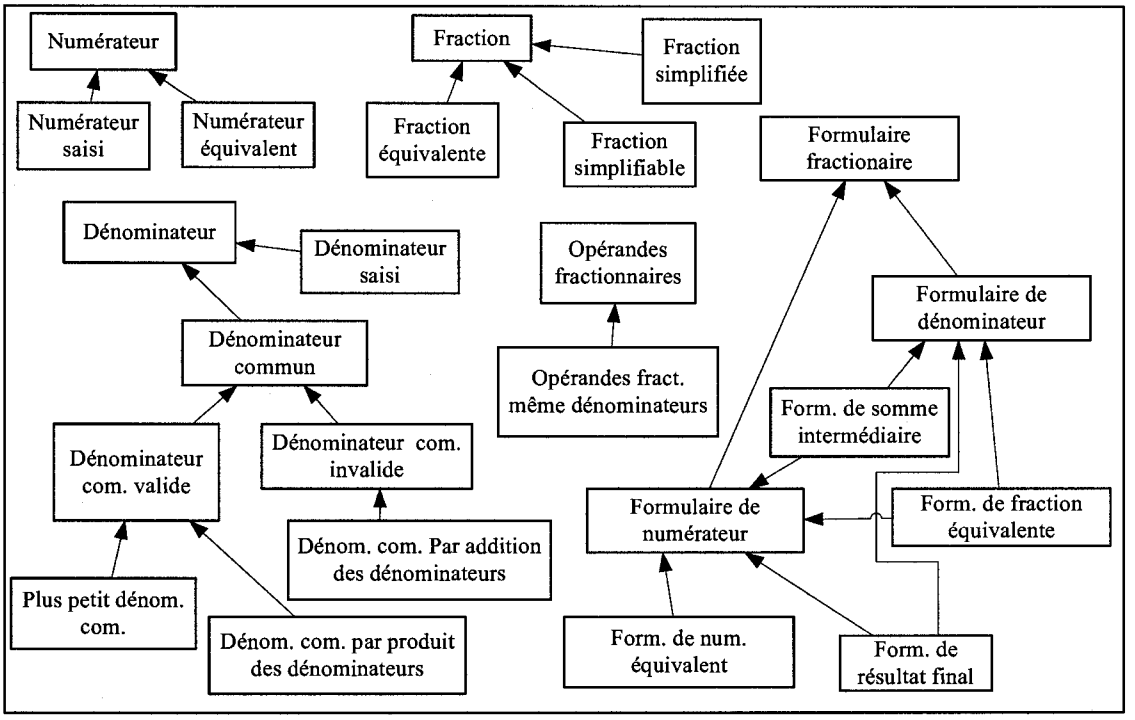

Figure 27 - La taxonomie des concepts dans le modèle pour l'addition de fractions.

Le nombre beaucoup plus grand de concepts (voir figure 27) nécessaires à la modélisation de ce domaine par rapport à l'addition en colonnes s'explique par le fait que la plupart des actions effectuées par l'apprenant correspondent à la saisie de nombres naturels. Aussi, puisque tout le processus menant au choix d'un dénominateur commun est représenté uniquement à l'aide de fonctions (voir figure 29), c'est-à-dire qu'il n'y a pas dans l'interface d'outils permettant d'effectuer les calculs intermédiaires, on obtient une complexité semblable dans les connaissances sémantiques et dans les connaissances procédurales. Le modèle de connaissance pour ce laboratoire ne compte qu'un seul contexte qui est nommé « Contexte

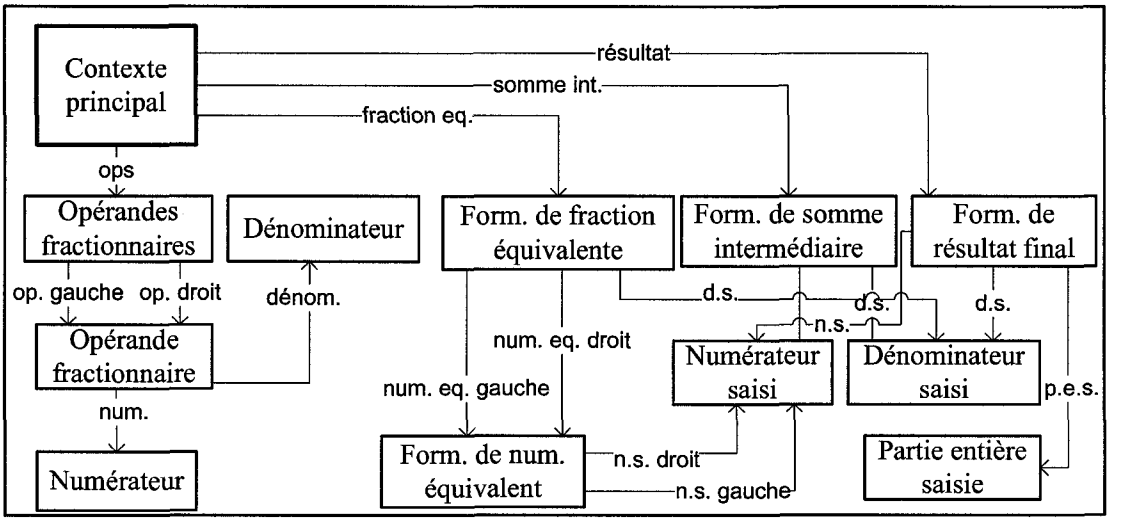

Figure 28 - Le contexte principale du laboratoire pour l'addition de fractions.

principal». Ce contexte a quatre points d'ancrage : «opérandes fractionnaires» (la donnée du problème), «formulaire de fraction équivalente», «formulaire de somme intermédiaire», «formulaire de résultat final» (voir figure 28). Les trois derniers sont les formulaires pour saisir la solution. Lorsque l'instance de ce contexte est créée, quatre instances doivent donc être créées par le script d'initialisation de ce dernier. Pour construire ces instances, d'autres instances qui les constituent devront être aussi créées. Par exemple, deux instances d'«opérande fractionnaire» comportant chacune une instance de «numérateur» et de «dénominateur» devront être créées pour initialiser les liens «op. gauche» et «op. droit» de l'instance d'«opérandes fractionnaires». Les concepts représentant les formulaires et les saisies sont distingués des concepts représentant le résultat des calculs mentaux. Par exemple, on retrouve un «Numérateur saisi» à la fois dans le «formulaire de somme intermédiaire» et dans le «formulaire de résultat final», par contre le premier est tiré de la «fraction équivalente» tandis que le deuxième est tiré de la fraction finale».

Les fonctions de ce laboratoire sont toutes associées à une règle d'instanciation. C'est dans ces règles que l'on retrouve les calculs mathématiques nécessaires pour faire les différentes étapes de l'addition (voir figure 29). Par exemple, la fonction « obtenir numérateur équivalent » s'instancie une fois pour chacune des «opérande fractionnaire» dès que le deuxième argument,

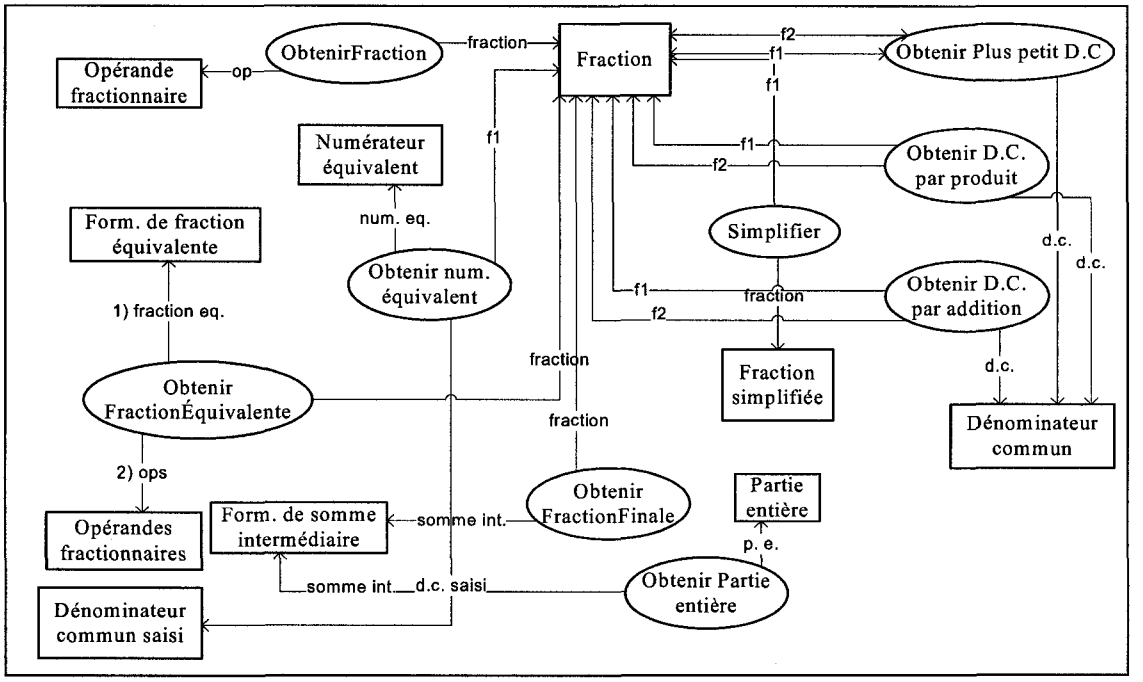

Figure 29 - Les fonctions du modèle de connaissance pour l'addition de fractions.

l'instance de « dénominateur commun saisi » remplace la valeur inconnue dans le «formulaire de fraction équivalente». En fait, c'est pour obtenir la valeur de son image, une instance de «numérateur équivalent», qu'un calcul est effectué. Il faut remarquer que ni les «opérande fractionnaire» ni les formulaires ne *sont* des «fraction», en effet, pour chacun de ces concepts, on trouve une fonction dont l'image est la fraction correspondante.

3.2.2 Les connaissances procédurales

Le but «additionner fractions » fait une requête pour obtenir l'unique instance du concept «opérandes fractionnaires». Les procédures complexes «additionner fractions» et «additionmer fractions avec le même dénominateur» sont des séquences totalement ordonnées (voir figure 30). qui peuvent satisfaire ce but. La deuxième n'est disponible que si la règle de classification a reconnu des «opérandes fractionnaires au même dénominateur». Cette procédure permet d'éviter les deux premiers sous-buts qui servent à établir la fraction équivalente et de passer directement à la somme intermédiaire. Le premier sous-but de l'addition complété, «simplifier opérandes» est satisfait par la procédure complexe de type

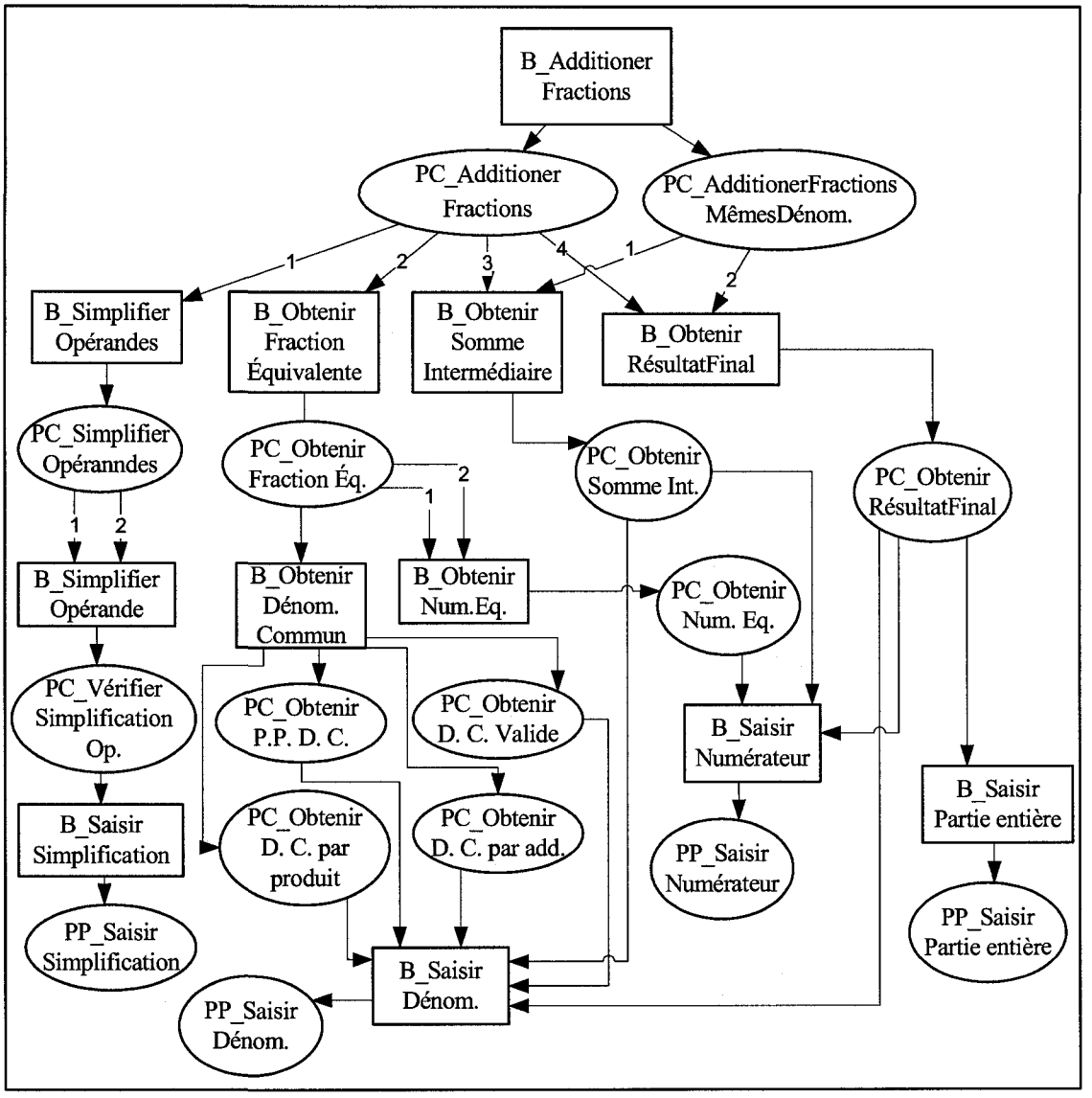

Figure 30 - Les connaissances procédurales pour l'addition de fractions.

