HAL
open science

OMAGE : Outils et Méthode pour la spécification des connaissances au sein d'un Atelier de Génie Educatif

Christophe Marquesuzaà

To cite this version:

Christophe Marquesuzaà. OMAGE : Outils et Méthode pour la spécification des connaissances au sein d'un Atelier de Génie Educatif. Autre [cs.OH]. Université de Pau et des Pays de l'Adour, 1998. Français. NNT : . tel-00003699

HAL Id: tel-00003699

https://theses.hal.science/tel-00003699

Submitted on 4 Nov 2003

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : 412

LABORATOIRE D'INFORMATIQUE APPLIQUÉE

THESE

présentée devant

l'UNIVERSITE DE PAU ET DES PAYS DE L'ADOUR

IUT de Bayonne - Pays Basque, Département Informatique

en vue de l'obtention du

DOCTORAT DE L'UNIVERSITE DE PAU ET DES PAYS DE L'ADOUR

Spécialité : **INFORMATIQUE**

par

Christophe MARQUESUZAÀ

OMAGE : OUTILS ET METHODE POUR LA SPECIFICATION DES CONNAISSANCES AU SEIN D'UN ATELIER DE GENIE EDUCATIF

Soutenue le 12 Février 1998 devant le jury composé de :

P. BAZEX	*Professeur à l'Université Toulouse III*
B. CAUSSE	*Professeur à l'IUT de Bayonne, Président*
G. GOUARDÈRES	*Professeur à l'IUT de Bayonne, Directeur de Recherche*
M. LAMURE	*Professeur à l'Université de Lyon I, Rapporteur*
T. NODENOT	*Maître de Conférences à l'IUT de Bayonne*
V. PRINCE	*Professeur à l'IUT de Paris VIII, Rapporteur*

IUT de Bayonne - Château Neuf - 64100 Bayonne - Tél. 05.59.46.32.19

N° d'ordre : 412

LABORATOIRE D'INFORMATIQUE APPLIQUÉE

THESE

présentée devant

l'UNIVERSITE DE PAU ET DES PAYS DE L'ADOUR

IUT de Bayonne - Pays Basque, Département Informatique

en vue de l'obtention du

DOCTORAT DE L'UNIVERSITE DE PAU ET DES PAYS DE L'ADOUR

Spécialité : **INFORMATIQUE**

par

Christophe MARQUESUZAÀ

OMAGE : OUTILS ET METHODE POUR LA SPECIFICATION DES CONNAISSANCES AU SEIN D'UN ATELIER DE GENIE EDUCATIF

Soutenue le 12 Février 1998 devant le jury composé de :

P. BAZEX	*Professeur à l'Université Toulouse III*
B. CAUSSE	*Professeur à l'IUT de Bayonne, Président*
G. GOUARDÈRES	*Professeur à l'IUT de Bayonne, Directeur de Recherche*
M. LAMURE	*Professeur à l'Université de Lyon I, Rapporteur*
U. NODENOT	*Maître de Conférences à l'IUT de Bayonne*
V. PRINCE	*Professeur à l'IUT de Paris VIII, Rapporteur*

IUT de Bayonne - Château Neuf - 64100 Bayonne - Tél. 05.59.46.32.19

Ezkerrik hainitz,
Merci beaucoup,
Muchas gracias,
Thank you very much...

Je tiens à remercier Monsieur Pierre BAZEX, Professeur à l'Université de Toulouse III, pour l'honneur qu'il me fait en participant à mon jury de thèse.

Ma gratitude va également à Monsieur Bernard CAUSSE, Professeur au département Informatique de l'Institut Universitaire de Technologie de Bayonne, pour l'intérêt qu'il a porté à mes travaux et pour avoir accepté d'être président de mon jury de thèse.

J'adresse mes sincères remerciements à Monsieur Guy GOUARDERES, Professeur au département Informatique de l'Institut Universitaire de Technologie de Bayonne, pour avoir accepté d'être mon directeur de recherche, pour la qualité de son encadrement et pour ses précieux conseils.

Je tiens aussi à remercier Monsieur Michel LAMURE, Professeur à l'Université de Lyon I pour avoir accepté d'être rapporteur de ce mémoire, pour ses remarques qui ont permis d'en améliorer la qualité et pour l'honneur qu'il me fait en participant à mon jury de thèse.

Je tiens à exprimer toute ma reconnaissance à Madame Violaine PRINCE, Professeur à l'Université de Paris VIII, pour avoir accepté d'être rapporteur de ce mémoire, pour ses remarques pertinentes et pour l'honneur qu'elle me fait en participant à mon jury de thèse.

Thierry, je t'adresse un grand, non, un énorme, non non, un gigantesque « merci pour tout ».

Je n'oublierai pas d'associer à mes remerciements toutes les personnes que j'ai pu côtoyer et qui, par leurs encouragements et leur affection ont permis que tout ceci soit possible. Je pense que vous vous reconnaîtrez dans ces quelques lignes et je vous dis à tous un grand merci « *Ezkerrik Hainitz Oroer* » .

Ce paragraphe est aussi particulièrement dédié à ma famille, à mes amis proches et à tous ceux que j'aime, présents ou partis trop tôt, et à qui je pense et penserai toujours.

Résumé

Les nouvelles technologies de l'information sont maintenant entrées de plain-pied au cœur de notre société. Elles provoquent de profonds changements dans notre vie de tous les jours, et plus particulièrement dans le monde du travail :

- les ouvriers sont devenus des personnels qualifiés chargés de piloter des machines sophistiquées. Ils ont maintenant un rôle de surveillance mais aussi de prise de décision rapide face à des situations imprévues (pannes, adéquation de la cadence de la machine aux directives reçues, etc...),
- le métier d'architecte est complètement bouleversé par les logiciels de Conception Assistée par Ordinateur. Ces logiciels performants libèrent les architectes de nombreuses tâches fastidieuses (dessin, recopie d'éléments de plan, documentation en vue de l'élaboration d'un devis, etc...) et leur donne par conséquent beaucoup plus de temps pour se concentrer sur les plans réalisés,
- les gestionnaires sont maintenant largement assistés par leurs ordinateurs et deviennent des experts en prise de décision (stratégique ou tactique) alors qu'autrefois leurs tâches se focalisaient principalement sur les aspects comptables.

Paradoxalement, le métier d'enseignant n'a pas vraiment évolué depuis Jules FERRY, même si les méthodes éducatives ont changé. Il est vrai que la relation maître-élève laisse peu de place à l'ordinateur (et ses facultés d'automatisation) et que toute tentative d'introduction de l'outil informatique à l'école se heurte à la méfiance des enseignants qui ont peur de perdre leur liberté en matière de choix éducatifs. En outre, les logiciels à finalité industrielle veulent intégrer une dimension « apprentissage » mais ne s'appuient pas sur les principes de transmission des connaissances maîtrisés par les enseignants.

Il n'est pas raisonnable de croire que ces paradoxes peuvent uniquement trouver des solutions dans la seule course technologique. En effet, les avancées en matière de multimédia, par exemple, ne présentent un intérêt dans l'enseignement que dans la mesure où elles sont intégrées dans un processus global de conception d'applications éducatives.

Nos travaux de recherche ont donc pour objectif principal de faciliter la tâche de l'enseignant dans la préparation de ses séquences pédagogiques en définissant les apports de l'outil informatique. Nous étudions ainsi quel doit être le support méthodologique d'un environnement informatique d'aide à la spécification des connaissances éducatives.

Nous avons par conséquent à répondre à deux problématiques complémentaires :

- développer les recherches pour formaliser les processus de transmission des connaissances utilisés par les enseignants dans leurs classes,
- en parallèle, et dans ce but, développer des environnements pour les enseignants qui leur permettront de décrire de tels processus.

Nous organisons alors nos travaux autour de trois axes principaux qui constituent les trois grandes parties de cette thèse.
Tout d'abord, nous avons choisi de baser nos recherches sur la mise en place d'enseignements axés sur la notion de situation d'enseignement qui permettent de situer l'apprenant au cœur même de l'apprentissage. Nous exposons ensuite la nécessité pour les enseignants de se reposer sur un processus de spécification formelle que nous définissons et pour lequel nous proposons un cycle de vie spécifique basé sur le prototypage rapide. Nous présentons enfin les outils et principes fondamentaux d'un environnement informatique dédié, pour lequel nous avons implanté un prototype, dont l'utilisation permet, et facilite, l'extraction ainsi que la manipulation des connaissances tout en respectant la diversité des approches pédagogiques.

Ainsi, les situations d'enseignements que nous avons ciblées sont celles qui sont préconisées dans le cadre de la formation des instituteurs en IUFM [1] sous le nom général de situations-problèmes. Ces dernières permettent de mettre les élèves en situation de projet (l'élève est actif dans sa formation) tout en répondant aux objectifs pédagogiques de l'enseignant et de l'institution. Le comportement de l'élève, ou du groupe, face à cette situation est alors le point de départ de situations de remédiation, de transfert ou de méta-cognition. L'activité de spécification consiste alors à faire apparaître les objectifs sous-tendant la situation-problème (ou à imaginer une situation correspondant à des objectifs donnés) ainsi que les directions que prendra l'enseignement selon les réactions des apprenants face à ces situations-problèmes.

En outre, seule une spécification formelle des connaissances fournit, non seulement un support non ambigu pour la communication des nombreux acteurs impliqués dans le projet éducatif, mais aussi permet une évaluation par prototypage rapide.
C'est la raison pour laquelle, après avoir pris en compte des environnements éducatifs existants, nous proposons notre propre processus de spécification des connaissances basé sur une spécification formelle évaluable par prototypage rapide.
Toutefois, la complexité d'utilisation d'un langage formel nécessite des connaissances, notamment mathématiques, pouvant rebuter un utilisateur non expert. Nous montrons donc que l'utilisation de

méta-outils CASE [2], permet de développer un environnement fournissant une réelle assistance à la fois adaptée aux besoins des acteurs (guidage dans le processus de développement) et suffisamment flexible pour permettre différentes façons de spécifier, différents points de vue sur une spécification et différents formalismes de représentations.

C'est dans cette optique que nous proposons une ontologie de l'enseignement s'appuyant sur une architecture orientée-objet qui dispose de fonctions de communication et de contrôle afin de disposer d'éléments de connaissance permettant de percevoir et d'agir sur leur environnement.
Nous avons développé un prototype de notre environnement. Il utilise le couplage entre un outil méta-CASE (« HARDY ») qui permet de fournir une interface diagrammatique supportant les différentes étapes du processus de développement que nous préconisons, et un générateur de système expert (« CLIPS ») qui assure la cohésion globale en fournissant un ensemble d'outils pour le guidage et la flexibilité du système.

L'une des perspectives de nos travaux de recherche consiste à prendre en compte le travail collaboratif qui est à la base de l'échange des connaissances, non seulement par un groupe d'enseignants en phase de spécification, mais aussi par un groupe d'apprenants en cours d'apprentissage basé sur les spécifications produites.
De même, la mise à disposition des différents concepts devrait pouvoir se faire à distance, par exemple au travers du réseau Internet.
En outre, une architecture fonctionnelle et structurelle à base d'agents cognitifs devrait être mise en place en collaboration avec des travaux développés au sein de notre laboratoire. En effet, une structure de système multi-agents, organisée autour des trois fonctions principales que sont la perception du contexte, la prise de décision et la mise en action (avec résolution des conflits, mécanismes de coopérations, etc...) nous permettrait de disposer d'un environnement respectant les trois principes fondamentaux de réutilisation, de guidage et de flexibilité.
Enfin, les différents principes et outils développés par cette thèse devraient s'étendre aux processus développés dans les logiciels à finalité industrielle et en particulier dans le domaine des systèmes de production automatisés qui est au cœur de notre laboratoire. Par exemple, cela pourrait concerner la communication entre le système physique de production, son système de pilotage et le système de prise de décision.

[1] Acronyme de « Institut Universitaire de Formation des Maîtres »
[2] Acronyme de « *Computer Aided Software Engineering* »

OMAGE : Outils et Méthode pour la spécification des connaissances au sein d'un Atelier de Génie Educatif

Table des matières

RESUME I

TABLE DES MATIERES V

LISTE DES FIGURES XI

LISTE DES TABLEAUX XIII

CHAPITRE 1 : INTRODUCTION 1

PREMIERE PARTIE : VERS UN ENVIRONNEMENT DE SPECIFICATION D'ENSEIGNEMENTS ... 5

RESUME DE LA PREMIERE PARTIE 5

PLAN DE LA PREMIERE PARTIE 7

CHAPITRE 2 : VERS UNE NOUVELLE CONCEPTION DE L'ENSEIGNEMENT 9

Paragraphe 1 : Construction des enseignements 9

1. Introduction 9

1.1. La technologie éducative 10

1.2. Comment mettre en place une technologie éducative ? 11

2. Le travail du pédagogue 12

2.1. Avant la classe : anticiper 14

2.2. Pendant la classe : évaluer et évoluer 15

2.3. Après la classe : réévaluer 15

3. De la modélisation d'une discipline à la transdisciplinarité 17

3.1. Quelques définitions 17

3.1.1. Les connaissances déclaratives 17

3.1.2. Les connaissances procédurales 18

3.1.3. La matrice disciplinaire 18

3.2. Vers un enseignement transdisciplinaire 19

Paragraphe 2 : De la situation d'apprentissage à la situation d'enseignement 22

1. Définition d'une situation d'enseignement 24
2. La métacognition 27
2.1. Définition 27
2.2. Importance de la métacognition 28

CHAPITRE 3 : DU ROLE DE L'INFORMATIQUE DANS L'ENSEIGNEMENT 29
Paragraphe 1 : Le génie éducatif 29
1. Présentation générale 29
2. Les Systèmes Tuteurs Intelligents 32
3. Vers des EIAO centrés sur l'utilisateur 34
Paragraphe 2 : Gestion et acquisition des connaissances 36
1. Trois principales approches 36
2. Techniques d'acquisition des connaissances 37
3. Méthodes d'acquisition des connaissances 41
3.1. KOD 41
3.2. KADS 42
3.3. CommonKADS 44

CHAPITRE 4 : LA SPECIFICATION DES ENSEIGNEMENTS 47
Paragraphe 1 : Importance de l'ingénierie des besoins 48
1. Qu'est ce qu'un besoin ? 49
2. Les activités de l'ingénierie des besoins 49
Paragraphe 2 : Vers une spécification formelle des connaissances 50
1. Objectifs 51
1.1. Un premier objectif : la communication 51
1.2. Un second objectif : une bonne conception 52
1.3. Objectifs connexes 53
2. Classification 54
3. Apports du génie logiciel 58
3.1. Quelques définitions 59
3.2. Les outils de prototypage 60
3.2.1. Les générateurs d'applications 61
3.2.2. Les boîtes à outils 62
4. Conclusion 63

DEUXIEME PARTIE : POUR UN ENVIRONNEMENT DE SPECIFICATION D'ENSEIGNEMENTS 67

RESUME DE LA DEUXIEME PARTIE 67

PLAN DE LA DEUXIEME PARTIE 69

CHAPITRE 5 : CAHIER DES CHARGES DE OMAGE 71

CHAPITRE 6 : SPECIFICATION DES CONNAISSANCES DANS UN ENVIRONNEMENT EDUCATIF 77

Paragraphe 1 : Le modèle du « Campus Virtuel » 79

1. Présentation des éléments principaux 79
2. Spécification de l'éditeur de scénario d'apprentissage 80
3. Conclusions 81

Paragraphe 2 : Proposition d'un processus de spécification en plusieurs phases 83

1. Modélisation du contexte général 85
2. Etablissement de la matrice disciplinaire 85
3. Définition des objectifs d'apprentissage 86
4. Mise en place des stratégies globales d'enseignement 87
5. Caractérisation des situations-problèmes 87
6. Caractérisation des situations d'apprentissage « classiques » 89

CHAPITRE 7 : UN CYCLE DE DEVELOPPEMENT ADAPTE AUX CONTRAINTES PEDAGOGIQUES 91

Paragraphe 1 : Le prototypage et ses outils 91

1. Différents cycles de prototypage 92
2. Les outils CASE et méta-CASE 97
   - 2.1. Définitions 97
     - 2.1.1. Qu'est ce que CASE ? 98
     - 2.1.2. Qu'est ce que méta-CASE ? 98
   - 2.2. Pourquoi de tels outils ? 99
   - 2.3. Quelques exemples de produits méta-CASE 101
     - 2.3.1. Quelques produits méta-CASE non commerciaux 101
     - 2.3.2. Quelques autres produits méta-CASE commerciaux 102

Paragraphe 2 : Notre environnement de développement d'enseignements 102

1. Cycles de vie des logiciels éducatifs 103
2. Conclusion : principes fondamentaux de notre environnement 106

TROISIEME PARTIE : NOTRE ENVIRONNEMENT DE SPECIFICATION D'ENSEIGNEMENTS.111

RESUME DE LA TROISIEME PARTIE 111
PLAN DE LA TROISIEME PARTIE 113

CHAPITRE 8 : ORGANISATION DES CONNAISSANCES 115
Paragraphe 1 : Représentation des connaissances par ontologie 115
1. Les ontologies, un mécanisme de spécification 117
2. Vers une ontologie de l'enseignement 119
Paragraphe 2 : Représentation structurelle 120
1. Définition du contexte général 122
1.1. Représentation conceptuelle 122
1.2. Mise en pratique 122
2. Matrice Disciplinaire 122
2.1. Représentation conceptuelle 124
2.2. Mise en pratique 126
3. Focus 126
3.1. Représentation conceptuelle 126
3.2. Mise en pratique 126
4. Objectifs et Stratégies 127
4.1. Représentation conceptuelle 127
4.2. Mise en pratique 128
4.3. Conclusion 129
5. Situations d'enseignement 129
5.1. Représentation conceptuelle 136
5.2. Mise en pratique 130
5.3. Conclusion 131
6. Vue d'ensemble 131

CHAPITRE 9 : PRINCIPES D'INTEGRATION DANS UN TI D'AIDE A LA SPECIFICATION 133
Paragraphe 1 : Architecture de notre environnement 133
1. Trois principes de base : guidage, flexibilité et réutilisation 134
1.1. Guidage 134
1.2. Flexibilité 138
1.2.1. Différents points de vue sur une spécification éducative 138
1.2.2. Différentes façons de produire des spécifications éducatives 139
1.3. Réutilisation 140
2. Du schéma M.V.C. à l'approche M.PV.C. 141
Paragraphe 2 : Quelle interface utilisateur ? 144

CHAPITRE 10 : LA REALISATION DE OMAGE....149
Paragraphe 1. Quels outils ?....149
1. Les paradigmes objet et logique....149
2. Outils logiciels....152
2.1. LISP....152
2.2. PROLOG....153
2.3. CLIPS....154
Paragraphe 2. Notre prototype....155
1. Logiciels choisis....155
1.1. HARDY....155
1.2. wxCLIPS....156
1.3. wxWindows....157
2. Présentation du processus de développement de notre prototype et des outils développés....157
2.1. Les différentes étapes du processus....158
2.1.1. Matrice disciplinaire....158
2.1.2. Objectifs d'apprentissage....159
2.1.3. Stratégies globales d'enseignement....159
2.1.4. Situations-problèmes....159
2.2. Outils associés....160

CHAPITRE 11 : CONCLUSIONS ET PERSPECTIVES....163
Paragraphe 1 : Conclusions....163
Paragraphe 2 : Perspectives....164

GLOSSAIRE DES DEFINITIONS ET ACRONYMES....167

REFERENCES BIBLIOGRAPHIQUES....191

REFERENCES BIBLIOGRAPHIQUES PERSONNELLES....217

ANNEXES

SOMMAIRE....A III

LISTE DES FIGURES....A VII

ANNEXE 1 : OUTILS POUR MODELISER LES ONTOLOGIES : MECANO, PROTEGE II ET PROTEGE/WIN..... A 1

ANNEXE 2 : DES EXEMPLES D'OUTILS META-CASE..... A 15

ANNEXE 3 : QUELQUES EXEMPLES DE SYSTEMES ET LANGAGES BASES SUR LES PARADIGMES « OBJET » ET « LOGIQUE »..... A 25

ANNEXE 4 : QUELQUES LOGICIELS DE PROGRAMMATION ETUDIES POUR NOTRE PROTOTYPE ..... A 39

ANNEXE 5 : LOGICIELS UTILISES POUR NOTRE PROTOTYPE..... A 49

ANNEXE 6 : LES AGENTS ..... A 59

Liste des Figures

Figure 1 : Chaînage notionnel [Develay, 1993] 19
Figure 2 : Double chaînage parallèle [Develay, 1993]....20
Figure 3 : Réseau notionnel [Develay, 1993]....20
Figure 4 : Double réseau, notionnel et méthodologique [Develay, 1993]....20
Figure 5 : Transdisciplinarité de l'apprentissage [Develay, 1993]....21
Figure 6 : Situation d'apprentissage....22
Figure 7 : Situation d'apprentissage....25
Figure 8 : Activité d'enseignement [Develay, 1993] 26
Figure 9 : Architecture d'un système tuteur intelligent [Nicaud et Vivet, 1988]....32
Figure 10 : Les trois niveaux de KOD [Vogel, 1988] 41
Figure 11 : Des données aux informations (d'après [Prince, 1996])....63
Figure 12 : Architecture des événements d'apprentissage au sein du « Campus Virtuel »....81
Figure 13 : Les trois étapes du travail du pédagogue 83
Figure 14 : Cycle de vie « évolutif » [Zamperoni et Gerritsen, 1994] 93
Figure 15 : Cycle de prototypage selon [Luqi, 1989]....94
Figure 16 : Les modèles de base du schéma OSM [Bézivin, 1995]....95
Figure 17 : Prototypes et Modèle logiciels [Krief, 1992]....97
Figure 18 : Cycles de vie des logiciels éducatifs [Nodenot, 1992] 104
Figure 19 : Notre cycle de vie des logiciels éducatifs 105
Figure 20 : Les trois dimensions de l'ingénierie des besoins [Pohl, 1994] 108
Figure 21 : Modélisation de l'activité d'enseignement 116
Figure 22 : Représentation conceptuelle du « Contexte général » 122
Figure 23 : Une définition du contexte de GLOM 122
Figure 24 : Représentation conceptuelle de la « Matrice Disciplinaire »....124
Figure 25 : Variante de la représentation conceptuelle de la « Matrice Disciplinaire »....125
Figure 26 : Matrice disciplinaire de GLOM....126
Figure 27 : Représentation conceptuelle du « Focus »....126
Figure 28 : Variante de la représentation conceptuelle du « Focus »....126
Figure 29 : « Focus » sur la matrice disciplinaire de GLOM 127
Figure 30 : Représentation conceptuelle des « Objectifs d'apprentissage » et des « Stratégies d'enseignement »....128
Figure 31 : Des objectifs d'apprentissage de GLOM....128
Figure 32 : Une stratégie d'enseignement de GLOM 128

Figure 33 : Des objectifs d'apprentissage de l'Anglais de spécialité 129
Figure 34 : Une stratégie d'enseignement de l'Anglais de spécialité.....129
Figure 35 : Représentation conceptuelle des « Situations d'enseignement » 130
Figure 36 : Situations d'enseignement de GLOM.....131
Figure 37 : Situations d'enseignement de l'Anglais de spécialité.....131
Figure 38 : Modélisation conceptuelle : vue d'ensemble 131
Figure 39 : Base de données pour la sauvegarde et la réutilisation 140
Figure 40 : Base de données pour la sauvegarde et la réutilisation de la matrice disciplinaire.....141
Figure 41 : Base de données pour la gestion des utilisateurs 141
Figure 42 : Schéma MVC.....142
Figure 43 : Trilogie M.PV.C. [Krief, 1992] 143
Figure 44 : Principales composantes de l'environnement de spécification. 147
Figure 45 : Ecran d'utilisation de HARDY 156
Figure 46 : Fonctionnement de notre environnement.....158
Figure 47 : Du rôle de l'informatique... 163

Liste des Tableaux

Tableau 1 : Le travail de l'enseignant 13
Tableau 2 : Taxonomie des stratégies pour la production de spécifications formelles 57
Tableau 3 : Caractérisation de l'activité d'enseignement.....116

Chapitre 1 : Introduction

« *Il n'est pas nécessaire d'espérer pour entreprendre ni de réussir pour persévérer.* »
Guillaume d'Orange-Nassau

Le cadre de notre travail se situe à la croisée de l'informatique et de l'enseignement. Ce champ est relativement vaste dans la mesure où ces termes recouvrent de nombreux domaines et notamment le génie logiciel, l'intelligence artificielle, la programmation, les bases de données, les interfaces homme-machine, le multimédia, les systèmes d'information, les sciences de l'éducation, la pédagogie, la didactique, etc…
En outre, un autre défi doit également être relevé dans la mesure où ces domaines font appel à de multiples compétences également inhérentes à de nombreux acteurs : pédagogues, instructeurs, apprenants, didacticiens, cogniticiens, analystes, programmeurs, spécialistes en interface, etc…
Par conséquent, il est primordial d'harmoniser ces connaissances et compétences diverses, que nous pouvons regrouper au sein de l'acronyme générique EIAO (Environnement Interactif d'Apprentissage avec Ordinateur).

Ainsi, Alain BOUVIER, chercheur en mathématiques, Professeur des Universités, spécialiste de la formation pour adultes et directeur d'IUFM, tenait en décembre 1996 les propos suivants dans la revue « *MAIF Infos* » : « *La mission des maîtres évolue, en France comme à l'étranger. Les parents, la société attendent aujourd'hui beaucoup plus de l'école primaire que jadis. Les disciplines traditionnelles demeurent, mais s'y ajoutent désormais des éléments originaux : l'apprentissage d'une langue étrangère, l'approche de nouvelles technologies… Par ailleurs, l'école n'a plus le monopole de l'enseignement du savoir. Les enfants apprennent autrement… par Internet par exemple. Au sein d'une même classe, les connaissances diffèrent et sont parfois, dans certains domaines, supérieures à celles de l'enseignant.*
Globalement, le système éducatif s'est adapté à l'évolution. La nouvelle génération des maîtres bénéficie d'une formation universitaire plus longue et plus ouverte sur l'extérieur. Le contenu et les méthodes de formation continuent eux aussi à changer. ».

Ainsi, nous avons choisi de focaliser nos recherches sur la mise en place d'outils et d'une méthode de spécification des connaissances au sein d'un atelier de génie éducatif (AGE). Dans ce cadre-là, le pédagogue doit alors être en mesure de spécifier lui-même les enseignements qu'il souhaite dispenser.

Ceci permet notamment d'établir une adéquation maximale d'une part entre les souhaits et besoins qu'il exprime et d'autre part la spécification résultante.
Nous fixons alors dans cette thèse les principes et outils nécessaires pour obtenir une telle analogie. Ainsi, nous avons défini un processus de spécification particulier issu des travaux en sciences de l'éducation, et centré autour du concept de situation-problème tel qu'il est préconisé à l'heure actuelle au sein des IUFM. Ce processus s'appuie sur un environnement informatique supporté par des outils méta-CASE favorisant la réutilisation des spécifications existantes tout en offrant des possibilités nécessaires de guidage à l'usager et de souplesse de manipulation.

Ce document est organisé en trois parties [3].

La première partie est une étude de l'existant en ce qui concerne la construction des enseignements telle qu'elle peut être préconisée en France, tant d'un point de vue éducatif ou pédagogique que de par le rôle que peut ou pourrait y jouer l'informatique.
L'objectif du chapitre 2 consiste à présenter comment ce processus de conception est abordé aujourd'hui de façon générale, indépendamment de l'apport de l'informatique. L'importance d'une prise en compte de la transdisciplinarité est également mise en évidence.
Le chapitre 3 se focalise sur la place occupée par l'informatique dans l'enseignement au travers de la notion de génie éducatif. L'acquisition puis la gestion des connaissances est alors une préoccupation majeure. Nous présentons ici les principales techniques et outils d'élicitation des connaissances que sont KOD, KADS et CommonKADS.
Nous présentons ensuite, dans le chapitre 4, l'importance des premières phases dans le processus de construction des enseignements. Le paragraphe 1 souligne ainsi l'importance de l'ingénierie des besoins et le paragraphe suivant insiste sur la nécessité de spécifier les connaissances de façon formelle.

En nous fondant sur ces bases, un dialogue a été instauré entre informaticiens et pédagogues afin de mettre en évidence ce qui constitue le cahier des charges de nos travaux de recherche (chapitre 5).
La seconde partie de ce document est alors consacrée à la recherche de solutions relatives aux principes et processus d'acquisition des connaissances éducatives par un système informatique dans un objectif de traitement automatique.
Dans le chapitre 6, nous abordons le processus de spécification des connaissances au sein d'un environnement éducatif. Après avoir donné dans le paragraphe 1 des exemples de systèmes existants

[3] *Note au lecteur :* La structure du document, du niveau le plus général au niveau le plus détaillé, est la suivante. Chaque partie majeure est divisée en chapitres composés de paragraphes principaux découpés eux mêmes en sous-paragraphes et ainsi de suite. Alors que la numérotation des parties et chapitres suit une progression arithmétique, la numérotation des paragraphes principaux est réinitialisée au sein de chaque chapitre.

prenant en compte cette phase, nous proposons dans le paragraphe 2 notre propre processus de spécification.
Dans un but d'intégrer ce dernier dans un environnement informatique, nous préconisons alors dans le chapitre 7 le rôle du prototypage qui permet une représentation précoce des besoins réellement exprimés et pour lequel l'utilisation d'outils « CASE » et « méta-CASE » devient ici primordiale.

La troisième partie a donc pour objectif de décrire la structure conceptuelle et profonde que nous avons choisie pour notre environnement d'élicitation, de représentation et de manipulation des connaissances.
Tout d'abord, nous décrivons dans le chapitre 8 l'organisation structurelle des connaissances que nous préconisons. Ainsi le paragraphe 1 montre que les connaissances peuvent être représentées sous forme ontologique. Le paragraphe 2 décrit la structure conceptuelle de chaque composant de notre environnement. Ceci est non seulement abordé d'un point de vue général, c'est-à-dire par rapport à la modélisation de la structure profonde, mais aussi d'un point de vue pratique car chaque modélisation s'appuie sur un exemple concret d'utilisation.
Ceci permet alors de mettre en avant les fonctions de guidage et de réutilisation ainsi que l'importance de la flexibilité d'un environnement de spécification que nous retrouvons au travers des outils que nous préconisons et que nous présentons dans le chapitre 9. Le paragraphe 1 considère ces trois fonctionnalités majeures et le paragraphe 2 est consacré à la définition de l'interface-utilisateur.
Un point de vue pratique relatif à la réalisation d'un prototype supportant les principes évoqués tout au long de ce document constitue le chapitre 10. Ainsi, nous décrivons quelques outils logiciels que nous avons utilisés pour la programmation ainsi que les limitations que nous nous sommes fixées.

Après avoir dressé les conclusions et perspectives de nos travaux de recherches (chapitre 11), nous fournissons un glossaire des principaux termes et acronymes utilisés puis nous exposons les références bibliographiques que nous avons utilisées.

Les annexes de ce document sont scindées en six parties.
L'annexe 1 est une présentation de trois projets et outils existants pour représenter des connaissances de façon ontologique : MECANO, Protégé II et Protégé/Win.
L'annexe 2 est constituée par une exposition de plusieurs outils « méta-CASE » commerciaux et non-commerciaux que nous avons étudiés.
L'annexe 3 est un recueil de systèmes et langages basés sur les notions d'objet actif, d'acteur et d'agent.
L'annexe 4 est un aperçu de plusieurs logiciels de programmation basés sur les paradigmes objet et logique : CLIPS, ART-IM, KES, Level 5, Vax OPS5, OPS5, ARGOS-II et SNARK.

Ensuite, les différents logiciels que nous avons utilisés pour la programmation de notre prototype (wxWindows, wxCLIPS, HARDY, Tex2RTF et RTFHTM) sont exposés en annexe 5.
Enfin, l'annexe 6 présente la notion d'agents et de systèmes multi-agents ainsi que divers exemples de programmes utilisant notamment des agents logiciels sur Internet.

Première Partie :
Vers un environnement de spécification d'enseignements

« Penser ne sert à rien ; il faut penser à quelque chose. »
Jules Renard

Résumé de la première partie

Dans cette première partie, nous avons pour ambition principale de présenter le cadre général de nos recherches. C'est ce que l'on peut qualifier, dans un processus de développement, de phase préliminaire qui rend compte des raisons qui ont motivé ces travaux. Nous répondons à la question suivante : « Pourquoi un environnement de spécification des connaissances relatives à l'enseignement ? ».

Ainsi, le chapitre 2 est consacré à l'enseignement en général en considérant également l'apport des nouvelles technologies qui incite à une remise en question menant à une véritable technologie éducative. Nous démontrons également dans quelles mesures une approche de l'enseignement basée sur la notion de situation-problème permet une réelle trandisciplinarité ainsi qu'un véritable transfert de la situation d'apprentissage vers une situation d'enseignement.

Le chapitre 3 situe ensuite la place de l'informatique dans ce qui est appelé l'EAO (Enseignement Assisté par Ordinateur) et même l'EIAO, acronyme désignant au départ l'Enseignement Intelligemment Assisté par Ordinateur et qui, compte tenu de l'évolution propre à l'ensemble de ces composants désigne maintenant le terme Environnement Interactif d'Apprentissage avec Ordinateur. Nous décrivons alors ses principales composantes ainsi que différents techniques et outils facilitant l'acquisition et la gestion des connaissances.

Le chapitre 4 recentre la place du processus de spécification des connaissances au sein du cycle de construction d'enseignement. Nous présentons les raisons pour lesquelles les connaissances pédagogiques doivent être spécifiées de façon formelle. Ceci nous amène alors à proposer un cycle de

vie relatif à la construction d'enseignements basé sur une spécification formelle et favorisant le prototypage rapide et la réingénierie des connaissances.

Plan de la première partie

CHAPITRE 2 : VERS UNE NOUVELLE CONCEPTION DE L'ENSEIGNEMENT 9
Paragraphe 1 : Construction des enseignements *9*
1. Introduction 9
1.1. La technologie éducative 10
1.2. Comment mettre en place une technologie éducative ? 11
2. Le travail du pédagogue 12
2.1. Avant la classe : anticiper 14
2.2. Pendant la classe : évaluer et évoluer 15
2.3. Après la classe : réévaluer 15
3. De la modélisation d'une discipline à la transdisciplinarité 17
3.1. Quelques définitions 17
3.1.1. Les connaissances déclaratives 17
3.1.2. Les connaissances procédurales 18
3.1.3. La matrice disciplinaire 18
3.2. Vers un enseignement transdisciplinaire 19
Paragraphe 2 : De la situation d'apprentissage à la situation d'enseignement *22*
1. Définition d'une situation d'enseignement 24
2. La métacognition 27
2.1. Définition 27
2.2. Importance de la métacognition 28

CHAPITRE 3 : DU ROLE DE L'INFORMATIQUE DANS L'ENSEIGNEMENT 29
Paragraphe 1 : Le génie éducatif *29*
1. Présentation générale 29
2. Les Systèmes Tuteurs Intelligents 32
3. Vers des EIAO centrés sur l'utilisateur 34
Paragraphe 2 : Gestion et acquisition des connaissances *36*
1. Trois principales approches 36
2. Techniques d'acquisition des connaissances 37
3. Méthodes d'acquisition des connaissances 41
3.1. KOD 41
3.2. KADS 42
3.3. CommonKADS 44

CHAPITRE 4 : LA SPECIFICATION DES ENSEIGNEMENTS 47
Paragraphe 1 : Importance de l'ingénierie des besoins *48*
1. Qu'est ce qu'un besoin ? 49
2. Les activités de l'ingénierie des besoins 49

Paragraphe 2 : Vers une spécification formelle des connaissances *50*
1. Objectifs 51
1.1. Un premier objectif : la communication 51
1.2. Un second objectif : une bonne conception 52
1.3. Objectifs connexes 53
2. Classification 54
3. Apports du génie logiciel 58
3.1. Quelques définitions 59
3.2. Les outils de prototypage 60
3.2.1. Les générateurs d'applications 61
3.2.2. Les boîtes à outils 62
4. Conclusion 63

Chapitre 2 : Vers une nouvelle conception de l'enseignement

L'objectif de ce chapitre est de répondre à la question suivante : « Quelle est la place de l'information au sein du monde de l'enseignement tel qu'il est conçu ou perçu à l'heure actuelle ? »
Pour ce faire, nous allons tout d'abord présenter dans le premier paragraphe comment l'enseignement est abordé aujourd'hui en présentant l'approche préconisée en France pour la formation des enseignants dans les Instituts Universitaires de Formation des Maîtres (IUFM) et, notamment, autour de la mise en œuvre de situations-problèmes.
Dans le second paragraphe de ce chapitre, nous recentrons notre étude sur le rôle joué par l'informatique en général, ses possibilités et ses outils, au sein du domaine de l'éducation. Ainsi, nous commençons par présenter le rôle joué par les notions inhérentes à l'intelligence artificielle qui a abouti premièrement à la création de systèmes tuteurs intelligents (STI), puis à la mise en place de véritables ateliers de génie éducatif (AGE).

Paragraphe 1 : Construction des enseignements

Au sein de ce premier paragraphe, nous allons tout d'abord présenter dans quelle mesure il convient de reconsidérer le processus d'enseignement afin de mettre en place une véritable « technologie éducative ». Nous décrivons ensuite les trois étapes fondamentales dans le travail quotidien du pédagogue : à savoir la préparation, la régulation et l'évaluation. Ensuite, dans le troisième paragraphe, nous fournissons un ensemble de définitions qui sont à la base de nos travaux et qui ont pour objectif de mettre en place une réelle approche transdisciplinaire de l'enseignement. Ceci nous permet enfin, dans le quatrième paragraphe, d'aborder les notions de situation d'enseignement et de situation d'apprentissage en nous centrant sur une approche basée sur la notion de situation-problème.

1. Introduction

Apprentissage et enseignement, ces deux concepts, qui semblent identiques au premier abord, se révèlent pourtant très distincts.
Le mot apprentissage renvoie à une conjonction de plusieurs sémantiques : enseigner, former, éduquer, instruire ; ce qui rend délicat le positionnement entre ce qui est propre à l'apprenant et à l'enseignant.

De plus, deux difficultés supplémentaires empêchent de distinguer clairement l'enseignement et l'apprentissage :

- sur un plan méthodologique, nous pouvons situer le début et la fin d'un enseignement, mais c'est délicat pour l'apprentissage.
- il est tout aussi facile d'observer les résultats de l'apprentissage et donc de l'enseignement mais il est beaucoup plus difficile d'accéder aux processus de l'apprentissage.

La difficulté philosophique consiste à envisager l'élève et ses acquis, savoir et savoir-faire, au sein de la dimension universelle de l'apprentissage.

On a longtemps posé comme allant de soi que les technologies modernes de l'information et de la communication sont d'emblée par elles-mêmes « éducatives », confondant ainsi, un peu rapidement, innovation technologique et innovation pédagogique. [Mottet, 1983] montre que l'intégration de ces moyens nouveaux suppose tout un travail d'invention pédagogique et que les enseignants praticiens sont appelés aujourd'hui à dépasser leur rôle traditionnel de transmetteurs de connaissances pour devenir les technologues de leur propre pratique.

1.1. La technologie éducative

D'une façon générale la « *technologie éducative* » peut se définir comme l'intégration de la technologie à l'éducation. [Mottet, 1983] distingue trois significations principales qui correspondent non seulement à des approches différentes mais également à des niveaux différents d'intégration : la technologie dans l'éducation, la technologie de l'éducation et la technologie des processus d'apprentissage.

En un premier sens, la technologie éducative désigne les divers outils, procédés, documents et supports matériels dont se servent les enseignants et les élèves à des fins pédagogiques. C'est ce qu'on peut appeler la « *technologie dans l'éducation* ».
En un deuxième sens, la technologie éducative est l'étude des différentes façons d'agencer et de mobiliser l'ensemble des moyens dont dispose, à ses différents niveaux, un système éducatif pour réaliser au mieux les objectifs qu'il se fixe. C'est ce qu'on peut nommer, à proprement parler la « *technologie de l'éducation* », qui se propose d'analyser et d'améliorer le fonctionnement global d'un système d'enseignement ou de formation, notamment en recherchant l'utilisation optimale des ressources disponibles. La technologie éducative se définit ici précisément comme l'application d'une démarche technologique à l'éducation. Cette démarche, qui s'inspire de *l'approche systémique* [Le Moigne, 1990], n'est pas simplement descriptive ; elle ne se borne pas à dresser l'inventaire des ressources existantes, elle est également prescriptive, voire prospective. En effet, dans la technologie

éducative, tout nouveau moyen technique implique la réorganisation du système dans lequel il s'insère, quel que soit le niveau où l'on se place. Il ne s'agit plus ici de partir seulement d'un matériel nouveau à introduire dans un cadre de fonctionnement qui n'est pas lui-même remis en cause. Il s'agit plutôt de fixer un résultat à atteindre par rapport auquel les différents moyens dont on dispose seront combinés.

En un troisième sens, la technologie éducative est une certaine conception de l'action pédagogique. C'est la *technologie des processus d'apprentissage*. Il faut en effet concevoir que l'éducation elle-même est une technologie, et que l'enseignant et-ou le formateur sont des technologues de l'apprentissage, des constructeurs de situations éducatives.

Ainsi, pour s'approprier, à des fins éducatives, le fonctionnement d'un outil, il faut savoir quelle fonction éducative il est susceptible de remplir, d'où la nécessité de savoir *analyser* et *formaliser* ce qu'on lui fait traiter [Balacheff *et al.*, 1993].

L'usage de l'ordinateur requiert la formalisation des processus d'enseignement et d'apprentissage qu'on veut lui faire gérer. La rénovation de la pédagogie est donc le passage obligé de l'innovation technologique. C'est seulement par l'introduction de la rigueur et de la rationalité dans le travail pédagogique que l'on peut espérer y intégrer peu à peu les apports des technologies modernes de l'information et de la communication.

Les conditions réelles d'intégration de la technologie à l'éducation ne se trouvent donc pas dans le degré de raffinement technologique des outils proposés mais bien dans la conception de nouvelles pratiques qui donnent toute leur place aux outils qu'on y introduit.

1.2. Comment mettre en place une technologie éducative ?

Pour répondre à cette question, il est nécessaire de considérer trois points principaux.

Penser la pédagogie en termes de technologie signifie en premier lieu un *déplacement de la situation d'enseignement à la situation d'apprentissage*. Il faut définir l'acte pédagogique du point de vue du résultat à atteindre, c'est-à-dire du point de vue de l'élève qui apprend et non du point de vue de l'enseignant, dont le rôle se limite à construire les conditions de réussite des élèves. Par ailleurs, les élèves constituent eux-mêmes des sources d'interaction réciproque favorisant l'apprentissage de la communication sociale ; ils représentent les uns pour les autres des ressources mutuelles. La pédagogie de groupe apparaît comme un moyen essentiel qui a sa place dans l'ensemble collectif des ressources éducatives.

Ainsi, l'une des lignes de développement de la technologie éducative aboutit à cette idée de constituer au sein de l'école des « systèmes multimédia d'auto-apprentissage », c'est-à-dire des systèmes ouverts et coordonnés de ressources éducatives multiples qui incluraient non seulement les enseignants avec leurs compétences spécifiques, non seulement les élèves eux-mêmes avec leur expérience et leur

personnalité propre, mais aussi les supports d'enseignement les plus divers, les mémoires documentaires et les machines.

Ensuite, le pédagogue, en tant que concepteur d'une action éducative, doit être à même d'en *évaluer* les résultats. Les démarches et procédures qu'il met en œuvre sont autant d'hypothèses qu'il faut soumettre à la sanction du réel. Ce n'est pas seulement au terme de son travail qu'il lui faut mesurer le chemin parcouru par les élèves, mais aussi dans le déroulement même de l'action de manière à en contrôler constamment la progression et à y apporter les modifications nécessaires. Le pédagogue doit s'efforcer de conduire des dispositifs dans lesquels les élèves auront eux-mêmes un rôle d'auto-régulation à jouer par rapport à leurs propres apprentissages, des dispositifs permettant de prendre du recul et d'analyser l'activité dans laquelle on se trouve soi-même impliqué. L'élève doit pouvoir, lui aussi, avoir une attitude expérimentale par rapport à sa propre action, d'où l'importance de la *métacognition*.

Enfin, la technologie éducative introduit l'idée d'une pédagogie communicable, d'une pédagogie susceptible d'être mise en commun, échangée, partagée. Ceci a pour conséquence de favoriser un travail interdisciplinaire et même *transdisciplinaire*.

La technologie éducative ne consiste donc pas à offrir aux enseignants une simple boîte à outils. Elle est une exigence nouvelle par rapport à l'éducation, un ensemble de principes pour la conduite pédagogique, un ensemble de questions polémiques qui obligent à redéfinir constamment ce que l'on fait quand on enseigne. La première de ses interpellations est d'obliger les enseignants à s'expliquer, c'est-à-dire à cerner et définir clairement leurs objectifs, à formaliser leurs démarches, à évaluer leurs résultats. Elle est une exigence de rigueur et d'explicitation qui appelle les enseignants à un redoublement réflexif sur leurs propres pratiques.

Nous avons considéré les critères à mettre en place pour une technologie éducative. Nous allons maintenant dresser un état des lieux du travail du pédagogue dans son quotidien.

2. Le travail du pédagogue

Comme nous l'avons vu dans [Lecomte, 1995], l'acte créatif comporte quatre étapes :

- La *préparation*, c'est-à-dire l'analyse préliminaire du problème ainsi que le recueil d'informations et-ou de matériel.
- L'*incubation*, pendant laquelle l'activité est essentiellement inconsciente. Le problème « mûrit » dans l'esprit du créateur.

- L'*inspiration*, qui constitue évidemment l'acte majeur du processus. C'est le fameux « *Eureka !* » d'Archimède.
- L'*évaluation* est la phase de vérification pendant laquelle la personne teste la solution ou examine la valeur du produit. Cela implique la faculté de se remettre en cause.

Nous pouvons transposer ceci aux différents rôles du pédagogue en situation d'enseignement. Ces activités sont multiples [Pierre *et al.*, 1992] :

- organiser et enrichir le milieu.
- observer les activités et analyser les situations. Ceci consiste alors à reconnaître les situations « mathématisables » à l'intérieur d'une activité pluridisciplinaire et à faire un choix parmi les notions pouvant être dégagées.
- évaluer le niveau de développement des enfants, c'est-à-dire observer l'enfant dans une activité pour déceler ses possibilités d'action et de raisonnement, les stratégies qu'il met en œuvre, puis adapter les situations aux besoins et aux possibilités.
- définir des objectifs éducatifs et susciter des activités intégrant ces objectifs. Il convient alors de proposer des activités interdisciplinaires s'appuyant sur un vécu et une activité réelle de l'enfant.
- contrôler et apprécier les acquis afin de pouvoir adapter de nouvelles procédures, mettre en place des actions de « réajustement » et offrir de nouvelles motivations.

Le travail du maître se déroule donc en trois temps en interaction : avant, pendant et après la classe. Toutefois, l'émergence des nouvelles technologies recentre les tâches de l'enseignant traditionnelles (préparation, régulation et évaluation) selon les trois pôles correspondant (anticipation, évaluation et réévaluation). D'où le tableau suivant :

	Conception traditionnelle	**Apport des nouvelles technologies**
Avant la classe	Préparer	Anticiper
Pendant la classe	Réguler	Evaluer et évoluer
Après la classe	Evaluer	Réévaluer

Tableau 1 : Le travail de l'enseignant

2.1. Avant la classe : anticiper

Cette phase consiste à prendre en compte l'observation des comportements, des intérêts des enfants, et à tirer parti de l'expérience, des représentations, des connaissances et du savoir-faire acquis. Cela permet alors d'anticiper sur les problèmes que vont rencontrer les apprenants. Il convient par conséquent pour le maître de déterminer ce qu'il est plausible d'espérer en précisant ses intentions puis en ajustant sa démarche.

Pour chaque étape de la scolarité (cycle, année, trimestre, mois, semaine et jour), le maître se charge de définir différents programmes d'activités afin de respecter une progression qu'il se fixe. Ainsi, il est important de :

- déterminer les objectifs particuliers en relation avec l'objectif général, c'est-à-dire prévoir les capacités que l'on veut faire acquérir en fonction de la compétence à construire (ne pas travailler dans l'éphémère) ;
- déterminer les domaines de son travail (activités physiques, activités de communication et d'expression orales ou écrites, activités esthétiques, activités scientifiques et techniques) ;
- articuler les objectifs disciplinaires à l'objectif général pour permettre la mobilisation, la construction des différents modes de représentation, l'appropriation de différents langages et outils de la culture.

Pour assurer la cohésion de son travail, le pédagogue doit prévoir l'organisation matérielle du milieu et des situations afin d'obtenir, de la part de l'élève, l'engagement dans l'activité, l'identification de la tâche et la compréhension de sa portée.
Le maître doit donc mettre à la disposition des enfants des moyens de découvrir, d'échanger, de réinvestir. Il doit alors prévoir :

- la durée et l'alternance des activités,
- les formes de travail (collectif, individuel ou de groupes).
- les procédures d'évaluation adaptées à l'âge de l'apprenant et à la capacité d'appréciation objective de ses propres actes afin de tirer un parti positif des évaluations du maître et des évaluations ébauchées parmi les camarades.

En résumé, durant cette phase préliminaire d'anticipation, il s'agit pour le maître d'élaborer un projet qui implique de :

- bien poser le *problème*,
- bien *définir les objectifs* afin de savoir ce que l'on cherche et pourquoi,

- prévoir les *activités* en terme de *tâches*, *ressources, outils*, *techniques* et *matériaux* nécessaires, introduits à bon escient et suffisamment riches sans être pléthoriques.
- aménager le *temps* : programmation et organisation (planification) de la séance,
- donner des *consignes* ou *contraintes,*
- émettre des *hypothèses de solutions*.

2.2. Pendant la classe : évaluer et évoluer

Le rôle et l'attitude du maître sont multiples durant cette phase car il doit être en mesure de proposer les enseignements préparés tout en veillant à une remise en cause permanente en fonction de l'évolution des apprenants face aux situations anticipées. Il se retrouve donc en situation perpétuelle de gestion dynamique des connaissances.

Tout d'abord, il doit veiller à la qualité de sa présence, de sa voix et de son élocution. Durant cette évaluation, il convient non seulement de laisser à l'apprenant la plus grande initiative possible (respecter la recherche), mais aussi de lui donner le temps de tâtonner, d'expérimenter, et d'ébaucher des procédures de résolution. Pendant ce temps, le pédagogue doit également observer les essais, les erreurs, et les variations personnelles.

Ensuite, une évaluation de l'ensemble de ces actions permet au maître d'identifier les acquis et besoins individuels. Le pédagogue doit alors organiser le travail de façon à favoriser les interactions et inciter l'apprenant. Ceci consiste à lui offrir de multiples sollicitations pour entreprendre, fournir un effort efficace, tirer parti de ses échecs et de ses réussites afin de mener l'activité à son terme et de résoudre les problèmes posés au départ et-ou découvert durant l'enseignement.
Pour ce, le maître aide en :

- donnant une *consigne* au bon moment et en s'assurant de sa compréhension,
- rappelant des *acquis* antérieurs,
- suggérant des *situations analogues* déjà rencontrées montrant parfois (ni prématurément, ni complètement, ni systématiquement) comment s'y prendre,
- *évaluant* les résultats des activités avec les enfants.

2.3. Après la classe : réévaluer

Dans un troisième temps, le pédagogue doit s'interroger : « L'objectif visé a-t-il été atteint ? par qui ? dans quelles conditions ? Les projets pédagogiques et d'activités étaient-ils bien adaptés ? ».

Cette réévaluation s'effectue ici à deux niveaux relatifs aux acquis antérieurs des apprenants et aux résultats obtenus suite à l'action pédagogique.

Pour parvenir à une juste appréciation de l'état réel du développement de chaque enfant, le pédagogue doit à la fois *évaluer ses attitudes et ses acquis*.
En ce qui concerne l'évaluation des attitudes, il convient de prendre en compte les observations faites pendant le déroulement des activités :

- l'attitude de l'enfant face aux données de la situation, c'est-à-dire son engagement dans la tâche (en qualité et en durée) et sa façon de s'organiser pour résoudre la situation.
- la nature des difficultés rencontrées et des erreurs résolues.
- l'attitude face à ces difficultés, face à l'aide qui lui a été apportée et l'assistance demandée.
- l'attitude face aux évaluations des camarades.

Pour l'évaluation des acquis, le pédagogue se charge d' :

- examiner la production de l'enfant et de la situer par rapport aux travaux antérieurs,
- apprécier en quoi elle témoigne de la consolidation de certaines capacités et en quoi la performance attendue a été ou non atteinte (noter les progrès, les stagnations, etc…),
- vérifier que des situations permettant de s'approprier les compétences nécessaires pour construire la nouvelle capacité ont été offertes.

Il convient aussi, pour le pédagogue, d'examiner les variables de la situation mise en place et de remettre éventuellement en question l'activité, le moment de la journée et-ou le dispositif de la situation (proximité avec les intérêts des enfants, supports offerts à l'activité).
La *réévaluation* permet donc d'avoir des retours d'informations (mécanisme de *rétroaction*) et, le cas échéant, de modifier la *stratégie* [4] utilisée. En effet, le pédagogue peut prévoir, en fonction des informations ainsi recueillies :

- soit d'intervenir sur une ou des variables de la situation,
- soit des situations voisines pour consolider les acquis,
- soit des situations de transfert,
- soit des situations de remédiation pour aider à l'acquisition (réajustement de l'objectif).

Nous avons vu quels sont les rôles du pédagogue dans son travail d'aide à l'acquisition de connaissances par les apprenants. Nous allons maintenant étudier dans quelles mesures une approche transdisciplinaire de l'enseignement permet de faciliter cette tâche.

4 Rappelons qu'une stratégie est une manière dont l'enseignant ou les élèves vont agencer les ressources, utiliser les contraintes ou y résister. Donc, en définitive, la manière d'organiser les différents composants définit la stratégie pédagogique.

3. De la modélisation d'une discipline à la transdisciplinarité

Chaque enseignant doit être tout autant un didacticien qu'un pédagogue. Il lui importe donc de clarifier l'épistémologie de sa discipline, afin de mettre en cohérence objets d'enseignement, méthodes et techniques d'enseignement, théories de référence.

Un *modèle commun* à partir duquel il serait possible, pour les enseignants, de comparer leurs disciplines, est à proposer. C'est ce que [Develay, 1993] tente de faire, en suggérant qu'une discipline scolaire soit définie par des *objets* qui lui sont spécifiques, des *tâches* qu'elle permet d'effectuer, des savoirs *déclaratifs* dont elle vise l'appropriation, des savoirs *procéduraux* dont elle réclame aussi la maîtrise, enfin une *matrice* qui la constitue en tant qu'unité épistémologique, intégrant les quatre éléments précédents et lui donnant sa cohérence.

3.1. Quelques définitions

3.1.1. Les connaissances déclaratives

Les *connaissances déclaratives*, terme emprunté à la psychologie cognitive, sont de l'ordre du discours, du savoir alors que les *connaissances procédurales* sont de l'ordre de l'action, du savoir-faire.

Notons que le passage du déclaratif au procédural, ou inversement est une question importante pour comprendre les difficultés d'apprentissage des élèves.

Quatre types de *connaissances déclaratives* peuvent aider à caractériser une discipline :

- la distinction entre *fait* et *notion*

A un certain niveau, une notion est explicative d'un ensemble de faits. Pour être assimilée, une notion doit s'ancrer sur un ensemble de faits dont elle cherche la cohérence. Ainsi, ce sont deux opérations d'abstraction étroitement liées. On ne peut pas enseigner de notions (exemple : la respiration) indépendamment de faits (exemple : le poisson respire par les branchies, l'homme par les poumons). Et un fait ne prend de sens que par rapport à la notion qui l'englobe.

De plus, pour un niveau d'enseignement donné, il faut distinguer les notions au pouvoir explicatif le plus englobant : les concepts intégrateurs (exemple : l'organisme).

Identifier les concepts intégrateurs pour un niveau d'enseignement donné permettrait de déceler la structure de la discipline à cette étape de la scolarité.

- les *champs notionnels* ou *trames notionnelles*

Ce sont l'ensemble des notions qui, mises en synergie, donnent un sens à celle que l'on souhaite enseigner. Ainsi, si un élève n'a pas assimilé une notion, il n'a peut-être pas assimilé un élément constitutif de la trame de cette notion.

- *les registres de conceptualisation*

Ceux-ci permettent de formuler différemment les mêmes concepts en s'adaptant au niveau des élèves. Ceci permet également aux enseignants de mieux identifier les niveaux d'exigence et de mettre en place des stratégies didactiques appropriées.

- *les concepts intégrateurs*

Les concepts intégrateurs, à un niveau d'enseignement donné, organisent en une structure cohérente l'ensemble des faits et des notions abordés. Ils constituent en quelque sorte les principes organisateurs, au niveau notionnel, d'une discipline enseignée. Leur identification pour une discipline et pour un niveau d'enseignement donnés met en cohérence l'ensemble des connaissances à enseigner autour du « noyau dur » de la discipline.

3.1.2. Les connaissances procédurales

Elles sont constituées par une suite organisée d'actions (méthodes, techniques, procédures, stratégies) permettant d'atteindre le but poursuivi.
C'est un ensemble de méthodes et de techniques permettant la création des concepts.
Il arrive que certaines stratégies aient déjà été utilisées par le sujet, qui a ainsi identifié un *algorithme* de résolution. Mais, à d'autres moments, le sujet doit inventer une stratégie : une *heuristique* qui pourra éventuellement devenir un algorithme. Dans la résolution de certains problèmes, le sujet doit effectuer des tâches composites : utiliser des algorithmes qu'il maîtrise et inventer des heuristiques.
Par conséquent, réaliser une tâche, c'est manipuler des objets de départ et, grâce à un ensemble de connaissances procédurales et des compétences déclaratives (capacités), obtenir d'autres objets.
La notion de connaissance procédurale peut être analysée en terme de *capacités* qui sont du type : anticiper, appliquer, classer, se décentrer, s'adapter, déduire, faire des analogies, induire, dialectiser, conjecturer, gérer un risque, planifier, émettre une hypothèse, etc… Les capacités présentent un caractère transversal car les compétences procédurales à développer dans une discipline peuvent être regardées, comme les connaissances déclaratives, en termes de *registres de formulation*.

3.1.3. La matrice disciplinaire

Des liens étroits unissent les notions de matrice disciplinaire, de connaissance procédurale et de concepts intégrateurs.
La matrice disciplinaire représente le principe d'intelligibilité d'une discipline donnée c'est-à-dire son cadre de référence. Le sens métaphorique de matrice (nom commun de l'utérus) renvoie à l'image de moule, de creuset qui constituerait le fondement de la discipline, son essence.

Une matrice disciplinaire est constituée par le point de vue qui, à un moment donné, est porté sur un contenu disciplinaire et en permet la mise en cohérence. Ce point de vue est constitué par le choix d'une identité pour la discipline considérée. Il entraîne à privilégier, de fait, certains concepts, certaines méthodes, techniques, théories, valeurs, et amène en dernier ressort à valoriser certains objets d'enseignement.
Nous remarquons donc que les matrices disciplinaires ne sont pas les mêmes aux différents niveaux de la scolarité.

3.2. Vers un enseignement transdisciplinaire

Un professeur n'enseigne plus aujourd'hui une discipline, mais plusieurs. Il serait donc plus juste de parler de champs disciplinaires que de disciplines. En effet, il suffirait de citer des notions comme celles d'énergie, de système, de transformation, de structure, des connaissances procédurales telles que émettre une hypothèse, classer, etc…, pour se rendre compte que les mêmes concepts intégrateurs et les mêmes connaissances procédurales sont présents au sein de disciplines différentes, laissant alors percevoir une possible *interdisciplinarité*. L'interdisciplinarité est fréquemment abordée, par les enseignants de disciplines différentes, en termes d'activités à conduire en commun davantage qu'en termes de concepts ou de méthodes communes à différentes disciplines. Citons par exemple les PAE, acronyme de projets d'activités éducatives. En outre, l'interdisciplinarité peut mener à la *transdisciplinarité*, à condition de clarifier ce qui peut être commun, au niveau des concepts et des connaissances procédurales de différentes disciplines.

Selon [Develay, 1993], il existe cinq manières de concevoir une discipline d'enseignement : le chaînage notionnel, le double chaînage parallèle, le réseau notionnel, le double réseau notionnel et méthodologique et la transdisciplinarité.

Avec un chaînage notionnel, la discipline est conçue comme une succession de notions, en relation linéaire et chronologique les unes avec les autres. Tel est le cas par exemple pour les opérations mathématiques : addition, soustraction, multiplication puis division. Ici, la logique des contenus seule détermine la logique des apprentissages.

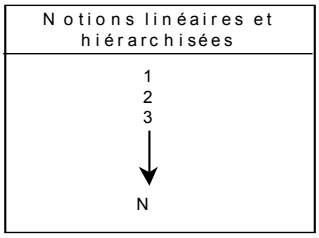

Figure 1 : Chaînage notionnel [Develay, 1993]

Avec un double chaînage parallèle, la discipline est conçue comme une succession de notions, linéaires et hiérarchisées comme précédemment, et aussi comme un ensemble de méthodes, plus difficilement hiérarchisables.

Cette conception renvoie généralement à deux temps d'enseignement : le temps de cours pour les notions et le temps de travaux pratiques pour les méthodes.

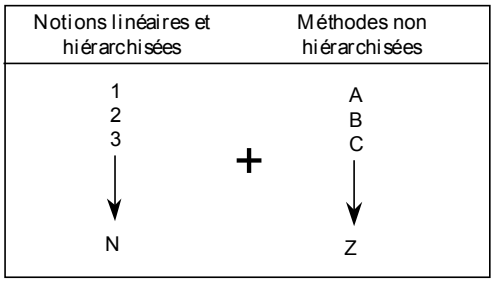

Figure 2 : Double chaînage parallèle [Develay, 1993]

Avec le réseau notionnel, la discipline est conçue comme un ensemble de notions en relation à l'intérieur d'un réseau. On retrouve ici la notion de trame notionnelle ou champ notionnel.

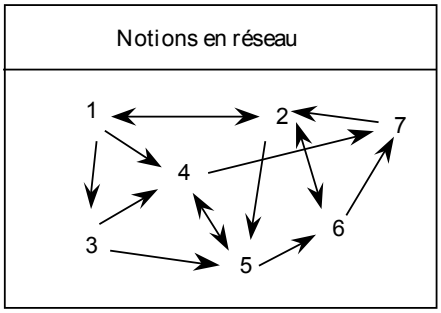

Figure 3 : Réseau notionnel [Develay, 1993]

Avec une organisation selon un double réseau, notionnel et méthodologique, aux notions reliées en réseaux, comme précédemment, s'ajoutent non pas des méthodes mais des connaissances procédurales disciplinaires qui permettent d'acquérir ces notions, en effectuant des *tâches* disciplinaires données. Pour envisager son enseignement, l'enseignement a ici d'abord à raisonner en terme de tâches disciplinaires.

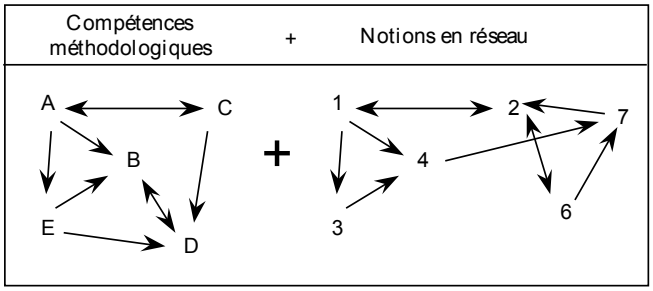

Figure 4 : Double réseau, notionnel et méthodologique [Develay, 1993]

Dans une conception transdisciplinaire, la discipline d'enseignement est envisagée comme précédemment, constituée de connaissances procédurales et de notions en réseaux. Mais, dans ce cas, on s'intéresse aux compétences notionnelles et méthodologiques qui pourraient être communes à plusieurs disciplines. La discipline considérée est resituée par rapport à d'autres disciplines, afin de pouvoir déterminer deux types de transdisciplinarité : la transdisciplinarité instrumentale qui s'intéresserait aux contenus notionnels et la transdisciplinarité comportementale qui s'intéresserait aux connaissances procédurales et aux attitudes. De plus, selon [Develay, 1993], seule cette dernière conception peut réellement fonder des pratiques interdisciplinaires parce que transdisciplinaires.

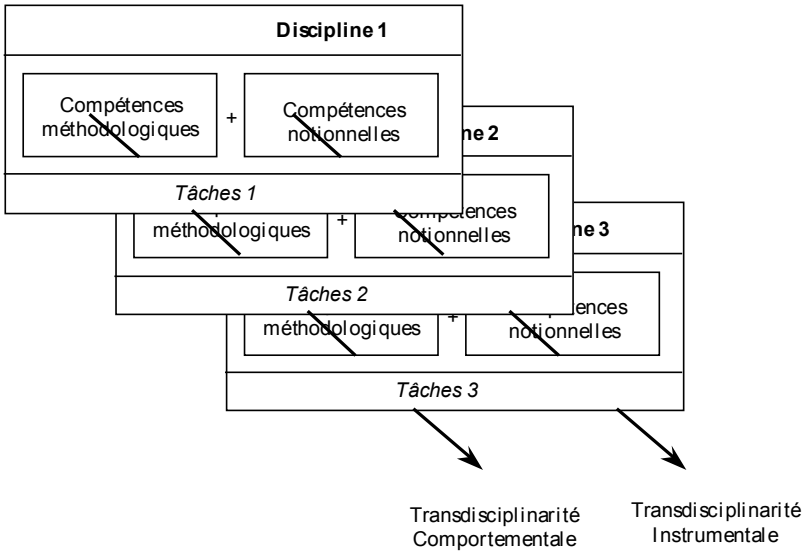

Figure 5 : Transdisciplinarité de l'apprentissage [Develay, 1993]

En outre, si l'on adopte une approche pédagogique de la transdisciplinarité, il existe deux possibilités d'envisager les rapports entre disciplines dans l'enseignement :

- les activités (tâches) anticipent sur les contenus qu'elles sont censées faire découvrir.

Citons par exemple un petit groupe de professeurs qui envisagent un PAE en commun en considérant en priorité ce qu'ils pourront faire faire aux élèves.

- la réflexion sur les contenus précède le choix d'activités qui en découlent.

Par exemple, un groupe de professeurs où chacun met à plat les concepts intégrateurs de sa discipline, les connaissances procédurales qu'il favorise. Après cette explicitation des contenus ils conviennent des activités qu'ils pourraient mettre en commun.

Cette approche pédagogique de la transdisciplinarité doit permettre de mieux cerner les points de convergence entre les disciplines et les spécificités de chacune d'entre elles.

Cependant l'identification des concepts et des connaissances procédurales caractéristiques de chaque discipline ne résout pas *ipso facto* la question de la transdisciplinarité, car un même concept à l'œuvre

dans plusieurs disciplines n'a pas dans chacune la même signification [5]. De même, nous devons notamment tenir compte du registre de conceptualisation de la notion ainsi que du registre de formulation de la compétence.

L'enseignement, pour arriver à une réelle technologie éducative, doit donc se baser sur une approche transdisciplinaire. Il convient alors d'étudier quels sont les principes et les moyens à mettre en place afin que les situations d'enseignements deviennent réellement des situations d'apprentissage.

Paragraphe 2 : De la situation d'apprentissage à la situation d'enseignement

A partir des travaux de [Chevallard et Joshua, 1985], de [Develay, 1993], de [Houssaye, 1988], et de [Meirieu, 1994], nous pouvons schématiser une situation d'apprentissage autour de trois pôles : le savoir, l'enseignant et l'apprenant.

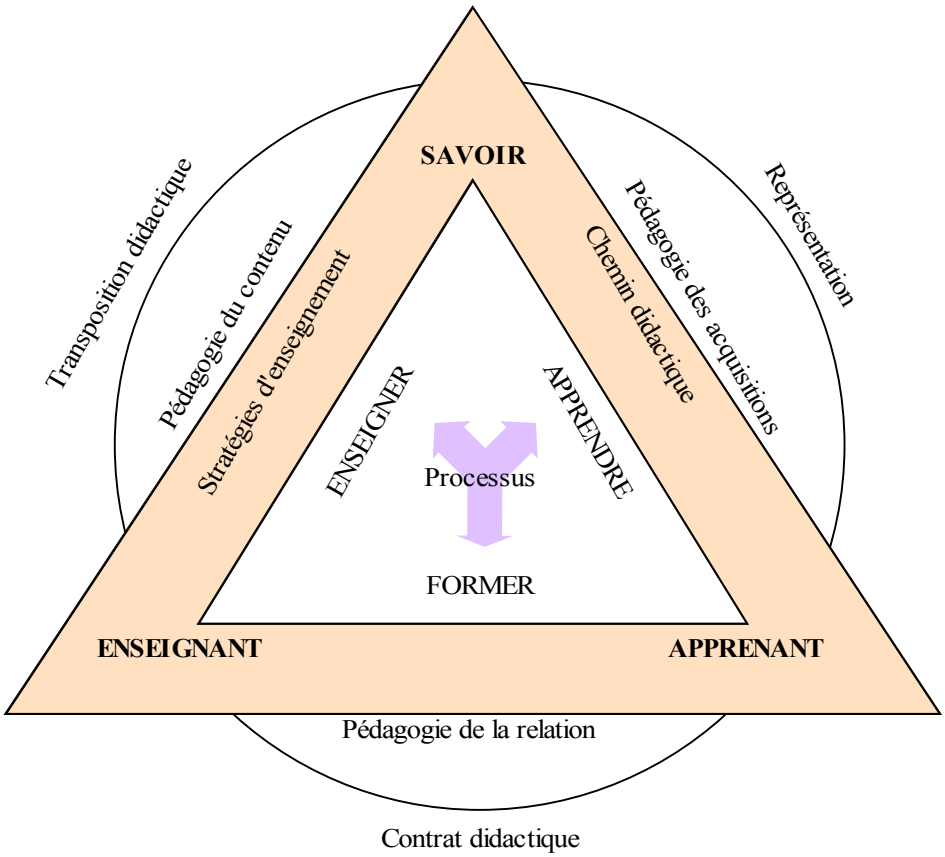

Figure 6 : Situation d'apprentissage

[5] Nous retrouvons ici la notion de *point de vue* telle qu'elle peut être perçue en génie logiciel.

Il s'agit de tenter de comprendre comment se structure ce triangle en considérant à parité égale les trois sommets mais aussi les trois côtés afin de mettre en place des s*ituations d'enseignement* régulées et régulatrices aboutissant à un apprentissage basé sur une *pédagogie différenciée*.

Aux trois éléments essentiels, sont associés trois processus, trois pédagogies ainsi que trois concepts relatifs à la didactique des disciplines.

◊ La *représentation* matérialise un (ou plusieurs) obstacle(s) pouvant se rapprocher de la notion à enseigner. Ce concept de représentation occupe toute sa place en didactique parce qu'il permet de comprendre le fonctionnement de l'apprenant à partir de la relation de ce dernier au savoir.

L'expression des représentations des élèves en situation de classe indique à l'enseignant les écarts de la pensée scientifique de ses élèves, soit les *obstacles* à la compréhension de la situation en jeu.

Ces obstacles constituent alors par rapport à la prévision a priori de la situation, autant d'*objectifs* d'apprentissage/enseignement dont la caractérisation permet à l'enseignant d'inventer des *situations* didactiques correspondantes. Le franchissement des *objectifs-obstacles* déduits s'assortit d'un progrès intellectuel des élèves, c'est-à-dire à un apprentissage.

◊ La *transposition didactique* représente la structure des connaissances perçue et maîtrisée par l'enseignant par rapport aux savoir et savoir-faire de référence.

Ainsi, le maître doit intégrer trois logiques : des contenus, des élèves et sa propre logique pédagogique. La transposition didactique s'exprime en terme de tâches, objets, connaissances procédurales et déclaratives.

◊ Le *contrat didactique* est une relation implicite à caractère local dans le couple enseignant-apprenant, chaque partie déterminant ses responsabilités vis à vis de l'autre. Un contrat solide est à la base d'un enseignement débouchant sur un apprentissage. En effet, l'apprentissage n'est possible que si l'élève veut apprendre et que le maître dispose de suffisamment de connaissances et de moyens pour faire apprendre.

◊ La *pédagogie de la relation* s'exerce entre l'enseignant et l'apprenant. Au sein de ce rapport, le *profil de l'apprenant* doit toujours être considéré afin d'adapter au plus juste les informations qui lui sont données et qui devront déclencher le phénomène mental d'acquisition de connaissances en faisant émerger le désir d'apprendre.

Ainsi, il est très important de fournir la bonne information, le bon outil et la bonne consigne au bon moment au risque d'échouer totalement. Il s'agit donc de placer l'élève dans une situation-problème à la fois accessible et difficile.

◊ La *pédagogie des acquisitions* s'opère entre l'apprenant et le savoir qui sont reliés par un *chemin didactique*. L'analyse par objectifs fournit à la fois un référentiel et un référent, un outil pour la gestion d'une pédagogie différenciée et un support pour la négociation d'une pédagogie contractuelle car chaque individu et ses acquis antérieurs sont pris en compte.

Il convient pour l'élève d'adopter un point de vue extérieur afin de pouvoir juger objectivement son travail. Les objectifs de surface doivent donc être couplés aux *opérations mentales* profondes de l'élève pour assurer un bon résultat. Trois conditions doivent alors se réguler réciproquement en permanence : simplicité d'utilisation, conformité aux apports théoriques et fécondité pour la pratique.

◊ Une *pédagogie du contenu* s'impose entre l'enseignant et le pôle du savoir. Elle se matérialise par différentes *stratégies d'enseignement* qui doivent se fonder sur deux préceptes :

- on ne peut enseigner qu'en s'appuyant sur le sujet, ses acquis et les stratégies d'apprentissage familières,
- l'action didactique doit s'efforcer de faire émerger l'information permettant cette articulation, et elle doit également se donner pour objectif l'enrichissement des compétences de l'apprenant, de ses capacités et l'expérimentation de nouvelles stratégies.

Une stratégie d'enseignement comprend des opérations étroitement liées de :

- saisie des données, donc de choix d'outils,
- traitement des données, donc de choix de démarches.

De toute façon, la pratique didactique doit s'efforcer de faire varier les stratégies d'enseignement de manière à ce que les sujets puissent utiliser leurs stratégies d'apprentissage.

Instruire, c'est donc construire un environnement éducatif pour les élèves, créer des dispositifs qui ont la propriété non pas d'enseigner mais de faire apprendre. D'un projet pédagogique à sa réalisation, il y a tout un chemin à faire ; et c'est le chemin de l'élève, non celui de l'enseignant. *Il ne suffit donc pas de savoir pour savoir transmettre. La transmission du savoir passe par des relais technologiques*. A proprement parler, le savoir ne se transmet pas, il s'acquiert, il se construit. Avec les moyens dont il dispose, l'enseignant ne peut qu'instituer des contextes d'apprentissage plus ou moins fonctionnels, plus ou moins opératoires. La technicité de l'enseignant ne réside pas dans une logique de l'exposition didactique, mais dans la mise en œuvre de situations autorisant le cheminement et la régulation des apprentissages de l'élève.

1. Définition d'une situation d'enseignement

Chaque individu étant unique et possédant ses propres modes de réflexions, il est par conséquent illusoire de parler de théorie de l'apprentissage. Ainsi, il est plus commun de proposer une théorie de l'enseignement.

La *situation d'enseignement* rappelle une situation proche, réussie antérieurement. Il devient possible d'anticiper les *critères de réussite* de la tâche.

Dans le cas où l'apprenant est confronté à une tâche, il y a apprentissage si cette tâche est nouvelle. Sinon l'apprentissage n'est que *machinal* quand il y a seulement placage d'informations dépourvues

de signification, sans intégration des informations à mémoriser à des structures cognitives antérieures. Tout apprentissage oblige donc à surmonter un (ou des) *obstacles* qui constitue(nt) l'essence même de la tâche nouvelle.

La situation d'enseignement qu'un enseignant propose, recèle, selon lui, des *objectifs* qui correspondent aux obstacles qu'il présuppose et que ses élèves auront à surmonter. L'obstacle présent pour l'élève, dans la situation proposée par le maître, est le déclencheur espéré de son apprentissage.

Chaque *problème* à résoudre pour surmonter un obstacle nécessite que l'apprenant utilise un *programme de traitement* approprié à ce problème. Chaque programme de traitement consiste le plus souvent en un travail d'adaptation d'un programme de traitement utilisé dans une autre situation.

Ainsi, un travail cognitif important, dans une situation d'enseignement, est l'identification et la caractérisation des familles de problèmes présents. Non seulement pour optimiser son enseignement, l'enseignant devrait parvenir à identifier les tâches que sa discipline nécessite de résoudre au niveau où il enseigne, mais il devrait aussi avoir une idée relativement précise, à travers ces obstacles, des familles de problèmes à résoudre. Il devrait, de surcroît, aider ses élèves à identifier les activités mentales (correspondant aux programmes de traitement précédents) qu'ils ont dues mettre en œuvre, afin d'espérer le réinvestissement ou le transfert ultérieur. Identifier une famille de problèmes revient donc à prélever, dans une situation, les indices pertinents pour sa résolution. Une *situation-problème* est un cas particulier d'une situation plus générale pour laquelle on connaît à la fois une procédure de solution et les propriétés relationnelles qui permettent de justifier cette procédure. Reconnaître la classe d'un problème requiert une expérience du domaine auquel il appartient.

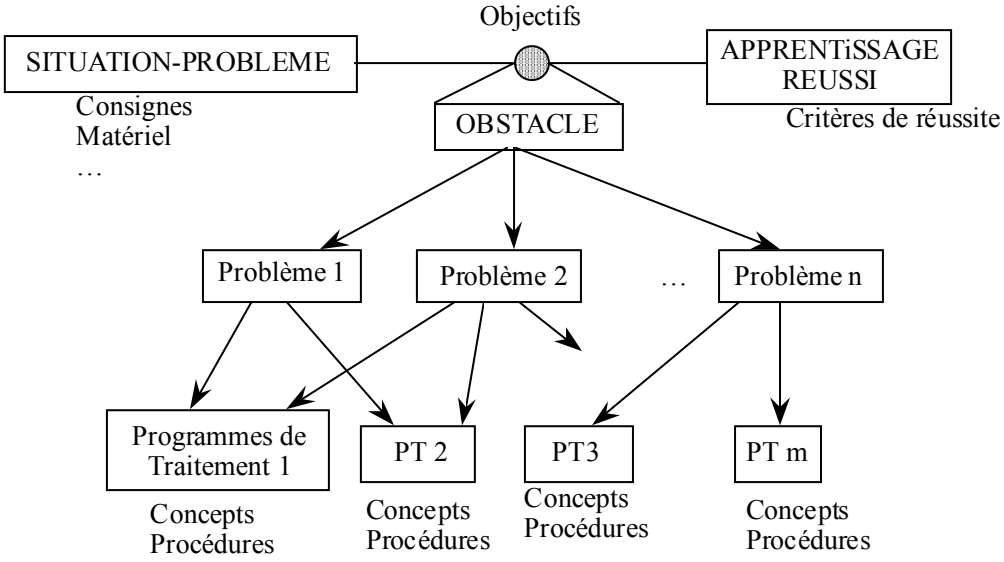

Figure 7 : Situation d'apprentissage

En outre, il convient de présupposer, pour un apprentissage donné que certaines notions, certaines procédures ont été antérieurement maîtrisées, sinon, à chaque apprentissage il conviendrait de remonter à la maîtrise de l'alphabet.

De plus, il ne faut pas confondre apprentissage et substitution ou simple addition de connaissances ; l'apprentissage permet de générer des connaissances connexes appelant elles-mêmes de nouveaux apprentissages. Seule l'activité de *transfert* permet de s'assurer que l'on a appris. Le transfert postule que des connaissances ou des compétences acquises dans un contexte donné peuvent être utilisées dans un autre contexte.

