Conception d'un atelier d'expérimentation de logiciels éducatifs : application en géométrie

Marilyne Macrelle-Rosselle

Département de formation doctorale en informatique
UFR STMIA

École doctorale IAEM Lorraine

Conception d'un atelier d'expérimentation de logiciels éducatifs Application en géométrie

THÈSE

présentée et soutenue publiquement le 21 septembre 2001

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Marilyne MACRELLE-ROSSELLE

Composition du jury

Président et rapporteur interne :	Nacer Boudjlida	Professeur, UHP, Nancy 1
Rapporteurs :	Nicolas Balacheff	Directeur de recherches CNRS, Grenoble
	Alain Derycke	Professeur, USTL, Lille 1
Examinateurs :	Monique Grandbastien	Professeur, INAPG, Paris (Directrice de Thèse)
	Jean-Marc Labat	Professeur, Paris 5
	smoulins	Maître de conférences, UJF, Grenoble 1

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Université HENRI POINCARÉ
NANCY 1

Madame **MACRELLE - ROSSELLE**

DOCTORAT de l'UNIVERSITE HENRI POINCARE, NANCY-I

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER

Nancy, le 9 octobre 2001 n° 561

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ, NANCY 1
LE PRÉSIDENT
J.M. BURLET

Université Henri Poincaré, Nancy-I
24-30 rue Lionnois - B.P. 3069 - 54013 NANCY CEDEX
Tél. : 03 83 85 48 00 - Fax : 03 83 85 48 48

Remerciements

Je tiens d'abord à remercier ***Nacer BOUDJLIDA***, professeur Université Henri Poincaré (Nancy 1), pour l'important travail critique qu'il a fourni lorsqu'il a rempli son rôle de rapporteur interne au LORIA. Merci aussi pour avoir présider le jury de soutenance.

Je remercie ensuite ***Nicolas BALACHEFF***, directeur de recherches CNRS et directeur du laboratoire Leibniz (Grenoble), et ***Alain DERYCKE***, professeur à l'Université des Sciences et Technologies de Lille et directeur de recherche du Laboratoire Trigone, pour l'intérêt qu'ils ont porté à ce travail et pour avoir accepté d'en être les rapporteurs.

Merci à ***Jean-Marc LABAT***, professeur à Paris 5, qui a accepté de se joindre aux rapporteurs dans le jury et pour son intérêt pour ce travail.

Je remercie ***Monique GRANDBASTIEN***, Professeur à l'INA-PG (Paris), qui a encadré ce travail et qui m'a encouragée à le mener à terme.

Un grand merci à ***Josette MORINET-LAMBERT***, maître de conférences à l'Université Henri Poincaré (Nancy 1), pour son aide très concrète tout au long de ma recherche et au début de la rédaction.

Je remercie ***Virginie GOVAERE*** et mon mari, ***Frédéric ROSSELLE***, pour leurs encouragements et pour leur relecture minutieuse du manuscrit.

Je pense aussi à mon fils, ***Paul***, si adorable et si sage pendant la rédaction.

Merci aux chercheurs avec lesquels et j'ai pu collaborer, en particulier à ***Stéphanie JEAN-DAUBIAS***, maître de conférences à l'Université Claude Bernard (Lyon 1) et ***Jean-Michel BAZIN***, maître de conférences à l'IUFM de Reims, pour leur soutien et leurs encouragements.

Merci à mes collègues de bureau successifs ***Charun SANRACH***, ***Nicolas VAN LABEKE*** et ***Christophe CHOISY*** pour leur bonne humeur et pour l'excellente ambiance de travail.

Merci à ***Cyrille DESMOULINS***, maître de conférences à L'Université Joseph Fourier (Grenoble 1) qui m'a encadrée au tout début de ce travail.

Enfin merci à toutes les personnes qui m'ont aidée et encouragée tout au long de ce travail.