séquence «simplifier opérandes» qui a un sous-but pour chacune des «opérande fractionnaire» qu'elle obtient par requête sur son unique paramètre («opérandes fractionnaires»). La procédure complexe de type conditionnelle «vérifier simplication de l'opérande» a une condition qui ajoute le sous-but «saisir simplication» si la fraction déduite de l'«opérande fractionnaire» n'est pas une «fraction simplifiée». La procédure primitive «saisir simplification» peut être faite dans un dialogue qui s'affiche une fois que l'apprenant appuie

sur le bouton «Simplify» qui se trouve sous l'opérande (voir figure 26). Le sous-but «obtenir fraction équivalente» fait une requête pour obtenir l'instance du «formulaire de la fraction équivalente» qu'il passe à la procédure complexe « obtenir fraction équivalente » qui est une séquence ayant comme premier sous-but « obtenir dénominateur commun » puis le sous-but « obtenir numérateur équivalent » pour chacun des deux «opérande fractionnaire». Ces sous-buts peuvent être satisfaits dans un ordre quelconque. Plusieurs procédures complexes (de type séquence à un seul sous-but) peuvent satisfaire le but «obtenir dénominateur commun», chacune d'entre elles fait une requête afin d'obtenir une instance de «dénominateur commun ». Par exemple, la procédure complexe «obtenir le plus petit démoninateur commun» et la procédure « obtenir démoninateur commun par addition des dénominateurs» qui est une procédure complexe invalide qui obtient une instance de la fonction «obtenir dénominateur commun par addition des dénominateurs». Cette dernière représente l'addition des dénominateurs des deux «opérande fractionnaire». Toutes ces procédures ont pour seul sous-but, «saisir dénominateur» qui reçoit en paramètre le «dénominateur commun» obtenu et un «formulaire de dénominateur». La procédure primitive «saisir dénominateur» modifie l'instance du «formulaire de dénominateur» de façon à refléter la saisie effectuée par l'apprenant, c'est à dire en liant cette dernière instance à une nouvelle instance de «dénominateur saisi» qui contient elle-même la *valeur* provenant de l'instance de «dénominateur commun».

L'ensemble de ces procédures est un cas représentatif des ambiguïtés décrites au chapitre 2. En effet, il est fort possible que pour un problème donné, le «plus petit dénominateur commun» corresponde au «dénominateur commun obtenu en faisant le produit des dénominateurs». La procédure complexe «obtenir dénominateur valide» permet de reconnaître tous les autres dénominateurs valides. Cette dernière a la plus faible priorité puisqu'on préfère créditer à l'apprenant une des procédures plus précises.

Les procédures complexes «obtenir numérateur équivalent », «obtenir somme intermédiaire» et «obtenir résultat final» sont des séquences à un seul sous-but construites sur le même modèle, c'est-à-dire qu'elles obtiennent une instance de fonction à partir d'une requête

qu'elles passent ensuite à leur sous-but respectif qui représente la saisie du résultat du calcul mental correspondant. Les trois procédures primitives qui satisfont ces sous-buts sont toutes similaires, elles inscrivent dans une instance de formulaire une valeur provenant d'une instance représentant le résultat d'une opération mentale.

Contrairement au cas du «dénominateur commun», pour les «numérateur équivalent», un comparateur est suffisant (il n'y qu'une seule valeur possible). Pour les numérateurs et dénominateurs des deux derniers formulaires, il en est de même. Cependant, une modélisation complète comporterait des procédures complexes alternatives pour modéliser les simplifications possibles, par exemple pour simplifier la fraction déduite du «formulaire de la somme intermédiaire» si elle est une «fraction simplifiable».

3.2.3 Conclusion sur la modélisation de l'addition de fractions

La modélisation du «formulaire pour la fraction équivalente» est particulièrement intéressante, puisque qu'elle montre que l'architecture des connaissances force l'auteur du modèle à rendre explicite tous les liens significatifs. En effet, chacun des « formulaire de numérateur équivalent» doit avoir un lien vers la fraction auquel ils correspondent. Bien qu'il semble intuitif de les associer, il faut rendre ce lien formel afin que l'on puisse par exemple obtenir à l'aide d'une requête, le « numérateur équivalent » d'une fraction. Bien qu'il aurait été possible de modéliser le domaine en utilisant qu'une procédure primitive, il aurait été plus difficile dans ce cas de produire des messages d'aide précis.

Ce laboratoire montre comment les règles de classification sont utiles pour évaluer les différentes saisies possibles d'un apprenant. En particulier, ce laboratoire illustre comment modéliser les domaines où l'on retrouve de «l'édition sur place», c'est-à-dire que le résultat d'une saisie est présenté exactement là où cette dernière est faite. Ce laboratoire donne aussi un exemple pertinent de procédure invalide, c'est-à-dire une erreur qui ne pourrait pas être traitée de façon précise par l'agent pédagogue. Finalement, bien que nous aurions pu modéliser la saisie des simplifications à l'aide d'un contexte, nous voulions montrer qu'il était possible de l'éviter dans certains cas.

3.3 La machine

Ce domaine créé de toutes pièces est constitué de deux parties différentes : la première pour les ajustements sur une machine «quelconque», puis la seconde pour la réalisation d'un rapport de l'entretien effectué. Un laboratoire simulant cette machine est implémenté à l'aide de trois types de boutons (voir figure 31). Les ajustements consistent à : 1) passer en revue un ensemble d'interrupteurs possédant deux états (allumé/éteint) qui doivent tous être allumés; appuyer cinq fois sur le bouton A; 3) appuyer sur le bouton B jusqu'à ce qu'un témoin lumineux apparaisse. L'interface pour effectuer le rapport a trois sections : a) entrer le nombre de fois où le bouton B a dû être appuyé avant l'apparition du témoin; b) préciser de quelle couleur était ce dernier (rouge, jaune ou vert); c) indiquer quels sont les interrupteurs manipulés, avec leur état courant (voir figure 31).

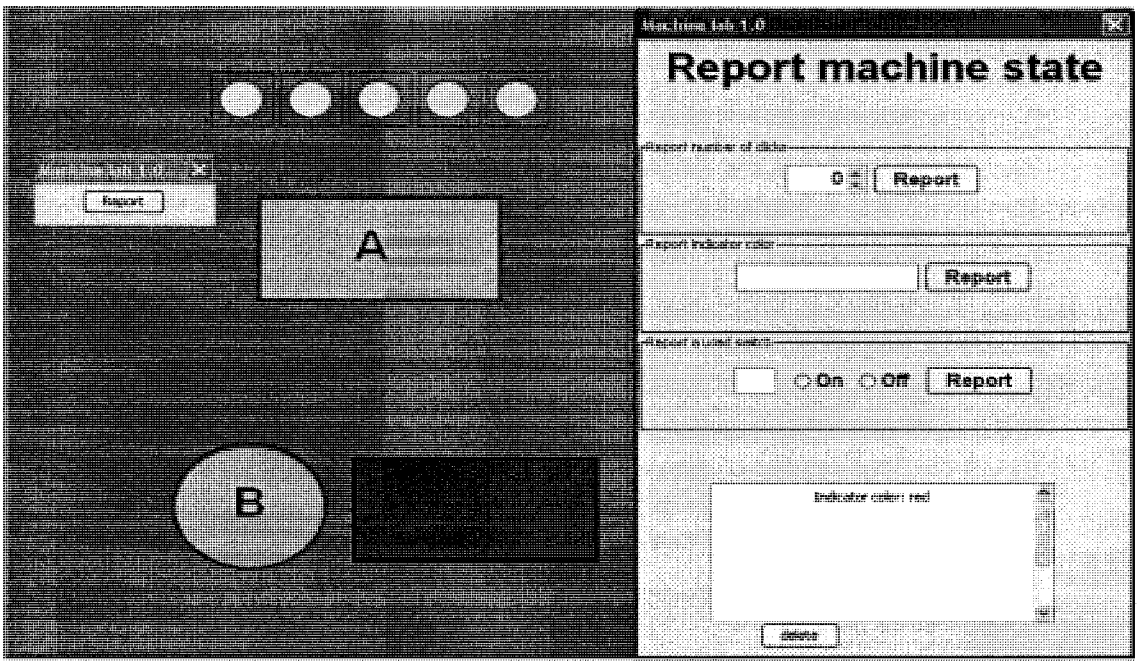

Figure 31 - Une capture d'écran du laboratoire machine (avec le contexte du rapport).

Ce laboratoire permet de valider deux éléments importants de l'architecture des connaissances, d'abord les changements de contexte, puis l'utilisation d'un noyau de simulation. De plus, ce laboratoire a permis de valider certaines idées au niveau de l'agent

interface, en particulier, les mécanismes pour associer à une procédure primitive plusieurs manipulations de l'interface, par exemple la séquence : 1) écrire le numéro de l'interrupteur dans la boîte de texte; 2) indiquer son état avec le bouton radio approprié et 3) cliquer sur le bouton « Report »[1].

3.3.1 Connaissances sémantiques.

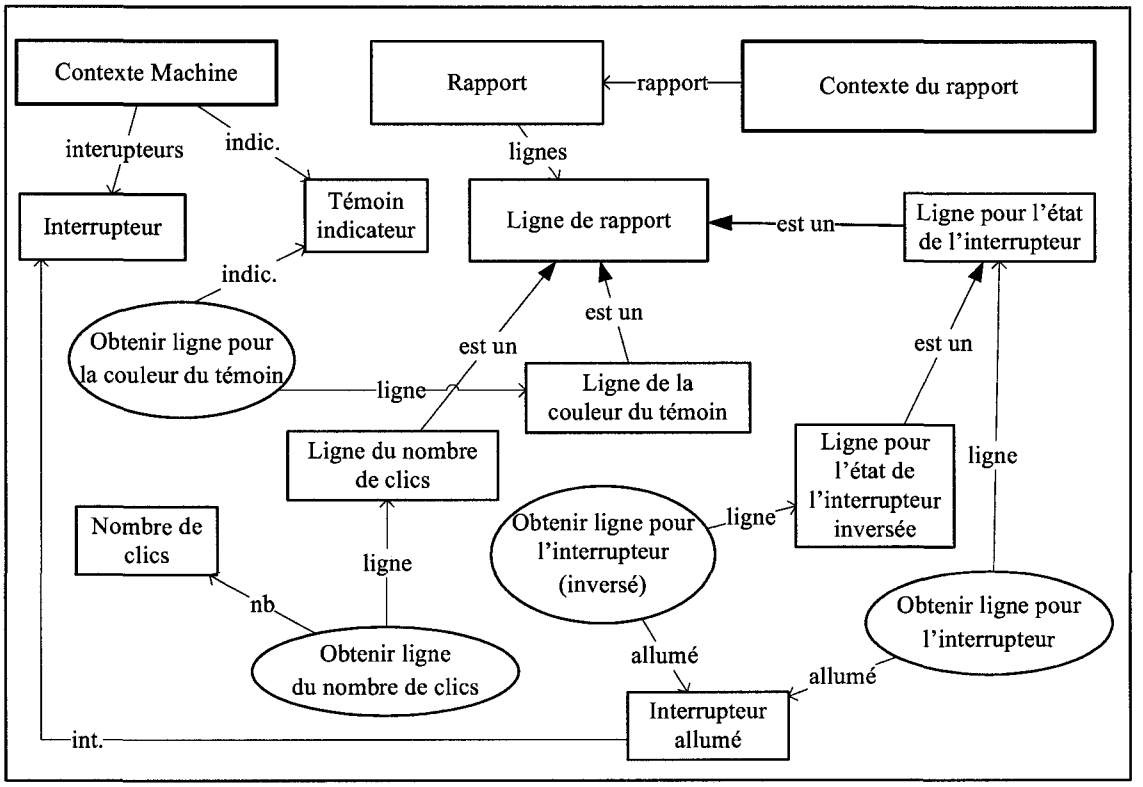

Figure 32 - L'ontologie du laboratoire machine.

Contrairement aux laboratoires précédents, on retrouve dans celui-ci deux contextes différents. Une instance du premier contexte, «machine» est créée au départ, puisque c'est la première fenêtre présentée à l'apprenant. C'est le bouton «Report», qui ne fait partie d'aucune procédure primitive, qui déclenche la création d'une instance du contexte «Contexte de rapport» et qui fait la transition du contexte machine vers ce dernier. Le contexte «machine» a

1.À ce moment, il n'était pas encore possible de simplifier les opérandes dans l'addition de fractions.

deux points d'ancrage qui lient à ce dernier l'ensemble des interrupteurs et le témoin indicateur. Le «Contexte de rapport» en a un seul, le «Rapport» qui contient un ensemble de «Ligne de rapport» représentant chacune des entrées dans celui-ci (voir figure 32).

Un très simple noyau de simulation est utilisé dans ce laboratoire, en effet le nombre de clics nécessaires sur le bouton B est un élément opérationnel du laboratoire qui ne fait pas partie des connaissances (tant qu'il n'est pas connu).

Bien que l'affichage de la fenêtre dédiée au contexte du rapport ne cache pas la fenêtre du contexte de la machine, nous avons dû avoir recours à des concepts qui modélisent des éléments que l'apprenant doit se rappeler. Il s'agit du «nombre de clics» et des «interrupteurs allumés». En effet, s'il est possible de trouver la couleur du témoin lorsque le rapport est ouvert, le «nombre de clics» et l'ensemble des «interrupteurs allumé» ne sont jamais directement accessibles dans le contexte «machine».

Toutes les lignes sont évaluées à l'aide de comparateurs, en fait il n'y a aucune règle de classification dans ce laboratoire dont le mandat était plutôt d'explorer les différentes procédures complexes de type itération.