[Develay, 1993] propose alors de modéliser l'activité d'enseignement par la figure suivante :

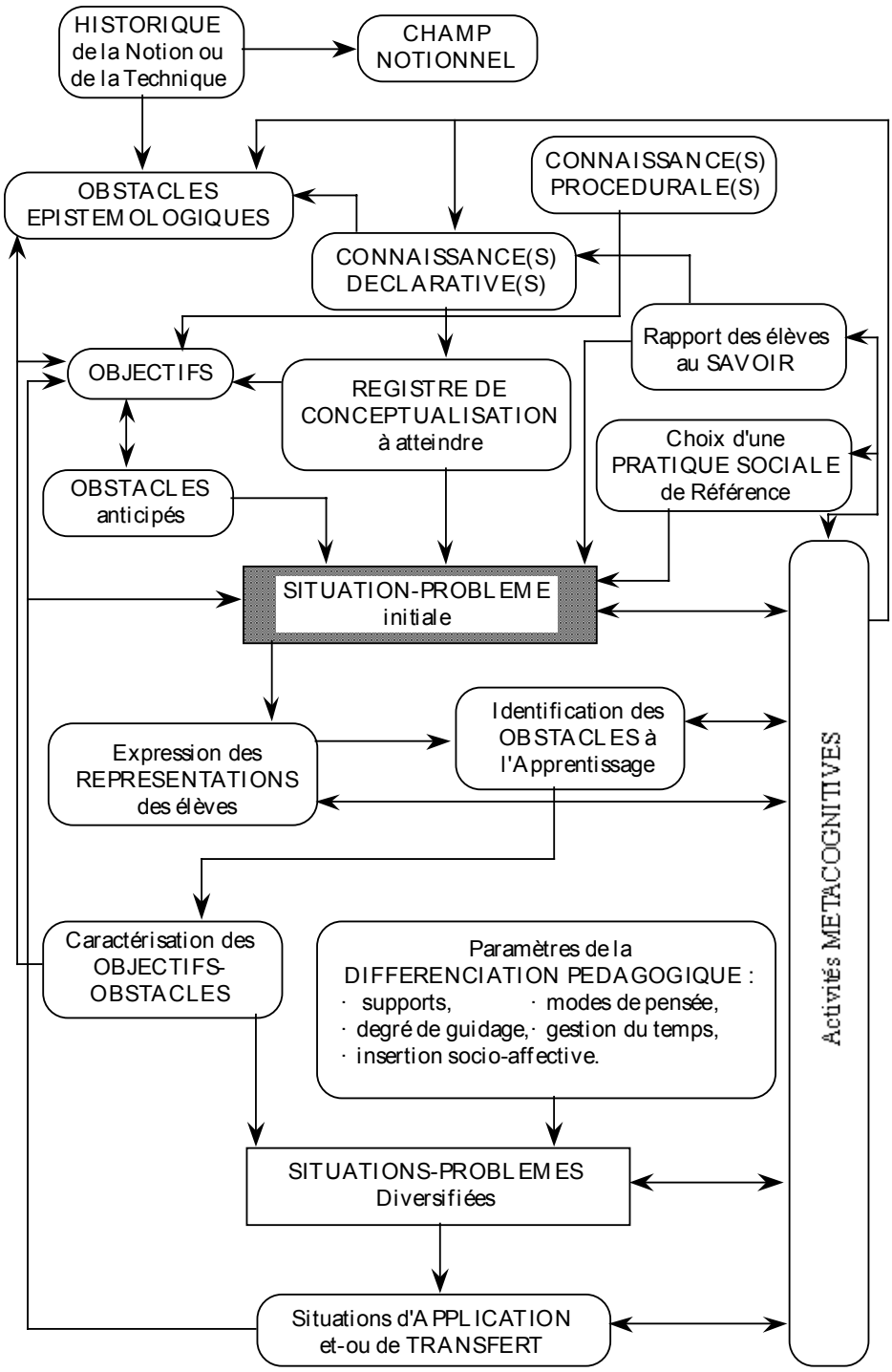

Figure 8 : Activité d'enseignement [Develay, 1993]

De nombreuses boucles de *rétroaction* introduisent constamment des *régulations*, de sorte que ce schéma est à analyser de manière *systémique* et non de manière linéaire. Un enseignant envisagera d'abord, par exemple, les épreuves d'évaluation qu'il peut proposer à ses élèves, avant de songer à la situation d'apprentissage proprement dite. Un autre songera plutôt aux représentations de ses élèves avant de s'intéresser à définir précisément le registre de conceptualisation de la notion qu'il souhaite faire acquérir. Nous retrouvons donc à ce niveau les deux approches de processus de construction des connaissances (ascendante et descendante) évoquée dans le paragraphe 3.3.
L'ensemble de ces activités est également facilité par des *activités métacognitives*. En effet, le fait de comprendre permet d'apprendre.

2. La métacognition

2.1. Définition

« *Connais-toi toi-même !* ». Par cet adage, Socrate signifiait déjà que la connaissance de soi est à la base de tout apprentissage efficace. Le terme de *métacognition*, apparu dans les années 70, caractérise précisément cette réflexion sur sa propre cognition (processus mentaux).
Cette définition a donné lieu à deux phénomènes de natures différentes [Noël, 1995] :

- Phénomène 1 :

  la connaissance de ses propres processus mentaux et du produit de ses processus ;
- Phénomène 2 :

  la régulation que l'on opère sur ces processus.

En outre, ces phénomènes sont décomposables en trois étapes distinctes [Noël, 1995] :

- Etape 1 : le processus métacognitif

  Le sujet a conscience des activités cognitives qu'il effectue ou de leur produit. Par exemple, l'apprenant décrit comment il prend des notes pendant une leçon.
- Etape 2 : le jugement métacognitif

  Le sujet exprime ou non un jugement sur son activité cognitive ou sur le produit mental de cette activité. Par exemple, l'apprenant évalue l'efficacité de sa technique de prise de notes.
- Etape 3 : la décision métacognitive

  Le sujet peut prendre la décision de modifier ou non ses activités cognitives ou leur produit ou tout autre aspect de la situation en fonction du résultat de son jugement cognitif. Par exemple, l'apprenant, selon son évaluation, va modifier sa technique de prise de notes.

Si la métacognition comprend ces trois étapes, on parle alors de *métacognition régulatrice*.

2.2. Importance de la métacognition

La métacognition semble être un des facteurs qui influencent le plus favorablement l'apprentissage. De nombreux auteurs s'accordent sur le fait que transmettre une information seule sur de nouvelles stratégies à mettre en œuvre n'a que peu d'efficacité. Par exemple, des cours magistraux sur les techniques d'étude, employés seuls, ne provoquent pas chez l'apprenant la rupture nécessaire à un changement de stratégie. C'est plutôt la qualité de l'analyse par l'apprenant de ses propres stratégies et de son propre contexte de travail qui est déterminante pour améliorer l'efficacité de son apprentissage. Il est donc important de promouvoir le développement de la métacognition chez les apprenants, c'est-à-dire le développement de la prise de conscience du contexte dans lequel ils travaillent et du but qu'ils poursuivent : c'est à chacun de construire sa propre méthode selon son style personnel.

En effet, les résultats des travaux des chercheurs en sciences de l'éducation sont troublants [Maret, 1995]. Les chercheurs ont initialement pensé qu'il existe une stratégie mentale idéale, permettant une efficacité maximale dans l'apprentissage de telle ou telle donnée. Il aurait suffit de la repérer pour l'enseigner ensuite aux élèves. Or, il semble qu'il n'y ait pas une méthode idéale d'amélioration de ses manières d'apprendre, mais que le simple fait qu'un apprenant pratique la métacognition, quelle qu'en soit la forme, constitue, pour lui, un facteur positif d'apprentissage.
Par exemple, la présentation de certaines analogies peut avoir une importante fonction didactique. Par exemple, un circuit électrique peut être comparé à un circuit d'eau : la batterie correspond à la pompe, les fils aux tuyaux, le courant au débit, …

Le rôle de l'enseignant est donc plutôt de faire émerger à la conscience des apprenants leurs propres démarches et de leur fournir des grilles d'analyse de celles-ci. Les travaux sur la métacognition conduisent donc à remettre en question certaines formes traditionnelles d'enseignement dans la mesure où cette activité permet aux apprenants de s'interroger sur la manière dont ils ont abordé la tâche et dont ils analysent leurs stratégies. Les activités métacognitives constituent donc pour l'apprenant le complément de l'expression des représentations pour aider à installer des procédés permanents d'auto-évaluation. Elles peuvent aider l'apprenant à décontextualiser les situations cognitives, dans le but d'en identifier les schémas d'actions, les procédures à réinvestir et à transférer dans des situations nouvelles.

Nous avons considéré de multiples points à prendre en compte pour développer des situations d'enseignement aboutissant à un apprentissage. Nous allons maintenant voir dans quelles mesures les outils informatiques actuels permettent de prendre en compte tout ou partie de ces aspects.

Chapitre 3 : Du rôle de l'informatique dans l'enseignement

Dans ce paragraphe, nous allons présenter comment l'informatique intervient dans le domaine de l'enseignement ; ce que nous entendons par le terme générique « génie éducatif ». Ainsi, nous abordons notamment les apports de l'intelligence artificielle avec la notion de système tuteur intelligent pour aborder ensuite le terme plus général d'EIAO pour lequel l'utilisateur doit se situer au cœur du processus d'enseignement. Ceci nous amène ensuite à étudier les différentes façons d'aborder l'acquisition et la gestion des connaissances.

Paragraphe 1 : Le génie éducatif

Nous allons tout d'abord considérer ce que nous entendons par le terme de « génie éducatif ». Nous présentons notamment l'importance de l'intelligence artificielle au travers du fonctionnement général des systèmes experts. Nous proposons ensuite différents systèmes au travers desquels nous précisons la place de l'utilisateur qui doit être au cœur même du processus d'enseignement.

1. Présentation générale

La conception des EIAO, dans les approches initiales, partait volontiers d'une analyse des connaissances de l'élève et d'une analyse de type pédagogique largement basée sur une logique de transmission de connaissances. Nous parlions alors d'Enseignement Intelligemment Assisté par Ordinateur.

Le but premier de l'EIAO est de s'intéresser non seulement à l'extraction des connaissances au sens de l'ingénieur cogniticien, mais également de fermer la boucle de retour des connaissances. Ainsi, les interactions entre l'intelligence artificielle et la didactique ouvrent de nouvelles voies de recherche.

Dans les approches actuelles, la conception part d'une analyse de ce que seront les situations didactiques créées. Il s'agit alors de prendre en compte les interventions du maître et des élèves, et de se placer davantage dans une logique de re-création ou reconstruction des connaissances. Nous parlons maintenant *d'Environnement Interactif d'Apprentissage avec Ordinateur* [Bruillard et Vivet, 1994].

Les différentes disciplines concernées par l'EIAO (informatique, intelligence artificielle, didactique, psychologie cognitive, sciences de l'éducation) se rejoignent notamment dans des problématiques d'usage et de conception des EIAO.

Ainsi, d'après [Balacheff, 1994], l'enjeu de l'intelligence artificielle dans le champ de la didactique ne sera donc pas que l'ordinateur se comporte comme un « professeur », mais qu'il soit capable de créer des conditions favorables à la construction par l'élève de connaissances acceptables en référence à un objet d'enseignement, en lui assurant des *rétroactions pertinentes*.
D'où un phénomène central : la *transposition didactique*. La question est de savoir quelles représentations des connaissances élaborer pour la modélisation computationnelle de processus dont la finalité est l'apprentissage humain.
Les comportements de la machine doivent être examinés mais aussi la façon dont ils sont produits. Ce problème est envisagé en terme de fidélité entre la réalisation physique d'une représentation et la réalité : nous parlons alors de *transposition informatique*. Dans le contexte de l'EIAO, ceci signifie une contextualisation de la connaissance qui peut avoir des conséquences importantes sur le résultat des apprentissages.
Ainsi, la mise en œuvre computationnelle des divers objets d'enseignement est le fruit de cette double transposition, qui, pour [Balacheff, 1994] est « *un véritable processus de modélisation et donc de théorisation des objets d'enseignement et de leurs conditions d'existence* ».

Nous avons défini de façon succincte les principales composantes du génie éducatif. Considérons maintenant la place de l'intelligence artificielle.

Selon [Farreny et Ghallab, 1987], les travaux d'intelligence artificielle relatifs à la *représentation des connaissances* peuvent être regroupés en cinq classes de représentation : la logique (logique des prédicats du premier ordre, logiques non standards : modale, temporelle, probabiliste, floue, …), les représentations procédurales, les réseaux sémantiques, les « *frames* » et les systèmes de règles de production.
Ainsi, la représentation des connaissances, essentielle en intelligence artificielle, recouvre plusieurs problèmes dont ceux soulevés par :

- le formalisme de représentation,
- les méthodes et les moyens d'accès aux connaissances,
- la modification des connaissances,
- la production de nouvelles connaissances (inférence).

En outre, il convient de considérer les problèmes générés par la mise en œuvre des langages informatiques et des techniques de programmation symbolique à chaque type de représentation.
De manière générale, l'étude des logiques formelles a permis de dégager des méthodes de résolution de problèmes, c'est-à-dire des méthodes de découverte et justification des solutions, tout en évaluant rigoureusement certaines propriétés ou limites de ces méthodes. Néanmoins, de nombreux systèmes de résolution de problèmes ont pu être construits et donner relativement satisfaction sans que soient préalablement postulés ou établis les rapports qu'ils entretiennent avec une logique formelle

déterminée. Tel est le cas, en général, des *systèmes à base de règles* qui vont de pair avec le concept de *systèmes experts* qui abordent avec un relatif succès certaines tâches concrètes comme le diagnostic, la prévision, la planification d'actions qui sont des activités souvent difficiles à représenter sous forme d'algorithmes sûrs et définitifs.

Précisons maintenant les différents types d'EIAO. Une première classification peut se faire selon l'existence de deux familles de mécanismes :

- le *processus piloté par le diagnostic* ou processus d'apprentissage.

Cela concerne par exemple l'intervention opportuniste du système dans un environnement d'apprentissage. Ces mécanismes assurent la correction mais aussi une fonction d'aide,

- le *processus piloté par les stratégies pédagogiques, ou* processus d'enseignement.

Il existe aussi une nette distinction, quel que soit le domaine d'application, entre les environnements d'apprentissage et les précepteurs intelligents : les premiers utilisent une pédagogie de découverte alors que les seconds guident l'apprentissage.

Un autre critère de classification des EIAO est le degré d'initiative laissé à l'élève, ou de façon duale le degré de directivité du système. Pour pouvoir maintenir une interaction pertinente en permettant une initiative réelle de l'élève, la machine doit avoir la capacité de prendre en compte ses intentions et les conceptions qui sous-tendent ses actions. Ce problème est bien plus complexe que celui de la modélisation de la connaissance dite experte.

Nous distinguons trois types de système :

① Les *systèmes tuteurs* laissent très peu d'initiative aux apprenants. Nous entendons par tuteur la traduction du mot anglais « *tutor* » qui signifie « *private instructor* ». Il serait plus correct de le traduire par *précepteur* mais la communauté EIAO semble avoir fait un autre choix : les tuteurs informatiques se comportant le plus souvent comme une armature qui maintient l'élève dans le droit chemin.

② Au contraire, les *micromondes* ou *environnements d'apprentissage* laissent toute initiative à l'élève dans le cadre des contraintes syntaxiques et lexicales ordinaires de la communication. L'idée est de fournir à l'élève un ensemble d'objets élémentaires et un ensemble d'outils primitifs à partir desquels il peut construire des objets de plus en plus complexes alors qu'il progresse dans son exploration (principe d'évolution conjointe du micromonde et de son utilisateur).

③ Les *systèmes* dits de « *coach »* sont des tentatives pour trouver une solution intermédiaire. Dans un tel environnement, l'évaluation de l'activité de l'élève et des productions auxquelles elles conduisent est une tâche complexe.

Nous allons maintenant fournir une description rapide d'un certain type d'EIAO, les systèmes tuteurs intelligents.

2. Les Systèmes Tuteurs Intelligents

L'objectif est d'écrire des logiciels capables de « *s'adapter à l'élève »,* par opposition aux didacticiels classiques où les réactions de l'élève sont prévues en fonction de différents profils préprogrammés. Plusieurs architectures pour la construction de ce type de logiciels ont été proposées. La communauté scientifique se réfère à l'architecture de STI suivante [Nicaud et Vivet, 1988] :

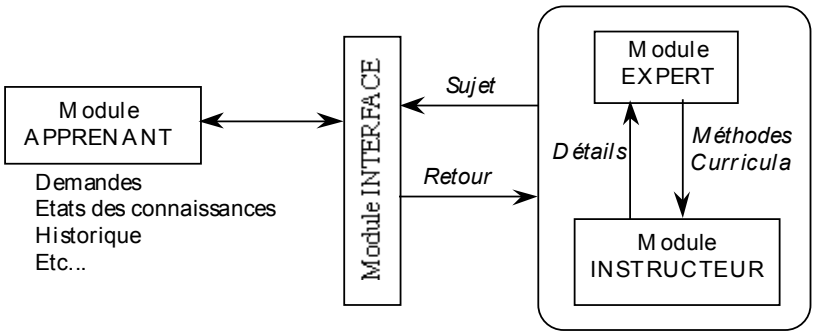

Figure 9 : Architecture d'un système tuteur intelligent [Nicaud et Vivet, 1988]

Ainsi, un système tuteur intelligent est un système composé de quatre modules travaillant en coopération dans un objectif d'apprentissage :

- le module *expert* est spécialiste du sujet ; il sait résoudre les problèmes et a une véritable représentation des connaissances du domaine de l'enseignement. Ce module fournit tout autant le *curriculum* que les *méthodes d'enseignement* de bas niveau.
- le module *instructeur* fournit aux apprenants le *matériel* nécessaire à traiter le sujet selon différentes façons basées sur les *requêtes* des apprenants et leurs *réponses*. L'instructeur est l'interlocuteur de l'élève, il dirige la session, développe des stratégies pédagogiques, propose des exercices, sait expliquer et comprendre les erreurs de l'élève
- le module *interface* fournit les interactions entre le pédagogue et les apprenants.
- le module *apprenant* enregistre les informations de base sur ceux-ci durant tout l'apprentissage, incluant le matériel et ses *préférences*. L'apprenant doit être modélisé pour que, à un instant donné d'utilisation du système, le modèle puisse rendre compte des connaissances de l'élève (exactes ou erronées).

[Woolf, 1996] étend même ceci avec la notion de STI multimédia recherche visant à fournir des outils et un cadre de travail permettant aux non-programmeurs de réaliser seuls des STI.
Ainsi, selon [Woolf, 1996], les simulations usuelles à base de scénarios fournissent souvent seulement peu de chemins pour une situation et pas de possibilité d'adaptation de la présentation afin de

connaître les besoins de l'utilisateur ou leurs connaissances individuelles. Elles classent souvent les actions des apprenants comme correctes ou incorrectes dans un contexte limité. Cependant, les systèmes à base de connaissances permettent de suggérer qu'une action de l'apprenant est hors propos ou trop tardive, en la comparant avec une action d'un expert. Ces systèmes permettent de raisonner sur des objectifs et des situations d'apprentissage vers lesquels la simulation devrait être dirigée. Une vérification dynamique, des calculs simultanés, et un raisonnement sur les actions fournissent des comparaisons entre les actions de l'apprenant et de l'expert en temps réel. Ainsi, puisque les recommandations du système représentent le contexte courant, elles sont souvent judicieuses et exactes.

Le principal intérêt de ce travail est de fournir aux apprenants des expériences réelles et précises dans une variété de domaines. Les apprenants résolvent des problèmes, leurs actions fournissent de vrais effets sur les simulations, et les retours du STI permettent de garder les apprenants en action et en apprentissage.

Pour se concentrer uniquement sur les besoins des utilisateurs, [Jackson *et al.*, 1996], tout comme [Rosson et Carroll, 1996], adoptent une approche de conception centrée sur l'utilisateur et basée sur la *construction par assemblage* (« *scaffolding* »). Ceci permet alors de travailler sur des représentations acquises par le passé, de s'y référer par analogie et d'assurer un couplage étroit entre les actions de l'usager et les retours visuels fournis en retour de ces actions par l'ordinateur, ce qui permet de raffiner et réviser les schémas utilisés.

Vis à vis des EIAO, le terme STIC (*Système de Transfert Intelligent de Connaissances*) désigne aussi les systèmes développés dans d'autres domaines que l'éducation proprement dite, c'est-à-dire, dans des domaines où il existe un transfert de connaissances entre une machine et un humain. [Lima, 1995] propose SINTONIA [6], acronyme de « *a System for INtelligent Tutoring based on an Object Notation Interacting with the Author* », qui permet aux auteurs de STIC de concevoir et de réaliser des processus de diagnostic contextuels et intelligents nécessaires à la construction et à la maintenance du modèle de l'utilisateur.

Nous avons dressé un aperçu général des possibilités offertes par différents systèmes tuteurs intelligents. Il est également important de signaler ici le modèle du « *Campus Virtuel* » [Paquette *et al.*, 1996], développé par le laboratoire LICEF [7], qui est un atelier de génie éducatif sur lequel nous reviendrons tout au long de ce document. Son objectif principal concerne la formation à distance et consiste à soutenir le processus autonome et individuel de construction des connaissances par l'apprenant, tout en intégrant l'importante dimension collaborative de l'apprentissage. Le « *Campus*

6 SINTONIA est réalisé dans l'environnement Smalltalk/V.
7 Site WWW du LICEF (http://unitl.teluq.uquebec.ca/~licef/html/projets.html) Septembre 1996

Virtuel » est un système informatique qui met en relation un réseau d'acteurs et de ressources diversifiés. Les acteurs se regroupent à l'occasion d'événements d'apprentissage : programmes cours, modules ou activités d'apprentissage. Du point de vue informatique, le « *Campus Virtuel* » est un système regroupant plusieurs logiciels qui permettent aux différents acteurs de jouer leur rôle dans le système d'apprentissage.

Quelle que soit l'architecture supportant un système dédié à l'enseignement ou à l'apprentissage, il est primordial que l'utilisateur soit le centre d'intérêt principal du système. C'est ce que nous allons démontrer dans le paragraphe suivant.

3. Vers des EIAO centrés sur l'utilisateur

Dans le passé, l'axe central de l'enseignement portait sur le contenu. Le curriculum était structuré autour de modules de base tels que la littérature, l'histoire et les sciences ; chacun d'entre eux était divisé, par des experts, en des paquets plus petits, plus aisément manipulables et enseignés selon un plan de cours préétabli.

La nouvelle approche, centrée sur l'apprenant (« *learner-centered* ») [Norman et Spohrer, 1996], s'assimile à l'approche centrée sur l'utilisateur (« *user-centered* ») mais l'accent est porté ici non pas sur l'interface mais sur les besoins, les capacités et les intérêts de l'apprenant. La conception centrée sur l'apprenant va souvent de pair avec l'approche basée sur la notion de problème (« *problem-based approach* ») ; les problèmes sont alors choisis en fonction des intérêts et des besoins des apprenants. La focalisation se fait ici davantage sur l'apprenant et les problèmes issus du monde réel, plutôt que sur une analyse structurée du contenu du curriculum bien que les deux aspects soient bien évidemment nécessaires.

Cette philosophie n'est pas nouvelle mais l'outil informatique apporte des capacités technologiques puissantes. En effet, tout le monde est abreuvé de toutes parts par des informations diverses. Cependant, les données restent inutiles sans structure, les faits sont dénués de sens hors d'un cadre conceptuel. Ainsi, les informations ne sont pas forcément des connaissances. [Neumann et Horwitz, 1996], par exemple, introduisent un nouveau paradigme dans la technologie de l'éducation : « *l'hypermodèle* ». Ce dernier a pour but de faire le lien entre un modèle et la partie du monde physique que ce modèle représente, entre les « faits et figures » qui nous sont offertes par le monde réel et les associations mentales que nous construisons pour les expliquer. L'hypermodèle utilise un ordinateur afin de transformer ces informations en connaissances (notamment par l'inclusion de liens sémantiques).

De plus, selon [Lin *et al.*, 1996], il y a une différence entre la communication et l'interaction réelle. La communication implique le transfert d'informations depuis une source vers un récepteur. L'interaction réelle signifie la communication dans les deux sens, chaque information produite par chaque partie

influençant les actions de l'autre. Les interfaces utilisateurs qui implémentent une interaction réelle sont appelées *interfaces adaptatives* (elles adaptent leurs caractéristiques à celles des utilisateurs).

[Edelson *et al.*, 1996] ont implanté « *The Collaboratory Notebook* » comme une base de données hypermédia. Les apprenants disposent d'un ensemble fixé de types de pages qui correspondent à un modèle de tâche pour la demande de renseignements scientifiques qui inclut des questions, des conjectures favorables ou défavorables, des plans, des étapes dans les plans, des informations et des commentaires [8]. L'objectif est de libérer les apprenants pour leur permettre de se concentrer sur le contenu en reportant une partie du lourd travail du processus de structuration sur l'outil.

[Guzdial *et al.*, 1996] préconisent les stratégies de résolution de problèmes des experts comme modèles utilisés pour les apprenants. En effet, la façon la plus commune utilisée par les experts pour résoudre un problème est de le comparer à des cas déjà traités. De nombreux répertoires de cas ont ainsi été développés dans cette optique de réutilisation par des étudiants. Ils contiennent les expériences et les résultats de problèmes similaires posés à différents étudiants. L'examen de ceux-ci peut aider les apprenants à trouver des pistes de recherche de solutions en se projetant sur les effets induits, etc…

Les objectifs du projet européen KAMP [9], au sein duquel notre équipe de recherche est partie prenante, sont de fournir aux enseignants et étudiants un ensemble d'outils puissants permettant un enseignement interactif multimédia et à distance via le réseau Internet. La finalité consiste alors à amener le savoir vers les gens plutôt que le contraire, ce qui constitue l'approche courante.
Le concept de classe virtuelle est le modèle sur lequel portent toutes les activités pédagogiques de KAMP. Les pages Web incorporent alors des outils permettant aux apprenants de communiquer par courrier électronique ou par vidéo conférence, de participer à des groupes de discussion (« *chat* »), d'accéder à des ressources pédagogiques sous forme de pages web. De même, des outils sont à la disposition des enseignants afin qu'ils construisent leurs enseignements selon une structure hiérarchique bien définie (cours, modules, activités, et exercices) mais aussi afin de pouvoir notamment gérer les inscriptions et suivis des apprenants.
La mise en place d'enseignements via l'ensemble de ces outils offre alors aux apprenants un degré de liberté fort dans une acquisition des connaissances où ils sont individuellement responsables de leur progression.

[8] Des conventions consistantes pour la sémantique des types de lien rendent plus facile la navigation dans la base de données hypermédia.
[9] Acronyme de « *Knowledge Assurance in Multimedia Publishing* » au sein du Projet ET 1013.

Nous pouvons également citer ici d'autres projets de développement d'enseignement à distance via Internet et notamment les sites francophones comme « *Télé-université* » [10], qui est à l'origine du « *Campus Virtuel* » sur lequel nous reviendrons dans la seconde partie de cette thèse (chapitre 6) ou bien « *Cyberscol* » [11] et anglophones comme « *The Open University* » [12] ou encore « *The Virtual University* » [13].

Ainsi, parmi les auteurs qui ont intégré cet ensemble de préceptes dans l'architecture des systèmes d'EIAO, nous retrouvons un large spectre d'approches à la fois sur le style d'enseignement et sur l'utilisation de la technologie.
Un invariant demeure toutefois. En effet, quel que soit le type d'EIAO utilisé pour développer l'apprentissage, il est nécessaire de rester cohérent et rigoureux vis à vis de l'acquisition et de la gestion des connaissances.

Paragraphe 2 : Gestion et acquisition des connaissances

Définissons tout d'abord la notion de connaissance. [Brunet et Ermine, 1994] donnent les définitions suivantes : une *connaissance* se différencie d'une *donnée* par le fait qu'elle est « sémantisée » et d'une *information* par le fait qu'elle enrichit le modèle que l'individu ou le système se fait du monde.

1. Trois principales approches

La gestion des connaissances relève de sciences ou techniques diversifiées : sciences des organisations, sciences de l'information, techniques documentaires, intelligence artificielle, etc… Dans des perspectives opérationnelles, il existe trois principales approches qui dérivent sur des technologies : le connexionisme, le cognitivisme et l'hypothèse systémique.
Le connexionnisme est une école de pensée fortement typée des sciences cognitives. Disons seulement que son hypothèse de base est que les phénomènes de cognition sont explicables (et quelque peu imitables) à partir de modélisations neuronales, inspirées de modèles de fonctionnement du cerveau élaborés par certaines branches des neurosciences.
Le cognitivisme est une autre école de pensée dans les sciences cognitives qui s'oppose souvent à l'école connexionniste. La connaissance est un ensemble de symboles qu'on peut manipuler de manière « computationnelle ». Traiter la connaissance, c'est donc *établir* des systèmes de symboles

[10] Se référer à l'adresse suivante : http://www.teluq.uquebec.ca/Alice/alice.html
[11] Se référer à l'adresse suivante : http://Cyberscol.cscs.qc.ca/
[12] Se référer à l'adresse suivante : http://www.open.ac.uk/
[13] Se référer à l'adresse suivante : http://www.vu.org/

qui permettent de les *manipuler* (sous forme de modèles). Les techniques de génie cognitif se conçoivent comme une activité de transformation de modèles pour arriver à un modèle final opérationnel (souvent un logiciel). On part du monde réel (l'expert, par exemple), on recueille la connaissance (modèle pratique), on la modélise (modèle cognitif), on en fait un modèle adapté à une solution informatique (modèle symbolique), et enfin, on programme (modèle informatique). Il est inutile d'insister, après cette cascade de modèles, pour être convaincu que, même si le prix à payer pour obtenir une application opérationnelle, le résultat final est considérablement appauvri par rapport à la richesse de la connaissance que peut détenir un être humain.
Alors que dans les approches courantes les connaissances peuvent être localisées (système d'information, de décision, d'opération), dans l'hypothèse systémique, « l*a connaissance est partout »*. Elle n'est donc pas un attribut propre à un des sous-systèmes, elle existe cependant en tant que telle, comme un patrimoine propre au système. La connaissance n'est pas matérialisable telle quelle (par modélisation, apprentissage, etc…), mais perceptible à travers ses interactions avec les autres sous-systèmes.

En outre, selon la définition du Petit Larousse, « *apprendre c'est acquérir des connaissances* ». Pourtant, il existe une différence entre l'acquisition et la construction des connaissances utilisables par une machine. D'un côté on se soucie de modéliser efficacement les connaissances tandis que de l'autre côté, on est principalement concerné par la modélisation informatique des processus d'acquisition eux-mêmes. Cependant, si l'on s'en réfère aux travaux de [Thomas *et al.*, 1996], même si il y a complémentarité entre ces deux modalités, nous ne pouvons les dissocier.
Ainsi, selon [Rialle, 1993], « capturer » la connaissance pour la conserver dans un système à base de connaissances est une tâche complexe, coûteuse en temps et souvent en moyens matériels et humains. D'où la problématique du *cogniticien* qui doit palier la double incomplétude suivante :

- celle de l'expert, relative aux facultés de modélisation des connaissances et d'utilisation d'outils de développement de systèmes à bases de connaissances,
- celle des outils d'intelligence artificielle.

Nous allons étudier différentes techniques existantes supportant le processus d'acquisition des connaissances et nous présentons ensuite quelques méthodes et outils correspondants.

2. Techniques d'acquisition des connaissances

Il existe plusieurs techniques d'acquisition directe des connaissances [Rialle, 1993] :

- *l'interview*, utile au tout début du processus, elle s'avère pratiquement inappropriée à un transfert de connaissances complexes. Elle doit être structurée et le plus souvent associée à d'autres techniques d'acquisition.

  L'interview structurée est un moyen pour le cogniticien de garder le contrôle du transfert et de son orientation dans la direction appropriée en évitant les effets de digression. Il ne s'agit cependant pas de contraindre l'expert en lui imposant notre compréhension du domaine mais de lui apporter une aide pour la structuration de l'énonciation de connaissances.

  Dans cette technique, le cogniticien se doit de posséder, au moins en partie, le vocabulaire et certaines connaissances et concepts de base du domaine.

  L'interview permet de faire ressortir une taxonomie des entités constitutives du domaine. Elle sert donc à élaborer un réseau informel de connaissances.
- *l'analyse de protocoles* consiste à regarder l'expert en situation de résolution réelle. Son inconvénient majeur réside dans le coût important et l'incomplétude dus à la verbalisation qui s'avère parfois impossible, incomplète et-ou inexacte.
- le *questionnaire* est peu coûteux et complémentaire de l'interview. Il nécessite toutefois une formulation non ambiguë des questions mais offre la possibilité de travail avec plusieurs experts.
- les *tests de mémorisation*, …

La technique la plus répandue pour l'acquisition des connaissances est *l'interview* qui, comme son nom l'indique, nécessite qu'un cogniticien interviewe un expert.
L'approche normale consiste à enregistrer la conversation en entier, puis à la transcrire et l'analyser afin d'identifier et d'extraire les éléments pertinents de connaissance. L'analyse des transcriptions doit fournir des connaissances pratiques et la transcription forme un bon enregistrement de la source de ces connaissances. Cependant, l'analyse des transcriptions est coûteuse en temps, génère beaucoup d'informations non pertinentes, et ne fournit aucune garantie sur la complétude des connaissances acquises.
Des techniques alternatives pour obtenir une transcription, comme le traitement d'interviews soigneusement structurées, ou l'interrogation d'un expert avec un historique de cas (*analyse de protocole*), ont été développées pour fournir des transcriptions plus structurées. Cependant, de telles transcriptions allègent les problèmes associés à l'analyse de ces transcriptions, mais elles ne les effacent pas.

Nous avons présenté différentes techniques destinées à l'acquisition des connaissances. Toutefois, pour [Goguen, 1996], les techniques usuelles d'élicitation des informations tacites, telles que les questionnaires, les interviews, les groupes de concertation s'avèrent inadéquats et il convient alors

d'utiliser « *l'ethnométhodologie* ». Dans cette approche, l'analyste regroupe les informations selon des situations naturelles où les participants sont engagés dans des activités usuelles. De plus, l'analyste n'impose pas d'objectif, c'est-à-dire des catégories préconçues permettant d'expliquer ce qui se produit. L'analyste utilise plutôt les catégories que les participants utilisent eux-mêmes implicitement pour communiquer.
Cette définition demeure toutefois confuse et afin de surmonter certains de ces problèmes, les cogniticiens se sont donc penchés dans le domaine de la psychologie pour produire des techniques comme le *tri de cartes* (« *card sort* »), les *grilles de répertoires* (« *repertory grid* ») et les *grilles émaillées* (« *laddered grid* »).

Les *grilles émaillées* sont des questions prédéfinies visant à persuader un expert de développer entièrement une hiérarchie taxonomique [Burton *et al.*, 1988]. A partir d'un simple terme du domaine, des questions permettent d'éliciter les super-classes, les sous-classes ou les membres des classes, qui sont liées à l'objet existant dans la grille hiérarchique. Des questions typiques incluent « Quel est le *terme* et un exemple ? », ou « Quels sont les autres exemples de *terme-1* autres que *terme-2* ? ». En appliquant de façon répétitive cette même procédure aux objets nouvellement élicités, une taxonomie approfondie peut être construite.

Le *tri de cartes* [Burton *et al.*, 1988] est une technique simple mais étonnamment efficace dans laquelle un expert catégorise les cartes qui représentent des termes à partir du domaine des connaissances. Les noms des différents termes du domaine sont écrits sur des cartes d'index individuelles, puis on présente à l'expert cette pile de cartes et on l'interroge pour qu'il les classe en piles selon un de ses propres critères. Ensuite, la classification de chaque carte est notée, les cartes sont mélangées et on demande à nouveau à l'expert de répéter cette procédure en utilisant un critère de tri différent. Cette procédure est répétée jusqu'à ce que l'expert ne trouve plus de critères de différenciation des cartes. La sortie du tri des cartes est un ensemble de classifications de termes du domaine selon une ou plusieurs catégories sur plusieurs dimensions différentes.

Les *grilles de répertoires* sont une technique avec laquelle un expert fait des distinctions entre les termes du domaine sur des dimensions choisies. Les dimensions sont similaires aux catégories générées par le tri de cartes, mis à part qu'elles sont censées être des variables continues. Les dimensions sont générées de façon courante par la technique dite « *triadic* » [Boose, 1989] : en sélectionnant trois termes du domaine au hasard et en demandant à l'expert de qualifier une façon de différencier deux termes par rapport au troisième. Tous les termes du domaine de la grille sont ensuite classés selon chaque dimension. Il en résulte une grille dans laquelle chaque terme est catégorisé sur chaque dimension.

Une des fonctionnalités des grilles de répertoires, qui les met à part des autres techniques d'acquisition des connaissances est que les classifications dans une grille peuvent être analysées avec des statistiques, en utilisant une analyse par groupe (« *cluster analysis* »), afin de voir si l'expert a implicitement catégorisé les termes d'une certain façon. Le regroupement des concepts produits par l'analyse statistique de la grille de répertoires est représenté de façon usuelle par un « *dendogrammme* » (c'est-à-dire un diagramme en forme d'arbre), dans lequel chaque terme du domaine est une feuille, et la proximité dans l'arbre représente une similarité statistique. Il est possible de représenter le regroupement statistique avec une grille émaillée (c'est-à-dire une taxonomie), dans laquelle les termes du domaine forment les feuilles, et les classes indiquent le niveau de similarité entre les termes du domaine en utilisant une valeur en pourcentage (100% indique que deux objets sont identiques en tous points de vue, 0% indique qu'ils sont à l'opposé du spectre selon toutes les dimensions). L'expert et/ou le cogniticien est (sont) ensuite autorisé(s) à rationaliser cette taxonomie en affectant des noms significatifs à certaines classes et en en supprimant certaines autres.

Les bénéfices de ces trois techniques sont qu'elles fournissent en sortie des catégorisations et des relations ; qu'elles assurent un couverture complète des connaissances en nécessitant la catégorisation de tous les éléments ; et qu'elles sont relativement simples à administrer.
Nous allons maintenant étudier différentes méthodes et outils pour l'acquisition des connaissances.

3. Méthodes d'acquisition des connaissances

Dans ce paragraphe, nous allons considérer les deux principales méthodes que sont KOD, KADS et CommonKADS et pour lesquelles nous étudierons les outils qui les supportent.

3.1. KOD

KOD [Vogel, 1988] a pour objectif de proposer des cadres de collecte et de modélisation de la connaissance, et de permettre le passage de cette connaissance à une *information* manipulée par la machine.

Cependant, KOD possède une approche du développement de systèmes à base de connaissances plus originale dans la mesure où elle n'est pas uniquement basée sur les modèles mais aussi sur la linguistique. Nous allons aborder cette méthode en prenant en considération son cycle de vie et les différents modèles qui le composent.

Le cycle de vie se rapproche du cycle en « V » que l'on connaît en génie logiciel et s'inscrit dans une démarche qualité où sont pris en compte trois modèles fondamentaux (pratique, cognitif et informatique) auxquels sont associés des tests permettant de vérifier la qualité de la modélisation.

Si l'on reprend l'analogie faite avec le cycle de vie en « V » du génie logiciel, les modèles pratique, cognitif et informatique correspondent respectivement aux phases de spécification, de conception globale et de conception détaillée.

La figure suivante synthétise ces propos :

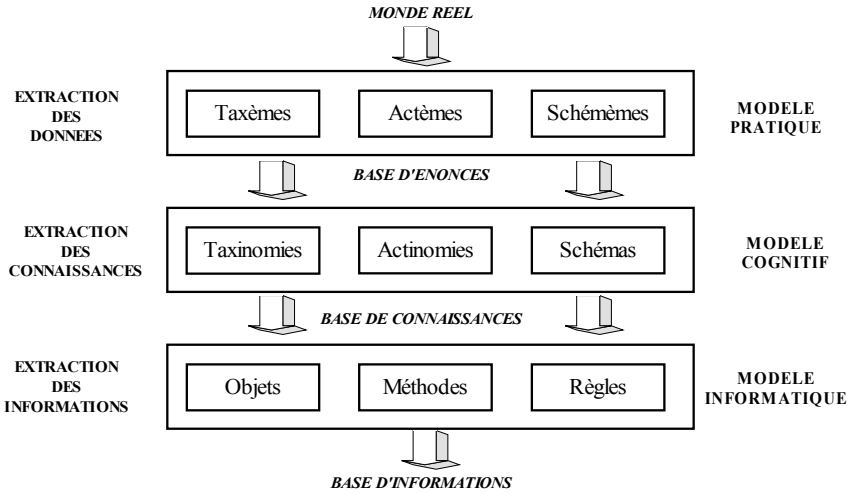

Figure 10 : Les trois niveaux de KOD [Vogel, 1988]

Le modèle pratique consiste à extraire du recueil de connaissances les entités de base qui vont être consignées dans un corpus, recueil de la sémantique du discours de l'expert. La réalisation de ce corpus implique de mettre en œuvre des théories issues de la linguistique, de déterminer des agrégats

de connaissances et de valider le modèle produit. Pour ce faire, il est conseillé de respecter les étapes suivantes :

- identifier la portée du projet en définissant la finalité du produit,
- voir à qui s'adresse l'application,
- lister et classifier les experts,
- cerner les différents champs de compétences.

Une technique à base d'interview peut être utilisée ici. Le modèle pratique est ensuite construit à partir des énoncés issus des protocoles verbaux où l'information pertinente se trouve enfouie sous un ensemble de détails inutiles ou contradictoires, et de redondances. Les entités de base qui se dégagent de ce modèle sont alors les taxèmes, les actèmes et les schémèmes.

Le modèle cognitif prend en compte les agrégats de connaissances décrits dans le corpus du modèle pratique, en les regroupant d'un manière structurée. Ce processus de classification est abordé d'un point de vue linguistique, de même que les structures de connaissance dégagées et manipulées dans ce modèle. Cette étape consiste donc à regrouper les concepts découverts lors de l'extraction des énoncés dans la phase précédente. Trois axes de modélisation se dégagent en pratiquant une analyse structurale et sémantique : les taxinomies, les actinomies et les schémas.

Par contre, le principal défaut de KOD est que cette méthodologie ne prend pas en compte le modèle informatique. Nous pouvons toutefois signaler des tentatives de couplage du modèle informatique avec le modèle objet comme, par exemple dans les travaux de [Montfort, 1996], avec le rapprochement entre KOD et OOA [Shlaer et Mellor, 1991] pour le développement d'un simulateur de pêche.

3.2. KADS

KADS [Tansley et Hayball, 1993], acronyme de « *Knowledge Acquisition Design System* » est une méthode d'élicitation des connaissances basée sur des modèles et préconisant l'utilisation de la linguistique tant au niveau de l'acquisition qu'à celui de la modélisation.

KADS [14] dispose de diverses techniques de recueil des connaissances et de différentes méthodes ascendantes d'analyse des données recueillies. Ceci permet de décomposer un problème puis de séparer l'analyse de la connaissance de son implémentation. KADS se concentre sur l'étude de faisabilité et, surtout, sur la définition du produit et la conception du systèmes et des modules. La conception détaillée, l'implémentation, ainsi que les autres étapes du cycle de vie ne sont pas fondamentales pour KADS qui repose sur l'hypothèse qu'il n'y a pas de retour sur la phase d'analyse, à partir de la conception ou de l'implémentation.

[14] KADS est la méthodologie la plus répandue pour le développement de systèmes à base de connaissances en Europe.

La méthodologie KADS, pour développer des systèmes à base de connaissances, a essayé de résoudre le problème de représentation des connaissances en suggérant que les connaissances soient simultanément analysées selon différents niveaux et différents modèles que nous allons maintenant décrire succinctement :

- le niveau *domaine* décrivant la connaissance statique du domaine indépendamment de son utilisation. La connaissance est structurée en étudiant :
  - ♦ les concepts, termes fondamentaux pour la modélisation,
  - ♦ les structures qui permettent de lier les concepts et leurs attributs,
  - ♦ les relations associant des concepts ou des structures,
  - ♦ les modèles qui regroupent des relations autour d'un thème de résolution de problèmes.
- le niveau *inférence* qui contient des méta-connaissances portant sur le niveau domaine et décrivant la compétence fonctionnelle, c'est-à-dire les fonctionnalités (instanciation, classification, assemblage, comparaison) utilisées lors de la résolution de problèmes s'exerçant sur le domaine. Ces fonctions sont appelées « sources de connaissance » et leur arguments « méta-classes » ou « rôles » (comme par exemple une hypothèse, une question, une contrainte, un diagnostic ou un plan). Ces derniers indiquent comment un ensemble spécifique de concepts du domaine peut être utilisé. En combinant les sources de connaissances grâce à leurs arguments d'entrée et de sortie, il est possible de construire une structure d'inférences reflétant la compétence requise pour accomplir une tâche de résolution de problèmes.

Notons toutefois que, même si une telle structure décrit quelles inférences peuvent être créées, elle ne dit ni comment ni quand [Montfort, 1996].

- le niveau *tâche* décrivant les moyens pour atteindre un but lors de la résolution de problèmes. Il contient des buts décomposables en sous-buts et des tâches structurées. Le niveau tâche spécifie quelles inférences doivent être déclenchées, suivant le plan créé au niveau stratégie.
- le niveau *stratégie* qui décrit les plans de résolution du problème en fonction du contexte et des compétences.

KADS propose une bibliothèque de modèles d'interprétation. Fonctionnant comme une interface entre les données verbales brutes et l'implémentation, un modèle d'interprétation peut servir de modèle initial pour guider le cogniticien dans l'analyse des données (par exemple par combinaison ou sélection des modèles existants). Ces modèles d'interprétation sont très utiles pour planifier et mener des entretiens puis pour interpréter les données verbales recueillies. L'avantage des modèles d'interprétation par rapport à l'implémentation d'une maquette est qu'aucun engagement détaillé n'a lieu par rapport au langage d'implémentation.

Les trois modèles du processus d'analyse de KADS sont les suivants :

- le modèle des *processus* montre l'ensemble des processus organisationnels actuels et futurs. Il se présente sous la forme de tâches à exécuter par le système ou les acteurs.
- le modèle de *coopération* reflète comment le système futur va interagir ou coopérer avec les autres composantes internes du système.
- le modèle d'*expertise* décrit par quels moyens s'effectue la résolution de problèmes ou le comportement de l'expert au niveau des composantes du futur système.

3.3. CommonKADS

CommonKADS est le nom de la méthodologie développée par le projet KADS-II qui a été fondé avec le programme CEC ESPRIT [Wielinga, 1993]. CommonKADS considère le développement de systèmes à base de connaissances comme un processus de modélisation. L'analyse des connaissances est réalisée en créant un nombre de modèles qui représentent les connaissances selon différents points de vue. CommonKADS recommande un certain nombre de modèles différents, qui se basent sur la représentation des différents aspects d'une organisation, et supporte le processus entier d'ingénierie des connaissances jusqu'à la production d'une spécification détaillée.
En effet, la méthodologie KADS pour le développement des systèmes à base de connaissances a essayé de résoudre le problème de l'acquisition des connaissances en suggérant que les connaissances soient représentées et analysées simultanément selon différents niveaux [Tansley et Hayball, 1993]. CommonKADS a étendu et raffiné ces représentations recommandées. En particulier, CommonKADS a introduit une classification ontologique pour les connaissances du domaine [Wielinga, 1993].
CommonKADS vise à fournir une approche détaillée et réutilisable pour l'analyse des connaissances requises. Il est par conséquent important de comprendre comment les résultats de différentes techniques d'élicitation des connaissances pourraient être appliquées (« *mappées* ») sur les modèles CommonKADS.

Le modèle clé, le *modèle d'expertise*, est divisé en trois niveaux représentant différents points de vue sur les connaissances expertes :

- les *connaissances du domaine* représentent les connaissances déclaratives dans la base de connaissances ;
- les *connaissances d'inférence* représentent les inférences basées sur les connaissances et qui sont exécutées durant la résolution du problème ;
- les *connaissances des tâches* définissent un ordonnancement procédural sur les inférences.

Les contenus de ces trois niveaux peuvent être définis graphiquement, ou en utilisant le langage de modélisation conceptuelle de CommonKADS.

Les connaissances du domaine dans le modèle d'expertise représentent les connaissances déclaratives qui ont été acquises. CommonKADS suggère que chaque élément des connaissances déclaratives soit classé dans un des quatre types d'ontologie suivants :

- les *concepts* : ce sont des classes d'objets du monde réel ou mental appartenant au domaine étudié et représentant des objets physiques ou des états ;
- les *propriétés* : ce sont les attributs des concepts ;
- les *expressions* : ce sont des formulations de la forme « la *propriété* du *concept* est *valeur* » ;
- les *relations* : ce sont des liens entre deux éléments des connaissances du domaine.

Une fois que les éléments des connaissances du domaine ont été classés, ils peuvent ensuite être utilisés dans les *modèles du domaine*, qui montrent les relations entre différents éléments de connaissance. Par exemple, un modèle du domaine pourrait montrer tous les exemples acquis d'un concept étant la *cause* d'un autre; ou bien il pourrait montrer une hiérarchie taxonomique de concepts, connectés entre eux par des relations de type « *est-un* ».
Cependant, il n'y a pas de techniques d'acquisition des connaissances qui génèrent des sorties dans une forme directement utilisable en entrée des modèles CommonKADS. A la place, les techniques d'acquisition des connaissances produisent typiquement des transcriptions textuelles, ou des catégorisations des termes de domaine sur plusieurs dimensions différentes. Ceci signifie que le cogniticien classe ces éléments en ontologie de domaine de CommonKADS. Ceci est souvent un travail coûteux ; la principale difficulté est basée sur le fait que CommonKADS fournit peu de supports pour identifier les connaissances pertinentes, ou pour classer les connaissances en ontologies.
[Kingston, 1994] pense alors qu'il est possible d'automatiser la plupart du travail d'identification et de classification des connaissances du domaine en identifiant et en exploitant la structure des connaissances initiales.
Dans ce domaine, des travaux de recherche ont déjà été effectués ; notamment sur la génération de règles de production à partir d'une grille de répertoires [Shaw et Gaines, 1987], et sur la production d'une trame logique commune pour la communication entre différentes techniques d'acquisition des connaissances [Reichgelt et Shabolt, 1991]. Cependant, personne n'a encore essayé d'utiliser la structure des connaissances initiales afin d'opérer les classifications requises pour l'ontologie du domaine CommonKADS.

[Kingston, 1994] décrit comment une telle classification peut être opérée avec une boite à outils TOPKAT (acronyme de « *The Open Practical Knowledge Acquisition Toolkit* ») qui basée sur la notion d'hypertexte et destinée aux cogniticiens. TOPKAT a été développée par le laboratoire d'intelligence artificielle d'Edimbourg AIAI, acronyme de « *Artificial Intelligence Application*

Institute », et est implémentée avec HARDY et wxCLIPS [15]. Notons toutefois qu'à l'heure actuelle, TOPKAT demeure seulement un prototype sophistiqué.

Nous pouvons également préciser à ce niveau qu'il existe d'autres outils comme le système SHELLEY [Anjewierden *et al.*, 1992] supportant KADS, la méthode des tâches génériques (« *Generic Tasks* ») de [Chandrasekharan, 1987], ou encore MACAO [Aussenac, 1989].

Nous avons dressé un état de l'art succinct des principales approches relatives à l'acquisition et à la gestion des connaissances.
Nous allons dans le cahpitre suivant démontrer que la construction des connaissances relatives au domaine de l'enseignement doit respecter un processus rigoureux et cohérent, dans le cadre d'une véritable ingénierie des besoins, pour déboucher sur une spécification formelle pouvant servir de base sûre et fiable à la suite du cycle de développement des connaissances.

[15] Nous reviendrons ultérieurement dans ce document sur ces outils logiciels que nous utilisons aussi pour implémenter notre prototype.

Chapitre 4 : La spécification des enseignements

En juillet 1996, un des fleurons de l'industrie spatiale française, ARIANE 5, a explosé quelques secondes après le décollage. Observons brièvement les raisons de cette autodestruction.
Il s'agit tout simplement d'une conversion, non protégée au niveau du système de référence d'inertie, d'un flottant 64 bits en entier 16 bits… La valeur était trop importante, ce qui a déclenché une exception transmise (comme toutes les données au système de référence d'inertie) à l'ordinateur de bord (le coeur qui contrôle le programme de vol). Ce dernier a pris cette donnée comme une donnée normale, ce qui a provoqué un comportement aberrant et le déclenchement de la procédure d'autodestruction.
Il faut cependant noter que ce code servait déjà sur ARIANE 4 et qu'il n'y avait jamais eu de problème car la valeur en question, liée à la vitesse horizontale, était beaucoup plus faible sur ARIANE 4, ce qui ne posait aucune difficulté de conversion. Les responsables n'avaient pas jugé utile de revoir la spécification pour le passage de ARIANE 4 à ARIANE 5…
Petit détail amusant, le calcul à l'origine de l'erreur ne sert, en fait, qu'avant le décollage et il continue ensuite pendant une quarantaine de secondes. C'est aussi une spécification de ARIANE 4 qui ne s'avère en fait pas utile pour ARIANE 5. De même il y a un système de référence d'inertie de secours mais qui est doublée à l'identique (même hard et même soft)… Celui de secours avait produit la même erreur quelques millièmes de secondes plus tôt…
Le programme en question était pourtant écrit en Ada qui est un langage pouvant détecter des erreurs de conversion de type. Il avait été spécifié par la défense des USA pour cette sécurité entre autres afin que, lorsque ce type d'erreur se produit, Ada lance une autre fonction censée réagir en conséquence. Or, les programmeurs n'avaient pas implémentés une telle fonction réparatrice (bien que d'autres variables en disposaient dans le programme).
La question qui se pose alors est : « A qui en incombe la faute ? ».
Ce n'est pas la faute des programmeurs qui ont suivi à la lettre les spécifications (qui étaient en fait fausse); ce n'est pas la faute des auteurs des spécifications (qui travaillaient pour ARIANE 4 laquelle a une trajectoire où l'erreur ne peut se déclencher); ce n'est pas la faute des personnes testant les programmes au sol car ce n'est pas le programme en question qui a été testé sur la trajectoire d'ARIANE 5, mais une simulation (les auteurs du rapport ont reproduit la même erreur en testant le programme réel sur la trajectoire d'ARIANE 5).

Que penser de cet exemple ? Tout simplement qu'une erreur de spécification, c'est-à-dire au départ dans le processus de développement, peut avoir des conséquences énormes et très coûteuse pour le produit final en fin de cycle.

Nous allons étudier dans ce chapitre quels sont les moyens à mettre en oeuvre pour aboutir à la spécification d'enseignements plus conformes aux desiderata initiaux des pédagogues.

Paragraphe 1 : Importance de l'ingénierie des besoins

Le champ communément appelé « Analyse des systèmes » a été appliqué en premier aux systèmes d'information, et a par conséquent eu une orientation applicative et organisationnelle. Le domaine de l'ingénierie des besoins (« *requirements engineering* ») [Siddiqi et Sherakan, 1996] vise à incorporer un aspect ingénierie à l'analyse des systèmes.

Les produits les plus largement connus, et peut-être les plus significatifs, de cet aspect ingénierie sont le développement de méthodes et de leurs outils d'automatisation associés. Malheureusement, beaucoup de ces méthodes prescriptives portent peu ou pas d'attention à la façon dont le contexte influe sur la décomposition et l'évolution. Ainsi, les personnes dont l'intérêt se focalise sur le contexte trouvent ces méthodes inadéquates. La frontière existant entre la pratique et la recherche est donc encore très importante. La prudence conventionnelle relative à l'ingénierie des besoins évolue toutefois rapidement et les recherches les plus récentes prennent en compte la notion de *contexte*.

Les développements en ingénierie des besoins suivent les grandes lignes suivantes dans le développement des systèmes. Dans la première vague des développements de systèmes, l'accent était porté sur l'écriture de *code*. Le développement des petits et gros systèmes était alors perçu comme une activité simple et non comme un processus organisé en plusieurs étapes. L'étape suivante a vu l'introduction de la notion de *cycle de vie* dont la phase d'élaboration du *cahier des charges* était la première. Ensuite, vint l'adoption des modèles de *développement incrémentaux* et l'approbation du fait que l'implémentation s'effectue souvent à partir de spécifications incomplètes. Ainsi, l'évolution de l'ingénierie des besoins a bénéficié à la fois des paradigmes issus des Systèmes d'Informations et du génie logiciel. A l'heure actuelle, c'est un ensemble de techniques issues d'un mélange de ces deux approches qui est utilisé et la suprématie d'un paradigme sur l'autre demeure impossible à affirmer. Par conséquent, comme nous l'avons vu dans [Siddiqi et Sherakan, 1996], trois questions fondamentales restent à l'esprit :

- Qu'est ce qu'un besoin ?
- Quelles activités doivent être incluses dans l'ingénierie des besoins ?
- Vers où, dans la pratique, appliquer les recherches ?

1. Qu'est ce qu'un besoin ?

C'est l'état de ce qu'un système devra faire sans se référer à comment il devra le faire. Toutefois cette étape reste liée à celle de conception mais la distinction demeure pour éviter les chevauchements de tâches des différents auteurs de chaque phase.

Une autre distinction commune réside dans la séparation entre les besoins fonctionnels (ou comportementaux) et non fonctionnels [16]. Dans le passé, la plupart des chercheurs ont concentré leurs travaux sur les besoins fonctionnels. L'article de [Boehm, 1996] reflète les dernières tendances vis à vis des besoins non fonctionnels. Depuis quelques temps, la communauté des chercheurs du domaine s'accorde sur la nécessité d'élargir la vue sur les besoins pour considérer le contexte dans lequel le système évoluera. Les travaux de [Borgida *et al.*, 1985] sur la modélisation conceptuelle en tant que base de l'ingénierie des besoins constituent une indication majeure pour les chercheurs dans ce domaine. L'article de [Nissen *et al.*, 1996] se réfère aux expériences récentes dans l'application des techniques de modélisation conceptuelle.

Plus récemment, [Jackson, 1995] a avancé une autre façon de considérer le contexte. Il pense que les défauts dans le développement actuel des logiciels reposent sur les caractéristiques et la structure de la solution plutôt que sur le problème. Un logiciel, selon lui, est la description d'une machine souhaitée, et son développement implique la construction de cette machine. Les besoins reposent sur les objectifs, et le but d'une machine se trouve en dehors de la machine elle-même, dans le contexte du problème. Ainsi, [Jackson, 1995] préconise un déplacement vers une approche orientée-problème qui aura pour but de distinguer différentes caractéristiques et structures dans le domaine d'application. [Jackson, 1995] affirme qu'une méthode ne peut être puissante que si elle exploite les fonctionnalités spécifiques du problème, et comme celles-ci sont très nombreuses, il faut un répertoire de méthodes, chacune étant configurable pour une certaine classe de problèmes. Cette vue porte la connaissance, à la fois de l'expert du domaine et de l'analyste, au coeur de l'ingénierie des besoins.

Pour [Goguen, 1996], les besoins ne sont pas « *volatiles comme des papillons* ». Ce n'est pas le cas non plus le cas du travail de l'analyste qui doit trouver une façon convenable de capturer ces besoins. Ceux-ci émergent des interactions sociales entre les utilisateurs du système et l'analyste.

2. Les activités de l'ingénierie des besoins

La plupart des professionnels du génie logiciel pensent que la phase de rédaction du cahier des charges possède son propre cycle de vie dont les étapes possèdent différents labels.

[16] Cette distinction n'est toutefois pas toujours évidente au premier abord pour toutes les applications car certaines exigences qui apparaissent comme non fonctionnelles dans un premier temps, peuvent finalement s'avérer comme telles.

[Krasner, 1980] a identifié cinq étapes : l'identification des besoins et l'analyse du problème, la détermination des besoins, la spécification des besoins, la satisfaction des besoins et la gestion des changements des besoins. Plus récemment, [Jarke et Pohl, 1994] a proposé un cycle en trois étapes : élicitation, expression et validation.

Cependant, les phases peuvent être découpées en une série itérative. Le « *Inquiry Cycle Model* » proposé par [Potts *et al.*, 1994] repose sur cette vue qui intègre trois phases (documentation, discussion et évolution) et des garde-fous qui utilisent des scénarios pour identifier et valider les besoins. La validation est accomplie par des agents surveillant les besoins exprimés dans le but d'obtenir une justification plus claire des besoins. Sur les bases de cette validation, les agents peuvent annihiler ou changer un besoin dans la phase finale.

La communauté de génie logiciel a concentré ses efforts sur la phase d'analyse des problèmes, ce qui est aussi le cas de beaucoup de méthodes de décomposition. En général, ces méthodes sont conçues pour aider l'analyste à définir l'éventail de l'ensemble des solutions possibles. Plus spécifiquement, selon [Davis, 1990], la phase d'analyse des problèmes inclut l'apprentissage du problème, en comprenant les besoins des utilisateurs potentiels, en découvrant qui est vraiment l'utilisateur, et en comprenant toutes les contraintes de la solution. En sortie, le document de spécification des besoins est censé être une description complète du comportement externe du produit final.

Dernièrement, ce paradigme de décomposition a été étendu pour proposer que les systèmes puissent être construits à partir d'un répertoire standard de composants. [Jarke et Pohl, 1994] a suggéré, par exemple, que les projets de l'ingénierie des besoins qui sont réalisés d'un seul trait soient remplacés par une pratique qui rassemble les composants de façons innovantes plutôt que de continuer à réinventer en permanence les composants eux-mêmes. En effet, le plus gros défaut de la vue réductionniste (qui consiste à partitionner les choses en des parts plus petites) est que le contexte influera sur la décomposition. La juxtaposition de la vision et du contexte doit être au coeur de la gestion des besoins.

L'ingénierie des besoins est donc utilisée comme un exercice contractuel dans lequel le client et le développeur travaillent pour se mettre d'accord sur un état précis en non ambigu de ce que le développeur voudrait construire.

Ceci nous amène donc à la conclusion qu'il convient d'adopter une démarche rationnelle pour l'identification des besoins. Nous allons maintenant considérer dans quelle mesure les besoins initiaux des utilisateurs doivent se retrouver au sein du modèle de représentation des connaissances sous la forme d'une spécification formelle.

Paragraphe 2 : Vers une spécification formelle des connaissances

Une spécification formelle de logiciel est une spécification exprimée dans un langage dont le vocabulaire, la syntaxe et la sémantique sont définies formellement, et qui repose sur une base mathématique, couramment la logique formelle.

Les spécifications d'un système sont d'ordinaire présentées dans un mélange de langage naturel, de notations graphiques et de pseudo-code. Une spécification formelle, par contraste, utilise un langage mathématique précisément défini pour décrire les propriétés du système. Ceci laisse moins de place aux ambiguïtés dans le cahier des charges, et ouvre également la porte au contrôle des erreurs et des inconsistances. Les langages de spécification formelle fournissent en général une variété de constructeurs abstraits de haut niveau qui permettent la description des propriétés du système sans se préoccuper des détails de l'implémentation.
Sans une large audience, les nouvelles techniques peuvent être considérées comme trop immatures pour une adoption commerciale. Ainsi, introduire un nouveau processus de développement est délicat et difficile. Il est plus acceptable de passer par un changement incrémental dont les effets peuvent être identifiés et quantifiés.
[Gorm Larsen *et al.*, 1996] a confronté le développement en parallèle par deux équipes distinctes d'un système de traitement de messages. Une équipe a employé un processus conventionnel en utilisant des outils d'assistance par ordinateur tandis que l'autre équipe a opté pour un processus similaire mais utilisant en outre des spécifications formelles lorsque cela était approprié. En fin de compte, l'équipe ayant utilisé des spécifications formelles s'est avérée à la fois plus rapide et plus efficace dans le développement final.
Toutefois, [Lawrence, 1996] pose la question suivante : « *Avons-nous vraiment besoin de spécification formelle pour exprimer les exigences d'un système ?* ».
Les paragraphes suivants éclairent notre pensée.

1. Objectifs

1.1. Un premier objectif : la communication

Non seulement il est nécessaire de mieux comprendre les problèmes des clients, mais ils doivent eux aussi mieux savoir ce qu'ils désirent.
Les méthodes formelles utilisent une notation précise, souvent mathématique, que les clients ne maîtrisent pas d'ordinaire. En outre, elles sont claires et bien définies mais seulement pour ceux qui les comprennent. Les méthodes informelles utilisent le langage naturel dont le principal avantage est d'être à la fois familier aux développeurs et aux clients.
Les risques d'incertitude et d'ambiguïté non résolus sont dus au fait que d'importants acteurs ont des points de vue très divergents et accèdent à différentes informations ; par conséquent, ils ont différentes

compréhensions d'un même problème. Les méthodes à base d'heuristiques laissent chacun, développeurs, gestionnaires de projet, utilisateurs finaux, etc…, s'impliquer dans le développement de la spécification sans avoir à apprendre une notation abstraite. Une spécification des besoins écrite en langage naturel par un large panel de participants réduit l'ambiguïté au niveau le plus important (en terme de coût) : en se mettant d'accord sur un ensemble représentatif d'utilisateurs, d'attributs et de fonctions, on diminue les risques d'incertitude concernant le devenir du produit.

1.2. Un second objectif : une bonne conception

Les méthodes informelles peuvent produire des spécifications claires servant de base à la phase de conception. La combinaison de méthodes d'expression informelles des besoins avec un processus de critique solide est une façon très efficace pour s'assurer de la production d'une application adéquate et de bonne qualité.

Il est possible d'avoir des difficultés à utiliser des méthodes formelles tout simplement car elles sont trop formelles. Il est facile de dire « Soyez plus discipliné dans votre approche » mais il est plus délicat de le faire. Les techniques comme l'inspection, dont les bénéfices sont bien documentés, sont extrêmement difficiles à mettre en oeuvre même pour quelqu'un d'expérimenté.

Les méthodes de spécification formelle promettent beaucoup grâce à une meilleure consistance et une plus grande automatisation de tâches comme la mise en place de tests. Dans un tel cas, les bénéfices peuvent être diminués par une trop grande rigueur du formalisme qui peut exclure des utilisateurs potentiels. En effet, il faut être compétent avant d'être efficace. Avant de se lancer dans le cycle de développement, il faut bien s'assurer de la compréhension correcte des besoins du client. La résolution du bon problème est plus efficace car cela évite le gaspillage en terme de coûts de résolution d'un mauvais problème.

Il ne faut pas perdre de vue le fait d'utiliser à la fois des méthodes formelles et informelles. Il est possible d'utiliser des heuristiques pour développer une première extraction des besoins, puis de traduire ces informations en une notation formelle pour tirer parti de ce formalisme ; l'idée profonde demeurant toujours d'avoir une meilleure compréhension générale de ce que l'on est en train de faire.

Aussi, selon [Jackson, 1996], la spécification des besoins, mise en forme, peut être une formalisation.

Le principal objectif de la phase d'établissement du cahier des charges est d'élaborer un produit (le cahier des charges) qui n'est pas moins important que le code final. Investir dans le travail de compréhension par l'analyste du problème du client et non dans le travail d'expression écrite est à la fois risqué et coûteux. A moins que les concepteurs et les réalisateurs soient partie prenante dans les discussions des analystes, les mauvaises interprétations seront courantes, le fait de sous-traiter sera gênant, et la disparition inévitable des analystes rendra cauchemardesque la phase de maintenance.

Ainsi, nous pouvons nous demander ce à quoi le cahier des charges devrait ressembler. L'expérience prouve qu'un document libre de forme, rédigé en langage naturel, reflète trop pauvrement les investigations dues au travail de l'analyste. Ici, les méthodes formelles ont beaucoup à apporter.

1.3. Objectifs connexes

Il est difficile d'être sûr qu'une bonne compréhension des idées a été atteinte. Le fait d'enregistrer votre compréhension avec une notation formelle évite bien de désagréables surprises. Mais, bien que la traduction en une notation formelle soit pratique, le principal bénéfice est tiré lorsque la compréhension du problème issue de la formalisation retourne dans la discussion avec le client, chaque mauvaise surprise amenant de nouvelles discussions sur des points de confusion ou de doute. La formalisation, non seulement est demandeuse de plus d'informations, mais elle suggère aussi des façons d'extraire et de structurer plus efficacement ces informations. La notation formelle est également l'outil de l'analyste car elle permet aussi de guider le dialogue avec les clients, même ceux qui ne voient que les descriptions informelles. La question qui se pose alors est de savoir si les analystes eux-mêmes peuvent apprendre une notation formelle. Il y a bien évidemment des barrières culturelles.
La *clarification des idées* est donc un objectif atteint par la formalisation d'une spécification.

En outre, les modèles mathématiques simplifient l'état d'un problème par les informations qu'ils excluent. En effet, le problème ne réside pas dans la façon dont les abstractions de descriptions sont effectuées, mais plutôt dans la façon dont les modèles clarifient exactement les détails qui doivent être ignorés et les nuances du langage naturel rendent ceci difficile à exprimer. De plus, un modèle formel, en réduisant un problème à son essence, permet de le reconnaître facilement et naturellement. La recherche des fonctionnalités uniques d'un problème et, inversement, l'identification des opportunités de réutilisation, dépend alors de la précision de la caractérisation de l'essence des problèmes déjà rencontrés.
La *simplicité par omission,* qui amène aussi une *extraction des connaissances de base,* constitue donc également un avantage des spécifications formelles.

Enfin, une description formelle peut être *analysée de façon mécanique*. En effet, pour comprendre le problème du client, ceci aide à découvrir les conséquences de ce qui a été mis en évidence. A Carnegie-Mellon, ils développent des outils qui détectent des scénarios dans lesquels une spécification n'influe pas sur les intentions premières de l'analyste. Un consensus peut être trouvé, mais s'il s'agit alors d'implanter des exigences inconsistantes ou ayant des conséquences inattendues, le gain est minime. Les méthodes formelles n'apportent de grands bénéfices que si elles sont utilisées de façon discriminatoire. Comme toute ingénierie d'analyse, il n'y a pas de sens à se lancer dans une

formalisation complète sans objectif précis. Il est nécessaire de décider tout d'abord où résident les principaux risques de mauvaise compréhension et ensuite de formaliser uniquement ce qui est nécessaire pour minimiser ces risques. La compréhension du client est une chose nécessaire mais pas suffisante car il convient aussi d'être capable d'enregistrer cette compréhension. En portant une attention particulière, non seulement aux facteurs humains, mais aussi à la structure et à l'expression des besoins, il ne sera plus nécessaire d'organiser des projets importants autour de grosses équipes devant rester indissociables jusqu'à la fin du projet.

Nous avons présenté les principales raisons pour lesquelles une spécification doit être formelle. Nous allons maintenant étudier les différentes façons d'aborder la formalisation et nous donnons ensuite des exemples.

2. Classification

Selon [Wirsing, 1993], trois approches principales permettent d'aborder les spécifications formelles :

- *la méthode des assertions*. Elle a été développée dans les années 65 à 70 par Floyd, Dijskstra et Hoare pour la vérification de programmes impératifs. En utilisant la notion d'état, nous déterminons les pré- et post-conditions d'un programme. Beaucoup de langages utilisent ces idées et notamment Z, VDM, COLD ou RAISE.
- *la méthode algébrique axiomatique*. Elle puise ses origines dans les travaux de Guttag et Adj en 75. Leur but de départ était la description abstraite de structures de données se basant sur la logique équationelle. Les langages principaux pour cette approche sont CLEAR, OBJ, PLUSS, ASL et SPECTRUM.
- *la théorie des types*. Elle est basée sur la logique constructive, le lambda-calcul de second ordre et a été développée au début des années 70 par Martin-Løf et Girard. Le premier logiciel a été développé dans les années 70 par de Bruijn pour appuyer la description formelle des mathématiques. Autour de 85, les systèmes LF et DEVA ont été développés.

[Fraser *et al.*, 1994] abordent ce problème, non pas selon les différents langages ou les différentes approches évoqués précédemment, mais sur les stratégies et outils d'incorporation de spécifications formelles dans le cycle de vie.
Ainsi, une classification selon trois axes est proposée :

① *Degré d'assistance*

- *Formalisation libre*. Le processus repose ici sur les capacités innées de l'utilisateur qui formalise.

- *Formalisation assistée par ordinateur*. Le processus de résolution humaine est supporté par des méthodes utilisant les heuristiques, l'algorithmique et les bases de connaissance.

② *Degré de formalisme*

- *informel*. Ces techniques n'ont pas un ensemble de règles pour contraindre les modèles qui peuvent être crées. Le langage naturel et les graphiques non-structurés en sont des exemples typiques.
- *semi-formel*. Ces techniques ont des syntaxes définies (Exemples : descriptions textuelles et graphiques avec des facilités de contrôle limitées - SADT, SA/SD, …)
- *formel*. Ces techniques ont des syntaxes et des sémantiques rigoureusement définies. Il y a un modèle théorique sous-jacent au moyen duquel une description exprimée en notation mathématique peut être vérifiée (Exemples : langages de spécification basés sur la logique des prédicats, Réseaux de Pétri, …).

③ *Processus de formalisation*

- *Directe*. La formalisation des spécifications informelles est réalisée sans passer par des représentations intermédiaires (semi-formelles).
- *Transitionnelle* (par représentations intermédiaires)
  - ♦ *Séquentielle*. Les spécifications formelles sont dérivées à partir d'un ensemble final de modèles semi-formels.
  - ♦ *Parallèle par raffinements successifs*. Les spécifications formelles et semi-formelles sont produites par des raffinements successifs en parallèle.

Nous allons maintenant proposer ci-dessous quelques exemples relatifs à cette classification.
Dans les stratégies « libres » telles celles préconisées par [Kemmerer, 1990], ce sont les concepteurs qui formalisent en faisant éventuellement le lien entre les différentes représentations. Ceci s'avère coûteux en temps, donc en moyens, et repose beaucoup sur leurs compétences et sur des heuristiques de transition. De telles stratégies sont appropriées pour des problèmes initiaux succincts et bien structurés.

Les processus de formalisation « assistée par ordinateur » concernent des problèmes non triviaux qui nécessitent des validations et correspondent davantage à des utilisateurs non-experts en mathématiques. [Rosca, 1994] traite d'un type spécial de connaissances : les décisions et le processus de prise de décision dans le cycle de vie. Son objectif est de réduire la différence entre les approches manuelles et complètement automatisées pour capturer et utiliser la structure de décision à l'aide de mécanismes de raisonnement par analogie.
Il existe plusieurs exemples de stratégies « directes » assistées. [Harandi et Miriyala, 1991] définissent « *SPECIFIER* » qui permet une formalisation du système par dérivation de concepts, utilisation de

l'analogie et du raisonnement basés sur la différence à partir du langage « naturel » (anglais simplifié). [Aubord et Ibrahim, 1991], [Biebow et Szulman, 1991] et [Girardi et Ibrahim, 1993] proposent des approches d'un système de programmation automatique basé sur l'analyse lexicale, syntaxique et sémantique de spécifications exprimées en langage « naturel ». [Reubenstein et Waters, 1991] élaborent un système, appelé « *Requirements Apprentice* », permettant la formalisation de spécifications en langage naturel simplifié à l'aide de bibliothèques de cas, d'un cadre d'exécution et du « *Cake* » gérant le raisonnement. D'autres exemples comme ARIES, KATE et IRA sont abordés dans [Rosca, 1994].
Cependant, de telles stratégies demeurent encore trop opportunistes. En effet, la mise en œuvre d'un tel environnement nécessiterait une trop grande masse de travail dans l'acquisition de multiples variétés d'analogies et l'encodage de schémas constructifs [Vogel, 1991]. Dans l'avenir, cependant, ces stratégies devront être prises en compte dans le cas de problèmes de bas niveau syntaxique.

Nous nous focaliserons donc sur le processus de formalisation transitionnelle assistée par ordinateur qui repose sur l'utilisation de représentations semi-formelles intermédiaires. L'aide des outils informatique apporte ici de nombreux avantages :

- L'utilisateur est assisté dans sa tâche de modélisation ou structuration du problème ainsi que dans la traduction vers un autre formalisme. Ceci réduit alors les risques d'erreurs humaines ainsi que le coût de ce travail et notamment en personnel hautement qualifié, comme dans le cas de formalisation « libre ».
- Les problèmes de synchronisations sont évités dans le cas de formalisation transitionnelle parallèle.
- La remise en cause du travail déjà effectué est réalisable, d'où des possibilités accrues de prototypage.

Les stratégies de formalisation transitionnelle séquentielle partent d'un ensemble de spécifications semi-formelles qui sont ensuite formalisées par un outil informatique avec néanmoins des appels interactifs à l'utilisateur pour des compléments de précision. Ici, l'assistance se traduit par des algorithmes [17] ou des systèmes à base de connaissance. Ce type de processus de formalisation convient aux situations où les besoins détaillés et complets sont initialement mis en avant ou aisément détectables. Dans [Einsenstadt *et al.*, 1990], les auteurs mettent en relief la nécessité de la programmation visuelle et préconisent une représentation de programmes à l'aide d'hypertextes, de graphes, de tables et de règles d'écriture. Dans [Tue Ho et Vogel, 1988], les auteurs proposent une

[17] Notons ici la distinction entre heuristique et algorithme. Une méthode heuristique, telle que le défini le Petit Robert, est fondée sur l'approche progressive d'un problème donné avec des hypothèses provisoires. Elle diffère

gestion de configuration logicielle ayant une architecture logicielle en couches successives gigognes. [Fraser *et al.*, 1991] présentent des spécifications formelles générées à partir de spécifications en « Analyse Structurée » (« SA » ou « *Structured Analysis* » de [De Marco, 1979]).

Par contre, dans le cas des stratégies de formalisation transitionnelle parallèle, des représentations formelles et semi-formelles synchronisées du système sont produites par raffinements successifs parallèles des spécifications. A chaque nouveau niveau, l'apport de l'outil informatique permet la dérivation vers une spécification formelle correspondante. Celle-ci, en retour, peut être formellement comparée avec la spécification initiale. Ainsi, la formalisation transitionnelle parallèle permet aux spécifications semi-formelles et formelles intermédiaires de s'entraider de façon synergétique durant le processus de découverte des besoins et de raffinements.

Ce type de stratégie de formalisation est davantage approprié aux types de problèmes peu définis et peu structurés mais selon [Fraser *et al.*, 1994] de telles stratégies de formalisation n'ont pas été répertoriées.

Une synthèse est présentée dans le tableau suivant :

				Degré d'assistance	
				Libre	*Assisté par Ordinateur*
Degré	*Absent*	**Processus**	*Direct*	[Kemmerer, 1990]	[Harandi et Miriyala, 1991], [Aubord et Ibrahim, 1991] [Biebow et Szulman, 1991], [Girardi et Ibrahim, 1993], [Reubenstein et Waters, 1991]
de formalisme intermédiaire	*Semi-formel*	**de formalisation**	*Séquentielle*		[Einsenstadt *et al.*, 1990]
			Transitionnelle	[Kemmerer, 1990]	[Fraser *et al.*, 1994], [Tue Ho et Vogel, 1988]
			Parallèle par raffinements successifs	[Kemmerer, 1990]	***? ? ?***

Tableau 2 : Taxonomie des stratégies pour la production de spécifications formelles

donc des méthodes formelles qui reposent sur des bases effectives et démontrables (des algorithmes par exemple).

En résumé, l'utilisation de méthodes de spécification formelle conduit à des spécifications de grande qualité pour de multiples raisons :

- Elle améliore la mise en œuvre et la compréhension des besoins, aide à clarifier les souhaits des clients en révélant ou en évitant les contradictions et les ambiguïtés dans les spécifications, permet la vérification rigoureuse des spécifications et leur implémentation, et facilite le passage vers la réalisation [Choppy, 1988].
- Elle peut aboutir à un prototype qui sera évalué à la fois par les ingénieurs et les utilisateurs finaux.
- Les spécifications formelles peuvent être analysées par des outils mathématiques (sous forme de procédures) au niveau de la *syntaxe* (analyse syntaxique) et de la *complétude* dans le sens où la spécification inclut tous les composants, éléments et options du système qui ont été effectivement énumérés et non pas « pensés » ou « souhaités » par l'auteur.
- Les spécifications formelles facilitent la phase de test.
- Un support dans les domaines d'activité de développement comme le soutien des transformations et la vérification des implantations, l'analyse automatique de la complexité des programmes, la construction et la recherche de composants logiciels réutilisables [Wirsing, 1993].

Il convient cependant de fournir des outils appropriés aux utilisateurs car ces derniers ne sont pas forcément des experts en mathématiques ou en logique et, ainsi, il devient possible de prouver que l'implémentation correspond à la spécification. Selon [Wirsing, 1993], « *écrire des spécifications, c'est faire un pas dans la programmation de très très haut niveau* ».

3. Apports du génie logiciel

Il y a beaucoup de définitions du génie logiciel. Presque tout auteur d'un livre de génie logiciel fournit sa définition propre. De façon succincte, nous pouvons dire que le génie logiciel est l'application des principes informatiques et mathématiques dans une approche systématique, disciplinée et quantifiable du développement et de la maintenance de logiciels.
Le but du génie logiciel est d'obtenir des logiciels de haute qualité. Il y a beaucoup de facteurs qui déterminent la qualité d'un logiciel : la satisfaction du client, par la fiabilité, la correction, et la performance.

Acquérir des connaissances n'est pas un moyen suffisant en lui-même car il convient de le faire au sein d'un processus de développement structuré : le *cycle de vie*.

Les cycles de vie séquentiels, tel le *modèle en cascade* [Boehm, 1976], permettent une gestion de projet où les différentes phases et leur enchaînement sont faciles à planifier et à contrôler. En effet, dans de tels modèles, les activités sont fixes et séquentielles, ce qui permet de définir des critères précis de qualité et de réussite. Par contre, cela limite la créativité des développeurs qui est, par définition, non ordonnée.

Dans un modèle en cascade, plusieurs problèmes se posent :

- le développement est considéré comme un *problème technique* de traduction et d'enchaînement de phases,
- la *communication* entre les usagers n'intervient que lors de la définition du cahier des charges et en phase de ß-test d'où des inadéquations. L'intérêt du *prototypage* prend toute son importance ici car il offre une certaine « matérialisation » des souhaits plus concrète que des spécifications non exécutables,
- les phases de spécification et d'implémentation sont distinctes d'où des difficultés de *retour-arrière*. En effet, l'indépendance entre les différentes phases entraîne, à chaque modification ou mise à jour du logiciel, une remise en question partielle, voire complète, de chacune des phases.
- la phase de *maintenance* se situe bien souvent à l'extérieur du processus de développement, d'où un coût de la maintenance élevé car, jusqu'au codage, il n'y a aucun moyen d'avoir un aperçu du fonctionnement du logiciel. Ceci entraîne, très souvent, des malentendus entre le demandeur et le réalisateur du logiciel.