Je dédie cette thèse à la famille Varier et à Paul Delecroix.
Michel, Lucie et Albert Varier, membres du groupe national de diffusion et de recherche sur les Activités de Découvertes Techniques et Scientifiques (ADTS) au sein des CEMÉA (Centres d'Entraînement aux Méthodes d'Éducation Actives) m'ont permis de redécouvrir le plaisir de chercher et le plaisir de comprendre. Avec les membres du groupe, je me suis passionnée pour la construction des savoirs scientifiques. Les compétences et la reconnaissance que j'ai pu rencontrer dans ce groupe m'ont permis de sortir de la situation d'échec scolaire où je me trouvais.
Mon désir de reprendre mes études n'aurait pas pu se concrétiser sans la bienveillance de Paul Delecroix. Ce dernier était directeur de la formation continue à l'IUT de Lille I. Il m'a proposé la formation «multimédia». Cette formation, par correspondance et regroupements m'a permis de travailler à mon rythme, tout en conservant mon emploi de caissière. Elle a vraiment été un tremplin dans ma scolarité, tremplin qui m'a conduit jusqu'à cette thèse.

Table des matières

Table des figures

Liste des tableaux

Introduction

Le contexte de la recherche

Le domaine de l'informatique consacré aux **Logiciels Éducatifs Intelligents et Interactifs** peut être désigné par de nombreuses appellations : EAO (Enseignement Assisté par Ordinateur), EIAO (Enseignement Intelligemment Assisté par Ordinateur), EIAO (Environnements Intelligents d'Apprentissage avec Ordinateur), **EIAO (Environnements Interactifs d'Apprentissage avec Ordinateur)**, EIAH (Environnements Interactifs d'Apprentissage Humain) et TI (Tuteurs Intelligents) pour les sigles français, CAI (Computer Aided Instruction), ICAI (Intelligent Computer Aided Instruction), CAL (Computer Aided Learning), CBT (Computer Based Training), WBT (Web Based Training), ITS(Intelligent Tutoring Systems) pour les sigles anglais. *Nous avons, pour notre part, choisi la dénomination d'EIAO pour désigner ce domaine de recherche et de Logiciels Éducatifs pour les logiciels produits par ce domaine.* Notez que les Logiciels Éducatifs regroupent tous les types de logiciels utilisés à des fins éducatives, qu'ils soient intelligents ou non, interactifs ou non. Il englobe en particulier les logiciels qualifiés d'EIAO.

Les Logiciels Éducatifs ont pour objectif de favoriser les apprentissages. Pour atteindre cet objectif, un moyen est de mettre de nouveaux outils à la disposition des deux catégories d'utilisateurs directs : l'apprenant et l'enseignant. Ces outils, faits pour être pris en main par l'apprenant et/ou l'enseignant, sont conçus avec deux buts principaux. Le premier est d'aider les apprenants dans l'apprentissage d'une *discipline académique* (par exemple : mathématiques, français, langues, sciences physiques, *etc.*) ou d'un *savoir faire professionnel.* Le deuxième est de proposer de nouveaux outils aux enseignants pour enrichir leurs pratiques pédagogiques.

La recherche dans ce domaine doit permettre de faire évoluer les produits en adéquation aux besoins et aux nouvelles technologies. Pour cela, les chercheurs montent des expérimentations et des évaluations de nouveaux produits ou de nouvelles fonctionnalités. Plusieurs disciplines de recherche sont concernées : Didactique des disciplines, Sciences de l'Education, Psychologie Cognitive, Informatique, Sciences Cognitives, *etc.*.

En conséquence, la recherche a produit des *prototypes* de logiciels éducatifs intelligents et interactifs (Logiciels Éducatifs) pour l'enseignant et pour l'apprenant. Ils implantent des fonctionnalités comme l'explication, la simulation, la résolution de problème, le diagnostic, le support dans la réalisation d'une tâche élémentaire. Par exemple, l'édition d'une figure géométrique, l'aide à l'élaboration d'une démonstration (ou preuve) et l'aide à la rédaction de cette démonstration sont des fonctionnalités actuellement implantées dans le domaine de la géométrie.