3.3.2 Les connaissances procédurales

Les procédures complexes «faire l'entretien de la machine» et «faire les ajustements» sont de types séquence (voir figure 33). La procédure «vérifier les interrupteurs» est une itération de type pour-chaque non ordonnée. La procédure complexe «vérifier un interrupteur» est une procédure complexe de type conditionnel dont la condition vérifie l'état de l'interrupteur. Si ce dernier est éteint, le sous-but « allumer un interrupteur » est retenu. La procédure «Opérer le bouton A» est une procédure complexe d'itération de type répétition (5 fois). La procédure «Opérer le bouton B jusqu'à l'apparition de couleur » est une procédure d'itération de type tant-que et sa condition vérifie l'attribut couleur de l'instance de «Témoin indicateur». La procédure primitive «appuyer sur le bouton B» fait appel au noyau pour mettre à jour le compteur interne et vérifier si la couleur doit être affichée.

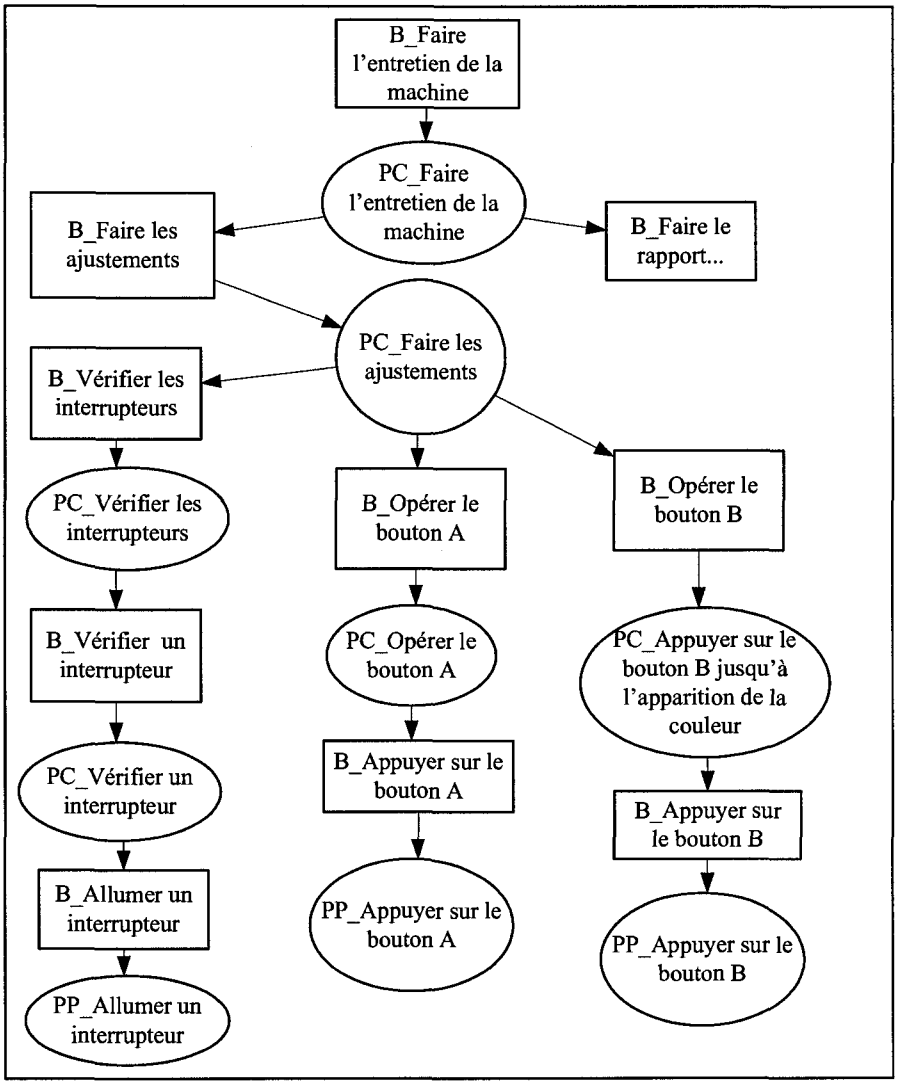

Figure 33 - Les connaissances procédurales pour la machine.

La procédure complexe «rapporter le nombre de clics» (voir figure 34) fais une requête pour obtenir l'instance de «nombre de clics» qui a été créée dans le script d'action de la procédure primitive «appuyer sur le bouton B» lors de sa dernière exécution qui a déclenché l'apparition d'une couleur sur le témoin indicateur. De même, la procédure complexe «rapporter l'état des interrupteurs» est une itération du type pour-chaque utilisant aussi le rappel (l'ensemble des «interrupteur allumé»). Finalement, la procédure «rapporter la couleur du témoin» est une simple séquence à un sous-but qui fait une requête pour obtenir l'instance de la fonction

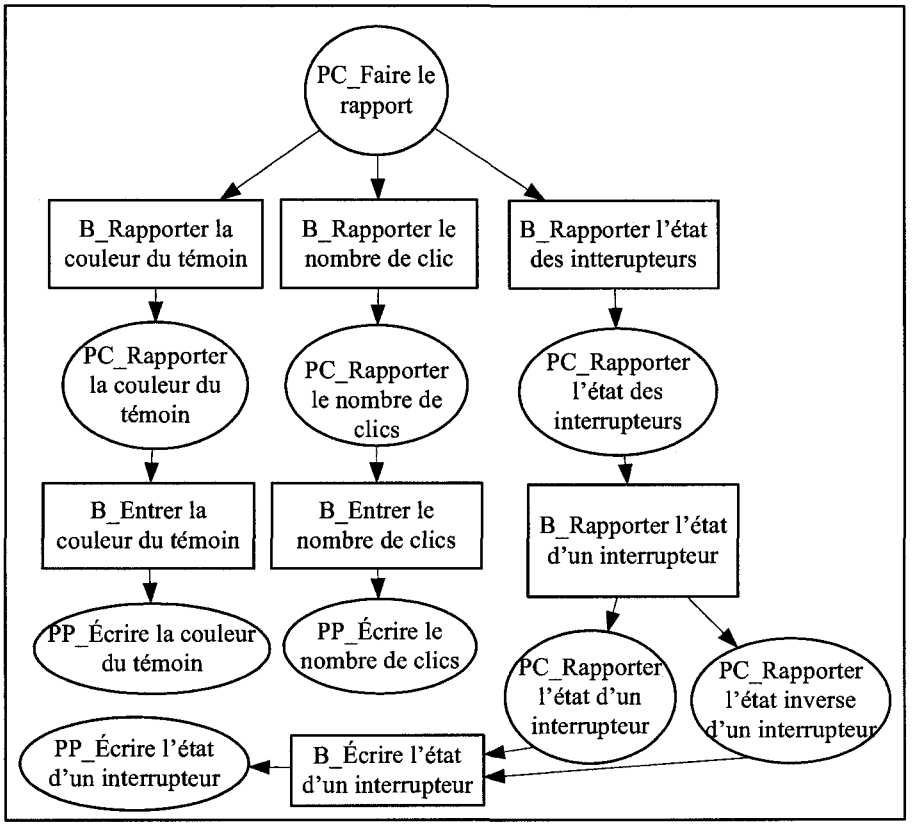

Figure 34 - Les connaissances procédurales pour le rapport.

«obtenir la ligne pour la couleur du témoin». La règle d'instanciation de cette fonction obtient la couleur du «témoin indicateur» même si celle-ci se trouve dans le contexte «machine». Celui-ci étant toujours visible, il n'est pas pertinent de modéliser un rappel.

3.3.3 Conclusion sur la modélisation de la machine

Si on ajoute une procédure invalide «Appuyer une seule fois sur le bouton B» on obtient une séquence ayant le même sous-but qu'une itération. Ceci peut devenir un des cas d'ambiguïté présentés au chapitre 2 lorsque le noyau est configuré pour qu'un seul clic soit nécessaire pour faire apparaître la couleur du témoin lumineux. Ce troisième et dernier laboratoire a permis d'illustrer deux autres originalités de l'architecture de connaissance, les changements de contexte et le noyau de simulation du laboratoire. De plus, il a montré une façon de faire pour

représenter des éléments qui doivent être rappelés par la suite, une alternative parfois intéressante aux requêtes sur la mémoire épisodique (voir chapitre 2).

3.4 Discussion

À l'aide d'exemples issus des modèles de connaissance présentés dans ce chapitre, nous montrons dans cette section que l'architecture de connaissance a bel et bien été conçue et implémentée en suivant des lignes directrices établies au chapitre 2.

3.4.1 La sémantique des unités de connaissance est précise

En ajoutant l'environnement, c'est-à-dire le niveau des instances entre celui des concepts et celui des cognitions, la sémantique des modèles de connaissance est devenue beaucoup plus claire que dans les modèles conçus pour être interprétés par le MGC. De plus, la notion de contexte a permis de préciser encore davantage la nature des instances, en particulier, distinguer celles qui se trouvent dans l'ontologie du domaine, celles qui sont représentées dans l'interface du laboratoire et celles qui sont le produit d'opérations mentales. Cette distinction n'est pas présente dans les MTT et peut rendre floue la sémantique d'un WME. Dans le laboratoire de l'addition de fractions, l'exemple du «dénominateur commun», un résultat mental par rapport au «démoninateur commun saisi» qui s'inscrit dans un «formulaire de dénominateur» illustre bien ces nuances.

D'autres efforts sont aussi allés dans ce sens, que ce soit la construction des procédures complexes en fonction des structures de contrôles élémentaires, le fait de nommer explicitement tous les attributs existants entre les concepts ou bien le resserrement des actions possibles dans les scripts d'actions.

En somme, les efforts de clarté se résument par la distinction que nous pouvons faire entre les différents types de connaissance procédurales. Par exemple, dans le laboratoire de l'addition en colonnes, on distingue la procédure complexe «additioner colonne» de la requête à la fonction «somme» qu'elle exécute et de la procédure primitive «entrer somme» qui satisfait

son unique sous-but. En effet, nous distinguons le plan mental de haut niveau (qui s'inscrit dans un plan plus général pour résoudre le problème), l'opération arithmétique mentale et l'action atomique qui est associée à la saisie du résultat de celle-ci dans l'interface du laboratoire.

3.4.2 Il y a des unités boîtes noires et boîtes de verre

L'ajout des relations et des fonctions au sein des structures et l'ajout des mécanismes de requête ont grandement contribué à permettre une modélisation qui s'adapte aux objectifs pédagogiques. En effet, pour un même but, on peut offrir une procédure complexe qui a recours à une requête pour obtenir un résultat et une autre qui fait appel à un sous-but qui sera satisfait par une procédure représentant une méthode pour l'obtenir. Par exemple, dans le laboratoire des fractions, on pourrait imaginer un contexte où serait réifiée une procédure complexe permettant de calculer le plus petit dénominateur commun.

Nous savons qu'il est intéressant que les agents puissent scruter une règle de classification ou d'instanciation. Jusqu'à maintenant, nous considérons que cette barrière est justifiée, parce que nous n'avons pas l'objectif d'enseigner des procédures complexes purement mentales, c'est-à-dire des procédures qui donneraient lieu à des procédures primitives dont les scripts d'actions manipuleraient uniquement des instances du contexte de raisonnement. De plus, nous insistons sur le fait que si la façon de faire pour obtenir le résultat d'une requête est pédagogiquement intéressante, il suffit de modéliser une procédure complexe qui la réifie.

3.4.3 Il est facile d'examiner les unités de connaissances

Face à CTAT, qui n'offre que des exemples d'une envergure semblable (entre autres, l'addition en colonnes et l'addition de fractions) la comparaison est avantageuse pour notre approche. Pour un effort de modélisation que nous jugeons semblable, il est clair que le modèle obtenu a un net potentiel supérieur pour être examiné par un agent pédagogue afin d'orienter les stratégies pédagogiques dont il dispose. Par exemple, il serait bien plus aisé pour cet agent de décrire à l'apprenant une procédure complexe d'une certaine complexité, comme «obtenir

fraction équivalente» dans le laboratoire de fractions (plusieurs requêtes et une séquence de sous-but partiellement ordonnée) que de tenter de faire de même en examinant un groupe de règles de production liées par chaînage dans lequel les calculs sont directement inscrits sous forme de code.

De plus, de nombreux «pointeurs à contresens» sont présents dans les structures afin de permettre la navigation d'une façon qui est plutôt propre à l'agent pédagogue qu'à l'agent expert. Par exemple, les requêtes qui permettent de parcourir les attributs de type lien à l'envers. Finalement, au chapitre 2, nous avons montré que tous les agents avaient un accès facile aux structures de connaissance grâce à la mémoire de travail partagée.

3.4.4 Bilan

En intégrant dans les structures de connaissance épisodique les processus pour la résolution automatique et le suivi de l'apprenant, l'architecture de connaissance décrite dans ce mémoire va plus loin que celle du MGC. En effet, elle fournit un environnement d'exécution comparable à celui de CTAT. En prenant un virage pragmatique qui nous a amenés à repenser l'architecture théorique d'ASTUS, nous sommes arrivés à modéliser plusieurs domaines simples, mais différents. Ceci a montré que l'architecture de connaissance n'était pas conçue de façon à épouser spécifiquement un domaine ou un autre, ce qui est un avantage indéniable par rapport aux approches des STI propres à un domaine comme Andes. Cependant, seule la modélisation de domaines réels pourra démontrer l'atteinte des objectifs du prototype de la plate-forme ASTUS.

3.5 Conclusion

Ce chapitre, nous permet de dire que l'architecture de connaissance présentée fait plus que celle du MGC dont elle s'inspire, mais surtout, est plus claire sur ce qu'elle tente de faire et ce qu'elle réussit ou non à faire. De plus, elle se compare avantageusement, du moins sur certains points, à l'architecture des MTT qui a déjà fait ses preuves.

Chapitre 4

Le prototype

D'abord, il est important de rappeler que le design du prototype s'inspire essentiellement du découpage classique en quatre modules des STI. De plus, à l'image de l'architecture conceptuelle d'ASTUS, il distingue les composantes génériques (les agents) des composantes dépendantes du domaine (ressources, noyaux, interfaces, etc.). La présentation des composantes du prototype suit approximativement l'ordre dans lequel ils interviennent lors de la résolution d'un problème par un apprenant.