En outre, l'émergence des méthodes objets ainsi que la création de nouvelles applications concernant le multimédia et les sciences de l'éducation implique la remise en cause des modèles conventionnels en cascade ou en spirale et se recentre autour de la notion de *prototypage*.
Nous allons dans un premier temps fournir quelques définitions relatives au prototypage et nous présentons ensuite des outils de prototypage.

3.1. Quelques définitions

[Krief, 1992] définit les notions de *modèle,* de *prototype,* et de *prototypage.*

Modèle : Un modèle M est un couple {D, O} où D est une représentation symbolique ou représentation figurative d'un certain domaine et O un ensemble d'outils permettant la manipulation et l'exploration de D.

Prototype : Un prototype P est un modèle expérimental, c'est-à-dire un modèle du modèle M. En conséquence, P = {D, O} avec D_p une représentation de M et O_p un ensemble d'outils spécialisés pour la construction et la manipulation de D_p.

Ainsi, la réalité peut être perçue selon deux niveaux d'abstraction différents : le niveau *modèle* à partir duquel il nous est possible de *manipuler la* représentation d'une certaine réalité et le niveau *prototype* qui nous permet de *construire* cette représentation de *la réalité,* ainsi que les outils s'y rattachant.
A cette distinction entre modèle et prototype, correspond respectivement les attitudes de l'utilisateur et du concepteur d'un point de vue *cycle de vie* du logiciel.
Un logiciel est considéré comme *modèle* par un *utilisateur* et comme un *prototype* par un *concepteur* (qui a aussi une attitude d'utilisateur vis-à-vis des outils de génie logiciel).
D'où les définitions suivantes données par [Krief, 1992] :

- *Modèle logiciel* : C'est un couple {D, O} ou D est la représentation informatique des concepts du domaine traité et O un ensemble de procédures permettant la manipulation de D. Il sert de lien entre D et le monde extérieur, c'est-à-dire l'utilisateur du modèle.
- *Prototype logiciel* P : C'est un logiciel expérimental, c'est-à-dire un modèle d'un logiciel L. En conséquence, P = {D_p, O_p} avec D_p une représentation de L et O_p est un ensemble d'outils spécialisés pour la construction et la manipulation de D_p.
- *Prototypage* : C'est un processus de construction de logiciels ayant pour but l'obtention d'informations sur l'adéquation et la pertinence de la conception d'un logiciel par le concepteur. Le prototype est habituellement utilisé comme un précurseur à l'écriture d'un système informatique. Un prototype se distingue typiquement du produit final par le fait qu'il soit plus rapidement développé, plus facilement paramètrable et manipulable au détriment de l'efficacité et de la performance. Le prototypage est donc utilisé pour extraire rapidement des informations sur le logiciel à venir.

Nous allons maintenant étudier deux grandes familles d'outils de prototypage : les *générateurs d'application* et les *boîtes à outils*.

3.2. Les outils de prototypage

L'activité de prototypage a pour but de mettre *rapidement* entre les mains d'un expert un prototype *pertinent,* c'est-à-dire un prototype qui est propre à rendre compte de la structure et du comportement du logiciel prototypé. Pour atteindre cet objectif, le concepteur va s'entourer d'outils logiciels, appelés *outils de prototypage,* faciles d'utilisation, permettant une génération et une adaptation rapides des différentes versions du prototype.

Il existe une grande diversité d'outils de prototypage. Cette diversité est liée, d'une part, à la variété des situations dans lesquelles l'activité de prototypage peut s'appliquer et, d'autre part, à la diversité des prototypes. [Carey et Mason, 1983], auteur d'une étude générale sur les techniques, outils et méthodologies de prototypage considérait, dans le début des années 80, que la diversité des outils de prototypage reflétait le fait que le concepteur n'avait pas à sa disposition d'outils réellement adaptés à une méthodologie de développement orientée prototype.

Aujourd'hui, s'il est encore tôt pour parler de méthodologie de prototypage, nous pouvons toutefois remarquer que les environnements de prototypage ont une approche sensiblement identique, qui s'inspire fortement de l'approche objet [Gupta *et al.*, 1989].

Nous pouvons classer la plupart des outils de prototypage en deux grandes familles non disjointes : les générateurs d'applications et les boîtes à outils.

3.2.1. Les générateurs d'applications

Ils furent initialement développés comme des aides à la programmation, permettant au concepteur de décrire son prototype en remplissant, en complétant et en assemblant des *schémas préfabriqués (*graphiques ou textuels).

Par exemple, un générateur d'applications graphiques permet au concepteur de dessiner son application en assemblant des représentations graphiques symbolisant des schémas de programmation (boucle, conditionnelle, affectation, appel de sous-programme), de construction (bus, registre, multiplexeur, porte logique) [Gupta *et al.*, 1989], d'interface utilisateur (bouton d'activation, bouton radio ou à cocher, champ de saisi avec défilement, menu déroulant, séquencement de dialogues) [Hullot, 1986], [Brandeis et Kertesz, 1988], [Lewis *et al.*, 1989].

Chaque famille de schémas est généralement regroupée dans une base de données spécialisée dans laquelle le concepteur puise les composants de son prototype. Cela a pour conséquence d'unifier la présentation des applications résultantes.

La description terminée, le générateur complète automatiquement les détails d'implantation de l'application et génère un code exécutable.

L'avantage de certains de ces générateurs réside dans l'assistance constante fournie au concepteur. Cette assistance peut être implicite (accès à certains choix interdits en fonction des choix déjà effectués) ou explicite (documentation en ligne*)*.

L'inconvénient majeur de ce type de générateur réside dans le fait que leur efficacité est généralement liée à leur degré de spécialisation, c'est-à-dire que les générateurs les plus efficaces sont ceux qui ne couvrent qu'une partie restreinte d'un domaine précis, pouvant ainsi offrir des bases de données très complètes sur le domaine.

De plus, de par le séquencement des différentes tâches à accomplir, ces générateurs d'applications présentent un *procédé de construction* spécifique au type d'applications générées.

Ainsi, un générateur d'applications en gestion de fichiers commencera par interroger le concepteur sur le format des fichiers mis en œuvre puis sur les différents index qui leur seront appliqués. Les fonctions de telles applications étant parfaitement maîtrisées, le concepteur indiquera les fonctions qu'il veut voir apparaître dans son application (création de nouveaux fichiers, insertion, suppression de fiches, génération d'étiquettes).

3.2.2. Les boîtes à outils

Considérées comme des applications génériques, elles fournissent des outils généraux nécessaires au développement d'une application, telles que la gestion des fenêtres, la mise en œuvre de menus déroulants, la gestion des événements clavier, souris. Elles se présentent généralement sous forme de boîtes à *objets* dans lesquelles le concepteur puise les structures et les fonctions dont il a besoin pour construire son application.

Le formalisme utilisé pour implanter ces *boites à outils* hérite directement de concepts développés par la programmation par objet. Plus que de simples bibliothèques de fonctions et de structures, ces générateurs offrent, d'une part, un formalisme délibérément orienté *représentation des connaissances* et, d'autre part, une approche de la conception d'applications dirigées par les données, appelée *approche par objet.*

Ces *boîtes à outils* sont, pour la majorité d'entre elles, intégrées à un langage de programmation contrairement aux générateurs d'applications précédemment cités. Le concepteur a donc un accès direct aux *objets* préexistants, ce qui lui offre la possibilité de les adapter, de les modifier, de les réutiliser suivant les besoins du prototype.

L'inconvénient majeur de ce type de générateur générique réside dans l'absence de tout *procédé de construction.* Le concepteur est livré à lui-même. Son apprentissage se fait généralement en consultant le code des applications déjà existantes (par exemple : les structures utilisées pour telle représentation, le mécanisme mis en œuvre pour intercepter des événements clavier) et en utilisant les outils de son environnement de programmation :

- les *traceurs,* qui, pour un environnement de programmation, offrent la possibilité de visualiser en mode interactif certains aspects du déroulement dynamique du programme et de contrôler l'état du système à des instants bien déterminés. Citons par exemple ici les travaux de [Teitelman, 1978], [Rich, 1981] et [Wertz, 1985].
- les *vérificateurs de programmes* [Floyd, 1967], [Hoare, 1969], [Hantler et Icing, 1976], [King, 1976], les *évaluateurs symboliques* [Goossens, 1981], les évaluateurs partiels [Sestoft et Sondergaard, 1988] et les *éditeurs syntaxiques* [Donzeau Gouge *et al.*, 1980].

A l'heure actuelle, il existe bon nombre de formalismes *pertinents* pour construire des prototypes ; seuls manquent des outils permettant d'utiliser pleinement et *rapidement* ces formalismes.

4. Conclusion

La représentation des connaissances en un format convenable pour une analyse est donc essentielle pour un bon travail d'ingénierie des connaissances. Il est parfois recommandé d'utiliser plus d'une représentation, ce qui suggère qu'aucune représentation en particulier soit entièrement adéquate pour représenter les connaissances initiales. Il semble plutôt qu'il y ait différents types de connaissances convenant à différentes représentations.

Pour ce, il est donc nécessaire de maîtriser le cycle de transformation suivant :

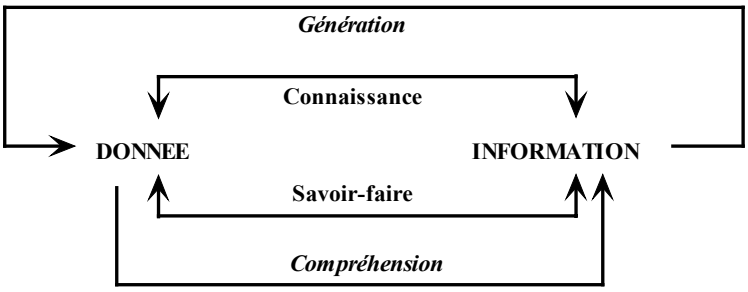

Figure 11 : Des données aux informations (d'après [Prince, 1996])

Les définitions suivantes sont alors données par [Prince, 1996] :

- Une *donnée* représente la matière à l'état brut.
- *L'information* est un signifié transporté par une donnée. C'est le résultat final de la transformation (fort complexe) de la donnée. L'information est là où commence le sens, et le sens ne peut s'opérer sans connaissances.
- La *connaissance* est le mode d'emploi permettant de transformer les données en informations. C'est donc ce qui est associé au passage entre le signifiant et le signifié. L'attribution de sens est le rôle fondamental de la connaissance.
- La *compréhension* est alors la transformation des données qui a pour résultat la création de l'information. La *génération* est l'opération symétrique.
- Le *savoir-faire* est un mode d'optimisation de l'attribution de sens à des données (en acquisition ou en génération), « court-circuitant » le passage par les connaissances nécessaires qui auraient abouti au même résultat. Contrairement aux connaissances, le savoir-faire n'obéit pas à la loi de compositionalité et n'est pas auto-discursif (savoir le décrire ne revient pas à savoir l'appliquer). Le savoir-faire provient d'informations devenues connaissances que l'agent (artificiel ou humain) va raccourcir, optimiser, agencer de manière à gagner du temps et de la mise en œuvre dans ses mécanismes cognitifs.

Cependant, comme le souligne [Prince, 1996], « *il ne suffit pas de demander à un expert de s'exprimer sous forme de règles de production, il faut encore être en mesure de rendre compte de la signification profonde des différentes propositions qui sont placées en prémisses et conclusions* ».

Ainsi, deux points de vue peuvent alors apparaître :

- les machines se doivent d'acquérir une sorte de « compréhension » de la langue, ou bien
- les experts sont capables de faire une traduction systématique des messages et des connaissances dans une représentation accessible par la machine.

Dans le premier cas, [Grice, 1975] montre qu'une théorie de l'expression langagière rationnelle possède suffisamment de points communs avec une expression logiquement structurée, ce qui implique, via des transformations contrôlées, une traduction pertinente utilisable par le système.
Toutefois, les aspects polymorphes de la langue imposent de sérieuses limitations tout comme les phénomènes de rhétorique tels que la métaphore ou bien de contraction et d'expansion (polysémie, paraphrase) qui ne sont pas rares même dans les textes techniques.
Dans le second cas, il est nécessaire que les experts fassent un effort afin d'exprimer leurs connaissances selon un modèle commun de spécification, un niveau de difficulté accessible et dans un temps convenable. En outre la subjectivité de la représentation peut aboutir à des omissions d'informations jugées initialement superflues mais qui peuvent ultérieurement se révéler pertinentes et qui sont alors irrémédiablement perdues. C'est toutefois vers ce second point de vue que nous avons choisi de situer nos travaux de recherche et pour lesquels nous allons présenter des principes et outils permettant de pallier le mieux possible aux inconvénients cités ci-dessus.

Ainsi, nous définissons une méthode permettant la spécification des connaissances pédagogiques au sein d'un atelier de génie éducatif.
[Rumbaugh *et al.*, 1994] donnent les définitions suivantes :

- Une *méthode* est un guide général pour aider des gens dans une tâche. Elle doit ainsi pouvoir s'appliquer à des situations différentes et être utilisée par des gens différents.
- Une *méthode informatique* doit fournir des guides pour aider à réaliser des programmes complexes.

Une méthode se décline alors en quatre composants :

1. un ensemble de *concepts fondamentaux de modélisation* permettant de faciliter l'expression des connaissances sur le problème et la solution. Ces concepts sont indépendants de leur visualisation. Ils sont aussi les entrées aux différents outils comme les outils de traçage ou les générateurs de code. L'ensemble de concepts de modélisation doit être relativement restreint afin de disposer de capacité d'expression suffisante mais pas trop restreint pour éviter que la modélisation de choses usuelles ne devienne trop complexe.

2. les *notations* et les *modèles de description,* généralement graphiques, qui permettent la compréhension et la modification de la méthode [18]. Un modèle est donc une représentation formelle du système selon un certain niveau d'abstraction.
3. un *processus de développement* itératif permettant la construction pas à pas des modèles selon différents niveaux de détails. Le processus doit décrire quels modèles construire mais aussi quand et comment les faire. Il fournit aussi des méthodes d'évaluation des modèles produits.
4. Un ensemble *d'indications*, de *trucs et astuces* pour le développement (comme par exemple le concept de « *pattern* »).

Tout au long de cette thèse, nous décrivons de façon précise ces quatre composants que nous appliquons à la spécification des connaissances pédagogiques au sein d'un atelier de génie éducatif.
La première partie de ce document nous a donc permis de fixer la terminologie (point 1), c'est-à-dire les concepts fondamentaux qui seront à la base de la modélisation des connaissances et notamment la notion de situation-problème.
Les notations et différents modèles de description (point 2) sont présentés dans la troisième partie de cette thèse (chapitre 8) tandis que le processus de développement préconisé (point 3) fait l'objet de la seconde partie (chapitre 6).
Il est toutefois important de noter que, comme le dit [Rumbaugh *et al.*, 1994], « *il ne faut pas attendre qu'une méthode dise tout sur tout* » car « *la créativité de l'utilisateur demeure* ». Ceci pose donc dès à présent des restrictions quant aux résultats des recherches que nous allons présenter même si, bien entendu, nous essayons de limiter le plus possible les limites de notre système. Ainsi, nous proposons aussi des *outils* dont le but est de faciliter la tâche du pédagogue selon trois principaux axes : réutilisation des connaissances, guidage et flexibilité dans le processus de développement et la notation. Le pédagogue joue alors à la fois le rôle d'expert du domaine et de concepteur de la spécification.

[18] Le plus souvent une méthode est assimilée à sa notation et c'est une erreur.

Deuxième Partie :
Pour un environnement de spécification d'enseignements

Résumé de la deuxième partie

Tout au long de la première partie de cette thèse, nous avons exposé les différents éléments clés sur lesquels reposent nos travaux de recherche.

C'est ainsi que le chapitre 2 fixe les bases des domaines que sont l'enseignement et l'informatique. Nous avons présenté les principes pédagogiques qui sont aujourd'hui au centre de l'enseignement-apprentissage dans le système éducatif français et qui participent à une nouvelle conception de la pratique pédagogique. Une véritable technologie éducative permet non seulement d'aborder une réelle transdisciplinarité dans l'enseignement, mais aussi de mettre en place des situations d'enseignement pertinentes pour un apprentissage réussi.

Nous avons ensuite présenté dans le chapitre 3 le rôle joué par l'informatique au sein de ce système. Le génie éducatif est alors un domaine au sein duquel les différentes approches relatives à la mise en place d'EIAO performants doivent situer l'utilisateur au centre même de leurs préoccupations premières. Quelle que soit la directivité du système d'apprentissage, nous avons abordé différentes techniques et méthodes répondant toutes à une problématique commune. En effet, l'essence de la solution se situe dans une « bonne » gestion et acquisition des connaissances.

Le chapitre 4 montre les raisons pour lesquelles l'étape fondamentale pour l'élicitation des connaissances se situe au départ du processus de construction. C'est dans cette optique que l'ingénierie des besoins est un domaine essentiel qui facilite l'adéquation entre les besoins initiaux exprimés par le commanditaire d'un produit et le cahier des charges informatique « correspondant ». Ce cahier des charges doit alors donner lieu à une spécification, au sens informatique du terme, qui doit être formelle pour de nombreuses raisons dans le paragraphe 2. Le génie logiciel offre ici un apport important grâce à l'utilisation d'outils de prototypage rapide permettant une évaluation précoce des besoins réellement exprimés.

C'est en s'appuyant sur cet ensemble de principes que nous avons jeté les bases, dans le chapitre 5, du cahier des charges de l'environnement informatique d'aide à la spécification des connaissances pédagogiques. Résolument orienté vers une manipulation par un enseignant non-expert en informatique, ce document a été mis en place en collaboration avec divers acteurs impliqués dans les EIAO et sert de point de départ à nos travaux de recherche.

La seconde partie de cette thèse est alors consacrée à la mise en évidence et à une meilleure définition de la méthodologie de spécification des connaissances pédagogiques que nous préconisons.

Nous pouvons alors recentrer nos recherches dans le chapitre 6 sur l'acquisition des connaissances dans un environnement éducatif. Nous étudions, dans un premier temps, comment ce processus est abordé par différents auteurs et notamment [Paquette *et al.*, 1996] avec le projet du « *Campus Virtuel* », avant de proposer notre propre processus de spécification des connaissances basé sur les fondements pédagogiques émergeant de la première partie de cette thèse.

Le chapitre 7 est ensuite consacré à la définition d'un processus de spécification des enseignements répondant à l'ensemble des contraintes exprimées dans les chapitres 5 et 6. En effet, à l'heure actuelle, il est préconisé que le développement des produits s'appuie sur un cycle de vie centré sur le prototypage. Nous en présentons les avantages ainsi que le point de vue de différents spécialistes du domaine. En outre, ce processus ne peut se concevoir sans le support d'outils adéquats. C'est dans ce sens que la technologie CASE et méta-CASE apporte un complément. Nous pouvons alors proposer, en fin de chapitre, un cycle de développement spécifique servant de support à la méthodologie que nous préconisons.

Plan de la deuxième partie

CHAPITRE 5 : CAHIER DES CHARGES DE OMAGE 71

CHAPITRE 6 : SPECIFICATION DES CONNAISSANCES DANS UN ENVIRONNEMENT EDUCATIF 77

Paragraphe 1 : Le modèle du « Campus Virtuel » 79

1. Présentation des éléments principaux 79
2. Spécification de l'éditeur de scénario d'apprentissage 80
3. Conclusions 81

Paragraphe 2 : Proposition d'un processus de spécification en plusieurs phases 83

1. Modélisation du contexte général 85
2. Etablissement de la matrice disciplinaire 85
3. Définition des objectifs d'apprentissage 86
4. Mise en place des stratégies globales d'enseignement 87
5. Caractérisation des situations-problèmes 87
6. Caractérisation des situations d'apprentissage « classiques » 89

CHAPITRE 7 : UN CYCLE DE DEVELOPPEMENT ADAPTE AUX CONTRAINTES PEDAGOGIQUES 91

Paragraphe 1 : Le prototypage et ses outils 91

1. Différents cycles de prototypage 92
2. Les outils CASE et méta-CASE 97
   - 2.1. Définitions 97
     - 2.1.1. Qu'est ce que CASE ? 98
     - 2.1.2. Qu'est ce que méta-CASE ? 98
   - 2.2. Pourquoi de tels outils ? 99
   - 2.3. Quelques exemples de produits méta-CASE 101
     - 2.3.1. Quelques produits méta-CASE non commerciaux 101
     - 2.3.2. Quelques autres produits méta-CASE commerciaux 102

Paragraphe 2 : Notre environnement de développement d'enseignements 102

1. Cycles de vie des logiciels éducatifs 103
2. Conclusion : principes fondamentaux de notre environnement 106

Chapitre 5 : Cahier des charges de OMAGE

Dans la première partie de cette thèse, nous avons tout d'abord considéré dans quelle mesure l'informatique intervient à l'heure actuelle dans le domaine de l'enseignement. Nous avons ensuite mis en évidence la nécessité d'une approche rigoureuse et cohérente permettant d'aboutir à un cahier des charges présentant de façon claire les besoins initiaux puis à une spécification formelle présentant parfaitement les connaissances impliquées.

Ceci est d'autant plus vrai dans le domaine de l'enseignement où, bien trop souvent, les logiciels éducatifs, malgré leurs fonctionnalités de plus en plus ouvertes, notamment par l'apport du multimédia, ne répondent pas aux besoins des pédagogues et demeurent par conséquent inutilisés puis inutiles.

Nous allons donc définir de façon précise quels doivent être les outils et fonctions d'un environnement informatique de spécification des connaissances relatives à l'enseignement. Le chapitre ci-dessous se base sur une interview structurée qui a été instaurée entre un enseignant d'Anglais de spécialité à l'Institut Universitaire de Technologie Informatique de Bayonne : Jacques Meyranx (JM), et un informaticien chercheur en EIAO : moi-même (CM).

Avant de poursuivre plus loin dans nos investigations, il est important de préciser ici que l'ensemble des points relatifs aux approches de l'enseignement et de l'informatique dans l'enseignement qui ont été présentés dans la première partie ce cette thèse ont été mutuellement validés par les deux protagonistes du dialogue.

Un environnement d'assistance à la spécification

CM : Qu'attendez-vous d'un environnement informatique de spécification d'enseignements ?

JM : Je souhaite plusieurs choses. Tout d'abord, qu'il *m'assiste dans ma démarche*. Mais attention, je ne veux surtout *pas être contraint ni restreint* pour tout ce qui est relatif à l'aspect *créatif* de l'enseignement. Même si ma façon de construire mes enseignements se base sur des principes connus et reconnus en Sciences de l'Education, je souhaite disposer d'un *ensemble d'outils* suffisamment *souples* pour me permettre de modéliser *ma* façon de concevoir.

CM : Comment concevez-vous cette *flexibilité* ?

JM : A plusieurs niveaux car chaque personne a *sa propre conception d'un même problème*, son propre *point de vue*.

Par conséquent, nous allons mettre en place un environnement informatique qui sera constitué par ensemble d'outils qui devront non seulement assister le pédagogue dans son travail de spécification des connaissances mais aussi lui permettre différentes façons de spécifier. Cette souplesse est une condition nécessaire afin de ne pas annihiler la créativité de l'enseignant.
Il convient cependant de proposer un fil directeur pour la spécification, une méthodologie.

Un environnement supportant une méthodologie en plusieurs étapes

CM Pourriez-vous détailler les principes méthodologiques que vous souhaiteriez suivre.

JM Elle est basée sur la notion de *situation-problème*. En effet, une situation-problème permet d'évaluer les *obstacles réels* rencontrés par un apprenant par rapport non seulement aux *objectifs* fixés *au départ* par le pédagogue mais aussi aux *obstacles* et *acquis présupposés*. Ceci permet ainsi de définir de façon plus adéquate les *situations d'enseignement-apprentissage plus traditionnelles* à mettre en place par exemple sous forme de cours, travaux dirigés et travaux pratiques.

CM Je pense qu'il convient maintenant de détailler, de façon la plus précise possible, les différents *éléments* de cette *méthodologie* tant du point de vue des *étapes* dans la construction des situations d'enseignement que de celui des *composants* de chaque étape.

JM Je vais te préciser ceci selon une méthodologie unilatérale de construction d'enseignements mais il est important de souligner que dans la pratique, *l'ordre des diverses étapes est aléatoire* et un environnement informatique, pour être réellement utilisable, devra permettre cette liberté…

CM Il serait aussi intéressant que vous précisiez pour chaque étape quelle est l'aide de façon pratique que vous vous attendez à trouver de la part de l'ordinateur.

Les étapes suivantes abordent le travail de spécification selon une démarche résolument descendante. Toutefois, dans le cas où un pédagogue n'adopte pas une telle démarche, il conviendra de lui fournir les moyens répondant à ses besoins.

Mise en place du contexte

JM Dans un premier temps, il convient de mettre en place un *modèle du domaine* qui est un découpage fonctionnel du domaine en sous-systèmes opérationnels.

CM Comment se conçoit-il ? Quelles en sont ces composantes ?

JM C'est aussi à ce niveau qu'il faut fixer les valeurs d'un certain nombre de paramètres comme le « *niveau* » de l'apprenant. En effet, pour chaque *notion* à prendre en compte dans l'enseignement-apprentissage, il convient de lui associer un *registre de conceptualisation* ou *de formulation*. Nous pouvons choisir par exemple de quantifier ceci en accord avec les

niveaux académiques usuels dans le système français (Cycle des apprentissage premiers, Classe de 3°, de Terminale S, IUT Informatique 1° année, Maîtrise de Physique, …). Aussi, le fait de fixer une valeur en début de méthodologie a pour effet de rester valable pour l'ensemble des étapes suivantes.

De même, il convient à ce niveau de donner les valeurs d'autres « *variables* » comme, par exemple, une partie de ses *connaissances acquises*.

CM Voilà donc la première étape de définition des éléments du domaine. Est-ce tout pour cette l'étape ?

JM Pas exactement, nous devons ensuite sélectionner, parmi ces sous-systèmes, ceux sur lesquels nous allons concentrer notre travail.

Définition de la Matrice Disciplinaire

JM L'étape suivante marque nettement une évolution. Après avoir dressé le contexte général de travail, nous modélisons à ce niveau le *contexte particulier* dans lequel le pédagogue situe son enseignement. Cependant, il n'y a pas vraiment de rapport entre l'étape précédente et celle-ci dans la mesure où chaque domaine ne se détaille pas en *matrice disciplinaire*. Nous conservons tout de même la première étape afin de garder trace des différentes façons d'aborder un même enseignement par différents pédagogues.

CM Comment concevez-vous cette matrice ?

JM Dans la matrice disciplinaire, le pédagogue fixe *l'univers du discours* en terme de *notions* auxquelles sont attachées des *compétences*. Ces dernières sont en fait les *principes unificateurs* relatifs à *l'objet* que l'on veut faire acquérir à l'apprenant et se rapportant soit au domaine ou étant transdisciplinaire. Ces objets sont reliés par trois types de *liens* : composition, référence et héritage qui représentent respectivement les liens d'agrégation, d'association et d'héritage au sens courant du génie logiciel. En fait, ces notions sont des *concepts intégrateurs* au sens de [Develay, 1993]. Il conviendra donc de les relier à des notions moins englobantes et aussi à des *faits*.

La matrice disciplinaire constitue donc le noyau dur de la discipline dans le sens où elle est la trame notionnelle regroupant les champs notionnels. Elle permet également d'aborder une réelle trandisciplinarité dans l'enseignement.

Une approche permettant la transdisciplinarité

CM Comment abordez-vous la trandisciplinarité ?

JM Nous pensons que, sur une même trame notionnelle, il est possible d'appliquer des connaissances procédurales inhérentes à l'enseignement de disciplines ou matières différentes.

CM Mais plus précisément...

JM Ainsi, par exemple, sur la trame notionnelle d'une certaine matière, GLOM par exemple, on peut appliquer les connaissances procédurales de GLOM mais on peut y appliquer tout autant les connaissances procédurales relatives à une autre matière, par exemple l'Anglais de spécialité (cas 1). De même ces mêmes connaissances procédurales relatives à l'Anglais de spécialité peuvent s'appliquer sur une trame notionnelle relative aux mathématiques ou à la gestion (cas 2). On peut alors parler de *trandisciplinarité instrumentale* (cas 1) et *procédurale* (cas 2) au sens de [Develay, 1993].

Mise en évidence des notions importantes dans la Matrice Disciplinaire

CM Voilà donc une bonne synthèse de l'étape d'établissement de la matrice disciplinaire. Quelle est l'étape qui suit ?

JM Il conviendrait de *focaliser* l'enseignement à produire sur tout ou partie de ce contexte représenté sous forme de matrice disciplinaire. Ensuite, par rapport à ce *focus*, il est nécessaire de fixer les objectifs d'apprentissage que le pédagogue veut mettre en place.

Mise en place des objectifs d'apprentissage et des stratégies d'enseignement

CM Considérons cette étape.

JM Disons que je voudrais pouvoir exprimer le *point de départ et d'arrivée* de ma situation d'apprentissage ainsi que les différents *objectifs* que je compte fixer pour l'apprenant ; ces objectifs ayant un *nom* bien entendu ainsi qu'un ensemble *d'actions à effectuer,* non seulement lorsque cet objectif est mis en place (en tant que *préliminaire* et sous certaines *conditions*), mais aussi lorsqu'il est atteint.

CM Et comment relier ces différents objectifs ?

JM En effet, à ces objectifs, il convient d'appliquer ce que j'appelle une (ou plusieurs) *stratégie(s) d'enseignement*. Nous définissons ici aussi deux *niveaux* :

- *global* : entre les objectifs d'apprentissage,
- *local* : entre les sous-objectifs d'apprentissage.

CM Pour cette étape, tout comme pour l'étape précédente, il convient donc de soigner particulièrement ces deux niveaux car ils permettront de prendre en compte puis de gérer les *changements de stratégies*. Mais précisons d'abord quels sont les *liens* entre objectifs.

JM Il doit être possible de préciser différents types de liens comme la *précédence*, la *concurrence* ou la *conséquence* qui ont une réelle signification d'un point de vue pédagogique.

CM Et je suppose que cette affectation de type doit pouvoir se faire à tout moment, c'est-à-dire lors de la création du lien mais aussi ultérieurement en donnant par exemple la possibilité de créer un lien *indéfini* entre deux objectifs puis en précisant son type dans un second temps.

JM Voilà !

CM De même, je suppose que deux stratégies d'enseignement peuvent être relatives aux mêmes objectifs d'apprentissage et ne différer que par le type des liens reliant ces objectifs.

JM Exactement et on retrouve ici les besoins de souplesse qui permettent une grande part de créativité dans l'enseignement.

Mise en place de situations-problèmes

CM Et quelle est la suite de la modélisation des connaissances ?

JM Nous allons considérer la mise en place de *situations-problèmes.*

CM Comment la concevez-vous ?

JM Il convient de mettre en correspondance chaque stratégie avec un obstacle anticipé bloquant l'apprentissage. A cet obstacle correspond alors un certain nombre de paramètres comme des critères de franchissement ou d'échec ainsi que des ressources disponibles, non seulement d'un point de vue matériel pédagogique mais aussi par rapport aux techniques de raisonnement mises en jeu. De même il faut fixer le temps imparti pour la résolution du problème et les contraintes d'organisation pour l'apprenant.

Nous avons donc donné un aperçu des différentes étapes de la méthodologie de spécification des enseignements.

Nous allons maintenant revenir sur le rôle que peut y jouer un environnement informatique. L'informaticien occupe ici un rôle important d'information du pédagogue au sujet des possibilités offertes par l'informatique et qui demeurent parfois, et malheureusement trop souvent, inconnues ou insoupçonnées par les non-initiés.

Apports de l'informatique

CM D'un point de vue général, quelle est l'aide ou quelles sont les fonctionnalités que vous vous attendez à trouver dans un environnement informatique.

JM Tout d'abord, le premier apport que j'attends d'un ordinateur est la possibilité de *mémoriser son travail* pour pouvoir le *réutiliser* ultérieurement.

Mémoriser et récupérer son travail

CM C'est la moindre des choses…

JM Oui mais il serait en outre intéressant de *pouvoir consulter ou récupérer le travail de ses collègues* tant pour des *raisons pédagogiques et didactiques transdisciplinaires* en ce qui me concerne comme enseignant de l'Anglais de spécialité que pour prendre en considération les différentes façons de travailler de chacun et combler par exemple des lacunes personnelles pour proposer de *nouveaux schémas mentaux de raisonnement*. En effet, je n'ai pas la

prétention de tout connaître dans mon domaine d'enseignement, ni du point de vue du *savoir*, que du *savoir faire* et du *savoir faire faire*.

Assistance à la spécification

CM Il convient donc de disposer d'un certain nombre d'outils de sauvegarde mais aussi de suivi de la spécification afin de pouvoir assister l'utilisateur au moment opportun. Par exemple un pédagogue utilisateur inexpérimenté dans la manipulation des différents outils de l'environnement pourra se voir proposer une *aide* de façon *automatique* et *systématique* pour consulter des éléments en mémoire.

JM C'est intéressant, mais encore…

CM *Certains outils ou fonctionnalités peuvent aussi être temporairement masqués* pour éviter des erreurs ou malfaçons et assurer l'utilisateur d'obtenir un *résultat cohérent*.

JM Par exemple ?

CM Et bien par exemple, un utilisateur *débutant* pourrait se voir proposer un *processus très directif* de spécification d'enseignements : l'ordre des étapes étant contraint (en commençant par la construction de la matrice disciplinaire pour finir par celle des situations plus traditionnelles) et en obligeant par exemple la définition complète de chaque élément de chaque étape avant de créer un autre élément.

JM Et du point de vue de *l'interface-utilisateur* ?

CM Chacun doit pouvoir disposer de son propre formalisme de représentation des connaissances (une notation graphique par exemple). Cependant, avec la diversité des possibilités, nous pensons que chaque modélisation proposée doit être traduite en un *formalisme fédérateur unique*. Et c'est ceci qui doit être conservé en mémoire au sein de l'ordinateur qui doit se charger ensuite de le *traduire* selon les desiderata, selon le profil de chacun. Ceci étant d'autant plus justifié si on désire réutiliser le travail produit par d'autres personnes.

JM Tout ceci me semble convenable, reste maintenant à développer un *prototype* démontrant la validité de ces souhaits…

Ce chapitre constitue donc le cahier des charges de l'environnement d'aide à la spécification des connaissances que nous souhaitons développer.

Nous allons maintenant considérer quels sont les moyens à mettre en œuvre pour disposer d'un tel environnement de spécification d'enseignements.

Chapitre 6 : Spécification des connaissances dans un environnement éducatif

Il convient maintenant de préciser dans quel cadre doit se faire la spécification des connaissances par des pédagogues.

[Bruillard et Vivet, 1994] préconisent une approche qui s'articule autour de quatre propositions :

① Bâtir les projets en partant de l'élève en situation d'apprentissage et mettre ce concept de situation d'apprentissage au cœur de la démarche de conception.

② Définir un système de spécification des situations permettant de préciser et de contraindre le rôle des acteurs et des outils.

③ Baser l'évaluation sur ces spécifications, donnant des implications a priori sur les outils et centrer ainsi cette évaluation sur l'analyse globale des situations plutôt que sur les performances techniques des outils (séparer évaluation technique, ergonomique et éducative).

④ Structurer une démarche méthodologique intégrant l'ensemble des acteurs dans le processus de conception des EIAO.

Ceci est également repris par [Schank et Kass, 1996] qui précisent qu'un environnement d'apprentissage doit non seulement fournir un accès vers des informations pertinentes, mais il doit aussi effectuer les trois points suivants :

- générer des *buts* qui vont motiver les apprenants pour accéder aux informations;
- fournir un *contexte réel* dans lequel situer l'accès aux connaissances des apprenants;
- confronter les apprenants avec des défis spécifiques qui nécessitent de leur part *d'analyser* les informations auxquelles ils accèdent, puis de les *utiliser*.

[Schank et Kass, 1996] ont ainsi conçu une architecture d'environnement d'apprentissage répondant à ces pré-requis et baptisée GBS (acronyme de « *Goal-Based Scenario* »). Les produits qui en résultent sont des cours structurés comme une collection de *scénarios*, chacun impliquant la poursuite d'un *objectif* précis au travers d'un ensemble *d'activités* conçues pour atteindre un ensemble de *compétences* bien ciblées. Dans GBS [19], l'apprentissage, la réalisation et la vérification d'un objectif ne sont pas séparés en trois phases distinctes comme dans la plupart des cours traditionnels. L'apprentissage et la vérification sont plutôt vus comme des produits de l'étape de réalisation qui représente alors la totalité du cours. Les apprenants ont un objectif motivant à atteindre et leur capacité

[19] [Schank et Kass, 1996] illustrent cette approche avec la description d'un exemple (« *Broadcast News* ») qui enseigne les études sociales aux universitaires.

à réaliser cet objectif est la seule vérification. La seule motivation nécessaire pour susciter la soif de savoir vient de l'incapacité initiale de l'apprenant à atteindre ce but.

[Eden *et al.*, 1996] pensent que nous devons reconceptualiser notre système éducatif. Son approche pour concevoir des outils éducatifs consiste à remettre en question les approches constructivistes et instructivistes. D'un côté, l'approche constructiviste permet aux apprenants de se lancer dans un auto-apprentissage mais, sans guidage, ils peuvent crouler sous le poids des options innombrables et l'opacité des objectifs à atteindre. Les systèmes instructivistes, au contraire, sont conçus dans des objectifs spécifiques mais offrent peu de place à la création.
Le système proposé par [Eden *et al.*, 1996], plutôt que d'être totalement ouvert, permet de concevoir des environnements *orientés-domaine*. Par contraste avec les environnements de programmation généraux, ce système permet de personnaliser les applications produites en fonction du domaine. Cette spécificité donne un plus haut degré de *contextualisation* aux supports éducatifs créés qui sont mis à la disposition des apprenants.
Plutôt que de définir aussi rapidement que possible des *capacités* à atteindre, il est nécessaire de se concentrer sur l'expression des *situations d'apprentissage* qui doivent non seulement gérer les capacités de façon très rapide mais aussi, et surtout, devraient permettre aux apprenants de s'investir totalement dans la recherche de solutions (grande motivation).
L'implication dans des problèmes réels implique que le choix des tâches et des objectifs soit sous le contrôle de l'apprenant de par le fait que le système soit non seulement dirigé par l'utilisateur (« *learner-controlled* ») mais aussi d'un grand soutien pour celui-ci. Cette exigence illustre bien selon [Eden *et al.*, 1996] la limitation des environnements de programmation généraux (comme Logo, Scheme ou Smalltalk), qui pêchent dans l'aide apportée, et la limitation des STI qui dirigent peu ou mal l'utilisateur.
Ainsi, [Eden *et al.*, 1996] ont développé des outils de conception d'environnements orientés-domaine, comme « *Agentsheets* » [20], ou des environnements particuliers comme « *HyperGami* » [21], qui ont tous deux été utilisés de l'école élémentaire jusqu'au milieu industriel.

Nous avons vu trois approches relatives à la spécification des connaissances éducatives. Nous allons maintenant présenter de façon plus détaillée dans le paragraphe 1 un autre environnement plus particulièrement orienté vers l'enseignement et l'apprentissage à distance : le « *Campus Virtuel* » [Paquette *et al.*, 1996]. Nous définissons ensuite dans le paragraphe 2 le processus de spécification des

[20] Pour plus d'informations, se reporter à l'adresse suivante :
http://www.cs.edu.colorado.edu/~l3d/systems/agentsheets/
[21] Pour plus d'informations, se reporter à l'adresse suivante :
http://www.cs.edu.colorado.edu/~l3d/systems/hypergami/

connaissances basé sur la notion de situation-problème telle que nous l'avons préconisée dans la première partie.

Paragraphe 1 : Le modèle du « Campus Virtuel »

Le « *Campus Virtuel* » [Paquette *et al.*, 1996] est un système informatique pour la formation à distance destiné à soutenir le processus autonome et individuel de construction des connaissances par l'apprenant, tout en intégrant l'importante dimension collaborative de l'apprentissage. Ainsi, les acteurs se regroupent à l'occasion d'événements d'apprentissage : programmes cours, modules ou activités d'apprentissage. Du point de vue informatique, le « *Campus Virtuel* » est un système regroupant plusieurs logiciels qui permettent aux différents acteurs de jouer leur rôle dans le système d'apprentissage.

1. Présentation des éléments principaux

Reprenons tout d'abord des définitions. Les *acteurs* représentent tout ce qui interagit et échange des informations dans le système : les personnes, les morceaux de logiciels ou les documents. Certains acteurs transmettent des informations tandis que d'autres en reçoivent. De même, des acteurs peuvent être tantôt récepteurs, tantôt émetteurs. Selon [Jacobson, 1993], les acteurs sont essentiellement définis par leurs *rôles*, c'est-à-dire par la façon dont ils interagissent avec les autres acteurs dans les traitements.
Dans le « *Campus Virtuel* », cinq catégories d'acteurs sont présentes : l'apprenant, l'instructeur, l'expert du domaine, le gestionnaire et le concepteur, chacun défini par un ensemble précis de rôles dans le « *Campus Virtuel* ».

Chaque *rôle* d'un acteur peut être vu comme un ensemble de traitements ou de cas d'utilisations que cet acteur peut exécuter. Chaque *traitement* est un ensemble d'actions ou d'opérations qu'un utilisateur peut accomplir en interagissant au sein du « *Campus Virtuel* ». Un traitement est une des façons d'utiliser le système. L'ensemble de tous les traitements définit ce pour quoi le système est utilisé, développé et quelles sont les différentes façons d'utiliser le système.
Par exemple, un des rôles de l'apprenant est de se conduire comme un navigateur (« *Navigator* »), c'est-à-dire de circuler à travers toutes les activités d'apprentissage, les médias et les outils de scénario pédagogique pour atteindre certains objectifs d'apprentissage. Ici, certaines des actions impliquées sont les suivantes :

- Le navigateur accède au scénario en affichant les activités dans une unité d'apprentissage (ou en affichant les unités d'apprentissage dans un cours, ou bien les cours dans un curriculum).

- Si le navigateur voit un scénario de cours, il peut sélectionner une des unités d'apprentissage et ouvrir son propre scénario où les activités sont affichées.
- Le navigateur sélectionne enfin une activité qu'il veut explorer ou atteindre. Lorsqu'il ouvre cette activité, le système vérifie si les contraintes du scénario, comme les activités prérequises, sont respectées.
- Si les contraintes ne sont pas violées, les entrées, par exemple les documents à consulter ou les outils à utiliser, les sorties, par exemple la production à réaliser, et l'objectif de l'activité sont présentées à l'apprenant.
- Lorsqu'il adopte le rôle de preneur de décision de l'acteur gestionnaire, l'apprenant (ou l'équipe) va définir son propre scénario personnalisé en adaptant les scénarios proposés par l'acteur concepteur.
- Le système gardera la trace de l'interaction de l'apprenant avec l'environnement pour le superviseur interactif et pour faciliter le diagnostic par l'acteur instructeur.

2. Spécification de l'éditeur de scénario d'apprentissage

Au centre des différents éléments composant le « *Campus Virtuel* », se trouve le *scénario d'apprentissage* qui fournit plusieurs plans de travail possibles pour l'utilisateur. Le scénario d'apprentissage est le point de référence commun et le sujet de collaboration entre les différents acteurs. C'est essentiellement un réseau d'événements d'apprentissage (curriculum, cours, modules, activités), de médias didactiques ou d'outils de production utilisés ou produits dans ces activités. Chaque utilisateur suit (ou supporte, supervise, gère, conçoit) un certain scénario d'apprentissage. Ce scénario contient différentes références aux médias (documents), pour travailler sur des plans et des modèles de connaissance. Chaque utilisateur possède sa propre vue du scénario d'apprentissage, des médias et de l'ensemble des ressources utilisées. Ainsi, chaque document ou outil peut être paramétré différemment selon chaque utilisateur. Inversement, chaque outil et document peut accumuler des données sur ses utilisateurs.

Le scénario d'apprentissage d'un événement d'apprentissage est un graphe composé de tous les chemins d'apprentissage possibles entre les événements de sous-apprentissage qui le constituent. En outre des chemins, le média didactique et les outils de production sont nécessaires pour exécuter un événement d'apprentissage.
Les événements sont structurés de façon hiérarchique. Au plus haut niveau, nous trouvons le curriculum et au niveau le plus bas, se trouvent les activités d'apprentissage qui ne peuvent être décomposées. Les noeuds du scénario d'apprentissage d'un curriculum sont les cours. Les noeuds du scénario d'apprentissage d'un cours sont les modules et ainsi de suite.

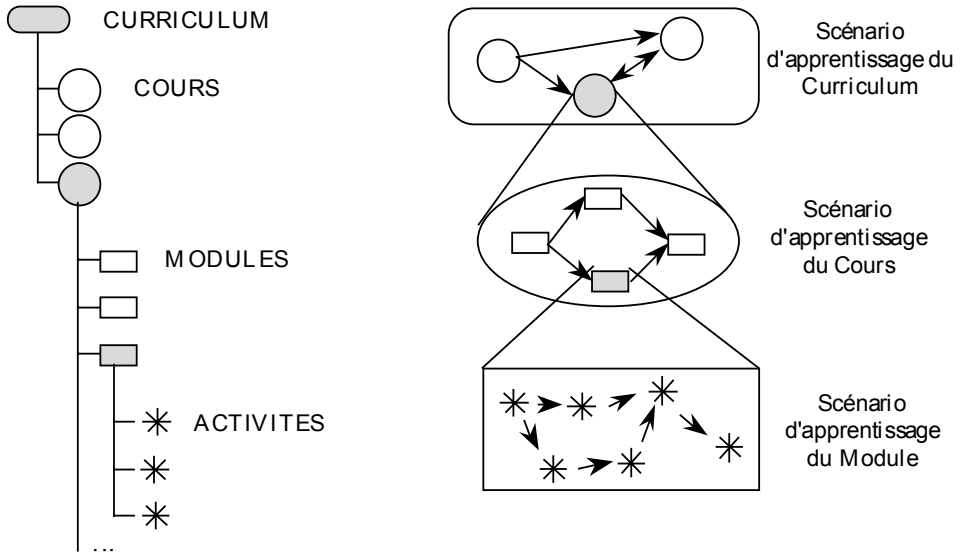

Figure 12 : Architecture des événements d'apprentissage au sein du « Campus Virtuel »

Le scénario d'apprentissage définit donc le chemin d'apprentissage par défaut qui sera proposé à l'apprenant pour un événement d'apprentissage. C'est le scénario d'apprentissage établi par le concepteur. Un scénario d'apprentissage peut être adapté par l'apprenant (ou une équipe) pour devenir un scénario d'apprentissage personnalisé. Une adaptation peut consister par exemple à l'ajout d'un livre à lire sur une connaissance manquante ou le changement de la description d'un papier à produire, etc…

Les objectifs de l'éditeur sont de permettre au concepteur de définir ses scénarios d'apprentissage avec une interface utilisateur graphique et de stimuler la réutilisation d'événements d'apprentissage prédéfinis en facilitant leur assemblage. Une version de visualisation est nécessaire pour l'apprenant. La principale différence avec la version concepteur est basée sur la limitation des fonctionnalités d'édition. Cependant, un apprenant peut adapter le scénario en ajoutant et déplaçant des objets. Une autre particularité de la version de visualisation est la possibilité de contrôle et de navigation. Dans la version de visualisation apprenant, la navigation à travers certaines parties du scénario est contrôlée en fonction de la progression de l'apprenant.

3. Conclusions

Les objectifs du « *Campus Virtuel* » sont donc la réalisation de l'aspect navigationnel requis pour les différents utilisateurs. Ceci consiste en particulier à donner aux concepteurs des outils permettant de créer des chemins d'apprentissage navigationnels que devront suivre les apprenants. L'éditeur de scénario d'apprentissage en est une partie.

Nous pouvons toutefois émettre quelques remarques à propos de l'environnement développé par [Paquette *et al.*, 1996]. En effet, même si l'utilisateur reste la préoccupation majeure du « *Campus Virtuel* », particularité essentielle comme nous l'avons souligné dans la première partie de la thèse, peu de précisions concernant la prise en mains de l'éditeur de scénario sont fournies. De même, les éléments qui sont à l'origine de la structure de cet éditeur demeurent flous. Or, nous devons toujours garder en mémoire qu'un produit n'est utilisable et par conséquent utilisé que s'il répond « parfaitement » aux besoins qui sont à l'origine de sa création.

En outre, la place occupée par les différents composants du processus d'apprentissage (cours, modules, activités), impose l'existence d'une méthodologie plus en amont et dont le but est de faire émerger les différents besoins en terme de composants. C'est dans cette optique que le processus d'enseignement-apprentissage basé sur la notion de situation-problème place l'apprenant au cœur même du processus d'apprentissage.

Cette remarque est également pertinente en ce qui concerne le projet européen KAMP évoqué dans la première partie (chapitre 3, paragraphe 1). Son modèle de classe virtuelle est découpé selon quatre niveaux hiérarchiques. Un cours est composé par un agencement de différents modules auxquels correspond un ensemble non ordonné d'activités pour lesquelles des exercices de vérification (« *assets* ») doivent être mis en place. Cependant même si ce découpage permet de simplifier la gestion des ressources pédagogiques, il n'est acceptable que pour des apprenants dont les besoins sont relativement stables [Gouardères *et al.*, 1997]. Il serait par conséquent intéressant de disposer d'un modèle prenant davantage en compte les besoins individuels qui évoluent en permanence. La mise en place de liens pédagogiques, non seulement au sein même des différents niveaux hiérarchiques, mais aussi entre ces niveaux est un élément de réponse. Par conséquent, un système d'aide à la définition de ces liens s'avérerait ici tout à fait pertinent.

De même, selon [NKambou, 1996], l'absence d'une organisation claire de la matière à enseigner est l'une des faiblesses majeures des EIAO qui peut être résolue par la notion de *curriculum* qui est une organisation explicite de la matière permettant la planification de l'enseignement, le contrôle et l'évaluation des apprentissages réalisés.

Au sein des systèmes tuteurs intelligents, [NKambou, 1996] préconise donc que le curriculum contienne la structure de la matière indépendamment de son enseignement afin de clarifier la frontière entre les composants et la planification dans l'architecture des systèmes. Dans ce sens, le curriculum est défini comme une représentation structurée de la matière à enseigner en terme de *capacités*, *d'objectifs* (dont l'atteinte contribue à l'acquisition des capacités) et de *ressources pédagogiques*. [Nkambou, 1996] définit l'ensemble de ces connaissances en se fondant non seulement sur une expertise du domaine issue des travaux de [Anderson, 1988], mais aussi sur une analyse cognitive des tâches du domaine [Polson, 1993] et des théories pyschopédagogiques [Gagné, 1985].

D'où les définitions suivantes :

- Un objectif est la description d'un ensemble de comportements ou performances dont l'apprenant doit se montrer capable pour être reconnu compétent.
- Une ressource pédagogique est un objet (exercice, problème, test, simulation, etc…) utilisé par le système d'enseignement pour supporter l'apprentissage par l'apprenant de capacités le rendant capable des performances indiquées dans les objectifs.

Cette structure du curriculum permet donc de déterminer dynamiquement la situation d'enseignement à proposer aux apprenants. « CREAM-Tools » est un atelier de génie didacticiel pour l'acquisition des curriculums suivant l'approche CREAM [22] qui a été implémentée avec le langage Smalltalk-80 dans le cadre du projet SAFARI [Frasson et Gauthier, 1994].
Cependant, dans cette approche, un objectif décrit uniquement le résultat à atteindre dans un cours plutôt que le processus d'enseignement ou les moyens mis en œuvre.

Ceci nous amène par conséquent à proposer un cycle de spécification des connaissances basé sur la notion de situation-problème que nous allons détailler dans le paragraphe suivant, et qui s'appuie sur les principes évoqués dans la première partie de cette thèse (chapitre 2 et IV).

Paragraphe 2 : Proposition d'un processus de spécification en plusieurs phases

Dans ce paragraphe, nous allons aborder la spécification des connaissances par l'enseignant en nous reportant aux principes évoqués dans le chapitre 2 (paragraphe 1, § 2 et 3). Nous pouvons alors découper ce travail en trois étapes principales auxquelles il convient d'appliquer les éléments pédagogiques facilitant une approche transdisciplinaire de l'enseignement.
Cela nous amène alors à définir un processus spécifique de spécification.

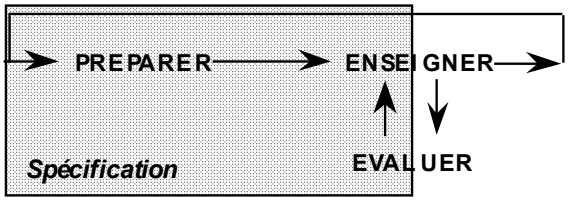

Figure 13 : Les trois étapes du travail du pédagogue

La phase de préparation ou d'anticipation est le point de départ.
La *matrice disciplinaire* constitue un outil privilégié pour la phase de préparation en ce sens qu'elle permet de modéliser le *contenu* à enseigner en terme de *savoir déclaratifs* et *procéduraux*.

[22] Acronyme de « *Curriculum REpresentation and Acquisition Model* » [NKambou, 1996].

On peut transmettre un savoir et-ou un savoir-faire en proposant des *objectifs d'apprentissage* puis des *stratégies d'enseignement* puis une *situation-problème.* Cette dernière est définie comme une situation didactique dans laquelle il est proposé au sujet une *tâche* qu'il ne peut mener à bien sans effectuer un apprentissage précis.

Il convient alors de considérer deux niveaux de préparation :

- au niveau micro : on s'intéresse à une situation-problème particulière (obstacle-objectif, matériel, contraintes, etc…)
- au niveau macro : on s'intéresse à l'organisation générale de la transmission de la matrice disciplinaire et donc à la définition de l'enchaînement des situations-problèmes perçues ici comme des « boîtes noires ».

La phase d'enseignement est l'étape suivante durant laquelle le pédagogue doit effectuer une évaluation dynamique des connaissances de l'apprenant et, en fonction des résultats de celle-ci, veiller à proposer une évolution permanente.

C'est exploiter le *canevas* issu de la préparation et prendre les décisions adéquates par rapport au comportement de l'apprenant.

L'enseignant effectue des choix en permanence et en temps réel ; ceci nous amène à introduire dans l'environnement des connaissances susceptibles de gérer totalement ou en partie la *prise de décision*.

On doit prendre en compte ici deux niveaux de guidage :

- au niveau micro : les tableaux de remédiation correspondant à une situation-problème;
- au niveau macro : les modifications potentielles (ou l'éventuelle reconstruction) d'un enchaînement préétabli de situations-problèmes.

L'évaluation, et la réévaluation, se conçoivent classiquement comme :

① *diagnostique* : en début de réalisation, elle permet de mettre en place l'obstacle adéquat avec les bonnes consignes et ressources.

② *formative* : en cours de réalisation pour apprécier les processus utilisés par les apprenants.

③ *sommative* : en fin de réalisation, elle permet de mesurer les écarts entre les objectifs initiaux et l'acquisition elle-même qui peut être validée dans le cadre de situations d'application et de décontextualisation ou de transfert des connaissances. Cette phase peut, si l'acquisition n'est pas validée, renvoyer à des activités de remédiation.

Des retours-arrières sont alors envisageables et le cycle d'enseignement se poursuit.

Par conséquent, pour spécifier leurs idées, les pédagogues se doivent de respecter un processus de formalisation construit en six étapes principales :

- Définition du contexte général,
- Etablissement de la matrice disciplinaire,
- Définition des objectifs d'apprentissage,
- Mise en place des stratégies globales d'enseignement,
- Caractérisation des situations-problèmes,
- Caractérisation des situations d'apprentissage « classiques ».

Il est important de préciser avant tout que la démarche que nous préconisons ne se veut surtout pas universelle d'un point de vue pratique. En effet, l'approche explicitée ci-dessous se veut résolument descendante dans la modélisation mais nous reviendrons ultérieurement sur cet aspect car d'un point de vue pédagogique, un pédagogue pourra par exemple dériver une matrice non pas à partir des contenus-cibles mais des scénarios qu'il utilise (démarche ascendante), ou adopter une démarche hybride qui lui est propre.

1. Modélisation du contexte général

Dans un premier temps, il convient de mettre en place un *modèle du domaine* qui est un découpage fonctionnel du domaine en sous-systèmes liés entre eux par des *liens associatifs* simples. Ceci nous permet de poser les bases les plus générales sur le contexte dans lequel l'enseignement va se situer et que nous allons détailler dans les phases ultérieures du processus de développement.

2. Etablissement de la matrice disciplinaire

Nous pensons pouvoir dégager les *connaissances déclaratives* (notions) et *procédurales* (celles qui, mises en oeuvre, permettent d'accéder aux notions) que privilégie chaque enseignant dans son domaine.

A l'issue de cette étape, nous pouvons dégager des *concepts intégrateurs* à partir desquels ces notions s'organisent en *champs notionnels*. La fusion de ces champs et l'intégration des connaissances procédurales associées aux notions élémentaires, permettront de représenter la *matrice disciplinaire* au niveau retenu [23].

[23] Notons toutefois que la matrice disciplinaire ne constitue que le point de départ de notre environnement sur lequel vont venir se greffer tout un ensemble d'opérations destinées à mener à bien la spécification d'un enseignement-apprentissage.

A la faveur de ces opérations, nous pouvons identifier des champs disciplinaires sécants propices à une réelle transdisciplinarité.
Il s'agit donc dans un premier temps de constituer une matrice des notions par sous-domaine.
Dans le système des connaissances déclaratives à enseigner-apprendre, il s'agit toutefois de délimiter les frontières. Nous repérons ainsi des notions qui sont à l'extérieur du système ; elles sont de deux types :

- « *contextuel* », c'est-à-dire des notions de référence explicite ou implicite ;
- « *stratégique* » dans la mesure où il s'agit de notions essentielles à l'appropriation des notions de l'intérieur ; pratiquement, elles constitueront des pré-requis ou plutôt des pré-acquis.

Trois sortes de liens permettent de mettre en rapport les entités constituant le modèle de base : des *liens de référence* bidirectionnels qui relient deux entités dont la sémantique de l'une fait référence à celle de l'autre, des *liens de composition* uni-directionnels (relation de tout à partie) et des liens *d'héritage* au sens orienté-objet courant et qui sont plus particulièrement destinés à des pédagogues informaticiens ou déjà experts avec le système.
A ce niveau, se pose un problème : comment rattacher un ensemble de savoir-faire à un ensemble de notions ? Nous pouvons attacher une (ou plusieurs) compétence(s) à une notion en tant qu'attribut(s) de cette notion dans la mesure où la compétence ne se rapporte qu'à cette notion, c'est-à-dire s'il n'existe qu'un *point de vue* unique sur cette compétence.
Il est également important de souligner à ce niveau que la matrice disciplinaire ne se construit que pour un *registre de formulation*, un *registre de conceptualisation* et un *niveau d'exigence* donnés.
Le concept de « Notion Hors Contexte » apparaît également à ce niveau. C'est un concept permettant de caractériser une notion au même sens que précédemment mais qui restreint le champ d'application de la matrice disciplinaire.

3. Définition des objectifs d'apprentissage

Le pédagogue dresse à ce niveau les *compétences générales* que l'on veut faire acquérir par l'apprenant. Ce sont des principes unificateurs qui sont de trois ordres : *disciplinaires*, *transversaux* et *langagiers* [24].
La définition de ces objectifs d'apprentissage doit respecter deux points de vue complémentaires.
A chaque compétence générale est associé un ensemble de *concepts particuliers* (propres au domaine). Ces derniers permettront l'acquisition des compétences générales qui sont les objectifs d'apprentissage. Les concepts particuliers sont issus de la matrice disciplinaire. En effet, ils peuvent être soit des notions, soit des attributs de notions, soit des compétences liées aux notions.

De même, à ce niveau, le pédagogue doit également avoir la possibilité de raffiner chaque objectif d'apprentissage en un ensemble de compétences de plus bas niveau.

4. Mise en place des stratégies globales d'enseignement

Le pédagogue définit un *ordonnancement*, à travers différents *cheminements* possibles, des objets d'enseignement associés aux objectifs d'apprentissage. Ces cheminements représentent ainsi les différents *scénarios d'enseignement* planifiés par le pédagogue, c'est-à-dire la *progression* prévue, les différentes étapes qui permettront l'acquisition de connaissances.
A ce niveau, le pédagogue peut prévoir le parcours de l'ensemble des concepts particuliers liés à un objectif d'apprentissage avant de passer à l'objectif suivant. Cependant, aucune restriction n'est imposée et il peut tout autant prévoir un scénario où n'interviennent que deux concepts particuliers (issus ou non d'un même objectif).
Notons que chaque cheminement possède une signification pédagogique et didactique subjective (propre au pédagogue qui le définit). Ainsi, la notion de « sens » est importante durant la définition (le pédagogue définit des chemins orientés et non des chaînes).

Le pédagogue donne ainsi *a priori* la *stratégie d'apprentissage prévue*, comme par exemple « apprendre en faisant », « apprendre en critiquant » ou « apprendre par analogie », associée à chaque scénario d'enseignement.

Les cheminements entre objets associés aux objectifs d'apprentissage permettent de définir alors deux sortes de situations d'enseignement :

- des *situations-problèmes* qui représentent des objectifs-obstacles,
- des *situations d'apprentissage* « classiques » comme par exemple pour le transfert, la remédiation, le renforcement ou la métacognition.

5. Caractérisation des situations-problèmes

D'une façon générale, la *résolution de problèmes,* qui amène l'acquisition de connaissances, c'est-à-dire l'apprentissage, respecte le plan suivant :

① La *tâche* d'apprentissage est référée à son contexte. Cette tâche déclenche une représentation des *buts à atteindre*, de *critères de réussite* en même temps qu'elle conduit à s'interroger sur la *planification* qu'elle impose.

[24] Dans notre problématique, nous nous restreindrons aux compétences disciplinaires.

② Simultanément sont activées des *connaissances déclaratives et procédurales* dont l'intrication entraîne une *rétroaction* au niveau des représentations de la planification.

Ainsi, pour chaque scénario prévu, le pédagogue va définir une *situation-problème* en termes de *consignes*, *matériel* et de *critères d'achèvement* qui représentent des *variables globales de définition et d'évaluation pédagogique*. Les valeurs données à chacune de ces variables pourront de plus être *contraintes* par la stratégie d'apprentissage prévue.
Par exemple « apprendre en critiquant » sur un ensemble de données peut se concevoir de façon globale, point par point, en critiquant deux à deux, etc… De même le « travail de groupe » peut se faire de diverses façons :

- « en compétition » : un même sujet et une même technique pour tous les groupes ;
- « en comparaison » : un même sujet et une technique différente par groupe ;
- « en complémentarité » : sur des connaissances complémentaires portant sur un même sujet et avec une technique identique, ou différente, pour tous les groupes.

Nous devons aussi proposer au pédagogue des moyens permettant de découper une situation-problème en un ensemble de *séances*, chacune ayant une *durée* et un *but* bien précis.
La définition des situations-problèmes par le pédagogue lui permet donc de vérifier si les *obstacles* des apprenants qu'il a supposés sont bien réels. Ainsi la mise en place des situations-problèmes aboutit soit à un succès soit à un échec de l'apprenant. En cas de réussite, c'est-à-dire franchissement de l'objectif-obstacle par l'apprenant, nous pouvons supposer que le pédagogue a mal « pensé » son enseignement (niveau de difficulté insuffisant par exemple), mais cela lui donne des renseignements sur le profil de l'apprenant qui seront utiles ultérieurement pour la mise en place d'autres situations. En cas de non-franchissement de l'objectif-obstacle (on ne parlera toutefois pas d'échec d'un point de vue pédagogique), il convient de prendre en compte la valeur associée à chaque variable d'évaluation pédagogique (ainsi qu'éventuellement l'ensemble des valeurs prises par chacune d'entre elle). Ceci permettra de donner des informations sur les raisons de ce non-franchissement et par suite, ceci permettra au pédagogue de mettre en place des situations d'apprentissage adaptées.

Le pédagogue doit aussi prévoir une *planification* de l'ensemble des situations-problèmes qu'il a définie avec des *points d'arrêt* nécessaires pour établir une évaluation ponctuelle de l'apprentissage et un éventuel *réordonnancement*.

6. Caractérisation des situations d'apprentissage « classiques »

Tout comme les situations-problèmes permettent de faire apparaître les limites de l'apprenant, les situations d'apprentissage « classiques » permettent le dépassement de ses limites. Par conséquent, la mise en place de ces différentes situations se fait de façon identique à celle des situations-problèmes.

Nous proposons ainsi un processus spécifique permettant la mise en place de la spécification des connaissances relatives à l'enseignement-apprentissage.
Toutefois ceci reste limité si on n'y adjoint pas des outils servant de support. Le génie logiciel, grâce notamment au prototypage et aux technologies CASE et méta-CASE, répond à ces attentes. C'est ce que nous démontrons dans le chapitre suivant.

Chapitre 7 : Un cycle de développement adapté aux contraintes pédagogiques

Dans le chapitre 4, nous avons mis en évidence les principaux défauts des cycles de vie séquentiels et nous avons alors défini les caractéristiques essentielles d'une activité de prototypage.
Il convient à présent, non seulement de considérer les différentes approches à mettre en œuvre pour intégrer pleinement cette activité, mais aussi d'étudier les outils informatiques qui facilitent ceci. Nous proposons alors notre propre processus de développement appliqué à la spécification des connaissances pour l'enseignement.

Paragraphe 1 : Le prototypage et ses outils

Le prototypage doit être intégré dans le cycle de vie où les phases de spécification et conception deviennent fondamentales. L'ensemble de l'activité de prototypage, pour être mené à bien, doit donc évoluer dans un environnement *spécialisé*, un *environnement de prototypage*. Cet environnement doit permettre :

- une mise en œuvre *rapide* du prototype, selon les principes du prototypage rapide ou RAD (acronyme de « *Rapid Application Development* ») c'est-à-dire plus immédiate que l'approche conventionnelle (sans prototype). Le concepteur doit pouvoir adapter rapidement les différentes spécifications, améliorer les prototypes et, par conséquent, présenter rapidement à l'expert différentes versions d'un logiciel prototype qui *tourne ;*
- une compréhension *aisée.* Le prototype s'adresse à l'expert, généralement non-informaticien, et présente une version *expérimentale* du logiciel. A ce titre, il doit offrir toutes caractéristiques fonctionnelles, conviviales et ergonomiques permettant la manipulation de l'ensemble des paramètres du prototype.

Le concepteur développe son prototype dans un formalisme, un langage de programmation (de prototype), qui se veut adapté ou adaptable aux problèmes traités. Autant le formalisme mis en œuvre par un *modèle* est souvent spécifique au problème traité ou à une famille restreinte de problèmes, autant le formalisme utilisé pour l'élaboration d'un *prototype* doit être suffisamment *générique* pour permettre la construction de différents prototypes.

Trois caractéristiques essentielles sont nécessaires à un formalisme facilitant l'activité de prototypage [Krief, 1992] :

- *la modularité :*

*L*e formalisme doit permettre au concepteur de partitionner ses prototypes en modules *indépendants.* Des études de [Letovsky et Soloway, 1986] ont montré que la plupart des problèmes qui surviennent lors de la modification d'un logiciel, prototype ou non, sont dus à une forte inter-dépendance existant entre les différents modules du logiciel. Une bonne modularité facilitera donc les ajustements successifs du prototype.

- *la réutilisabilité :*

*L*e formalisme doit permettre de récupérer aisément des modules existants et de les intégrer dans le prototype en cours. Ces modules doivent posséder une *interface* correctement définie (spécification et fonctions), afin de permettre au concepteur de les manipuler sans connaître le détail de leur implantation. L'ensemble de ces modules *réutilisables* [Jones, 1984] constitue un formalisme complémentaire *intégré,* puisqu'ils sont écrits dans le langage de prototypage.

- *l'abstraction de données :*

*L*e formalisme doit permettre de représenter, de manipuler et de faire cohabiter des connaissances hétérogènes : interface utilisateur (fenêtre, souris, menus), concepts issus de la programmation parallèle (multi-processeurs, multi-tâches, synchronisation de processus) ; cette diversité des concepts manipulés se retrouve dans les projets mettant en œuvre plusieurs domaines disjoints. De plus, l'abstraction des concepts améliore la *modularité* structurelle des prototypes et facilite la création et l'échange de composants fonctionnels et structurels *réutilisables.*

Les quatre points suivants sont quatre perceptions différentes mais non antagonistes de cycles de vie basés sur le prototypage.

1. Différents cycles de prototypage

[Zamperoni et Gerritsen, 1994] préconisent un cycle de vie « hybride » basé sur le développement en spirale et le prototypage incrémental au sens de [Boehm, 1988]. Les auteurs spécifient précisément les diverses tâches, activités et leur interconnections selon un schéma non ambigu de phases de projet :

- tout d'abord une phase de préparation et d'analyse débouchant sur un premier prototype qui sera amélioré en phase d'évolution, puis
- une phase de terminaison où sont effectués les tests et enfin,
- la maintenance.

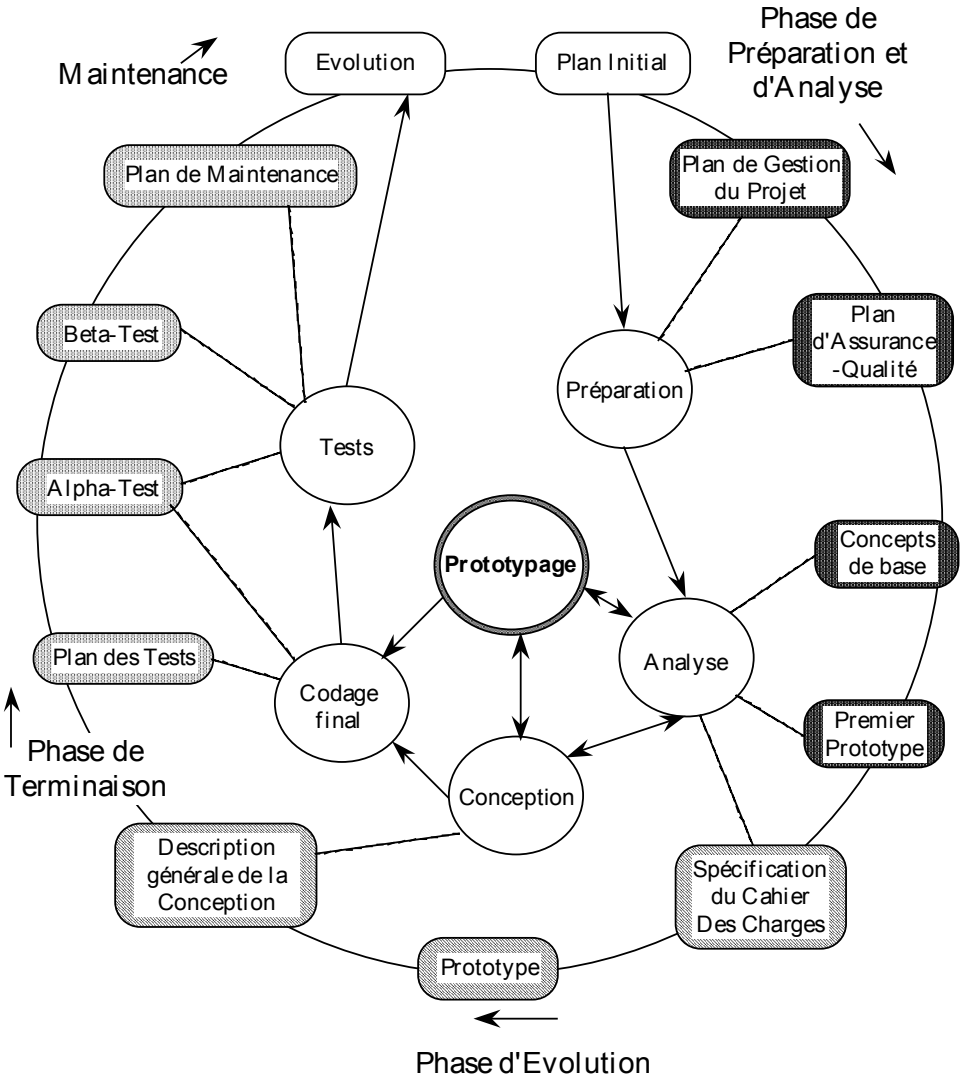

Figure 14 : Cycle de vie « évolutif » [Zamperoni et Gerritsen, 1994]

Pour démontrer ses propos, [Zamperoni et Gerritsen, 1994] ont appliqué ces différents préceptes en comparant deux projets développés au TNO dans le domaine de la production et l'exploration de gaz et pétrole selon les deux types de cycles de vie (en cascade et en spirale avec prototypage incrémental).

Selon [Luqi, 1989], le prototypage est un processus *interactif* et *itératif* de choix de décision entre l'expert du domaine et le concepteur. L'expert se distingue de l'utilisateur final dans la mesure où il participe à la conception et au développement du produit. Son attitude doit être critique par rapport au prototype : il fournit au concepteur, d'une part, l'ensemble des informations spécifiques au domaine traité et, d'autre part, un regard averti sur le logiciel en cours d'élaboration. Le prototype sert donc de support au dialogue entre le concepteur et l'expert.

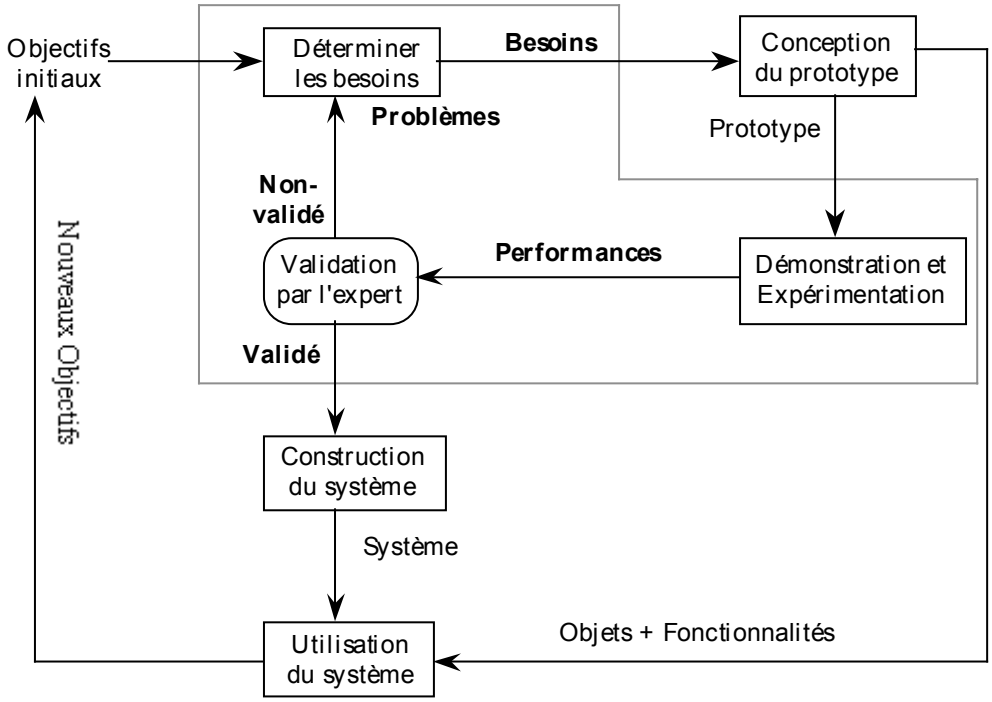

Figure 15 : Cycle de prototypage selon [Luqi, 1989]

Dans un premier temps, l'expert et le concepteur définissent ensemble les spécifications et les besoins des parties critiques du système élaboré. Dans un second temps, le concepteur développe un premier prototype (d'où un problème de *représentation des connaissances*) qui sera évalué par l'expert lors de démonstrations et expérimentations. S'il ne répond pas aux besoins spécifiés, c'est-à-dire au cahier des charges, ou s'il comporte des incohérences, l'expert identifie les différents problèmes et reprend la première phase de prototypage avec le concepteur. Ce processus se poursuit jusqu'à ce que l'expert valide le prototype, c'est-à-dire qu'il le considère à l'état de modèle.

[Bézivin, 1995] propose une première tentative de définition du *modèle des besoins* dans un cycle de vie à objets dans un cadre nommé schéma OSM (acronyme de « *Object Select & Merge* ») et dans la plate-forme correspondante d'atelier de génie logiciel générique OSMOSIS (acronyme de « *OSM, Organisation Scheme and Information System* »). Il s'agit d'une vision descendante (l'analyse de domaine) menée en parallèle avec une vision ascendante (la construction de la bibliothèque de base des composants ressource ou modèle technique des moyens). La conception doit alors réaliser la synthèse (ou la liaison) entre les *objets métiers* du modèle d'analyse de domaine et les *objets techniques* du modèle des moyens. Il y a d'abord sélection des composants amont et aval utiles à la réalisation des *scripts de besoins*. On procède ensuite à la réification des scripts de besoins c'est-à-dire à la construction des objets qui vont les représenter dans le modèle de conception. Pour ceci on peut créer des *composants de jonction* en utilisant différents mécanismes (héritage simple ou multiple, agrégation, schémas génériques, etc…). La phase de conception est suivie d'une phase

d'implémentation qui réalise l'adaptation finale du modèle de conception à la configuration cible (dimensionnement. optimisation. etc…). Cette façon de voir les choses se démarque très nettement de l'approche classique par raffinage de classes.

Le problème essentiel dans ce schéma OSM consiste à préciser, d'une part, comment se fait la sélection des objets métier et des objets techniques et, d'autre part, comment s'opère leur composition. La réponse à cette double question se trouve dans le modèle des besoins. On peut donc définir le modèle de conception comme une image du modèle des besoins complétée par des éléments du modèle d'analyse de domaine et du modèle des moyens techniques.

Le développement du logiciel sépare généralement la spécification du problème de l'élaboration de la solution. L'analyse concerne habituellement le problème alors que la conception et l'implémentation concernent la solution.

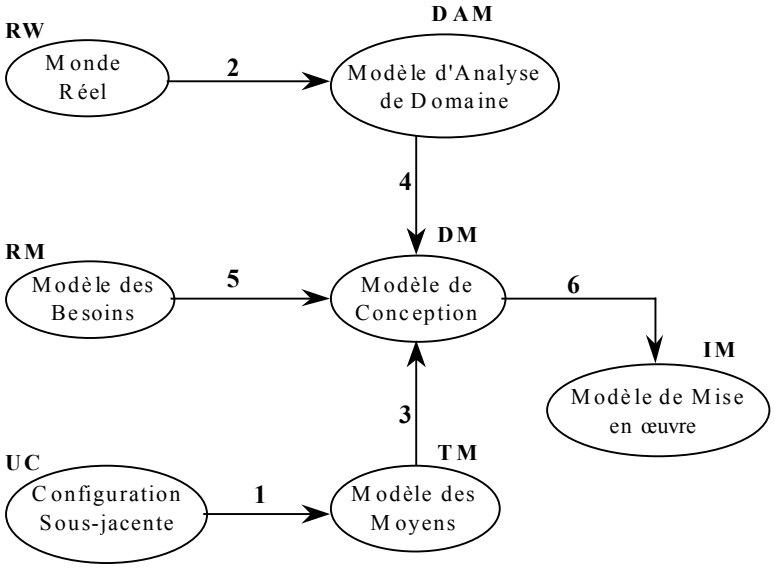

Figure 16 : Les modèles de base du schéma OSM [Bézivin, 1995]

L'analyste construit les objets du modèle d'analyse de domaine (DAM) à partir de l'observation du monde réel (②). Le concepteur de librairie (ou modèle de configuration) construit les objets du modèle des moyens (TM) à partir de l'observation de la configuration (①). Le concepteur d'application construit les objets du modèle de conception (DM) par sélection et fusion des objets du modèle d'analyse et du modèle des moyens techniques (③,④). Son rôle est central et prépondérant dans le développement par objets. Il s'appuie, dans son travail, sur le modèle des besoins (RM) qui précise ses objectifs (⑤). L'implémenteur construit enfin les objets du modèle de mise en œuvre (le code) à partir du modèle de conception représentant l'architecture du logiciel (⑥).

Ce schéma de travail, baptisé *schéma OSM*, fixe de manière claire les objectifs et l'interdépendance des phases de développement par objets.

Le modèle d'analyse de domaine ne cherche pas à réaliser des fonctions, mais plutôt à représenter le monde réel d'une façon aussi fidèle et neutre que possible (formalismes OMT, etc…).

Le modèle des besoins se compose principalement d'un ensemble de *scripts de besoins*. Un script de besoins est une entité décrivant une partie de ce que le système doit faire. Cette spécification d'un objectif se situe dans le contexte d'un modèle d'analyse de domaine et s'appuie sur un modèle de moyens techniques.