Le travail de recherche présenté dans ce mémoire s'inscrit dans le domaine des EIAO et s'appuie sur le constat suivant : s'il est vrai que de nombreux prototypes sont développés dans les laboratoires, force est de constater que peu d'entre eux les quittent pour une utilisation effective dans les classes.

Outre les raisons liées à la nature expérimentale des approches ou des moyens matériels

mis en œuvre dans certains de ces prototypes de recherche, nous apportons cependant d'autres explications. Parmi celles-ci, la plus importante pour notre propos est que chaque prototype n'offre souvent qu'une partie des fonctionnalités existantes (simulation ou résolution de problème ou diagnostic ou explication).

Nous illustrons la situation d'un utilisateur qui veut réaliser une activité impliquant plusieurs fonctionnalités complémentaires, par exemple la résolution d'un exercice en géométrie, à la figure 1. Il dispose pour cela de fonctionnalités complémentaires (par exemple, l'analyse de l'énoncé ou l'édition de figure). Ces fonctionnalités sont implantées dans des prototypes (par exemple, édition de figure est implantée dans *CABRI-Géomètre* et dans *CHyPre*). L'utilisateur qui veut accéder à ces fonctionnalités doit alors «jongler» entre les différents prototypes. De plus, certains prototypes peuvent tourner en parallèle de ceux utilisés lors de l'activité (par exemple le prototype PACT analyse les interactions de l'utilisateur avec *CABRI-Géomètre* ou *CHyPre*).

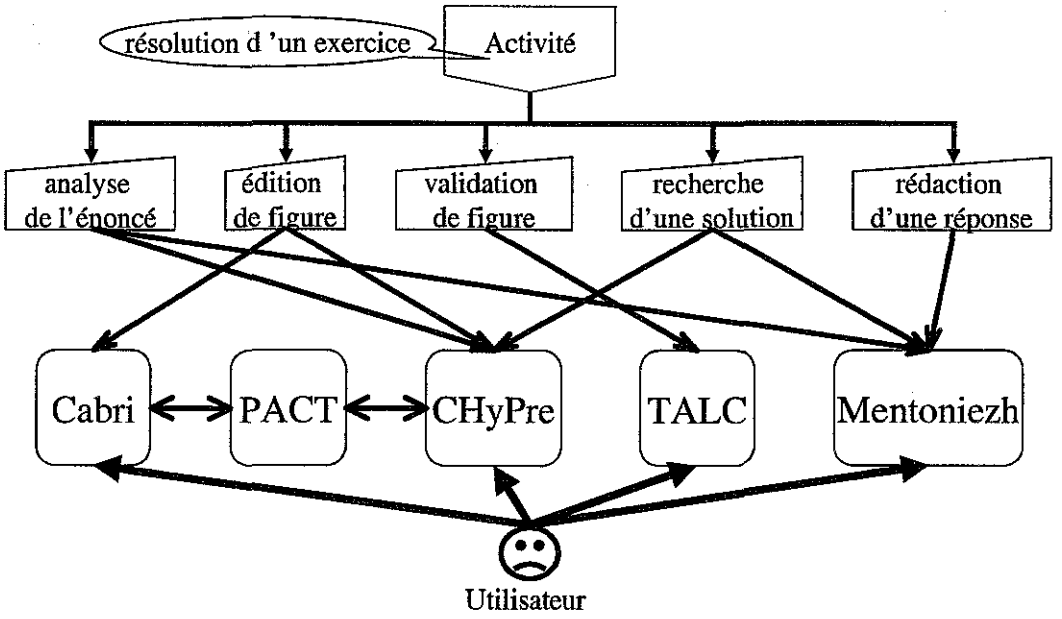


Figure 1 – Exemple d'activité

Par conséquent, il n'est aisé ni à l'enseignant de proposer une activité pédagogique impliquant plusieurs fonctionnalités complémentaires, ni à l'apprenant de repérer quelle fonctionnalité de quel prototype utiliser dans les tâches qu'il doit accomplir, ni aux chercheurs d'évaluer les prototypes ou d'expérimenter une nouvelle fonctionnalité. Pour remédier à ces difficultés, nous aimerions qu'un utilisateur puisse employer ensemble des fonctionnalités déjà implantées dans différents prototypes dans un environnement unique adapté.