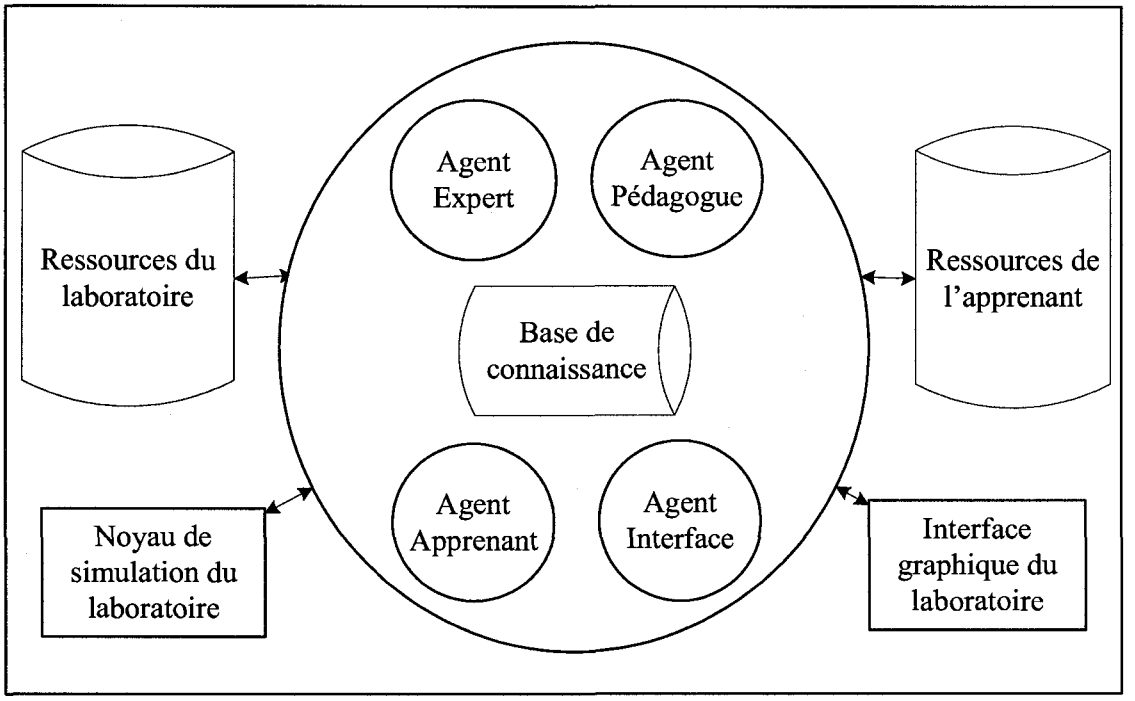

Figure 35 - Un aperçu complet du prototype.

Bien qu'ils soient désignés par le terme agent, les composantes génériques d'ASTUS ne se comportent pas réellement comme tel. En effet, bien qu'individuellement ils constituent des

processus intelligents, le comportement de l'ensemble est complètement déterministe, il s'agit essentiellement des boucles externes et internes décrites en introduction.

4.1 Chargement des ressources

L'initialisation du prototype débute par le chargement du **répertoire des apprenants**, du **répertoire des laboratoires** et du **registre des inscriptions**. Le répertoire des apprenants contient l'ensemble des matricules et des mots de passe qui permettent d'identifier ceux-ci. Le répertoire des laboratoires contient simplement une liste des laboratoires disponibles. Le registre des inscriptions spécifie quels apprenants ont accès à quels laboratoires. Par la suite, l'utilisateur doit s'identifier et choisir un laboratoire à l'aide d'une boîte de dialogue (voir figure 37 dans l'annexe A).

Ensuite, le prototype procède au chargement des ressources du laboratoire et de l'apprenant sélectionnés. La principale ressource associée à chaque laboratoire est évidemment **le modèle de connaissance** qui a été décrit dans les chapitres précédents. Bien que la notion de problème à résoudre ait été touchée précédemment, cete dernière n'a pas été définie formellement. Pour chaque laboratoire, un ensemble de **problèmes** est offert. Pour chacun d'entre eux, on trouve : un nom, un fichier de données (optionnel), le but qui correspond à sa résolution (un même laboratoire peut comporter plusieurs types de problèmes) et d'autres métadonnées comme la durée et la difficulté estimée par l'auteur.

Les problèmes sont des données brutes et s'inscrivent toujours au sein d'une **activité**. Le prototype offre trois types d'activité :

1) **la résolution de problème** qui correspond en fait au déroulement normal de la boucle interne qui a été décrit jusqu'ici.
2) **la démonstration** qui permet à l'apprenant de suivre, étape par étape, la résolution d'un problème qui lui est présenté.
3) **le test** dans lequel l'apprenant est laissé à lui-même face au problème, puisqu'aucun suivi n'est effectué.

Pour utiliser le mode démonstration, il faut d'abord avoir une unité de mémoire épisodique qui résulte d'une résolution de problème complétée. L'auteur de la démonstration doit utiliser la résolution automatique de l'agent expert pour créer cette unité. Pour le mode test, l'enseignant peut spécifier un temps maximum alloué à l'apprenant et bloquer le mécanisme d'interruption/ reprise qui est offert pour la résolution de problème et la démonstration. Il n'y a pas à proprement parler d'outils pour évaluer un test complété, l'évaluation doit être faite soit en observant l'état final de l'environnement, soit en consultant le journal des actions (voir section 4.4).

Le prototype implémente trois variantes de la boucle externe. Cette dernière propose à l'apprenant une séquence d'activité. Le choix de la variante s'effectue avant l'initialisation de l'agent pédagogue, au moment où le prototype identifie dans le fichier de configuration ou à l'aide d'une boîte de dialogue (voir figure 38 dans l'annexe A) le comportement à adopter. Trois options sont disponibles :

1) l'apprenant choisit lui-même l'activité dans une liste;
2) l'activité est choisie dans un agenda préparé par l'auteur;
3) l'agent pédagogue choisit l'activité en fonction d'un curriculum.

On nomme **curriculum** l'ensemble des **objectifs pédagogiques** que le laboratoire permet d'atteindre. Les différentes théories de l'enseignement conçoivent les objectifs pédagogiques différemment, cependant l'implémentation retenue est davantage influencée par l'approche béhavioriste. D'abord, il y a deux niveaux d'objectifs, les objectifs généraux et les objectifs spécifiques. Les **objectifs généraux** sont décrits à l'aide d'un nom, d'une courte description et d'un ensemble de sous objectifs, pouvant être eux aussi généraux ou spécifiques. Chacun de ces sous objectifs influence l'atteinte de l'objectif avec un certain poids.

Mager [41] définit les objectifs à l'aide de trois composantes : 1) le comportement observé; 2) les conditions dans lesquelles il est réalisé; 3) un seuil de réussite. Les **objectifs spécifiques** s'en inspirent, en définissant le comportement observé comme étant la capacité de l'apprenant à satisfaire un but particulier. L'auteur du curriculum peut appliquer certaines **contraintes** sur la façon dont le but doit être atteint : préciser quelle procédure doit être utilisée; préciser

davantage un argument au niveau du but ou d'une procédure; définir un niveau maximum d'aide offerte par le pédagogue, etc. L'auteur doit aussi spécifier les conditions d'atteinte de l'objectif, c'est-à-dire le nombre de fois où le but doit être atteint en respectant les conditions. Lors de travaux futurs, nous tenterons de développer un algorithme permettant d'établir un seuil d'atteinte approximatif qui tient compte des structures de représentations de connaissance.

L'**agenda** est une séquence d'activités définie par l'auteur qu'il juge être suffisante pour satisfaire l'ensemble des objectifs présents dans le «curriculum» que celui-ci soit défini formellement ou implicitement (à partir des activités choisies). L'agenda peut être souple, puisque l'auteur peut spécifier que certaines activités sont optionnelles.

L'**historique** de l'apprenant est l'ensemble des activités auquel il a participé. Pour chacune d'entre elles, on conserve le temps qui y a été consacré et un statut. En plus de l'activité «en cours», on conserve également tant les activités «terminées» que celles qui ont été «abandonnées». Finalement, les activités «interrompues» peuvent être reprises à un moment ultérieur. Une unité de mémoire épisodique supplémentaire est produite dans ce dernier cas.

Pour chaque apprenant, on dispose d'un **profil** contenant son matricule, son nom, sa langue usuelle ou toute autre information susceptible de modifier le comportement de l'agent pédagogue. Ensuite, les données les plus importantes concernant l'apprenant sont chargées, il s'agit de la trace exhaustive des actions qu'il a produites et de sa mémoire épisodique. **Le journal des actions** se situe à un niveau inférieur, car il est complètement indépendant du processus de suivi de l'apprenant. Il est constitué de l'ensemble des actions détectées par l'agent interface, que ce soit les procédures primitives ou les transitions de contexte. Il est possible d'associer les éléments de la mémoire épisodique et ceux du journal des actions par un identificateur unique qui est commun aux deux (c'est le résultat stocké dans la mémoire épisodique pour les procédures primitives). Le journal des actions est essentiel pour reproduire la solution de l'apprenant lors de la reprise d'une activité interrompue ou lors d'un test pour en faire l'évaluation. En effet, c'est là que sont conservées les données pour reconstituer les instances entrées par l'apprenant (voir chapitre 2).

Le **journal des rétroactions** conserve toutes les actions pédagogiques effectuées en fonction de la structure des connaissances épisodiques. Ceci permet entre autres de facilement mesurer la quantité d'aide qui a été accordée par l'agent pédagogue pour chaque épisode.

Des structures très simples ont été développées pour ajouter des **connaissances didactiques** au modèle des connaissances. Essentiellement, il s'agit de l'approche classique où une chaîne de caractère contient des variables qui sont remplacées par des valeurs au moment de produire la rétroaction. Comme le proposait le modèle d'ASTUS, on peut définir de telles connaissances sur les procédures, mais aussi sur les buts, les concepts, etc. Les travaux futurs tenteront de réduire la dépendance de l'agent pédagogue vis-à-vis de telles connaissances, en générant des messages simples, mais qui découlent des structures de représentation de connaissance.

Finalement, le chargement des ressources se termine par **l'historique et la mémoire épisodique de l'expert** qui ont été obtenus par des résolutions automatiques.

4.2 Boucle externe et boucle interne

Après le chargement des ressources, la boucle externe s'amorce :

1) L'agent pédagogue déclenche le choix de l'activité.
2) L'agent interface prépare l'environnement et l'interface du laboratoire.
3) L'agent expert démarre le processus de suivi (création de l'épisode courant).
4) L'agent pédagogue décide s'il doit intervenir avant la première action de l'apprenant.
5) L'agent interface affiche l'interface du laboratoire et la fenêtre du pédagogue.
6) Une fois que l'apprenant effectue une première action, la boucle interne s'amorce :
   6.1) L'agent interface signale l'action à l'agent expert et bloque l'interface.

   6.2) L'agent expert détermine si l'action était prévue par le processus de suivi.

   6.3) Si c'est le cas : le suivi est mis à jour.
   Sinon, l'action est ajoutée à la séquence des actions non prévues.

6.4) L'agent apprenant met à jour son modèle de l'apprenant.

6.5) L'agent pédagogue décide s'il fournit une rétraction.

6.6) L'agent interface produit les effets de la rétroaction et débloque l'interface.

Pendant l'exécution de la boucle interne, l'interface du laboratoire est bloquée, il y a donc un délai minimal qui est imposé entre chaque action effectuée par l'apprenant. Au moment d'écrire le mémoire, nous jugeons difficile d'évaluer l'ampleur de ce problème, puisque la modélisation de l'apprenant n'est pas implémentée et que la complexité du processus de décision de l'agent pédagogue est minimale. De plus, le processus de suivi bénéficierait certainement de plusieurs optimisations. Des travaux futurs sont nécessaires pour élaborer les contraintes qui feront en sorte que le prototype puisse garantir un temps de réaction qui ne dérange pas l'apprenant. Le déroulement normal de ces boucles est perturbé si l'apprenant tarde à faire une action. Après un premier délai, l'agent interface bloque l'interface du laboratoire pour permettre à l'agent pédagogue d'agir. Si l'apprenant ne réagit pas face à cette action pédagogique, l'agent interface considère que l'apprenant fait une pause et affiche une boîte de dialogue (voir figure 44 dans l'annexe A) qui permet à ce dernier de signaler son retour. Finalement, dans le cas d'une démonstration ou d'un test, le comportement des agents est beaucoup plus simple que celui ci-dessus, principalement parce que le processus de suivi n'est pas utilisé.

4.3 L'agent expert

L'agent expert introduit au chapitre 2 est l'agent dont l'implémentation est la plus avancé. En fait, elle est suffisante pour atteindre les objectifs qui ont été fixés pour ce travail en tenant compte des limitations qui ont été mentionnées dans les chapitres précédents. En mode de développement, la fenêtre de l'agent expert offre à l'auteur différents outils pour observer le modèle de connaissance et l'épisode courant sous forme graphique lors de la résolution automatique ou du suivi de l'apprenant (voir figures 41, 46-49 dans l'annexe A).

4.4 L'agent interface

L'agent interface est actuellement conçu et implémenté dans le cadre de la maîtrise de Mikaël Fortin, la description qui suit ne reflète pas l'ensemble des défis rencontrés dans ce projet. L'agent interface a deux fonctions principales : 1) reconnaître les actions effectuées par l'apprenant, tel qu'il en a été question dans les chapitres précédents, et 2) afficher le contexte courant de l'environnement. Pour accomplir cette dernière fonction, il doit pouvoir afficher chaque instance qui se trouve dans le contexte visible, y compris l'instance du contexte elle-même. En effet, la vue de l'instance du contexte est généralement responsable de l'organisation graphique générale de la fenêtre (*layout*). Les composants graphiques (étiquettes, panneaux, listes, etc.) servant à représenter une même instance sont regroupés au sein d'une **vue,** un patron utilisé pour afficher toutes les instances issues d'un même concept. Par contre, il n'y a pas nécessairement une vue par concept, puisque certains sont abstraits et d'autres jamais représentés graphiquement. Les composants graphiques (boutons, menus déroulants, boîtes de texte, etc.) qui servent à manipuler une instance doivent s'enregistrer auprès d'un **gestionnaire d'évènement**. Cette approche flexible s'oppose à l'approche plus simple qui consiste à offrir une librairie de composants graphiques spécialisés (comme le fait CTAT). En effet, le développeur de l'interface du laboratoire peut ainsi utiliser ses propres composants graphiques dérivés de la librairie Swing[1], tant que ceux-ci s'enregistrent auprès d'un des gestionnaires offerts dans la librairie de l'agent interface ou bien d'un gestionnaire propre au domaine.