Un script de besoins est composé tout d'abord d'une facette décrivant ce qui doit être réalisé. Le niveau de formalisation de cette description est variable. Une seconde caractéristique est la facette qui spécifie les conditions de déclenchement d'un script. Le contrat réalisé par le script est également précisé par un couple de *pré-condition* et *post-condition*. Enfin le script est lié aux différents objets utilisés du DAM ou du TM par une relation *d'utilisation*. Les scripts de besoins ont également des *relations mutuelles* comme des relations d'utilisation, de conditions d'enchaînement (avant ou après) ou de spécialisation.

En suivant la démarche proposée, les scripts de besoins vont pouvoir être exploités pour déterminer les objets métiers et les objets techniques à importer dans le modèle de conception.

Selon [Krief, 1992], l'activité de prototypage s'effectue en deux étapes : la *conceptualisation* et les *tests*.

Conceptualiser consiste à *réifier* les entités constituant le domaine étudié. Pour chaque type d'objet, il faut tout d'abord déterminer ses caractéristiques statiques et dynamiques, puis les intégrer au sein d'une classe qui, à travers ses instances, permettra de manipuler le *prototype*.

La résolution de problèmes passe par une première étape de formalisation de l'énoncé qui est une spécification informelle. Ceci permet d'expliciter l'énoncé, de supprimer les redondances, les incohérences et les informations superflues ainsi que de lever les ambiguïtés ou les contraintes implicitement ajoutées, qui rendent la résolution impossible.

La deuxième étape, dans le processus de prototypage, consiste à tester et manipuler, donc à rendre actif le prototype implanté.

La résolution d'un problème est issue de la synergie entre un expert du domaine et un concepteur. Cette résolution aboutit à l'obtention d'un *modèle logiciel* permettant de formaliser et de résoudre les problèmes inhérents à ce domaine. Ceci est obtenu par validation, remise en cause et affinements successifs de divers *prototypes logiciels*.

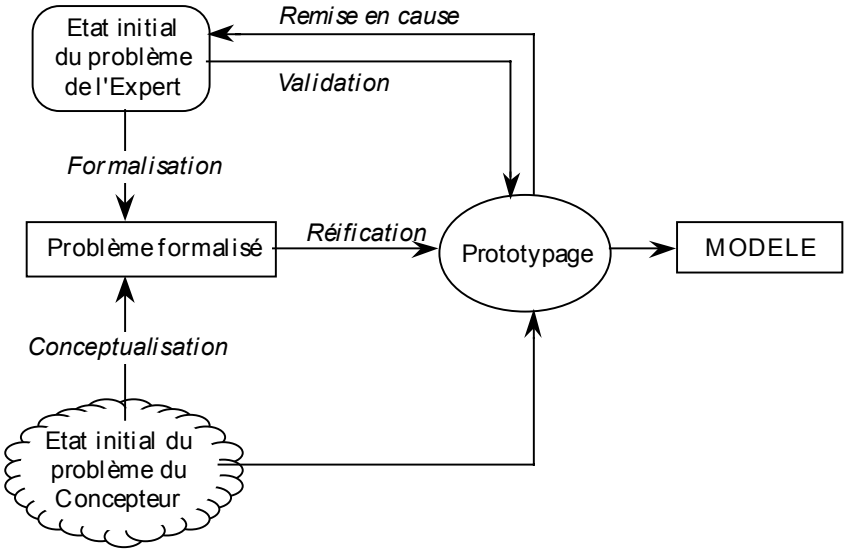

Figure 17 : Prototypes et Modèle logiciels [Krief, 1992]

En conclusion, nous pouvons dire que la conception incrémentale, au cœur de laquelle se trouve le prototypage, rend le processus de développement d'un logiciel visible tout en réduisant les risques et en améliorant sa manipulation. Lorsqu'un logiciel ne franchit pas un test d'acceptation, à la fois le développeur et le client peuvent « voir » et se mettre d'accord sur l'effet des imperfections sur la (les) fonctionnalité(s) requise(s).

Nous allons maintenant préciser dans quelle mesure des outils CASE ou méta-CASE peuvent être bénéfiques pour un processus de développement d'applications en général et de spécification de connaissances éducatives en particulier.

2. Les outils CASE et méta-CASE

Il est tout d'abord important de fournir quelques définitions sur ces termes ou plutôt acronymes. C'est ce que nous faisons dans le premier paragraphe. Le second paragraphe justifie la nécessité d'utiliser de tels outils pour lesquels nous présentons les bases conceptuelles. Dans le troisième paragraphe, nous concluons en fournissant une liste non exhaustive de quelques produits commerciaux et non commerciaux.

2.1. Définitions

En génie logiciel plusieurs approches permettant de décrire la spécification de logiciels sont utilisées. Beaucoup d'entre elles présentent la spécification graphiquement, à l'aide de diagrammes. Il y a de nombreuses techniques différentes de description, qui sont applicables à diverses étapes du développement du logiciel. Ces techniques sont appelées *méthodes de développement de logiciels*. Il

convient de souligner ici que les méthodes de développement de logiciels peuvent être utilisées non seulement pour le développement de logiciels, mais aussi juste pour la spécification d'une information donnée. Parmi les approches de développement de logiciels les plus populaires, citons MERISE, SADT, SA-SD, ou encore SSADM et bon nombres d'approches orientées objets comme OOA, OMT, Booch, etc...

2.1.1. Qu'est ce que CASE ?

L'acronyme de « *Computer Aided Software Engineering* » est un terme générique décrivant un support automatisé ou semi-automatisé pour le génie logiciel.

Les outils CASE sont utilisés par bon nombre de gens travaillant sur des projets de logiciels, depuis l'analyse des besoins, à travers diverses étapes de conception, de codage et de maintenance, jusqu'aux changements et améliorations majeures. Les objets de toutes ces activités seront mémorisés dans une base de données spéciale, appelée bibliothèque logicielle (*« software repository»*).
L'information qui est mémorisée dans la bibliothèque logicielle et qui décrit les éléments d'un projet logiciel, est appelée *spécification du logiciel*. Le format de la spécification du logiciel peut varier largement, en fonction de l'information concernant le logiciel qui est mémorisée. Dans le cas le plus simple, ce peut être le texte source d'un programme. Dans beaucoup de cas, cependant, la structure de cette information est beaucoup plus sophistiquée.
Un large éventail d'outils CASE existe aujourd'hui. Ils supportent la plupart des activités, telle que la planification, la gestion de projets, la spécification des besoins, l'analyse et la conception de système, la documentation, le prototypage, le codage, le déboguage ou le « *reverse-engineering* ». Les outils CASE peuvent être utilisés à diverses étapes du développement de logiciels. Les outils CASE les plus fréquemment utilisés sont les compilateurs.

2.1.2. Qu'est ce que méta-CASE ?

C'est un support automatisé ou semi-automatisé pour développer des outils CASE.
La plupart des systèmes méta-CASE sont justes des outils CASE génériques, paramétrables. Les caractéristiques basiques des systèmes méta-CASE sont qu'au lieu de fournir un environnement de génie logiciel, comme la plupart des systèmes traditionnels CASE le font, ils sont capables de fournir un nombre illimité d'environnements. Dans les systèmes méta-CASE, les environnements sont définis isolément des principaux composants logiciels. Les systèmes méta-CASE requièrent qu'une méthode (ou plusieurs) soit donnée avant que l'outil soit utilisé pour la production réelle. Donc, un système méta-CASE peut générer plusieurs outils CASE, en fonction de la méthode utilisée.

La technologie méta-CASE a pour but de capturer la spécification de l'outil CASE souhaité puis de générer cet outil CASE à partir de cette spécification.

2.2. Pourquoi de tels outils ?

Le cycle de développement d'un logiciel est le processus par lequel les systèmes sont spécifiés, conçus, implémentés, testés et maintenus. Une *méthode de développement* définit, non seulement comment prendre en compte une ou plusieurs activités dans le cycle de développement, mais aussi les entrées et sorties de ces activités. Une méthode modélise une partie du cycle de vie.
En outre, de façon conventionnelle, les cycles de vie suivent un développement en cascade ou en spirale (pour le prototypage). Cependant, ce sont juste des modèles qui ne représentent pas la réalité dans la mesure où chaque organisation suit ses propres standards. Ceci reflète la structure de l'organisation, son style de gestion, et l'importance relative accordée à la qualité, au coût, à son expérience, et à d'autres facteurs et niveaux de capacités. Il ne peut donc y avoir un cycle de vie unique standard parce que le caractère unique des organisations est l'essence même des systèmes commerciaux compétitifs qui suivent une amélioration permanente.
La solution idéale est de personnaliser les méthodes selon le cycle de vie propre à l'utilisateur. Cependant, sans les services d'automatisation fournis par les outils CASE, la plupart des méthodes ne sont pas attractives, et malheureusement la technologie utilisée en général pour créer des outils CASE ne permet pas une telle personnalisation. Bien sûr, ceci est source de motivation et le résultat est la technologie méta-CASE.

Il en résulte donc des outils méta-CASE, appelés aussi méta-outils, qui sont des outils CASE pour construire des outils CASE. La plupart des méta-outils existants impliquent :

- la personnalisation d'une bibliothèque de données, fournie avec l'outil méta-CASE, et destinée à la méthode de génie logiciel supportée,
- des fonctionnalités de construction d'éditeurs (à la fois diagrammatiques et textuels), et
- un langage de haut niveau pour la manipulation des données de la bibliothèque.

En conclusion, construire des outils en utilisant des méta-outils provoque des avantages énormes. Les outils peuvent être développés pour correspondre au processus de développement unique d'un individu ou d'une organisation, rendant le développement plus efficace, et de plus grande qualité.

L'objectif d'un méta-outil est donc de saisir la spécification de l'outil nécessaire puis de générer l'outil à partir de la spécification.

La méta-technologie dépend de la compréhension du domaine auquel nous voulons l'appliquer. A partir d'exemples d'applications dans le domaine, il convient de dériver un modèle d'applications typiques à ce domaine : c'est la recherche d'une application générique. Une application générique est spécialisée pour créer une application spécifique en fournissant des valeurs pour ses paramètres. Ces paramètres peuvent être soit des données soit du code. Les paramètres de l'application générique servent ensuite de base pour la spécification d'une application spécifique. L'application générique est la base du système d'exécution supportant les applications spécifiques.
Si les paramètres sont assez simples, ils peuvent être fournis directement au système d'exécution. Habituellement, ce n'est pas le cas et le système utilisant la méta-technologie est amélioré en permettant à la spécification d'être produite dans une certaine façon plus commode et significative puis en utilisant un générateur pour créer les paramètres.
Cela conduit à un système méta-CASE avec trois composants :

- le composant de *spécification* utilisé pour créer la spécification de l'outil,
- le composant de *génération* utilisé pour transformer la spécification en des paramètres pour l'outil générique,
- le composant *d'exécution*, c'est-à-dire l'outil CASE générique lui-même.

Par conséquent, les trois raisons principales à l'utilisation d'un environnement méta-CASE sont :

- vous avez votre approche méthode,
- vous voulez modifier les méthodes standards,
- vous voulez intégrer plusieurs méthodes.

Ainsi, les principaux avantages de l'approche méta-CASE améliorant les environnements de développement de logiciels sont :

- une grande réduction du temps et coût de développement d'un environnement CASE.
- le système applique une description formelle de l'environnement de développement. La description formelle d'un environnement particulier est moins ambiguë et plus facile à comprendre et à analyser qu'une description informelle.
- une comparaison d'environnements équivalents est facilitée par une définition exprimée dans un méta-modèle de description commun.
- le méta-modèle commun semble fournir une bonne base pour l'intégration d'environnements différents, ce qui reste encore aujourd'hui une des tâches les plus délicates en génie logiciel.

En outre, nous pouvons citer les travaux de [Eriksen et Stage, 1997] qui démontrent qu'une introduction de la technologie CASE dans les phases initiales du cycle de développement augmente grandement la qualité du produit final dans la mesure où le concepteur peut se concentrer sur la compréhension des besoins de l'utilisateur.

[Tolvanen, 1997] démontre également que des environnements méta-CASE augmentent la flexibilité et l'adaptabilité des méthodologies à plusieurs niveaux :

- le test des méthodologies par prototypage,
- la construction de bibliothèques méthodologiques, et
- la construction d'outils CASE spécifiques à l'application.

C'est dans ce sens que l'utilisation d'outils CASE et méta-CASE au sein d'un environnement de spécification des connaissances éducatives s'impose. En effet, il s'agit d'une part pour le pédagogue utilisateur de manipuler des éléments de connaissance supportés par des outils CASE dont la production est issue d'un environnement méta-CASE pour lequel nous allons détailler la structure fonctionnelle et les modalités de représentation de ses composants dans la troisième partie de cette thèse.
Nous allons maintenant prendre connaissance de divers outils méta-CASE commerciaux et non-commerciaux qui existent.

2.3. Quelques exemples de produits méta-CASE

Nous présentons ci-dessous une liste non exhaustive de produits méta-CASE, commerciaux ou non, et pour certains d'entre eux une présentation plus complète est fournie en annexe 2.

2.3.1. Quelques produits méta-CASE non commerciaux

Nous pouvons citer les produits suivants :

- « *ConceptBase* » de Rheinisch-Westfälische Hochschule à Aachen (Allemagne) est un gestionnaire déductif d'objet pour des méta-bases de données. Il implémente « *Telos* », un langage de modélisation conceptuelle aux propriétés déductives et *orientées* objet.
- « *GOODSTEP* » de l'université de Dortmund (Allemagne) est une base de données orientée objet pour des traitements de logiciel. Au dessus de la base de données deux générateurs d'outils et un logiciel modélisation de traitement et système de représentation ont été construits.
- « *HotDraw* » du département d'informatique à l'université d'Urbana-Champaign dans l'Illinois (U.S.A.), est un cadre d'infographie pour les éditeurs graphiques structurés et animés. Il a été utilisé pour créer beaucoup d'outils CASE différents.
- « *Metaview* » de l'Université d'Alberta au Canada.
- « *HARDY* » de l'AIAI à Edimbourg (Royaume-Uni) est un outil diagrammatique basé sur la notion d'hypertexte. Il s'exécute sur Sun et PC.
- « *Meta-CASE* » à l'université de Sunderland au Royaume-Uni.

- « *Method Engineering* » de l'université de Twente aux Pays-Bas.

2.3.2. Quelques autres produits méta-CASE commerciaux

Nous pouvons citer les produits suivants :

- « *MetaEdit* » par MetaCase Consulting Oy à Jyväskylä (Finlande) est un outil méta-CASE (PC) disposant d'une grande bibliothèque de méthodes.
- « *Graphical Designer* » par Advanced Software Technologies, Inc. est un produit Unix avec une interface utilisateur graphique intuitive et facile. Il supporte de nombreuses approches orientées objet et génère le code. Il peut être personnalisé et aussi supporter d'autres approches, comme SA et SD.
- « *ObjectMaker* » par Mark V Systems Ltd. à Encino, CA (U.S.A.), est un outil multi-plate-forme pour l'analyse et la conception. Plus de 30 approches sont supportées et de nouvelles approches peuvent être additionnées. L'outil peut être augmenté avec des modules de génération de code et de reverse engineering.
- « *IPSYS TBK* (*Tool Builder's Kit*) / *Toolbuilder* » pour les stations de travail X.
- « *Intersolv »* de la société PVCS.
- « *Graphtalk* » de RANK XEROX distribué par Parallax Software Technologies.
- « *STP* » (Software Through Pictures).
- « *REFINE* » par Leverage Technologists, Inc. à Rockville, MD (U.S.A) permet l'analyse et la réinginérie pour une variété de langages, incluant Ada, C, COBOL et FORTRAN.
- « *Paradigm Plus* » par Protosoft Inc., à Houston, Texas (U.S.A.).

Nous allons maintenant présenter le processus de développement que nous utilisons et y étudier la place occupée par le prototypage.

Paragraphe 2 : Notre environnement de développement d'enseignements

Nous allons tout d'abord présenter comment est abordé le cycle de développement des logiciels éducatifs en général. Ceci nous amène à proposer un processus « hybride » d'extraction des connaissances relatives à l'apprentissage basé sur le prototypage rapide et intégré au sein d'un environnement méthodologique respectant deux principes fondamentaux que sont le guidage et la flexibilité.

1. Cycles de vie des logiciels éducatifs

De façon usuelle, la réalisation d'un logiciel éducatif respecte les étapes suivantes.
Dans un premier temps, les pédagogues fournissent une description textuelle ou orale du produit qu'ils désirent. Ceci sert de support aux informaticiens qui se chargent de fournir un produit « qui tourne » respectant ce cahier des charges. A ce niveau, apparaissent déjà deux restrictions majeures. D'une part, les spécifications des pédagogues ne sont pas obligatoirement complètes, ce qui en terme de génie logiciel peut s'avérer très coûteux, et d'autre part, il y de forts risques d'ambiguïté et de redondance d'informations, sources également d'erreurs. Par conséquent, il incombe aux informaticiens de résoudre des problèmes d'ordre purement pédagogique !
Vient ensuite une phase d'évaluation du logiciel éducatif par les pédagogues pour vérifier si il y a bien adéquation entre les souhaits et la réalité. Ceci génère des conflits entre les différents acteurs car il peut y avoir remise en cause des structures de données choisies, création de nouveaux problèmes par effet de bord, etc…
Il est clair que cette méthode de travail est à bannir.

L'existence d'une équipe pluridisciplinaire (pédagogues, informaticiens, ergonomes, etc…) pour la conception d'EIAO fait qu'une difficulté importante réside dans les problèmes de compréhension entre les différents acteurs : comment comprendre le langage de l'autre et ses intentions. Même si chacun de délimite strictement son domaine d'intervention, il en résulte que chacun attend que les autres assurent les compléments indispensables et cela ne conduit pas à l'émergence de dynamiques suffisantes. Par conséquent, selon [Bruillard et Vivet, 1994], la coordination de cette équipe doit être assurée par un spécialiste EIAO : le didacticien.

[Nodenot, 1992] définit M.A.G.E., un Méta Atelier de Génie Educatif. Il y préconise notamment l'utilisation d'un cycle de vie spécifiquement adapté aux logiciels éducatifs. Basé sur la collaboration des différents acteurs, il est présenté notamment sous deux formes différentes illustrées par la figure suivante.
Le langage profond servant de support à la communication entre les acteurs se nomme « Spec » [Berzins et Luqi, 1990]. Basé sur la logique des prédicats du premier ordre et les types abstraits de données, [Nodenot, 1992] a démontré qu'il pouvait servir à la définition de logiciels éducatifs. Cependant la complexité de ce langage le rend raisonnablement impossible à manipuler par un pédagogue non-expert en informatique.
De plus, bien que diminuant le risque d'inadéquation entre les besoins des pédagogues et le produit final, l'informaticien occupe toujours la place de concepteur du produit.

Notre point de vue est alors de situer le commanditaire du produit, ici le pédagogue, au centre même du processus de construction. En effet, lui seul connaît ses besoins et l'informaticien doit alors lui fournir les moyens de s'exprimer de la façon la plus autonome possible afin d'aboutir à une spécification formelle des besoins qui sert ensuite de base de travail validée pour le développement informatique du produit.

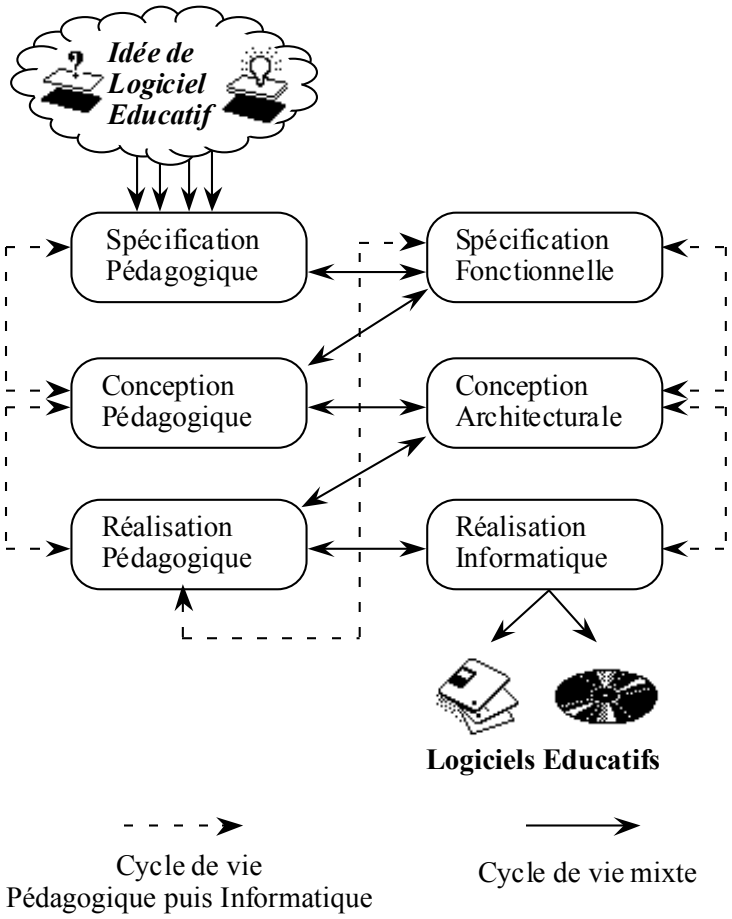

Figure 18 : Cycles de vie des logiciels éducatifs [Nodenot, 1992]

Nous préconisons donc l'utilisation d'un environnement logiciel résolument orienté vers une utilisation par des pédagogues non-experts en informatique ; ceci permettant de diminuer les risques de dysfonctionnement dus à une trop grande diversité de communications entre acteurs.

Ceci nous conduit à proposer un cycle de vie « hybride » basé sur la notion de prototypage :

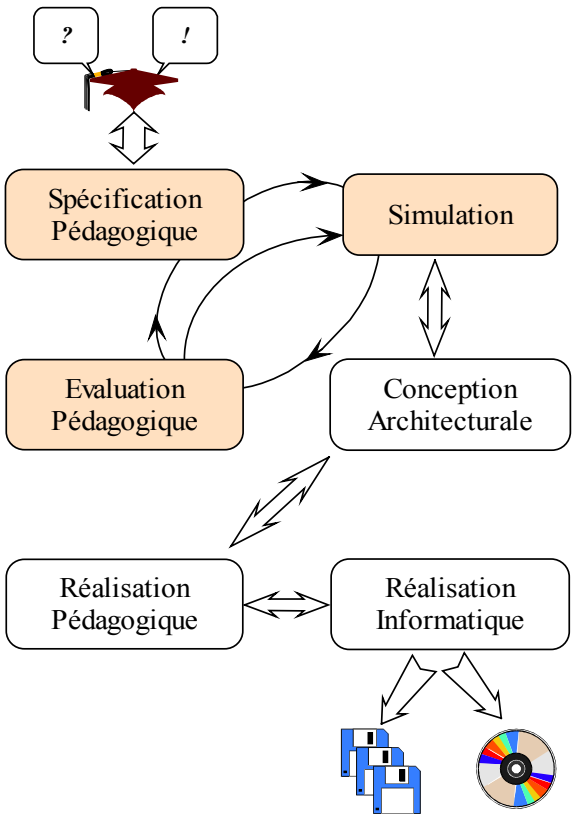

Figure 19 : Notre cycle de vie des logiciels éducatifs

Durant la phase de spécification, l'idée est de chercher à définir les situations que l'on veut créer. Pour cela, nous avons besoin d'un système de spécification, pour les décrire, voire pour raisonner dessus et déterminer quels moyens sont nécessaires pour les mettre en œuvre.
Cette spécification doit dégager le ou les différents rôles confiés aux outils logiciels ainsi que les contraintes générales intervenant dans leur conception.

Ensuite une phase de prototypage, qui suppose aussi une certaine expertise technique, consiste en la traduction des spécifications en terme de contraintes sur la conception et réalisation technique correspondante. Selon [Woolf et Hall, 1995], les simulations peuvent prendre plusieurs formes classées en deux grandes catégories : celle basée *sur les scénarios* et celle basée s*ur les connaissances*.
D'un côté les *simulations basées sur les scénarios* engagent l'utilisateur dans des situations types mais la plupart de ces simulations ne fournissent que peu de chemins de résolution, tiennent peu compte des connaissances externes et restent peu souples pour une adaptation aux besoins de l'utilisateur courant.
D'un autre côté, les *simulations basées sur les connaissances* contiennent un modèle de la situation et peuvent utiliser un planificateur, un identificateur de plan ou un modèle de l'utilisateur. Ces simulations nécessitent toutefois des représentations complexes et des structures de contrôle sophistiquées pour répondre avec souplesse à l'utilisateur.

Alors que les simulations basées sur les scénarios classent souvent les actions des apprenants comme correctes ou non et fournissent peu de conseils, les systèmes à base de connaissances peuvent raisonner sur les buts pédagogiques et identifier ceux vers lesquels la simulation sera dirigée.

La phase de validation, basée sur le concept de situation permet de mettre en place trois types d'évaluation identifiés par [Bruillard et Vivet, 1994]:

① *Intérêt éducatif*

Il est centré sur la pertinence éducative de la situation construite (les outils en étant une partie prenante),

② *Evaluation technique*

Elle concerne la production technique elle-même, permettant de juger la façon dont les outils respectent les contraintes spécifiées,

③ *Evaluation ergonomique*

Elle analyse l'adéquation entre le produit (ses fonctionnalités et les modes d'appropriation des usagers) et les fonctions pour lesquelles il est construit (problématique de l'ergonomie cognitive).

Cette phase est ici particulièrement importante dans le mesure où, pour la plupart des applications, elle n'est réalisée que tardivement, ce qui n'autorise que des modifications de surface et provoque une obsolescence rapide. L'évaluation doit donc être un processus dynamique itératif et synchrone en temps réel. Les travaux de [Millet, 1997], menés au sein de notre laboratoire, offrent ici une complémentarité nécessaire dans la mesure où ils proposent une méthode générale d'évaluation automatique des connaissances reposant sur le concept d'agent réviseur. Ceci permet la réingénierie des connaissances à partir d'une évaluation multi-critères (informatique, ergonomique, didactique,...).

Cette *approche incrémentale* basée sur le prototypage a pour principal avantage une meilleure implication des utilisateurs dans la conception de l'application, donc une diminution des risques d'inadaptation de celle-ci à leurs besoins réels.

Le respect de l'ensemble de ces principes permet alors de prendre en compte de façon la plus précise possible les besoins et attentes des différents acteurs en proposant des architectures fonctionnelles et logicielles d'outils. Cependant un aspect primordial pour la construction des enseignements est négligé : comment aider les enseignants à utiliser de tels outils.

Le paragraphe suivant vise ainsi à préciser le quand et le pourquoi des outils à mettre en place.

2. Conclusion : principes fondamentaux de notre environnement

Toutefois, nous ne nous concentrons pas sur la mise en place de logiciels éducatifs, mais nous essayons plutôt de faciliter la tâche de l'enseignant dans la préparation de ses séquences pédagogiques en définissant les apports de l'outil informatique. L'objectif que nous nous sommes fixés consiste à leur proposer un environnement facilitant l'extraction et la manipulation des connaissances relatives à l'enseignement-apprentissage.

Cependant l'environnement de spécification pédagogique que nous proposons ne pourra susciter l'intérêt des utilisateurs, c'est-à-dire du personnel enseignant que si, grâce à cet environnement, tout enseignant peut jouer trois rôles complémentaires :

① celui de bâtisseur dont les idées seront mises en valeur, affinées et mises en application grâce aux fonctionnalités de l'outil,

② celui de l'instructeur qui pourra modifier, paramétrer tout ou partie d'enseignements archivés par l'environnement en les adaptant ainsi aux situations particulières de sa classe,

③ celui de l'évaluateur qui peut ainsi se mettre en position d'apprentissage et par la suite critiquer les situations pédagogiques spécifiées.

Nous retrouvons une analogie avec les trois phases significatives de l'enseignement telles que nous les avons présentées dans le chapitre 6 paragraphe 2.

D'un point de vue plus technique, un tel environnement de spécification des connaissances doit répondre au cahier des charges suivant :

- il doit assister les pédagogues dans leurs choix lorsqu'ils sont en train de spécifier leurs connaissances. Cette aide doit prendre plusieurs formes :
  - ♦ un support méthodologique,
  - ♦ un système d'aide à la sélection de composants logiciels en fonction de critères pédagogiques.
- il doit être capable d'exécuter les spécifications décrites par les pédagogues afin de faciliter leur évaluation d'où l'importance *des fonctions de prototypage* agissant sur les spécifications pédagogiques.

Notre but est aussi de rendre les enseignants confiants dans l'informatique et ce, en démontrant que l'outil informatique ne limite pas forcément la créativité et peut rendre le travail de l'enseignant plus rapide et efficace. Il s'agit donc de favoriser la réutilisation des spécifications de composants pédagogiques déjà réalisés mais aussi de favoriser la réutilisation de méthodes efficaces de spécification mises en œuvre par des enseignants. Le processus de spécification envisagé ne doit donc pas être figé, fermé mais susceptible d'évolution et supportant la réutilisation.

Dans le domaine de l'enseignement (comme d'ailleurs dans un cas plus général), la spécification commence par la confrontation limitée et floue entre un système à produire et une représentation

totalement informelle de la finalité du système. Il convient donc d'aboutir à un agrément entre enseignants, didacticiens, experts du domaine sur une spécification claire et bien comprise [25].

Pour examiner les besoins spécifiques qu'ont les enseignants désireux de spécifier des enseignements (voire des logiciels éducatifs), il est intéressant de partir d'un schéma général relatif à l'ingénierie des besoins présenté dans [Pohl, 1994].

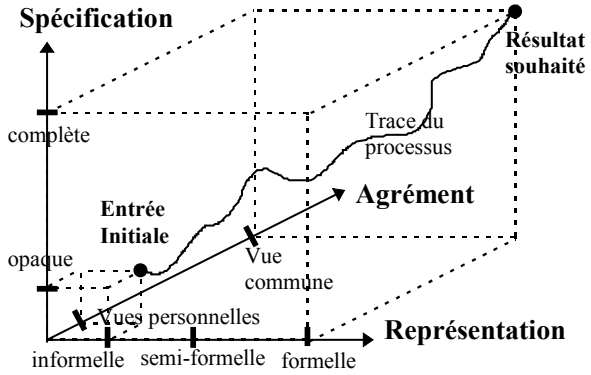

Figure 20 : Les trois dimensions de l'ingénierie des besoins [Pohl, 1994]

Ce schéma est basé sur trois axes :

- l'axe *représentation* va d'un niveau totalement informel (langue naturelle) à des représentations très formelles (spécification en Z,...),
- l'axe *spécification* traite de la clarté et de la complétude de la spécification produite. Il s'agit d'un axe orthogonal au premier axe car on peut très bien disposer d'un formalisme très complexe et aboutir à des spécifications très pauvres,
- enfin l'axe *agrément* stipule que l'activité de l'ingénierie des besoins doit se penser comme un processus de négociation entre partenaires ayant des compétences complémentaires.

Les deux axes les plus fondamentaux pour la spécification d'enseignements semblent donc être l'axe spécification et l'axe agrément, l'objectif étant que les outils d'aide à la spécification d'enseignements conduisent différents experts à se mettre d'accord sur une spécification non ambiguë et claire. Ces experts n'ont aucun a priori sur le(s) formalisme(s) de représentation mis à leur disposition mais c'est tout de même à ce niveau que bien des problèmes se posent :

- pour fournir aux pédagogues une *assistance* durant le processus de spécification, il faut bien être capable d'interpréter les modèles produits par ces derniers. Il est donc nécessaire que le formalisme d'expression de ces modèles soit non ambigu et interprétable par un ordinateur (à l'aide d'algorithmes ou d'inférences par système expert),

[25] Cette spécification n'étant pas forcément très formalisée, comme on l'exige souvent en informatique pour obtenir des spécifications exécutables donc vérifiables par prototypage avant d'entamer la conception du système.

- si l'on veut que les différents acteurs de la spécification (pédagogues, didacticiens, experts du domaine,...) ne se sentent pas contraints par le formalisme de représentation mis à leur disposition, il faut que les spécifications produites puissent être analysées selon différents points de vue (celui du pédagogue, celui de l'expert du domaine, celui de l'apprenant,...).

Par conséquent, ces quelques points montrent que pour que des enseignants acceptent des outils d'aide à la spécification des connaissances, il faut que l'environnement proposé fournisse une vraie *assistance adaptée* aux besoins des acteurs et soit en même temps suffisamment *flexible* pour autoriser différentes façons de spécifier, différentes vues sur une spécification et différents formalismes de représentation.

En outre, l'activité de prototypage, intégrée au sein de l'environnement méthodologique que nous préconisons, est la clé de voûte d'une résolution efficace et rapide d'un problème de représentation des connaissances. Il convient alors de présenter précisément :

- le (ou les) modèle(s) de représentation de connaissances,
- le (ou les) outil(s) nécessaires pour manipuler, tester et valider les connaissances spécifiées à partir de ces différents formalismes.

C'est l'objectif de la troisième partie de cette thèse.

Troisième Partie : Notre environnement de spécification d'enseignements

Résumé de la troisième partie

Après avoir défini dans la première partie le cadre général de nos travaux de recherche, nous avons présenté dans la seconde partie les principes généraux que doit respecter un environnement d'aide à la spécification des connaissances éducatives par un pédagogue. La recherche d'une solution nous a conduit à proposer un processus de spécification basé sur une représentation formelle des connaissances centrée autour des concepts de matrice disciplinaire et de situation-problème. Le prototypage rapide permet ici de mettre en place une évaluation précoce des besoins réellement exprimés par le pédagogue. Nous allons maintenant approfondir les différents outils et principes que nous avons effectivement utilisés dans le cadre de cette thèse.

Le chapitre 8 a pour objectif de préciser l'organisation des connaissances que nous préconisons. Ainsi, dans le paragraphe 1, nous montrons que la notion d'ontologie répond à nos attentes et nous permet alors de définir une ontologie propre au domaine de l'enseignement. Le paragraphe 2 permet ensuite de matérialiser la structure profonde de notre environnement de spécification pour chacune des diverses étapes du processus que nous avons précédemment défini.

Le chapitre 9 permet de mettre en évidence les trois principes majeurs, non disjoints, que doit respecter notre environnement : le guidage de l'usager, une flexibilité dans la manipulation et des possibilités de réutilisation des connaissances. Le paragraphe 1 permet également de mettre en évidence l'organisation des différents composants de l'environnement autour d'une architecture Modèle - Point de Vue - Contrôleur tendant vers une architecture à base d'agents. Cette architecture est également appliquable à l'interface-utilisateur diagrammatique de notre environnement décrite dans le paragraphe 2.

Enfin, le chapitre 10 présente la partie pratique de nos travaux de recherche. Nous y exposons non seulement les différents logiciels que nous utilisons, mais aussi les restrictions que nous avons effectuées pour notre prototype.

Plan de la troisième partie

CHAPITRE 8 : ORGANISATION DES CONNAISSANCES 115
Paragraphe 1 : Représentation des connaissances par ontologie 115
1. Les ontologies, un mécanisme de spécification 117
2. Vers une ontologie de l'enseignement 119
Paragraphe 2 : Représentation structurelle 120
1. Définition du contexte général 122
1.1. Représentation conceptuelle 122
1.2. Mise en pratique 122
2. Matrice Disciplinaire 122
2.1. Représentation conceptuelle 124
2.2. Mise en pratique 126
3. Focus 126
3.1. Représentation conceptuelle 126
3.2. Mise en pratique 126
4. Objectifs et Stratégies 127
4.1. Représentation conceptuelle 127
4.2. Mise en pratique 128
4.3. Conclusion 129
5. Situations d'enseignement 129
5.1. Représentation conceptuelle 136
5.2. Mise en pratique 130
5.3. Conclusion 131
6. Vue d'ensemble 131

CHAPITRE 9 : PRINCIPES D'INTEGRATION DANS UN TI D'AIDE A LA SPECIFICATION 133
Paragraphe 1 : Architecture de notre environnement 133
1. Trois principes de base : guidage, flexibilité et réutilisation 134
1.1. Guidage 134
1.2. Flexibilité 138
1.2.1. Différents points de vue sur une spécification éducative 138
1.2.2. Différentes façons de produire des spécifications éducatives 139
1.3. Réutilisation 140
2. Du schéma M.V.C. à l'approche M.PV.C. 141
Paragraphe 2 : Quelle interface utilisateur ? 144

CHAPITRE 10 : LA REALISATION DE OMAGE....149
Paragraphe 1. Quels outils ?....149
1. Les paradigmes objet et logique....149
2. Outils logiciels....152
2.1. LISP....152
2.2. PROLOG....153
2.3. CLIPS....154
Paragraphe 2. Notre prototype....155
1. Logiciels choisis....155
1.1. HARDY....155
1.2. wxCLIPS....156
1.3. wxWindows....157
2. Présentation du processus de développement de notre prototype et des outils développés....157
2.1. Les différentes étapes du processus....158
2.1.1. Matrice disciplinaire....158
2.1.2. Objectifs d'apprentissage....159
2.1.3. Stratégies globales d'enseignement....159
2.1.4. Situations-problèmes....159
2.2. Outils associés....160

Chapitre 8 : Organisation des connaissances

L'objectif de ce chapitre est de définir la meilleure représentation des connaissances dans le cadre de l'environnement résolument orienté dans un cadre pédagogique.
La variété des acteurs intervenant ainsi que la diversité de leurs opinions nous amène à étudier un champ à la croisée de plusieurs matières et nous conduit à proposer dans le paragraphe 1 une représentation des connaissances sous forme ontologique. Le paragraphe 2 est ensuite consacrée à la définition de la représentation structurelle des divers composants de chaque étape du processus de spécification mais aussi à l'enchaînement de l'ensemble de ces étapes.

Paragraphe 1 : Représentation des connaissances par ontologie

Tout au long des deux premières parties de cette thèse, nous avons démontré la pertinence de l'approche de l'enseignement par l'intermédiaire des concepts de matrice disciplinaire et de situation-problème qui permettent également une réelle transdisciplinarité. Nous avons précisé la définition de chaque terme ainsi que leur place dans un processus de construction d'enseignement.
Nous pouvons alors résumer l'activité d'enseignement par le tableau suivant :

Phases de l'enseignement	**Concepts invoqués**
Analyse	Connaissances déclaratives (Fait / Notion) et Connaissances procédurales
	Objectifs noyaux ou notions-clés
Réflexion	Connaissances procédurales
	Registre de formulation
	Actions
	Buts
	Connaissances déclaratives
	Registre de conceptualisation
	Champs notionnels (internes / externes)
	Niveau d'exigence
	Niveau des élèves
	Invariants
	Concepts intégrateurs
Anticipation	Représentations acquises des élèves

	Difficultés conceptuelles des élèves
	Objectifs prévus
	Obstacles anticipés
	Objectifs-obstacles
	Ressources disponibles
	Techniques disponibles
	Contraintes imposées
	Temps consacré
Evaluation	Obstacles réels
	Objectifs atteints
	Connaissances acquises et non acquises
Validation ou Remédiation	Situations d'application
	Situations de transfert
	Activités de Remédiation

Tableau 3 : Caractérisation de l'activité d'enseignement

La figure suivante reprend ceci de façon schématique :

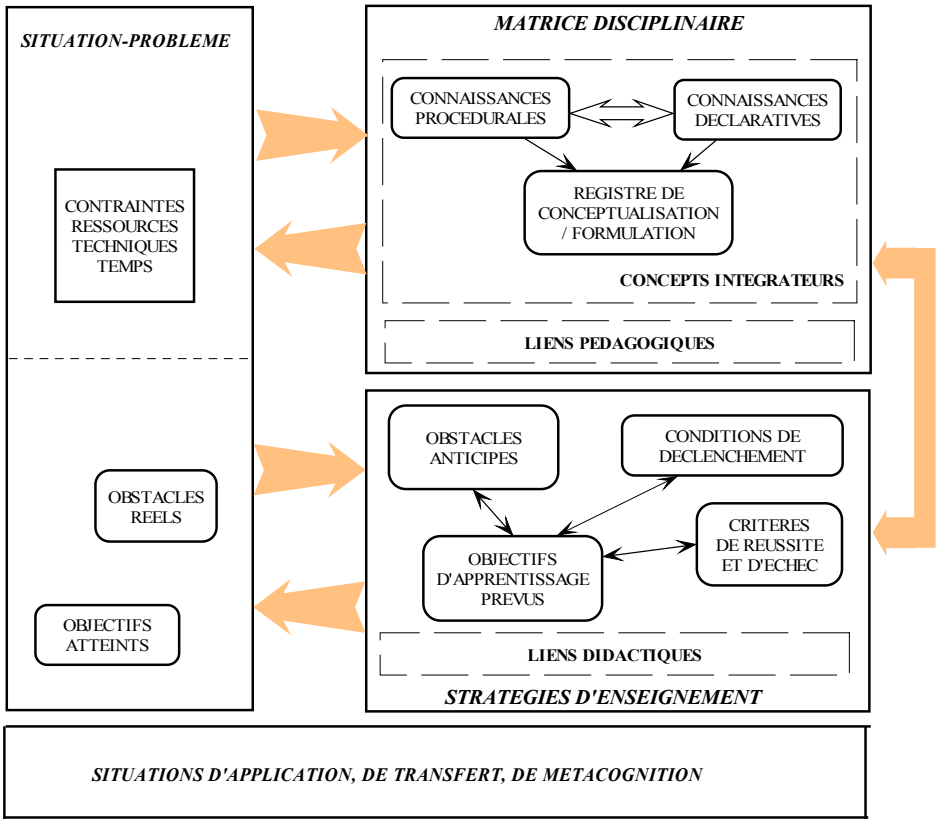

Figure 21 : Modélisation de l'activité d'enseignement

Nous allons maintenant proposer une organisation des connaissances qui réponde le mieux possible à l'ensemble des contraintes que nous avons précisé dans les deux premières parties de cette thèse.

1. Les ontologies, un mécanisme de spécification

Une propriété clé pour tout système d'information est sa structure ou architecture sous-jacente [Jacobson, 1993]. D'un point de vue chronologique, les systèmes d'information ont été développés de façon isolée, en fonction des *besoins des utilisateurs* du moment. Ceci a ainsi donné lieu à des systèmes adéquats mais dont la compatibilité avec les autres systèmes est limitée et la capacité de s'adapter aux changements des besoins en information des utilisateurs reste superficielle.
Une façon possible de résoudre ce problème est d'avoir une standardisation du contenu sémantique du système d'information. Les résultats de recherche dans le domaine des ontologies peuvent ici contribuer à ce processus de standardisation [26].

En effet, considérons les processus suivants : réserver une chambre d'hôtel pour un client ; prendre un rendez-vous avec une personne ou emprunter un livre à la bibliothèque.
En pratique, l'implémentation d'un système pour réaliser ces tâches sera probablement très différente de par son architecture logique et des perspectives pratiques. Les modèles de données et de processus sous-jacents de chaque application auront certainement très peu de ressemblance malgré le fait qu'une vue conceptuelle « naïve » identifiera clairement de nombreuses caractéristiques communes. D'un point de vue conceptuel, ce sont des instances du même processus : par exemple, le fait qu'un livre donné ne soit pas louable par deux personnes différentes en même temps est similaire au fait qu'un seul rendez-vous soit pris à la fois.

[Gruber, 1993] donne la définition suivante : « *une ontologie est une spécification d'une conceptualisation ; c'est une spécification explicite d'un sujet donné* ».
C'est une représentation formelle et déclarative incluant le vocabulaire (ou les noms) permettant de se référer aux termes du domaine relatif au sujet courant ainsi qu'aux états logiques qui décrivent ce que sont les termes, comment ils sont reliés les uns aux autres et comment ils peuvent (ou ne peuvent pas) être liés entre eux. Les ontologies fournissent, par conséquent, un vocabulaire de représentation et de communication des connaissances relatives à un sujet ainsi qu'un ensemble de relations existantes entre les termes de ce vocabulaire. Une autre définition utilisée par exemple par les philosophes est que les ontologies sont des spécifications suffisamment non ambiguës dans un langage formel.

[26] Des exemples sont notamment le « *mammoth CYC ontology* » à Cycorp, le « *Knowledge Sharing Effort* » à Stanford, le « *Enterprise Integration Laboratory (EIL)* » à l'Université de Toronto et le « *Enterprise Ontology Project* » à l'AIAI d'Edimbourg.

Durant les dernières années, de nombreux laboratoires de recherche ont développé des ontologies dans un but d'inculquer aux systèmes informatiques une compréhension limitée des interfaces en langage naturel. [Gruber, 1993] définit par exemple des ontologies permettant le partage et la réutilisation des connaissances.

L'intelligence artificielle se base sur le postulat selon lequel ce qui existe est ce qui peut être représenté. En effet, lorsque la connaissance d'un domaine est représentée selon un formalisme déclaratif, l'ensemble des objets qui peut être représenté est alors appelé l'univers du discours. Cet ensemble d'objets ainsi que les relations descriptibles entre eux sont représentés dans le vocabulaire représentatif avec lequel un programme représente ces connaissances. Par conséquent, dans le contexte de l'intelligence artificielle, il est possible de décrire l'ontologie d'un programme en définissant un ensemble de termes représentatifs. Dans une telle ontologie, les définitions associent les noms des entités dans l'univers du discours (par exemple les classes, relations, fonctions, et autres objets) avec un texte (lisible par un utilisateur) décrivant la signification des noms et les axiomes formels qui contraignent l'interprétation et la bonne utilisation de ces termes.

La description d'une ontologie se fait alors en deux parties :

- la définition de la terminologie,
- la spécification des concepts.

La définition de la terminologie consiste en l'identification des concepts pertinents et de leurs relations pour un domaine donné. Ceci peut se représenter sous forme de matrice ou de réseau de noeuds et d'arcs [27].

La spécification des concepts est la définition formelle de chaque concept utilisé en terme de propriétés et de règles sémantiques. La notation utilisée peut, par exemple, se faire à l'aide de spécifications algébriques.

Il est intéressant ici de mentionner les travaux de [Gruber, 1992] qui propose « *Ontolingua* », un mécanisme permettant d'écrire des ontologies dans un format canonique (basé sur KIF, acronyme de « *Knowledge Interchange Format* » [Genesereth, 1991]), afin qu'elles puissent être facilement traduites dans une variété de représentations et de systèmes de raisonnement.

[Vernada et Zelm, 1996] proposent une ontologie pour la modélisation d'entreprise, « CIMOSA », avec un langage de modélisation basé sur un modèle supportant la notion de processus et dirigé par les événements.

[27] Nous verrons plus loin, dans cette partie, que cette représentation diagrammatique s'avère particulièrement intéressante.

Il est également important de signaler ici que ce principe de représentation des connaissances par ontologie est concrètement pris en compte dans les projets MECANO, PROTEGE II et PROTEGE/WIN issus de l'Université de Stanford décrits en Annexe 1.

Nous avons défini ce qu'est la notion d'ontologie et dans quelles mesures elle peut être utilisée comme mécanisme de modélisation des connaissances.
Nous allons maintenant établir comment nous pouvons adapter ce concept à la spécification des connaissances pédagogiques.

2. Vers une ontologie de l'enseignement

Pour modéliser et évaluer les processus d'enseignement, des outils et des techniques précis et formels de modélisation pédagogique sont nécessaires. Il convient également de représenter formellement les différents types de concepts, leurs propriétés associées et les relations existantes entre eux selon une terminologie propre au domaine. Nous retrouvons ici la définition d'une ontologie que nous souhaitons appliquer à la spécification des connaissances relatives à l'enseignement.

Dans le domaine de l'enseignement, nous pensons pouvoir constituer un lexique de termes standard utilisé par les différents acteurs. En effet, l'usage montre d'une part que même si le vocabulaire employé pourrait laisser croire à des distinctions sémantiques, bien souvent la signification est identique. De façon inverse, parfois, des termes identiques utilisés par des personnes différentes font référence à ces concepts différents. Nous pouvons citer par exemple le cas des différentes compétences que les enseignants doivent faire acquérir aux élèves du primaire (cycle des apprentissages premiers, cycle des apprentissages fondamentaux et cycle des approfondissements). Ces compétences, quoique enseignées depuis de nombreuses années ou décennies, sont maintenant consignées au sein de documents permettant de suivre la progression de l'élève tout au long de sa scolarité. Les enseignants utilisent par conséquent un vocabulaire standardisé qui apporte également de nombreux bénéfices du point de vue de l'échange des connaissances entre collègues.
C'est ainsi que les différents concepts que nous manipulons dans le cadre de l'enseignement-apprentissage sont ceux pour lesquels nous avons axé le processus de développement (seconde partie chapitre 6 paragraphe 2) : notion, objectif d'apprentissage, stratégie d'enseignement, situation-problème, situation d'enseignement, événement et lien didactique.

Nous définissons un modèle d'enseignement comme une abstraction de la réalité dans un objectif de compréhension commune, de simulation, de contrôle ou de représentation d'une situation pédagogique. C'est alors un consensus, une vue commune de la part d'un ensemble de pédagogues.

Par conséquent, il convient de développer une architecture de référence vue comme un modèle conceptuel de haut niveau qui traduirait, en fonction des vues de l'utilisateur, des spécifications des connaissances qui pourraient servir de point de départ pour les phases ultérieures de développement.
Un critère essentiel pour l'architecture résultante est que les concepts utilisés dans le modèle peuvent être lus et compris directement par l'ensemble des acteurs de l'EIAO.

Ainsi, les recherches dans le domaine des ontologies sont basées sur l'hypothèse selon laquelle il existe une structure sous-jacente universelle. Son acceptation présume une vue du monde platonique ou structuraliste [28]. Or, il est important de noter que nombre de chercheurs n'acceptent pas cette position. En particulier les adhérents du « *soft system* » comme [Checkland et Scholes, 1990] qui pensent que ces structures universelles ne peuvent exister mais qu'elles « *représentent une façon de voir les choses, et par conséquent une façon de comprendre les activités complexes du monde réel* » [Avison et Fitzgerald, 1995].
Cependant, [Van Belle, 1996] dont les travaux de recherche sont axés sur la mise en œuvre d'une architecture de référence pour un modèle d'entreprise basé sur la notion d'ontologie, réfute ce second point de vue par l'argument principal suivant. Si l'on se réfère à une partie limitée de la gestion d'entreprise, comme la gestion financière par exemple, la taxonomie du système de comptabilité a été capable de servir avec plus ou moins de succès de modèle générique de l'activité depuis plus de cinq siècles. C'est dans cette optique que nous pensons que la spécification des connaissances dans le domaine de l'enseignement-apprentissage, basée sur la notion de situation-problème, peut se baser sur le modèle, ou plutôt le méta-modèle que nous proposons ci-dessous.

Paragraphe 2 : Représentation structurelle

L'ensemble des modélisations que nous allons présenter ci-dessous constituent la vision du système qu'a l'utilisateur, c'est-à-dire le pédagogue. Ainsi, malgré les risques de redondance et de non complétude des éléments participant aux différentes modélisations conceptuelles, leur présence se justifie par une pertinence pédagogique.
Nous allons tout d'abord présenter chaque composant majeur du méta-modèle [29], c'est-à-dire chaque phase du processus de développement, puis nous reviendrons sur un modèle simplifié global

[28] C'est ce qui est désigné par [Jayaratna, 1994] comme une vue ontologique du monde par des méthodologies structurées.
[29] En effet, par définition, un méta-modèle est un modèle qui décrit d'autres modèles c'est à dire que le méta-modèle pour le modèle d'une méthode décrit les concepts de la méthode ainsi que leurs relations. Il définit les modèles pouvant être construits avec la méthode et, comme le dit [Rumbaugh et al., 1994] il permet également de décrire les informations pouvant être capturées par des outils CASE supportant la méthode.

permettant un aperçu général du processus de spécification des connaissances que nous avons présenté.
Pour chaque élément clé du méta-modèle, nous développons ensuite un cas pratique d'utilisation qui constitue en fait un modèle, c'est-à-dire une instanciation de ce méta-modèle.
Le domaine sur lequel nous travaillons concerne l'enseignement du GLOM [30] ainsi que de l'Anglais de spécialité qui peut venir y puiser des exemples d'application linguistique [31].

Ceci nous permet par la suite de mettre en évidence les diverses fonctions des outils utiles pour la spécification des connaissances. Ces différents outils s'appliquent alors aux éléments mis en évidence durant la première modélisation conceptuelle.

Il est important de signaler tout d'abord que l'ensemble de ces méta-modèles est représenté avec la notation propre aux modèles objets dans la méthode OMT [Rumbaugh *et al.*, 1994]. Le choix de cette méthode se justifie non seulement par l'importance de cette méthode dans le monde du génie logiciel mais aussi par une puissance d'expression supérieure à d'autres méthodes objet comme OOA [Shlaer et Mellor, 1991]. En effet, par exemple, avec OMT la représentation des associations d'agrégation est explicite alors qu'elle ne l'est pas avec OOA.
Notons aussi que les différents exemples de ce paragraphe sont issus d'une application développée avec le prototype que nous avons implanté et utilisent par conséquent une notation diagrammatique à base de noeuds et d'arcs que nous justifions dans un paragraphe ultérieur de ce document.

[30] Acronyme de « Génie Logiciel : Outils et Méthodes ». Terme désigné dans le Programme Pédagogique National des IUT et regroupant les matières d'enseignement de l'ACSI (Analyse et Conception des Systèmes d'Information) et des bases de données.
[31] Ce choix est essentiellement dû à la composition de notre équipe de recherche.

1. Définition du contexte général

1.1. Représentation conceptuelle

Figure 22 : Représentation conceptuelle du « Contexte général »

Dans un premier temps, il convient de mettre en place un *modèle du domaine* qui est un découpage fonctionnel du domaine en sous-systèmes opérationnels (que nous nommons « domaine » dans le schéma) liés entre eux par des « liens associatifs » simples comportant uniquement un « nom » et une « description ».
A ce niveau, il est possible de fixer les valeurs d'un certain nombre de paramètres comme le « *niveau* » de l'apprenant. De même, il est possible à ce niveau de donner les valeurs d'autres « *variables* » comme par exemple une partie de ses connaissances acquises, etc...
Cependant, nous avons choisi de considérer ceci de façon plus fine et d'associer ces paramètres à un niveau de raffinement ultérieur pour la définition de la matrice disciplinaire. A un ou plusieurs domaines, correspond donc une ou plusieurs matrices disciplinaires.

1.2. Mise en pratique

Dans notre exemple, nous avons le schéma suivant :

Figure 23 : Une définition du contexte de GLOM

Dans le domaine de l'enseignement des bases de données, par exemple, nous pouvons distinguer plusieurs sous-systèmes comme la « modélisation des données » ou « des traitements » qui font référence à la notion de « méthodes » et par conséquent à un « ensemble d'extensions ». La dynamique des traitements induit le concept de « requête » qui permet de faire notamment des « valuations de la base ».

Après avoir dressé le contexte général de travail, nous modélisons ensuite, grâce à la matrice disciplinaire, le contexte particulier dans lequel le pédagogue situe son enseignement.

2. Matrice Disciplinaire

C'est le cœur de notre système. Le pédagogue fixe *l'univers du discours* en terme de *notions* auxquelles sont attachées des compétences. Comme nous l'avons vu dans la seconde partie (chapitre 6), ces notions sont en fait les principes unificateurs relatifs à l'objet que l'on veut faire acquérir à l'apprenant et se rapportant soit au domaine, soit transdisciplinaire. Ces objets sont reliés par trois types de liens, composition, référence et héritage, qui représentent respectivement les liens d'agrégation, d'association et d'héritage au sens courant du génie logiciel.

En fait, ces notions sont des *concepts intégrateurs* au sens de [Develay, 1993]. La matrice disciplinaire constitue donc le *noyau dur de la discipline* dans le sens où elle est la *trame notionnelle* regroupant les *champs notionnels*.

2.1. Représentation conceptuelle

Figure 24 : Représentation conceptuelle de la « Matrice Disciplinaire »

Chaque matrice disciplinaire se définit en fonction du contexte de l'enseignement-apprentissage par un nom, une description textuelle du rôle de la matrice ainsi que le nom de la personne qui opte pour cette modélisation (« propriétaire »).

L'ensemble des éléments clés de ce modèle (matrice, lien et notion) peut être mémorisé dans une bibliothèque, ce qui permet notamment une réutilisation des éléments existants mais également une comparaison des modélisations courantes avec celles effectuées par d'autres personnes.

Une matrice disciplinaire est composée par un ensemble de « notions » et de « liens » (dans le sens où nous les avons définis dans les deux premières parties de cette thèse).

Nous avons identifié trois sortes de liens : référence, composition et héritage qui se caractérisent par un « nom », une « description », un « propriétaire » ainsi qu'un ensemble de « contraintes » qui ont pour rôle d'assurer la cohérence du modèle. Par exemple, on ne peut avoir deux notions liées à la fois par un lien de composition et un lien d'héritage.

Une notion se définit notamment par un « registre de conceptualisation et de formulation » qui permettent de fixer le niveau d'enseignement dans le contexte courant. Les notions sont liées par des liens sur lesquels nous pouvons éventuellement faire porter une valeur de multiplicité. Ceci permet par exemple au pédagogue de définir des cardinalités de type (0 ; n) et même d'y ajouter une « contrainte » avec par exemple des cardinalités de type (2 ; n).

La présence de deux liens, nommés « Départ » et « Arrivée », entre les objets « notion dans son contexte » et « lien dans son contexte » permet de modéliser qu'il est possible de lier par un même lien une notion de départ avec plusieurs notions d'arrivée.

Une notion s'exprime aussi par l'intermédiaire de ses « propriétés » intrinsèques et des « opérations » qu'il est possible de faire. Nous avons choisi de représenter les notions sous forme ontologique car nous pensons qu'il existe une représentation générique de ce concept. Par exemple, plusieurs pédagogues peuvent désigner différemment le concept « d'attribut », de « champ dans une table » ou de « propriété » alors que ces trois éléments désignent la même chose. Les opérations sur ces éléments doivent alors être identiques. La « spécification formelle » ainsi que les « paramètres » de l'opération nous permettent de valider la définition proposée par le pédagogue par rapport à la définition ontologique profonde.

Nous proposons toutefois une variante à ce méta-modèle dans la mesure où la notion n'est pas directement composée de propriétés et d'opérations mais plutôt de caractéristiques qui sont des

propriétés et des opérations. Cette nuance conceptuelle se justifie par rapport à l'étape suivante du processus de développement dont nous étudions le méta-modèle dans le paragraphe suivant.

Figure 25 : Variante de la représentation conceptuelle de la « Matrice Disciplinaire »

2.2. Mise en pratique

Figure 26 : Matrice disciplinaire de GLOM

C'est à ce niveau que nous envisageons la *transdisciplinarité*. En effet, cela consiste à appliquer des connaissances procédurales inhérentes à l'enseignement de disciplines ou matières différentes sur une même trame notionnelle. Ainsi, par exemple, sur la trame notionnelle de GLOM, on peut appliquer les connaissances procédurales de GLOM comme dans la figure ci-dessus mais on peut y appliquer tout autant les connaissances procédurales relatives à l'Anglais de spécialité (cas 1). Ces mêmes connaissances procédurales relatives à l'Anglais de spécialité pourraient s'appliquer aussi sur une trame notionnelle relatives aux mathématiques ou à la gestion (cas 2). On peut alors parler de *trandisciplinarité instrumentale* (cas 1) et *procédurale* (cas 2) au sens de [Develay, 1993].

3. Focus

Nous entendons par ce terme le fait que le pédagogue doit focaliser son enseignement sur une partie de la matrice disciplinaire.

3.1. Représentation conceptuelle

Figure 27 : Représentation conceptuelle du « Focus »

Le pédagogue doit ainsi sélectionner une notion ou plutôt un groupe de notions, pour lesquelles il va définir quels sont les « pré-requis » et les « post-requis » du processus d'enseignement.
Une variante existe ici dans la mesure où un niveau de granularité plus fin peut être défini. En effet, les pré-requis et post-requis peuvent porter sur les caractéristiques d'une notion (ses propriétés et opérations) plutôt que sur la notion dans sa globalité. Ceci explique alors *a posteriori* la variante du schéma concernant la matrice disciplinaire.

Figure 28 : Variante de la représentation conceptuelle du « Focus »

3.2. Mise en pratique

Pour notre exemple, nous allons nous focaliser sur les notions et liens en rapport avec le « Modèle Conceptuel des Données » et le « Modèle Logique des Données » :

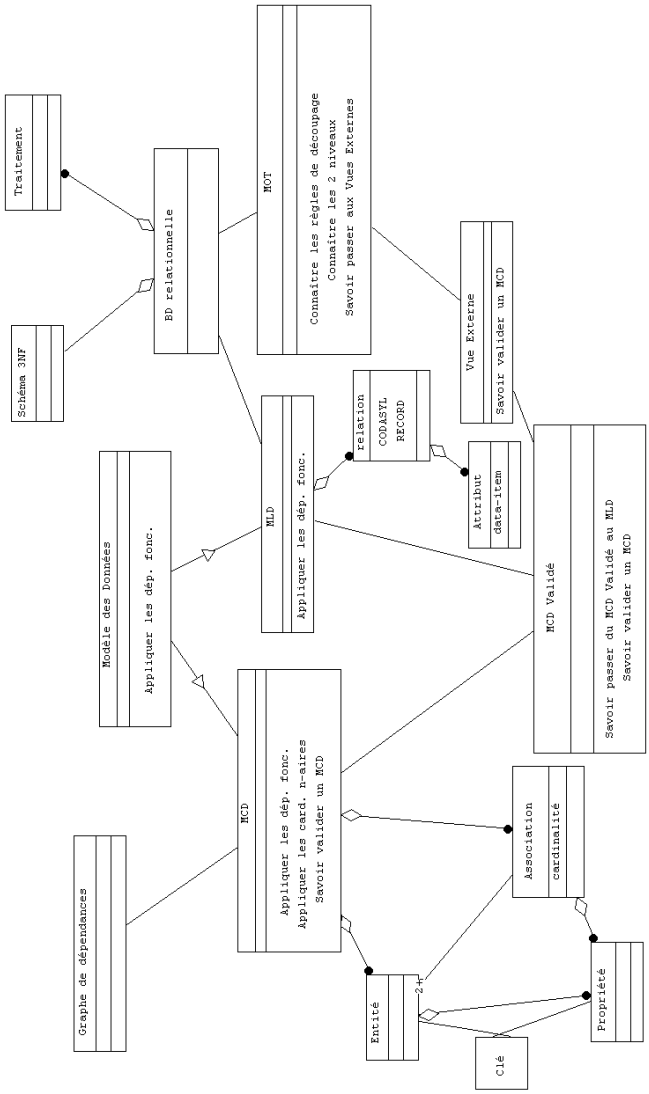

Figure 29 : « Focus » sur la matrice disciplinaire de GLOM

4. Objectifs et Stratégies

Par rapport à cette focalisation sur le contexte modélisé par la matrice disciplinaire, le pédagogue doit définir un ensemble d'objectifs d'apprentissage qui, une fois liés par des liens didactiques, constituent ce que nous appelons des stratégies d'enseignement.

4.1. Représentation conceptuelle

Le pédagogue associe à chaque notion ou plutôt groupe de notions des « objectifs pédagogiques ». Ceux-ci peuvent être complexes et raffinés en objectifs d'apprentissage plus simples (lien réflexif d'agrégation). Un « objectif-obstacle » et un objectif d'apprentissage « classique » sont des objectifs pédagogiques particuliers.

A chaque objectif (état), nous appliquons une (ou plusieurs) « stratégie(s) d'enseignement » qui représente en fait des objectifs d'apprentissage entre lesquels sont fixés des « liens didactiques » de type « précédence », « concurrence » ou « conséquence ».
Des critères de « pré-conditions » et de « post-conditions » sont aussi nécessaires pour permettre d'évaluer la stratégie d'enseignement choisie. En effet, il se peut que la stratégie choisie diffère de celle réellement utilisée (ou modélisée) ; des outils d'évaluation doivent alors permettre une assistance au pédagogue concepteur. En outre, l'évaluation du succès ou de l'échec de chaque stratégie d'enseignement permet de faire évoluer les « situations d'enseignement » à proposer.
De même que nous pensons qu'il est possible de représenter une « notion » de façon ontologique, nous retrouvons aussi ici le concept de « stratégie d'enseignement ontologique ».
A chaque stratégie d'enseignement mise en place, une (ou plusieurs) situation d'enseignement doit être proposée. Cette situation d'enseignement, selon que ce soit une situation-problème ou une situation plus « classique », fera appel aux concepts « d'objectif-obstacle » et-ou « d'objectif d'apprentissage ».

Figure 30 : Représentation conceptuelle des « Objectifs d'apprentissage » et des « Stratégies d'enseignement »

4.2. Mise en pratique

Nous allons tout d'abord préciser les objectifs d'apprentissage puis les stratégies d'enseignement associées.
C'est à ce niveau que l'on va préciser davantage le contexte spécifique de l'apprentissage. Les objectifs et par conséquent les stratégies d'enseignement différeront en fonction de la matière : GLOM ou bien l'Anglais de spécialité.
Dans le cas du GLOM, ceci donne par exemple le schéma suivant :

Figure 31 : Des objectifs d'apprentissage de GLOM

D'autres objectifs d'apprentissage sont également envisageables. Par exemple, il est concevable de segmenter les objectifs que nous présentons ci-dessus en plusieurs groupes, chaque groupe constituant un ensemble d'objectifs d'apprentissage.
Par rapport aux objectifs d'apprentissage fixés dans la figure ci-dessus, une stratégie d'enseignement possible (parmi de nombreuses autres) consiste à lier ces différents objectifs par les liens de précédence suivants :

Figure 32 : Une stratégie d'enseignement de GLOM

Ceci est une façon parmi d'autres de modéliser des stratégies d'enseignement de GLOM. En effet, une autre stratégie d'enseignement pourrait tout aussi bien se baser sur un autre ordonnancement des liens didactiques de précédence reliant ces objectifs d'apprentissage. De même, il est tout à fait pertinent de concevoir la présence d'autres types de lien exprimant la conséquence ou la concurrence entre plusieurs objectifs d'apprentissage.

Dans le cas de l'Anglais de spécialité, on pourra par exemple retrouver les objectifs suivants :

- Savoir exprimer la comparaison,
- Savoir exprimer la différence,
- Savoir mettre en place un raisonnement ascendant,
- Savoir mettre en place un raisonnement descendant, etc...

Ceci donne le schéma suivant :

Figure 33 : Des objectifs d'apprentissage de l'Anglais de spécialité

Une stratégie d'enseignement peut alors être :

Figure 34 : Une stratégie d'enseignement de l'Anglais de spécialité

4.3. Conclusion

A travers ces deux exemples [32], nous avons montré qu'une même matrice disciplinaire permet de servir de référentiel commun à des domaines d'enseignement différents.
Nous allons maintenant considérer les situations d'enseignement à mettre en place.

5. Situations d'enseignement

Les situations d'enseignement, que ce soit des situations-problèmes ou bien des situations plus conventionnelles (renforcement, remédiation, métacognition ou transfert), permettent au pédagogue de mettre en place les différents moyens qu'il juge utile pour aboutir à l'apprentissage des connaissances.

[32] Bien entendu ce nombre reste insuffisant et il conviendra de valider ceci sur de nombreux cas pratiques.

5.1. Représentation conceptuelle

Pour chaque scénario, c'est-à-dire chaque stratégie d'enseignement, le pédagogue propose différentes « situations d'enseignement » qui peuvent être par exemple des « situations-problèmes ». Dans ce cas, le pédagogue doit donc prévoir un certain nombre « d'obstacles » dont le franchissement lui permettra d'évaluer si l'apprentissage a été effectué ou non.
En situation d'enseignement, un même obstacle proposé à un groupe d'élèves peut correspondre, pour chacun d'eux, à des activités très différentes, en fonction des problèmes qu'ils rencontrent. Cela peut être par exemple :

- en français : faire un exposé, lire un texte, …
- en géographie : commenter un document, analyser un paysage, …
- en biologie : interpréter les résultats d'une expérience, …
- en mathématiques : résoudre un problème, faire une trace géométrique, …
- en EPS : arbitrer un match, …

Dans le cadre d'une situation-problème, un « objectif-obstacle » est à effectuer par l'apprenant à partir d'un ensemble de « consignes » données en usant des compétences procédurales et déclaratives qui sont des « caractéristiques des notions » dans le sens où elles sont présentées dans une matrice disciplinaire. Ces consignes sont relatives à plusieurs points : le « matériel » à utiliser, les « techniques » à mettre en œuvre, le « temps prévu » pour surmonter l'obstacle et différentes « contraintes organisationnelles ».

Les « situations d'enseignement » plus « classiques » (transfert, renforcement ou métacognition) permettent de tirer parti des résultats issus de l'évaluation des situations-problèmes et de vérifier si ceux-ci sont conformes à la réalité.

Figure 35 : Représentation conceptuelle des « Situations d'enseignement »

5.2. Mise en pratique

Dans les deux exemples simples suivants, nous nous restreignons à la définition de situations-problèmes. Les formalismes utilisés pour la modélisation des différents composants sont les suivants :

- un rectangle symbolise un obstacle,
- le texte libre associé représente les différentes contraintes,
- un ovale représente une situation-problème,
- un composant avec un trait au-dessous et au-dessus représente une notion de la matrice disciplinaire,

- un trait modélise les relations d'utilisation.

Les deux figures suivantes sont des exemples de définition de situations-problèmes appliquable à l'enseignement de GLOM et à celui de l'Anglais de spécialité :

Figure 36 : Situations d'enseignement de GLOM

Figure 37 : Situations d'enseignement de l'Anglais de spécialité

Chaque élément des schémas comporte également des pré-conditions internes de déclenchement qui n'apparaissent pas dans le schéma [33].

5.3. Conclusion

La mise en place des situations d'enseignement permet donc au pédagogue d'établir des liens avec les notions, ou plutôt les caractéristiques des notions, définies à travers la matrice disciplinaire et qui sont essentielles pour l'acquisition des connaissances.

La mise en place des situations d'enseignement permet aussi au pédagogue de « mesurer » la différence entre les objectifs supposés et les objectifs réels, et par conséquent d'affiner son enseignement. En effet, lorsque le pédagogue a caractérisé les points bloquant l'apprentissage (que ce soit après avoir mis en place des situations-problèmes ou bien directement, « par expérience »), le pédagogue doit être en mesure de les lever en définissant des situations extrêmement adaptées : les *situations* de *remédiation*, *de transfert*, de *renforcement*, de *métacognition*, etc…

Ceci permet aussi au pédagogue d'acquérir de nouvelles connaissances sur lui-même mais aussi sur les apprenants ; c'est que l'on peut nommer « l'expérience ».

6. Vue d'ensemble

Le schéma ci-dessous représente une vue synthétique des modélisations que nous avons détaillées tout au long de ce paragraphe.

Figure 38 : Modélisation conceptuelle : vue d'ensemble

[33] A l'heure actuelle, dans notre prototype, ces pré-conditions s'expriment à l'aide de prédicats mais il conviendrait de définir un langage d'expression résolument orienté-pédagogie directement manipulable par tout pédagogue non expert en mathématiques.

Notons cependant que la spécification des connaissances pour la mise en œuvre d'enseignement, dénotée « spécification EIAO » sur le schéma, est un processus de spécification présenté ici de façon descendante mais qui doit pouvoir se concevoir non seulement de façon ascendante mais aussi de façon aléatoire, c'est-à-dire non pas selon le hasard mais selon les besoins et envies des pédagogues à un moment donné.

Nous avons fourni un ensemble de modèles représentant la structure conceptuelle de notre méthode de spécification des connaissances relatives à l'acquisition des connaissances fondée sur un enseignement centré sur la notion de situation-problème.

Il convient aussi de spécifier et planifier l'enseignement, ce qui implique la possibilité de pouvoir définir et gérer non seulement le niveau de contrôle attaché à chaque composant au sein de chaque étape mais aussi les transitions entre ces étapes.
En outre, une méthodologie ne se conçoit pas sans un ensemble d'outils permettant sa mise en œuvre. A partir de l'ensemble des définitions que nous avons présentées ci-dessus, nous allons mettre en évidence une architecture supportant les outils de base et les fonctionnalités associées de notre méthodologie de spécification de connaissances éducatives par et pour des enseignants.

Chapitre 9 : Principes d'intégration dans un TI d'aide à la spécification

A l'Exposition Universelle de 1933 à Chicago, l'engouement pour les merveilles de la technologie avait atteint un tel sommet que l'un des slogans était : « *La science trouve, l'industrie applique, l'homme s'adapte* ». En entrant dans le XXI° siècle, [Norman, 1996] propose de transformer ce slogan en « *L'homme propose, la science étudie, la technologie s'adapte* ».
En effet, une évidence s'impose : les hommes et les machines ne se comprennent pas. Le flot d'informations que distillent certaines machines excède bien souvent les capacités des hommes. Les informations qu'il reçoit demeurent dissociées, au point qu'il ne sait comment les réunir pour leur donner sens et agir. Et c'est aux techniques d'interaction homme-machine qu'il appartient de concilier les deux parties en mettant au jour les antagonismes et les moyens d'une vie commune.

Revenons succinctement sur les principales fonctionnalités de l'environnement de spécification des connaissances que nous préconisons. A cause de la diversité des acteurs et de leurs façons d'éliciter leurs connaissances, l'architecture proposée doit permettre aux pédagogues de s'exprimer dans des sémantiques équivalentes, et doit par conséquent être capable de s'accommoder de différents formalismes. Les différents outils devront alors autoriser une extraction facile du modèle profond de représentation des connaissances. Ceci pourra alors être utilisé pour communiquer avec les ingénieurs systèmes, les concepteurs et les autres acteurs du domaine à l'aide d'une terminologie de la discipline compréhensible.
Le méta-modèle que nous avons spécifié dans le chapitre précédent supporte ces contraintes. Cependant, nous souhaitons que tout pédagogue soit capable, par lui-même et sans l'intervention d'un informaticien, de spécifier ses connaissances. L'environnement informatique que nous proposons doit donc aller plus loin dans les fonctionnalités qu'il met à disposition afin d'être réellement accepté et donc utilisé par le milieu enseignant.

Paragraphe 1 : Architecture de notre environnement

Pour intégrer les propositions énoncées ci-dessus au cœur de notre environnement, trois services de base sont alors prépondérants : le guidage à l'usager, une grande souplesse d'utilisation et des facilités de réutilisation.

Nous allons ensuite présenter dans quelle mesure une approche de type Modèle - Point de Vue - Contrôleur [Krief, 1992] respecte ces contraintes.

1. Trois principes de base : guidage, flexibilité et réutilisation

Bien que nous l'ayons déjà souligné dans ce chapitre, il est important de rappeler que la présentation en trois sous-paragraphes distincts de ces trois principes de base demeure abstraite dans la mesure où ces trois fonctionnalités majeures sont intrinsèquement jointes.

1.1. Guidage

De façon générale, il s'agit de prévoir différentes façons de travailler, sans en privilégier une aux dépens des autres. Ainsi, lorsque le pédagogue est dans une certaine situation, il est donc nécessaire de connaître les différentes options qui lui sont offertes et de l'assister ensuite de manière adéquate. Le guidage doit alors se concevoir au moins à deux niveaux :

- un guidage local qui prend son sens dans le cadre d'une tâche particulière à accomplir,
- un guidage global qui consiste à s'assurer que les différentes tâches accomplies par l'usager conduisent vers le but à atteindre et si ce n'est pas le cas à assister l'usager dans le choix des tâches effectuées.

Face à cette dualité dans l'assistance, plusieurs approches ont été proposées pour représenter les façons d'opérer d'un usager face à un problème. [De Haan *et al.*, 1991] distingue :

- des modèles pour analyser la connaissance que l'usager doit avoir, afin de pouvoir opérer ; ce sont des approches basées sur des grammaires qui permettent de représenter les différentes tâches en termes de ce qu'elles peuvent signifier pour l'usager (« *Task Action Grammar* »),
- des modèles permettant d'analyser, de décrire, de prédire la performance de l'usager lorsqu'il est confronté à une tâche (GOMS). Ces modèles ont pour point fort de prendre en compte l'aspect cognitif de l'usager ce qui facilite ensuite l'assistance (cf paragraphe suivant)
- des modèles plus centrés sur l'interaction homme-machine qui cherchent à fournir une représentation complète du système tel qu'il doit être perçu par un usager totalement compétent (CLG et ETAG). Ces modèles permettent d'établir des liens de transposition (« *mappings* ») entre les différents niveaux d'analyse d'une tâche complexe.

Ces modèles sont différents mais permettent de représenter ce qu'un usager doit savoir et doit savoir-faire pour accomplir des tâches proposées par un ordinateur.
Les modèles obtenus vont permettre de :

- analyser les similarités et différences dans la façon dont les tâches sont réalisées,
- prédire certains aspects du comportement de l'usager par une formalisation des connaissances selon différents niveaux d'abstraction et, par voie de conséquence,
- faciliter une assistance du processus mené par l'usager.

Cependant, ces modèles sont, selon nous, insuffisants car ils sont orientés vers l'activité (« *activity-oriented* ») et ne sont pas suffisamment centrés sur le *contexte* dans lequel ces tâches sont réalisées.

Nous préconisons plutôt des modèles d'assistance dits orientés vers la décision (« *decision-oriented* ») comme ceux proposés en EIAO par [Mengelle, 1995], [Péninou, 1993], ou ceux décrits dans le domaine des systèmes d'information par [Rolland *et al.*, 1995].

Dans le cadre de la spécification des connaissances éducatives, ces modèles vont se manifester par un contexte c'est-à-dire un couple (situation, décision).

La *décision* représente un choix qu'un acteur de la spécification peut faire à un moment donné, lorsqu'il se trouve dans une *situation* donnée. Une décision est alors une intention, un but, qui peut se manifester sous trois formes [Plihon et Rolland, 1995] :

- une action consistant à transformer la spécification en cours (« *executive-based context* »),
- une action consistant à faire un choix parmi différentes alternatives de modification d'une spécification (« *choice-based context* »),
- une action complexe conduisant à activer un plan de spécification qui correspond en fait à un graphe d'activation de différentes actions de choix (forme d'action précédente).

Représenter la multiplicité des choix et actions qu'un pédagogue doit effectuer au cours d'une spécification, c'est-à-dire représenter le processus global, consiste alors à décrire différents graphes de contextes. Chaque graphe correspond alors à un ensemble d'activités hiérarchisées conduisant à un but [34].

De plus, avec une représentation des connaissances orientée vers la décision, l'assistance devient réellement active car l'initiative revient également à l'environnement de spécification et non plus seulement à l'usager. La forme prise par l'assistance peut bien sûr être « classique » et proposer [McClure, 1989] :

- un contrôle, statique ou dynamique, des commandes que l'usager veut activer. C'est ainsi que l'environnement méthodologique doit permettre un masquage des fonctions de l'interface en fonction du contexte.
- une visualisation de l'état courant des éléments spécifiés et/ou des actions déjà réalisées par un usager afin d'assurer un suivi au plus juste permettant par la suite de proposer les outils les plus pertinents au bon moment.

- un guidage de l'usager en listant les actions valides dans un contexte donné ou en fournissant une liste d'actions permettant d'atteindre un objectif donné,
- une prévision par simulation d'une action donnée.

Toutefois, l'assistance doit selon nous prendre d'autres formes supplémentaires par emprunt à l'EIAO en mettant à la disposition des enseignants des formes de systèmes tuteurs intelligents dédiés à la spécification. Ces systèmes tuteurs intelligents ont en effet, de par leur conception, vocation à partager l'initiative des interactions avec l'usager. Dans la mesure où ils utilisent l'activité de l'usager comme moteur de l'assistance, ceci permet d'offrir de nombreuses capacités d'assistance dont certaines nous semblent tout à fait appropriées à un environnement de spécification pour pédagogues :

- l'assistance par formation minimale,
- l'assistance par compagnonnage,
- l'assistance par introduction d'un perturbateur, ou encore
- l'assistance par préceptorat avisé.

Le principe fondamental de l'assistance par formation minimale [Bertrand, 1993] est qu'on apprend mieux une théorie, et dans notre contexte une méthodologie, par la pratique que par la théorie. C'est le principe du « *learning by doing* ». Ce type d'assistance vise à guider davantage l'usager dans son exploration du système pour que celui-ci ne soit pas débordé par la masse d'informations à manipuler. Les systèmes d'aide produits [Caroll *et al.*, 1988] permettent de simplifier dans un premier temps la complexité du système en limitant les fonctionnalités mises à disposition de l'usager. En avançant par étapes dans l'exploration du système et conforté dans des manipulations simples, l'usager aborde progressivement les éléments plus complexes du système. Ces systèmes d'aides misent donc sur l'exploration intuitive de l'usager contrairement à la pensée systémique qui tracerait d'avance le meilleur chemin possible.

[Beyou, 1992] définit l'assistance par compagnonnage comme « *l'apprentissage par l'action sous le guidage d'une ou plusieurs personnes jouant le rôle du maître* ». Ici l'environnement intègre un système de formation mettant trois bases de connaissances à disposition de l'usager :

- une base d'informations organisées de façon à favoriser le libre accès aux connaissances qui y sont contenues,
- une base de règles de dépannage (règles, plans, procédures) et
- une base de cas de pannes solutionnés.

Un pilote gère ces composants selon le contexte dans lequel se situe l'usager.