Nous pensons, comme de nombreux auteurs l'ont fait remarquer, qu'il est nécessaire de favoriser l'utilisation conjointe de plusieurs prototypes possédant des fonctionnalités complémentaires. Par ailleurs, cet usage permettrait d'offrir un environnement plus riche aux apprenants et aux enseignants.

Pour cela deux approches sont possibles. La première approche consiste à refaire un produit intégrant plusieurs fonctionnalités en développant à nouveau entièrement toutes les fonctionnalités désirées. Cette méthode nécessite énormément d'efforts et d'expertises pour la conception et l'implantation de chaque fonctionnalité. La conséquence est qu'en dépit de ces efforts consi-

dérables, les prototypes résultants ne peuvent pas implanter toutes les fonctionnalités désirées.

La seconde approche consiste à faire coopérer les prototypes au sein d'un environnement unique afin de tirer partie de leurs fonctionnalités complémentaires. Cette approche a l'inconvénient de ne pas pouvoir être directement mise en œuvre car ces prototypes n'ont été conçus ni pour être intégrés ni pour coopérer. Cependant de nombreux auteurs ([Cheikes 98, Koedinger 98, Ritter 96, Ritter 98, Suthers 97]) la privilégient actuellement en proposant des architectures de composants (un composant pour un prototype ou un composant pour une fonctionnalité) qu'il est possible d'assembler pour élaborer un environnement donné.

Notre approche consiste à étudier les conditions requises pour amener les prototypes à coopérer. Pour cela, nous définissons un environnement permettant l'utilisation conjointe de plusieurs prototypes pouvant si possible coopérer voire interagir les uns sur les autres (inter-opérer) et ainsi partager des ressources et échanger des données. Nous appelons cet environnement un **atelier** dans la suite de ce mémoire.

Objectifs du travail

Notre objectif principal est de spécifier et de développer un atelier, c'est-à-dire une plate-forme logicielle, permettant l'utilisation conjointe de plusieurs prototypes. À long terme, l'atelier doit permettre à l'enseignant de choisir les fonctionnalités dont il a besoin en fonction de l'activité pédagogique qu'il a prévue avec les apprenants. Il doit aussi permettre de réduire la charge cognitive de l'apprenant en lui présentant uniquement la fonctionnalité dont il a besoin à un moment donné.

À court terme, l'atelier sera expérimenté en recherche afin de montrer la faisabilité d'un tel projet. Dans le cadre de ce travail, nous limitons le développement aux fonctionnalités dédiées à ces expérimentations sans chercher à finaliser une interface ergonomique de sélection destinée à l'enseignant, futur utilisateur.

Les justifications du besoin de cet atelier sont multiples.
Il s'agit d'abord, en «intégrant» des prototypes existants, de faciliter la réutilisation de ceux-ci (notion de composants réutilisables). En effet, cette réutilisation permet de capitaliser l'expertise nécessaire et d'optimiser les efforts. Par conséquent, elle facilite la conception et la réalisation d'expérimentations sans avoir à reconstruire tous les composants logiciels nécessaires.
Il s'agit ensuite de permettre à l'utilisateur de choisir les fonctionnalités dont il a besoin. Pour cela, nous présentons un éventail de fonctionnalités proposées par défaut. Elles servent de briques de base à la définition de l'expérimentation. Il faut donc identifier les fonctionnalités souhaitées par l'utilisateur, permettre un accès aisé à celles-ci (les indexer) et finalement faciliter la sélection et l'enchaînement de ces briques par l'utilisateur.
Il s'agit enfin de favoriser l'accès à des ressources à distance. En effet, la formation à distance se développe et nécessite de permettre l'accès à des ressources qui ne peuvent pas être implantées localement. De plus, en permettant à chaque prototype d'être situé sur des machines différentes éventuellement sur un réseau d'ordinateurs, le passage du paradigme «un logiciel sur une machine» vers le paradigme «des logiciels distribués sur un réseau d'ordinateurs» est facilité.