Pour reconnaître les actions qui sont considérées atomiques pour tous les autres agents, l'agent interface les considère décomposables. En effet, pour chaque action, qu'elle soit liée à une procédure primitive ou à une transition de contexte, il existe un **script d'interaction**[2]. Une **interaction** est une action élémentaire dans l'interface, par exemple un clic sur un bouton. Cette décomposition supplémentaire, qui à notre connaissance, n'existe dans aucun STI, permet d'associer à une seule «étape» (au sens de la boucle interne), une séquence de

1.The Swing architecture http://java.sun.com/products/jfc/tsc/articles/architecture/

2.Ne pas les confondre avec les scripts d'actions, i.e. ceux qui agissent sur l'environnement.

manipulations de l'interface. Par exemple, entrer un chiffre dans une boîte de texte puis cliquer sur un bouton. La puissance des scripts n'est pas encore déterminée formellement, par contre, il est clair qu'ils doivent gérer des interactions qui annulent des interactions précédentes ou bien qui vérifient l'état d'un composant graphique. En effet, ce type d'interaction est nécessaire pour la manipulation de menus déroulants ou encore de cases à cocher. En plus de préciser la nature de l'interaction produite, le gestionnaire d'évènement permet de retrouver la ou les instances qui ont été manipulées par l'interaction. En général, c'est l'instance associée à la vue dont un des composants graphiques a été manipulé qui est ainsi retournée. Pour signaler une action effectuée à l'agent expert, l'agent interface doit récupérer toutes les instances qui correspondent aux arguments de cette action à partir du script exécuté. Pour faciliter cette tâche, un niveau supplémentaire, celui des **macro-interactions**, a été ajouté entre les interactions et le script d'interaction. En effet, il est possible d'associer un sous-script aux entrées et aux sélections (voir chapitre 2).

L'agent interface est également responsable de la fenêtre du pédagogue (voir figure 36) qui permet à l'agent pédagogue de communiquer des messages à l'apprenant. Quatre types de message sont disponibles : neutres (noir), avertissements (jaune), erreurs (rouge) et approbation (vert). La fenêtre affiche également le temps écoulé depuis le début de l'activité en cours et permet à l'apprenant de demander de l'aide en cliquant sur un bouton.

L'agent interface donne aussi accès, en mode de développement, à des outils pour examiner l'environnement sous forme graphique et d'y entrer des instances. La fenêtre de l'agent interface permet également de simuler les actions normalement issues de l'apprenant afin de tester le modèle de connaissance avant d'écrire les scripts d'interaction.

4.5 L'agent apprenant

Comme il est mentionné dans la description du suivi de l'apprenant, c'est l'agent apprenant qui tranche lorsqu'une ambiguïté persistante survient. Il n'y a pas encore de modèle de l'apprenant qui permet à l'agent apprenant de prendre une décision éclairée. Par conséquent, le

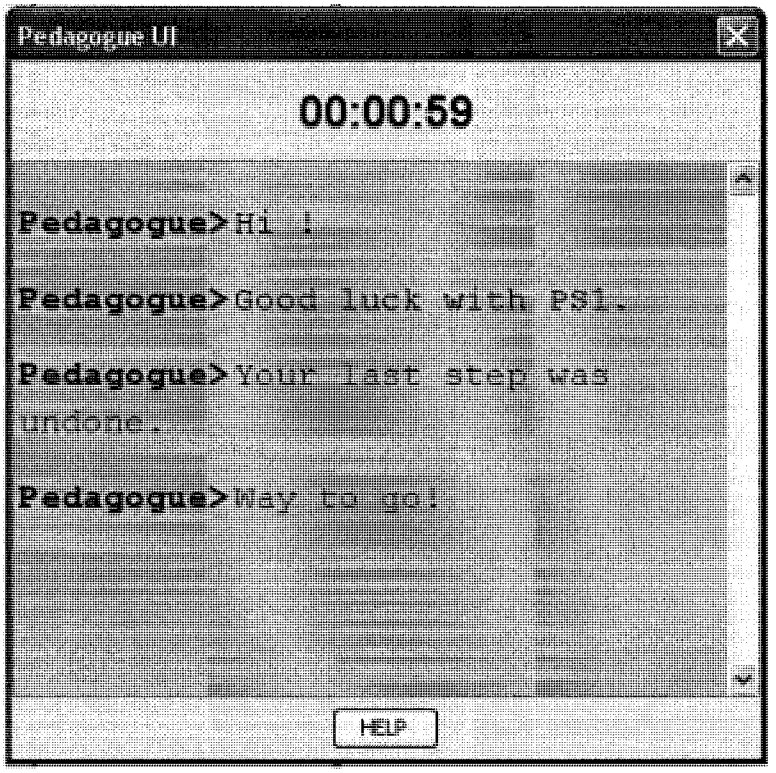

Figure 36 - La fenêtre du pédagogue.

choix se fait en donnant le bénéfice du doute à l'apprenant : si l'ambiguïté concerne une procédure valide et une procédure invalide, on reconnaît à l'apprenant la procédure valide; s'il y a une ambiguïté entre deux procédures valides, c'est celle avec la plus haute priorité qui est créditée à l'apprenant; si l'ambiguïté touche deux procédures invalides, une d'entre elles est choisie arbitrairement.

4.5.1 Considérations pour travaux futurs

Pendant la période où l'architecture de connaissance a été conçue et développée, plusieurs discussions du groupe ASTUS ont touché la modélisation de l'apprenant. L'idée dominante de ces discussions était que la nature hiérarchique des connaissances procédurales se prête bien à une modélisation de l'apprenant à l'aide des réseaux bayésiens. Nous examinerons les interrogations possibles de l'agent pédagogue sur un tel modèle de l'apprenant; en voici deux qui ont retenu notre attention : 1) Quelle est la prochaine procédure primitive ayant la plus

forte probabilité d'être effectuée par l'apprenant au «prochain coup» ? 2) Quelle est la probabilité que l'apprenant puisse atteindre un but, en particulier le but initial qui représente la résolution complète du problème ? Ces deux interrogations sont associées à des évènements dont la probabilité est particulièrement intéressante : 1) la probabilité qu'une procédure possible soit choisie par l'apprenant pour un but donné [p. ex. p(PC1 | But1)]; 2) la probabilité que l'apprenant puisse compléter une procédure choisie pour satisfaire un but. [p. ex. p(compléter-PC1 | PC1 pour But1)]. Bien qu'un modèle basé sur des statistiques accumulées par les apprenants pourrait s'appuyer sur la littérature [14], notre intérêt est plutôt dirigé vers les heuristiques qui permettraient d'établir les valeurs initiales d'un tel modèle. En effet, il nous apparaît intéressant de déduire des structures de représentation de connaissance des données telles que la difficulté que pose la réussite d'une procédure complexe. Ces déductions pourraient être basées sur des modèles issus des expérimentations en psychologie cognitive. Ceci constituerait une première étape vers un modèle de l'apprenant qui peut agir en tant que simulation d'un apprenant.

4.6 L'agent pédagogue

Puisque la réalisation d'un agent pédagogue complet est un projet d'envergure en soi, nous avons réalisé une implémentation minimale. La première fonction de cet agent pédagogue minimal est de choisir l'activité à chaque itération de la boucle externe. Plus précisement, l'agent pédagogue doit choisir une activité lorsque le mode retenu est celui basé sur un curriculum. L'agent pédagogue a comme objectif global d'aider l'apprenant à atteindre le plus rapidement possible l'ensemble des objectifs pédagogiques du curriculum. Le processus de choix de l'activité doit donc prendre les décisions qui optimiseront ce processus d'apprentissage. À partir de la mémoire épisodique de l'expert d'un laboratoire donné, l'agent pédagogue peut estimer le potentiel d'une activité vis-à-vis des objectifs qui ne sont toujours pas atteints. En effet, à partir de cet ensemble de connaissances épisodiques, on peut vérifier à combien de reprises il aurait été possible de satisfaire les objectifs en tenant compte des contraintes qui leur sont appliquées (voir section 1). À l'initialisation du prototype et à chaque

fois qu'une activité est terminée, l'agent pédagogue fera donc le choix de l'activité qui maximise l'atteinte d'un ou plusieurs objectifs. Toutefois, l'algorithme qui fera le choix optimal en fonction des objectifs à satisfaire et du potentiel de chaque activité disponible n'a pas encore été conçu. Par conséquent, c'est simplement l'activité qui sollicite le plus grand nombre de fois l'objectif qui est le plus faiblement atteint, qui est choisi.

La deuxième fonction de l'agent pédagogue est évidemment de réagir en fonction des actions produites par l'apprenant. Pour ce faire, un système expert simple a été conçu et implémenté. Ce système expert a accès dans sa mémoire de travail à toute la base de connaissance et ses règles fournissent ultimement une séquence d'actions pédagogiques qui sont envoyées à l'agent interface pour qu'elles se manifestent dans l'interface du laboratoire ou dans la fenêtre du pédagogue, par exemple, un message pour l'apprenant et l'annulation de la dernière étape qu'il a effectué parce qu'elle n'était pas prévue par le suivi. Pour le moment, les actions pédagogiques sont limitées. Il est possible d'envoyer un des quatre types de message (voir section 4), d'annuler une action faite par l'apprenant, de faire une action à la place de celui-ci et de faire ou d'annuler une procédure complexe. C'est la présence des scripts d'interactions qui permet à l'agent apprenant de produire les manipulations des composants graphiques associés à une action que le pédagogue fait à la place de l'apprenant. Des travaux futurs du côté de l'agent interface permettront une gamme d'action beaucoup plus intéressante, par exemple de mettre en surbrillances les instances d'un concept ou bien de désactiver les composants graphiques qui sont liés au script d'interaction d'une procédure primitive donnée.

4.6.1 L'approche par situation

Les éléments de la base de connaissance et les évènements qui donnent le contrôle à l'agent pédagogue forment les **sources**. Les sources actuellement supportées sont : 1) un but atteint avec succès, 2) un but échoué, 3) une demande d'aide, 4) une action imprévue, 5) un délai et 6) la mise à jour du suivi. D'autres sources s'ajouteront plus tard, par exemple une faible probabilité de réussite d'une procédure complexe venant du modèle de l'apprenant. Un premier groupe de règles réagit en fonction des sources, tandis qu'un deuxième groupe de

règles décrit des **situations**. Une situation est un regroupement de sources pouvant déclencher une **stratégie d'intervention** particulière. Par exemple, lorsque que la situation qui combine les sources 1) et 6) se présente, la stratégie produit une action de type «message d'approbation». En tout temps, une situation courante est identifiée, soit par un jeu de priorité, soit parce qu'une situation plus complexe a englobé plusieurs situations plus simples. En effet, lorsqu'une situation est constituée d'un sous-ensemble de sources d'une autre situation, seule la dernière est retenue. Une fois la situation identifiée, la stratégie qui lui est associée est exécutée. Cette dernière peut également accéder à la base de connaissance afin de pouvoir ajuster la séquence d'actions qu'elle produit. Finalement, une **stratégie pédagogique** définit les associations entre situations et stratégies pédagogigues. Par exemple, une stratégie pédagogique plus passive peut ignorer la situation donnée en exemple ci-haut.

4.7 Conclusion

Ce chapitre a présenté un prototype complet et fonctionnel, parce qu'il supporte, certes simplement, l'itération des boucles externes et internes. Les limitations actuelles ont été clairement indiquées et les solutions temporaires qui contournent ces dernières ont été présentées. Finalement, ce chapitre a permis de rendre concret le rôle de l'architecture de connaissance.

Conclusion

Ce mémoire présente un portrait clair et réaliste de l'avancement des travaux du groupe ASTUS. Il est le fruit d'un important changement de cap dans les priorités du groupe. En effet, la priorité accordée dans les travaux passés de Mayers, Fournier-Viger et Najjar à la plausibilité cognitive de l'architecture de connaissance est passée au second plan derrière son rôle pédagogique. Par exemple, la justification première de la représentation particulière des procédures complexes est le fait qu'elle facilite l'enseignement d'un savoir-faire.

Ce mémoire a d'abord fourni, pour la première fois au groupe ASTUS, des critères tangibles pour évaluer comment une architecture de connaissance permet à un agent expert de jouer son rôle dans un STI. Ces critères ont permis de mieux comprendre les particularités de l'approche globale d'ASTUS, en la comparant aux approches concurrentes. Puis, ce mémoire a présenté une architecture des connaissances originale, élaborée en fonction de lignes directrices claires. Ces lignes directrices ont fixé les caractéristiques essentielles que l'on doit retrouver dans une architecture de connaissance pour atteindre les objectifs sous-jacents à la conception du prototype de la plate-forme ASTUS. Ensuite, des exemples simples, mais pertinents ont mis en évidence les particularités de cette architecture. Nous avons discuté sur la façon dont ces exemples montrent que l'architecture développée possède les caractéristiques recherchées. Finalement, ce mémoire a montré comment cette architecture de connaissance s'intègre dans une version fonctionnelle, mais limitée du prototype. Cette intégration a montré comment l'architecture de connaissance permet à l'agent expert de jouer son rôle lors d'un cycle complet de l'exécution des boucles externes et internes.

En faisant de choix d'être pragmatique au moment de concevoir la nouvelle architecture de connaissance, nous avons ramené à des perspectives plus réalistes certaines ambitions de l'architecture d'ASTUS qui n'avaient jamais été matérialisées. Cependant, cela n'a pas été suffisant pour permettre d'arriver rapidement à la modélisation d'un domaine complet, tel que nous l'avions espéré au départ. Toutefois, l'ensemble des modèles simples fournis en exemple

regroupe sensiblement l'ensemble des défis que peut poser la modélisation d'un domaine qui pourrait être réellement utilisée auprès d'apprenants.