[Frasson *et al.*, 1996] définit l'assistance par introduction d'un perturbateur comme un compagnon particulier dont l'objectif est de renforcer la confiance de l'usager. Le perturbateur peut ainsi fournir

[34] Cette représentation par graphe est de plus en accord avec le choix de l'interface utilisateur diagrammatique que nous préconisons dans la section 2 de ce chapitre.

de bonnes ou de mauvaises solutions aux problèmes que les utilisateurs rencontrent durant leurs activités. Ceci amène donc l'usager à remettre en question ses croyances et connaissances et par conséquent de gagner en autonomie.

L'assistance par préceptorat avisé [Mengelle, 1995] associe prudence et opportunisme pour apporter une assistance à l'usager à deux niveaux :

- une assistance à la réalisation des tâches élémentaires,
- une assistance dans le choix des fonctionnalités d'outils pour un but donné de l'usager.

Une fonction de surveillance est ainsi associée à chaque outil qui est capable d'analyser le travail local de l'usager, de le conseiller pour l 'amener à corriger ses erreurs et, en cas de difficulté persistante, de faire appel à un pilote chargé de rechercher une fonctionnalité d'outil différente.

La particularité de ces fonctions de systèmes tuteurs intelligents est qu'elles sont basées sur le contexte d'évolution de l'utilisateur. Cependant, elles ne doivent pas dissuader les pédagogues de leur tâche initiale et l'interaction dans de tels systèmes de guidage doit notamment :

- laisser les utilisateurs maîtres de leur travail (par exemple dans la façon de résoudre un problème, de consulter l'aide, etc…),
- être prudente par rapport à la fréquence des interventions (de trop nombreuses interruptions peuvent annuler les bénéfices d'une découverte autonome des connaissances).

Par conséquent, tout environnement de spécification des connaissances pédagogiques se doit d'intégrer une assistance à l'usager couplée à une souplesse dans la présentation et la manipulation.

Par exemple, dans le prototype que nous avons implanté, nous avons choisi de permettre aux pédagogues de sauvegarder tout ou partie de leur travail en vue d'une réutilisation ultérieure (cf. paragraphe 1.3). L'assistance se manifeste ensuite selon deux formes distinctes :

- le pédagogue lance manuellement une recherche pour trouver des concordances entre la spécification en cours et des spécifications existantes sauvegardées, et-ou
- l'environnement, en fonction du profil du pédagogue et de l'état d'avancement de ses travaux, lance cette même recherche de façon automatique.

Des outils de visualisation des résultats permettent ensuite au pédagogue de choisir ou non leur intégration dans le travail en cours.

En outre, la définition d'un processus de spécification comportant des étapes distinctes pour lesquelles une hiérarchisation existe permet de définir des critères d'achèvement d'une étape et ainsi de pouvoir guider le pédagogue-utilisateur tout au long de son travail de spécification. Ceci peut se traduire par exemple par des propositions de réutilisation de spécifications existantes.

Par conséquent, l'assistance à l'usager ne peut se concevoir sans une souplesse de l'ensemble des éléments du système.

1.2. Flexibilité

Introduire de la flexibilité dans le processus de spécification des logiciels éducatifs englobe selon nous deux aspects :

- autoriser *différents points de vues* sur une spécification,
- offrir *différentes façons de spécifier* aux enseignants.

Dans les prochains paragraphes, nous justifions ces deux aspects et nous décrivons des pistes pour implémenter ces fonctionnalités dans un environnement de spécification des connaissances pédagogiques.

1.2.1. Différents points de vue sur une spécification éducative

Un système d'enseignement est, comme un système d'information, un environnement qui peut et doit s'analyser selon des perspectives très diverses. La spécification d'un module pédagogique peut ainsi être étudiée selon la perspective :

- de la justesse des concepts que l'apprenant va devoir acquérir au cours de ce module. Ces concepts sont-ils *auto-suffisants* (forment-ils un ensemble cohérent et fortement couplé ?), sont-ils décrits selon le *bon registre de conceptualisation* pour l'apprenant supposé utiliser le module (un même concept comme le concept de démocratie peut s'étudier de la maternelle à l'université), selon le *bon niveau de granularité* (un concept complexe doit s'étudier en détail mais le détail ne doit pas cacher les propriétés intrinsèques du concept),...
- de la justesse des stratégies mises en œuvre pour faciliter l'acquisition des concepts. Ces stratégies sont-elles *adaptées aux concepts* à acquérir, sont-elles *adaptées aux mécanismes d'acquisition des connaissances des apprenants* supposés, laissent-elles *la place à d'autres stratégies* (stratégies instructivistes/constructivistes) si les apprenants ont un comportement non classique,...
- de la justesse des scénarios proposés aux apprenants pour leur faire accepter l'apprentissage des concepts. Ces scénarios sont-ils *judicieux et motivants*, sont-ils *décrits de façon suffisamment précise*, et tient-on compte *des apprenants non réceptifs* à un scénario donné ?

Avec ces quelques exemples nous pouvons voir que, comme pour des systèmes d'information plus classiques, différents acteurs doivent coopérer et échanger leurs vues sur de mêmes spécifications mais qu'en plus pour un même objet et un même acteur, différentes perspectives doivent être mises en

corrélation. Pour obtenir l'agrément sur une spécification donnée, il faut que les différents acteurs aient les moyens d'interpréter une spécification pédagogique en s'appuyant sur le formalisme qui leur convient le mieux : tous doivent partager la même spécification « profonde » mais chacun a sa propre lecture de la spécification produite car chacun a les moyens (le formalisme) d'interpréter la spécification selon sa perspective propre [35].
En outre, dans notre environnement, l'aboutissement à une même spécification peut et doit se concevoir selon des moyens différents en fonction des besoins des pédagogues.

1.2.2. Différentes façons de produire des spécifications éducatives

Il semble qu'il n'existe pas de voie royale conduisant à de bonnes spécifications des connaissances en enseignement tout comme il n'en existe actuellement pas pour des logiciels plus classiques. Face à ce constat, nous pensons nécessaire de privilégier une voie constructiviste pour la spécification d'enseignements. En ce sens, il convient alors de produire des environnements qui n'imposent pas une façon de spécifier aux enseignants mais qui :

1) partent du principe qu'il peut exister plusieurs façons de procéder pour arriver à un résultat donné. Cet objectif accrédite encore plus les modèles de processus orientés-décision car ces modèles sont justement pensés pour que l'assistance puissent prendre en compte différentes situations, et dans une situation donnée pour pouvoir décrire différentes actions concurrentes. Ces modèles, tel le modèle MACT [Péninou, 1993] sont utilisés de manière classique pour l'assistance aux apprenants utilisant des systèmes tuteurs intelligents, c'est-à-dire pour pouvoir prendre en compte les comportements fort divers d'apprenants face à un logiciel éducatif.
2) sont suffisamment ouverts et tolérants pour que des spécifications non cohérentes du point de vue du système soient tout de même acceptées : l'expérience montre en effet que de telles spécifications sont souvent porteuses de sens, le chaos initial pouvant être le meilleur point de départ pour des spécifications riches en signification. Dans le cadre de nos travaux, nous n'avons aucune expérience concernant ce dernier point mais des expériences très intéressantes sont décrites dans [Nissen *et al.*, 1996].

Par conséquent, la prise en compte des principes de guidage et de flexibilité permettent de fournir aux pédagogues-utilisateurs un environnement puissant et adapté leur permettant d'aboutir à la spécification de leurs connaissances éducatives. En outre, nous montrons dans le paragraphe suivant que cela facilite également la sauvegarde et la réutilisation de ces connaissances.

[35] Nous retrouvons ici les principes évoqués dans les premiers chapitres de cette thèse.

1.3. Réutilisation

La façon la plus commune utilisée par les experts pour résoudre un problème est de le comparer à des cas déjà traités. Alors que nous pouvons ne pas disposer dans le curriculum de suffisamment de temps pour une variété de problèmes, [Guzdial *et al.*, 1996] pense que l'on peut promouvoir de la flexibilité en demandant aux apprenants de résoudre les types de problèmes qui nécessitent de leur part de considérer la situation selon une variété de perspectives et d'examiner nombre de solutions alternatives, en leur fournissant les expériences, c'est-à-dire les cas d'autres apprenants sachant qu'ils doivent être motivés pour examiner, comprendre et analyser les décisions des autres.
Ainsi, les bibliothèques de cas permettent aux utilisateurs de trouver les solutions pertinentes et de voir comment les connaissances peuvent être généralisées et transférées. Il est alors nécessaire de disposer d'outils permettant d'une part la sauvegarde de spécifications existantes mais aussi disposant de fonctions dont le rôle est de proposer une récupération des cas existants. Ceci impose implicitement plusieurs fonctionnalités qui sont de l'ordre du suivi de la modélisation en cours, de l'évaluation de celle-ci puis de la prise d'initiative par la machine afin d'assister l'utilisateur.
Nous proposons d'adapter ces principes aux enseignants en phase de spécification des connaissances.

Le minimum que l'on puisse faire est donc de mettre en place dans notre environnement des fonctions de sauvegarde telles qu'elles sont présentes dans les logiciels « usuels » de programmation ou même de bureautique. Nous allons présenter ci-dessous les ajouts que nous y apportons.
Tout d'abord, notre environnement permet d'effectuer une sauvegarde pour chaque étape clé de la spécification des connaissances. Ainsi, nous proposons de différencier chaque modélisation en associant à chaque représentation une extension spécifique en fonction de sa signification :

«.**EIAO** » représente la spécification dans sa globalité,
«.**md** » pour la matrice disciplinaire,
«.**oa** » pour les objectifs d'apprentissage,
«.**se** » pour les stratégies d'enseignements, et
«.**spb** » pour les situations-problèmes.

Figure 39 : Base de données pour la sauvegarde et la réutilisation

En outre, il convient de disposer d'une plus grande souplesse dans la sauvegarde afin d'être en mesure de réutiliser les représentations déjà effectuées. Par conséquent, notre environnement doit permettre à tout pédagogue-utilisateur de mémoriser seulement certaines parties de ses spécifications en fonction de ses besoins et pas seulement dans sa globalité.

Cette granularité a été obtenue en associant aux différents outils de représentation des connaissances, des fonctionnalités de couplage avec une base de données. C'est ainsi que nous utilisons une base de données relationnelles [36] avec les possibilités d'effectuer tout un ensemble de requêtes plus ou moins sophistiquées selon les desiderata des pédagogues.

Figure 40 : Base de données pour la sauvegarde et la réutilisation de la matrice disciplinaire

La souplesse de ces différentes fonctions, c'est-à-dire la flexibilité de notre environnement, repose sur la prise en compte du profil de l'utilisateur. En effet, en fonction de celui-ci, la puissance de réutilisation est variable. Par exemple [37], un utilisateur novice pourrait disposer de fonctions lui permettant de récupérer uniquement ses propres spécifications. Un utilisateur plus expérimenté pourrait avoir accès non seulement à ses modélisations mais également à celles des utilisateurs novices. Enfin, un utilisateur confirmé pourrait quant à lui avoir accès à toutes les spécifications existantes.

Figure 41 : Base de données pour la gestion des utilisateurs

Dans ce paragraphe, nous avons essayé de décrire, dans le cadre d'un environnement de spécification pédagogique, les fonctionnalités nécessaires pour rendre des pédagogues confiants dans l'outil informatique, et par conséquent, aussi efficaces que possible. Nous avons focalisé notre attention sur l'assistance au processus de spécification et sur la flexibilité que doit avoir un environnement de spécification. Cette dernière caractéristique est essentielle car elle conditionne les contraintes d'usage imposées à l'utilisateur et donc, dans bien des cas, son acceptation ou son refus des outils proposés.
Nous allons maintenant définir dans quelles mesures une structuration profonde des connaissances à l'aide d'une architecture du type Modèle - Point de Vue - Contrôleur répond aux exigences que nous nous sommes fixées.

2. Du schéma M.V.C. à l'approche M.PV.C.

L'approche M.V.C., acronyme de Modèle-Vue-Contrôleur [Krasner et Pope, 1988], permet d'appréhender la construction d'un programme selon trois parties distinctes :

[36] Notre choix s'est porté sur Microsoft Access® dont le couplage avec notre environnement de programmation se fait par des liens ODBC (acronyme de « *Open DataBase Connectivity* »).
[37] C'est ce que nous avons choisi d'implanter dans notre prototype.

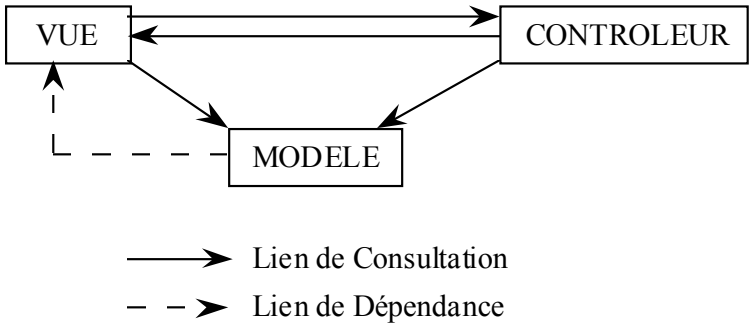

Figure 42 : Schéma MVC

Le *modèle* représente la structure et les comportements sur lesquels on travaille et que l'on veut représenter, indépendamment de toute considération relative à l'interface.
La *vue* représente l'interface de sortie qui sert à représenter extérieurement cette structure et à animer ses comportements. Les *vues* représentent donc les différents moyens d'accès au même *modèle* (graphiques, textuels, par menus, etc…).
Le *contrôleur* représente l'interface d'entrée qui permet d'interagir avec le *modèle* ou sa représentation, c'est-à-dire la *vue*. C'est un objet qui gère les *événements* provenant, par exemple, du clavier ou de la souris. A chaque *vue* est associé un *contrôleur* spécifique, ce qui permet de gérer les événements de façon *contextuelle*, en fonction de la *vue* associée. A chaque instant, seul un contrôleur est actif : celui dont la vue est active.

Cette démarche a l'avantage de dissocier les aspects relatifs à la représentation interne (structurels et fonctionnels) d'un programme, de ses aspects en rapport avec la représentation externe (figuratif et ergonomique).
Cependant, dans le cadre de la spécification des connaissances éducatives, nous avons précédemment démontré qu'il est intéressant de disposer, non pas de un seul, mais de plusieurs points de vues sur le produit à construire.
Tel est l'objectif de l'approche Modèle - Point de Vue - Contrôleur [Krief, 1992] qui se base à l'origine sur le schéma M.V.C.

L'approche M.PV.C. est une méthode de conception par objet facilitant la mise en œuvre de multiples outils d'interprétation, de manipulation et de simulation de structures. En effet, elle permet de dissocier les aspects mécaniques d'une application, de ses aspects évaluatoires ou sémantiques ; elle favorise donc la conception de multiples mécanismes de parcours, de multiples interprétations et de multiples représentations d'une même structure.

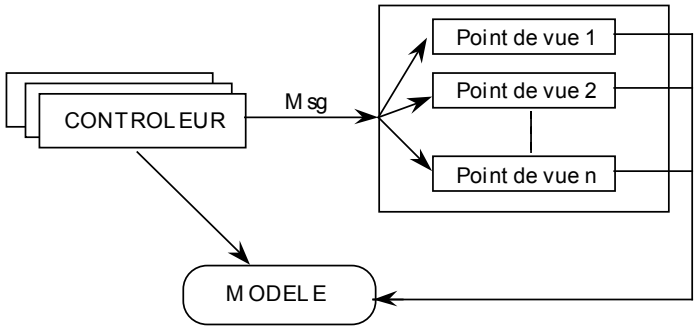

Figure 43 : Trilogie M.PV.C. [Krief, 1992]

Avec l'approche M.PV.C. les trois parties distinctes d'un programme sont :

- une partie *modèle* qui décrit l'organisation des données manipulées par le programme,
- une partie *contrôleur* qui décrit le mécanisme de parcours de ces données,
- une partie *point-de-vue* décrivant l'interprétation à donner à ce parcours.

La mise en œuvre de cette approche offre au concepteur plusieurs possibilités pour :

- concevoir graphiquement de multiples parcours d'un même modèle ;
- concevoir de multiples points-de-vue ;
- expérimenter des systèmes M.PV.C., c'est-à-dire la combinaison de chacun de ces parcours associés à un ou plusieurs points-de-vue ;
- construire de multiples outils de mise au point personnalisés ;
- affiner les outils de l'environnement de base ou enrichir ce dernier de nouveaux outils.

Nous proposons que chaque élément intégré dans notre environnement soit décrit suivant cette trilogie Modèle - Point de Vue - Contrôleur qui présente de nombreux intérêts dans le cadre de la spécification des connaissances. Elle permet notamment de gérer différents points de vue sur un même scénario pédagogique, favorisant ainsi la prise en compte des divers acteurs d'un EIAO : apprenant, pédagogue ou instructeur. Selon le type d'utilisateur utilisant l'environnement, des informations spécifiques peuvent alors être produites. Pour une même spécification, par exemple, l'interface de manipulation des connaissances est différente selon le profil d'utilisation établi.

Toutefois, comme nous le verrons dans les conclusions et perspectives de cette thèse, nous pensons qu'il serait plus adéquat d'approfondir davantage cette architecture par l'intermédiaire du paradigme « agents ». En effet, il conviendrait de disposer d'un environnement dont les différentes entités autonomes interagissent pour la résolution de problèmes, ce qui correspond parfaitement avec la définition d'un système multi-agents telle qu'elle est proposée dans [Labidi et Lejouad, 1993].

Paragraphe 2 : Quelle interface utilisateur ?

Le rôle des ingénieurs, programmeurs, concepteurs et chercheurs doit donc être d'apprendre à concevoir pour les gens, en prenant en compte les connaissances, les attentes et les capacités d'un non-expert. Cela s'appelle la *conception centrée sur l'utilisateur* et lorsque cette philosophie est suivie, [Norman, 1996] parle d'*ergonomie cognitive*. [Ganascia, 1996], quant à lui, évoque l'*ingénierie d'interface*, nourrie de sciences cognitives.
L'ergonomie est l'étude de l'interaction homme-machine qui concerne en premier lieu les aspects physiques de la conception (taille des composants, leur disposition, …). L'ergonomie cognitive souligne le côté mental de la conception, la facilité de compréhension et d'utilisation de l'équipement. C'est le manque d'ergonomie cognitive dans la conception qui a conduit à cette situation moderne de complexité, de confusion et d'erreur.

En outre, il est important de souligner que la communication ne s'opère pas uniquement par la parole ou l'écriture, mais aussi par les notations et les dessins qui peuvent nous permettre de mieux nous faire comprendre. C'est ce que nous retrouvons dans l'expression commune « un graphique vaut parfois mieux qu'un long discours ».
[Faure, 1996] pense que des croquis sont nécessaires à la communication car ils permettent de préciser la pensée de leur auteur. En effet, la possibilité de retravailler ces schémas à mesure que la pensée s'affine permet ensuite d'aboutir à un brouillon suffisamment élaboré pour pouvoir être traité et remis au propre automatiquement.
[Larkin et Simon, 1987] montrent comment un graphique peut donner lieu à des calculs plus rapides qu'un ensemble de phrases. En effet, dans une description de type linguistique, tous ces éléments doivent être laborieusement déduits et composés par des règles mentales explicites. En revanche, un graphique, par l'intermédiaire de l'œil, fournit ces éléments le plus simplement du monde car les représentations graphiques bénéficient grandement de l'appareil sensoriel de la reconnaissance. On passe d'un transfert des connaissances du domaine cognitif vers le domaine perceptuel. Cependant, on n'utilise pas non plus n'importe quel schéma ou graphique n'importe comment. De plus, les représentations graphiques ont ceci de particulier que leur utilisation et leur compréhension passent par un apprentissage, tout comme le langage ou les mathématiques. [Larkin et Simon, 1987] définit alors un « bon » graphique par le critère suivant : les objets perceptuels sont alignés sur les objets conceptuels.
De même, nous voulons tous davantage d'informations pour pouvoir enfin nous concentrer sur l'information qui seule compte vraiment. [Rao, 1996] montre que les représentations visuelles s'y prêtent particulièrement bien.

Un *diagramme* est également un outil fondamental dans les activités d'analyse et de conception. En effet, c'est un moyen intuitif permettant d'exprimer des relations entre concepts et la plupart des gens peuvent y avoir recours contrairement aux langages et notations formelles. Bien que de tels diagrammes sont aisément « dessinables » à l'aide d'un crayon et d'une feuille de papier, toute modification conséquente nécessite de tout reprendre ; d'où des problèmes de vérification de consistance, etc… Une *aide* peut alors être fournie par des outils de dessin assisté par ordinateur conventionnels, mais ceux-ci souffrent de deux restrictions principales. Premièrement, les outils ne sont pas exactement spécifiques pour le type de diagramme souhaité ; par exemple, lorsqu'une image est effacée ou déplacée, nous ne connaissons pas les *liens* existants avec cette image depuis d'autres images, ce qui permet de mettre à jour les liens. Deuxièmement, dans la plupart des cas, le diagramme ne peut être directement *évalué* car il doit d'abord être retranscrit manuellement dans une représentation adéquate.

Il est, par conséquent, important de disposer d'outils CASE qui surmontent ces défauts et dont les moyens de personnalisation sont étendus. C'est dans cette optique que nous personnalisons l'interface de l'environnement de spécification en fonction de l'étape dans le processus d'élicitation des connaissances dans laquelle se trouve le pédagogue.

En outre, bien que la plupart des méthodes de conception utilisées aujourd'hui dans la pratique utilisent une représentation graphique de l'information (à l'aide de diagrammes), il est important de distinguer l'information *conceptuelle*, de l'information *graphique*. L'information graphique contenue dans les diagrammes, telle que la forme de divers éléments, la longueur de lignes et l'agencement général peut être un facteur majeur pour la lisibilité du diagramme mais n'a aucun rapport avec l'information propre qui doit être acheminée par le diagramme. Il est donc nécessaire d'utiliser un système permettant de mémoriser ces deux niveaux d'information. C'est dans ce sens que les utilisateurs continueront probablement à préférer les techniques graphiques, tandis que les informations pour contrôler la « correction » de la spécification du logiciel sont mémorisées dans la couche conceptuelle. Le niveau conceptuel est alors utilisé pour contrôler la « correction » de la spécification du logiciel. Ceci est nécessaire car, en général, les diagrammes conçus par l'utilisateur n'ont pas tous une signification correcte.

De plus, une interface diagrammatique peut également se structurer autour d'une approche Modèle - Point de Vue - Contrôleur :

- Le *Modèle* est responsable du maintien de l'état et de la mise en évidence du comportement nécessaire pour supporter l'interface utilisateur.
- Les *Points de Vue* sont en charge de l'affichage des versions mises à jour du *Modèle*.
- Le *Contrôleur* a pour rôle de transposer les actions de l'utilisateur vis à vis des changements d'état dans le *Modèle*.

En appliquant cette approche, nous pouvons établir un parallèle avec l'outil méta-CASE « Hotdraw » cité en annexe 2. Avec ce dernier, trois agents dérivés assurent une approche MVC : « *Drawing* », assure le dessin, « *DrawingView* », permet de le visualiser, et « *DrawingController* » analyse les gestes de l'utilisateur et les traduit en changements au niveau du dessin.
Une approche « agents » permet également une amélioration à ce niveau. Pour rendre des objets aussi réutilisables que possible, nous ne voulons pas coder de façon physique les contraintes entre eux. En effet, les objets ne devraient être responsables que pour leur état propre, et pas de l'état d'autres objets. Par contre, s'il y a une contrainte entre les états de deux objets, alors cette contrainte doit être enregistrée quelque part. Ainsi, des changements pour le premier objet doivent se traduire en changements pour le second. Une solution consiste alors à avoir un agent en charge de maintenir une liste de contraintes entre objets dépendants. Chaque fois qu'il y a des changements pour un objet, c'est notifié à tous ses dépendants et il doit alors effectuer l'action appropriée. En général, un objet peut avoir plusieurs dépendants, et le changement de l'objet requerra seulement de changer certains de ses dépendants. Mais chaque dépendant sera capable de déterminer si le changement de l'objet était significatif et pourra ignorer ceux qui ne le sont pas.

En conclusion, notre environnement de spécification pédagogique dispose des caractéristiques suivantes :

- il assiste les pédagogues dans leurs choix lorsqu'ils sont en train de spécifier leurs connaissances ; cette aide prenant plusieurs formes :
  - ♦ un support méthodologique,
  - ♦ un système d'aide à la sélection de composants logiciels en fonction de critères pédagogiques.
- il est capable, avec des fonctions de prototypage, d'exécuter les spécifications décrites par les pédagogues afin de faciliter leur évaluation.
- il doit également disposer d'une interface conviviale assurant les deux points précédents.

Ceci peut se résumer par la figure suivante :

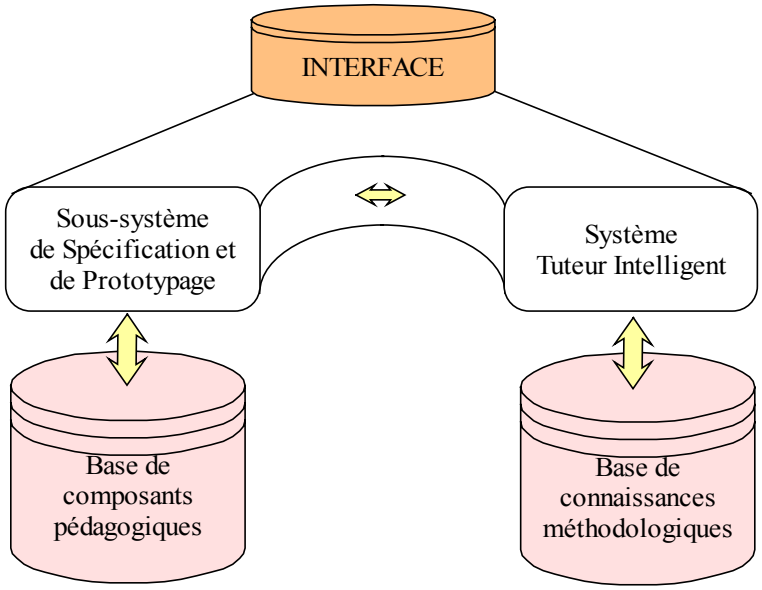

Figure 44 : Principales composantes de l'environnement de spécification.

Le sous-système de spécification et de prototypage met à disposition des usagers une base de composants correspondant aux différentes connaissances à mettre en place tandis que le sous-système exploitant les connaissances méthodologiques supporte le guidage et la flexibilité.

La représentation des connaissances sous forme ontologique nous a ainsi permis de mettre en place des fonctions de haut niveau qui permettent la réutilisation des connaissances. La flexibilité est assurée grâce à la structure Modèle - Point de Vue - Contrôleur et les fonctionnalités de tuteur intelligent permettent de guider le pédagogue pour l'extraction et la manipulation des connaissances qu'il souhaite spécifier.

Le chapitre suivant est consacré à la réalisation pratique que nous avons développée. Nous y présentons les différents outils logiciels que nous avons étudié, ainsi que les choix et limites que nous avons imposés.

Chapitre 10 : La réalisation de OMAGE

Nous avons mis en évidence la représentation structurelle des différents composants d'un environnement de spécification des connaissances au sein d'un atelier de génie éducatif. Il convient également de définir quels sont les outils qui permettent de respecter au mieux l'ensemble de ces contraintes.
Tel est l'objectif du paragraphe 1 tandis que le paragraphe 2 concerne la mise en place du prototype que nous avons implanté afin de démontrer la pertinence de nos propos.

Paragraphe 1. Quels outils ?

Tout d'abord, il est important de signaler que pour obtenir les meilleurs résultats possibles pour une application donnée, il convient de choisir les outils adéquats plutôt que de se forcer à utiliser un outil inadapté.
Ainsi, nous pensons que des outils assurant le couplage des paradigmes « objet » et « logique » permettent de respecter les contraintes de l'environnement de spécification que nous avons définies dans les chapitres précédents [38].

1. Les paradigmes objet et logique

Selon [Arapis, 1990], le but principal de la conception d'un modèle de données est d'avoir un outil avec lequel on peut décrire une partie du monde de façon adéquate ; ce qui signifie que le modèle fournit au concepteur un ensemble nécessaire et suffisant d'abstractions permettant la représentation aisée et fidèle des propriétés appropriées au centre d'intérêt alors que les propriétés inadéquates sont rejetées. Cet ensemble d'abstractions doit permettre la description à la fois statique et dynamique des aspects du monde. En effet, la dynamique dans une application est nécessaire pour faire évoluer le système avec le temps et pour la réutilisation et les combinaisons d'entités existantes.
Cependant, pour la modélisation des aspects dynamiques d'une application, les modèles relationnels classiques fournissent un support très faible voire inexistant.
Par conséquent, certains pensent la spécification des aspects dynamiques comme une activité séparée et proposent un modèle indépendant du modèle de données comme par exemple les Réseaux de Pétri [Peterson, 1981].

Selon [Uustalu, 1994], une modélisation unifiant les paradigmes « objet » et « logique » doit permettre la représentation des points de vue statiques et dynamiques d'une application.
En effet, l'orienté objet permet la modularité, l'encapsulation et le partage alors que la programmation logique autorise un « raisonnement » et la déduction. L'orienté objet est une technique appropriée au traitement de façon ascendante de problèmes de complexité raisonnable. Cela n'est pas basé sur un formalisme particulier mais plutôt sur une heuristique de travail adaptée à divers formalismes.
Le paradigme logique, au contraire, tient davantage compte d'une approche descendante. Malheureusement, ce paradigme manque d'heuristiques puissantes permettant d'appliquer ce formalisme.

Ainsi, depuis le début des années 80, différents auteurs ont proposé de nombreux systèmes fusionnant la logique et l'orienté objet. Plusieurs questions importantes émergent alors : « Faut-il se baser sur le paradigme objet et y adjoindre les caractéristiques inhérentes au paradigme logique (ou réciproquement) ? » « Ou bien faut-il arriver à un consensus ? »
Ainsi, les trois moyens permettant la fusion des deux paradigmes sont les suivants [39] :

- deux langages objet et logique sont interfacés,
- un langage objet est étendu avec des constructeurs logiques,
- un langage logique est étendu avec des constructeurs objets.

Dans le premier cas, nous pouvons citer Prolog/Loops [Koschmann et Evens, 1988] qui fournit des outils permettant d'appeler des objets Loops depuis Prolog mais aussi de définir des buts Prolog depuis Loops sous forme de messages. Il y a également PCE [Anjewierden, 1986] qui est un système de programmation objet dédié aux environnements graphiques de fenêtrage et qui est lié à Prolog via un petit ensemble prédéfini de commandes. ProWindows [Quintus, 1988] est une version commerciale de PCE et supporte l'abstraction, l'héritage, le polymorphisme et le transfert de messages.
Dans le second cas, nous trouvons Orient84/K [Ishikawa et Tokoro, 1986], KSL/Logic [Ibrahim et Cummins, 1990] et CBL [Biema *et al.*, 1990]. Les constructeurs logiques, sous forme de règles par exemple, sont considérés comme des objets. De même, ALF [Mellender, 1988] ou Prolog/V [Digitalk, 1988] incorporent des aspects de Prolog en les programmant en Smalltalk.
Dans le troisième cas, qui est le plus répandu, nous pouvons citer les travaux de [Shapiro, 1983] et de [Zaniolo, 1984] qui ont influencé les travaux actuels en montrant comment exprimer avec les outils logiques existants la notion d'objet.

[38] Nous verrons toutefois dans les conclusions et perspectives de cette thèse qu'une approche basée sur la notion d'agent serait plus appropriée.
[39] L'annexe 3 présente de façon plus détaillée ces différentes approches.

Les avantages de l'extension de la programmation logique avec des concepts de programmation objet sont nombreux :

- Le paradigme *objet* est un modèle à la fois simple et puissant pour l'écriture de programmes dont les composants sont des objets communiquant par envoi de messages. De plus, les programmes qui en résultent deviennent plus courts, plus compréhensibles et possèdent moins d'erreurs.
- L'*encapsulation* permet de considérer les différents objets comme des boîtes noires accessibles via leur interface qui est caractérisée par l'ensemble des messages recevables et dont le degré de flexibilité commande le *polymorphisme*. De plus, la prise en compte de cette notion encourage le *prototypage* rapide.

Cependant, un mécanisme supplémentaire à la programmation logique doit spécifier comment la *communication* entre modules est réalisée.

- La notion de *classe* permet une programmation structurée. En effet, dans la programmation objet, une classe est une collection d'opérations et d'états, ce qui permet au programmeur de traiter les changements d'états avec le formalisme objet. Il est possible ici de définir clairement la communication entre modules ainsi que leur réutilisation.
- La programmation logique est particulièrement adaptée à la définition de relations entre entités alors que la programmation objet est pauvre pour spécifier les relations entre classes (mis à part par héritage).
- *L'héritage* améliore la *réutilisation*.
- Il est possible et simple de spécifier, avec la programmation logique concurrente, les notions de *concurrence*, de *synchronisation* et de *changement d'états*.
- Le paradigme objet peut être facilement traité de façon *graphique* d'où une plus grande compréhension par visualisation pour des non-spécialistes.

TROLL [Jungclaus *et a l.*, 1996] est un langage de spécification pour la modélisation conceptuelle de systèmes d'information. Il vise à intégrer un style déclaratif, basé sur la logique pour la spécification de systèmes et la représentation des connaissances, avec des mécanismes de structuration connus des modèles de données sémantiques et orientés-objets [Peckham et Maryanski, 1988], et avec des approches de modélisation de processus et de systèmes concurrents [Hoare, 1985].

Parmi les exemples relatifs à cette approche, nous pouvons aussi citer : Prolog++ [Moss, 1990], ObjVProlog [Malenfant *et al.*, 1989], ESP [Chikayama, 1984] et CLIPS [Giarratano et Riley, 1991].

En se basant sur la figure 44 qui représente un aperçu du fonctionnement général de l'environnement de spécification, nous préconisons que chaque composant du modèle profond de représentation des connaissances soit modélisé avec le paradigme objet tandis que le paradigme logique (prédicats

exprimés avec la logique du premier ordre par exemple) permet de spécifier les contraintes de manipulation.

Il convient maintenant de présenter les principaux outils logiciels existants afin de déterminer quel est celui qui prend le mieux en compte les multiples contraintes que nous nous sommes fixées.

2. Outils logiciels

LISP, PROLOG sont les principaux langages utilisés en intelligence artificielle. CLIPS est aussi un langage largement utilisé. La présentation qui suit donne un aperçu des capacités de ces langages [40].

2.1. LISP

LISP, acronyme de LISt Processing language, est le principal outil de programmation en intelligence artificielle. C'est la « langue natale » de la discipline car, d'après [Farreny et Ghallab, 1987], 95% des programmes en intelligence artificielle seraient écrits en LISP contre 1% en Prolog et 4% dans divers langages.

Conçu par J. McCARTHY au M.I.T. en 1958, LISP est le plus vieux langage de programmation en usage aujourd'hui. Il fut pendant longtemps confiné à la recherche en intelligence artificielle. Le champ d'application privilégié de LISP reste la programmation symbolique, mais son utilisation hors de ce champ est de plus en plus large.

LISP est disponible aujourd'hui sur presque toutes les machines universelles, depuis les micro-ordinateurs personnels les moins onéreux jusqu'aux super-calculateurs. Par ailleurs, il existe plusieurs machines, dites *machines LISP*, à architecture adaptée, ou entièrement spécifiques au langage (Lambda Machine, MAIA, Texas Explorer, Symbolics, Xerox).

Les principales qualités du langage LISP sont :

- la richesse en moyens de représentation et de manipulation symbolique ;
- la simplicité de la syntaxe repose sur la seule structure de liste sans instruction ni mot-clé, mais uniquement avec un ensemble de fonctions ;
- la puissance du langage, et en particulier l'équivalence complète entre données et programmes ;
- la souplesse : gestion dynamique des objets et des ressources ;
- l'interactivité : LISP est principalement interprété ;
- l'ergonomie : LISP est précurseur en matière d'environnement de programmation intégré, exceptionnellement fourni en multiples outils d'aide à la mise au point ;

[40] ART-IM, OPS 5, KES ou Level 5 sont d'autres outils logiciels que nous avons considérés et que présentons brièvement en annexe 4.

- l'extensibilité particulièrement aisée de LISP.

On peut néanmoins trouver à LISP des défauts :

- le programme LISP tend à être verbeux et à manquer de concision, notamment par rapport au formalisme mathématique ;
- la lisibilité du code est réduite s'il n'est pas correctement paragraphé ;
- la performance des interpréteurs LISP fut pendant longtemps médiocre mais ce n'est plus le cas aujourd'hui. LISP exige beaucoup de mémoire mais ce n'est plus un facteur limitatif aujourd'hui ;
- l'absence de standardisation et les différences très significatives qu'il peut y avoir entre les diverses implémentations de LISP. La portabilité réduite d'un programme LISP est un handicap sérieux à la diffusion du langage. Mentionnons cependant une proposition de standard : CommonLisp vers laquelle tend à converger une famille de dialectes LISP.

2.2. PROLOG

PROLOG, acronyme de PROgrammation en LOGique, a été initialement conçu par Alain COLMERAUER et son équipe de l'Université de Marseille au début des années 70. Jusqu'au début des années 80, PROLOG était essentiellement utilisé et développé dans des centres de recherche européens (notamment Edimbourg, Budapest, Lisbonne et Toulouse). En déclarant PROLOG comme importante source d'inspiration pour la définition de nouveaux langages de programmation, le projet japonais de *cinquième génération* d'ordinateurs a fortement contribué à l'élargissement international de son audience.

Du point de vue de la programmation, une part de l'attrait de PROLOG vient de son caractère déclaratif :

- en constituant un programme PROLOG, l'utilisateur définit une *base de connaissances* dans le style de *rédaction associative* commun dans la méthodologie des systèmes-experts ;
- l'interpréteur des instructions PROLOG fonctionne comme un type particulier de moteur d'inférence.

Le langage d'expression des connaissances de PROLOG est fondé clairement sur le langage des prédicats du premier ordre. En tant que moteur d'inférence, PROLOG intègre deux mécanismes puissants : d'une part, celui d'*unification* pour réaliser le *filtrage* des connaissances, et, d'autre part celui de *retour-arrière* dans l'enchaînement des cycles d'inférence.

Des interpréteurs PROLOG, avec environnements plus ou moins riches, sont disponibles sur de nombreux modèles d'ordinateurs, depuis les plus gros systèmes jusqu'aux micro-ordinateurs personnels.

2.3. CLIPS

CLIPS [Giarratano et Riley, 1991], issu du Johnson Space Center de la NASA, est l'acronyme de « *C Language Integrated Production System* ». Ses origines datent de 1984.
CLIPS est un outil de production de systèmes experts qui fournit un environnement complet pour la construction de systèmes experts basés sur la notion de règles et-ou d'objets.
Les principales caractéristiques de CLIPS sont :

- *La représentation des connaissances*.

CLIPS est un outil permettant de manipuler un large éventail de connaissances reposant sur trois paradigmes de programmation différents basés sur les règles, l'orienté-objet et le procédural.

- *La portabilité.*

CLIPS est écrit en C pour des raisons de portabilité (PC, Macintosh, VAX, Sun) et de vitesse d'exécution.

- *L'intégration et l'extensibilité*.

CLIPS peut être inclus dans du code procédural, appelé comme sous-routine, et intégré avec des langages comme C, FORTRAN ou Ada.

- *La vérification et la validation*.

CLIPS dispose d'un ensemble de fonctionnalités permettant la vérification et la validation des systèmes experts, notamment pour la conception modulaire et le partitionnement d'une base de connaissances, pour le contrôle et la vérification des contraintes statiques et dynamiques portant sur les arguments de fonctions ou sur les attributs de classes, et pour une analyse sémantique des règles permettant de déterminer si des inconsistances peuvent empêcher une règle de se déclencher ou de générer une erreur.

Lorsque CLIPS est utilisé comme un langage à base de règles de production, la *représentation des connaissances* en CLIPS est assurée avec des règles selon une syntaxe proche de LISP. L'inférence est basée sur une *stratégie de contrôle par chaînage avant* selon l'algorithme de RETE [Forgy, 1982] qui traite uniquement les faits ayant changé, avec les règles qui sont potentiellement affectées, plutôt que de répéter le filtrage de toutes les règles avec tous les faits.

COOL, acronyme de « *Clips Object Oriented Language* » est la facette orientée-objet de CLIPS avec les caractéristiques principales des langages orientés-objets courants comme la modularité, la réutilisabilité, l'abstraction, l'encapsulation, l'héritage, le polymorphisme, la notion de classe ou de message.

Ses principaux défauts sont le manque de support pour un *chaînage arrière* efficace et le fait que CLIPS ne dispose pas de *boîte à outils graphique*. Il existe cependant des versions spéciales de CLIPS qui sont citées en annexe 4 et notamment « wxCLIPS » sur lequel nous reviendrons dans le chapitre suivant et en annexe 5.
En outre, une des principales raisons de l'utilisation de générateurs de systèmes experts plutôt que celle d'un langage procédural est la simplification de la spécification du *contrôle*.

Nous pouvons également citer ici le langage fonctionnel interprété « Scheme » [Arditi et Ducasse, 1996]. C'est un langage résultant d'un effort de simplification de LISP disponible sur la plupart des machines et systèmes d'exploitation. Son objectif est de faciliter la définition de fonctions dans un objectif de réalisation de spécifications formelles. La facilité de manipulation de ce langage repose sur un ensemble très limité de primitives de base [41] dont la combinaison permet de réaliser des fonctions très élaborées. En outre, Scheme dispose d'une plate-forme objet reposant sur le modèle STROBE [Cerri, 1996] spécialement développée pour la spécification d'agents.

Considérons maintenant quels sont les langages que nous avons utilisés pour implanter un prototype de notre environnement de spécification des connaissances répondant le mieux possible aux différents critères que nous avons développés tout au long de cette thèse.

Paragraphe 2. Notre prototype

1. Logiciels choisis

1.1. HARDY

Nous avons choisi de proposer aux pédagogues d'utiliser des outils informatiques leur permettant de représenter les connaissances à l'aide de diagrammes formés par un ensemble de noeuds et d'arcs.
« HARDY [42]» [Smart et Rae, 1995], développé par un laboratoire d'intelligence artificielle à l'Université d'Edimbourg (Laboratoire AIAI), répond à de telles exigences.
En effet, alors qu'avec un outil CASE traditionnel, la méthode de développement est fixe, HARDY est un outil méta-CASE ouvert à un nombre illimité de méthodes. La représentation des connaissances dans HARDY est donc suffisamment universelle pour convenir avec la plupart, si ce n'est toutes, des

[41] La spécification du langage contient seulement en quelques pages...
[42] HARDY est un outil méta-CASE, ou méta-outil, tel que nous l'avons précisé dans la seconde partie (chapitre VII, section 2).

méthodes de développement et cette caractéristique peut être atteinte en stockant non seulement la spécification du logiciel, mais aussi la méthode qui a été utilisée pour la construire.

HARDY permet à l'utilisateur de construire un *diagramme type* ou bien d'utiliser un type *prédéfini*. L'utilisateur peut ensuite sélectionner un type et produire *rapidement* des diagrammes sous forme essentiellement de *noeuds* reliés par des *arcs*. Lors de la construction d'un diagramme, les arcs restent solidaires des noeuds lorsque ceux-ci sont déplacés, et les valeurs peuvent être données aux *attributs* des arcs et des noeuds spécifiés dans le diagramme type. De même, il est possible et aisé de spécifier des contraintes de liaison entre les différents nœuds selon les différents arcs.
Une fois créés, les diagrammes peuvent être transformés en une variété de formats qui permettent au modèle du système sous-jacent d'être traité par un autre programme. Ainsi, HARDY peut être utilisé, par exemple, comme un outil de représentation des connaissances directement manipulable par un système à base de connaissances. De même, il peut être transformé en un document mélangeant du texte et des graphiques afin, par exemple, de rendre beaucoup moins ardue la documentation de procédures organisationnelles. HARDY peut aussi être utilisé pour visualiser des diagrammes générés par d'autres programmes.
Pour atteindre un haut niveau de personnalisation, HARDY possède une *interface de programmation d'applications* (communément nommée API, acronyme de « *Application Programmer's Interface* ») qui permet d'intercepter des *événements* intervenant sur chaque diagramme comme par exemple la sélection d'un élément de menu ou le clic sur un noeud de diagramme. Ceci permet ensuite d'implémenter des algorithmes de mise en page spécialisés ou d'animer des scénarios spécifiques.

La figure ci-après donne un aperçu des possibilités d'utilisation que nous avons exploitées pour mettre en place un prototype de notre environnement.

Figure 45 : Ecran d'utilisation de HARDY

En outre, du point de vue de l'interface, un des avantages de HARDY est que pour une même représentation profonde d'un objet, il est possible d'avoir plusieurs représentations de surface.

1.2. wxCLIPS

« wxCLIPS » est une extension de CLIPS qui a aussi été écrite au laboratoire « *Artificial Intelligence Applications Institute* » de l'Université d'Edimbourg. Différentes versions sont disponibles sur des plates-formes Windows, Sun Open Look et Motif.

wxCLIPS est l'union de wxWindows et CLIPS, c'est-à-dire un ensemble de fonctions CLIPS 6.0 avec éventuellement ses extensions de logique floue, et disposant d'une large portion des fonctionnalités graphiques wxWindows.
wxCLIPS a été développé pour permettre aux programmeurs CLIPS d'écrire des programmes portables et graphiques. Ceci permet d'avoir un style de programmation guidé par la notion d'événement ainsi qu'un ensemble de fonctions d'interface utilisateur graphique.
wxCLIPS est disponible dans trois formats :

- en tant que bibliothèque, ou bien
- intégré dans d'autres produits, ou bien
- en tant qu'exécutable avec une interface simple.

En outre, HARDY possède une version simplifiée du noyau du CLIPS 6.0, « wxCLIPS », mais aussi des fonctions spécifiques pouvant être appelées depuis wxCLIPS. Ceci permet notamment de mettre en place des fonctions puissantes permettant le contrôle de la modélisation, la génération de boîtes de dialogue avec l'utilisateur, la génération de documents ou encore le couplage avec une base de données par l'intermédiaire de liens ODBC. L'utilisation du langage « wxCLIPS » est aussi d'autant plus évidente que « HARDY » a été programmé avec « wxCLIPS ».

De plus, wxCLIPS et HARDY sont des outils existant sur des plates-formes hétérogènes (Windows 3.x, Windows 95, Windows NT, Unix, …) car ils sont supportés par « wxWindows » [Smart, 1995b].

1.3. wxWindows

wxWindows, développé aussi à « *l'Artificial Intelligence Applications Institute* » (A.I.A.I.) de l'Université d'Edimbourg, a été conçu pour fournir des moyens flexibles et peu coûteux permettant de maximiser le rendement investi dans le développement d'applications à interface utilisateur graphique. C'est une bibliothèque de classes C++ qui supporte Open Look (XView), Motif et Windows.

2. Présentation du processus de développement de notre prototype et des outils développés

Le prototype que nous avons développé supporte une version simplifiée du processus de développement que nous avons détaillé dans le chapitre 6.
Il est important de souligner ici qu'un travail important a été réalisé afin de proposer une interface-utilisateur la plus convenable possible et notamment en français. En effet, nous avons personnalisé les divers et nombreux barres de menus et menus flottants contextuels grâce à la gestion des événements que nous permettent les logiciels wxCLIPS et HARDY.

Nous allons présenter ci-après de façon succincte les différentes étapes que nous avons mises en place ainsi que les outils associés.
L'annexe 5 de cette thèse consacre une place plus importante aux logiciels utilisés pour implanter le prototype et ses différents outils.

2.1. Les différentes étapes du processus

Même si nous avons souligné que la mise en œuvre de la spécification devait pouvoir se concevoir selon les besoins courants des pédagogues et non pas selon un processus ascendant ou descendant, le prototype que nous avons implanté supporte uniquement un processus descendant (de la matrice disciplinaire vers les situations-problèmes) [43].

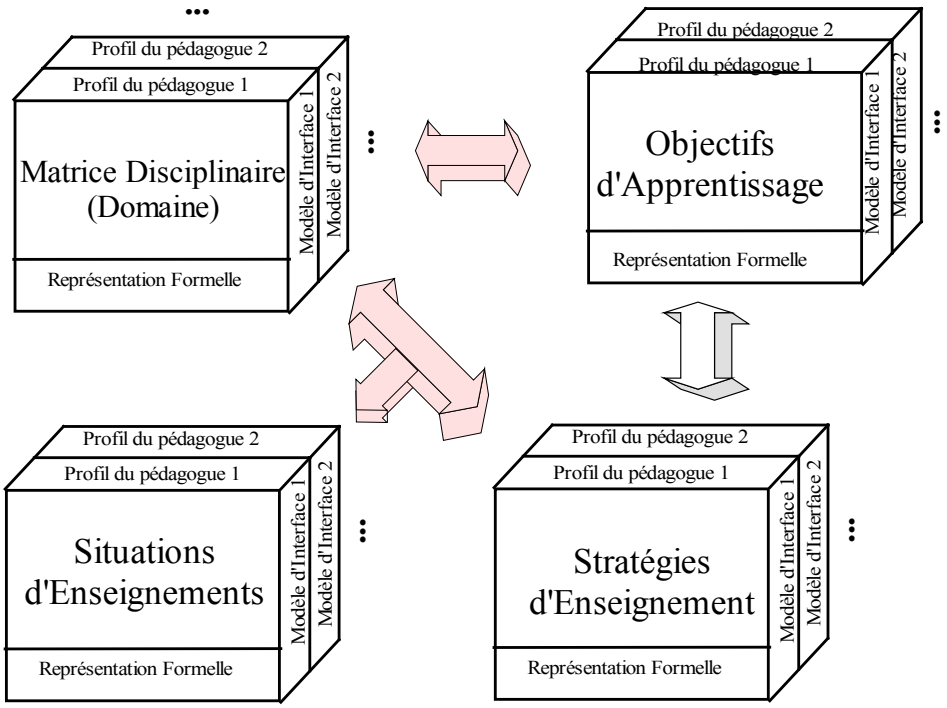

Figure 46 : Fonctionnement de notre environnement

2.1.1. Matrice disciplinaire

Servant de base, elle permet de définir le *contexte* de travail. Grâce à un formalisme graphique proche de celui du modèle objet de OMT [Rumbaugh *et al.*, 1994], le pédagogue fixe *l'univers du discours* en terme de *notions* auxquelles sont attachées des *compétences*. Dans ce modèle, les objets possèdent des *attributs* représentant les sous-notions relatives à la notion composante. De même, les objets ont des

[43] La mise en place d'un processus non contraignant ne constitue à notre avis que des problèmes inhérents à la programmation et non à la conception du produit.

méthodes qui représentent les compétences qui sont en fait les *principes unificateurs* relatifs à l'objet que l'on veut faire acquérir par l'apprenant. Ces principes se rapportent au domaine ou bien sont transdisciplinaires.
Ces objets sont reliés par trois sortes de *liens* : composition, référence et héritage qui représentent respectivement les liens d'agrégation, d'association et d'héritage au sens courant du génie logiciel.

Nous n'avons pas implanté dans notre prototype l'étape de mise en place du focus sur la matrice disciplinaire qui est un sous-ensemble de la matrice disciplinaire. Pour des raisons de commodité, nous considérons que le pédagogue spécifie directement les parties du focus de la matrice.

2.1.2. Objectifs d'apprentissage

A chaque matrice disciplinaire, c'est-à-dire à chaque carte de diagramme de la matrice, peuvent être associés un ou plusieurs *objectifs d'apprentissage* que le pédagogue fixe pour les apprenants.
Nous avons choisi de représenter les *objectifs d'apprentissage* que le pédagogue fixe pour les apprenants à l'aide d'un modèle dynamique tel qu'il est d'ordinaire abordé dans les méthodes objet comme OOA [Shlaer et Mellor, 1991] ou OMT [Rumbaugh *et al.*, 1994] en émettant cependant des restrictions. En effet, nous assimilons la représentation des objectifs d'apprentissage à celle des états qui, une fois reliés, constituent des *stratégies d'enseignement*.

2.1.3. Stratégies globales d'enseignement

Les différentes stratégies d'enseignement représentent donc les différents cheminements, c'est-à-dire les scénarios d'apprentissage possibles prévus parmi les objectifs d'apprentissage. A ce niveau, nous ne contraignons pas l'ordre dans lequel l'apprenant devra acquérir les différents objectifs. Ce sont en fait les différents états parcourus qui représenteront la progression de l'apprenant. L'évolution de la dynamique est basée sur les principes Evénement - Condition - Action : lorsqu'un *événement* se produit, si les *conditions* sont vérifiées alors il convient d'effectuer les *actions*.
Cette planification permet donc de surveiller si les actions de l'apprenant respectent bien la stratégie fixée au départ par le pédagogue.

2.1.4. Situations-problèmes

A chaque cheminement possible défini par le pédagogue est associé un ou plusieurs obstacles supposés par l'enseignant ; le franchissement de cet obstacle constitue une acquisition de connaissances par l'apprenant.

A cet obstacle correspond une *situation-problème* qui permet de comprendre si l'apprenant peut ou non franchir l'obstacle ainsi que les raisons de ce franchissement ou de l'échec.
Une situation-problème se définit en terme de *notions à acquérir*, de *tâches* à réaliser, de *consignes*, de *matériel*, de *temps* et de *critères de réussite et d'échec*.
Cette planification est le point essentiel de l'enseignement puisqu'elle permet d'assurer l'évaluation puis le contrôle et la supervision des connaissances effectives de l'apprenant puis de proposer dans une étape suivante les situations les plus appropriées.
Au sein de notre prototype, nous nous sommes limités à la définition des situations-problèmes et nous ne fournissons pas les outils pour la mise en place des situations plus traditionnelles (transfert, métacognition, etc...) qui pourraient toutefois l'être aisément en intensifiant le travail de programmation.

2.2. Outils associés

Des outils de gestion de l'environnement sont aussi implantés afin de gérer les inscriptions, les droits d'accès ainsi que les fonctions pour une base de données permettant l'échange de documents.
Le couplage de l'environnement de programmation HARDY-wxCLIPS avec la base de données Microsoft Access via des liens ODBC nous a conduit à définir une base de données comportant cinq tables permettant non seulement la gestion des utilisateurs, mais aussi la sauvegarde des spécifications en fonction des quatre étapes implantées.

En ce qui concerne la gestion des utilisateurs, nous avons défini quatre types d'utilisateurs (simple utilisateur, utilisateur avancé, super utilisateur et administrateur) dont les pouvoirs sont de plus en plus importants notamment pour la réutilisation de spécifications existantes.
Nous avons choisi de donner la possibilité à chaque pédagogue-utilisateur de stocker tout ou partie de son travail dans une base de données. La structure que nous avons donnée à cette base permet une grande flexibilité et fournit de puissantes possibilités de réutilisation. En effet, pour un schéma à stocker, nous avons choisi de mémoriser l'ensemble des noeuds présents dans ce schéma avec, pour chaque noeud, plusieurs informations :

- son nom,
- sa description textuelle courte,
- le nom du propriétaire (réalisateur), ainsi que
- le nom et le chemin de sauvegarde du fichier contenant le schéma.

Cette organisation nous permet ainsi en fonction du type d'utilisateur de varier la puissance de recherche pour assister le pédagogue lors de la modélisation. Ainsi, un *simple utilisateur* n'aura accès

dans la base de données, qu'à ses propres sauvegardes. Par contre, un *utilisateur avancé* pourra consulter, ou se verra proposer, des spécifications issues de son travail mais aussi de celui de tous les simples utilisateurs. De même, un *super utilisateur* aura à disposition l'ensemble des sauvegardes de tous les pédagogues [44].
Notons ici que notre environnement dispose aussi de fonctions de plus haut niveau permettant par exemple à un *administrateur* de trouver des incomplétudes dans la base de données. Ainsi, il est possible de parcourir la base de connaissances afin de supprimer les éléments dont le fichier physique de sauvegarde a déjà été effacé.

Pour chaque étape du processus de spécification, les outils plus conventionnels mis à disposition de l'utilisateur sont :

- afficher la liste des éléments présents sur la carte,
- afficher la liste des liens d'un certain type, de tous les types,
- rechercher un élément ou un lien particulier,
- visualiser la représentation détaillée d'un élément,
- renommer la carte de diagramme (modification propagée dans l'ensemble de la spécification),
- sauvegarder tout ou partie de la spécification,
- consulter les spécifications sauvegardées,
- visualiser les spécifications sauvegardées,
- récupérer les spécifications sauvegardées,
- passer d'une étape du processus à une autre sans difficulté,
- etc...

Nous avons choisi, dans notre prototype, de proposer automatiquement la consultation de la base de données chaque fois que le pédagogue-utilisateur définit un nouvel élément pour lequel il ne donne une valeur qu'à l'attribut « *nom* ». Cette option [45], quoique basique, démontre qu'il est possible de guider l'usager au fur et à mesure de sa spécification, et que des critères plus élaborés permettraient une plus grande finesse dans l'assistance proposée.

Un autre aspect important que nous avons implanté concerne la génération de documents associée à la spécification. Ainsi, nous avons mis en place deux fonctionnalités permettant à un pédagogue d'avoir une vue plus globale de son travail.

---

[44] Notons toutefois ici une restriction dans la mesure où la définition du type de l'utilisateur demeure statique dans l'état actuel du prototype mais l'ajout de fonctions de guidage spécifiques permettrait de résoudre ce problème.

Ainsi, il est possible de générer, à partir de la spécification courante, un document détaillant l'ensemble des étapes, avec pour chaque étape la définition de chacun des composants ainsi que leurs interactions. La génération de ce document qui se présente sous la forme d'un document consultable dans tout traitement de texte s'effectue en deux étapes. Dans un premier temps nous avons implanté des fonctions wxCLIPS permettant de générer un document au format LaTEX qui comprend des images représentant les diagrammes des spécifications ainsi que le détail de leur composition. Ensuite, par l'intermédiaire du logiciel « Tex2RTF » développé également au laboratoire AIAI, nous avons généré un fichier au format « RTF ».

De même, il est possible de générer un fichier au format « HTML », par l'intermédiaire du logiciel « RTFHTM », qui transforme un fichier au format « RTF » en un fichier au format « HTML » qui peut ensuite être consultable sur Internet [46].

Même si les documents générés ne sont qu'une définition textuelle, l'objectif de ces générations de documents est de montrer que la génération d'un prototype sous la forme d'un programme exécutable, par exemple dans le langage JAVA, est tout à fait réalisable et raisonnable.

[45] Il est également possible pour le pédagogue d'annuler ce caractère automatique par une fonction accessible dans la barre de menu.

[46] Bien entendu à condition de mettre le fichier sur un serveur... Nous transformons aussi les images visualisables dans un traitement de texte au format « BMP » vers le format « GIF ».

Chapitre 11 : Conclusions et perspectives

« It is said that one machine can do the work of fifty ordinary people.
No machine, however, can do the work of one extraordinary person. »

Paragraphe 1 : Conclusions

Le domaine sur lequel nous avons situé nos travaux de recherche est l'enseignement qui, pour le moment, n'a été qu'effleuré par les nouvelles technologies. Même si nous pouvons l'expliquer par les habitudes corporatistes fortes, la puissance des outils informatiques actuels ainsi que l'émergence du multimédia et de l'Internet tendent à gommer le fossé entre les enseignants et les nouvelles technologies.
L'objectif que nous nous sommes fixés consiste ainsi à définir un environnement informatique d'assistance à la construction d'enseignements.
La figure suivante illustre alors le schéma d'informatisation que nous voulions éviter.

Figure 47 : Du rôle de l'informatique...

Par conséquent, nous avons choisi de placer le pédagogue-utilisateur au centre de nos préoccupations pour la définition des outils informatiques à mettre à sa disposition. A partir des besoins exprimés par un enseignant (chapitre 5), nous avons donc défini, non seulement une méthode permettant à l'enseignant de spécifier formellement ses besoins, mais aussi une architecture informatique sur laquelle repose cette méthode.
Comme nous l'avons vu dans [Rumbaugh *et al.*, 1994], une méthode se décline notamment avec les composants suivants :

- un ensemble de *concepts fondamentaux de modélisation* qui permettent de faciliter l'expression des connaissances sur le problème et la solution. La première partie de ce document nous a donc permis de fixer la terminologie issue des travaux en sciences de l'éducation, et basée sur les concepts de matrice disciplinaire et de situation-problème.
- un *processus de développement* itératif permettant la construction progressive des modèles. Celui-ci fait l'objet de la seconde partie de la thèse.
- les *notations* et les *modèles de description* dont le rôle réside dans la compréhension et la modification de la méthode sont décrits dans la troisième partie.

Quoi que nécessaires, ces composants ne sont pas suffisants dans la mesure où l'environnement informatique a pour but d'aider les enseignants à mieux structurer leurs idées et à les guider efficacement sans que cela ne les contraigne trop dans leur façon quotidienne de procéder.
Des outils « CASE » ou meta-CASE apportent ici un avantage pour l'élaboration d'une spécification formelle que l'on peut évaluer par prototypage et qui permet, de façon précoce, de faire ressortir les différences entre les besoins souhaités et ceux réellement exprimés (modélisés). En outre, pour être facilement compréhensibles, et par conséquent utilisables et utilisés, ces outils doivent fournir une vraie assistance dans le processus tout en étant suffisamment flexibles.
Se pose alors le choix d'une représentation structurelle des connaissances. Nous avons choisi une approche basée sur la notion d'ontologie. Ceci nous permet de décrire, non seulement le vocabulaire et la signification de chaque élément de celui-ci, mais aussi les règles logiques qui décrivent les relations possibles et les contraintes entre éléments.
Le couplage des paradigmes objet et logique assure également la prise en compte de la représentation profonde des connaissances à l'aide d'une approche Modèle - Point de Vue - Contrôleur [Krief, 1992].
En ce qui concerne la réalisation effective d'un prototype de notre environnement, nous avons utilisé un couplage en HARDY, qui est un logiciel diagrammatique de modélisation des connaissances, et wxCLIPS qui correspond au générateur de système expert aux caractéristiques orientées-objets CLIPS auquel est adjoint une interface de programmation (API) graphique.

Paragraphe 2 : Perspectives

L'environnement informatique que nous avons construit permet donc à tout pédagogue de spécifier de façon formelle les enseignements qu'il souhaite mettre en place. Il est également possible d'autoriser différents points de vue sur une même spécification sans imposer une façon unique de procéder à celle-ci. La flexibilité, indispensable pour une utilisation effective du système par des enseignants, est donc bien assurée. Toutefois, il convient de pousser plus avant les recherches en ce qui concerne l'assistance au processus de spécification afin que notre environnement dispose réellement des fonctionnalités d'un système tuteur intelligent utilisant pleinement des possibilités d'aides de type « compagnonnage » ou de type perturbateur.

Notre environnement méthodologique de spécification des connaissances doit être capable d'évoluer, c'est-à-dire de s'enrichir de nouvelles façons de procéder non prévues dans la conception initiale des pédagogues. Considérant avec [Lehmann, 1989] que les capacités d'évolution d'un système tiennent principalement à la façon dont celui-ci est conçu et programmé, nous croyons que les architectures à base d'agents [Labidi et Lejouad, 1993] sont les mieux adaptées pour produire de tels environnements. Dans une telle architecture, chaque agent va disposer de :

- un savoir-faire qui est une interface permettant la déclaration des connaissances et des compétences de l'agent, et
- des croyances qui sont des connaissances pas forcément objectives sur lui-même et sur les autres agents et qui lui permettent d'assurer trois fonctions essentielles pour assurer l'assistance à l'usager et être capable d'évoluer : percevoir, décider et agir.

En effet, les architectures plus classiques, qu'elles soient fonctionnelles ou à objets, n'ont pas ces capacités car les flots de contrôle passent d'une fonction à une autre ou d'un objet à un autre, alors que des agents ont la possibilité d'analyser en permanence ce qui se passe dans leur environnement.

De plus, selon [Neches *et al.*, 1991], l'utilisation des ontologies est la base du partage des connaissances entre agents intelligents. [Gruber, 1993] définit aussi des ontologies permettant le partage et la réutilisation des connaissances et fournit alors la définition suivante : « *une ontologie est la description des concepts et relations pouvant exister pour un agent ou une communauté d'agents* ».
En outre, nombre de chercheurs s'orientent actuellement vers une approche multi-agents pour concevoir et développer des EIAO. Citons par exemple [Ayala et Yano, 1996] ou encore [Frasson *et al.*, 1996]. Ceci a donné naissance à de nouveaux systèmes appelés tuteurs intelligents distribués [Futtersack, 1994]. La plupart des architectures proposées reposent sur des systèmes organisés autour d'un planificateur pédagogique supervisant les sessions d'apprentissage et le fonctionnement des autres agents (résolveur, explicateur, etc…) [Goblet et Benslimane, 1995]. Le besoin d'opportunisme est assuré par une planification dynamique des activités et des réactions du système. Cette approche permet de résoudre les problèmes de contrôle, de communication et de distribution des connaissances de chacun. De plus, le caractère imprévisible de l'apprenant oblige souvent le planificateur à reconsidérer ses plans et à réviser les connaissances pour améliorer son efficacité.
L'étude des travaux de [Millet, 1997], qui a défini SERAC, un système d'évaluation et de révision automatisé des connaissances en EIAO, basé sur une architecture centrée sur un agent réviseur, est ici une étape à court terme pour l'avancement de nos travaux de recherche.
L'annexe 6 de cette thèse revient sur cette notion d'agent pour la construction d'EIAO et pour la recherche d'information sur Internet.

De même, il conviendra de considérer les travaux de [Moinard et Joab, 1997] relatifs au projet DIAPASON (système d'aide à la formation fondé sur la simulation pour les chargés de conduite du réseau électrique moyenne tension en partenariat avec EDF-GDF). Dans ce système, le suivi et l'évaluation pas à pas des actions de l'utilisateur concernent non seulement les résultats obtenus mais aussi la méthode de travail employée en comparaison avec un résolveur dynamique et paramétrable couplé au dispositif industriel simulé.

Dans les perspectives de recherche, il serait également intéressant de prendre en compte les impacts dus au travail collaboratif, notamment pour l'intégration des fonctions de communication synchrone et asynchrone. Il conviendra alors de se positionner par rapport aux projets existants et notamment les projets du « *Campus Virtuel* » que nous avons abordés dans ce document mais aussi PRODITE [47], acronyme de « *PROject for Distributed Interactive Teaching Environment* », dont le but est la création d'une boîte à outils permettant à un enseignant de pouvoir créer ses cours, ses examens et exercices en utilisant une interface WAS (acronyme de « *Web Authoring System* »), le but final étant de rendre l'utilisation des divers outils transparente à l'utilisateur.

Cependant il est important de conclure en signalant que, pour l'enseignement et l'apprentissage, tout comme pour la plupart des phénomènes complexes, il n'y a pas de règle d'or, pas de méthode miracle dont l'application résoudra tous les problèmes d'instructions dans tous les domaines, pour tous les types d'apprenants. Ainsi, même si la technologie informatique doit servir de catalyseur pour une nouvelle forme d'éducation, les approches traditionnelles gardent toujours leur place. Les enseignants vont continuer à élaborer les programmes des cours, à faire leur travail en utilisant des méthodes différentes, et à être les guides et les maîtres des élèves qui ont besoin d'aide. L'informatique prouve alors toute sa pertinence dans la complémentarité qu'elle apporte, tant pour l'élève que pour l'enseignant.

Enfin, il serait intéressant de replacer nos travaux de recherche en fonction de la problématique de notre laboratoire de recherche qui est celle des systèmes de production informatisés. Ces derniers, pour être efficaces, nécessitent une forte coopération entre les décisions de l'opérateur humain et celles prises par la machine. L'approfondissement de nos travaux dans le domaine de l'interaction homme-machine pourrait, par exemple, concerner la communication entre le système physique de production, son système de pilotage et le système de prise de décision.

[47] Pour de plus amples renseignements, se reporter à l'adresse web suivante : http://lithsun11.epfl.ch:5000/www/milieu.htm

Glossaire des définitions et acronymes

A.G.D. : Atelier de Génie Didacticiel.

A.G.E. : Atelier de Génie Educatif.

A.G.L. : Atelier de Génie Logiciel.

Acteur [Agha *et al.*, 1993] : Les acteurs sont des entités autonomes concurrentes communiquant de façon asynchrone par envoi de messages.

Agent [Prince, 1996] : Toute entité jouant un rôle dans un système d'information : individus, systèmes informatiques et leurs accessoires.

Algorithme [Weinberg, 1995] : C'est une méthode de résolution de problème issue de l'intelligence artificielle. Dans un problème de recherche, la démarche « algorithmique » se base sur un inventaire systématique de toutes les possibilités existantes. Plus généralement, l'algorithme désigne une suite d'opérations élémentaires logiques régies par un ordre rigoureux permettant de résoudre certains problèmes ou d'effectuer certaines opérations. Par exemple, le mode d'emploi pour construire un meuble en kit n'est rien d'autre qu'un algorithme.

Alternance [Meirieu, 1994] : Modèle pédagogique articulant des activités sur un terrain de « production » (stage d'observation ou d'initiation) et des activités dans un lieu de formation spécifique. Cette articulation ne peut être véritablement interactive que si, dans un premier temps, l'apprenant identifie dans le stage, les difficultés qu'il rencontre (ou que rencontrent les professionnels qu'il observe), les transforme, dans un deuxième temps, en objectifs d'apprentissage, et recherche, dans un troisième temps, dans le cadre du dispositif de formation, des ressources lui permettant d'effectuer ces apprentissages et dont la pertinence sera, enfin, vérifiée, lors du retour en stage. L'articulation " difficultés / objectifs / ressources », est donc constitutive d'une véritable alternance et doit présider à l'élaboration des outils de liaison entre stage et formation théorique.

Analyse [Weinberg, 1995] : C'est une démarche de la pensée qui consiste à décomposer un objet ou une notion en éléments simples. Plus généralement, l'analyse correspond à toute démarche de pensée qui se propose de décrire les éléments constitutifs d'un objet donné.

Apprendre [Develay, 1993] : Cela renvoie à une conjonction de plusieurs sémantiques : enseigner, former, éduquer, instruire et amène une difficulté de positionnement entre l'apprenant et l'enseignant. La forme la plus élaborée de l'apprentissage est donc la compréhension.

Archétype mental [Meirieu, 1994] : Schéma organisateur combinant des algorithmes procéduraux ou des opérations mentales (*prototype*), maîtrisé dans une situation donnée et sur un objet déterminé et pouvant être mis en œuvre dans une situation nouvelle ou en face d'un objet nouveau dont l'analyse aura montré qu'ils se prêtaient à un traitement identique.

B.C. : Base de Connaissances.

B.D. : Base de Données.

Behaviorisme [Meirieu, 1994] : Conception de l'activité intellectuelle qui s'attache aux corrélations entre les stimuli extérieurs et les comportements. Cette conception a inspiré les premiers travaux de la « *pédagogie par objectifs* » qui s'efforçaient de traduire systématiquement les contenus de programme en comportements attendus de l'apprenant. Elle est aujourd'hui largement remplacée par le *mentalisme*.

CASE : Acronyme de « Computer Aided Software Engineering ». Terme générique décrivant un support automatisé ou semi-automatisé pour le génie logiciel.

Cahier des charges : Il permet de fixer de façon la plus claire et la plus précise possible l'ensemble des besoins exprimés par les utilisateurs du produit.

Capacité [Meirieu, 1994] : Activité intellectuelle stabilisée et reproductible dans des champs divers de connaissance ; terme utilisé souvent comme synonyme de « savoir-faire ». Aucune capacité n'existe « à l'état pur » et toute capacité ne se manifeste qu'à travers la mise en œuvre de contenus.

Capacité méthodologique transdisciplinaire [Meirieu, 1994] : Possibilité de discerner et de mettre en œuvre, à partir des habituels intitulés d'exercices scolaires, les opérations mentales requises selon les procédures personnelles identifiées comme les plus efficaces. Il faut noter que les mêmes intitulés (comme « démontrer », ou « mémoriser ») peuvent recouvrir des opérations mentales différentes, alors que des intitulés différents (« vérifier » et « transposer », par exemple) peuvent parfois requérir des opérations mentales identiques. La maîtrise méthodologique peut donc être décrite comme la capacité

de se saisir d'un objet en fonction de ses contraintes propres et du projet d'appropriation que l'on en a. Cette maîtrise méthodologique se manifeste par l'interaction d'un projet et d'un objet et non par l'application d'une méthode donnée à un objet indifférencié.

Champs notionnels [Develay, 1993] : Ce sont l'ensemble des *notions* qui, mises en synergie, donnent un sens à celle que l'on souhaite enseigner. Ainsi, si un élève n'a pas assimilé une notion, il n'a peut-être pas assimilé un élément constitutif de la trame de cette notion.

Classe de problèmes [Meirieu, 1994] : Ensemble de problèmes ayant une structure commune leur permettant d'être résolus par l'usage d'un même *programme de traitement*. Une classe de problèmes ne doit être ni sous-spécifiée (trop vaste pour être corrélée à un programme de traitement efficace), ni sur-spécifiée (trop limitée et restreignant l'usage du programme de traitement à des conditions reproduisant exactement la situation initiale de sa présentation ou de son apprentissage). C'est par un travail de recherche des conditions d'efficacité du programme de traitement que l'on est progressivement capable d'identifier une classe de problèmes.

Clôture productive (ou principe d'économie) [Meirieu, 1994] : Activité déployée au moindre coût en face d'une difficulté (aussi bien dans le cadre d'une pédagogie des situations-problèmes que dans le cadre d'une pédagogie du projet ou d'une pédagogie par alternance) et qui permet de venir à bout de la difficulté sans apprendre. La tâche du formateur, soucieux de faire effectuer des apprentissages, est donc d'empêcher la clôture productive et d'articuler à la difficulté repérée une situation didactique précisément ciblée et évaluée.

Coach [Mengelle, 1995] : Type de guidage à l'usager dans un système tuteur intelligent.

Compétence [Meirieu, 1994] : Savoir identifié mettant en jeu une ou des capacités dans un champ notionnel ou disciplinaire déterminé. Plus précisément, on peut nommer compétence la capacité d'associer une *classe de problèmes* précisément identifiée avec un *programme de traitement* déterminé.

Concept Intégrateur [Develay, 1993] : Les concepts intégrateurs, à un niveau d'enseignement donné, organisent en une structure cohérente l'ensemble des faits et des notions abordées. Ils constituent en quelque sorte les principes organisateurs, au niveau notionnel, d'une discipline enseignée. Leur identification pour une discipline et pour un niveau d'enseignement donné met en cohérence l'ensemble des connaissances à enseigner autour du « noyau dur » de la discipline.

Conception cognitive [Prince, 1996] : Ensemble cohérent de connaissances aussi bien déclaratives que procédurales possédées par un agent et potentiellement mis en œuvre par lui pour effectuer diverses tâches qui lui incombent (production, interprétation ou acquisition des messages par exemple).

Conflit socio-cognitif [Meirieu, 1994] : Interaction cognitive entre des sujets ayant des points de vue différents. Pour que l'interaction ait réellement lieu, il convient que chaque sujet prenne en compte le point de vue d'autrui et intériorise le conflit socio-cognitif. Il y a alors conflit de centrations, contradiction et, si elle est surmontée, progression intellectuelle. On observe que de nombreuses situations de communication ne sont pas interactives dans la mesure où les sujets y abandonnent leur représentation ou l'imposent à autrui. La mise en groupe d'apprentissage constitue un dispositif où le mode de fonctionnement sollicite une véritable interaction.

Connaissance [Prince, 1996] : La connaissance est le mode d'emploi permettant de transformer les données en informations. C'est donc ce qui est associé au passage entre le signifiant et le signifié. Dans un système artificiel, les connaissances obéissent à la loi de compositionalité et sont auto-discursives.

Consigne-but [Meirieu, 1994] : Définition d'un *projet* à réaliser dans une *situation didactique* en termes de » produit fini » et renvoyant essentiellement au registre des motivations des apprenants.

Consignes-critères [Meirieu, 1994] : Consignes permettant au sujet de déterminer si la *tâche* (ou le produit) qu'il doit réaliser est conforme aux attentes que le formateur manifeste à son égard. La connaissance de l'ensemble des critères de qualité du produit fini oriente ainsi l'activité du sujet qui sait précisément ce que l'on attend de lui. Les consignes-critères concernent donc la tâche à réaliser (ce à quoi l'on déterminera la conformité de ce que l'on a fait) et non l'objectif à atteindre (ce à quoi l'on déterminera l'exactitude de ce que l'on a compris) ; car, il ne saurait y avoir d'appropriation de l'objectif avant l'apprentissage, sauf à penser que cet apprentissage est déjà effectué. Le formateur peut, toutefois, dans certains cas, considérer que la conformité aux consignes-critères fonctionne comme *indicateur* de l'objectif visé.

Consignes-procédures [Meirieu, 1994] : Consignes multiples proposant des stratégies différenciées permettant d'effectuer, selon des itinéraires multiples, l'opération mentale requise par le dispositif didactique mis en place.

Consignes-structure [Meirieu, 1994] : Consignes ayant un caractère obligatoire pour tous les

apprenants dans un dispositif didactique déterminé ; pour le formateur, elles incarnent la structure de l'opération mentale à effectuer ; pour le forme, elles décrivent le mode de fonctionnement du dispositif.

Contrat didactique [Develay, 1993] : Ensemble de devoirs (implicites ou explicites) établis d'un commun accord entre l'enseignant et l'apprenant dans le but d'améliorer les connaissances de ce dernier.

Constructivisme : Théorie de l'apprentissage située par PIAGET à mi-chemin entre le *Behaviorisme* et l'Idéalisme.

Critères [Meirieu, 1994] : Eléments permettant au sujet de vérifier qu'il a bien réalisé la tâche proposée et que le produit de son activité est conforme à ce qu'il devait obtenir. La liste de ces critères peut être établie avec les apprenants comme le proposent les théoriciens de l'évaluation formatrice ; elle est parfois nommée alors « carte d'étude ».

Décentration [Meirieu, 1994] : Première phase de l'opération mentale de déduction. Au cours de la décentration le sujet se place en face de ses propres hypothèses ou productions avec le point de vue d'autrui et s'efforce de les considérer comme s'il n'en était pas l'auteur. Il se situe ainsi envers elles du point de vue de leurs conséquences. Il sera alors en mesure dans une deuxième phase, de stabiliser ou modifier ses propositions initiales.

Déclaratif [Weinberg, 1995] : La connaissance « déclarative » est un savoir qui porte sur un contenu, sur des attributs d'un objet donné. Par exemple, pour construire une phrase, il faut connaître des mots et leur sens (connaissances déclaratives).
Déclaratif [Develay, 1993] : Les connaissances déclaratives - terme emprunté à la psychologie cognitive - sont de l'ordre du discours, du savoir alors que les *connaissances procédurales* sont de l'ordre de l'action, du savoir-faire.

Décontextualisation [Meirieu, 1994] : Opération par laquelle un sujet utilise une acquisition dans un autre contexte que celui qui en a permis l'apprentissage. Elle est ainsi la première phase de l'identification d'un acquis, qui doit se prolonger par la *mentalisation*. Il faut préférer la notion de décontextualisation à celle de transfert, dans la mesure où cette dernière suppose une antériorité de l'acquisition, alors que c'est au terme du processus de décontextualisation et mentalisation que l'acquisition est véritablement stabilisée.

Déduction [Weinberg, 1995] : La déduction est l'inverse de l'induction. Elle consiste à partir de propositions données - les prémisses - pour en tirer des conclusions par enchaînements (inférence) logiques.

Déduction (pensée déductive) [Meirieu, 1994] : Opération mentale par laquelle un sujet se place du point de vue des compétences d'un acte ou d'un principe, met ceux-ci à l'épreuve de leurs effets et stabilise ou modifie sa proposition initiale. Une modalité particulière de la pensée déductive est l'évaluation réflexive : évaluation dans laquelle le sujet intègre le point de vue d'autrui (lecteur, auditeur, correcteur...) sur son propre travail pour le modifier.

Dialectique (pensée dialectique) [Meirieu, 1994] : Opération mentale par laquelle un sujet met en interaction des lois, notions, concepts, fait évoluer des variables dans des sens différents, pour accéder à la compréhension d'un système.

Didactiques des disciplines [Meirieu, 1994] : Réflexions et propositions sur les méthodologies à mettre en œuvre pour permettre l'appropriation de contenus spécifiques. Quoique très légitimement élaborées à la lumière de la réflexion épistémologique afférente à chaque discipline et à des apports de la psychologie cognitive, les didactiques spécifiques comportent toujours, plus ou moins explicitement, des choix de valeurs, des représentations de la culture, du sujet apprenant, de l'éducateur et de son rôle.

Didactique générale [Meirieu, 1994] : Elaboration de modèles d'intelligibilité de l'apprentissage adossés aux apports de la psychologie cognitive, porteurs - implicitement ou explicitement - de valeurs, ouverts à une opérationalisation possible et permettant d'intégrer les spécificités disciplinaires. La didactique générale est ainsi une invention de modèles qui tentent d'articuler quatre pôles : le pôle psychologique, le pôle axiologique, le pôle praxéologique, le pôle épistémologique.