Pour ce faire, nous avons mis l'accent sur les quatre axes de travail suivants :

- la conception de l'atelier ;
- la réalisation d'une maquette pour montrer la faisabilité d'un tel atelier ;
- la définition des conditions de l'utilisation conjointe de plusieurs prototypes ;
- et la formulation de recommandations («guidelines») pour favoriser la communication et la coopération inter-logiciels et cela, dès la conception des produits.

Ces axes de travail correspondent aux objectifs de recherche ci-dessous. Il s'agit d'étudier et de définir :

- des modèles pour décrire (indexer) une ressource pédagogique, plus particulièrement des applications, afin de la retrouver et de faire appel à ses fonctionnalités ;
- des concepts et des outils pour la communication de données entre applications ;
- des concepts, des langages et des normes pour commander un logiciel de l'extérieur ;
- des concepts, des langages pour exporter les interfaces d'un logiciel ;
- et des concepts, des langages et des outils pour la collecte de traces d'interactions didactiques.

Ces objectifs de recherche prennent place dans le contexte particulier de l'EIAO. La problématique générale présentée ici, est détaillée au chapitre 1.

Domaine d'application : la géométrie

Nous nous proposons d'aborder cette problématique à partir de la réalisation d'un atelier d'expérimentation des Logiciels Éducatifs de géométrie. Le choix du domaine d'application de ce travail se justifie par les raisons énoncées ci-dessous.

Premièrement, les travaux antérieurs de l'équipe *Informatique et Formation* concernant l'enseignement de la géométrie dans le plan (avec les environnements d'apprentissage *Calques 2* [Bernat 94a] et *CHyPre* [Bernat 94b] et le prototype *TALC* [Desmoulins 94] et dans l'espace (avec *Dessiner l'Espace* [Bernat 89], *Pratiquer l'Espace* [Bernat 91] et *Calques 3D* [Van Labeke 99]), nous permettent de disposer d'une expertise dans le domaine des logiciels pour l'enseignement de la géométrie.

Deuxièmement, les travaux nationaux (*CABRI-Géomètre* [Baulac 90, Baulac 92], *TéLéCABRI* [Tahri 93], *Géospace* [Authier 98], *Cabri 3D* [Qasem 97], *CABRI-Euclide* [Luengo 97a], *CABRI-DéFI* [Giorgiutti 91, Baulac 91], *Atelier de Géométrie 3D* [Lepine 97]) et internationaux (*PACT* [Ritter 96] et *Geometer's Sketchpad* [Jackiw 95]) de la recherche en EIAO fournissent de nombreux autres prototypes ou produits aux fonctionnalités variées parmi lesquelles l'**édition de figure** (*CABRI-Géomètre* [Baulac 90, Baulac 92], *Geometer's Sketchpad* [Jackiw 95]), l'**élaboration d'une démonstration** (*Mentoniezh* [Py 90]), *etc.*. Cette variété de fonctionnalités proposées permet de disposer d'une combinatoire de possibilités de coopération entre prototypes.

Troisièmement, certains de ces prototypes ont été conçus dans des équipes voisines, ainsi leurs auteurs sont disponibles pour une éventuelle adaptation.

Quatrièmement, la géométrie est un domaine bien formalisé qui a donné naissance à de nombreuses représentations des concepts géométriques ; ainsi c'est un bon champ d'expérimentation pour permettre l'échange de connaissances du domaine.