De plus, ce mémoire ne fournit pas toutes les justifications théoriques pour les choix de conception qui ont été faits. Nous avons pris cette décision, car plusieurs d'entre eux ont été faits par intuition, et qu'ils seront raffinés et présentés dans le cadre d'une thèse de doctorat.

Lorsque les travaux sur l'agent interface et sur l'agent pédagogue seront plus avancés, nous aurons tous les éléments en place pour mettre en évidence l'originalité de notre architecture de connaissance. En particulier, nous pourrons montrer des exemples qui débutent au niveau des interactions produites par l'apprenant dans l'interface du laboratoire, jusqu'à une rétroaction du pédagogue qui en plus de fournir un message d'aide, modifie cette dernière. À ce moment, le groupe ASTUS ne sera plus très loin du prototype de la plate-forme décrit en introduction. Des travaux complémentaires devront être faits, comme l'élaboration d'un modèle simple, mais fonctionnel de l'apprenant, pour couvrir, du moins partiellement, tous les aspects fondamentaux d'un STI. L'étape suivante sera la modélisation de domaines plus complexes, comme un laboratoire virtuel pour le cours «Génie génétique I» du département de biologie pour lequel plusieurs étapes préliminaires ont déjà été franchies.

Les possibilités de travaux futurs sont immenses, mais le plus important à moyen terme sera certainement la conception d'une nouvelle génération de l'architecture de connaissance qui étendra les possibilités de l'agent expert de deux façons importantes. Cette future architecture de connaissance étendra la représentation des connaissances sémantiques aux notions théoriques qui sont présentement représentées uniquement sous forme procédurale. De plus, elle contiendra des connaissances procédurales de plus haut niveau qui encoderont des stratégies de résolution de façon explicite, entre autres capable d'interpréter les connaissances sémantiques mentionnées précédemment.

Annexe A

Les outils du prototype

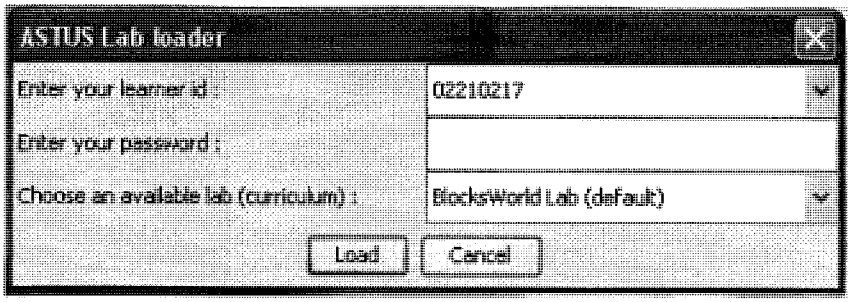

Figure 37 - Boîte de dialogue pour le choix du laboratoire.

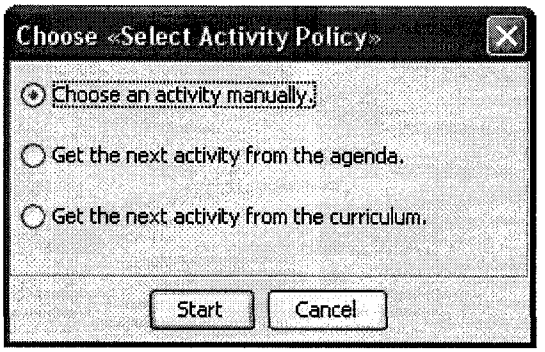

Figure 38 - Boîte de dialogue pour choisir le type de boucle externe.

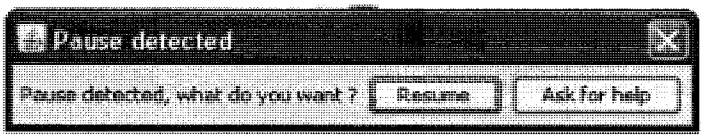

Figure 44 - Boîte de dialogue affichée par l'agent interface lorsqu'il détecte une pause.

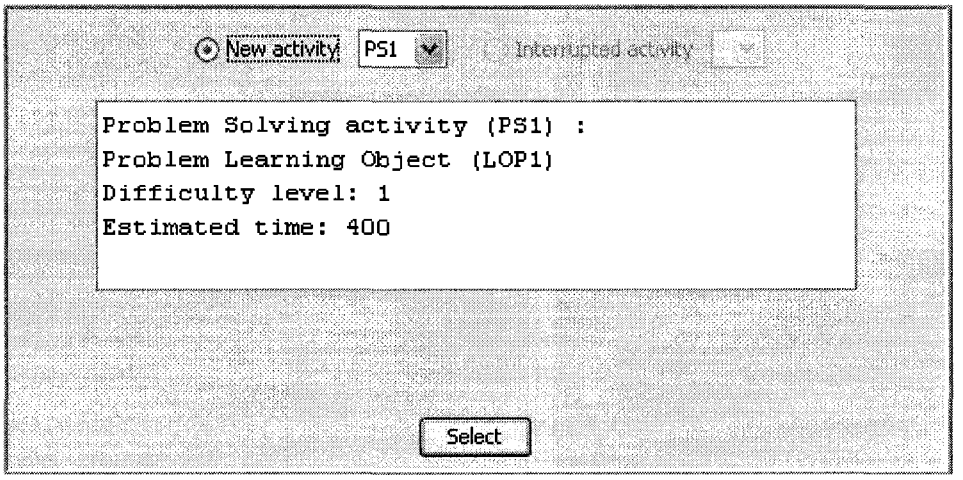

Figure 39 - Boîte de dialogue pour le choix de l'activité par l'apprenant.

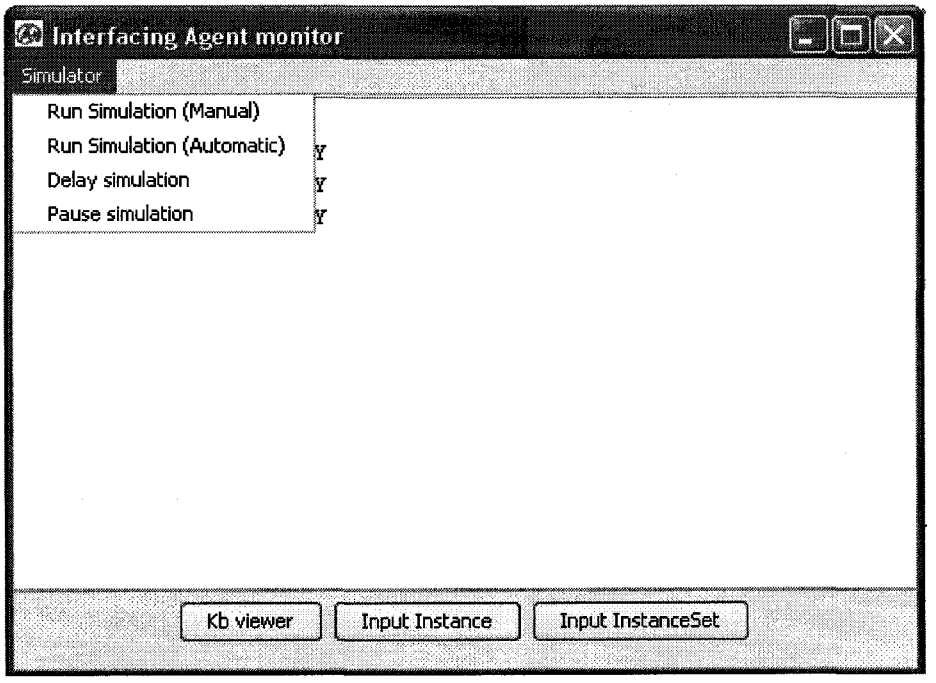

Figure 40 - Fenêtre de contrôle de l'agent interface.

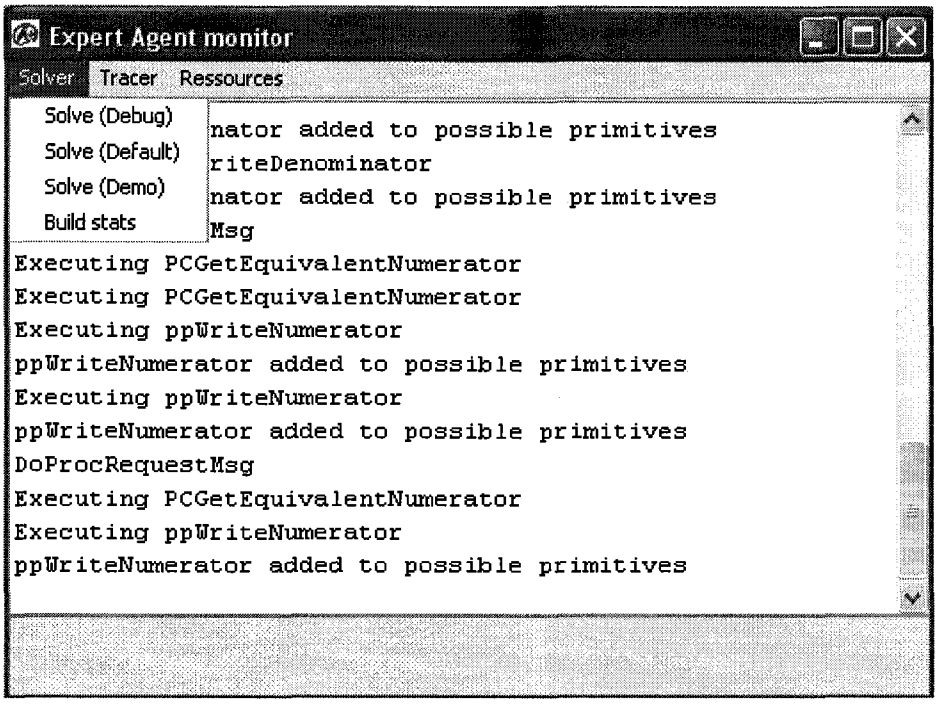

Figure 41 - Fenêtre de contrôle de l'agent expert.

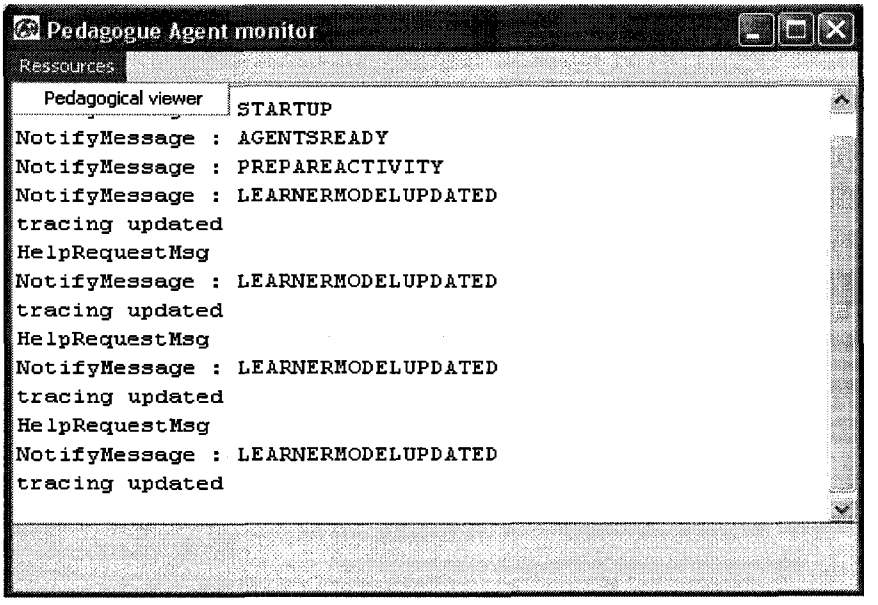

Figure 42 - Fenêtre de contrôle de l'agent pédagogue.

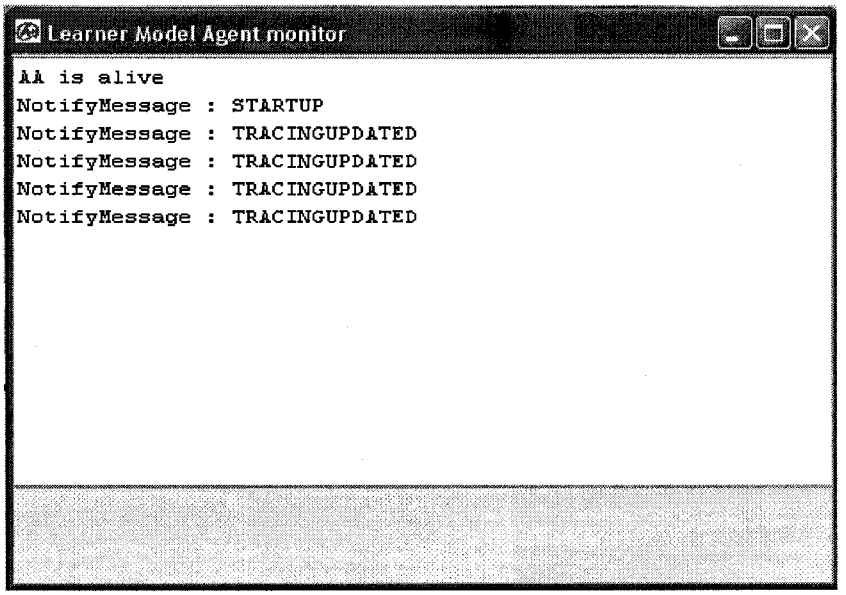

Figure 43 - La fenêtre de contrôle de l'agent apprenant.

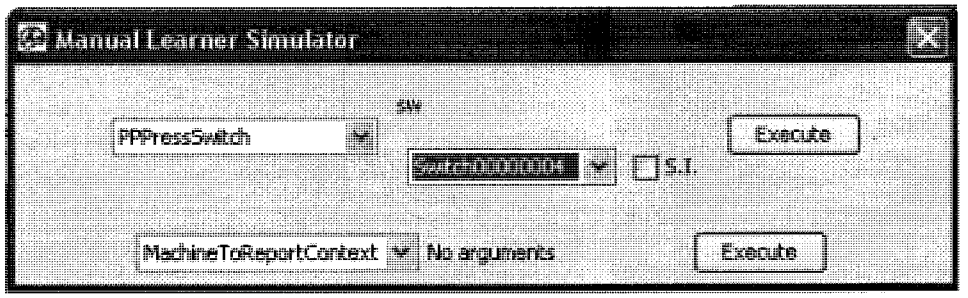

Figure 45 - Fenêtre du simulateur des actions de l'apprenant.