Discipline [Develay, 1993] : Elle est constituée par un ensemble d'objets d'enseignement spécifiques (manuels de cours et d'exercices, cartes, instruments, …). Toutefois, il serait donc plus juste de parler de champs disciplinaires que de disciplines. En effet, les mêmes *concepts intégrateurs* et les mêmes *connaissances procédurales* sont présents au sein de disciplines différentes, laissant alors percevoir une possible interdisciplinarité voire même une transdisciplinarité.

Dispositif [Meirieu, 1994] : Construction didactique élaborée à partir d'une opération mentale que l'on veut faire effectuer au sujet pour l'amener à une acquisition donnée. Le dispositif met en œuvre des matériaux et des consignes-structure qui, ensemble, incarnent l'opération mentale. Il doit être isomorphe à celle-ci et négociable selon plusieurs stratégies. Il peut être individualisé ou interactif ;

dans ce dernier cas, il s'agit d'un groupe d'apprentissage.

Divergence (pensée divergente) [Meirieu, 1994] : Opération mentale par laquelle un sujet met en relation des éléments appartenant à des domaines différents, établit des associations nouvelles, des rapports originaux entre les choses, les mots, les notions, les registres d'explication.

Donnée [Prince, 1996] : Une donnée est tout signifiant susceptible d'être capté, enregistré, transmis ou modifié par un agent cognitif de traitement de l'information, naturel ou artificiel (l'ensemble de ces fonctions n'est pas obligatoire pour tout agent). C'est la matière à l'état brut.

E.A.O. : Enseignement Assisté par Ordinateur.

E.I.A.O. : Environnements Interactifs d'Apprentissage avec Ordinateur.

Enigme [Meirieu, 1994] : Savoir entrevu qui suscite le désir de son dévoilement. L'énigme naît ainsi de ce que l'apprenant sait déjà et dont le formateur sait montrer le caractère partiel, ambigu, voire mystérieux. Le désir de savoir peut ainsi émerger face à une situation-problème si celle-ci est construite à partir d'une évaluation diagnostique des compétences et capacités d'un sujet. Le déjà-là problématisé offre la possibilité de son dépassement.

Enseigner [Develay, 1993] : Enseigner vient du latin *insignire* qui vient lui-même de *signare* : placer un signe distinctif, un *signum*. Enseigner, c'est laisser sa marque, son empreinte chez le sujet. Ainsi, enseigner évoque des images impositives, descendantes, à la manière d'un moule opérant sur une matière assez malléable pour prendre forme et la garder.

Epistémologie d'une discipline [Meirieu, 1994] : Caractère spécifique d'une démarche disciplinaire attaché à la fois aux contraintes de son objet et à ses modalités internes de validation. Le statut de la preuve et le « critère du vrai » sont, ainsi, constitutifs d'une épistémologie disciplinaire.

Ergonomie [Norman, 1996] : C'est l'étude de l'interaction homme-machine, mais elle concerne en premier lieu les aspects physiques de la conception (taille des composants, leur disposition, …).

Ergonomie cognitive [Norman, 1996] : Elle souligne le côté mental de la conception, la facilité de compréhension et d'utilisation de l'équipement. Dans l'ergonomie cognitive, l'homme est placé au premier plan, la technologie au second, c'est la garantie que le résultat final correspondra aux besoins des gens à qui il est censé servir. C'est le manque d'ergonomie cognitive dans la conception qui a

conduit à cette situation moderne de complexité, de confusion et d'erreur.

Expert [Prince, 1996] : Un expert est un agent cognitif humain possédant soit des connaissances soit un savoir-faire (et éventuellement les deux) sur un sujet ou un domaine particulier, le distinguant des autres agents.

Fait [Develay, 1993] : cf. *Notion*.

Finalités [Meirieu, 1994] : Représentations de l'homme, de la culture et de l'éducation qui président au choix des contenus didactiques (objets d'apprentissage) et des modèles pédagogiques (méthodes d'apprentissage). Les finalités sont, en ce sens, plus révélées par les pratiques mises en œuvre que par les déclarations d'intention. Les décisions sur ce qu'il convient d'apprendre et la manière de s'y prendre réfractent toujours les finalités de l'éducateur, c'est-à-dire ses conceptions de « l'homme cultivé » du sujet apprenant, des rapports entre formateurs et formés ; elles sont donc porteuses - fût-ce à son insu - d'un projet éthique et politique.

GLOM : Acronyme de « Génie Logiciel : Outils et Méthodes ». Ce terme, désigné dans le PPN (Programme Pédagogique National) des IUT, regroupe les matières d'enseignement de l'ACSI (Analyse et Conception des Systèmes d'Information) et des bases de données.

Groupe d'apprentissage [Meirieu, 1994] : Groupe de travail dans lequel le formateur s'assure de la participation de chaque membre par une distribution opportune des matériaux de travail, et de la progression de chacun d'entre eux par la mise en place d'un mode de fonctionnement groupal garantissant l'effectuation individuelle de l'opération mentale requise.

Heuristique [Weinberg, 1995] : Tout comme un algorithme, une heuristique est une méthode de résolution de problème issue de l'Intelligence Artificielle. Les méthodes « heuristiques » sont fondées sur la probabilité de réussite d'une solution donnée. Ces stratégies consomment moins de temps et d'énergie, elles sont plus « intelligentes » mais comportent une part d'incertitude. Les programmes d'intelligence artificielle sont construits sur un mode heuristique. Ainsi, un programme de jeu d'échecs ne repose pas sur un inventaire systématique de tous les coups jouables, mais analyse la valeur des possibilités de réussite d'un nombre limité de coups, exactement comme le fait le joueur humain, dont la machine ne fait que copier partiellement le mode de raisonnement.

Heuristique [Farreny et Ghallab, 1987] : C'est un moyen quelconque (un critère, une procédure, un ensemble de règles) destiné à réduire les alternatives et guider les choix non-déterministes que doit faire un algorithme de recherche. Les choix effectués grâce à une heuristique ne sont pas

nécessairement irrévocables mais si l'heuristique est efficace, on reviendra plus rarement sur ces choix. Le plus souvent une heuristique exploite une information spécifique au problème posé. Elle n'est pertinente et efficace que pour ce problème-là, et ne peut être immédiatement généralisable à d'autres problèmes. Ainsi, le même algorithme de recherche devra utiliser, suivant les problèmes, différentes heuristiques.

I.A. : Intelligence Artificielle.

I.A.D. : Intelligence Artificielle Distribuée.

IA symbolique [Weinberg, 1995] : La logique de fonctionnement de la plupart des ordinateurs actuels repose sur une « démarche symbolique » qui consiste à représenter toutes les données (images, chiffres, mots) et les règles de raisonnement sous forme de symboles abstraits (x, y, $\forall$, $\exists$, <, ==>, etc…).

I.H.M. [Selker, 1996] : Interface Homme-Machine. Elle cherche à créer des scénarios sous forme d'actions et de réponses ordinateur-utilisateur qui répondent à des objectifs de communication et de relation entre l'individu et la tâche que l'on cherche à faciliter par ordinateur. Ainsi, l'IHM se définit non par des méthodes mais par les effets que ces méthodes peuvent produire.

Indicateur [Meirieu, 1994] : Comportement observable à partir duquel on peut inférer de l'atteinte d'un objectif ou de la maîtrise d'une capacité. Il n'y a pas de déductibilité des indicateurs à partir de l'objectif ou de la capacité et ceux-ci ne sont pas réductibles à la somme des indicateurs qui peuvent permettre d'en inférer l'existence. En ce sens, le choix des indicateurs et leur pertinence sont toujours questionnables.

Indicateurs de correspondance [Meirieu, 1994] : Eléments caractérisant une situation d'exécution d'une tâche et permettant au sujet de repérer si ses stratégies cognitives sont ou non en phase avec les règles de conduites - implicites ou explicites - qui sont proposées. L'efficacité intellectuelle consiste alors, en cas de distorsion, à inventer des stratégies différenciées de « tuilage » progressif entre la personnalité cognitive du sujet et les règles de conduite dictées par la situation.

Indicateurs de réussite [Meirieu, 1994] : Sélection de quelques critères de réussite d'une tâche suffisamment significatifs pour en avoir une représentation minimale permettant d'en engager la réalisation. La liste de ces indicateurs peut être complétée, par des pauses méthodologiques, au cours de la réalisation de la tâche.

Indicateurs de structure [Meirieu, 1994] : Indicateurs dont la présence permet d'identifier une *classe de problèmes* à laquelle s'adapte un p*rogramme de traitement* spécifique. Contrairement aux *indicateurs de surface*, les indicateurs de structure sont rarement énoncés dans l'intitulé scolaire d'un exercice ou d'un problème ; leur appréhension nécessite donc un apprentissage et celui-ci s'effectue par l'appropriation interactive d'une classe de problèmes et d'un programme de traitement, la première ne pouvant être appréhendée que par l'intelligence du second et le second n'ayant de sens que par l'identification de la première.

Indicateurs de surface [Meirieu, 1994] : Indicateurs afférents aux matériaux utilisés et au cadre spécifique dans lequel se pose un problème et qui ne sont pas constitutifs de la structure de la classe à laquelle appartient le problème. La *décontextualisation* consiste, pour une large part, à retrouver, dans des situations présentant des indicateurs de surface différents, les mêmes *indicateurs de structure*.

Induction [Weinberg, 1995] : Elle consiste à établir une loi générale à partir de cas particuliers. Par exemple, si j'observe régulièrement que les feuilles tombent à l'automne, je peux en tirer par induction que « chaque automne, les feuilles tombent ». Bien entendu, cette loi n'est pas démontrée. Elle est tirée de l'expérience, puis généralisée.

Induction (pensée inductive) [Meirieu, 1994] : Opération mentale par laquelle un sujet confronte des éléments pour en faire émerger le point commun. L'induction peut être réalisée à différents niveaux et concerner le regroupement sur une caractéristique commune (classes d'objets ayant un élément ou une fonction en commun), sur une relation commune (spatiale, temporelle, analogique, sémantique...) ou sur une structure abstraite commune (élaboration conceptuelle proprement dite).

Information [Prince, 1996] : L'information est un signifié transporté par une donnée. Ce signifié est dépendant des procédures de décodage des données (c'est-à-dire d'extraction de l'information) et de l'univers cognitif des agents qui mettent en oeuvre des procédures. C'est le résultat final de la transformation (fort complexe) de la donnée. L'information est là où commence le sens, et le sens ne peut s'opérer sans connaissances.

Interface graphique : Affichage sous forme graphique (icones, menus, boutons, …) de commandes accessibles à l'utilisateur.

Internet : Réseau reliant plusieurs millions d'utilisateurs d'ordinateurs dans le monde. On y échange du courrier électronique (e-mail), des informations et des opinions (newsgroups), on y consulte des banques d'informations (Gopher, Wais, *WWW*).

Invariants structurels (dans un apprentissage) [Meirieu, 1994] : Eléments fixes qui, pour effectuer un apprentissage précis, ne peuvent être contournés, quel que soit le sujet apprenant. Ces invariants peuvent être décrits en termes de contenus de connaissance (programme), d'activités à effectuer (progression taxonomique) ou d'*opérations mentales* à réaliser (situation articulant des dispositifs adaptes). Les mêmes invariants structurels devront être négociés par chacun des sujets selon des *variables-sujet* qui lui sont propres.

J.I.T.O.L. : Acronyme de « *Just-In-Time Open Learning* » = « Juste ce qu'il faut, juste à temps ».

KADS : Acronyme de « *Knowledge Acquisition Design Systems* ». Méthode d'acquisition des connaissances issue du projet européen ESPRIT et qui a donné lieu à une automatisation sous le nom de système SHELLEY [Anjewierden *et al.*, 1992].

L.P.P.O. : Logique des Prédicats du Premier Ordre.

Logique floue [Weinberg, 1995] : La notion de sous-sensemble flou (« *fuzzy set* ») a été introduite à Berkeley dans les années 60 par le Professeur Lofti A. ZADEH. Dans la logique des ensembles, un élément *x* appartient ou n'appartient pas à un ensemble A. Dans la logique floue, un élément *x* peut être affecté d'un degré d'appartenance à A pouvant varier de 0 (appartient) à 1 (n'appartient pas). Avec la logique floue, on entre dans le monde du possible au lieu du certain, du vraisemblable au lieu du vrai.

M.V.C. [Krasner et Pope, 1988] : Modèle Vue Contrôleur.

M.PV.C. [Krief, 1992] : Modèle Point de Vue Contrôleur.

Matériaux [Meirieu, 1994] : Ensemble de documents, outils, ressources fournis par le formateur dans une situation didactique et qui seront mis en œuvre dans le dispositif proposé. La maîtrise préalable des matériaux doit être suffisamment assurée pour permettre d'engager l'activité ; elle requiert donc une évaluation diagnostique des compétences des sujets.

Matrice disciplinaire [Develay, 1993] : Elle représente le principe d'intelligibilité d'une *discipline* donnée c'est-à-dire son cadre de référence. Le sens métaphorique de matrice (nom commun de l'utérus) renvoie à l'image de moule, de creuset qui constituerait le fondement de la discipline, son essence. Une matrice disciplinaire est constituée par le point de vue qui, à un moment donné, est porté

sur un contenu disciplinaire et en permet la mise en cohérence. Ce point de vue est constitué par le choix d'une identité pour la discipline considérée. Il entraîne à privilégier, de fait, certains concepts, certaines méthodes, techniques, théories, valeurs, et amène en dernier ressort à valoriser certains objets d'enseignement. Nous remarquons donc que les matrices disciplinaires ne sont pas les mêmes aux différents niveaux de la scolarité.

Médiation [Meirieu, 1994] : Désigne à la fois ce qui, dans le rapport pédagogique, relie le sujet au savoir et sépare le sujet de la situation d'acquisition. Elle assure ainsi, contradictoirement mais indissolublement, la transmission du savoir et l'émancipation du sujet. Constituant un point fixe par rapport auquel apprenant et formateur se « mettent en jeu », elle est aussi ce grâce à quoi ils se « dégagent ». Des institutions, des règles, des objets, des méthodes peuvent constituer des médiations.

Mentalisation [Meirieu, 1994] : Opération par laquelle un sujet se représente une acquisition en l'absence de tout élément matériel ayant servi ou pouvant servir à son acquisition.

Mentalisme [Meirieu, 1994] : Conception qui considère que l'activité mentale de l'apprenant ne peut être « traduite » en comportements observables. La pédagogie par objectifs d'inspiration mentaliste affirme donc l'existence d'une rupture entre, d'une part, les objectifs généraux (formules en termes de compétence ou de capacité) qui sont analysés pour identifier les opérations mentales requises et permettre la construction de dispositifs, et, d'autre part, les objectifs opérationnels qui perdent leur caractère de " traduction », des objectifs généraux pour devenir de simples indicateurs d'évaluation.

MétaCASE : C'est un support automatisé ou semi-automatisé d'outils CASE.

Métacognition [Meirieu, 1994] : Activité par laquelle le sujet s'interroge sur ses *stratégies d'apprentissage* et met en rapport les moyens utilisés avec les résultats obtenus ; il peut ainsi stabiliser des *procédures* dans des *processus*.

Métamodèle : C'est un modèle qui décrit d'autres modèles.

Métarègle [Farreny et Ghallab, 1987] : Ce sont des granules de connaissances opératoires susceptibles de préciser la manière d'utiliser les règles.

Méthode [Rumbaugh *et al.*, 1994] : C'est un guide générique destiné à aider les gens à effectuer des activités. Une méthode comporte plusieurs éléments : un ensemble de *concepts* dont le rôle consiste à capturer les connaissances à propos d'un problème et des solutions correspondantes ; un ensemble de

vues et de *notations* dont le rôle est de présenter les différentes connaissances de façon compréhensible par l'utilisateur ; un *processus de développement* pas à pas itératif pour construire les modèles et leurs implémentations ; et une collection inorganisée *d'indications et d'intuitions utiles*.

Méthode [Meirieu, 1994] : Terme désignant un ensemble de moyens mis en œuvre pour effectuer un apprentissage un ou plusieurs dispositifs, un traitement individuel ou interactif de ceux-ci, des matériaux et des outils, une démarche, un certain degré de guidage (directivité), etc…

Méthode [McInally, 1997] : Une *méthode* est alors une activité organisée dans un but donné. C'est une encapsulation des outils et procédures qui sont en relation avec un aspect de la méthodologie.

Méthodologie [McInally, 1997] : Une *méthodologie* est un système de méthodes et de règles utilisée dans une discipline particulière.

Modèle [Rumbaugh *et al.*, 1994] : C'est une représentation formelle d'un système à un niveau d'abstraction donné. Un modèle se construit à partir de concepts.
Modèle [Krief, 1992] : Un modèle M est un couple {D, O} où D est une représentation symbolique ou représentation figurative d'un certain domaine et O un ensemble d'outils permettant la manipulation et l'exploration de D.

Modèle logiciel [Krief, 1992] : C'est un couple {D, O} ou D est la représentation informatique des concepts du domaine traité et O un ensemble de procédures permettant la manipulation de D. Il sert de lien entre D et le monde extérieur, c'est-à-dire l'utilisateur du modèle.

Modèle pédagogique [Meirieu, 1994] : Construction théorique mobilisant une représentation du sujet apprenant et du savoir qu'il convient de lui proposer, ainsi qu'un projet éthique implicite ou explicite. Le modèle permet de sélectionner des informations et de proposer des institutions et des activités didactiques particulières. Toute pédagogie est ainsi porteuse d'un modèle qu'elle privilégie au nom de ses finalités ; toute didactique renvoie ainsi à des représentations et à des valeurs, même si elle tente de « naturaliser » le modèle qu'elle propose en occultant les choix qu'elle a effectués. La pédagogie par alternance, la « pédagogie du projet », la pédagogie des situations-problèmes, la classe en « collectif-frontal »,... sont des modèles pédagogiques.

Multimédia : Intégration sur un même support de données de différentes natures (texte, son, image fixe ou animée).

Multimodal [Coutaz, 1996] : Le multimodal, tout comme le multimédia, sollicite les capacités sensori-motrices de l'homme. Mais un système multimodal a la faculté d'abstraire donc de comprendre ce qui est reçu ou émis selon les mutliples canaux de la communication humaine. Un système mutlimédia n'a pas ces capacités. Du point de vue technique, le multimodal et le multimédia se distinguent par la capacité à modéliser la sémantique des informations échangées avec l'utilisateur.

Notion [Develay, 1993] : A un certain niveau, une notion est explicative d'un ensemble de *faits*. Pour être assimilée, une notion doit s'ancrer sur un ensemble de faits dont elle cherche la cohérence. Ainsi, ce sont deux opérations d'abstraction étroitement liées. On ne peut pas enseigner de notions (*exemple :* la respiration) indépendamment de faits (*exemple :* le poisson respire par les branchies, l'homme par les poumons). Et un fait ne prend de sens que par rapport à la notion qui l'englobe.

Notion clé : cf. *Concept intégrateur*.

Notion-noyau [Meirieu, 1994] : Elément-clé ou concept organisateur dans un ensemble de contenus disciplinaires. Les notions-noyaux - comme la respiration la colonisation, la description, la proportionnalité... - permettent de réorganiser les programmes autour de points forts et de construire des situations didactiques pour permettre leur acquisition. Elles sont toujours appréhendées à un *registre* donné *de formulation*.

O.O. : Orienté(e) Objet.

O.O.A. : Object Oriented Analysis : méthode d'analyse orientée-objet [Shlaer et Mellor, 1991].

Objectif [Meirieu, 1994] : Dans la perspective première de la « pédagogie par objectifs », l'objectif pédagogique opérationnel doit définir un comportement observable à réaliser par l'apprenant (exemple : « être capable d'identifier les pronoms relatifs dans un texte »). Progressivement, la notion d'objectif s'est dégagée de cette acception behavioriste pour désigner une habilité intellectuelle complexe invisible et dont le comportement observable n'est qu'un indicateur possible (exemple : « être capable de transformer un texte oral en texte écrit ») ; dans cette perspective, la pronominalisation et l'usage des pronoms relatifs sont des *indicateurs de réussite*). Aujourd'hui, nous employons le mot d'objectif pour désigner une *compétence* à acquérir en termes de corrélation entre une *classe de problèmes* et un *programme de traitement* ; on peut alors parler « d'objectif opératoire » (exemple : « être capable d'effectuer une contraction de texte en identifiant la nature du texte concerné et en utilisant les outils correspondants »).

Objectif-obstacle [Meirieu, 1994] : Objectif dont l'acquisition permet au sujet de franchir un palier décisif de progression en modifiant son système de représentations et en le faisant accéder à un *registre* supérieur *de formulation*.

Obstacle-objectif [Meirieu, 1994] : Difficulté émergeant dans la réalisation d'une *tâche* et qui permet d'engager un apprentissage pour la surmonter. Mais l'émergence de l'obstacle, si elle permet de finaliser la poursuite d'un objectif d'apprentissage, n'engage pas le sujet de manière automatique vers celui-ci : le sujet peut tenter de ne pas apprendre, en contournant l'obstacle, ou bien renoncer à poursuivre la tâche. Le rôle du formateur est donc ici d'empêcher la *clôture productive* pour que, dans la *tâche* qu'il propose à l'apprenant, l'objectif-obstacle qu'il a défini comme devant être objet d'acquisition devienne, pour lui, un obstacle-objectif.

Ontologie [Gruber, 1993] : Spécification formelle d'une conceptualisation. C'est une façon de représenter à la fois les connaissances et les primitives de manipulation de ces connaissances.

Opération mentale [Meirieu, 1994] : Activité intellectuelle par laquelle un sujet saisit et traite de l'information.

Outil [McInally, 1997] : Un *outil* est un artefact utilisé pour exécuter une procédure dans le cadre de la méthode.

P.A.C. [Coutaz, 1988] : Présentation Abstraction Contrôle.

Pédagogie [Meirieu, 1994] : Réflexion sur l'éducation de l'enfant et, par extension, sur l'éducation de l'adulte en tant que, chez lui, la genèse ne s'arrête pas avec la jeunesse. La pédagogie s'interroge sur les finalités à affecter à cette éducation, sur la nature des connaissances qu'elle doit contribuer à transmettre et sur les méthodes qu'elle doit utiliser. Au sein de la réflexion pédagogique, la didactique s'intéresse, plus particulièrement, à l'organisation des *situations d'apprentissage*.

Planification (d'un curriculum) [Develay, 1993] : Elle concerne la place faite aux ressources et aux contraintes en terme de matériel, de techniques, d'organisation et de temps à consacrer à l'activité.

Pratique(s) sociale(s) de référence [Meirieu, 1994] : Activités par rapport auxquelles un apprentissage prend du sens pour un sujet apprenant. Ces activités peuvent être extra-scolaires (ainsi l'arpentage peut-il être une pratique sociale de référence pour la géométrie), ou intra-scolaires (ainsi un journal scolaire peut-il être une pratique sociale de référence pour l'acquisition de l'orthographe, un

élevage de cobayes dans la classe, le devenir pour l'acquisition de notions biologiques, une situation-problème mathématique pour l'acquisition de concepts, etc…). Par ailleurs, des pratiques sociales de référence peuvent renvoyer à la fonctionnalité d'un apprentissage (on apprend à lire pour pouvoir consulter un mode d'emploi ou un magazine de télévision) ou sa place dans une dynamique imaginaire (on apprend aussi à lire pour accéder à un secret). La notion de pratique sociale de référence permet de comprendre en quoi un savoir scolaire ne peut pas simplement être compris comme transposition d'un savoir savant.

Procédural [Weinberg, 1995] : Le raisonnement « procédural » désigne l'ensemble des règles à suivre pour atteindre un objectif donné ; c'est un « savoir-faire ». Par exemple, pour construire une phrase, il faut posséder les règles de grammaire.
Procédural [Develay, 1993] : Les connaissances procédurales sont constituées par une suite organisée d'actions (méthodes, techniques, procédures, stratégies) permettant d'atteindre le but poursuivi. C'est un ensemble de méthodes et techniques permettant la création des *concepts*.

Procédure [Meirieu, 1994] : Eléments d'une *stratégie d'apprentissage* ayant été compris comme efficaces dans une situation donnée et pouvant être reproduits dans une situation du même ordre.

Procédure [McInally, 1997] : Une *procédure* est un ensemble d'instructions écrites pour une tâche spécifique et à exécuter dans le cadre de la méthode.

Processus [Meirieu, 1994] : Eléments d'une *stratégie d'apprentissage* ayant été mis en œuvre de manière conjoncturelle, en fonction de circonstances favorables. Il faut isoler, dans un processus aléatoire, une procédure efficace pour pouvoir la reproduire.

Processus de développement [Rumbaugh *et al.*, 1994] : Un processus est un guide indiquant comment produire un *modèle*. Il fournit un support pour le développement en décrivant les artefacts à produire ainsi que les étapes à respecter pour les produire.

Profil pédagogique [Meirieu, 1994] : Mode de représentation de l'activité cognitive des sujets à partir de quatre paramètres (gestion du quotidien, apprentissages mécaniques, opérations complexes, créativité) et du type d'évocations mentales (visuelles ou auditives) utilisées pour les gérer. A. de la GARANDERIE, qui a popularisé cette expression, parle d'une « langue maternelle pédagogique » relativement stable chez un sujet, pouvant être toutefois modifiée par l'acquisition d'habitudes complémentaires. La théorie de référence est ici la psychologie introspective (voir style cognitif, stratégie d'apprentissage et système de pilotage de l'apprentissage).

Programme de traitement [Meirieu, 1994] : Ensemble d'opérations mentales appliqué à une *classe de problèmes* déterminée et permettant d'obtenir la solution de chacun d'eux. Le théorème de Pythagore est ainsi un programme de traitement permettant de résoudre les problèmes appartenant à une même classe se caractérisant par quatre *indicateurs de structure* : 1) l'existence d'une situation géométrique ou d'un possible codage géométrique d'une situation, 2) l'existence d'un angle droit, 3) la possibilité de construire un triangle rectangle, 4) la possibilité de connaître la mesure de deux des trois côtés de ce triangle... C'est également, par exemple, un programme de traitement spécifique qui, dans un exercice de contraction d'un texte argumentaire, permet de distinguer les arguments et les exemples... L'apprentissage d'un programme de traitement s'effectue par élaboration progressive d'*archétypes mentaux* à partir de la confrontation de situations prototypiques.

Projet [Meirieu, 1994] : Dans le registre didactique ce terme désigne d'abord l'attitude du sujet-apprenant par laquelle il se trouve en situation active de recueil et d'intégration d'informations ; les informations ainsi intégrées et mentalisées peuvent être considérées comme des connaissances. Par extension, ce terme peut designer la tâche qui finalise les activités de recueil d'informations du sujet.

Prototypage [Krief, 1992] : C'est un processus de construction de logiciels ayant pour but l'obtention d'informations sur l'adéquation et la pertinence de la conception d'un logiciel par le concepteur. Le prototype est habituellement utilisé comme un précurseur à l'écriture d'un système informatique. Un prototype se distingue typiquement du produit final par le fait qu'il soit plus rapidement développé, plus facilement paramètrable et manipulable au détriment de l'efficacité et de la performance. Le prototypage est donc utilisé pour extraire rapidement des informations sur le logiciel à venir.

Prototype [Krief, 1992] : Un prototype P est un modèle expérimental, c'est-à-dire un modèle du modèle M. En conséquence, P = {D, O} avec D_p une représentation de M et O_p un ensemble d'outils spécialisés pour la construction et la manipulation de D_p.

Prototype logiciel [Krief, 1992] : C'est un logiciel expérimental P, c'est-à-dire un modèle d'un logiciel L. En conséquence, P = {Dp, Op} avec Dp une représentation de L et Op est un ensemble d'outils spécialisés pour la construction et la manipulation de Dp.

Prototype mental [Meirieu, 1994] : Ensemble de *procédures* (*algorithme*) ou d'*opérations mentales* construit dans une *situation* donnée face à un objet déterminé. Il faut identifier la nature de ce prototype, la structure de l'objet et le projet d'appropriation constitutif de la situation pour que ce prototype puisse fonctionner comme archétype et être expérimenté dans d'autres circonstances du

même ordre.

Psychologie différentielle [Develay, 1993] : Elle s'intéresse aux variations inter-individuelles à rechercher des noyaux de cohérence dans la conduite des individus, montrant des stabilités quant à leur manière d'agir dans des situations diverses, à des distances temporelles plus ou moins étendues.

Registre de conceptualisation [Develay, 1993] : Il permet de récapituler le champ notionnel des différentes *notions* en s'adaptant au niveau des élèves. Ceci permet également aux enseignants de mieux identifier les niveaux d'exigence et de mettre en place des *stratégies* didactiques appropriées.

Registre de formulation [Meirieu, 1994] : système de *représentations* d'un phénomène situé à un certain niveau d'abstraction ou de modélisation. Le passage d'un registre de formulation à un autre s'effectue par l'identification et l'acquisition de l'*objectif-obstacle*.
Registre de formulation [Develay, 1993] : Tout comme les *connaissances déclaratives* qui possèdent un *registre de conceptualisation*, les *compétences procédurales* à développer dans une discipline peuvent être regardées en termes de *registres de formulation*.

Répertoire cognitif [Meirieu, 1994] : Mémoire de travail constituée d'*indicateurs de réussite* corrélés à des types de *tâche*, d'indicateurs de structure de classes de problèmes corrélés à des *programmes de traitement* et d'*indicateurs de correspondance* corrélés à des *stratégies* personnelles efficaces.

Représentation [Meirieu, 1994] : Dans le domaine de l'apprentissage, désigne la conception que le sujet a, à un moment donné, d'un objet ou d'un phénomène. Si l'on retient l'hypothèse piagetienne qui fait de l'accès à l'abstraction le vecteur central de la construction de l'intelligence, on peut considérer que l'apprentissage consiste à passer d'une représentation de type métaphorique à une représentation de plus en plus conceptualisée. Par ailleurs, les représentations qu'un sujet se fait, à un moment donné, de plusieurs types de « réalités », appartenant même à des disciplines différentes, sont vraisemblablement articulées autour de principes explicatifs communs ou paradigmes.

Réseaux de neurones [Weinberg, 1995] : Les modèle connexionnistes ou modèles « réseaux de neurones » correspondent à des modes de fonctionnement inspirés de l'organisation des cellules du cerveau. Selon ces modèles, les processus mentaux sont envisagés comme un réseau de micro-unités qui traitent en parallèle et simultanément des informations partielles. Ces modèles, qui s'opposent au schéma « computationnel », envisagent le processus mental comme une suite d'opérations successives. Ils semblent plus adaptés pour reproduire des mécanismes tels que la perception.

S.M.A. : Système Multi-Agents

S.T.I. : Système Tuteur Intelligent.

S.T.I.C [Lima, 1995] : Système de Transfert Intelligent des Connaissances. Terme générique utilisé pour les EIAO relatifs à des domaines autres que l'éducation où il existe un transfert de connaissances entre une machine et un humain.

Savoir-faire [Prince, 1996] : Le savoir-faire, pour un agent cognitif naturel ou artificiel, est un mode d'optimisation de l'attribution de sens à des données (en acquisition ou en génération), « court-circuitant » le passage par les connaissances nécessaires qui auraient abouti au même résultat. Contrairement aux connaissances, le savoir-faire n'obéit pas à la loi de compositionalité et n'est pas auto-discursif (savoir le décrire ne revient pas à savoir l'appliquer).

Scénario : Ensemble de situations didactiques. C'est l'organisation des stratégies d'enseignement et d'apprentissage.

Sémantique [Farreny et Ghallab, 1987] : Elle traite du sens attribué aux expressions d'un langage selon des *interprétations* particulières : la sémantique régit la manière de comprendre les expressions du langage en vue d'applications déterminées.

Situation : Elle équivaut à la notion de contexte. C'est-à-dire qu'en fonction des *objectifs* à atteindre, la situation déclenche l'activation ou non de *tâches*.

Situation d'apprentissage [Meirieu, 1994] : Situation (ensemble de dispositifs) dans laquelle un sujet s'approprie de l'information à partir du *projet* qu'il conçoit. Il s'appuie, pour ce faire, sur des *capacités* et des *compétences* déjà maîtrisées qui lui permettent d'en acquérir de nouvelles. Les situations d'apprentissage peuvent ainsi apparaître en dehors de toute structure scolaire et de toute programmation didactique. Contrairement aux *situations d'enseignement*, il n'y a ici aucune action du pédagogue.

Situation d'apprentissage [Develay, 1993] : Elle rappelle une situation proche, réussie antérieurement. Il devient alors possible d'anticiper les *critères de réussite* de la *tâche*. La situation d'apprentissage qu'un enseignant propose, recèle, selon lui, des *objectifs* qui correspondent aux *obstacles* qu'il présuppose que ses élèves auront à surmonter. L'obstacle présent pour l'élève, dans la situation proposée par le maître, est le déclencheur espéré de son apprentissage.

Situation d'enseignement : Mise en place de situations-problèmes permettant d'atteindre les objectifs d'apprentissage.

Situation didactique [Meirieu, 1994] : Situation d'apprentissage élaborée par le didacticien qui fournit, d'une part, des *matériaux* permettant de recueillir l'information et, d'autre part, une *consigne-but* permettant de mettre le sujet en situation de *projet*. Une évaluation diagnostique dans le champ socio-affectif permet de s'assurer que la consigne-but est effectivement susceptible de mobiliser le sujet. Une évaluation diagnostique dans le champ cognitif permet de s'assurer que le sujet dispose bien des *capacités* et *compétences* lui permettant de traiter l'information. La situation ainsi didactisée permet de faire échapper l'apprentissage à l'aléatoire de rencontres et concordances fortuites.

Situation-problème [Meirieu, 1994] : Situation didactique dans laquelle il est proposé au sujet une *tâche* qu'il ne peut mener à bien sans effectuer un apprentissage précis. Cet apprentissage qui constitue le véritable *objectif* de la situation-problème, s'effectue en levant l'*obstacle* à la réalisation de la *tâche*. Ainsi la production impose l'acquisition, l'une et l'autre devant faire l'objet d'évaluations distinctes. Comme toute *situation didactique*, la situation-problème doit être construite en s'appuyant sur une triple évaluation diagnostique (des motivations, des *compétences* et des *capacités*).
Situation-problème [Develay, 1993] : C'est donc un cas particulier d'une situation plus générale pour laquelle on connaît à la fois une procédure de solution et les propriétés relationnelles qui permettent de justifier cette procédure.

Stratégie [Pierre *et al.*, 1992] : Une *stratégie* est une manière dont l'enseignant ou les élèves vont agencer les *ressources*, utiliser les *contraintes* ou y résister. Donc en définitive la manière d'organiser les différents composants définit la stratégie pédagogique.

Stratégie d'apprentissage [Meirieu, 1994] : Mode de *représentation* de l'activité cognitive des sujets à partir de la description des comportements intellectuels efficaces dans des situations didactiques précises. La stratégie d'un sujet s'articule ainsi à un style cognitif personnel relativement stable mais dépend aussi de l'objet de l'apprentissage. On peut distinguer, dans une stratégie, cinq types de variables : les outils (plutôt visuels ou plutôt auditifs...), la démarche (plutôt globale ou plutôt analytique...), le degré de guidage (directivité), l'insertion socio-affective (usage plus ou moins poussé de l'interaction sociale...), la gestion du temps. La théorie de référence est ici la didactique (voir profil pédagogique, style cognitif et système de pilotage de l'apprentissage).

Stratégie d'enseignement : C'est une des façons d'agencer, d'ordonnancer, les différents objectifs

d'apprentissage que le pédagogue fixe pour l'apprenant.

Structuration [Meirieu, 1994] : Réorganisation de la cohérence épistémologique de connaissances ou de concepts découverts à l'occasion d'apprentissages spontanés ou didactiques. La rationalité notionnelle apparaît ainsi au terme du processus d'apprentissage, comme une mise en perspective des résultats obtenus et non comme la démarche de leur appropriation.

Style cognitif [Meirieu, 1994] : Mode de représentation de l'activité cognitive des sujets à partir de *variables-sujet* relativement stables, indépendantes des situations didactiques mises en place et des stimulations de l'environnement. Les styles cognitifs les plus utilisés sont « la dépendance » et « l'indépendance par rapport au champ ». La théorie de référence est ici la psychologie expérimentale (voir profil pédagogique. stratégie d'apprentissage et système de pilotage de l'apprentissage).

Syntaxe [Farreny et Ghallab, 1987] : La syntaxe d'un langage permet de spécifier la manière de construire des expressions correctes dans ce langage.

Synthèse [Weinberg, 1995] : La synthèse est le contraire de l'analyse. Elle consiste à rassembler des propositions séparées en une proposition unique. Depuis HEGEL, l'art de la synthèse est de formuler deux thèses contradictoires : « Les sciences humaines sont ennuyeuses » (1) et « Les sciences humaines sont passionnantes » (2). Une synthèse est que les thèses précédentes ne sont que partiellement vraies.

Système expert [Farreny et Ghallab, 1987] : Selon Edward FEIGENBAUM de l'Université de Stanford (USA), « les systèmes experts sont des programmes conçus pour raisonner habilement à propos de tâches dont on pense qu'elles requièrent une expertise humaine considérable » (définition assez peu éclairante). Fondamentalement, un système expert est un moteur d'inférence (programme d'exploitation des connaissances) relativement général exploitant une collection séparée, sujette à évolutions, d'unités de savoirs-faire concernant un domaine particulier d'expertise humaine.

Système personnel de pilotage de l'apprentissage (S.P.P.A.) [Meirieu, 1994] : Mode de représentation de l'activité cognitive des sujets à partir des outils fournis par l'approche systémique. Le S.P.P.A. désigne la manière dont la personne perçoit, stocke et communique l'information. Les théoriciens de cette approche insistent sur le fait que les sujets peuvent être répartis selon une double polarité : ceux qui apprennent plutôt « par production » (vérification et reconstruction), et ceux qui apprennent plutôt « par consommation » (intériorisation et compréhension). La concordance entre le système de pilotage de l'apprentissage du formé et le système de pilotage de l'enseignement du

formateur garantit, pour eux, l'efficacité de la *situation didactique* (voir profil pédagogique, style cognitif et stratégie d'apprentissage).

Système de représentations [Meirieu, 1994] : Ensemble de conceptions (images, métaphores, modèles) permettant d'organiser les données de la perception et prétendant rendre compte du « réel ». Un sujet dispose toujours d'un système de représentations d'une réalité ; ce dernier peut se situer à un plus ou moins grand niveau d'abstraction (dans un champ disciplinaire on parlera, pour designer ce niveau. de *registre de formulation*).

Tableau de suggestions et de remédiations [Meirieu, 1994] : Ensemble de propositions fournies à un apprenant qui se trouve confronté à un exercice ou à une *situation-problème*. Ces propositions suggèrent différents itinéraires (« on peut commencer par... ou par..., utiliser tel ou tel outil... ») et renvoient à des exercices de remédiation précis en fonction des difficultés rencontrées (« si vous ne parvenez pas à... revoyez l'exercice... ou faites appel à... »). Un tableau de suggestions et de remédiations peut être élaboré avec les apprenants. Cette élaboration est éminemment formatrice et constitue un excellent exercice de *métacognition*.

Tâche [Meirieu, 1994] : Entendue au sens large, la tâche est l'objet auquel doit aboutir l'activité du sujet. En situation de formation, il existe des tâches spécifiquement scolaires (un commentaire de carte, la récitation d'une poésie, un compte-rendu d'expérience, etc…) et des tâches qui renvoient à des *pratiques sociales de référence* extra-scolaires (un article pour le journal de la classe, une maquette d'une ville, la mise à jour du livre de comptes de la coopérative scolaire, etc…). Pour engager efficacement l'activité d'un sujet, la tâche doit faire l'objet d'une représentation relativement précise ; en d'autres termes, la *consigne-but* (définition de la tâche) doit être accompagnée de *consignes-critères* : ces dernières doivent être élaborées en prenant en compte la réussite fonctionnelle de la tâche (liées à l'efficacité de l'objet) et la réussite académique de la tâche (liées aux exigences propres au " contrat didactique » entre le formateur et l'apprenant).

Tâche [Develay, 1993] : En situation d'enseignement, nous parlerons de *tâche* et non pas de problème car la même tâche proposée à un groupe d'élèves peut correspondre, pour chacun d'eux, à des activités très différentes, en fonction des problèmes qu'ils rencontrent. LEONTIEV définit une tâche comme « un but donné dans des conditions déterminées ». Chaque discipline peut donc être caractérisée par des tâches qui constituent les *activités* que les élèves doivent réussir

Trame notionnelle : cf. *Champ notionnel*.

Transdisciplinarité [Develay, 1993] : Dans cette conception, une discipline d'enseignement est

envisagée comme un ensemble de connaissances procédurales et de notions en réseaux. Mais, dans ce cas, on s'intéresse aux compétences notionnelles et méthodologiques qui pourraient être communes à plusieurs disciplines. La discipline considérée est resituée par rapport à d'autres disciplines, afin de pouvoir déterminer deux types de transdisciplinarité : la transdisciplinarité instrumentale qui s'intéresserait aux contenus notionnels et la transdisciplinarité comportementale qui s'intéresserait aux connaissances procédurales et aux attitudes.

Transposition didactique [Develay, 1993] : Elle représente la structure des connaissances perçue et maîtrisée par l'enseignant par rapport aux savoirs et savoir-faire de référence. La transposition didactique correspond en dernier ressort à un travail de réorganisation, de présentation, de genèse de connaissances pré-existantes en vue de leur enseignement. Elle s'exprime en terme de tâches, objets, connaissances procédurales et déclaratives.

Variables-sujet [Meirieu, 1994] : Eléments qui, pour effectuer un même apprentissage, diffèrent d'un sujet à un autre et lui permettent de négocier les mêmes *invariants structurels* de manière spécifique. Ces variables-sujet peuvent être décrites avec des *modèles* différents, suivant la théorie de référence utilisée : on parlera, selon les cas, de *profil pédagogique*, de *style cognitif*, de *système de pilotage* ou de *stratégie d'apprentissage*.

W.W.W. : W3 : World Wild Web : C'est la « toile d'araignée mondiale » qui représente le versant multimédia d'Internet.

WYSIWIG : « *What You See Is What You Get* »

Références Bibliographiques

[Agha *et al.*, 1993]
« *A Foundation for Actor Computation* »
G. AGHA, I.A. MASON, S. SMITH, C. TALCOTT
Technical Report
Juillet 1993

[Anderson, 1988]
« The expert module »
J.R. ANDERSON
Polson and Richardson Eds, pp. 21-53
1988

[Ang et Hong, 1994]
« AMS Formalism : An Approach to Office Modeling and OIS Development »
J.S.K. ANG, J. HONG
Data Base, Vol. 25, n°4, pp. 25-38
Novembre 1994

[Anjewierden, 1986]
« How about a Prolog Object ? »
A. ANJEWIERDEN
Draft Paper, Dept. of Computing
Université d'Amsterdam, Pays-Bas
1986

[Anjewierden *et al.*, 1992]
« *Shelley - computer aided knowledge engineering* »
A. ANJEWIERDEN, J. WIELEMAKER, C. TOUSSAINT
Knowledge Acquisition, 4(1), Special issue : « The KADS approach to knowledge engineering »
1992

[Arapis, 1990]
« Specifying Object Life-Cycles »
C. ARAPIS
Centre Universitaire d'Informatique, Université de Genève
Edité par Dennis Tscichritzis et auteurs
Juillet 1990

[Arditi et Ducasse, 1996]
« *La programmation : une approche fonctionnelle et récursive avec Scheme* »
L. ARDITI, S. DUCASSE
Editions Eyrolles, Paris
1996

[Armstrong *et al.*, 1995]
« WebWatcher : A Learning Apprentice for the World Wide Web »
R. ARMSTRONG, D. FREITAG, T. JOACHIMS, T. MITCHELL
AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments
Mars 1995

[Aubord et Ibrahim, 1991]
« Des intentions des concepteurs de didacticiels »
A. AUBORD, B. IBRAHIM
13èmes journées francophones sur l'informatique
Formation Intelligemment Assistée par Ordinateurs, pp. 181-195
1991

[Aussenac, 1989]
« *Conception d'une méthodologie et d'un outil d'acquisition de connaissances expertes* »
N. AUSSENAC
Thèse de Doctorat en Informatique
Université Paul Sabatier, Toulouse III
1989

[Avison et Fitzgerald, 1995]
« Information Systems Development : Methodologies, Techniques and Tools »
D.E. AVISON, G. FITZGERALD
McGrawhill, London
1995

[Ayala et Yano, 1996]
« Intelligents Agents to Support the Effective Collaboration in CSCL Environment »
G. AYALA, Y. YANO
Proceedings of ED-TELECOM 96, pp. 19-24
Juin 1996

[Balacheff, 1994]
« Didactique et Intelligence Artificielle »
N. BALACHEFF
Article extrait du livre « Recherches en Didactique des Mathématiques »
Editions La Pensée Sauvage, vol. 14, n° 1.2, pp. 11-42
1994

[Balacheff *et al.*, 1993]
« *EIAO : point de vue des disciplines* »
N. BALACHEFF, G.L. BARON, P. DILLENBOURG, M. GRANDBASTIEN, R. GRAS, F. MADAULE, P. MENDELSOHN, A. NGUYEN-XUAN, J.F. NICAUD
Journées EIAO de Cachan, AFCET / AFIA / AI-ED / ARC / ARDM
Editions Eyrolles, Tome 1, pp. 7-14
Février 1993

[Bertrand, 1993]
« *Théories contemporaines de l'éducation* »
Y. BERTRAND
Collection « Synthèse », Chronique Sociale, Lyon
1993

[Berzins et Luqi, 1990]
« Sotware Engineering with Abstractions »
V. BERZINS, LUQI
Addison-Wesley Publishing Company
1990

[Beyou, 1992]
« Vers un système d'enseignement du dépannage intégrant des connaissances évolutives »
C. BEYOU
Communication au 7ème symposium canadien sur les technologies pédagogiques
Montréal, Canada
1992

[Bézivin, 1995]
« Technologie objet et ingénierie des besoins : une réconciliation nécessaire »
J. BEZIVIN
Revue « L'OBJET : Logiciel, bases de données, réseaux »
Vol. 1, n°1, pp. 21-26
Printemps 1995

[Biebow et Szulman, 1991]
« Validation de cahiers des charges rédigés en langage naturel »
B. BIEBOW, S. SZULMAN
Génie Logiciel & Systèmes Experts, n° 23, pp. 58-67
Juin 1991

[Biema *et al.*, 1990]
« The constraint-based paradigm : Integrating object-oriented and rule-based programming »
M.V. BIEMA, G.Q. MAQUIRE, S. STOFLO
23rd Annual Hawaï International Conference on Systems Sci.
Kaila-Kona, Los Alamitos, CA
IEEE Company Society Press, Vol. 2, pp. 358-366
Janvier 1990

[Boehm, 1976]
« Software Engineering »
B. BOEHM
IEEE Transactions on Computers, C25(12), pp. 1226-1241
1976

[Boehm, 1988]
« A spiral Model of Software Development and Enhancement »
B. BOEHM
IEEE Computer, 12(5), pp. 61-72
1988

[Boehm, 1996]
« Identifying Quality-Requirement Conflicts »
B. BOEHM
IEEE Software, pp. 25-35
Mars 1996

[Booch, 1994]
« *Object-Oriented Analysis and Design with Applications* »
G. BOOCH
Benjamin Cummings, Redwood City, 2nd Edition
1994

[Boose, 1989]
« *A survey of knowledge acquisition techniques and tools* »
J. BOOSE
Knowledge Acquisition, 1(1)
1989

[Borgida *et al.*, 1985]
« Knowledge Representation as the Basis for Requirements Specifications »
A. BORGIDA, S. GREENSPAN, J. MYLOPOULOS
Computer, pp 82-91
April 1985

[Brandeis et Kertesz, 1988]
« Le livre d'Hypercard »
P. BRANDEIS, J. KERTESZ
Editions « P.S.I. »
1988

[Bruillard et Vivet, 1994]
« Concevoir des EIAO pour des situations scolaires - Approche méthodologique »
E. BRUILLARD, M. VIVET
Article extrait du livre « Recherches en Didactique des Mathématiques »
Vol. 14, n° 1.2, pp. 275-304
Editions « La Pensée Sauvage »
1994

[Brunet et Ermine, 1994]
« Problématique de la gestion des connaissances des organisations »
E. BRUNET, J.-L. ERMINE
Article extrait de la revue « Ingénierie des systèmes d'information »
Vol. 2 n° 3, pp. 263-291
Editions « Hermes »
1994

[Buchanan et Sutherland, 1984]
« Rule-based expert systems : the MYCIN esperiments of the Stanford Heuristic Programming Project »
B.G. BUCHANAN, G.L. SUTHERLAND
Addison-Wesley
1984

[Buhr, 1984]
« *System Design with Ada* »
R.J.A. BUHR
Prentice Hall, Englewood Cliffs, New-Jersey
1984

[Burton *et al.*, 1988]
« *Knowledge Elicitation Techniques in Classification Domains* »
A.M. BURTON, N.R. SHADBOLT, G. RUGG, A.P. HEDGECOCK
Proceedings of the ECAI-88 : The 8th European Conference on Artificial Intelligence
1988

[Carey et Mason, 1983]
« Information system prototyping: techniques, tools and methodologies »
T.T. CAREY, R.E.A. MASON
INFOR, 21, #3, pp. 177-191
Août 1983

[Carroll *et al.*, 1988]
« *The Minimal Manual, Human-Computer Interaction* »
J.M. CARROLL, P.L. SMITH-KERKER, J.R. FORD, S. MAZUR
n°3, pp. 132-153
1988

[Cerri, 1996]
« *Cognitive Environments in the STROBE Model* »
S.A. CERRI
Actes du congrès Euro-AIED'96
European Conference on Artificial Intelligence in Education, Lisbonne, Portugal
1996

[Chandrasekharan, 1987]
« *Towards a Functional Architecture for Intelligence based on Generic Information Processing Tasks* »
B. CHANDRASEKHARAN
Actes du 10ème IJCAI, Milan, Italie, pp. 1183-1192
1987

[Checkland et Scholes, 1990]
« *SoftSystems Methodology in Action* »
P. CHECKLAND, J. SCHOLES
John WILEY & Sons Ltd.
1990

[Chevallard et Joshua, 1985]
« *La transposition didactique : du savoir savant au savoir enseigné.* »
Y. CHEVALLARD, M.A. JOSHUA
La Pensée Sauvage, Grenoble
1985

[Chikayama, 1984]
« Unique features of ESP »
T. CHIKAYAMA
1984 Int'l Conference on 5th Generation Computer Systems, Amsterdam, pp. 292-298
Novembre 1984

[Choi et Minoura, 1994]
« *User Interface System Based on Active Objects* »
S. CHOI, T. MINOURA
ACM Ada Letters, Special Fall 1994, Volume 14, pp. 16-25
1994

[Choppy, 1988]
« Maquettage et prototypage : panorama des outils et techniques »
C. CHOPPY
Génie Logiciel & Systèmes Experts, n° 11, pp. 6-13
Mars 1988

[Coutaz, 1988]
« Interface Homme-Machine : Conception et réalisation »
J. COUTAZ
Thèse de Doctorat de l' Université Joseph Fourier, Grenoble, Spécialité Informatique
1988

[Coutaz, 1996]
« L'art de communiquer à plusieurs voies »
J. COUTAZ
Revue « La Recherche », n° 285, pp. 38-43
Mars 1996

[Davis, 1990]
« Software Requirements, Analysis and Specification »
DAVIS
Prentice-Hall, Englewood Cliffs, N.J.
1990

[Davison, 1991]
« Design Issues for Logic Programming-based Object Oriented Languages »
A. DAVISON
The PARLOG Group, Dept. of Computing, Imperial College, London SW7 2BZ
Rapport Technique
Mai 1991

[De Haan *et al.*, 1991]
« *Formal Modelling Techniques in Human-Computer Interaction*»
G. DE HAAN, G.C. VAN DER VEER, J.C. VAN VLIET
Acta Psychologica 78, pp. 26-76
1991

[De Marco, 1979]
« Structured Analysis and System Specifications »
T. DE MARCO
Yourdon Press Computing Series
Prentice-Hall Company, Englewood Cliffs, NJ
1979

[Develay, 1993]
« De l'apprentissage à l'enseignement
Pour une épistémologie scolaire »
M. DEVELAY
Collection Pédagogies
ESF éditeur, Paris, 2ème édition
1993

[Digitalk, 1988]
« Prolog/V documentation with Smalltalk/V mac »
DIGITALK INC.
1988

[Donzeau Gouge *et al.*, 1980]
« Programming Environments Based on Structured Editors : the MENTOR Experience »
V. DONZEAU GOUGE, G. HUET, G. KAHN, B. LANC
Rapport de Recherche n°26, INRIA, Rocquencourt,
Juillet 1980

[Edelson *et al.*, 1996]
« The Collaboratory Notebook »
D.C. EDELSON, R.D. PEA, L.M. GOMEZ
Communications of the ACM, Vo. 39, n° 4, pp. 32-33
Avril 1996

[Eden *et al.*, 1996]
« Making Learning a Part of Life »
H. EDEN, M. EISENBERG, G. FISCHER, A. REPENNING
Communications of the ACM, Vol. 39, n° 4, pp. 40-42
Avril 1996

[Ehrich et Mahr, 1985]
« Fundamental of Algebraic Specification I: Equations and Initial Semantics »
H. EHRIGH, B. MAHR
Springer-Verlag, Berlin
1985

[Ehrich *et al.*, 1988]
« Towards an algebraic semantics for database specification »
H.D. EHRICH, K. DROSTEN, M. GOGOLLA
Proceedings of the 2nd IFIP WG 2.6 Working Conference on Database Semantics « Data and Knowledge »
R. A. Meersman, A. Sernadas, Eds. North-Holland, Amsterdam, pp. 119-135
1988

[Einsenstadt *et al.*, 1990]
« Visual Knowledge Engineering »
M. EINSENSTADT, J. DOMINGUE, T. RAJAN, E. MOTTA
IEEE Transactions on Software Engineering
Vol. 16, n°10, pp. 1164-1177
Octobre 1990

[Emerson, 1990]
« Temporal and modal logic »
E.A. EMERSON
Formal Models and Semantics, J. Van Leeuwen, ED. Elsevier Science Publishers B.V, Amsterdam, pp. 995-1072
1990

[Engels *et al.*, 1992]
« Conceptual modelling of database applications using an extended ER model »
G. ENGELS, M. GOGOLLA, U. HOHENSTEIN, K. HÜLSMANN, P. LÖHR-RICHTER, G. SAAKE, H.D. EHRICH
Data Knowledge Engineering 9, 2, pp. 157-204
1992

[Eriksen et Stage, 1997]
« *Do CASE Tools Support Method Learning ? »*
L.B. ERIKSEN, J. STAGE
5th BCS ISM conference, Preston
Springer-Verlag Ltd
1997

[Eriksson et Musen, 1993]
« *Metatools for knowledge acquisition* »
H. ERIKSSON, M. MUSEN
Vol. 10, n° 3, pp. 23-29
1993

[Ernst et Newell, 1969]
« GPS : A Case Study in Generality and Problem Solving »
G.W. ERNST, A. NEWELL
Academic Press
1969

[Fade, 1980]
« ARGOS-II : contribution à la réalisation d'un résolveur automatique de problèmes »
B. FADE
Thèse de 3ème cycle, Université Paul Sabatier, Toulouse
Avril 1980

[Farreny et Ghallab, 1987]
« *Eléments d'intelligence artificielle* »
H. FARRENY, M. GHALLAB
Traité des Nouvelles Technologies
Série Intelligence Artificielle
Editions HERMES

1987

[Faure, 1996]
« Et l'ordinateur à stylo ? Peut mieux faire ! »
C. FAURE
Revue « La Recherche » n° 285, pp. 86-89
Mars 1996

[Ferber, 1989]
« *Objets et agents : une étude des structures de représentation et de communications en Intelligence Artificielle* »
J. FERBER
Thèse de Doctorat d'Etat des Mathématiques, spécialité Informatique,
Université Pierre et Marie Curie - Paris 6
Juin 1989

[Ferber, 1995]
« *Les systèmes multi-agents. Vers une intelligence collective* »
J. FERBER
InterEditions, Paris
1995

[Findeisen, 1994]
« *The EARA Model for Metaview. A Référence* »
P. FINDEISEN
Department of Computing Science, University of Alberta
Juin 1994

[Floyd, 1967]
« Assigning Meaning to Programs »
R.W. FLOYD
Proc. Arner. Math Soc. Symp. in Applied Mathemancs, 19, pp. 10-32
Providence, Rhodes Island
1967

[Forgy, 1982]
« RETE : a fast algorithm for the many pattern many object pattern match problem »
C.L. FORGY
Artificial Intelligence, 19, 1, pp. 17-37
Septembre 1982

[Fowler *et al.*, 1994]
« Formal Methods in the IS Domain : Introducing a Notation for Presenting Object-Z Specifications »
D.C. FOWLER, P.A. SWATMAN, E. WAFULA
Working Paper 1994-1a
Centre for Information System Research
Swinburne University of Technology
P.O. Box 218, Hawthorn, Victoria 3122, Australia
Décembre 1994

[Fraser *et al.*, 1991]
« Informal and Formal Requirements Specification Languages : Bridging the Gap »
M.D. FRASER, K. KUMAR, V.K. VAISHNAVI
IEEE Transactions on Software Engineering, vol. 17, n°5, pp. 454-466
Mai 1991

[Fraser *et al.*, 1994]
« Strategies for Incorporating Formal Specifications in Formal Development »
M.D. FRASER, K. KUMAR, V.K. VAISHNAVI
Communications of the ACM, vol. 37, n° 10, pp. 74-86
Octobre 1994

[Frasson et Gauthier, 1994]
« *A gradual software environment for developing tutoring systems* »
C. FRASSON, G. GAUTHIER
7th International Conference on Systems Research, Informatics and Cybernetics,
Baden-Baden, Germany, pp. 73-78
1994

[Frasson *et al.*, 1996]
« An Actor-based Architecture for Intelligent Tutoring Systems »
C. FRASSON, T. MENGELLE, E. AÏMEUR, G. GOUARDERES
Third International Conference ITS 96, pp. 57-65
Montréal, Canada
1996

[Futtersack, 1994]
« *QUIZ : Une architecture multi-agents pour un tuteur intelligent* »
M. FUTTERSACK
Thèse de Doctorat de l'Université Paris VI, Spécialité Informatique
1994

[Gadwal et Boloix, 1993]
« *ETL Reference Manual* »
D. GADWAL, G. BOLOIX
Department of Computational Science, University of Saskatchewan
Mars 1993

[Gadwal *et al.*, 1993]
« *EDL/GE User's Manual* »
D. GADWAL, P. LO, B. MILLAR
Department of Computing Science, University of Alberta
Décembre 1993

[Gagné, 1985]
« *The conditions of learning and the theory of instruction* »
R.M. GAGNE
CBS College Publishing, 4th edition
1985

[Ganascia, 1996]
« Demain, des appareils bien élevés ! »
J.G. GANASCIA
Revue « La Recherche », n° 285, pp. 106-109
Mars 1996

[Genesereth, 1991]
"*Knowledge Format. Principles of Knowledge Rrepresentation and Reasoning*"
M.R. GENERESETH
Proceedings of the Second International Conference,
Cambridge, MA, Morgan Kaufmann, pp. 599-600
1991

[Giarratano et Riley, 1991]
« Expert Systems - Principles and Programming »
J. GIARRATANO, G. RILEY
PWS Publishing Company, Boston, 2nd edition
1994

[Girardi et Ibrahim, 1993]
« An approach to improve the effectiveness of software retrieval »
M.R. GIRARDI, B. IBRAHIM
Proceedings of the Third Irvine Software Symposium
D. J. RICHARDSON, R.N. TAYLOR Editions, pp. 89-100
University of California
Avril 1993

[Goblet et Benslimane, 1995]
« *Conception d'un tuteur intelligent : représentation centrée agents du domaine d'enseignement et étude des interactions.* »
X. GOBLET, A. BENSLIMANE
Environnements Interactifs d'apprentissage avec Ordinateurs
Journées EIAO-ENS - Cachan, France, Tome 2 - pp. 279-289.
1995

[Goguen, 1996]
« Formality and Informality in Requirements Engineering »
J. GOGUEN
Proc. IEEE Int'l Conf. Requirements Eng., IEEE CS Press, Los Alamitos, Calif.
1996

[Goossens, 1981]
« La méta-évaluation au service de la compréhension automatique de programme »
D. GOOSSENS
Thése de 3ème cycle - L.I.T.P., Université Paris VIII,
Janvier 1981

[Gorm Larsen *et al.*, 1996]
« Applying Formal Specification in Industry »
P. GORM LARSEN, J. FITZGERALD, T. BROOKES
IEEE Software, pp 48-56
Mars 1996

[Gouardères *et al.*, 1997]
« *The KAMP Project* »
G. GOUARDERES, T. NODENOT, I. OCHOA
Management Report 10/97 - 12/97, IUT/WP2T2/030
1997

[Grice, 1975]
« *Logic and Conversation* »
P. GRICE
Syntax and Semantics 3 : Speech Acts, pp. 41-58
Cole and Morgan Eds, New York Academic Press
1975

[Gruber, 1992]
« *Ontolingua: A mechanism to support portable ontologies* »
T.R. GRUBER
Stanford University, Knowledge Systems Laboratory, Technical Report, pp. 91-66, Revision
Juin 1992

[Gruber, 1993]
« *A translation approach to portable ontologies* »
T.R. GRUBER
Knowledge Acquisition, 5(2), pp. 199-220
1993

[Gupta *et al.*, 1989]
« An Object-Oriented VLSI CAD Framework »
R. GUPTA, W.H. CHENG, I. HARDONAC, M.A. BREUER
Computer, pp. 28-37
Mai 1989

[Guzdial *et al.*, 1996]
« Computer Support for Learning through Complex Problem Solving »
M. GUZDIAL, J. KOLODNER, C. HMELO, H. NARAYANAN, D. CARLSON, N. RAPPIN, R. HUBSCHER, J. TURNS, W. NEWSTETTER
Communications of the ACM, Vol. 39, n° 4, pp. 43-45
Avril 1996

[Hantler et Icing, 1976]
« An Introduction to Proving the Correctness of Programs »
S.L. HANTLER, J.C. ICING
RC 5893, IBM TJ. Watson Research Center, Yorktown Heights, NY
1976

[Harandi et Miriyala, 1991]
« *Automatic Derivation of formal software specifications from informal descriptions* »
M.T. HARANDI, K. MIRIYALA
IEEE Transactions on Software Engineering
Vol. 17, n° 10, pp. 1126-1142
Octobre 1991

[Hernandez *et al.*, 1993]
« ALBA : A parallel Language Based on Actors »
J. HERNANDEZ, P. DE MIGUEL, M. BARRENA, J.M. MARTINEZ, A. POLO, M. NIETO
ACM SIGPLAN Notices, Volume 28, n° 4, pp. 11-20
Avril 1993

[Hewitt, 1977]
« Viewing control structures as pattern of passing messages »
C. HEWITT
Journal of Artificial Intelligence, 8(3), pp. 323-364
1977

[Hoare, 1969]
« An Axiomatic Basis for Computer Programming »
C.A.R. HOARE
CA CM, 12, pp. 576-581
1969

[Hoare, 1985]
« Comunicating Sequential Process »
C.A.R. HOARE
Prentice-Hall, Englewood-Cliffs, N.J.
1985

[Houssaye, 1988]
« *Le triangle pédagogique* »
J. HOUSSAYE
Bern, Peter Lang
1988

[Hullot, 1986]
« SOS Interface, un générateur d'interfaces homme-machine (pour Macintosh) »
J.M. HULLOT
Actes des 3èmes JLOO, Bigre+Globule #48, pp. 69-78, Paris
1986

[Ibrahim et Cummins, 1990]
« KSL/logic : Integration of logic with objects »
M.H. IBRAHIM, F.A. CUMMINS
1990 Int'l Conference on Computer Languages
New-Orleans, Los Alamitos, CA
IEEE Company Society Press, pp. 228-235
Mars 1990

[Ishikawa et Tokoro, 1986]
« Concurrent object-oriented knowledge representation language Orient84/K »
Y. ISHIKAWA, M. TOKORO
OOPSLA'86 : OOP Systems, Languages end Applications, Portland
SIGPLAN Notices, 21(11), pp. 232-241
1986

[Jackson, 1983]
« *System Development* »
M.A. JACKSON
Prentice Hall, Englewood Cliffs, New Jersey
1983

[Jackson, 1995]
« Software Requirements and Specifications »
M. JACKSON
Addison-Wesley, Reading, Mass.
1995

[Jackson, 1996]
« Requirements Need Form, Maybe Formality »
D. JACKSON
IEEE Software, pp. 21-22
Mars 1996

[Jackson *et al.*, 1996]
« A Learner-Centered Tool for Students Building Models »
S.L. JACKSON, S.J. STRATFORD, J. KRAJICK, E. SOLOWAY
Communications of the ACM, Vol. 39, n° 4, pp. 48-49
Avril 1996

[Jacobson, 1993]
« *Le Génie Logiciel Orienté-Objet* »
I. JACOBSON
Paris, Editions Addison-Wesley
1993

[Jarke et Pohl, 1994]
« Requirements Engineering in 2001 : (Virtually) Managing a Changing Reality »
M. JARKE, K. POHL
Software Engineering, pp. 257-266
Novembre 1994

[Jayaratna, 1994]
« Understanding and Evaluating Methodologies. NIMSAD : A systemic Framework »
N. JAYARATNA
Mac-Grawhill, London
1994

[Joachims *et al.*, 1995]
« WebWatcher : Machine Learning and Hypertext »
T. JOACHIMS, T. MITCHELL, D. FREITAG, R. ARMSTRONG
School of computer Science
Carnegie Mellon University
Mai 1995

[Jones, 1984]
« Reusability in Programming: a Survey of the State of the Art »
T.C. JONES
IEEE Transactions on Software Engineering, 10, #5, pp. 488-494
1984

[Jungclaus *et al.*, 1996]
« TROLL - A Language for Object-Oriented Specification of Information Systems »
R. JUNGCLAUS, G. SAAKE, T. HARTMANN, C. SERDANAS
ACM Transactions on Information Systems, Vol. 14, n° 2, pp. 175-211
Avril 1996

[Kemmerer, 1990]
« Integrating Dormal Methods into the Development Process »
R.A. KEMMERER
IEEE Sofware, pp. 37-50
Septembre 1990

[King, 1976]
« On Generaling Verification Conditions for Correctness Proofs »
J.C. KING
Programmiersprachen, 4. Fachtagung der G.l. Erlangen 1976
Ed. A.J. Schneider & M. Nagl, Springer Verlag, Berlin
1976

[Kingston, 1994]
« *Linking Knowledge Acquisition with CommonKADS Knowledge Representation* »
J.K.C. KINGSTON
Proceedings of the BCS SGES Expert Systems'94
St John's College, Cambridge
Décembre 1994

[Koschmann et Evens, 1988]
« Bridging the gap between object-oriented and logic programming »
T. KOSCHMANN, M.W. EVENS
IEEE Software, 5(5), pp. 36-42
1988

[Krasner, 1980]
« Requirements Dynamics in Large Software Projects, A Perspective on New Directions in the Software Engineering Process »
G. KRASNER
Proc. IFIP, Elsevier, New York, pp. 211-216
1980

[Krasner et Pope, 1988]
G.E. KRASNER, S.T. POPE
« A Cookbook for Using the Model-View Controller User Interface Paradigm in Smalltalk-80 »
Journal of Object-Oriented Programming, 1(3), pp. 26-49
1988

[Krief, 1992]
« Utilisation des langages objets pour le prototypage »
P. KRIEF
Collection Etudes et Recherche en Informatique
Editions MASSON
1992

[Labidi et Lejouad, 1993]
« *De l'intelligence artificielle distribuée aux systèmes multi-agents* »
S. LABIDI, W. LEJOUAD
Rapport de recherche de l'INRIA, n° 2004, Programme 2 : Calcul symbolique, programmation et génie logiciel
1993

[Larkin et Simon, 1987]
« Why a Diagram is (Sometimes) Worth Ten Thousand Words »
J.H. LARKIN, H.A. SIMON
Cognitive Science, 11, pp. 65-99
1987

[Laurière, 1982]
« Intelligence Artificielle, résolution de problèmes par l'homme et par la machine »
J.L. LAURIERE
Techniques et Sciences Informatiques, 1 et 2, pp. 25-42 et 109-133
1982

[Lawrence, 1996]
« Do You Really Need Formal Requirements ? »
B. LAWRENCE
IEEE Software, pp. 20-22
Mars 1996

[Lecomte, 1995]
« Les fondements de la créativité »
J. LECOMTE
Sciences Humaines, n° 56, pp. 28-31
Décembre 1995

[Lehmann, 1989]
« *Les propriétés souhaitées des produits logiciels et du processus de développement des systèmes logiciels* »
M. LEHMANN
Revue Génie Logiciel et Systèmes Experts, n° 14
1989

[Le Moigne, 1990]
« *La modélisation des systèmes complexes* »
J.L. LE MOIGNE
AFCET Systèmes, Editions DUNOD
1990

[Letovsky et Soloway, 1986]
« Delocalized Plans and Program Comprehension »
S. LETOVSKY, E. SOLOWAY
IEEE Software, 3, #3, pp. 41-49
Mai 1986

[Lewis *et al.*, 1989]
« Prototypes from Standard User Interface Management Systems »
T.G. LEWIS, F. HANDLOSER, S. BOSE, S. YANG
Computer, pp. 51-60
Mai 1989

[Lima, 1995]
« La modélisation et la mise en oeuvre de processus de diagnostic par un système générateur de systèmes de transfert intelligent de connaissances »
C. LIMA
Sciences et Techniques Educatives,
Vol. 2, n° 2, pp. 173-201
1995

[Lin *et al.*, 1996]
« Adaptative Interaction through WWW »
F. LIN, R. DANIELSON, S. HERRGOTT
Proceedings of ED-TELECOM 96, pp. 173-178
Juin 1996

[Luqi, 1989]
« Sofware Evolution Through Rapid Prototyping »
LUQI
Computer, pp. 13-25
Mai 1989

[Malenfant *et al.*, 1989]
« ObjVProlog : Metaclasses in logic »
J. MALENFANT, G. LAPALME, J. VAUCHER
ECOOP'89 : European Conference on OOP, Nottingham
Cambridge University Press, pp. 257-269
1989

[Maret, 1995]
« L'art de penser »
P. MARET
Sciences Humaines, n° 56, pp. 16-17
Décembre 1995

[McAllister, 1988]
« *Modeling Concepts for Specification Environments* »
A. MCALLISTER
PhD Thesis. Dept of Computational Science, University of Saskatchewan. Research Report 88-1
Mars 1988

[McClure, 1989]
« *CASE in software automation* »
C. MCCLURE
Prentice Hall, Englewood Cliffs, New Jersey
1989

[McInally, 1997]
« Product Life Cycle Based Requirement Capture in Complex Instrument Design »
S. MCINALLY
5th BCS ISM conference, Preston
Springer-Verlag Ltd
1997

[Meirieu, 1994]
« Apprendre… oui, mais comment »
P. MEIRIEU
Collection Pédagogies
ESF éditeur, Paris
13ème édition
1994

[Mellender, 1988]
« An Integration of Logic and Object-Oriented Programming »
F. MELLENDER
SIGPLAN Notices, Vol. 23, n° 10, pp. 181-185
Octobre 1988

[Mengelle, 1995]
« Etude d'une architecture d'environnements d'apprentissage basés sur le concept de préceptorat avisé »
T. MENGELLE
Doctorat de l'Université Paul Sabatier, Spécialité Informatique, TOULOUSE III
Juillet 1995

[Mettrey, 1991]
« A Comparative Evaluation of Expert System Tools »
W. METTREY
IEEE Computer, Vol. 24, n° 2
Février 1991

[Millet, 1997]
« SERAC : un Système d'Evaluation et de Révision Automatisé des Connaissances en EIAO »
S. MILLET
Thèse de Doctorat de l'Université Paul Sabatier, Spécialité Informatique, TOULOUSE III
Novembre 1997

[Milner *et al.*, 1989]
« A calculus of mobile processes, parts i and ii »
R. MILNER, J.G. PARROW, D.J. WALKER
Technical Report ECS-LFCS-89-85, ECS-LFCS-89-86
Université d'Edimbourgh
1989

[Moinard et Joab, 1997]
« Dynamic Assessment for Operator Training »
C. MOINARD, M. JOAB
User Modeling : Proceedings of the Sixth International Conference, UM97
Anthony Jameson, Cécile Paris and Carlo Tasso Eds, pp. 255-267
Springer Wien, New York
1997

[Montfort, 1996]
« *Structuration des connaissances et modèle objet : Application à la réalisation d'un simulateur* »
V. MONTFORT
Thèse de Doctorat de l'Université de Pau et des Pays de l'Adour, Spécialité Informatique
1996

[Moss, 1990]
« An introduction to Prolog++ »
C. MOSS
Research Report DOC 90/10
Imperial College, London
Juin 1990

[Mottet, 1983]
« LA TECHNOLOGIE EDUCATIVE
Pour une optique recentrée »
G. MOTTET
Revue Française de Pédagogie, n° 63
Avril - Mai - Juin 1983

[Neches *et al.*, 1991]
« *Enabling technology for knowledge sharing* »
R. NECHES, R. E. FIKES, T. FININ, T.R. GRUBER, R. PATIL, T. SENATOR, W.R. SWARTOUT
AI Magazine, 12(3), pp. 16-36
1991

[Neumann et Horwitz, 1996]
« Linking Models to Data: Hypermodels for Science Education »
E.K. NEUMANN, P. HORWITZ
Proceedings of ED-TELECOM 96, pp. 221-226
Juin 1996

[Nicaud et Vivet, 1988]
« Les Tuteurs Intelligents : réalisations et tendances de recherches »
J. NICAUD, M. VIVET
TSI, Vol. 7, n° 1, pp. 21-45
1988

[Nissen *et al.*, 1996]
« Technology to Manage Multiple Requirements Perspectives »
H. NISSEN, M.A. JEUSFELD, M. JARKE, G.V. ZEMANEK, H. HUBER
IEEE Software, pp. 37-48
Mars 1996

[NKambou, 1996]
« Modélisation des connaissances de la matière dans un système tutoriel intelligent : modèles, outils et applications »
R. NKAMBOU
Thèse de Doctorat (Ph D) en informatique de l'Université de Montréal
Avril 1996

[Nodenot, 1992]
« M.A.G.E. : un Méta-Atelier de Génie Educatif »
T. NODENOT
Thèse de Doctorat de l'Université Paul Sabatier, Spécialité Informatique, TOULOUSE III
Novembre 1992

[Noël, 1995]
« La métacognition
L'art d'évaluer ses performances »
B. NOËL
Sciences Humaines, n° 56, pp. 23-25
Décembre 1995

[Norman, 1996]
« Grandeur et misère de la technologie »
D.A. NORMAN
Revue « La Recherche », n° 285, pp. 22-25
Mars 1996

[Norman et Spohrer, 1996]
« Learner-Centered Education »
D.A. NORMAN, J.C. SPOHRER
Communications of the ACM, Vol. 39, n° 4, pp. 24-27
April 1996

[Paquette *et al.*, 1996]
« Developing the Virtual Campus Environment »
G. PAQUETTE, C. RICCIARDI-RIGAULT, C. PAQUIN, S. LIEGEOIS, E. BLEICHER
Proceedings of ED-TELECOM 96
Juin 1996

[Peckham et Maryanski, 1988]
« Semantic data models »
J. PECKHAM, F. MARYANSKI
ACM Comput. Surv. 20, 3, pp. 153-189
1988

[Péninou, 1993]
« MACT, un modèle d'agents centrés tache pour la production de tuteurs Intelligents par l'AGDI »
A. PENINOU
Thèse de doctorat de l'université Paul Sabatier, Spécialité Informatique, Toulouse III
Septembre 1993

[Pernici, 1991]
« Objects with Roles »
B. PERNICI
Proceedings of the ACM/IEEE Intrenational Conference on Office Information Systems
SIGOIS Bulletin 11, 2-3, pp. 205-215
1991

[Peterson, 1981]
« Petri Net theory and the Modelling of Systems »
J. PETERSON
Prentice Hall, Englewood Cliffs, New Jersey
1981

[Pierre *et al.*, 1992]
« Orientations-Projets-Activités à l'Ecole Maternelle »
R. PIERRE, J. TERRIEUX, N. BABIN
Collection L'école au quotidien, Hachette Education
1992

[Plihon et Rolland, 1995]
« *Modelling ways-of-working* »
V. PLIHON, C. ROLLAND
7th International Conference on Advanced Information Systems Engineering, Jyvaeskylae, Finland
1995

[Pnueli, 1986]
« Application of temporal logic to the specification and verification of reactive systems : A survey of current trends »
A. PNUELI
Current Trends in Concurrency, J. De Bakker, W. De Roever, G. Rozenberg
Eds. Lecture Notes in Computer Science, Vol. 224, SpringerVerlag, Berlin
1986

[Pohl, 1994]
« The three dimensions of requirements engineering : A framework and its application »
K. POHL
Information Systems, 19 (2)
1994

[Polson, 1993]
« *Tasks analysis for an automated instructional design advisor* »
M.C. POLSON
CBS College Publishing, 4th edition
1985

[Potts *et al.*, 1994]
« Inquiry-Based Requirements Analysis »
C. POTTS, K. TAKAHASHI, A. ANTON
IEEE Software, pp. 21-32
Mars 1994

[Prince, 1996]
« *Vers une informatique cognitive dans les organisations »*
V. PRINCE
Editions MASSON
1996

[Puerta *et al.*, 1992]
« *A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools* »
A.R. PUERTA, J.W. EGAR, S.W. TU, M.A. MUSEN
Knowledge Acquisition, 4, pp. 171-196.
1992

[Quintus, 1988]
« ProWindows Reference Manual »
QUINTUS COMPUTER SYSTEMS INC.
Evaluation Version, CA
Août 1988

[Rao, 1996]
« Quand l'information parle à nos yeux »
R. RAO
Revue « La Recherche », n° 285, pp. 66-73
Mars 1996

[Reichgelt et Shabolt, 1991]
« *ProtoKEW: A knowledge based system for knowledge acquisition* »
H. REICHGELT, N. SHADBOLT
Editions D. SLEEMAN and O. BERNSEN
Recent Advances in Cognitive Science
1991

[Reppy, 1992]
« Higher-Order Concurrency »
J.H. REPPY
Technical Report 92-1285
1992

[Reubenstein et Waters, 1991]
« The requirements apprentice : automated assistance for requirements acquisition »
H.B. REUBENSTEIN, R.C. WATERS
IEEE Transactions on Software Engineering
Vol. 17, n° 3, pp. 226-240
Mars 1991

[Rialle, 1993]
« Contribution à la définition du profil de cogniticien : caractéristiques fonctionnelles, savoirs et savoir-faire dans l'élaboration d'une base de connaissances »
V. RIALLE
Conférence Hypermédias et Apprentissages, Lille, pp. 1-10,
Mars 1993

[Rich, 1981]
« Inspection Methods in Programming »
C. RICH
AI-TR-604, Artificial Intelligence Laboratory, MIT
Juin 1981

[Rickel et Johnson, 1997]
« *Integrating Pedagogical Capabilities in a Virtual Environment Agent* »
J. RICKEL, W.L. JOHNSON
Proceedings of the First International Conference on Autonomous Agents
1997

[Riecken, 1996]
« *Ces braves assistants qui assurent l'intendance* »
D. RIECKEN
Revue « La Recherche », n° 285, pp. 44-47
Mars 1996

[Riley, 1995]
« CLIPS F.A.Q. »
G. RILEY
NASA, Johnson Space Center
Janvier 1995

[Rolland *et al.*, 1995]
« An approach for defining ways-of-working »
C. ROLLAND, C. SOUVEYET, M. MORENO
Information Systems Journal, Vol. 20, n° 4
1995

[Roose, 1997]
« *Communication Homme-Machine et Ingénierie éducative* »
P. ROOSE
Mémoire de DEA de l'Université du Maine, Laboratoire LIUM, Le Mans
1997

[Rosca, 1994]
« Evolution and Reuse of Formal Specifications using Decision Structure and Analogy »
D. ROSCA
Thesis Proposal Draft
Department of Computer Science
Old Dominion University
Norfolk, VA 23529-0162
Juillet 1994

[Rosson et Carroll, 1996]
« Scaffolded Examples for Learning Object-Oriented Design »
M.B. ROSSON, J.M. CARROLL
Communications of the ACM, Vol. 39, n° 4, pp. 46-47
Avril 1996

[Rumbaugh *et al.*, 1994]
« Modélisation et conception orientées objet »
J. RUMBAUGH, M. BLAHA, W. PREMERLANI, F. EDDY, W. LORENSEN
Edition française revue et augmentée
Editions MASSON, PRENTICE HALL
1994

[Schank et Kass, 1996]
« A Goal-Based Scenario for High School Students »
R. SCHANK, A. KASS
Communications of the ACM, Vol. 39, n° 4, pp. 28-29
Avril 1996

[Selker, 1996]
« Scénarios pour ordinateurs à venir »
T. SELKER
Revue « La Recherche », n° 285, pp. 26-31
Mars 1996

[Sestoft et Sondergaard, 1988]
« A Bibliography on Partial Evaluation »
P. SESTOFT, H. SONDERGAARD
SIGPLAN Notices, 23, #2, pp. 19-27
1988

[Shapiro, 1983]
« Object-oriented programming in Concurrent Prolog »
R. SHAPIRO
New Genera. Computing, 1(1), pp. 25-48
1983

[Shaw et Gaines, 1987]
« *KITTEN: Knowledge Initiation and Transfer Tools for Experts and Novices* »
M.L.G. SHAW, B.R. GAINES
International Journal of Machine Studies, 27, pp. 251-280
1987

[Shlaer et Mellor, 1991]
« OBJECT LIFECYCLES
Modeling the World in States »
S. SHLAER, S.J. MELLOR
Yourdon Press
P T R Prentice Hall
Englewood Cliffs, New Jersey 07632
1991

[Siddiqi et Sherakan, 1996]
« Requirements Engineering : The Emergency Wisdom »
J. SIDDIQI, C. SHERAKAN
IEEE Software, pp. 15-22
Mars 1996

[Smart et Rae, 1995]
« HARDY User Guide version 1.3 »
J. SMART, R. RAE
Artificial Intelligence Applications Institute
University of Edinburgh
South Bridge, Edinburgh, EH1 1HN, UK
Février 1995

[Smart, 1995a]
« User Manual for wxCLIPS 1.43 »
J. SMART
Decision Support Group
Artificial Intelligence Applications Institute
University of Edinburgh
South Bridge, Edinburgh, EH1 1HN, UK
Juin 1995

[Smart, 1995b]
« User Manual for wxWindows 1.63 : a portable C++ GUI toolkit »
J. SMART
Artificial Intelligence Applications Institute
University of Edinburgh
South Bridge, Edinburgh, EH1 1HN, UK
Août 1995

[Spivey, 1989]
« *The Z Notation : A Reference Manual* »
J.M. SPIVEY
International Series in Computer Science
Prentice Hall, Hemel Hempstead, Hertforshire HP2 4RG, UK
1989

[Srivastana *et al.*, 1993]
« Coral++ : Adding Object-Orientation to a Logic Database Language »
D. SRIVASTANA, R. RAMKRISHNAN, P. SESHDRAVI, S. SUDARSHAN
Proceedings of the 19th VLDB Conference
Dublin, Ireland
1993

[Tansley et Hayball, 1993]
« *Knowledge bases systems analysis and design, a KADS développer's Hanbook* »
D.S.W. TANSLEY, C.C. HAYBALL
The BCS Practitioner Series, Prentice Hall
1993

[Teitelman, 1978]
« INTERLISP Reference Manual »
W. TEITELMAN
Xerox Palo Alto Research Center
Octobre 1978

[Thomas *et al.*, 1996]
« *Apprentissage de connaissances d'un modèle d'expertise guidé par ses structures* »
J. THOMAS, J.G. GANASCIA, P. LAUBLET
N. Aussenac, P.Laublet, C.Reynaud (eds), Tendances actuelles en acquisition et modélisation des connaissances.
Edition « Cépadues »
1996

[Tolvanen, 1997]
« *Training Methodology Researchers* »
J.P. TOLVANEN
5th BCS ISM conference, Preston
Springer-Verlag Ltd
1997

[Tue Ho et Vogel, 1988]
« *Une gestion de configuration logicielle* »
X. TUE HO, M. VOGEL
Génie Logiciel & Systèmes Experts, n° 13, pp. 66-75
Décembre 1988

[Uustalu, 1994]
« Extension of Structural Synthesis of Programs »
T. UUSTALU
Dept. of Teleinformatics, The Royal Institute of Technology Electrum 204
S-164 40 Kista (Stockholm), Sweden
Rapport Technique
1994

[Van Belle, 1996]
« *A Critique of Current Systems Engineering Methods: The Case for Ontology-Augmented Methodologies* »
J.P. VAN BELLE
8th Conference on Advanced Information Systems Engineering
Workshop on the Evaluation of Modelling Methods in Systems Analysis and Design, Heraklion (Grèce)
Mai 1996.

[Vernada et Zelm, 1996]
« *Business process modelling methods for enterprise integration : IDEF3 and CIMOSA* »
F.B. VERNADA, M. ZELM
Proceedings of the fourth international conference on Control, Automation, Robotics and Vision
ICARCV'96, Singapour, pp. 1585-1589
1996

[Vialatte, 1985]
« Description et applications du moteur d'inférence SNARK »
M. VIALATTE
Thèse de Doctorat de l'Université Paris VI, Spécialité Informatique
Mai 1985

[Vigny, 1997]
« Classification et utilisation de l'Éco-Résolution »
C. VIGNY
Mémoire de bibliographie, DEA de Production Automatisée
LURPA, ENS Cachan
Juillet 1997

[Vogel, 1988]
« *Génie cognitif* »
C. VOGEL
Editions MASSON
1988

[Vogel, 1991]
« La spécification des systèmes d'information en langage naturel »
C. VOGEL
Génie Logiciel & Systèmes Experts, n° 23, pp. 6-13
Juin 1991

[Weinberg, 1995]
« Notre logique ne l'est pas toujours… »
A. WEINBERG
Sciences Humaines, n° 56, pp. 18-19
Décembre 1995

[Wertz, 1985]
« Intelligence Artificielle : Application à l'analyse de programmes »
H. WERTZ
Editions MASSON, Paris
1985

[Wielinga, 1993]
« *Expertise Model: Model Definition Document* »
B. WIELINGA
ComonKADS Project Report
University of Amsterdam, KADS-II/M2/UvA/026/2.0
Octobre 1993

[Wirsing, 1993]
« Développement de logiciel et spécification formelle »
M. WIRSING
Technique et Science Informatiques, Volume 12 - n° 4, pp. 413-431
1993

[Woolf et Hall, 1995]
« Multimedia Pedagogues
Interactive Systems For Teaching and Learning »
B. WOOLF, W. HALL
IEEE Computer, pp. 74-80
Mai 1995

[Woolf, 1996]
« Intelligent Multimedia Tutoring Systems »
B. P. WOOLF
Communications of the ACM, Vo. 39, n° 4, pp. 30-31
Avril 1996

[Yourdon et Constantine, 1975]
« *Structured Design* »
E. YOURDON, L. CONSTANTINE
Yourdon Press
1975

[Zamperoni et Gerritsen, 1994]
« Integrating the Developers' and the Management's Perspective of an Incremental Development Life Cycle
A. ZAMPERONI, B. GERRITSEN »
12nd Annual Pacific Northwest Software Quality Conference, Portland, Oregon
Octobre 1994

[Zaniolo, 1984]
« Object-oriented programming in Prolog »
ZANIOLO
1984 International Symposium on Logic Programming, Atlantic City
IEEE Computer Society Press, pp. 265-270
1984

Références Bibliographiques personnelles

[Marquesuzaà *et al.*, 1996a]
« *Specifying Educational Software : goals and process* »
Christophe MARQUESUZAA, Jacques MEYRANX, Thierry NODENOT
Actes de la conférence ED-MEDIA 96, pp. 413-419
World Conference on Educational Multimedia and Hypermedia,
Boston, Massachussets, USA,
Juin 1996

[Marquesuzaà *et al.*, 1996b]
« *Towards pedagogically sound learning environments : The specification process* »
Christophe MARQUESUZAA, Jacques MEYRANX, Thierry NODENOT
Actes de la conférence CALISCE'96, pp. 87-95
Third International Conference on Computer Aided Learning and Instruction in Science and Enginneering,
Donostia - San Sebastian, SPAIN,
Juillet 1996

[Marquesuzaà *et al.*, 1997a]
« *Architecture and Principles of a Specification Environment Dedicated to Pedagogues* »
Christophe MARQUESUZAA, Jacques MEYRANX, Thierry NODENOT
Actes de la conférence SITE 97
8th International Conference Society for Information technology and Teacher Education
Orlando- Florida, USA,
Avril 1997

[Marquesuzaà *et al.*, 1997b]
« *The role of CASE tools in the process of specifying educational software* »
Christophe MARQUESUZAA, Jacques MEYRANX, Thierry NODENOT
5th Annual Conference on Methodologies
British Computer Society, Specialist Group on Information System Methodologies
Preston, Lancashire, UK,
Août 1997

Annexes

Présentation

Les annexes de ce document sont scindées en six parties.

L'annexe 1 est une présentation de trois projets et outils existants pour représenter des connaissances de façon ontologique : MECANO, Protégé II et Protégé/Win.

L'annexe 2 est constituée par une exposition de dix outils « méta-CASE » commerciaux et non-commerciaux que nous avons étudié.

L'annexe 3 est un recueil de systèmes et langages basés sur les paradigmes de programmation objet et logique.

L'annexe 4 est un aperçu de plusieurs logiciels de programmation basés sur les paradigmes objet et logique : CLIPS, ART-IM, KES, Level 5, Vax OPS5, OPS5, ARGOS-II et SNARK.

Les différents logiciels que nous avons utilisés pour la programmation de notre prototype (wxWindows, wxCLIPS, HARDY, Tex2RTF et RTFHTM) sont exposés en annexe 5.

Enfin, l'annexe 6 présente la notion d'agents et de systèmes multi-agents ainsi que divers exemples de programmes utilisant notamment des agents logiciels sur Internet.

Sommaire

SOMMAIRE ..... **A IIIII**

LISTE DES FIGURES ..... **A VII**

ANNEXE 1 : OUTILS POUR MODELISER LES ONTOLOGIES : MECANO, PROTEGE II ET PROTEGE/WIN ..... **A 1**

LE PROJET « MECANO » ..... A 1
Objectifs du projet ..... *A 1*
Le modèle d'interface de Mecano ..... *A 1*
L'environnement de développement d'interface de Mobi-D ..... *A 3*
« PROTEGE-II » ..... A 4
« PROTEGE/WIN » ..... A 7
OntologyEditor ..... *A 7*
LayoutEditor ..... *A 8*
LayoutInterpreter ..... *A 8*
EXEMPLE D'UTITILISATION DE PROTEGE/WIN ..... A 9
L'éditeur d'ontologies ..... *A 9*
1. Partie Classe ..... A 9
2. Partie Attribut ..... A 9
L'éditeur de mise en page de ces ontologies ..... *A 10*
L'interpréteur d'ontologies ..... *A 12*

ANNEXE 2 : DES EXEMPLES D'OUTILS META-CASE ..... A 15

«CONCEPTBASE » ..... A 15
« GOODSTEP » ..... A 16
« HOTDRAW » ..... A 16
« METAVIEW » ..... A 17
« HARDY » ..... A 19
« METAEDIT » ..... A 20
« GRAPHICAL DESIGNER 2.0 » ..... A 20
« OBJECT MAKER » ..... A 21
« INTERSOLV PVCS » ..... A 21
« IPSYS TBK » ..... A 22

ANNEXE 3 : QUELQUES EXEMPLES DE SYSTEMES ET LANGAGES BASES SUR LES PARADIGMES « OBJET » ET « LOGIQUE » ..... A 25

TROLL ..... A 25
AMS ..... A 26
POINT DE VUE DE [DAVISON, 1991] ..... A 26
CORAL++ ..... A 27
2MU ..... A 28
POINT DE VUE DE [FOWLER ET AL., 1994] ..... A 29
POINT DE VUE PAR ACTEURS SELON [AGHA ET AL., 1993] ..... A 30
ALBA ..... A 31
AOUIS ..... A 32
POINT DE VUE DE [ARAPIS, 1990] ..... A 33
ORM ..... A 34
OBJECT-Z ..... A 36
OODLE ..... A 37

ANNEXE 4 : QUELQUES LOGICIELS DE PROGRAMMATION ETUDIES POUR NOTRE PROTOTYPE ..... A 39

CLIPS ..... A 39
Qu'est ce que CLIPS ? ..... A 39
CLIPS 4.3 ..... A 40
1. La représentation des connaissances en CLIPS 4.3 ..... A 40
2. L'inférence dans CLIPS 4.3 ..... A 41
CLIPS 6.0 ..... A 41
Versions spéciales de CLIPS ..... A 42
ART-IM : AUTOMATED REASONING TOOL FOR INFORMATION MANAGEMENT ..... A 43
La représentation des connaissances en ART-IM 2.0 ..... A 43
L'inférence dans ART-IM 2.0 ..... A 43
L'environnement de production de ART-IM 2.0 ..... A 44
KES : KNOWLEDGE ENGINEERING SYSTEM ..... A 44
La représentation des connaissances en KES 2.6 ..... A 44
L'inférence dans KES 2.6 ..... A 44
L'environnement de production de KES 2.6 ..... A 45
LEVEL 5 ..... A 45
La représentation des connaissances en Level 5 ..... A 45
L'inférence dans Level 5 ..... A 45
L'environnement de production de Level 5 ..... A 45
VAX OPS5: VAX OFFICIAL PRODUCTION SYSTEM VERSION 5 ..... A 45
La représentation des connaissances en VAX OPS5 3.0 ..... A 46
L'inférence dans VAX OPS5 3.0 ..... A 46
L'environnement de production de VAX OPS5 3.0 ..... A 46
OPS5 ..... A 47
ARGOS-II ..... A 47
SNARK ..... A 47

ANNEXE 5 : LOGICIELS UTILISES POUR NOTRE PROTOTYPE ..... A 49

« WXWINDOWS » ..... A 49
Présentation ..... A 49
Avantages, inconvénients et perspectives ..... A 50
Utilitaires basés sur wxWindows ..... A 51

« WXCLIPS » ..... A 52
Introduction ..... *A 52*
Relation entre CLIPS, wxCLIPS et wxWindows ..... *A 52*
Utiliser wxCLIPS pour construire des interfaces utilisateurs graphiques (GUI) ..... *A 53*
Connexion avec les classes de bases de données ..... *A 53*
Aperçu général de « wxCOOL » ..... *A 54*
« HARDY » ..... A 54
Présentation ..... *A 54*
HARDY et wxCLIPS ..... *A 55*
TEX2RTF ..... A 56
RTF2THML ..... A 57

ANNEXE 6 : LES AGENTS ..... A 58

DEFINITION D'UN AGENT ..... A 59
LES SYSTEMES MULTI-AGENTS ..... A 59
EXEMPLES D'AGENTS LOGICIELS ..... A 62
Maxims ..... *A 62*
Softbot ..... *A 63*
Assistant M ..... *A 63*
NewT ..... *A 64*
CAP ..... *A 66*
RINGO ..... *A 66*
WebWatcher ..... *A 66*
COACH ..... *A 67*

Liste des Figures

Figure 1 : Architecture de MOBI-D ..... A 3
Figure 2 : Panneau de « ProteView » ..... A 6
Figure 3 : Exemple d'utilisation de « Ontology Editor » ..... A 10
Figure 4 : Exemple d'utilisation du « Layout Editor » ..... A 11
Figure 5 : Autre exemple d'utilisation du « Layout Editor » ..... A 12
Figure 6 : Exemple d'utilisation du « Layout Interpretor » ..... A 13
Figure 7 : Structure d'un schéma de classe en Object-Z ..... A 36

Annexe 1 :
Outils pour modéliser les ontologies : MECANO, Protégé II et Protégé/Win

Le projet « Mecano » [48]

Objectifs du projet

Le projet « *Mecano* » vise à surmonter des limitations actuelles d'outils de développement d'interfaces utilisateurs. « *Mecano* » fournit un environnement logiciel qui supporte toutes les facettes du développement d'interface, depuis le début de la conception jusqu'à l'exécution, grâce notamment à un langage universel de modélisation d'interfaces (MIMIC), des outils correspondants, et un cycle de développement intégré et interactif qui permet aux utilisateurs de disposer du contrôle et de la commande de l'automatisation (éditer, raffiner, prévisualiser et exécuter).

Le modèle d'interface de Mecano

Un modèle d'interface est une base de connaissances qui répond explicitement aux questions suivantes pour une interface quelconque :

- Qui sont les utilisateurs de l'interface ?
- Quelles tâches les utilisateurs doivent-ils exécuter en utilisant l'interface ?
- A quels objets de domaine l'interface doit-elle fournir un accès ?
- Comment les composants d'interface sont-ils présentés à l'utilisateur ?
- Quelles commandes et actions l'utilisateur peut-il exécuter sur l'interface ?

[48] Se référer au site Web à l'adresse suivante : http://smi-web.stanford.edu/projects/mecano/

Chacune de ces questions trouve une réponse, respectivement, dans un des cinq composants de modèle d'interface suivant :

1. *Le modèle de l'utilisateur.*

   Il définit les types d'utilisateurs de l'interface et les attributs pertinents de ces utilisateurs. Son but principal est d'influencer la génération d'interface. Il n'est cependant pas conçu pour être un modèle de l'état mental de l'utilisateur à un moment particulier pendant l'interaction.

2. *Le modèle des tâches de l'utilisateur.*

   C'est une représentation des tâches que l'utilisateur peut exécuter sur l'interface. Ces tâches peuvent différer des tâches du système, ou de l'application. Une tâche utilisateur a un but associé, elle peut inclure un nombre de sous-tâches qui sont exécutées selon une procédure particulière, et elle peut avoir à satisfaire un certain nombre de conditions avant de pouvoir être exécutée par l'utilisateur. Les définitions des tâches utilisateurs sont appliquées pendant le processus de génération d'interface pour dériver (déduire) le dialogue d'interface.

3. *Le modèle du domaine.*

   Il définit les objets auxquels l'utilisateur accède par l'interface. Les objets de domaine sont organisés en classes et ont des propriétés associées. Le modèle de domaine est utile pour la génération de la présentation d'interface.

4. *Le modèle de présentation.*

   C'est une vue des caractéristiques statiques d'une interface, essentiellement son agencement, et les attributs tels que les fontes et couleurs. Un environnement basé sur la notion de modèle est capable de générer la plupart des modèles de présentation en examinant le modèle de domaine et en suivant les principes de présentation de la plate-forme donnée.

5. *Le modèle de dialogue.*

   Il spécifie les commandes utilisateur, les techniques d'interaction, les réponses de l'interface et les séquences de commandes que l'interface permet pendant les sessions de l'utilisateur. Les spécifications de dialogue peuvent être dérivées en grande partie du modèle des tâches de l'utilisateur.

En outre, pour représenter les concepts définis par chacun des composants d'un modèle d'interface, il est nécessaire d'avoir *un langage de modélisation*. Un Modèle d'Interface de Mecano (MIM) est défini utilisant un langage orienté objet de modélisation appelé MIMIC.

MIMIC est un langage de définition qui suit les concepts orientés objet généraux de C++. Le langage fournit le cadre général pour définir des modèles d'interface génériques ou spécifiques à une application. Dans la méthode Mecano, l'environnement de développement Mobi-D fournit un modèle d'interface générique. Les développeurs démarrent avec un modèle générique et construisent des modèles spécifiques aux applications à partir desquels les interfaces sont générées.

L'environnement de développement d'interface de Mobi-D

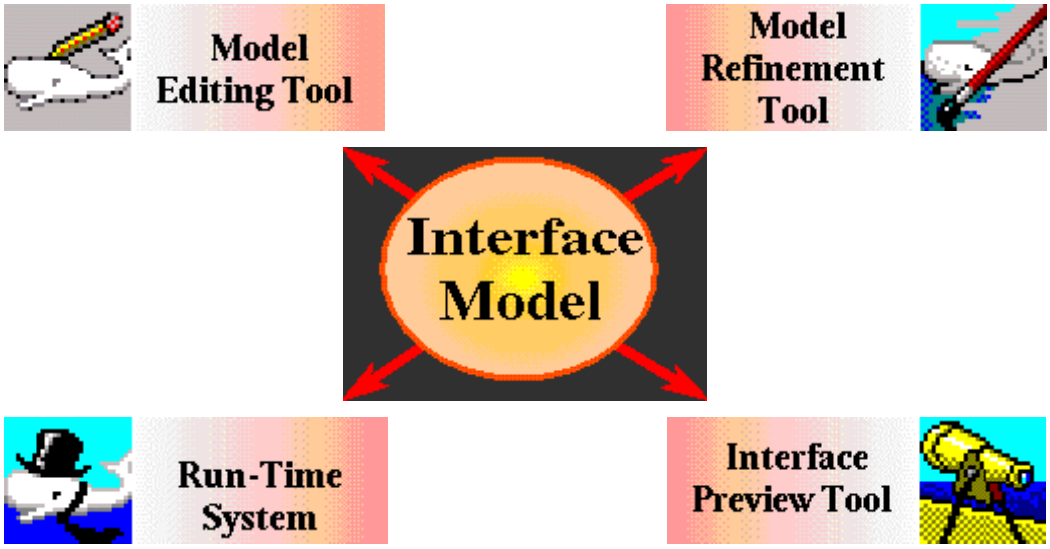

Figure 1 : Architecture de MOBI-D

Mobi-D est un concepteur d'interface basé sur la notion de modèle. C'est un environnement logiciel qui supporte toutes les facettes de conception, réalisation, et exécution d'une interface utilisateur. Ceci est réalisé grâce à une suite d'outils logiciels qui raffinent un modèle d'interface générique en un modèle d'interface spécifique à une application. Cette dernière peut alors être exécutée sur une plate-forme donnée.

Un modèle d'interface définit un vocabulaire commun pour exprimer des concepts des interfaces de toutes sortes. Il représente toutes les caractéristiques qui définissent la conception d'une interface, incluant, entre autres, les tâches utilisateurs, le dialogue, la présentation, et les préférences de l'utilisateur. Le modèle d'interface de Mecano est destiné à être une ressource pour la communauté HCI entière.

Mobi-D définit un cycle de développement itératif pour des interfaces qui inclut les phases suivantes :

1. L'édition des définitions basiques de modèle d'interface avec *l'outil d'édition de modèle*.
   Il permet aux développeurs de visualiser et modifier les modèles d'interface. Un développeur Mobi-D peut construire une spécification complète d'interface en utilisant cet outil. Toutefois, les développeurs démarrent en principe la conception des interfaces avec cet outil et appliquent ensuite l'outil de raffinement de modèle.
2. Le raffinement automatique du modèle d'interface de base avec *l'outil de raffinement de modèle*.

Il permet d'automatiser complètement ou partiellement le processus de conception comme l'agencement, la navigation par fenêtre, la structure de dialogue, et la spécification de commande. Cet outil donne aux développeurs le contrôle sur le processus d'automatisation.

3. La prévisualisation et la particularisation de l'interface générée avec *l'outil de prévisualisation d'interface.*

   Il permet aux développeurs de visualiser l'interface résultant de l'état courant du modèle d'interface.

4. Le test et l'exécution de l'interface résultante avec *le système d'exécution*.

« PROTEGE-II » [49]

Le projet Protégé-II dispose d'accointances avec d'autres projets comme Mecano qui vise toutefois à produire un outil indépendant du domaine qui soit plus général que Protégé-II.
L'architecture de Protégé-II permet aux développeurs de construire des systèmes à base de connaissances en sélectionnant et en modifiant les méthodes de résolution de problèmes ainsi que des ontologies relatives à un domaine. Protégé-II fournit un ensemble d'outils permettant de générer à partir d'ontologies des outils d'acquisition de connaissances propres à un domaine. Ces outils sont implémentés sur NeXT avec le système d'exploitation NeXTStep.

Le système Protégé-II est un ensemble d'outils pour la construction d'applications d'acquisition de connaissances. Ces outils permettent à l'utilisateur de modéliser un domaine et, ensuite, de générer semi-automatiquement une application pour l'acquisition de connaissances sur ce domaine.
Les outils sont les suivants :

- MAÎTRE

  C'est un éditeur d'ontologies pour non-programmeurs permettant de développer des systèmes à base de connaissances. Il permet aux utilisateurs de modéliser leurs connaissances dans un domaine à l'aide d'une hiérarchie de classes dans une structure de type « *frame* ». Le but de MAÎTRE est donc de permettre aux utilisateurs d'éditer et de construire le code MODEL qui peut ensuite être utilisé pour construire les outils d'acquisition des connaissances qui sont spécifiques au domaine.

[49] Se référer au site Web à l'adresse suivante : http://smi-web.stanford.edu/projects/protege/

- DASH [50]

  C'est un concepteur d'agencement et de dialogues qui produit les descriptions pour des outils d'acquisition de connaissances basés sur une ontologie donnée.

  DASH est un outil de méta-niveau qui génère des outils d'acquisition de connaissances à partir d'ontologies de domaine. L'entrée de DASH est une ontologie, qui peut être créée avec MAÎTRE. DASH analyse l'ontologie d'entrée et produit *une structure de dialogue* général pour l'outil d'acquisition des connaissances. Cette structure de dialogue consiste en un graphe où les noeuds représentent des fenêtres et les liaisons représentent la relation d'accessibilité aux fenêtres. Ensuite, DASH conçoit l'agencement des fenêtres spécifiées dans la structure de dialogue. DASH permet au développeur de personnaliser les outils d'acquisition des connaissances à l'aide d'un constructeur d'interface. Initialement, DASH génère un agencement par défaut, que le développeur peut ajuster manuellement. DASH mémorise ces réglages personnalisés de façon persistante dans une base de données. Dans les régénérations sous-jacentes de l'outil d'acquisition des connaissances (par exemple, motivées par des changements sur l'ontologie d'entrée), DASH applique de nouveau l'agencement mémorisé dans cette base de données.

  La sortie de DASH est une spécification d'un outil d'acquisition des connaissances. MEDITOR et MART peuvent opérationnaliser cette spécification.

- MEDITOR

  C'est un système de gestion d'interface qui produit des outils d'acquisition de connaissances. MEDITOR est un programme qui lit des spécifications (fichiers EO) d'outils d'acquisition de connaissances, et produit des outils d'acquisition de connaissances opérationnels. Le développeur peut utiliser l'outil de méta-niveau DASH pour générer les fichiers EO appropriés à partir des ontologies de domaine. La fonctionnalité de MEDITOR est analogue au comportement d'un interpréteur dans le sens où MEDITOR lit le fichier EO et le convertit dynamiquement aux fenêtres et widgets.

- Les outils KA

  Ce sont les outils d'acquisition de connaissance produits par MEDITOR.

- MART

  MART est similaire à un compilateur dans le sens où il produit un programme exécutable à partir de la spécification.

[50] Pour davantage informations sur DASH et PROTEGE-II, se référer à [Eriksson et Musen, 1993] et [Puerta *et al.*, 1992].

- MARBLE

  C'est un outil qui supporte les liens (par « *mapping* ») entre les ontologies de domaine et les ontologies de méthode.

- MODEL

  C'est un langage simple de représentation des connaissances basé sur une hiérarchie de classes, chacune étant un « *frame* ». MODEL est un sur-ensemble du langage orienté objet COOL, qui est implémenté dans le langage de programmation CLIPS.

  En plus de la réutilisation dans PROTEGE-II, les connaissances de domaine représentées dans MODEL peuvent être réutilisées par d'autres systèmes à base de connaissances. Ce type de réutilisation entre système est notamment le but du système « O*ntolingua* » [Gruber, 1992].

Ainsi, les interactions principales entre les outils fournis dans PROTEGE-II sont établies en utilisant le panneau « ProteView » :

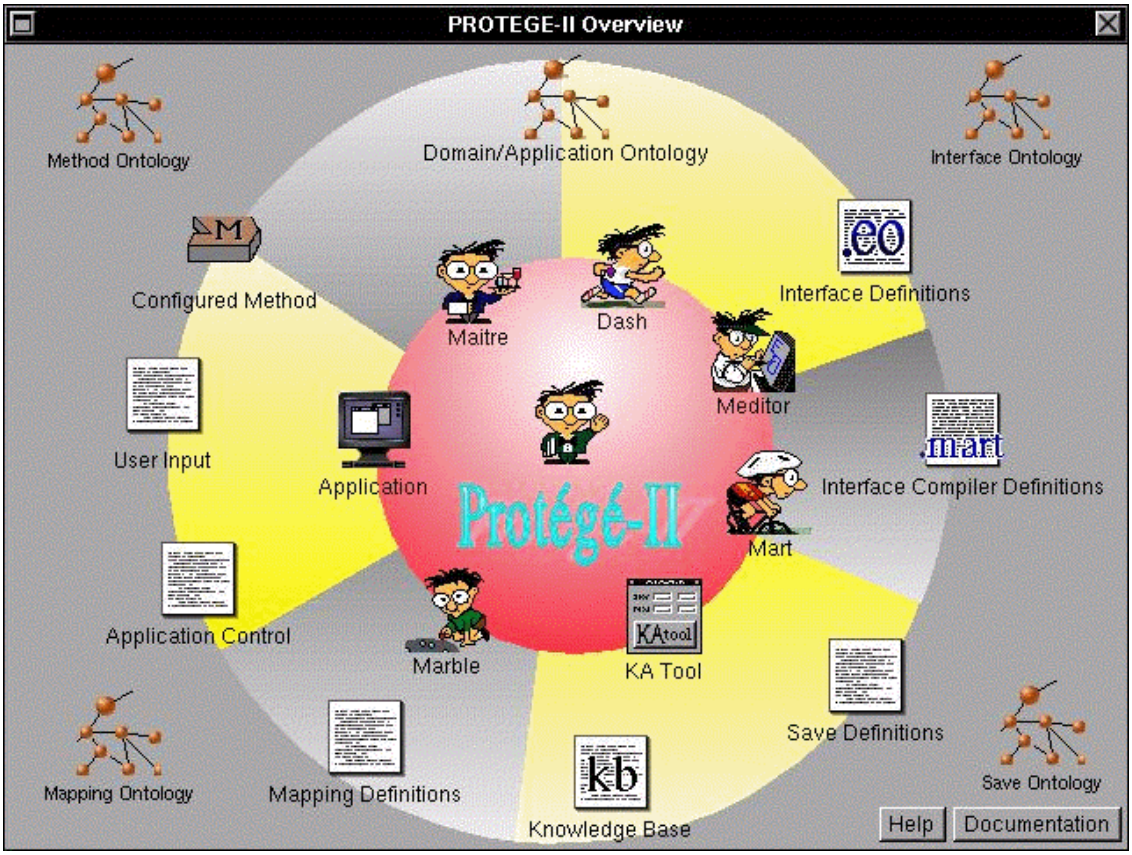

Figure 2 : Panneau de « ProteView »

« PROTEGE/Win » [51]

Protégé/Win est une ré-ingénierie du système Protégé-II pour les environnements Windows NT et Windows 95 [52].

On distingue donc trois étapes principales :

- La première étape consiste à modéliser le domaine du problème grâce à un outil appelé « *OntologyEditor* » qui assiste cette modélisation. Ce programme est un éditeur graphique pour un langage de modélisation. Ce langage permet à un utilisateur de décrire la structure d'un domaine (c'est à dire de construire une *ontologie*) comme une hiérarchie de classes.
- Le « *LayoutEditor* » prend ensuite une ontologie construite et génère automatiquement le masque de l'interface-utilisateur pour l'application d'acquisition des connaissances correspondante. L'utilisateur peut aussi appliquer le « *LayoutEditor* » pour modifier et étendre l'interface-utilisateur de l'application générée.
- Quand l'utilisateur est satisfait de l'agencement produit, le « *LayoutInterpreter* » exécute l'application et collecte les instances des classes de domaine. L'utilisateur peut mémoriser ces instances sur disque, et l'application peut les recharger ultérieurement pour consulter et éditer.

OntologyEditor

« *OntologyEditor* » est le nom d'un éditeur graphique pour un sous-ensemble du langage orienté objet de CLIPS (COOL). Ce langage permet à un utilisateur de modéliser un domaine comme un ensemble de classes, liées par héritage et pouvant contenir tout nombre de champs (appelés attributs ou « *slots* »). « *OntologyEditor* » permet à l'utilisateur de créer, charger, éditer, et sauvegarder des ontologies. Utiliser « *OntologyEditor* » est une alternative facile pour construire une ontologie avec un éditeur de texte. En effet, « *OntologyEditor* » évite à l'utilisateur de devoir connaître la syntaxe de COOL, et aussi détecte immédiatement les erreurs sémantiques et conflits. En outre, un éditeur graphique peut afficher la hiérarchie des classes selon une forme dans laquelle les relations d'héritage entre classes sont évidentes. L'éditeur peut aussi visualiser tout attribut qu'une classe hérite de ses parents ; cette information est en effet difficile à extraire pour un utilisateur en contrôlant l'ontologie avec un éditeur de texte.

[51] Se référer au site Web à l'adresse suivante : http://smi-web.stanford.edu/projects/prot-nt/

[52] Les auteurs de Protégé/Win n'ont pas toutefois encore commencé le travail sur le remplacement de la portion de Protégé-II concernant les méthodes de résolution de problème.

LayoutEditor

« *LayoutEditor* » est le nom d'un outil qui génère (un agencement pour) une interface utilisateur pour une application d'acquisition de connaissances à partir d'une ontologie créée par « *OntologyEditor* ». Le « *LayoutEditor* » génère un masque de saisie pour chaque classe dans l'ontologie auquel il ajoute des commandes de contrôle qui sont appropriées en fonction des attributs de cette classe. Le « *LayoutEditor* » permet aussi à l'utilisateur de modifier l'agencement du masque de saisie en changeant la taille et la position des boutons de contrôle, ainsi que le type de contrôle qui correspond à un attribut spécifique.

Ces moyens de personnalisation sont nécessaires pour deux raisons. Tout d'abord, l'algorithme d'agencement pour les « *widgets* » générés est brut, et les imprimés résultants ne sont pas esthétiques. Ensuite, le langage de modélisation du domaine n'est pas suffisamment riche pour saisir les informations nécessaires pour spécifier uniquement le type de contrôle qui devrait être généré pour chaque attribut. Par exemple, un attribut utilisé pour désigner la couleur aurait un ensemble de valeurs permises (par exemple : rouge, bleu, et vert). Le choix d'une couleur correspondant à cet attribut pourrait être présenté à l'utilisateur soit comme un ensemble de boutons « radio » étiquetés avec les couleurs, soit comme une liste de sélection. L'ontologie ne contient pas assez d'informations pour décider automatiquement lequel de ces choix est le plus approprié. Ainsi, quand plusieurs commandes de contrôles équivalentes pour un attribut sont possibles, le « *LayoutEditor* » permet à l'utilisateur de décider.

LayoutInterpreter

« *LayoutInterpreter* » est le nom d'un outil qui peut lire et exécuter l'agencement généré par le « *LayoutEditor* ». Chaque masque de saisie défini par le « *LayoutEditor* » devient un dialogue dans l'application d'acquisition des connaissances qui est interprétée. Quand l'utilisateur saisit des informations dans ces imprimés, des instances des classes de l'ontologie sont créées. Cet ensemble d'instances peut être mémorisé sur disque, et peut être rechargé ultérieurement.

La seule alternative pour interpréter l'agencement d'acquisition des connaissances est de générer une application native pour le système d'exploitation sur lequel l'outil doit s'exécuter. Cette alternative requiert de distribuer avec le système Protégé/Win un ensemble prohibitif de logiciels (compilateur, éditeur de liens, et bibliothèques interface utilisateur graphiques) pour que des applications puissent être produites.

Le « *LayoutInterpreter* » ressemble beaucoup au « *LayoutEditor* », dans la mesure où il y a une boîte de dialogue pour chaque classe de l'ontologie. La différence est que, dans le « *LayoutInterpreter* », les

boutons de contrôles sont fonctionnels et peuvent être utilisés pour acquérir l'information, tandis que, dans le « *LayoutEditor* », les boutons de contrôle peuvent plutôt être redimensionnés et repositionnés.

Exemple d'utilisation de Protégé/Win

Ce logiciel se compose de trois éditeurs :

- un éditeur d'ontologies (« *Ontology Editor* »),
- un éditeur de mise en page de ces ontologies (« *Layout Editor* »)
- un interpréteur de ces ontologies (« *Layout Interpretor* »)

L'éditeur d'ontologies

Il permet à l'utilisateur de définir le contexte dans lequel se situe l'application. Ainsi, il est possible de mettre en place sous forme d'arbre les différentes ontologies présentes.
Une fenêtre de l'éditeur d'ontologie est divisée en deux parties. La partie gauche permet de visualiser l'information de classe : la documentation et hiérarchie de classes pour une classe choisie. La partie droite affiche les informations relatives aux attributs.

1. Partie Classe

La partie supérieure gauche de la fenêtre d'ontologie visualise la hiérarchie de classes. Les classes héritées paraissent au dessous et sont indentées par rapport à leurs parents. Vous pouvez sélectionner une classe en cliquant sur l'image graphique associée à cette classe.
Vous pouvez choisir de visualiser toutes les classes dans l'ontologie, ou bien cacher les enfants de tout un ensemble de classes. Toute documentation qu'un utilisateur a associé avec une classe sera visualisée dans la partie basse gauche quand cette classe sera sélectionnée. Cette partie affiche aussi une liste de parents pour la classe choisie, si cette classe a plus de un parent.

2. Partie Attribut

Les attributs pour la classe choisie sont listés, un par ligne, dans la partie supérieure droite de la fenêtre d'ontologie. L'ensemble des attributs d'une classe sont les attributs définis dans cette classe, ainsi que tous les attributs hérités de tous les parents de cette classe. Les attributs hérités sont grisés

dans l'affichage. Chaque attribut peut avoir plusieurs propriétés, ou facettes. Deux facettes sont obligatoires pour chaque attribut : la cardinalité et le type.

Les valeurs pour les facettes de cardinalité et de type sont codées dans le graphisme (l'icône) visualisé à la gauche du nom de l'attribut. La cardinalité peut être simple ou multiple. Les valeurs permises pour la facette de type sont : booléen, réel (« float »), entier, instance, chaîne de caractères, et symbole.

Les autres facettes pour chaque attribut sont listées comme texte à droite du nom de l'attribut. Le décodage de ce texte dépend de la valeur du type de la facette pour cet attribut. Par exemple, un attribut entier peut avoir une facette de capacité qui décrit la capacité permise de la valeur de cet attribut. Un attribut de symbole aura une facette de symboles autorisés, qui est une liste des valeurs permises pour cet attribut.

Ceci donne par exemple pour un domaine relatif au « *guideline* » :

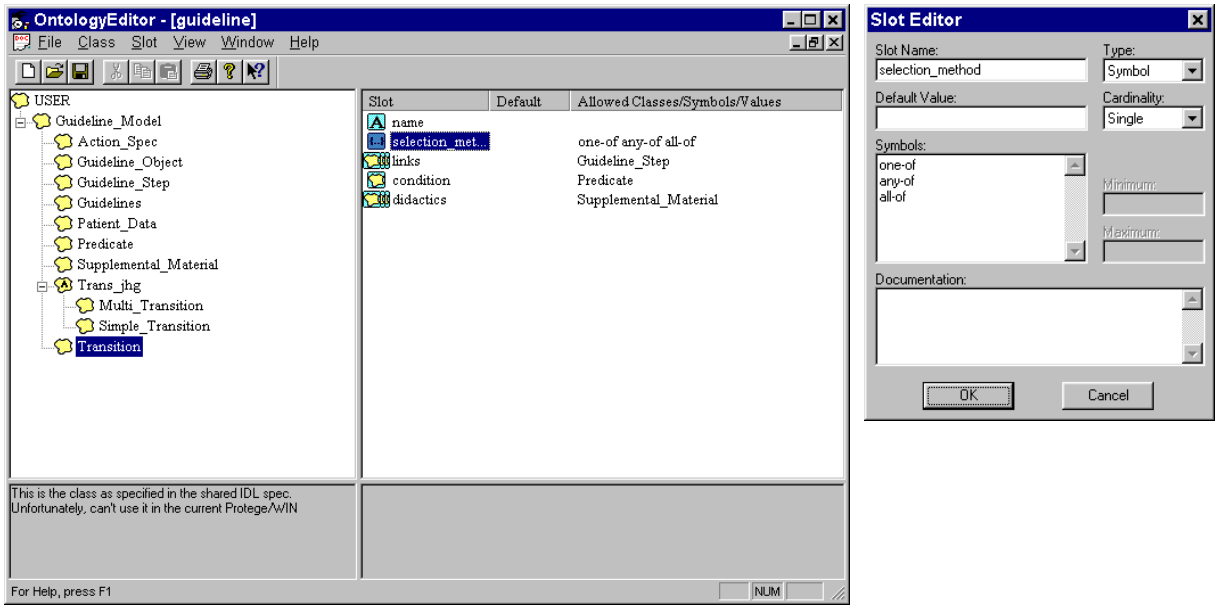

Figure 3 : Exemple d'utilisation de « Ontology Editor »

La sauvegarde du travail s'effectue ensuite sous forme de base de données (menu <File> <Save as Database>) ou bien sous forme de fichier (menu <Save As File…> : «<Nomfichier>.pont » pour « *Protégé ONTologies files* »).

L'éditeur de mise en page de ces ontologies

Vous devez d'abord charger l'ontologie avec l'éditeur d'ontologies (« *Ontology Editor* ») puis le lier (<Link> dans le menu <File>) avec l'éditeur de mise en page à l'aide d'une boîte de dialogue qui vous permet de sélectionner l'ontologie à laquelle vous voulez vous connecter. Quand vous sélectionnez une ontologie, une mise en page de cette ontologie sera générée automatiquement.

Le *«Layout Editor»* a deux types de fenêtres de visualisation. La fenêtre intitulée « *Form Hierarchy* » donne un aperçu général de l'application en visualisant l'arbre d'appel d'applications. Les autres fenêtres sont des fenêtres qui seront l'interface-utilisateur. Celles-ci affichent les commandes associées à une classe dans l'ontologie. Elles permettent à l'utilisateur de changer l'agencement et le type des commandes.

Une classe avec un grand nombre de « slots » peut générer un masque de saisie encombrant. Vous pouvez les découper en plus petites pièces en créant des sous-boîtes (« *subforms* ») liées au masque de saisie principal. Le *«Layout Editor»* générera automatiquement un bouton d'accès vers le masque de saisie principal.

Si l'on reprend l'exemple précédent de « *guideline* », cela donne :

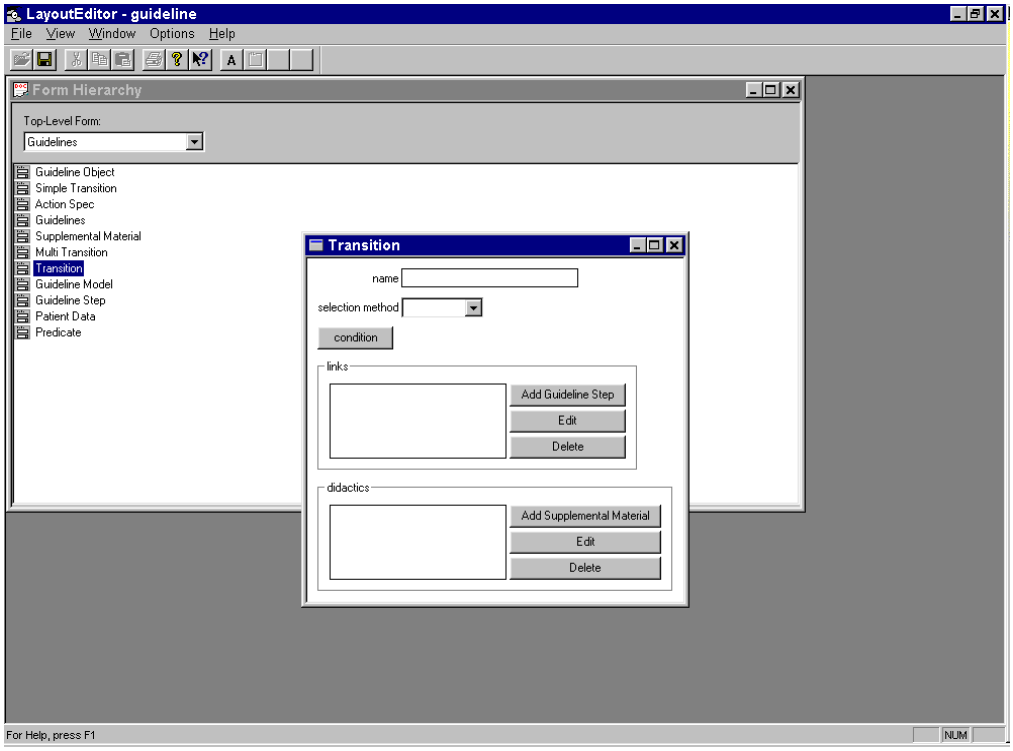

Figure 4 : Exemple d'utilisation du « Layout Editor »

Puis, une fois la mise en forme effectuée, on obtient par exemple :

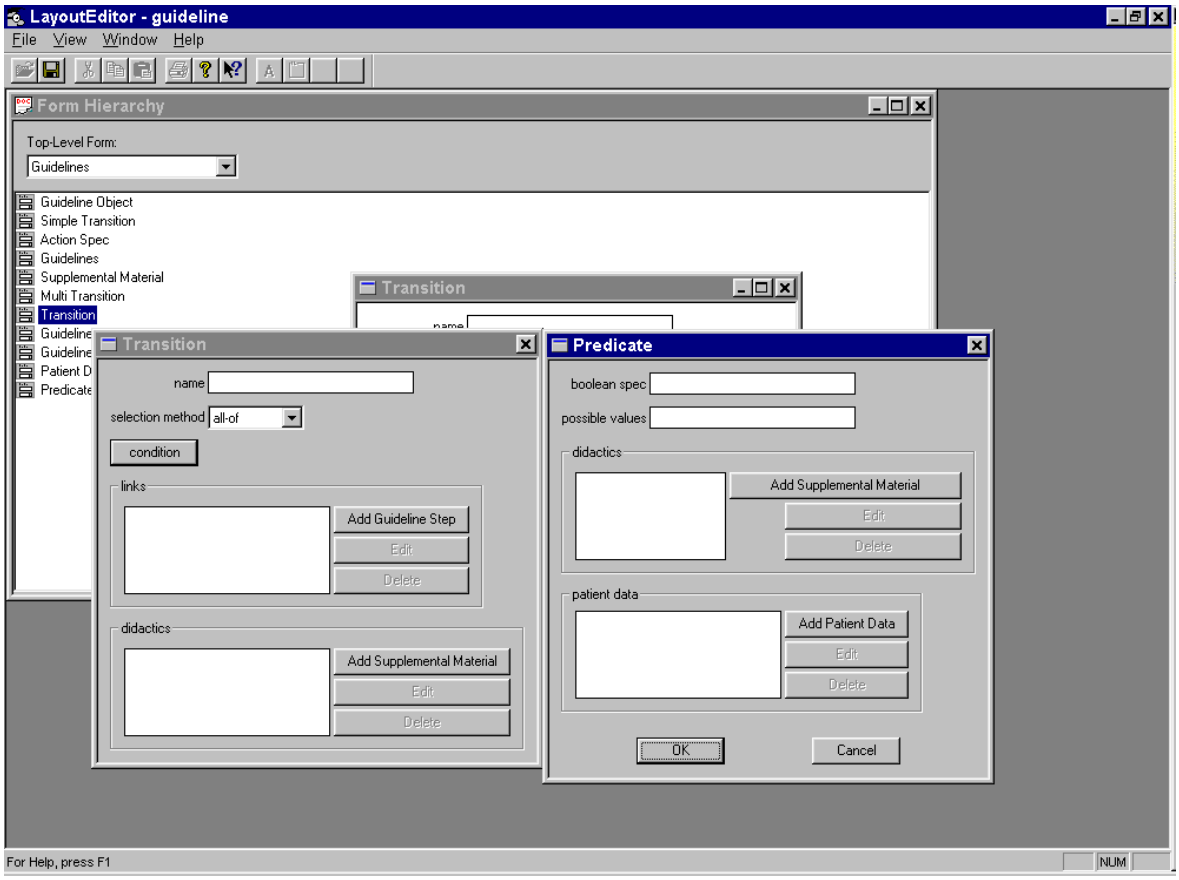

Figure 5 : Autre exemple d'utilisation du « Layout Editor »

La sauvegarde (menu <File> <Save As…> se fait dans un fichier dont l'extension est «.pkat » pour Protégé Knowledge Acquisition Tools.

L'interpréteur d'ontologies

Dans cette dernière étape, il convient de créer une nouvelle base de connaissances à partir des informations précédemment saisies.

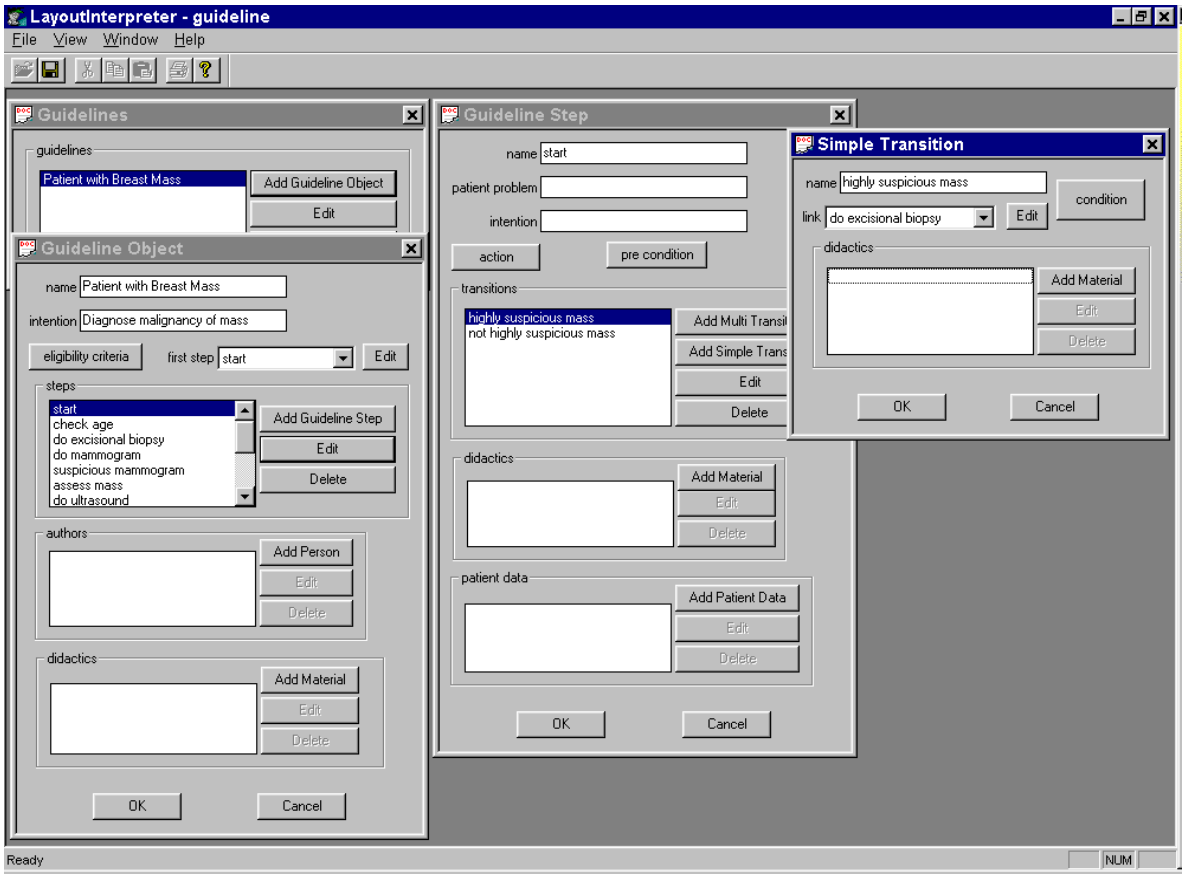

Figure 6 : Exemple d'utilisation du « Layout Interpretor »

La sauvegarde (menu <File> <Save As…>) se fait dans un fichier dont l'extension est «.pins » pour Protégé INStance Files.

Annexe 2 :

Des exemples d'outils méta-CASE

«ConceptBase» [53]

«ConceptBase » est un gestionnaire objet déductif et multi-utilisateurs destiné essentiellement pour la modélisation conceptuelle et la coordination dans des environnements de conception. Le système implémente « *O-Telos* », un dialecte de « *Telos* », qui effectue l'amalgame des propriétés de langages orientés objets et déductifs.

L'ensemble des classes, méta-classes, instances, attributs, règles, contraintes, et requêtes sont uniformément représentés comme des objets. Tout objet peut être mis à jour à tout moment dans le cycle de vie de la base d'objets. Même l'appartenance à une classe est considérée comme un objet et est sujette aux mises à jour.

La possibilité de méta-modélisation permet de représenter des langages hétérogènes de modélisation comme les diagrammes entité-association, OMT, etc… en tant que méta-classes « O-Telos » dont les règles et contraintes encodent les axiomes de chaque langage, par exemple la généralisation. Les méta-classes peuvent co-exister dans la même base objet. Les objets décrits dans un langage de modélisation peuvent être explicitement liés aux objets décrits dans un autre langage de modélisation.

«ConceptBase » respecte une architecture client-serveur. Les programmes clients peuvent se connecter au serveur « *ConceptBase »* et échanger des informations avec le processus de communication du protocole Internet TCP/IP.

L'interface de progammation de *«ConceptBase »* permet aux utilisateurs de créer leurs propres programmes clients en C ou Prolog. L'interface utilisateur de type X11 de *«ConceptBase »* offre une palette d'outils textuels, tabulaires, et graphiques pour éditer et parcourir la base d'objets.

[53] Se référer au site Web à l'adresse suivante : http://www-i5.informatik.rwth-aachen.de/CBdoc/cbflyer.html

« GOODSTEP » [54]

« *GOODSTEP* » est l'acronyme de « *General Object-Oriented Database for Software Engineering Processes* » (ESPRIT-III Project 6115).

L'objectif de « *GOODSTEP* » est d'améliorer et d'étendre la fonctionnalité d'un système de gestion de base de données pleinement orienté-objet pour produire une plate-forme correspondant aux applications telles que les ateliers de génie logiciel.

La base du projet est le système de gestion de base de données O2, qui inclut déjà beaucoup des fonctionnalités nécessaires. Le consortium a identifié les améliorations O2 nécessaires pour supporter pleinement la prochaine génération prévue d'ateliers de génie logiciel, telles que les déclencheurs (« *triggers* ») et les contraintes d'intégrité, la gestion des changements, les nouveaux mécanismes de transactions évoluées, des bibliothèques de classes spécialisées, la version (« *versioning* ») d'objets, les possibilités déductives, une interface graphique évoluée et un environnement d'aide à la programmation.

Ces outils supportent la modélisation conceptuelle durant la phase d'analyse d'un projet, la conception modulaire et la programmation déclarative. En outre, des générateurs d'outils sont développés pour exploiter les caractéristiques fournies par la base de données O2 améliorée.

Le système « *GOODSTEP* » est en cours de validation au travers de deux études de cas impliquant « British Airways » au Royaume-Uni et « Engineering » en Italie, qui font une utilisation pleine de la plate-forme développée.

« HOTDRAW » [55]

« *HotDraw* » est un système d'infographie bi-dimensionnelle écrit en Smalltalk pour des éditeurs de dessins structurés. Il a été utilisé pour créer beaucoup d'éditeurs différents, depuis des outils CASE jusqu'à un clone « *HyperCard* ». « *HotDraw* » permet de créer aisément de nouveaux dessins (éventuellement animés) et des outils spéciaux de manipulation pour vos dessins.

Les applications de « *HotDraw* » sont :

---

[54] Se reporter au site Web à l'adresse suivante :
http://www.dbis.informatik.uni-frankfurt.de/WF/Hall1/Overview.html

[55] Se référer au site Web à l'adresse suivante : http://st-www.cs.uiuc.edu/users/brant/HotDraw/HotDraw.html

- « *DrawingInspector* », programme qui est un inspecteur visuel pour Smalltalk.
- « *HotPaint* », simple programme de dessin.
- « *MovingDrawing* », programme d'animation simple de rectangles.
- « *NetworkEditor* » utilisant des animations pour résoudre des problèmes de rang -n (on spécifie les nœuds et poids sur ces nœuds).
- « *ObjectWorld* » permet d'enseigner des bases de Smalltalk.
- « *PERTChart* » permettant de créer et d'éditer des graphes de PERT.

« Metaview » [56]

Le système méta-CASE développé par le groupe de génie logiciel au département informatique de l'université d'Alberta s'appelle « *Metaview* ». C'est un système multi-utilisateur Unix (X Windows), avec des possibilités d'édition graphique.

Dans « *Metaview* », la consistance et la complétude de la spécification du logiciel sont contrôlées par des *contraintes*, qui peuvent être définies pour chaque méthode. Les contraintes sont conditions logiques qui doivent être satisfaites par la base de données de spécification. Les méthodes de développement utilisées sont modélisées avec un méta-modèle spécialement conçu et appelé « EARA/GE », acronyme de *« Entity - Aggregate - Relationship - Attribute with Graphical Extension* ». Le méta-modèle est conçu pour la modélisation de plusieurs méthodes de développement de logiciel. Comme son nom le suggère, « EARA » est basé sur le modèle de données entité-association, mais il inclut des extensions significatives, comme la spécialisation, l'agrégation et certains éléments de généralisation. Il est utilisé pour modéliser la partie conceptuelle des méthodes. Le méta-modèle a été créé à l'origine par [MacAllister, 1988]. Depuis lors, le méta-modèle a constamment évolué. La version actuelle de « EARA » est formellement décrite dans [Findeisen, 1994]. Le modèle « EARA » permet de représenter de façon uniforme à la fois une spécification particulière de logiciel et la méthode de développement utilisée pour construire la spécification. Ainsi, il y a deux niveaux dans le modèle « EARA » :

- le *niveau environnement* qui est utilisé pour décrire formellement diverses méthodes, et
- le *niveau spécification* qui est utilisé pour définir les conceptions réelles, ou, en général, les spécifications réelles de logiciel.

[56] Se référer au site Web à l'adresse suivante : http://web.cs.ualberta.ca/~softeng/Metaview/project.shtml

Par convention, une implémentation du modèle « EARA » inclut une base de données stockant les deux parties de l'application réelle.

Le modèle de données « EARA » est insuffisant pour assurer la « correction » d'une spécification. Par conséquents les modèles de méthodes de développement sont complétés avec des ensembles de *contraintes*, ou prédicats logiques, qui doivent être satisfaits par toute spécification « correcte » de logiciel. Il y a aussi quelques *méta-contraintes*, qui sont applicables à toutes les spécifications, indépendament du modèle utilisé. Les contraintes se classent dans une des deux catégories suivantes :

- *les contraintes de consistance*, qui gardent la consistance de la base de données de spécification, c'est à dire qui contrôlent que la spécification a du sens (« *makes sense* »),
- *les contraintes de perfection*, qui assurent que la spécification du logiciel est complète, c'est à dire qui contrôlent qu'il n'y a aucune information manquante.

Les méthodes peuvent être décrites avec un *langage* d'analyse spécialement conçu appelé langage de définition d'environnement (« *Environment Definition Language* » ou EDL) [MacAllister, 1988], [Gadwal *et al.*, 1993]. Les caractéristiques du langage correspondent strictement aux concepts contenus dans « EARA/GE ».
Pour construire des contraintes, les auteurs ont développé un langage de contraintes sur l'environnement (« *Environment Contrainsts Language* » ou ECL).
Les compilateurs EDL et ECL produisent un code objet, qui est ensuite mémorisé dans la bibliothèque de méthodes [57].
Pour modéliser les aspects graphiques des méthodes, les auteurs utilisent l'extension graphique (« *Graphical Extension* » ou GE) [Findeisen, 1994] pour le méta-modèle. Ceci permet de définir la représentation graphique des éléments conceptuels définis dans « *EARA* ».
Pour supporter la conversion automatique d'une spécification créée avec une méthode de développement vers la spécification équivalente décrite dans des termes d'une autre méthode, les auteurs ont conçu un langage de transformation d'environnement (« *Environment Transformation Language* » ou ETL) [Gadwal et Boloix, 1993].

Les définitions conceptuelles incluent des définitions des éléments du méta-modèle « EARA » : les types d'agrégat, de relation et d'entité, leurs attributs, les agrégations possibles et diverses associations entre ceux-ci. Les définitions conceptuelles peuvent être exprimées avec le langage de définition

[57] C'est une base de données dans laquelle plusieurs méthodes de développement modélisées avec EARA/GE peuvent résider. Une définition de méthode peut être extraite de la bibliothèque pour être utilisée dans la configuration du système « Metaview ». La bibliothèque de méthodes développée par le « Software Engineering Group » contient les DFD, des méthodes OO dont OMT, la méthode temps-réel Ward-Mellor, Telos…

d'environnement (EDL), ou bien les tables d'objets peuvent être créées manuellement (en utilisant un éditeur de texte).

Les contraintes conceptuelles sont un ensemble de prédicats qui définissent la consistance, la complétude, et, à un certain degré, la « correction » de la spécification logicielle préparée dans l'environnement donné. Les contraintes de consistance sont contrôlées après chaque changement de la base de données. Les transactions violant la consistance sont rejetées. Les contraintes de complétude sont contrôlées sur la demande explicite. Les contraintes peuvent être arrangées en niveaux. Le serveur de base (« *repository server* ») garantit qu'aucune contrainte d'un niveau donné n'est contrôlée avant que toutes les contraintes des niveaux inférieurs ne soient satisfaites. Les contraintes peuvent être exprimées avec le langage ECL.

Les définitions graphiques définissent l'aspect des objets lorsqu'ils sont présentés graphiquement.

Les définitions graphiques peuvent être préparées :

- dans une forme textuelle utilisant un langage de haut niveau (EDL),
- dans une forme textuelle utilisant le format des tables d'objets,
- en utilisant graphiquement un outil interactif appelé « *Graphical Definer* » (*encore en construction*)

Les contraintes graphiques sont un ensemble de prédicats qui permettent la consistance graphique d'objets. Elles ont pour but d'assurer que la représentation graphique soit lisible et en conformité avec la spécification conceptuelle mémorisée dans la base de données. Les contraintes graphiques peuvent être codées avec le langage ECL.

« HARDY » [58]

L'outil méta-CASE « *HARDY* » fait l'objet de l'annexe 5 de ce document.

[58] Se reporter au site Web à l'adresse suivante : http :// www.aiai.ed.ac.uk/~hardy/hardy/hardy.html

« METAEDIT » [59]

« *MetaEdit™ Personal 1.2* » est un outil méta-CASE dont la puissance repose sur un support multi-méthodes personnalisable et sur la génération de rapports personnalisés par l'utilisateur.
« *MetaEdit Personal* » est un outil méta-CASE graphique sous Windows qui contrôle automatiquement la consistance des modèles construits. « *MetaEdit* » permet de générer, grâce au langage de définition de rapport (« *Report Definition Language* »), des résumés, rapports de validation, définitions de données et même de code, directement à partir des modèles développés. « *MetaEdit* » inclut de façon prédéfinie la génération de code SQL et Smalltalk.
« *MetaEdit* » fournit le support des méthodes de modélisation traditionnelles (SA-SD, SA-RT, Jackson, modèle E-A, etc...) et orientées-objets (OMT, OOA, Booch, Moses, etc...).

« GRAPHICAL DESIGNER 2.0 » [60]

« *Graphical Designer 2.0* » fournit aux développeurs de logiciels professionnels des outils puissants d'aide à la conception et à la ré-ingénierie. Ces produits sont conçus pour aider les développeurs tout au long du cycle de vie du logiciel, et ce d'une façon intégrée avec les autres outils existants dans l'environnement. La famille « *Graphical Designer* » consiste en trois produits : « GDDraw », « GDPRO » et « GDMETHODBUILDER » qui s'exécutent dans un environnement multi-plateformes incluant UNIX et Windows NT / 95 et supportent une conception collaborative multi-utilisateurs.
« GDDraw » fournit de puissantes possibilités de dessin, incluant des palettes de dessin définies par l'utilisateur.
« GDPRO » fournit un support plein pour une variété de méthodologies de conception structurées et orientées objets (Use Cases, UML, OMT, Booch, et OOA), et inclut la génération de code et le reverse-engineering (Java, C++ et Smalltalk).
« GDMETHODBUILDER » permet aux utilisateurs d'étendre tous les aspects des méthodologies de conception, ou d'en construire des entièrement nouvelles. « *GDMETHODBUILDER* » fournit aussi un puissant macro langage de script nommé GDL. Ce langage est un langage propriétaire, simple

[59] Se reporter au site Web à l'adresse suivante : http ://www.jsp.fi/metacase/
[60] Se reporter au site Web à l'adresse suivante : http://davinci2.csn.net/~jefscot/index.html

similaire à d'autres langages de macros. Il permet la consultation et la mise à jour pleine du « *repository* » GDPRO.

« OBJECT MAKER » [61]

ObjectMaker® TDK™ est un méta-outil supportant de nombreuses méthodes de génie logiciel (OMT, Booch, Colbert, Fusion, OOA, etc…) et de traitement des processus (IDEF, ProNet, ProcessMapper, RINS, RADS).

Le support pour OML (« *OPEN Modeling Language* ») est également disponible [62]. OML supporte le cycle entier de développement orienté objet. Il a été conçu pour être beaucoup plus facile à apprendre que les notations traditionnelles, en étant plus intuitif et en requérant moins de mémorisation de sigles. A l'heure actuelle, OML a été adopté formellement par Donald Firesmith, Ian Graham, Brian Henderson-Sellers, et Meilir Page-Jones.

« INTERSOLV PVCS » [63]

INTERSOLV PVCS est le standard industriel pour la gestion de configuration logicielle sur des réseaux LAN avec une famille de solutions SCM qui peuvent être installées et utilisées isolément ou ensemble à tout moment. Il offre des possibilités pour le suivi et l'évolution de requêtes de changement et d'analyse pistant, la gestion de version, et la gestion distribuée.

PVCS revêt la plus large couverture de capacité et de systèmes d'exploitation, d'environnements de développement et de types d'objets pour améliorer la réutilisabilité et réduire les besoins en formation sur de nouveaux outils. PVCS est testé et éprouvé mondialement par des milliers de développeurs.

[61] Se reporter au site Internet de Mark V Systems à l'adresse suivante : http://www.markv.com/

[62] Pour de plus amples renseignements sur OPEN et OML, se reporter à l'adresse suivante : http://www.csse.swin.edu.au/cotar/OUVERT.

[63] Se reporter au site Web à l'adresse suivante : http://www.intersolv.com/solution/scm_top.htm

« IPSYS TBK » [64]

La boîte à outils IPSYS couvre et supporte le cycle de développement complet. L'ensemble des produits disponibles sont :

- Pour des applications client-serveur à grande échelle : « *IPSYS OBJECT IE* »
  « *IPSYS OBJECT IE* » est la boîte à outils supportant le cycle de vie complet pour une application client-serveur : planification, analyse, conception et génération de code. La méthode supportée est basée sur la méthode « *Engineering Information* » à laquelle ont été ajouté un certain nombre d'extensions orientées objets, particulièrement dans les domaines de spécification d'applications et de conception physique. La génération d'applications est disponible pour un nombre important de SGBD Relationnelles incluant Oracle et Sybase avec du SQL inclus dans du C ou C++, pour des interfaces-utilisateurs graphiques Windows ou Motif, et sur des plates-formes PC et Unix.
- Pour des systèmes temps réel : « *IPSYS HOOD Toolset* »
  La méthode HOOD (« *Hierarchical Object Oriented Design Method* ») a été spécifiée par l'agence spatiale européenne (ESA) pour fournir une approche convenable pour le développement de logiciel, principalement en Ada, mais encore en C++.
- « *IPSYS SSADM Toolset* »
  « *IPSYS SSADM Toolset* » fournit un environnement CASE à la fois complet, multi-utilisateurs et multi-projets pour « *SSADM 4+* » avec de nombreuses extensions.
- Pour construire des outils orientés objets : « *ToolBuilder* »
  « *IPSYS ToolBuilder* » permet de définir et représenter rapidement et facilement les méthodes et outils nécessaires pour le développement de logiciels comme par exemple IE, HOOD et SSADM ou toute autre approche spécifique de développement de logiciels (OMT, OOA, etc..). Tous les outils développés avec « *IPSYS ToolBuilder* » ont un « *repository* » distribué, pleinement multi-utilisateurs pour lequel il est possible d'avoir des vues homogènes graphiques et textuelles avec une navigation hypertextuelle rapide entre des « *frames* » d'information.

Dans « *ToolBuilder* », la base de données commune (« *repository* ») est un dictionnaire de données décrit en DDL (langage de description de données) ; l'interface utilisateur est décrite en FDL (langage de description de format) ; la description des graphiques décrivant tous les divers types de forme, ligne

[64] Se reporter à l'adresse Internet suivante : http://www.ipsys.com/

et texte qui peuvent être utilisés dans un diagramme et les relations entre ceux-ci se fait avec GDL (langage de description de graphique).

Ainsi, « ToolBuilder », disponible sur les plates-formes Windows NT et 95, offre les possibilités suivantes :

- des graphiques et du texte intégrés,
- un accès multi-utilisateurs en ligne et distribué du « *repository* »,
- le contrôle de cohérence,
- la génération de documents,
- la génération de code,
- une aide en ligne,
- de l'import-export intelligent permettant un travail multi-sites et multi-projets.

Annexe 3 : Quelques exemples de systèmes et langages basés sur les paradigmes « objet » et « logique »

Nous présentons ci-dessous [65] quelques exemples issus de nombreuses lectures relatives au couplage des paradigmes objet et logique, des plus simples au plus élaborés aboutissant à des systèmes d'objets actifs ou d'acteurs.

TROLL

TROLL [Jungclaus *et al.*, 1996] est un langage convenant parfaitement aux premiers stages du développement des systèmes d'informations, lorsque l'univers du discours doit être décrit. Avec TROLL, les descriptions des aspects statiques et dynamiques des entités sont intégrées dans les descriptions des objets. Pour décrire les propriétés statiques, le comportement et l'évolution dans le temps des objets, des sous-langages sont utilisés respectivement pour les données, les assertions temporelles et du premier ordre, et les processus. TROLL organise la conception du système grâce à une approche orientée-objet et au support d'abstractions telles que la classification, la spécialisation, les rôles [Pernici, 1991] et l'agrégation.

TROLL est un *langage de spécification* pour la modélisation conceptuelle de systèmes d'information qui fournit :

- des concepts de structurations en entités/objets issus des modèles sémantiques de données [Ehrich et Mahr, 1985] [Engels *et al.*, 1992] ;

[65] La présentation demeure relativement succinte. Pour de plus amples informations, il faut se reporter aux références bibliographiques

- des mécanismes de structuration formelle des modèles, mécanismes issus de la spécification algébrique [Peckham et Maryanski, 1988] [Ehrich *et al.*, 1988] ;
- des spécifications orientées-événements et orientées-processus ainsi que la spécification de l'évolution temporelle. Ces spécifications sont issues du domaine des systèmes concurrents et distribués [Hoare, 1985] [Milner *et al.*, 1989] et aussi des systèmes réactifs [Emerson, 1990] [Pnueli, 1986].

AMS

[Ang et Hong, 1994] décrit un système de gestion des activités (« *Activity Management System* » ou AMS) qui est un formalisme indépendant du domaine parfaitement adapté à la représentation hiérarchique de la connaissance procédurale.
Les caractéristiques du système sont les suivantes :

- Comme dans le formalisme à base de règles, le *contrôle* est séparé de la *connaissance*, et la programmation est remplacée par une représentation déclarative explicite de blocs de connaissances ;
- Le concept de *réseau d'activité* (« *Activity Network* »), permet d'obtenir une représentation hiérarchique explicite de la connaissance procédurale ;
- Le concept d'activité fournit au formalisme une représentation sous forme de règles ;
- Le concept de *blocs d'organisation de la mémoire pour des activités* (« *Memory Organization Packets for Activities* » ou MOPA) permet d'une part d'organiser et de reconstruire la connaissance de façon appropriée et d'autre part autorise la génération dynamique d'un réseau d'activité à divers niveaux d'abstraction.

Point de vue de [Davison, 1991]

D'après les trois façons de coupler les paradigmes objet et logique, [Davison, 1991] ne retient que la programmation logique étendue avec les paradigmes de la programmation orientée-objet.

Dans un effort de clarification, [Davison, 1991] forme quatre groupes de langages selon leurs différentes approches d'implémentation. Ceux-ci sont basés sur quatre vues distinctes :

- la vue des clauses (« *clauses view* »),
- la vue du processus de choix retenu (« *committed choice process view* »),
- la vue sur le processus de retour-arrière (« *backtracking process view* »),
- la vue sur la méta-interprétation (« *meta-interpretation view* »).

Ainsi, selon [Davison, 1991] la meilleure façon de prendre en compte la programmation orientée-objet consiste en une extension syntaxique des clauses logiques de Horn avec des notions de programmation orientée-objet.
Ceci implique notamment que d'autres paradigmes comme la programmation fonctionnelle peuvent être ajoutés et offre également des moyens simples pour le traitement de l'héritage (multiple en particulier), et de l'auto-communication.
Par contre, les langages à base de clauses ne conviennent pas exactement au traitement de la notion d'état, sauf pour la clarification de la sémantique. De même, le retour-arrière cause certains problèmes relatifs à l'annulation de la création d'objets, aux transferts de messages et aux changements d'états.

CORAL++

Coral^{++} [Srivastana *et al.*, 1993] est un langage de programmation de bases de données étendant le paradigme logique de Coral, qui est un langage de requête déductif, avec le paradigme objet de C^{++}.
La conception et la réalisation de Coral^{++} sont basées sur plusieurs points :

- le modèle de données est basé sur C^{++}.
- les définitions de classes et les appels de méthodes sont gérés par le compilateur C^{++}.
- les programmes déclaratifs Coral^{++} sont exprimés en termes de clauses logiques de Horn pouvant faire appel à des méthodes C^{++} sous forme de fonctions externes.

La différence entre Coral^{++} et un langage logique classique est que :

- les données sont persistantes,
- les expressions C^{++} peuvent être utilisées pour envoyer des messages aux objets.

Le traitement d'un programme Coral^{++} combine l'utilisation d'un processeur C^{++} et Coral^{++}. Le choix de C^{++} n'est cependant pas restrictif car selon [Srivastana *et al.*, 1993], tout langage orienté-objet convient.

2MU

[Uustalu, 1994] opte également pour l'extension de la logique avec des constructeurs objets. Selon lui, une différence importante entre les paradigmes objet et logique réside non pas dans leurs divers avantages mais plutôt dans le but pour lequel ils sont utilisés. Ainsi, la logique est perçue comme un formalisme et l'orienté-objet est ressenti comme une technique applicable à différents formalismes. Un objectif est de dégager plusieurs concepts orientés-objets généraux en terme de logique puis de construire un modèle formel d'un schéma orienté-objet basé sur la logique. La solution est davantage théorique que pragmatique, et son objectif est double : obtenir une sémantique de l'orienté-objet en terme de logique, puis considérer les changements structurels obtenus après programmation.

Selon [Uustalu, 1994], les *objets* et les *classes* sont représentés par des prédicats définis par une ou plusieurs clauses dont les arguments sont les *attributs*. Ces derniers sont, de ce fait, soit des variables impératives non logiques, soit des prédicats auxquels sont donnés des valeurs dans les clauses des objets. En outre, la valeur des attributs peut changer. Par conséquent, des constructeurs permettant la création, la suspension, la réactivation et/ou la suppression de prédicats doivent être créés ; ceci aura également pour effet de simuler un changement d'état pour un objet. *L'héritage* est une notion d'objet forte à prendre en compte d'un point de vue logique. Il convient en outre d'étendre cette notion dans la mesure où, si pour une classe donnée, l'invocation d'une méthode échoue, il convient non pas de s'arrêter et de signaler l'erreur, mais de rechercher si cette méthode est présente dans une super-classe dont la classe appartient au contexte (*héritage cumulatif*). Ceci peut être réalisé par des constructeurs spéciaux (« *cut* » en Prolog). Un langage orienté-objet doit posséder des *constructeurs* permettant l'appel explicite aux objets (par exemple, le transfert de message), mais certains appels doivent être masqués (notion d'encapsulation). De même, il doit y avoir des constructeurs permettant les appels implicites, mais aussi le partage (par exemple, l'héritage ou la délégation) contrôlé. [Uustalu, 1994] s'appuie sur la logique modale et fournit une méthodologie basée sur les clauses de Horn et le calcul de Hilbert et propose différentes règles permettant de modéliser des concepts orientés-objets pour un système basé sur la logique des prédicats.

Point de vue de [Fowler et al., 1994]

Selon [Fowler *et al.*, 1994], une des principales difficultés de l'utilisation de techniques de spécifications formelles lors de la phase de définition du cahier des charges réside dans l'encouragement de la participation et dans l'assurance de la mise au point de la clarté et de la précision lors du processus d'acquisition puis de validation des connaissances.
Pour que les utilisateurs participent à ce processus, ils doivent maîtriser certaines notions nécessitant un entraînement plus ou moins important selon leur degré d'engagement. Plus ils s'investissent dans la spécification, plus la validation est crédible et donc la spécification adéquate.
Par conséquent, il est nécessaire dans ce processus de présenter la spécification de façon à faciliter la meilleure compréhension possible. Malheureusement, les techniques traditionnelles en tiennent peu compte car les spécifications formelles sont généralement présentées comme des schémas inclus dans un texte explicatif. Une difficulté principale réside aussi dans la validation des spécifications formelles écrites en Object-Z par des utilisateurs non experts en mathématiques ni en orienté-objet. Ainsi, [Fowler *et al.*, 1994] étendent la notation diagrammatique MOSES [Henderson] permettant de décrire le comportement coopératif et interactif des composants du système.

[Fowler *et al.*, 1994] préconisent donc une notation :

- permettant une représentation dynamique du système,
- simple, graphique et expressive,
- supportant et décrivant une spécification orientée-objet même complexe.

Cette notation doit permettre la prise en compte de systèmes comportant des aspects distribués ou parallèles mais également plus simples.
La notation est composée de trois concepts :

① des diagrammes de vue générale sur la chaîne d'événements,

Chaque chaîne est composée d'un ensemble d'événements combinés pour la description du comportement du système vu par les utilisateurs à un haut niveau d'abstraction.

② des diagrammes de lien représentés par des icônes,

Chaque chaîne est décomposée en liens représentant des transitions entre états mais aussi illustrant les associations entre classes impliquées dans le changement d'états.

③ une carte d'événements.

Elle montre les différentes relations entre liens formant le chemin composant la chaîne. La technique des diagrammes de transition entre états est utilisée ici mais la structure Jackson [Jackson, 1983] conviendrait également.

Selon [Fowler *et al.*, 1994], la documentation d'une spécification doit respecter la structure suivante :

① La spécification doit commencer par une vue générale de la fonctionnalité du système. Cette vue doit être basée sur des diagrammes « libres » (« *free form* »).

② Le système, grâce à la notation précédente, est modélisé de façon à permettre la validation des souhaits exprimés par les utilisateurs. Certaines parties de la spécification Object-Z sont présentées pour montrer comment la spécification modélise formellement le comportement du système tel qu'il est décrit par les diagrammes.

③ Le système est finalement spécifié entièrement en Object-Z. Des diagrammes orientés-objets traditionnels montrant l'agrégation, l'association, l'héritage, etc… sont aussi ajoutés.

Pour chaque objet du système, l'utilisateur complète la définition de chacun de ses états en Object-Z et spécifie les contraintes d'application (données et méthodes). La dernière étape concerne surtout les concepteurs et réalisateurs familiers avec l'orienté-objet et Object-Z. La coordination d'opérations est assurée par le système qui est aussi un objet.

[Fowler *et al.*, 1994] étend maintenant ses recherches vers la génération (semi-automatique) de diagrammes MOSES étendus à partir de spécifications en Object-Z ainsi que vers le développement d'un support d'analyse de cette méthodologie.

Point de vue par acteurs selon [Agha et al., 1993]

Selon [Agha *et al.*, 1993], le *modèle d'acteur,* issu des travaux de [Hewitt, 1977], possède une notion imbriquée de composant local et d'interface, et par conséquent c'est un modèle naturel de base pour un traitement informatique ouvert et distribué. Les acteurs sont des entités autonomes concurrentes communiquant de façon asynchrone par envoi de messages. Ils peuvent être créés dynamiquement, ce qui à pour effet de changer la topologie du système. Le modèle d'acteur est relativement primitif mais permet néanmoins d'exprimer facilement un large éventail de paradigmes. En effet, il supporte l'encapsulation, le partage, et fournit une extension naturelle à la programmation fonctionnelle et objet pour des systèmes ouverts concurrents.

La *philosophie « acteur »* repose sur plusieurs principes :

- Chaque objet est un acteur (y compris les messages et les nombres),
- Un acteur est uniquement accessible par envoi de messages ou plutôt d'événements,
- Un acteur peut envoyer plusieurs messages simultanément mais en recevoir un seul à la fois,

- Les acteurs ont un ensemble d'*accointances*, c'est à dire un contexte représenté par l'ensemble des acteurs qu'il connaît et avec qui il peut communiquer par messages. C'est la seule façon de faire évoluer les accointances d'un acteur.

Ces principes reposent sur plusieurs caractéristiques :

- Le transfert de message est fondamental et garanti explicitement,
- Il y a un ordre d'arrivée non déterministe pour la délivrance de messages,
- Les acteurs sont des objets partagés évolutifs selon leur passé,
- L'allocation de ressource et la topologie d'interconnexion sont dynamiques.

Le langage d'acteur préconisé est une extension du lambda-calcul incluant des primitives de création et de manipulation d'acteurs. Le comportement d'un acteur est décrit par une lambda-abstraction incorporant le code à exécuter lorsqu'un message est reçu.