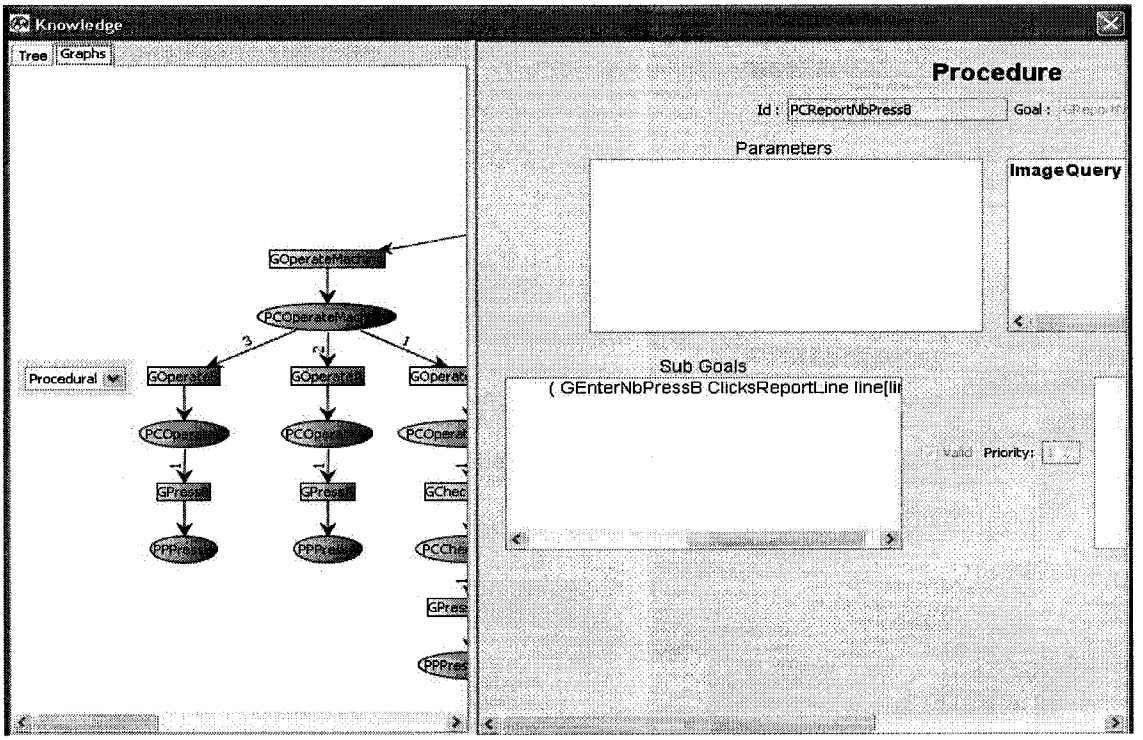

Figure 46 - Fenêtre qui permet de visualiser les connaissances procédurales.

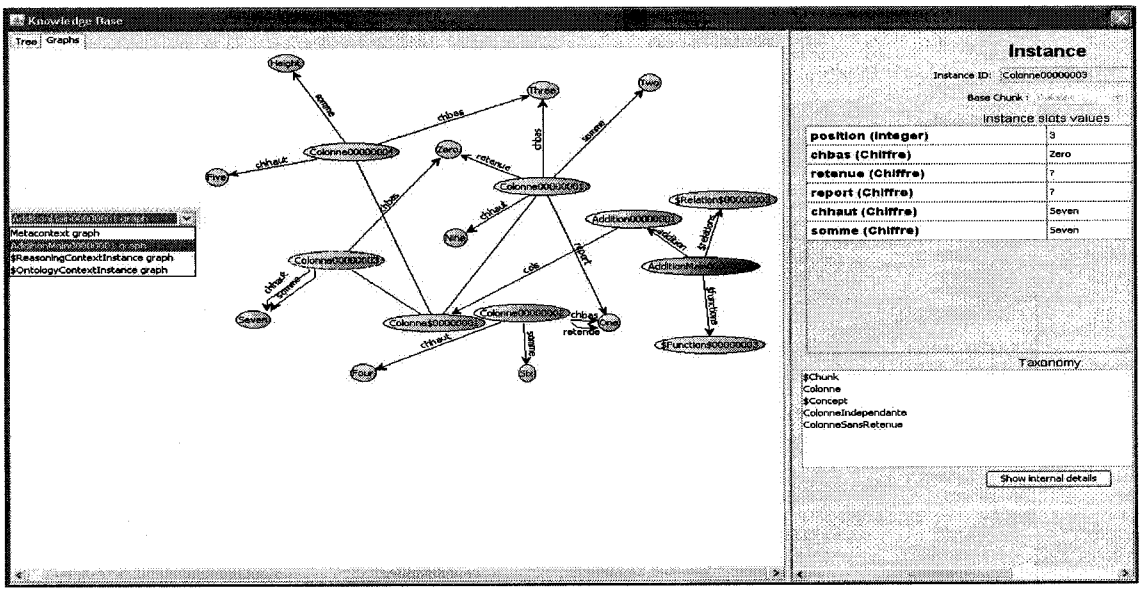

Figure 47 - Fenêtre qui permet de visualiser l'environnement.

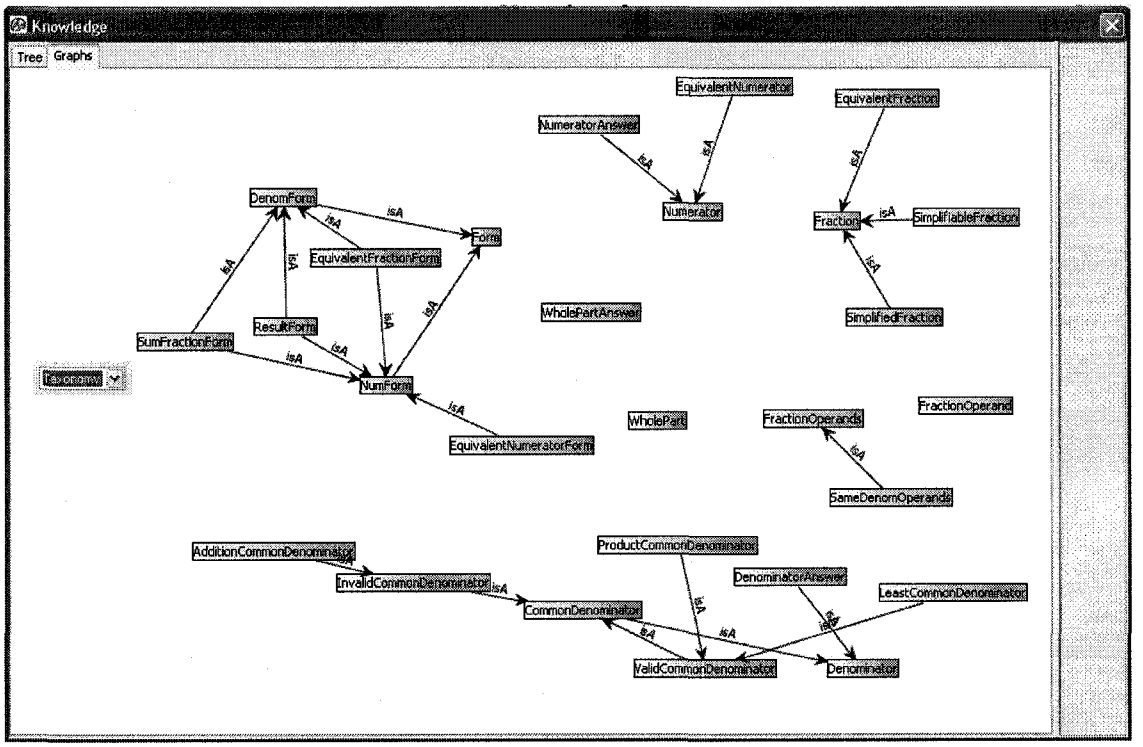

Figure 48 - Fenêtre qui permet de visualiser les connaissances sémantiques.

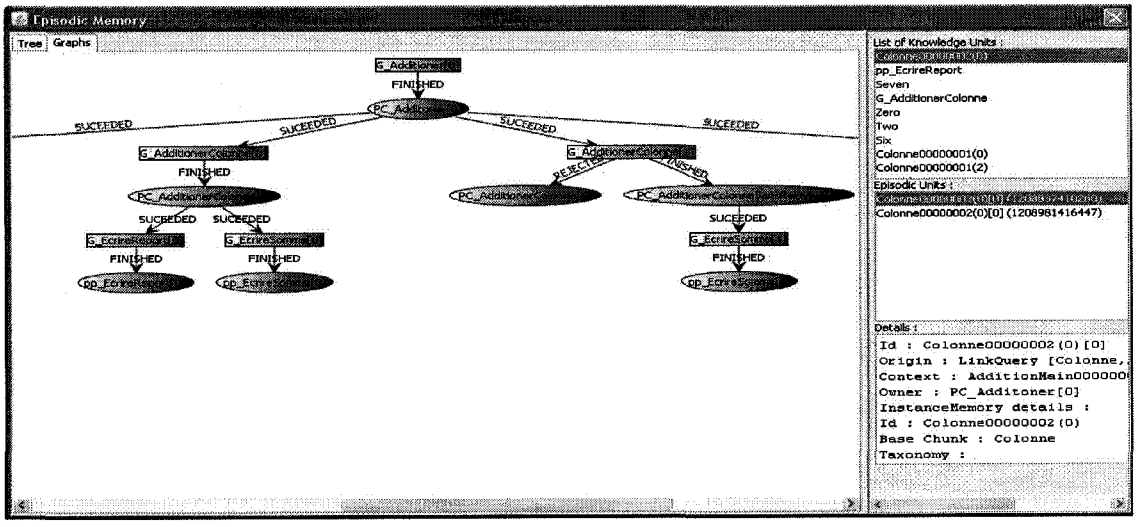

Figure 49 - Fenêtre qui permet de visualiser les connaissances épisodiques.

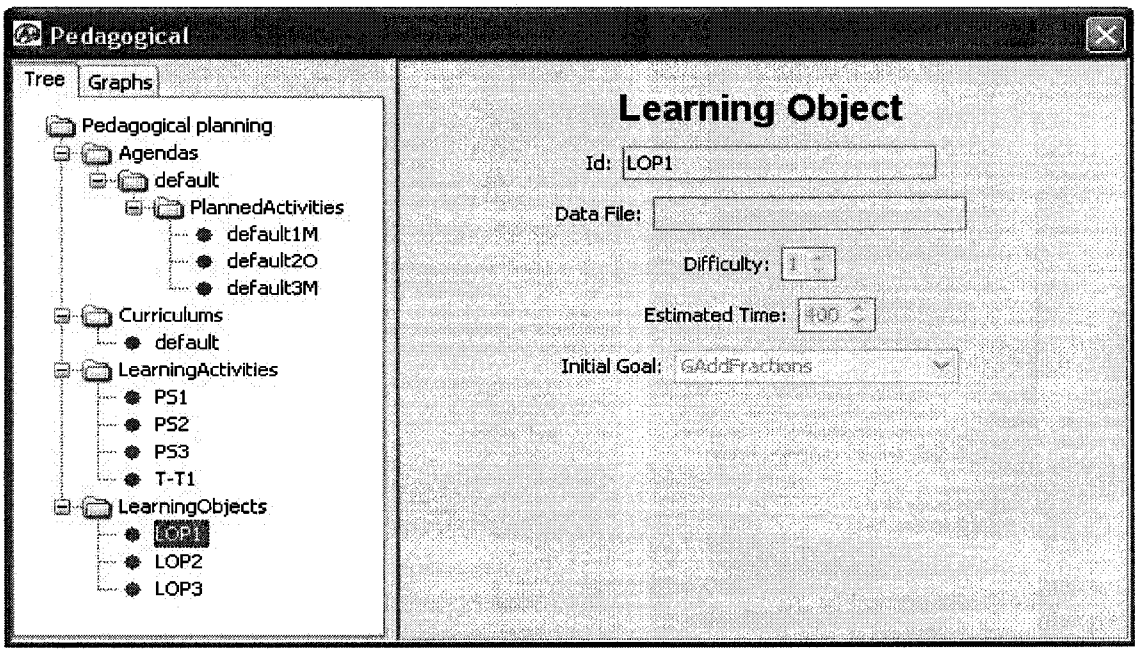

Figure 50 - Fenêtre pour visualiser les ressources pédagogiques.

Annexe B

Le monde des blocs

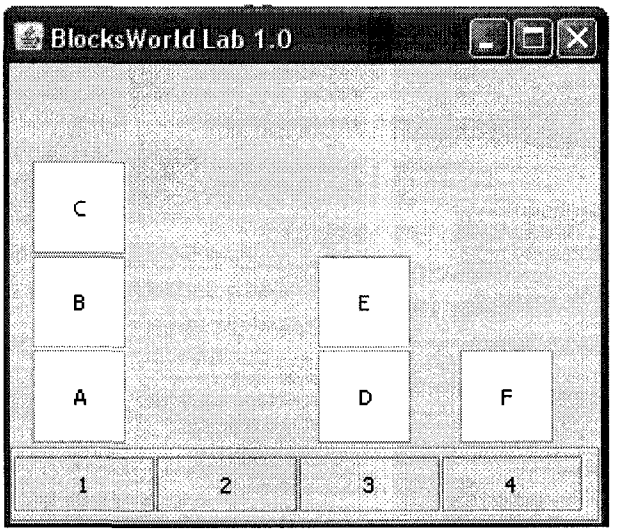

Figure 51 - Le monde des blocs configuré pour le problème des blocs à échanger.

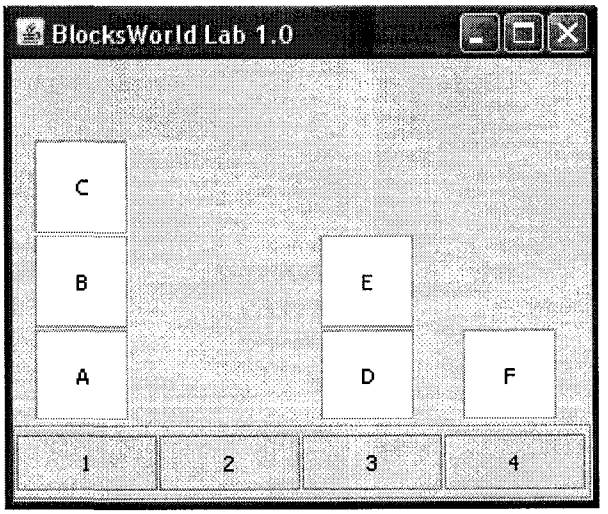

Figure 52 - Le monde des blocs configuré pour le problème de la pile à dupliquer.

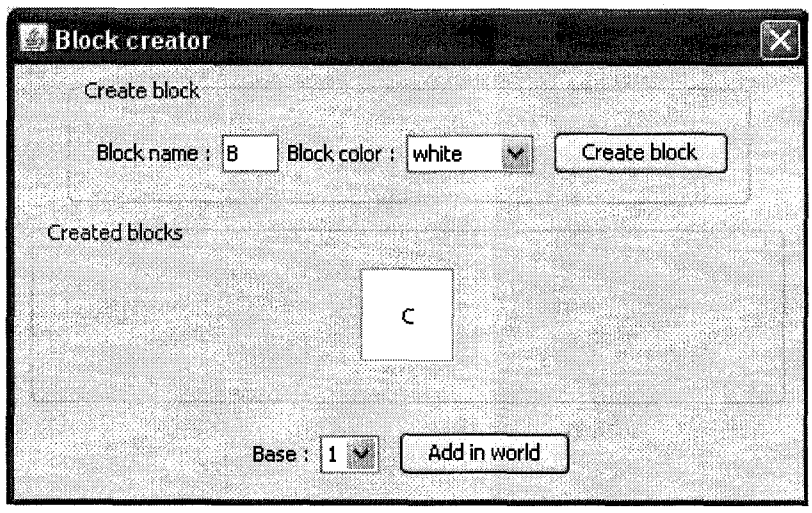

Figure 53 - Le contexte pour la création de blocs.

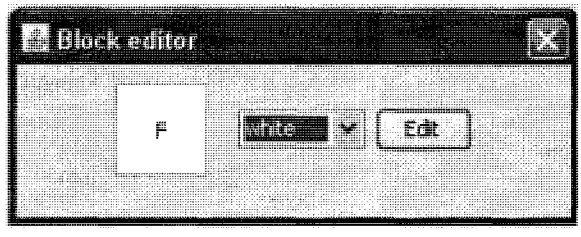

Figure 54 - Le contexte pour l'édition d'un bloc.

Bibliographie

[1] ADL, ADL Official SCORM overview http://www.adlnet.gov/

[2] Aleven, V., McLaren, B. M., Sewall, J., & Koedinger, K. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. M. Ikeda, K. D. Ashley, & T. W. Chan (Eds.), *Proc. of the 8th International Conference on Intelligent Tutoring Systems* (ITS 2006), (pp. 61-70). Berlin: Springer Verlag.

[3] Aleven, V., Sewall, J., McLaren, B. M., & Koedinger, K. R. (2006). Rapid authoring of intelligent tutors for real-world and experimental use. Kinshuk, R. Koper, P. Kommers, P. Kirschner, D. G. Sampson, & W. Didderen (Eds.), *Proc. of the 6th IEEE International Conference on Advanced Learning Technologies* (ICALT 2006), (pp. 847-851). Los Alamitos, CA: IEEE Computer Society.

[4] Altmann, E. et Trafton, J., 2002. Memory for goals : An architectural perspective. *Cognitive Science* 26, 39-83.

[5] Anderson, J. R. (1988). The expert module. M. Polson & J. Richardson (Eds.), *Handbook of Intelligent Training Systems*. Hillsdale, NJ: Erlbaum, 21-53.

[6] Anderson, J. R. & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Erlbaum.

[7] 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.

[8] Ashcroft, Edward; and Zohar Manna (1971). The translation of go to programs to 'while' programs. *Proc. of IFIP Congress*.

[9] Baader, F., Calvanese, D., McGuinness, D., Nardi, D. et Patel-Schneider, P. (Eds.), *The Description Logic Handbook : Theory, Implementation and Applications*. Cambridge University Press,pp. 495-505.

[10] 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.

[11] Bouchard, F., 2005. *Guides de conception d'interfaces pour les systèmes tutoriels intelligents.* Mémoire, Université de Sherbrooke.

[12] Brusilovsky P. (1993) Student as user: Towards an adaptive interface for an intelligent learning environment. P.Brna, S.Ohlson and H.Pain (Eds.) *Proc. of AI-ED'93, World Conference on Artificial Intelligence and Education*, Edinburgh, 23-27 August 1993, AACE, Charlottesville, p.386-393

[13] Chi, Min & VanLehn, K. (2007) The impact of explicit strategy instruction on problem-solving behaviros across intelligent tutoring systems. D. McNamara & G. Trafton (Eds.) *Proc. of the 29th Annual Conference of the Cognitive Science Society.* pp. 167-172 Mahwah, NJ: Erlbaum.

[14] Conati, C., Gertner, A., & VanLehn, K. (2002). Using Bayesian networks to manage uncertainly in student modeling. *User Modeling & User-Adapted Interaction.* 12(4), 371-417.

[15] Cooper, R. et Shallice, T., 2000. Contention scheduling and the control of routine activities. *Cognitive Neuropsychology* 17 (4), 297-338.

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

[17] Corbett, A., Anderson, J. R. and Koedinger, K. R. Cognitive Mastery Learning http://www.andrew.cmu.edu/user/jb8n/its2004/cognitive_mastery.ppt

[18] Davis, Randall, Shrobe, Howard, Szolovits, Peter : What is A Knowledge Representation? *AI Magazine* 14(1). Spring 1993, 17-33.

[19] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur and Yarden Katz. Pellet: A practical OWL-DL reasoner, *Journal of Web Semantics*, 5(2), 2007.

[20] Forgy, C. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artificial Intelligence*, 19, pp 17-37, 1982

[21] Fournier-Viger, P. (2005). *Un modèle de représentation des connaissances à trois niveaux de sémantique pour les systèmes tutoriels intelligents.* Mémoire de maîtrise, Université de Sherbrooke, Sherbrooke, Canada.

[22] Fournier-Viger P., Najjar, M., Mayers, A. & Nkambou, R. (2006). From Black-box Learning Objects to Glass-Box Learning Objects. *Proc. of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006).* LNCS, Vol. 4053, pages 258-267, Springer-Verlag, Berlin.

[23] Gagne, Robert (1965). *The Conditions of Learning.* New York: Holt, Rinehart and Winston.

[24] Gomez-Perez, Asuncion; Corcho, Oscar. Ontology Specification Languages for the Semantic Web. IEEE Intelligent Systems January/February 2002 (Vol. 17, No. 1) pp. 54-60

[25] Ghallab, M, Nau, D, Traverso, P : *Automated Planning : theory and pratice.* Morgan Kaufmann Publishers, May 2004, 663 pages.

[26] Grandbastien, Monique; Labat, Jean-Marc (Eds). *Environnements informatiques pour l'apprentissage humain*, Hermès science publications : Lavoisier, 2006

[27] Hafsaoui, M., 2005. *Élaboration d'une stratégie tutorielle dans un environnement intelligent d'apprentissage des sciences.* Mémoire, Université de Sherbrooke.

[28] Hamadouche, M., 2004. *Modèle de communication entre un laboratoire virtuel réflexif et un tuteur dans un système tutoriel intelligent.* Mémoire, Université de Sherbrooke.

[29] Heffernan, N, Choksey, S, : *An Evaluation of the Run-Time Performance of the Model-Tracing Algorithm of Two Different Production Systems: JESS and TDK* (Technical Report disponible à ftp://ftp.cs.wpi.edu/pub/techreports/pdf/03-31.pdf)

[30] Hustadt U. Do we need the Closed World Assumption in Knowledge Representation? *Proc. of 1st Workshop KRDB'94*, Saarbrücken, Allemagne, Septembre 1994.

[31] IEEE, 2005. Learning object metadata standard (LOM) of IEEE. http ://ltsc.ieee.org/wg12/index.html

[32] IMS Global Learning Consortium, 2005a. Content packaging information model et best practice and implementation guide, version 1.1.4 http://www.imsproject.org/content/packaging/index.cfm

[33] IMS Global Learning Consortium, 2005b. Learning design information model et best practice and implementation guide, version 1.0 http://www.imsglobal.org/learningdesign/index.cfm

[34] Kirschner, Paul A.; Sweller, John; Clark, Richard E. Why Minimal Guidance during Instruction Does Not Work: An Analysis of the Failure of Constructivist, *Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching Educational Psychologist*, v41 n2 p75-86 2006

[35] Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education,* 8, 30-43.

[36] Koedinger, K. R., Aleven, V. A. W. M. M., & Heffernan, N. T. (2003). Toward a Rapid Development Environment for Cognitive Tutors. U. Hoppe, F. Verdejo, & J. Kay (Eds.), *Proc. of the 11th International Conference on Artificial Intelligence in Education*, AI-ED 2003 (pp. 455-457). Amsterdam: IOS Press.

[37] Knublauch H., Fergerson R. W., Noy N. F., & Musen M. A. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. *Proc. of Third International Semantic Web Conference*, Hiroshima, Japan, 2004.

[38] Laird, Rosenbloom, Newell, John and Paul, Allen (1987). Soar: An Architecture for General Intelligence. *Artificial Intelligence*, 33: 1-64.

[39] Luger, G. F. (2005). *Artificial Intelligence : Structures and Strategies for Complex Problem Solving*. 5ème édition. Addison Wesley.

[40] Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R. (2007). Evaluating a simulated student using real students data for training and testing. *Proc. of the international conference on User Modeling.*

[41] Mager, Robert, F, *Preparing Objectives for Programmed Instruction.* San Francisco, California: Fearon Publishers, 1961

[42] Mayers, A., 1997. *Miace : Une architecture théorique et computationnelle de la cognition humaine pour étudier l'apprentissage.* Thèse de doctorat, Université de Montréal.

[43] Mayers, A., 2001. *Usager virtuel,* document non publié, Département d'informatique, Université de Sherbrooke.

[44] Mitrovic, A, Koedinger, K, Martin, B : A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling. *User Modeling 2003.* p. 147.

[45] Najjar, M., 2006. *Une approche cognitive computationnelle de représentation de la connaissance au sein des environnements informatiques d'apprentissage.* Thèse de doctorat, Université de Sherbrooke.

[46] Najjar, M., Mayers, A. & Fournier-Viger, P. (2006). A Novel Cognitive Computational Knowledge Model for Virtual Learning Environments. *The Int. Transactions on Advances in Engineering Education.* Vol 3 (4). p. 246-255.

[47] Norman, D. A. and Shallice, T. (1986). Attention to action: Willed and automatic control of behaviour. Davidson, R. J., Schwartz, G. E., and Shapiro, D. (Eds.). *Consciousness and Self-Regulation: Advances in Research and Theory.* Plenum Press.

[48] Reed K. S. (2007) *Cognition: theory and applications*, 7e Edition. Thomson Wadsworth.

[49] Ritter, Steven. Authoring Model-Tracing Tutors, Tech., Inst.,*Cognition and Learning,* Vol. 2, pp. 231-247.

[50] Ritter, S, Blessing, S, Wheeler, L : User Modeling and Problem-Space Representation in the Tutor Runtime Engine. *User Modeling* (2003) p. 147.

[51] Shapiro, J. Algebra Subsystem for an Intelligent Tutoring System, *International Journal of Artificial Intelligence in Education* (2005), 15, 205-228.

[52] Sirin E., Parsia B., Cuenca Grau B., Kalyanpur A. and Katz Y. Pellet: A practical OWL-DL reasoner, *Journal of Web Semantics*, 5(2), 2007.

[53] Sun Microsystems, Java BluePrints, Model-View-Controller J2EE pattern http://java.sun.com/blueprints/patterns/MVC-detailed.html

[54] Sweller, J., Van Merrienboer, J., & Paas, F. (1998). Cognitive architecture and instructional design. *Educational Psychology Review* 10: 251-296.

[55] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.,* 43(5-6) :907-928, 1995.

[56] VanLehn, K., Lynch, C., Schulze, K. Shapiro, J. A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., & Wintersgill, M. (2005). The Andes physics tutoring system: Lessons Learned. *International Journal of Artificial Intelligence and Education*, 15 (3), 1-47.

[57] VanLehn, K. (2006) The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education.* 16.

[58] VanLehn, K., Ohlsson, S., & Nason, R. (1994). Applications of simulated students: An exploration. *Journal of Artificial Intelligence in Education*, 5(2), 135-175.

[59] VanLehn, K., Bhembe, D., Chi, M., Lynch, C., Schulze, K., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., & Wintersgill, M. (2004). Implicit versus explicit learning of strategies in a non-procedural cognitive skill. J. C. Lester, R. M. Vicari, & F. Paraguacu, (Eds.), *Intelligent Tutoring Systems: 7th International Conference* (pp. 521-530). Berlin: Springer-Verlag Berlin & Heidelberg GmbH & Co. K.

[60] VanLehn, K, Lynch, C., Taylor, L., Weinstein, A., Shelby, R., Schulze, K., Treacy, D. & Wintersgill, M. (2002) Minimally invasive tutoring of complex physics problem solving. Cerri, SA, Gouarderes, G, Paraguacu, F (Eds.) *Intelligent Tutoring Systems*, 2002, 6th International Conference, Berlin: Springer, pages 367-376.

[61] W3C, OWL Web Ontology Language Overview http://www.w3.org/TR/owl-features/

[62] Welty, C., and Ferrucci, D. What's in an Instance? RPI Computer Science Technical Report. 1994.

[63] Wenger, E., 1987. *Artificial intelligence and tutoring systems : computational and cognitive approaches to the communication of knowledge.* Morgan Kaufmann Publishers Inc.