A l'inverse des notions de modularité et de composition qui sont statiques, les notions d'équivalence et de sécurité sont dynamiques, ce qui permet à l'interface entre composants d'évoluer avec le temps.

Le modèle d'acteur peut être comparé à trois autres modèles gérant la concurrence : CSP [Hoare, 1985], λ-calculus [Milner *et al.*, 1989] et Concurrent ML [Reppy, 1992]. Cependant, aucun de ces modèles ne garantit la sécurité de la spécification. En effet, un processus peut se comporter correctement de façon isolée mais incorrectement en présence d'autres processus.

ALBA

Selon [Hernandez *et al.*, 1993], dans le but de fournir des outils de construction de systèmes homogènes favorisant l'encapsulation et l'écriture modulaire de programmes parallèles et distribués, les modèles orientés-objets ont été couplés aux modèles concurrents d'où l'apparition de modèles de programmation concurrente orientée-objet. Ceux-ci permettent une conception et une réalisation dynamique du système par construction d'un ensemble de modules exécutables concurrents, appelés *objets*, interagissant entre eux par *envoi de messages*.

Il convient alors de décider comment ces objets interagissent pour résoudre un problème donné (communication et synchronisation).

ALBA, acronyme de « *A parallel Language Based on Actors* », est un langage de programmation conçu pour tirer parti des architectures parallèles basées sur le transfert de messages comme primitive d'interaction fondamentale de bas niveau. ALBA est orienté sur le développement parallèle et distribué d'applications.

Dans ALBA, chaque acteur s'exécute selon ces principes d'indépendance, d'autonomie et se *synchronise* avec les autres acteurs par envoi de *messages*.
De plus, un des principes de base est qu'un acteur est associé à un seul nœud (processeur) virtuel et ne peut être partitionné. Un processeur physique contient plusieurs nœuds logiques.
ALBA dispose de deux entités différentes de traitement : les *acteurs* et les « *threads* » qui sont des blocs uniques d'exécution caractérisés par un contexte correspondant à une activité interne d'un acteur (et un seul). Au sein d'un acteur, plusieurs « *threads* » peuvent être créés, s'exécuter en parallèle et partager les ressources (mémoires, ports, …). Ainsi, une application ALBA peut être vue comme un ensemble d'acteurs qui interagissent uniquement selon les messages envoyés par d'autres acteurs.
Un acteur peut envoyer un message à un autre acteur s'il connaît son nom. Les relations peuvent être dynamiques ; c'est à dire qu'il est possible d'avoir un ensemble d'acteurs qui peuvent recevoir un message et cet ensemble peut dynamiquement croître ou décroître.
Pour chaque acteur, des pseudo-variables d'environnement permettent à un acteur de déléguer à un autre acteur (et ainsi de suite) la terminaison d'une activité. Le dernier acteur à recevoir le message est capable de répondre directement à l'acteur source du message. Ce mécanisme de *propagation* permet en outre de libérer l'acteur intermédiaire une fois la transmission effectuée car il est prêt pour le traitement d'autres messages.
La *délégation* est un moyen d'accroître la connaissance d'un acteur ; c'est un mécanisme similaire à l'héritage mais qui est basé sur l'envoi de messages et fournit donc une plus grande flexibilité.
Contrairement au modèle d'acteur primitif qui supporte seulement la communication asynchrone, l'appel à distance de procédures (« *Remote Procédure Call* » ou RPC) a été aussi ajouté dans ALBA de façon non seulement à fournir un mécanisme de synchronisation entre acteurs de haut niveau, mais aussi à éviter l'utilisation répétée de l'attente sélective.

AOUIS

[Choi et Minoura, 1994] proposent un *système d'interface-utilisateur basé sur les objets actifs* (« *Active-Object User Interface System* » ou AOUIS) conçu comme un *système d'objets actifs* (« *Active Object System* » ou AOS) qui est un système basé sur la notion de transition et qui est particulièrement adapté à la conception de systèmes concurrents.
Le comportement d'un objet d'interface actif (OIA) est donc défini par des règles de transition, des états d'allocations et des routines événementielles fournies par sa définition de classe.
Le système préconisé utilise les concepts de :

- *règles de transition* (de production), qui sont des couples de Condition-Action parfaitement adaptés pour une synchronisation flexible,
- *conditions d'affectations*, qui permettent de maintenir des relations invariantes entre les différents états d'un objet,
- *routines d'événements* qui sont activées par des *messages* et déterminent le comportement des objets actifs car elles permettent de savoir si il est possible d'accéder aux différents états des objets.

Le comportement d'un objet actif est fourni par la définition de la classe de l'objet.

Ainsi, le but principal du système est de construire par composition hiérarchique un ensemble d'objets actifs conçus comme un assemblage de *briques logicielles*, ce qui permet de construire une interface utilisateur puissante.

En outre, des *descriptions déclaratives de vues multiples* peuvent être données pour chaque OIA. La séparation de la *vue* d'un objet par rapport à son *modèle* est également importante pour une modification aisée d'une interface utilisateur. Dans le système, les différentes vues sont basées sur différents objets interconnectés par des variables d'interface. Le modèle est une super-classe avec une partie publique contenant les différentes vues en méthodes. Les vues héritent de la super-classe et communiquent par une variable d'interface et une partie publique.

Point de vue de [Arapis, 1990]

[Arapis, 1990] propose un certain nombre d'extensions aux modèles orientés-objets de façon à décrire les aspects dynamiques des applications. Ceci permet d'une part aux objets de modifier leur comportement de façon dynamique et d'autre part d'exercer un contrôle sur l'évolution dynamique des objets grâce à des contraintes exprimées avec le langage de la logique propositionnelle temporelle (LPT).

Cinq concepts caractérisent le modèle :

- *Objets et Messages*

  Ces notions sont similaires à celles des systèmes orientés-objets courants.
- *Rôles et Contextes*

  Un objet joue un rôle particulier, ce qui dans les modèles orientés-objets classiques se traduit par analogie avec « un objet est une instance de classe ».

Cependant, dans le modèle proposé par [Arapis, 1990], ces notions diffèrent car un objet peut jouer plusieurs rôles simultanément et un rôle peut être joué par le même objet plusieurs fois dans sa vie. Ainsi, le modèle permet de représenter des objets changeant dynamiquement leur comportement durant leur existence. Le changement de rôle est causé soit par l'arrivée d'un message externe soit par auto-déclenchement.

Le contexte d'un objet délimite les rôles qu'il peut jouer et est contraint par la notion d'ordre temporel. Un objet appartient à un contexte unique (au sein duquel il peut avoir divers rôles) pour toujours.

- *Scripts*

  Cette notion est similaire à celle de scénario pour un acteur de théâtre. Ainsi, pour le modèle est prévu un script qui est un « pattern » de communication entre objets mais chacun d'entre eux pouvant échanger des messages avec des objets non prévus dans le script à condition de respecter les contraintes exprimées.

  De plus, plusieurs objets peuvent participer à plusieurs scripts en même temps.

ORM

[Pernici, 1991] propose le modèle ORM (« *Object with Roles Model* ») permettant de modéliser le comportement dynamique d'objets. Cette approche s'inscrit dans le cadre du projet ITHACA (« *An Integrated Toolkit for Highly Advanced Computer Applications* ») au sein du projet européen ESPRIT II dont un des buts consiste à réduire la quantité importante de code développé par une meilleure réutilisation d'objets existants.

Ainsi, [Pernici, 1991] introduit le concept de rôle, d'états d'un objet et de contraintes sur ceux-ci pouvant être spécifiées au moyen de messages représentés par des règles.

[Pernici, 1991] définit un modèle permettant de décrire le comportement d'un objet au travers de rôles. La création d'un rôle ainsi que des messages pouvant être pris en compte dans chaque rôle selon divers contextes est définie. Le cycle de vie d'un objet est modélisé par des séquences d'*états abstraits*. Les *transitions* entre états sont régies par des *règles*. Un objet peut jouer différents rôles à différents moments mais peut également jouer plusieurs rôles au même moment. Le modèle permet non seulement de décrire séparément le comportement des objets pour chaque rôle, mais également de donner différents comportements dans différents rôles en spécifiant les règles et contraintes qui régissent le comportement concurrent.

De plus, les instances de rôles d'un objet doivent pouvoir évoluer de façon indépendante les unes des autres. Cependant, cette assomption doit être plus restrictive pour prendre en considération les interactions possibles, la coordination, les conflits entre messages dans les différents rôles, et entre les divers états.

Ainsi [Pernici, 1991] distingue trois types d'interactions :

- les *rôles indépendants*, qui sont des rôles qui ne sont absolument pas en relation,
- les *rôles coordonnés*, ce qui implique l'existence de règles de coordination soit en terme de transitions entre états soit en terme de contraintes,
- les *rôles partageant des ressources*. Bien qu'évoluant en théorie de façon indépendante, des rôles partagent les même ressources et un contrôle via une contrainte pour chaque rôle doit être assuré.

Les instances de rôles peuvent être ajoutées ou supprimées d'un objet d'une classe donnée durant son cycle de vie. Seuls les types de rôle appartenant à une définition de classe peuvent être instanciés dans un objet d'une classe donnée.

La *coordination* des instances de rôles d'un objet donné est assurée au moyen de règles dont l'invocation est du type : toutes les règles sont considérées en parallèle, etc…

Les *règles* décrivent le comportement d'un objet pour un rôle ; elles expriment par exemple les séquences possibles d'états pour ce rôle. Elles définissent quels messages peuvent être envoyés et/ou reçus. Les règles sont divisées en deux catégories :

- Les *règles de transition entre états* décrivent les caractéristiques actives (le comportement) de chaque objet, c'est à dire l'état de départ du rôle et l'évolution de l'objet pour ce rôle en fonction des messages entrant et sortant. Ces changements d'états sont modélisés par un automate dont les transitions sont exprimées par des règles. De plus, il est possible de modéliser par des règles le comportement autonome des objets en créant des états supplémentaires et des messages de transition.

  Par exemple *Si état1 et message1 Alors état2*.
- Les *règles d'intégrité* décrivent les caractéristiques passives (statiques) en terme de contraintes sur le cycle de vie possible des objets de la classe.

  Par exemple : *Si état (en_route) Alors rôle (assurance) et état (assurée)*.

Les exceptions sont traitées comme des contraintes « classiques ».

OBJECT-Z

Le langage de spécification formelle Z [Spivey, 1989] développé à l'Université d'Oxford a été amélioré à l'Université du Queensland en incorporant les concepts orientés-objets de *classe* et d'*instance*.

La spécification avec Object-Z s'effectue en plusieurs étapes.

Il faut tout d'abord identifier les composants (ou *objets*) formant le système grâce à leurs *interactions*.

Le *comportement* de chaque classe d'objet est ensuite identifié puis spécifié avec des *Schémas de Classe*.

Ces derniers ont la structure suivante :

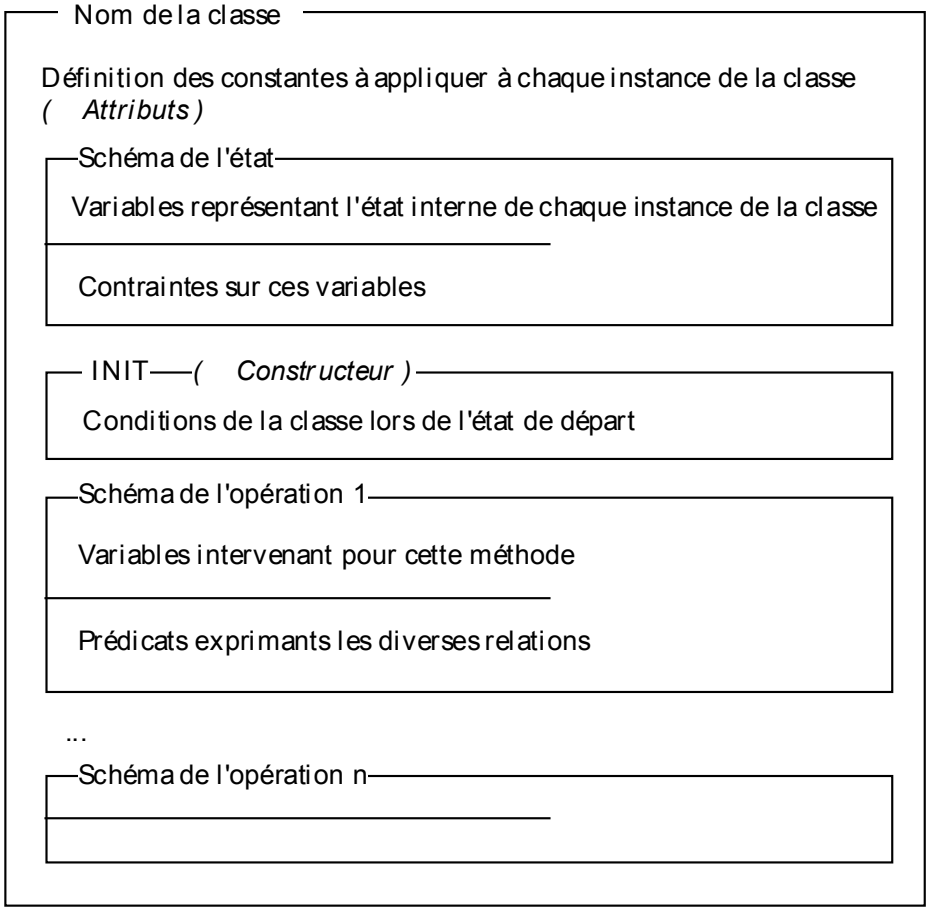

Figure 7 : Structure d'un schéma de classe en Object-Z

En plus des schémas de classe, Object-Z utilise des symboles puisés dans la logique formelle et les mathématiques.

La construction de spécifications s'effectue ensuite par combinaisons de schémas.

OODLE

OODLE, acronyme de « *Object-Oriented Design LanguagE* », de [Shlaer et Mellor, 1991], est une notation graphique indépendante d'un langage de programmation. Elle a pour buts de :

- Représenter les concepts d'encapsulation, d'héritage et de polymorphisme de façon intuitive,
- Représenter les types asbstraits de données,
- Représenter les constructeurs : visibilité des opérations, partitionnement du code, invocation des exceptions,
- La notation doit pouvoir être accessible « à la main » durant des réunions, etc… sans avoir nécessairement recours à des outils CASE,
- La notation doit, par conséquent, différer des outils CASE classiques,
- Prendre en compte différents points de vues.

OODLE est ainsi supportée par quatre composants :

- un *diagramme de classes* qui décrit la vue externe d'une classe au sens de [Booch, 1994] et de [Buhr, 1984].
- un *organigramme de la structure des classes* qui montre la structure interne du code des opérations de la classe au sens SD de [Yourdon et Constantine, 1975].
- un *diagramme de dépendances* qui décrit les relations Client-Serveur entre classes.
- un *diagramme d'héritage* qui décrit les liens d'héritage entre classes au sens du modèle d'information dans OOA [Shlaer et Mellor, 1991].

Annexe 4 : Quelques logiciels de programmation étudiés pour notre prototype

Cette annexe se base sur les travaux de [Mettrey, 1991] qui, dans son article, présente et compare quelques logiciels de programmation : CLIPS 4.3, ART-IM, KES, Level 5 et Vax OPS5. Nous complétons cette étude avec la présentation de OPS5, ARGOS-II et SNARK.

CLIPS

Qu'est ce que CLIPS ?

CLIPS [Giarratano et Riley, 1991], acronyme de « *C Language Integrated Production System* », est un outil de production de systèmes experts qui fournit un environnement complet pour la construction de systèmes experts basés sur la notion de règles et-ou d'objets.
Les origines de CLIPS datent de 1984 au Johnson Space Center de la NASA.
Les principales caractéristiques de CLIPS sont :

- *La représentation des connaissances*. CLIPS est un outil permettant de manipuler un large éventail de connaissances reposant sur trois paradigmes de programmation différents basés sur : les règles, l'orienté-objet et le procédural. La programmation basée sur les règles permet de représenter la connaissance à l'aide d'heuristiques, ce qui permet de spécifier un ensemble d'actions à exécuter dans une situation donnée. La programmation orientée-objet permet de modéliser des systèmes complexes de façon modulaire (avec des composants facilement réutilisables pour modéliser d'autres systèmes ou pour définir de nouveaux composants). Les fonctionnalités de programmation procédurale de CLIPS sont analogues à celles des langages tels que C, Pascal, Ada et LISP.

- *La portabilité*. CLIPS est écrit en C pour des raisons de portabilité et de vitesse d'exécution. Il a été installé sur plusieurs plateformes sans avoir recours à un changement de code (PC, Macintosh, VAX, Sun). CLIPS peut être porté sur tout système disposant d'un compilateur C ANSI. CLIPS est fourni avec son code source et peut donc être modifié ou adapté par l'utilisateur selon ses besoins spécifiques.
- *L'intégration et l'extensibilité*. CLIPS peut être inclus dans du code procédural, appelé comme sous-routine, et intégré avec des langages comme C, FORTRAN ou Ada. CLIPS peut être facilement étendu par un utilisateur grâce à l'utilisation de plusieurs protocoles bien définis.
- *Le développement interactif*. La version standard de CLIPS fournit un environnement de développement textuel interactif s'appuyant sur des aides de déboguage, une aide en ligne, et un éditeur intégré. Des interfaces proposant des fonctionnalités comme des menus déroulants, des éditeurs intégrés, et le fenêtrage multiple ont été développées pour les environnements Macintosh, Windows et X Window.
- *La vérification et la validation*. CLIPS dispose d'un ensemble de fonctionnalités permettant la vérification et la validation des systèmes experts, et notamment pour la conception modulaire et le partitionnement d'une base de connaissances, pour le contrôle et la vérification des contraintes statiques et dynamiques portant sur les arguments de fonctions ou sur les attributs de classes, et pour une analyse sémantique des règles permettant de déterminer si des inconsistances peuvent empêcher une règle de se déclencher ou de générer une erreur.
- *Une documentation complète*. CLIPS est fourni avec une documentation complète incluant un manuel de référence et un guide utilisateur.

CLIPS 4.3

1. La représentation des connaissances en CLIPS 4.3

Elle est assurée avec des règles selon une syntaxe proche de LISP, originellement développée pour ART. CLIPS supporte un langage de filtrage riche permettant de spécifier des conditions de règles. Bien que faisant penser aux fonctionnalités des langages de la famille OPS5, les opérateurs de filtrage de CLIPS sont plus complexes et plus puissants.

Le filtrage est effectué à la fois sur des séquences à champ unique ou multivalué composées de chaînes, symboles et nombres. Il est possible de spécifier des contraintes de filtrage sur les faits et les règles. Ainsi le raisonnement peut être effectué sur l'absence d'information aussi bien que sur la présence de ces informations.

Le déclenchement d'une règle n'est possible que si toutes les conditions du côté gauche de la règle sont vérifiées (ET logique). CLIPS autorise cependant la spécification de disjonctions (OU logique).

De même CLIPS autorise les constructeurs procéduraux sur le côté droit des règles (*if…then…else, while*). La disjonction et l'aspect procédural permettent d'exprimer en une seule règle des situations qui en nécessitent plusieurs dans les langages de la famille OPS5.

2. L'inférence dans CLIPS 4.3

Elle est basée sur une *stratégie de contrôle par chaînage avant*. Les conditions du côté gauche des règles sont comparées avec les faits de la base de connaissances. Les règles dont toutes les conditions sont vérifiées sont ensuite activées ; une règle donnée pouvant produire de multiples activations. Les règles activées sont alors placées dans un *agenda* (référencé comme un ensemble de conflits par certains générateurs de systèmes experts). Cet agenda contient toutes les activations courantes de règles. CLIPS sélectionne la règle à activer selon son *degré de priorité*. Ce degré est spécifié lors de l'écriture des règles et sa valeur évolue entre -10 000 et +10 000 avec par défaut la valeur 0. Si plusieurs règles ont le degré de priorité le plus élevé, l'agenda est considéré comme une pile et la dernière règle activée est sélectionnée. L'exécution d'une règle consiste à réaliser la (ou les) action(s) spécifiée(s) sur le côté droit de la règle. Après exécution, le cycle est répété jusqu'à ce que l'instruction *halt* apparaisse sur le côté droit d'une règle ou bien jusqu'à ce qu'il n'y ait plus de règles activées par les faits de la base de connaissances.
CLIPS dispose de l'algorithme de chaînage avant efficace RETE [Forgy, 1982]. Ce dernier traite uniquement les faits ayant changés, ainsi que les règles qui sont potentiellement affectées, plutôt que de répéter le filtrage de toutes les règles avec tous les faits. L'efficacité de cet algorithme est basée sur l'hypothèse selon laquelle les changements de données sont rares (par exemple si l'exécution d'une règle affecte moins de 5% des faits). Cette hypothèse serait complètement inadaptée pour des applications tutorielles (ainsi que pour toute application où les changements de données sont fréquents comme la simulation qualitative). Il est cependant possible de se conformer à cette hypothèse avec une pré-compilation des données (par exemple en ne tenant compte que des changements « significatifs »). L'inconvénient majeur de cette pré-compilation est qu'il faut écrire du code C et garder la trace des pointeurs vers les faits.
CLIPS 4.3 manque de support pour le chaînage arrière et la programmation OO (*ceci est corrigé avec COOL dans CLIPS 6.0*).

Les systèmes experts construits avec CLIPS s'exécutent soit de façon interprétée (par défaut) soit de façon compilée.

CLIPS 6.0

CLIPS version 6.0 comporte les améliorations et changements principaux suivants [Riley, 1995] :

- Des instances de classes COOL définies par l'utilisateur peuvent être filtrées sur le côté droit des règles.
- Le constructeur « *defmodule* » permet à une base de connaissances d'être modularisée et partitionnée.
- Des éléments non conditionnels peuvent dorénavant contenir des éléments conditionnels autres que ceux utilisés pour le filtrage. Les éléments conditionnels « *exists* » et « *forall* » ont été ajoutés. Des contraintes de communication peuvent être utilisées avec des variables à champs multiples.
- Le contrôle et la vérification des contraintes statiques et dynamiques est assuré de façon plus importante que dans CLIPS 5.1 par l'intermédiare des « *deftemplates* » qui peuvent contenir plusieurs attributs (« *slots* ») à champs multiples.
- Une interface CLIPS sous Windows est disponible sur les ordinateurs compatibles PC.

Après plusieurs années d'existence, la plupart des fonctionnalités orientées-objets majeures sont présentes dans CLIPS. Les améliorations futures devront prendre en compte le raisonnement temporel, l'héritage par généralisation et la persistance des objets.

Versions spéciales de CLIPS

Les principales variantes de CLIPS sont les suivantes :

- *HyperCLIPS* version 2.0 intègre la version Macintosh Hypercard 1.2 (ou plus) avec CLIPS 5.0 et est développé par COSMIC.
- *wxCLIPS*, développé par Julian SMART à l'A.I.A.I. d'Edimbourg, fournit une interface graphique à CLIPS 6.0 ainsi qu'un ensemble de fonctionnalités supplémentaires permettant de créer des applications GUI.
- *FuzzyCLIPS* 1.5 est une extension de CLIPS 5.1 prenant en compte la logique floue et est développé par Togai InfraLogic.
- *FuzzyCLIPS* 6.02 est une extension de CLIPS 6.02 permettant de représenter et manipuler des règles et des faits flous combinés éventuellement avec des règles et faits certains. Ce logiciel est développé par le Knowledge Systems Laboratory du National Research Council du Canada et est différent du logiciel cité ci-dessus développé par Togai InfraLogic.
- *CLIPsTOOL*, développé par le Docteur Terry FEAGIN, est une extension de CLIPS 6.0 qui permet notamment au programmeur CLIPS de créer des fenêtres, des éléments (boutons, champs de texte, …) à l'intérieur de fenêtres, menus, … et de récupérer des événements pour

déclencher des règles. Ce logiciel, écrit en X, utilise la boîte à outils XView (distribuée avec X), et peut s'exécuter sur les stations de travail Sun et IBM RS/6000.

- *DYNACLIPS* (DYNAmic CLIPS utilities) version 3.1, développé par Yilmas CENGELOGLU, est un ensemble de tableau noir, d'échange dynamique de connaissances et d'outils pour les agents. Cet ensemble de bibliothèques peut être lié avec CLIPS 5.1 ou 6.0 sous Unix.
- *AGENT_CLIPS* 1.0, développé par Yilmas CENGELOGLU, est un outil multi-agents pour Macintosh permettant à de multiples copies de CLIPS de s'exécuter en même temps. Chaque agent (CLIPS) peut envoyer des commandes aux autres agents actifs lors de l'exécution. AGENT_CLIPS manipule automatiquement des commandes en entrée. Le transfert de commande permet aux faits et aux règles de s'échanger durant l'exécution. C'est une forme d'échange de connaissances entre les agents intelligents. AGENT_CLIPS 1.0 est disponible sur Compuserve dans le forum AIEXPERT dans Libraries, Expert System.
- L'Université de médecine de Washington a développé un ensemble de fonctions pour les bases de données qui fournissent une interface Sybase pour CLIPS 6.0.

ART-IM : Automated Reasoning Tool for Information Management

ART-IM a été développé comme alternative aux versions de ART basées sur LISP. ART-IM est écrit en C et se base sur CLIPS 4.3.
ART-IM fournit un sous-ensemble de fonctionnalités issues de ART. Les principales différences avec ART sont le manque de support pour un chaînage arrière, les statistiques de performance, les points de vue, l'héritage multiple, et les relations d'héritage définies par l'utilisateur.

La représentation des connaissances en ART-IM 2.0

La principale différence avec CLIPS 4.3 est que ART-IM [Mettrey, 1991] supporte la notion de « *frame* » avec des possibilités d'héritage simple et de programmation orientée-objet.

L'inférence dans ART-IM 2.0

Elle est basée sur une stratégie de contrôle par chaînage avant ainsi que sur l'algorithme de Rete.

ART-IM supporte une programmation orientée-objet comme extension de son système de « *frame* ». Si on compare cependant ART-IM avec des langages orientés-objets comme Eiffel et C++, ART-IM souffre d'un manque de puissance pour le masquage d'informations. En effet, tous les schémas (« *schematas* ») et les attributs (« *slots* ») de ART-IM sont visibles et librement accessibles aux règles et procédures. De plus, ART-IM, au contraire de ART qui est basé sur LISP, ne supporte pas le chaînage arrière et la notion de points de vue (raisonnement hypothétique).

L'environnement de production de ART-IM 2.0

Les systèmes experts construits avec ART-IM s'exécutent soit de façon interprétée (par défaut) soit de façon compilée. Pour une version compilée de la base de connaissances, il est toutefois nécessaire de traduire les règles en C.

KES : Knowledge Engineering System

La représentation des connaissances en KES 2.6

Elle s'appuie sur une stratégie de contrôle des règles par chaînage avant ou arrière et sur un système de classes.
KES 2.6 manipule l'incertitude avec des facteurs de mesure de la croyance dont les valeurs évoluent entre -1,0 (incrédulité totale) et +1,0 (croyance totale).
Les classes peuvent s'interfacer avec le système de règles et peuvent être utilisées pour spécifier les attributs testés par les conditions des règles. Les classes supportent la notion d'héritage simple mais pas les attachements procéduraux. Des fonctions intrinsèques et-ou définies par l'utilisateur peuvent cependant être appelées par des règles en chaînage avant ou arrière.

L'inférence dans KES 2.6

Elle est basée sur une stratégie de contrôle par chaînage arrière ou par chaînage avant avec une prédominance pour le chaînage arrière. Lorsqu'un attribut indéterminé est rencontré en chaînage avant, le chaînage arrière est lancé. KES 2.6 ne se base pas sur l'algorithme de Rete mais sur un algorithme de parcours en profondeur d'abord et en chaînage avant.

L'environnement de production de KES 2.6

KES 2.6 ne dispose pas de boîte à outils graphique pour la construction d'applications GUI.

Level 5

La représentation des connaissances en Level 5

Le langage des règles de Level 5 ressemble à de l'anglais mais demeure faible en fonctionnalités d'expressivité et de filtrage (pas de variables à champs multiples, de filtres non-restreints). Level 5 fournit seulement des opérateurs relationnels pour les tests dans les conditions des règles.
Level 5 manipule l'incertitude en utilisant des facteurs de confiance dont la valeur varie entre 0 (incrédulité totale) et 100 (croyance totale) ; la valeur 50 pour un fait signifie qu'il peut être aussi bien vrai que faux.

L'inférence dans Level 5

Elle est basée sur une stratégie de contrôle par chaînage arrière. Cependant, Level 5 ne fournit pas de support direct pour un chaînage avant.

L'environnement de production de Level 5

Level 5 ne dispose pas de boîte à outils graphique pour la construction d'applications GUI et manque également de support pour une exécution compilée.

Vax OPS5: VAX Official Production System version 5

VAX OPS5 3.0 est la version OPS5 appartenant à DEC.

La représentation des connaissances en VAX OPS5 3.0

La syntaxe des règles en VAX OPS5 s'inspire de LISP et est similaire à celle de ART-IM et CLIPS. Les fonctionnalités de filtrage restent néanmoins inférieures à celles de ART-IM et CLIPS (pas de disjonctions dans la partie gauche des règles, pas de constructeurs de programmation procédurale du type *if…then…else* et *while* dans la partie droite des règles, etc…).

L'inférence dans VAX OPS5 3.0

Elle est fondée sur une stratégie de contrôle par chaînage avant basée comme CLIPS et ART-IM sur l'algorithme de Rete. Cependant, le système d'inférence ne supporte ni le chaînage arrière, ni la notion de « *frame* ». Toutefois, VAX OPS5 dispose de deux stratégies de résolution de conflits : Lex (Lexical sort) et MEA (Means-Ends-Analysis) qui sont plus sophistiquées que le mécanisme utilisé par CLIPS et ART-IM.

L'environnement de production de VAX OPS5 3.0

VAX OPS5 3.0 est écrit en Bliss (langage d'implémentation de système de DEC) et ne s'exécute que sur des machines DEC. Toutefois, il peut s'interfacer avec de nombreux langages supportés par DEC. Ceci permet une exécution par compilation rapide et la conception de systèmes experts importants. Par contre, cela implique des restrictions matérielles et logicielles. Un autre inconvénient est que le système d'inférence ne supporte pas le chaînage arrière ou la notion de « *frame* ».

OPS5

OPS5 manipule des règles avec des variables quantifiées. Son moteur d'inférence est principalement du type chaînage avant, non monotone, avec résolution de conflits centrée sur les rapports d'antériorité des faits dans la base de faits. Il n'y a pas de gestion des retours-arrière. Une des principales caractéristiques de ce système est de disposer d'un compilateur qui transforme les membres gauches des règles de la base de règles en un réseau permettant ainsi une étape de filtrage particulièrement efficace avec l'agorithme de RETE [Forgy, 1982].

ARGOS-II

ARGOS-II [Fade, 1980] est un générateur de systèmes experts qui réunit un langage d'expression des connaissances, des structures d'accueil pour les connaissances et un moteur d'inférence de type chaînage mixte, avec régime de contrôle par tentatives (c'est à dire retours-arrière restaurant les contextes antérieurs), non-monotone. ARGOS-II a été conçu pour assurer des tâches de planification d'actions et a été écrit en LISP. ARGOS-II manipule aussi les notions de *problèmes* (c'est à dire des faits à établir), de *procédures*, de *démons*, de *liaisons* et de *théorèmes*.

SNARK

SNARK, acronyme de « *Symbolic Normalized Acquisition Representation of Knowledge* », est un générateur de systèmes experts [Vialatte, 1985] écrit en Pascal.
On peut écrire en SNARK des règles avec variables quantifiées. Son moteur d'inférence est du type chaînage avant, non monotone, avec restriction sur les instances de règles tirées, et fonctionnement jusqu'à saturation de la base de faits. SNARK manipule aussi les notions de *démons* et de *métarègles*.

Il existe d'autres systèmes : GPS [Ernst et Newell, 1969], Emycin [Buchanan et Sutherland, 1984], Alice [Laurière, 1982],etc …

Cette étude renforce l'opinion selon laquelle chaque générateur de systèmes experts dispose de forces et de faiblesses, et qu'aucun ne couvre bien mieux que les autres toutes les applications et toutes les fonctionnalités.

Par conséquent, pour obtenir les meilleurs résultats possibles pour une application donnée, il convient de choisir les outils adéquats plutôt que de se forcer à utiliser un outil inadapté.

Annexe 5 :
Logiciels utilisés pour notre prototype

« wxWindows »

Présentation

wxWindows [Smart, 1995b] est une bibliothèque de classes C++ destinée aux interfaces-utilisateur graphiques (GUI) multi-plateformes. wxWindows supporte Open Look (XView), Motif et Windows à l'aide de 60 classes et 650 fonctions publiques.
wxWindows a été développé à l'Artificial Intelligence Applications Institute (A.I.A.I.) de l'Université d'Edimbourg. A l'origine, wxWindows était destiné à un usage interne pour un projet d'outil basé sur la notion de diagrammes et d'hypertextes, et conçu pour l'acquisition des connaissances (HARDY).
wxWindows a été conçu pour fournir des moyens flexibles et peu coûteux permettant de maximiser le rendement investi dans le développement d'applications GUI.
Alors que de nombreuses bibliothèques commerciales de classes existent déjà pour le développement multi-plateforme (telles CommonView ou XVT++), aucune ne présente l'ensemble des critères suivants :

- bas prix (libre de droit),
- disponibilité du code source,
- simplicité de programmation,
- support GCC (GNU C++),
- support de communication entre processus (« *interprocess communication* »).

En tant que logiciel libre de droit, wxWindows a bénéficié de commentaires, idées, corrections de bogues, améliorations et de l'enthousiasme des utilisateurs, spécialement via Internet. Ceci donne à wxWindows un certain avantage sur ses équivalents commerciaux. De plus, ceci lui confère une certaine robustesse vis à vis de la subjectivité d'un individu ou d'une société.

Dans l'écriture de wxWindows, le principe de complétude n'a pas été respecté pour des raisons évidentes de programmation simple et portable. Pour les projets ne nécessitant pas absolument des interfaces impeccables, ce compromis semble louable au vu des bénéfices de productivité.

wxWindows convient à trois API natives : Xview (Open Look), Motif, Windows. De plus, wxWindows supporte un sous-ensemble de DDE fonctionnant à la fois sur PC et sur Unix. Un modèle orienté-objet simple Client-Serveur est utilisé. Ceci facilite l'écriture de programmes communicant de façon synchrone. Sous Windows, d'autres programmes non-wxWindows peuvent communiquer avec des programmes wxWindows et réciproquement. Sous Unix, les programmes non-wxWindows doivent juste se conformer à un protocole simple lorsqu'ils communiquent via des "*sockets*" avec des programmes wxWindows.

L'utilisation d'une bibliothèque de classes indépendante de toute plateforme est primordiale parce que le développement d'applications GUI est très coûteux en temps et que le prolongement de la popularité de certaines GUI ne peut être garanti. Le code peut devenir très rapidement obsolète s'il s'adresse à une plateforme ou une audience inadéquate. wxWindows aide le programmeur à éviter ceci. Bien que wxWindows puisse ne pas convenir à toutes les applications, il fournit un accès à la plupart des fonctionnalités qu'un programme GUI nécessite couramment. En outre, l'interface avec XView, Motif et Windows est plus propre que les API natives. Ainsi, les programmeurs peuvent opter pour wxWindows même s'ils développent sur une seule plateforme.
wxWindows puise des éléments dans d'autres API GUI et ajoute quelques éléments qui lui sont propres. Bien entendu, wxWindows n'a pas l'ambition de recouvrir toutes (ou même une seule) API native de façon complète.

Avantages, inconvénients et perspectives

Les principaux avantages de wxWindows sont résumés ci-dessous :

- un coût faible (gratuité),
- un code source disponible,
- de nombreux exemples de programmes,
- une documentation en ligne et sur papier de plus de 200 pages,
- une API orientée-objet et simple d'utilisation,
- un affichage d'éléments selon le style XView, avec en plus une option d'affichage basée sur la notion de contrainte,
- une facilité de positionnement de ligne,

- des communications entre processus OO faciles (sous-ensemble de DDG) sous Windows et Unix,
- la génération de PostScript encapsulé sous Unix et une impression normale sous Windows PC,
- la possibilité de créer des DLL sous Windows et des bibliothèques dynamiques sous Sun,
- le support des dialogues avec l'imprimante et entre fichiers sous Windows, avec l'équivalent sous Unix,
- la possibilité de créer des métafichiers et de les copier dans le tableau de bord de Windows,
- des facilités de programmation (par contraintes) rapide d'écrans,
- des facilités d'aide hypertextuelle, avec une API pour l'appel à partir d'applications,
- un logiciel (« *wxBuilder* ») permettant la construction interactive d'interfaces simples, et la génération de code C++.

Quelques uns des points noirs de wxWindows sont les suivants :

- pas de support commercial (mais un support via Internet peut se révèler meilleur),
- des restrictions de style XView sur les relations parent-enfant (bien que celles-ci soient modelées pour les plateformes non-XView),
- un support minimal pour le choix des couleurs,
- pas encore de support OLE-2.

Les perspectives sont nombreuses et notamment :

- prise en compte de OLE-2 ;
- connectivité avec les bases de données (en cours) ;
- amélioration de wxBuilder ;
- intégration avec une boîte à outils existante pour le travail en réseau ;
- *« widget »* mutlimédia (en construction) ;
- amélioration des procédures d'installation et de création de fichiers ;
- ports : Mac, NeXT et OS/2.

Utilitaires basés sur wxWindows

Un certain nombre de fonctionnalités basées sur wxWindows viennent compléter celles constituant la principale bibliothèque de classes. Ainsi de nombreux logiciels ont été construits en se basant sur wxWindows : HARDY, wxCLIPS, Tex2RTF, PrologIO, QDB database, MEWEL, …

« wxCLIPS »

Introduction

wxCLIPS [Smart, 1995a] a été développé pour permettre aux programmeurs CLIPS d'écrire des programmes portables, graphiques et qui s'exécutent sous X et Windows. Ceci permet d'avoir un style de programmation guidé par la notion d'événement ainsi qu'un ensemble de fonctions GUI. En effet, l'acronyme wxCLIPS signifie bien une interface de CLIPS avec wxWindows, c'est-à-dire une bibliothèque GUI écrite en C++.

wxCLIPS supporte CLIPS 6.0 avec éventuellement ses extensions de logique floue.

wxCLIPS est composé de deux entités :

1. Une bibliothèque de fonctions C++ permettant d'accéder à un ensemble de fonctionnalités wxWindows.
2. Un environnement de développement simple et autonome pour développer des applications wxWindows utilisant ces extra-fonctions CLIPS.

La bibliothèque peut être utilisée par n'importe quel programme C++, pour lui donner un langage adaptable et un accès interactif aux fonctionnalités GUI. De façon autonome, wxCLIPS est une simple interface de développement utilisant cette bibliothèque.

wxCLIPS est constitué d'un ensemble de fonctions, plutôt que d'objets COOL (CLIPS Object Oriented Language), puisque par définition, les extensions utilisateurs basées sur le langage C sont réduites à des fonctions. Cependant, il existe maintenant un ensemble de classes appelé wxCOOL qui encapsule la plupart des fonctionnalités wxCLIPS de façon propre et orientée-objet.

Les extensions wxCLIPS de CLIPS ont été écrites par Julian SMART à l'Artificial Intelligence Applications Institute de l'Université d'Edimbourg. Différentes versions sont disponibles sur des plateformes Windows, Sun Open Look et Motif.

Relation entre CLIPS, wxCLIPS et wxWindows

• wxWindows

C'est une bibliothèque de classes C++ développée à l'A.I.A.I. par Julian SMART pour permettre un développement multi-plateforme. C'est un logiciel libre de droit.

• CLIPS

C'est un générateur de systèmes experts portable développé à la NASA qui supporte trois sortes de programmation : à base de règles, fonctionnelles et orientée-objet. CLIPS est écrit en C et est également libre de droits.

• wxCLIPS

C'est l'union de wxWindows et CLIPS, c'est à dire un ensemble de fonctions CLIPS disposant d'une large portion des fonctionnalités wxWindows. Ce logiciel est disponible dans trois formats : en tant que bibliothèque, ou bien intégré dans d'autres produits, ou bien en tant qu'exécutable avec une interface simple.

Utiliser wxCLIPS pour construire des interfaces utilisateurs graphiques (GUI)

wxCLIPS possède un environnement de développement basique composé d'une fenêtre avec des menus, une petite fenêtre d'entrée et une fenêtre de texte de sortie.

Toutefois, les programmes CLIPS sont en général saisis dans des fichiers possédant leur propre interface et non dans la fenêtre d'entrée standard qui n'est utilisable qu'en lecture.

La principale différence entre CLIPS standard et wxCLIPS réside dans le fait que la fonction (de lecture) ne récupère pas ses entrées de l'entrée standard. En fait, cette notion d'entrée standard n'existe pas en wxCLIPS. De telles fonctions doivent être remplacées par des appels à des fonctions GUI prédéfinies.

Connexion avec les classes de bases de données

wxCLIPS fournit un ensemble de classes pour accéder à un sous-ensemble de produits ODBC (« *Open DataBase Connectivity* ») de *Microsoft*. A l'heure actuelle, ce paquetage n'est disponible que sous MS Windows bien que ODBC puisse apparaître sous d'autres plateformes et bien qu'un émulateur SQL générique ou spécifique destiné aux classes ODBC puisse être fourni pour wxWindows ultérieurement.

ODBC possède une API unifiée pour une large variété de bases de données. Ceci est réalisé en interfaçant indirectement chaque base de données ou fichier avec un pilote ODBC parmi lesquels Dbase, Access, Excel et d'autres formats de fichiers.

Aperçu général de « wxCOOL »

Jusqu'en juillet 1995, les fonctionnalités wxCLIPS étaient conceptuellement orientées-objets mais implémentées seulement avec des fonctions wxCLIPS. Puisque le seul moyen de coupler CLIPS avec des programmes C ou C++ est de définir des fonctions utilisateurs, l'usage de ces fonctions est un prérequis. wxCOOL est un ensemble de classes CLIPS construites sur des fonctions utilisateurs, encapsulant la plupart des fonctionnalités wxCLIPS nécessaires.
Puisque wxCOOL est implanté en terme de fonctions wxCLIPS, wxCOOL alourdit une application CLIPS en terme de temps de chargement, de vitesse d'exécution et à un degré moindre, de capacité de mémoire. Ainsi, il est conseillé de coder des morceaux primordiaux en terme de vitesse d'exécution de l'application à l'aide de fonctions brutes wxCLIPS, en particulier lorsque beaucoup d'éléments GUI doivent être créés.

« HARDY »

Présentation

HARDY [Smart et Rae, 1995] est un outil conçu et développé par l'Artificial Intelligence Applications Institute (A.I.A.I.) à l'Université d'Edimbourg, essentiellement pour les applications à base de diagrammes. HARDY est exécutable sur stations de travail Unix avec Motif ou Open Look, et sur PC avec Windows.
Les possibilités de dessin de diagrammes de HARDY sont construites sur une structure de type *hypertexte* sur laquelle chaque diagramme possède sa propre *carte* (fenêtre). De même, plusieurs cartes peuvent être liées pour former un *arbre* ou un *réseau*. L'utilisateur peut se déplacer dans ce réseau, soit en suivant les *liens* d'hypertexte, soit en consultant *l'arbre d'index* et en cliquant sur un titre de carte.
Les diagrammes sont en général construits de façon *hiérarchique* dans la mesure où un noeud de haut niveau représente toute la portion du diagramme située en dessous. HARDY supporte ce type d'organisation grâce aux *cartes de développement* (« *expansion cards* »). Une carte de diagramme avec ses cartes de développement seront traitées comme une unité, un fichier par exemple.

Considérons plusieurs éléments d'une session HARDY.

La *fenêtre de contrôle* est la fenêtre principale de l'application car elle permet d'afficher la hiérarchie des cartes. Elle est également utilisée pour des tâches administratives variées comme le chargement et le stockage de fichiers d'index.
Une *carte de diagramme* possède une fenêtre de saisie et une *palette de symboles* paramétrables pour lesquels il est possible d'afficher les *attributs* des différents noeuds.
Le *gestionnaire de type de diagramme* est un outil permettant de définir et modifier les types de diagrammes.
En complément des cartes de diagrammes, l'utilisateur peut créer des cartes imbriquées de texte, pour documenter un diagramme par exemple. Nous parlons de *cartes hypertextes*.
HARDY utilise sept types de fichiers possédant différents types d'information :

- 1 - Fichier de la liste des définitions (**diagrams.def**)
- 2 - Fichier de bibliothèque des symboles (**.slb**)
- 3 - Fichier d'index des cartes de collection (**.ind**)
- 4 - Fichier de définition de diagrammes et d'hypertextes (**.def**)
- 5 - Fichier de cartes de diagrammes (**.dia**)
- 6 - Fichier de cartes d'hypertexte (**.hyp**)
- 7 - Fichier de cartes de texte (**.txt**)

HARDY et wxCLIPS

HARDY possède une version simplifiée du noyau du générateur de système expert CLIPS 6.0 mais aussi de nouvelles fonctions spécifiques qui peuvent être appelées depuis CLIPS, ce qui permet à HARDY d'être personnalisé à un degré supérieur à ce qu'il ne peut être avec le gestionnaire de type de diagramme. La plupart des structures accessibles sont référencées dans le code CLIPS par un identifiant de type entier, à l'exception des noeuds et arcs nommés et des autres *types* (à l'inverse des instances des structures de ces mêmes types) qui sont référencés par leur nom.
Un diagramme HARDY est donc un réseau de *noeuds* et *d'arcs* (plus généralement d'objets). Ils sont représentés visuellement par des *images* de noeuds et d'arcs. La distinction est nécessaire pour pouvoir associer différentes images à un même objet (par exemple, le même noeud apparaissant sur plusieurs cartes). Toutefois, de façon courante, chaque objet ne possède qu'une image unique [66]. Quand une image est créée, un objet sous-jacent est automatiquement créé et son identifiant peut être retrouvé à partir de l'image si besoin. Les objets sont associés à la carte de plus haut niveau alors que les images sont associées à la carte sur laquelle elles sont affichées.
Les noeuds et les arcs ont des *attributs* sous forme de chaînes de caractères.

La structure d'hypertexte est basée sur le concept *d'élément* hypertexte. Chaque type de carte possède sa propre idée de ce qu'est un élément - pour la carte de diagramme, chaque image est un élément et par conséquent contient une structure d'élément. Les éléments peuvent être liés entre eux par des *liens* hypertextes (ou *hyperliens*). Une carte possède toujours au moins un élément, appelé *élément spécial*. Ainsi, chaque carte qui, soit ne contient pas d'éléments (e.g. une carte texte), soit ne possède pas d'éléments appropriés, peut encore être liée à une autre carte ou élément.
En utilisant les fonctions adéquates, les éléments peuvent être retrouvés à partir des images, et chaque lien attaché aux éléments peut être parcouru, pour accéder aux autres éléments, cartes ou images connectés. Par commodité, il y a des fonctions qui manipulent les cartes de développement (qui sont des cartes liées à des éléments d'images via des liens spéciaux) sans avoir à accéder aux éléments et à traverser explicitement les liens.

TEX2RTF

Le format de texte RTF, acronyme de « *Rich Text Format* », a été mis au point par Microsoft afin de proposer une passerelle, un format de texte universel et compréhensible quel que soit le système d'exploitation (Mac OS, Windows, DOS, OS/2, Unix). RTF va bien plus loin que le format ASCII puisqu'il prend en compte des éléments de mise en page comme les attributs typographiques, le multicolonnage, les illustrations, les tabulations, les en-têtes et pieds de page ou les notes de bas de page.
C'est pour cet ensemble de raisons que nous avons choisi RTF comme format servant de support aux différents documents que nous générons avec notre prototype. Par l'intermédiaire de deux logiciels simples d'utilisation (et libres de droit) que nous avons couplés avec notre prototype, cela nous permet de générer d'une part un document lisible sous le traitement de texte Microsoft Word et d'autre part un document au format HTML accessible sur Internet [67].
Toutefois notre prototype ne génère pas directement un document au format RTF mais au format LaTEX que nous transformons ensuite en RTF avec le logiciel « Tex2RTF » développé également à l'AIAI d'Edimbourg par Julian SMART.

[66] Pour l'instant, il n'y a pas de fonctions permettant la création d'images supplémentaires pour des objets existants.
[67] A condition bien entendu de le mettre sur un serveur.

RTF2THML

Ce logiciel convertit les fichiers au format RTF vers le format HTML version 3 pour une visualisation à l'aide de navigateurs comme Netscape Navigator ou Internet Explorer par exemple.
C'est un programme libre de droit réalisé par Bertrand LE QUELLEC [68].

[68] http://www.mygale.org/09/blq/

Annexe 6 :

Les agents

Pour résoudre les problèmes ordinaires de la vie courante, nous disposons d'une grande diversité de moyens. Lorsqu'une méthode ne convient pas, nous pouvons en essayer une autre. Si l'information dont nous avons besoin n'est pas accessible, nous pouvons tenter de raisonner par analogie. Si un mode de raisonnement ne « marche » pas, nous pouvons en utiliser un autre. Si nous ne réussissons pas à résoudre un problème, nous pouvons faire marche arrière et chercher une nouvelle voie qui nous permettra de sauver la situation même si c'est de manière différente de celle que nous avions d'abord envisagée.
Si nous voulons transposer ceci à l'informatique le concept *d'agent*, selon Marvin MINSKY du M.I.T., saurait apprendre à nous connaître, interpréter nos directives, prendre des initiatives, bref nous aider sans que nous ayons trop à l'aider à le faire. Au fur et à mesure de son apprentissage, un agent devient alors de plus en plus « confiant » dans ses critères de décision [Riecken, 1996].

Cependant, la notion d'agent n'est pas nouvelle. Un grand nombre d'expressions et de termes différents ont été en effet utilisés par les chercheurs pour décrire les agents sur lesquels ils travaillaient. Il est aussi intéressant de remarquer que les recherches récentes sur les agents impliquent un grand nombre de disciplines différentes. En effet, les apports proviennent de chercheurs travaillant dans des domaines aussi variés que le génie logiciel, la robotique, le génie cognitif, les systèmes à base de connaissances, les bases de données, la résolution de problèmes, la planification, l'apprentissage automatique, les sciences cognitives, la psychologie, ou encore l'interaction homme-machine. L'étude des agents informatiques semble donc offrir une occasion unique d'intégrer de nombreux résultats importants provenant de très nombreux domaines de recherche différents.
La définition exacte d'un agent semble donc être assez confuse. De façon générale, les agents doivent réagir à leur environnement, apprendre par expérience et traiter des tâches utiles pour leur propriétaire.

Définition d'un agent

Un agent possède les caractéristiques suivantes [Ferber, 1989], [Ferber, 1995] :

- *Personnalisé*. Un agent doit s'améliorer pour s'adapter aux goûts et préférences de l'utilisateur.
- *Flexible*. Un agent doit avoir de l'initiative et proposer des suggestions.
- *Autonome*. Un agent doit être sensible à son environnement; il doit être capable de jugements.
- *Spécialisé*. Un agent doit être spécifique à un domaine particulier et il doit vous donner une aide équivalente à celle d'un professionnel.

Un *agent* est donc un système composant un *environnement dynamique complexe*. Un agent connaît l'environnement dans lequel il évolue ; il peut agir sur cet environnement car il dispose d'un ensemble d'objectifs à atteindre. En fonction du type d'environnement, un agent peut prendre différentes formes. Des robots autonomes sont des agents du monde réel ; des agents logiciels font partie du monde informatique, …
Les agents logiciels fournissent ici une assistance active et personnalisée pour les utilisateurs d'applications informatiques. Les agents logiciels différent des logiciels usuels dans la mesure où :

- ils sont actifs car ils prennent l'initiative pour aider l'utilisateur par des suggestions ou bien pour automatiser certaines tâches fastidieuses ;
- ils sont adaptatifs car ils apprennent les préférences (et leur évolution dans le temps) des utilisateurs ;
- ils sont personnalisés en fonction des connaissances qu'ils ont de l'utilisateur.

En tant que tel, il est alors possible de prendre la mesure d'un agent en termes de propriétés telles que sa correction, sa complétude, son efficacité et sa fiabilité, pour décider si ce logiciel peut être considéré comme un agent digne de confiance.

Les systèmes multi-agents

Les systèmes multi-agents sont une branche de l'intelligence artificielle distribuée. Dans ce sens, un système multi-agents est une société d'agents autonomes travaillant en commun pour aboutir à un objectif global qui peut-être la résolution d'un problème, la construction d'un plan, etc…
La communauté des systèmes multi-agents est partagée en deux écoles [Vigny, 1997] :

- *l'école « réactive »* : les agents composant ces systèmes ont un comportement simple et ne disposent d'aucune « intelligence » pour exécuter des tâches souvent complexes. Ces agents ont une connaissance plus parcellaire et un comportement de type réflexe ne prenant en compte ni une explicitation des intentions, ni

une élaboration de plans. Ils ne tiennent pas compte non plus de leur historique. L'intelligence d'un système multi-agents de ce type vient de l'interaction avec un grand nombre d'autres agents.

- *l'école « cognitive »* : de tels systèmes sont composés d'agents dits « intelligents ». C'est le principe le plus utilisé. En raison de leur sophistication et de leur capacité à raisonner sur le monde, ils peuvent travailler de manière relativement indépendante, et résoudre des problèmes complexes. Du fait de leur» intelligence», ces agents cherchant à satisfaire leurs buts peuvent être amenés à revoir leurs croyances et leurs plans avec les autres agents.

La démarche de conception de systèmes multi-agents proposée par [Goblet et Benslimane, 1995] est la suivante :

① Organisation du domaine en deux niveaux. Le premier permet de définir les différents objectifs d'apprentissage associés au profil de l'apprenant. Le deuxième niveau explicite les différents points de vue associés à chaque objectif d'apprentissage.

② Définition de dépendances entre agents permettant d'expliciter les différents types de liens possibles. Rappelons qu'un agent A est en relation de dépendance avec un agent B si l'activation de A provoque une réaction de B. Il existe ainsi plusieurs sortes de dépendances : coopérative, logique, cognitive [Roose, 1997].

③ Représentation centrée des agents du domaine d'enseignement. Ceci permet de percevoir le domaine d'enseignement comme une composante active. Les agents sont ainsi définis à partir des objectifs d'apprentissage et des points de vue [Roose, 1997].

④ Etude des interactions du système multi-agents à partir des dépendances entre agents.

⑤ Définition d'un modèle de contrôle et de communication entre les différents agents supportant les différentes interactions du système.

⑥ Définition d'une architecture multi-agents.

[Ayala et Yano, 1996] proposent des agents intelligents qui assistent les apprenants et coopèrent, dans des environnements d'apprentissage collaboratif assisté par ordinateur, pour créer des possibilités de collaboration effective au sein d'un communauté virtuelle de pratique. [Ayala et Yano, 1996] a développé deux sortes d'agents logiciels : les *agents médiateurs* qui ont pour rôle de faciliter la communication et la collaboration entre apprenants, et les *agents de domaine*, qui fournissent une assistance pour une application adéquate des connaissances du domaine dans le réseau.

Les agents médiateurs coopèrent en échangeant leur croyances sur les aptitudes, les responsabilités et les objectifs des apprenants selon l'architecture fournie dans la figure suivante.

Chaque agent médiateur est capable de construire une représentation des possibilités de collaboration de l'apprenant - dont il est en charge - dans le groupe, en considérant les aspects structurels et sociaux du développement des connaissances. L'agent médiateur propose à l'apprenant de prendre en charge des tâches qui requièrent l'application d'éléments de connaissances issus des possibilités de collaboration de cet apprenant, ce qui a pour résultat d'accroître les possibilités de collaboration entre apprenant, de créer des zones de développement proches et donc davantage de possibilités d'apprentissage.

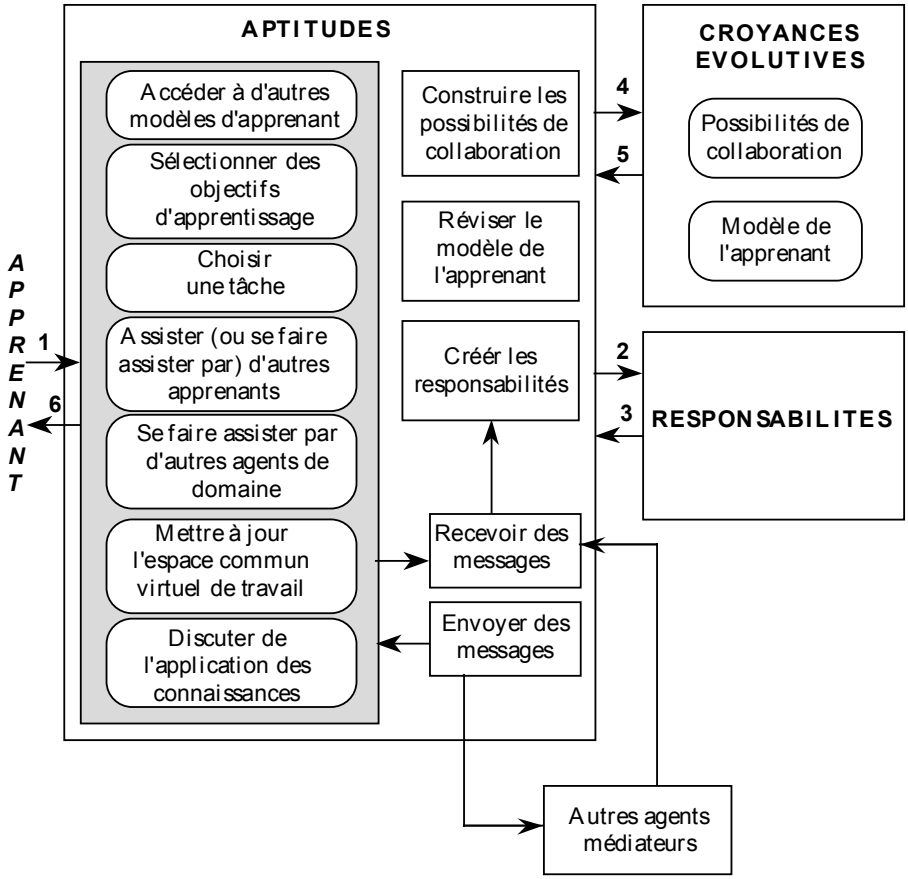

Figure 8 : Architecture d'un agent médiateur [Ayala et Yano, 1996]

C'est en ce sens que nous pouvons concevoir les différents outils et fonctions de notre environnement comme un ensemble *d'agents coopératifs* chacun ayant un rôle précis.

Par exemple, un agent serait chargé de sélectionner l'objet courant sur lequel va porter l'apprentissage. Un autre agent aurait pour mission de mettre à jour une base de connaissances contenant la liste chronologique des différents objets choisis ainsi que les paramètres de sélection de chacun d'entre eux. Un troisième agent pourrait demander au pédagogue de préciser, ou trouver automatiquement par déduction, la valeur de chaque variable pédagogique globale. D'autres agents auraient pour mission de veiller au bon déroulement de l'ensemble du système. Par exemple, nous pouvons imaginer que dans le cas d'une situation de transfert, le système privilégiera le parcours des liens de référence entre notions dans la matrice disciplinaire. Dans le cas d'une situation de remédiation, un agent devra aller chercher dans une base de connaissances un objet identique à celui étant à la cause du blocage mais dont le niveau de conceptualisation implique des difficultés moindres.

De même, en ce qui concerne le niveau de guidage général de l'apprenant, dans le cas d'un environnement d'apprentissage, le guidage sera nul et le système devra uniquement conserver les objets manipulés, l'ordre de parcours ainsi que le niveau de connaissances atteint. Par contre dans le cas d'un tuteur directif, le système devra vérifier la contrainte nécessaire suivante « une notion n'est maîtrisée que si *toutes* les notions composantes le sont ».

Même si une architecture à base d'agent semble, a priori, répondre à nos attentes, il conviendra de garder en mémoire les questions suivantes :
« Comment peut-on, grâce à des systèmes multi-agents, modéliser le principe de formation de coalition pouvant évoluer avec le temps ? »
« Comment chaque agent peut définir ses objectifs propres puis modéliser le fonctionnement global de recherche d'équilibre tout en gardant un système opérationnel ? »

Exemples d'agents logiciels

Les chercheurs travaillent à créer des logiciels capables d'assister l'utilisateur dans des tâches comme organiser un agenda, gérer le courrier électronique, …
Nous allons citer ci-dessous les exemples d'agents suivants : Maxims, Softbot, Assistant M, CAP, NewT, RINGO, WebWatcher et COACH dont nous avons puisé les références dans [Riecken, 1996], [Joachims *et al.*, 1995] et [Selker, 1996].

Maxims

Les travaux de Pattie MAES au MIT (« *Massachusetts Institute of Technology* »), par exemple, ont porté sur des agents informatiques capables de gérer le courrier électronique, d'organiser un agenda, de rassembler des informations et même d'aider au choix de divertissements dans le domaine du cinéma, de la télévision et de la lecture pour un utilisateur donné.
Elle a en particulier mis au point un agent nommé Maxims qui apprend à classer par ordre de priorité, effacer, faire suivre, trier et archiver le courrier électronique d'un utilisateur. Dans un premier temps, Maxims observe le comportement de l'utilisateur traitant sa correspondance électronique. Puis, avec sa permission, il se met un jour à le faire à sa place. Le courrier urgent est repéré et communiqué immédiatement à l'utilisateur, le courrier publicitaire est effacé et le courrier lu et signalé comme important par l'utilisateur est automatiquement archivé.
Un problème essentiel dans la mise au point de ce type d'agent concerne l'interaction entre agent et utilisateur. Le logiciel Maxims indique à l'utilisateur, au moyen d'une figure exprimant divers sentiments à l'écran, son degré d'assurance par rapport à telle ou telle tâche qu'il se propose d'effectuer. L'utilisateur garde donc entièrement le contrôle de l'agent et peut (ou non) lui donner le feu vert pour effectuer la tâche en question (effacer un message, par exemple). Le principe essentiel est celui-ci : Maxims commence par observer que des messages d'un certain type sont lus et archivés par l'utilisateur. Lorsqu'il pense avoir acquis une assurance suffisante, il présente à l'écran un visage exprimant cette relative confiance pour indiquer à l'utilisateur qu'il est prêt à archiver le message qui vient d'être lu. L'utilisateur peut alors lui donner le feu vert pour effectuer cette tâche. Ainsi, par la suite, lorsque des messages de même type seront lus par l'utilisateur, Maxims, encore plus confiant, pourra les archiver

automatiquement après lecture. L'approche suivie dans la mise au point de l'agent Maxims a également été appliquée par Pattie MAES au développement d'agents de gestion d'agenda, de filtrage de dépêches (NewT), de sélection de musiques etc…

Softbot

Oren ETZIONI et Daniel WELD, à l'université de Washington, ont également conçu et mis au point un agent informatique qu'ils appellent un *softbot* (pour « *software robot* »). Cet agent a pour fonction d'assister l'utilisateur dans l'exploration du réseau Internet. Son intérêt est triple. Premièrement, il fournit à l'utilisateur une interface expressive avec le réseau Internet. Deuxièmement, il détermine dynamiquement les logiciels nécessaires et leur ordre d'intervention pour effectuer une tâche spécifique sur Internet. Imaginons, par exemple, un utilisateur souhaitant envoyer un message électronique à Doug RIECKEN mais ne connaissant pas son adresse. Le softbot pourra faire appel dans ce cas au logiciel *netfind* pour rechercher son adresse électronique et comme il sait que pour utiliser le *netfind* il lui faut le nom de l'institution à laquelle appartient son correspondant, il commencera par consulter des bases de données bibliographiques à la recherche d'un rapport technique ou d'un article de RIECKEN susceptible de lui indiquer l'institution à laquelle celui-ci appartient. Troisièmement, le softbot pourra prendre automatiquement des mesures correctives lorsque certains éléments de son plan échouent. Il a donc la capacité de sélectionner dynamiquement une autre voie de façon à mener à bien une tâche pour l'utilisateur.

Assistant M

Au sein des Bell Labs, *Assistant M* est un autre type d'agent. Il se compose d'une série d'agents qui travaillent ensemble pour assister l'utilisateur d'un système de téléconférence multimédia. Ce type d'équipements, mettant à contribution des outils relevant de la micro-informatique et des télécommunications, permet à plusieurs personnes de travailler ensemble sans quitter leurs bureaux respectifs. Cette technologie crée une salle de conférence virtuelle où l'on peut participer à des séances de brainstorming, des réunions de conception, d'enseignement ou de simples rencontres. Cela permet aux participants de créer collectivement des documents, de présenter des diapositives, de dessiner et de prendre des notes sur des tableaux et de sauvegarder des informations pour un usage ultérieur.
Un écran d'ordinateur étant sensiblement plus petit qu'une salle de conférence, les participants sont à l'étroit pour travailler dans la salle virtuelle représentée sur leur moniteur. Le rôle de *Assistant M* est donc de gérer l'ensemble des documents créés pour l'utilisateur. *Assistant M* doit rassembler et organiser les divers documents (notes, listes et graphiques présentés au tableau, informations sauvegardées dans des fichiers) créés pendant la séance de travail dans la salle de conférence virtuelle, séance qui peut, parfois, se poursuivre plusieurs jours. *Assistant M* peut aider, par exemple, l'utilisateur à retrouver des documents connexes. Ceci est particulièrement utile lorsque l'utilisateur cherche des informations spécifiques. *Assistant M* peut ensuite présenter rapidement l'ensemble complet des informations utilisées lors d'une séance de travail.

Pour pouvoir remplir ce rôle, *Assistant M* doit effectuer plusieurs tâches. Tout d'abord, il doit observer les actions de l'utilisateur au cours d'une séance de travail ; ensuite, il doit reconnaître et « comprendre » ces actions et enfin, il doit gérer les divers objets sur lesquels portent ces actions. Imaginons, par exemple, qu'au cours d'une séance de travail, un utilisateur du système *Assistant M* et d'autres participants se servent d'un traitement de texte pour créer et modifier plusieurs documents. *Assistant M* s'efforcera de reconnaître l'ensemble des actions portant sur ces documents telles que : ajouter un paragraphe, introduire une image ou une illustration et supprimer un passage d'un document pour l'introduire dans un autre par un « couper-coller ». Ayant correctement interprété ces différents types d'actions, *Assistant M* peut assister l'utilisateur. I1 peut déduire des actions précédentes, par exemple qu'il existe un lien entre les deux documents impliqués dans le « couper-coller ». Par la suite, lorsque l'utilisateur se reporte à l'un des documents, *Assistant M* peut l'informer qu'il existe un autre document en relation avec le premier (il ne s'agit là que d'un exemple).

Lors de cette même séance de travail, les participants travaillent en commun sur beaucoup d'autres documents. Par la suite, l'utilisateur de M peut demander au système de faire apparaître en pile sur l'écran de son ordinateur l'ensemble des documents liés au document sur lequel il travaille. *Assistant M* lui présente alors ces documents dans un ordre déterminé par leur degré de relation avec le document de départ en lui expliquant le lien de chaque document avec celui-ci. Le document DocB, par exemple, se rapporte au premier document, DocA, du fait du « copier-coller » d'un paragraphe ; DocC se rapporte lui aussi au premier parce que l'utilisateur l'a ouvert et utilisé sept fois au moment de la création de DocA ; le document DocD se rapporte à DocA parce qu'il a été précédemment ouvert et placé à côté de DocA pendant la séance de travail, etc…

Le problème de recherche fondamental posé par *Assistant M* est celui de l'intégration des différents agents. Chaque agent est spécialisé dans un type de raisonnement. A l'heure actuelle, ces agents sont au nombre de cinq et sont spécialisés dans les raisonnements de type spatial, temporel, fonctionnel, structurel et occasionnel. Pour pouvoir reconnaître et interpréter des actions courantes dans une situation réelle, il est nécessaire de recourir à un grand nombre de types de connaissances et de raisonnements différents. L'objectif des études autour de l'assistant M est d'arriver à intégrer plusieurs agents spécialisés pouvant reconnaître et résoudre divers problèmes en s'appuyant sur une sorte de sens commun collectif du monde réel où travaille l'utilisateur.

NewT

NewT est une collection d'agents qui aident l'utilisateur à choisir les nouveaux articles à lire arrivant sur Internet. *NewT* permet de créer des agents (à l'heure actuelle quatre agents spécialisés en business, politique, sports et informatique) qui filtrent les nouvelles issues d'un service en ligne. Chacun de ces agents fournit des recommandations en fonction des actions passées de l'utilisateur. Ce sont donc des agents qui apprennent (« *learning agents* »). Ils observent en permanence le comportement de l'utilisateur et essaient de trouver des invariantes de façon à construire un « *pattern* ». Par exemple, ils peuvent remarquer que l'utilisateur lit chaque semaine les articles de tel journaliste dans le *LA Times*. Ainsi, ils proposeront chaque semaine les articles adéquats de façon automatique. Pour toute proposition automatique, l'utilisateur peut fournir une information signifiant l'intérêt positif ou négatif qu'il porte à cette proposition. Ceci permet à l'agent d'apprendre.

CAP

Un autre exemple d'agent informatique sera le système *CAP* (acronyme de « *Calendar Apprentice* ») conçu et mis au point par Tom MITCHELL et les étudiants de l'Université Carnegie-Mellon à PITTSBURGH. *CAP* s'adapte aux choix de son utilisateur en matière de planning. Ce système repose essentiellement sur la connaissance de l'emploi du temps de l'utilisateur et l'observation de la manière dont il organise ses rendez-vous. *CAP* commence par observer les habitudes de l'utilisateur avant de prendre la responsabilité d'organiser lui-même les rendez-vous. *CAP* sait se débrouiller avec la directive « *prendre un rendez-vous avec Chantal la semaine prochaine* ». Depuis le temps, l'agent a pu remarquer par exemple que l'utilisateur voit toujours Chantal entre midi et deux autour d'une pizza au *Laphitza*, le plus souvent le mardi. Il saura proposer un rendez-vous mardi à midi par courrier électronique et gérer la suite.
Le point critique des recherches de Tom MITCHELL concerne cette phase d'apprentissage du système *CAP*, reposant sur l'observation des habitudes.

RINGO

RINGO est un agent pour le WWW qui permet de mettre en relation des gens dont les goûts musicaux convergent ou semblent converger. Chaque utilisateur indique ses goûts musicaux à l'aide d'un coefficient allant de 1 à 7. Par exemple un utilisateur indique qu'il adore les Beatles mais déteste Madonna. Le système peut alors découvrir qu'un autre utilisateur a les mêmes goûts mais adore également Eric CLAPTON. Le système proposera alors à la première personne d'écouter de la musique d'Eric CLAPTON et cet utilisateur fournira alors son coefficient de qualité.
Le système possède beaucoup d'autres fonctionnalités et notamment le fait de pouvoir s'améliorer de façon autonome. En effet, il y avait au début 20 utilisateurs et 575 albums et au début de 1997, il y avait plus de 3000 utilisateurs pour 9000 albums. Ceci augmente considérablement les chances de trouver une personne ayant des goûts identiques aux siens.

WebWatcher

Une quantité phénoménale d'informations sont disponibles sur WWW et il devient alors très difficile de trouver efficacement une information particulière. *WebWatcher* est donc un logiciel permettant d'extraire des informations depuis une structure d'hypertexte. C'est un agent servant de guide dans WWW. Une fois que vous lui avez indiqué la sorte d'informations que vous recherchez, *WebWatcher* vous accompagne pour naviguer dans WWW en mettant en évidence des hyperliens qu'il juge intéressants. Sa stratégie pour donner des conseils se base sur les retours fournis par les navigations précédentes.

WebWatcher offre les quatre fonctionnalités suivantes :

- 1 - Mise en évidence sur la page WWW courante des hyperliens pour lesquels WebWatcher estime que l'utilisateur portera un intérêt ;
- 2 - Ajout sur la page WWW courante de nouveaux hyperliens selon l'estimation des intérêts de l'utilisateur ;
- 3 - Suggestion de pages en relation à la page courante ;
- 4 - Envoi de messages (sous forme de courriers électroniques) à l'utilisateur chaque fois que des pages spécifiées changent.

Lorsque *WebWatcher* suggère à l'utilisateur des hyperliens à suivre, l'utilisateur garde la maîtrise sur le système et peut ignorer les conseils. Ceci est important car *WebWatcher* peut donner des conseils imparfaits ou bien peut ne pas comprendre quelles sont les informations recherchées par l'utilisateur.

Le succès de *WebWatcher* dépend fortement de la qualité et de la quantité des connaissances dont il dispose. De part la nature dynamique propre à WWW, il semble difficile de garder la mainmise sur l'ensemble des connaissances présentes. Par conséquent, [Joachims *et al* ., 1995] explore des méthodes d'acquisition automatique des connaissances.

WWW est traité comme un graphe dont les noeuds sont des pages HTML et les liens des hyperliens. Il y a deux sortes de connaissances : celles contenues dans les noeuds, c'est à dire dans le texte des pages, et celles contenues dans les liens. La recherche d'informations dans le texte est traitée dans [Armstrong *et al.*, 1995]. Pour la recherche d'informations dans les hyperliens et les façons de s'en servir pour guider l'utilisateur, il convient de connaître pour chaque page, l'ensemble des pages lui faisant référence. Ainsi, l'algorithme de recherche préconisé par [Joachims *et al* ., 1995] se base sur le fait que deux pages ont un niveau d'intérêt équivalent si une troisième page pointe sur ces deux pages (en supposant de disposer de pages construites de façon non hétéroclite).

COACH

Depuis 1979, [Selker, 1996] construit des systèmes destinés à enrichir la communication entre l'ordinateur et l'utilisateur grâce à une couche sous-jacente de compréhension entre eux. Cette modélisation adaptative permet de diminuer la quantité d'informations que l'utilisateur doit chercher lui-même et elle augmente donc le temps qu'il peut consacrer à son travail. Jusqu'en 1985, des articles prévoyaient qu'un modèle adaptatif de l'utilisateur en temps réel était réalisable.

Le système COACH (*« COgnitive Adaptative Computer Help »*) [Selker, 1996] fut créé dans les années 1980 pour prouver que les *agents adaptatifs* pouvaient être utiles dans les scénarios d'interface utilisateur. COACH est un système qui enregistre l'expérience de l'utilisateur pour produire une aide personnalisée. Il apprend petit à petit à connaître son utilisateur afin de lui préparer à bon escient une aide correspondant à ses besoins, son niveau d'expérience et de compréhension. C'est un agent d'interface qui enseigne à l'utilisateur plutôt que d'agir pour son compte. COACH est un système qui n'intervient pas dans les actions de l'utilisateur mais qui les commente de façon opportune. COACH peut choisir l'explication ou l'exemple, la syntaxe, le minutage, le sujet, le style et le niveau d'aide en fonction de l'expérience et de la compétence constatées de l'utilisateur. Une description qui

présente une commande ou une fonction est utile au démarrage mais risque d'être ignorée si elle apparaît trop souvent. Des exemples montrant comment exécuter une procédure sont souvent précieux… jusqu'à ce que l'on maîtrise la procédure.

Le système COACH enregistre la compétence de l'utilisateur pour les choses qui peuvent être apprises (sur les plans syntaxique et conceptuel) à mesure qu'elles sont mises en œuvre. Il garde trace du moment où l'utilisateur a utilisé tel ou tel aspect du système, pour diriger son intervention et sa tactique pour l'aider. COACH est conçu pour expérimenter des modèles d'adaptation à l'utilisateur et peut être adapté à divers types d'application. Dans une même étude sur l'apprentissage du langage informatique LISP, le système COACH a permis à ses utilisateurs de faire cinq fois plus d'exercices que des témoins.

Grâce à des mécanismes d'apprentissage automatique, des paradigmes d'EAO peuvent oublier les programmes figés ou le style « salle de classe » pour se concentrer sur les besoins individuels de l'utilisateur. Le scénario d'enseignement déplace l'élève dans une situation d'apprentissage par la pratique. Le système COACH prouve que la technique des agents peut remplacer avec succès un formateur humain pour apporter une aide personnalisée pendant que l'élève cherche les solutions. Actuellement, [Selker, 1996] met le paradigme COACH à l'œuvre dans une interface utilisateur graphique.

L'industrie clame que la prochaine génération d'interfaces utilisateur inclura des agents intelligents. Le véritable intérêt de ces systèmes adaptatifs, ne sera pas que les gens aimeront leurs capacités d'adaptation, mais qu'ils ne remarqueront pas le mécanisme.

Mais attention, dans un article paru dans le journal « Libération » le 04 octobre 1996, la journaliste Nicole PERNICAUT dresse un aperçu général des « agents intelligents » existant sur Internet et conclue sur les risques qu'ils impliquent. En effet, au lieu de passer des heures à chercher sur Internet, un « agent intelligent » s'en charge ; il part fouiner sur le réseau pour voir ce qui vous intéresse. A partir de quelques mots, il livre une sélection « sur mesure » et, si elle ne plaît pas, on peut le lui dire, il comprend et il s'amende, toujours prêt à affiner sa recherche en fonction de vos desiderata. Avec à la clé une transformation radicale des modes de consommations de l'Internet vers un usage de plus en plus personnalisé : l'internaute est épié, scruté dans ses moindres désirs… pour le meilleur comme pour le pire.

Toutefois, nous pouvons nous demander si ces logiciels de personnalisation, qui modifient les modes de consommation de l'internaute, sont l'avenir du réseau des réseaux ? Que va-t-il se passer lorsque les internautes, déjà bien en peine aujourd'hui de contrôler toutes les sources d'information sur Internet, ne dépendront plus que de la sagacité des programmes informatiques ? En effet, « l'intelligence » d'un agent peut servir toutes les causes…

OMAGE : Outils et Méthode pour la spécification des connaissances au sein d'un Atelier de Génie Educatif

Résumé

Les nouvelles technologies de l'information sont entrées au cœur de notre société et provoquent de profonds changements dans notre vie quotidienne, notamment dans le monde du travail. Or le métier d'enseignant n'a pas vraiment évolué, même si les méthodes éducatives changent, car toute tentative d'introduction de l'informatique se heurte à la méfiance des enseignants qui ont peur de perdre leur liberté de choix éducatifs. De plus, les avancées technologiques n'ont d'intérêt que si elles sont intégrées dans un processus global de conception d'applications éducatives. Nos recherches ont donc pour objectif principal de faciliter la tâche de l'enseignant dans la préparation de ses séquences pédagogiques. Nous définissons ainsi le support méthodologique d'un environnement informatique d'aide à la spécification des connaissances éducatives.
Nous organisons alors nos travaux autour de trois axes.
Tout d'abord, nous proposons la mise en place d'enseignements axés sur la notion de situations-problèmes au sens IUFM car elle met les apprenants en situation de projet tout en répondant aux objectifs pédagogiques fixés.
Nous exposons ensuite la nécessité pour les enseignants de se reposer sur un processus de spécification formelle que nous définissons et pour lequel nous proposons un cycle de vie basé sur le prototypage rapide. Nous proposons aussi une ontologie de l'enseignement s'appuyant sur une architecture orientée-objet.
Nous montrons enfin que l'utilisation de méta-outils CASE permet de développer un environnement ayant une assistance adaptée et suffisamment flexible pour permettre différentes façons de spécifier et différents points de vue et-ou formalismes de représentation sur une spécification. Le prototype développé couple le méta-outil CASE HARDY, qui fournit une interface diagrammatique supportant les étapes du processus de développement, et le générateur de système expert CLIPS qui assure la cohésion globale en terme de guidage et de flexibilité.

Mots-clés :

Enseignement assisté par ordinateur (EAO) Génie logiciel Intelligence artificielle
Ingénierie des connaissances Approche objet Prototypage
Spécification formelle

MATICES : a Method And Tools for knowledge specification In an Courseware Engineering System

Abstract

New information technologies have now thoroughly entered our society and result in deep changes in our daily life, mainly in the working world. However, even if educational methods have changed, the teacher's job has not really evolved, because any attempt for introducing computer technology into this area raises the suspicion of teachers who fear to lose their freedom to make educational decisions. Moreover, technological improvements have no interest if they are not integrated in a global process of educational applications design. Our research therefore chiefly aims at facilitating the teacher task of preparing of his/her pedagogical sequences. We thus define the methodological support for a computer aided environment for educational knowledge specification.
Our work is then organised around three major axes.
First of all, we propose to organise teaching according to the notion of « problem-situations » because it puts the learners in the situation of a project while responding to the pedagogical objectives initially proposed.
Next, we raise the necessity for teachers to rely on a formal specification process that we duly define, and for which we propose a life-cycle based on rapid prototyping. We also propose an ontology of teaching founded upon an object-oriented architecture.
We finally show that the utilisation of meta-CASE tools allows to develop an environment which both supplies an adapted assistance and is sufficiently flexible to provide different specification methods and to allow different points of view and-or representation formalisms on a same specification. The developed prototype couples the meta-CASE tool HARDY, that offers a diagrammatic interface supporting all the stages of the development process, and the expert system generator CLIPS that ensures the global cohesion in terms of guidance and flexibility.

Keywords :

Computer-aided learning (CAL) Software engineering Artificial intelligence
Knowledge engineering Object approach Prototyping
Formal specification

Laboratoire d'Informatique Appliquée - LIA - UPPA

IUT de Bayonne Pays Basque - Château Neuf, Place Paul Bert - 64 100 BAYONNE