Enfin, nous avons prévu une collaboration avec les chercheurs (Nicolas Balacheff, Vanda Luengo) travaillant en didactique des mathématiques et des auteurs de logiciels (Jean Marie Laborde, auteur de *CABRI-Géomètre*) qui étaient intéressés par un tel atelier dans le cadre de l'appel a projets[1] du PRC-GDR IA de 1996.

[1] Le projet soumis dans ce cadre a été accepté mais non financé. La collaboration prévue dans ce cadre n'a donc pas eu lieu.

Exemple introductif

Afin d'esquisser ce vers quoi nous tendons avec cet atelier, nous présentons ici un scénario d'expérimentation de plusieurs logiciels complémentaires sur un type d'exercice de géométrie. Celui-ci consiste à démontrer une propriété à partir d'un énoncé définissant des objets géométriques et des propriétés donnés en hypothèse.
Soit, par exemple, l'énoncé suivant (voir figure 2) :

```
Soient A, B, C, K quatre points alignés tels que
  B soit le milieu de [A C]
  et C soit le milieu de [B K].
Soit M le milieu de [E K].
(A M) coupe (B E) en F.

Montrer que F est le mileu de [A M].
```

Nous considérons que la résolution de l'exercice par un apprenant comporte 3 *phases* :

1. comprendre l'énoncé ;
2. rechercher une solution ;
3. rédiger une réponse.

La *phase* 1 commence par l'**édition de la figure**[2]. Elle se poursuit par l'**analyse de l'énoncé** qui réclame l'identification des hypothèses du problème et de la conclusion à laquelle l'apprenant doit aboutir. Les hypothèses constituent les faits à partir desquels l'apprenant pourra raisonner dans la phase suivante, tandis que la conclusion n'a qu'un statut de conjecture (c'est-à-dire de fait à prouver). La *phase* 2 constitue le processus de **résolution de problème**. Elle comprend la recherche d'une solution et implique la mise en œuvre de raisonnements (**aide au raisonnement**). La *phase* 3 consiste à présenter la solution par écrit pour une validation de celle-ci par d'autres apprenants ou par l'enseignant. Elle implique pour cela, un formalisme de rédaction de solution (**aide à la rédaction**).
Nous ne disposons pas d'un prototype unique nous permettant de suivre toutes ces *phases* de l'exercice. Cependant, l'utilisation de plusieurs prototypes permet de traiter cet exercice de bout en bout.
◇ Ainsi, l'apprenant construit une figure géométrique correspondant à l'énoncé de l'exercice afin de s'appuyer sur cette figure pour résoudre l'exercice. Il utilise ici les fonctionnalités d'**édition** et d'**exploration** de figure offertes par *CABRI-Géomètre* [Baulac 90, Baulac 92].
◇ Ensuite, l'apprenant demande si sa figure est correcte vis à vis de l'énoncé. Il utilise la fonctionnalité de **diagnostic de correction de figure** offerte par *TALC* [Desmoulins 94].
◇ Ensuite, l'apprenant analyse l'énoncé pour différencier les hypothèses de la conclusion. Il utilise les fonctionnalités d'**analyse de l'énoncé**, de **diagnostic** et d'**aide** de *Mentoniezh* [Py 90].
◇ Ensuite, il utilise les fonctionnalités d'**aide au raisonnement**, d'**extraction de sous-figures** et de **rappels de cours** de *CHyPre* [Bernat 94b] pour disposer d'une aide visuelle lors de sa recherche d'une solution.
◇ Finalement, il utilise la fonctionnalité d'**aide à la rédaction** de *Mentoniezh* pour permettre à l'apprenant de structurer la présentation de sa solution.
De plus, parallèlement à toutes ces étapes, les fonctionnalités d'**analyse des interactions** et d'**aides dans l'interaction** du tuteur *PACT* [Ritter 96] aident l'apprenant dans son interaction avec les prototypes *CABRI-Géomètre* et *CHyPre*.

[2]Le gras dans cette section, signale une fonctionnalité proposée à l'apprenant par un prototype pour accomplir sa tâche

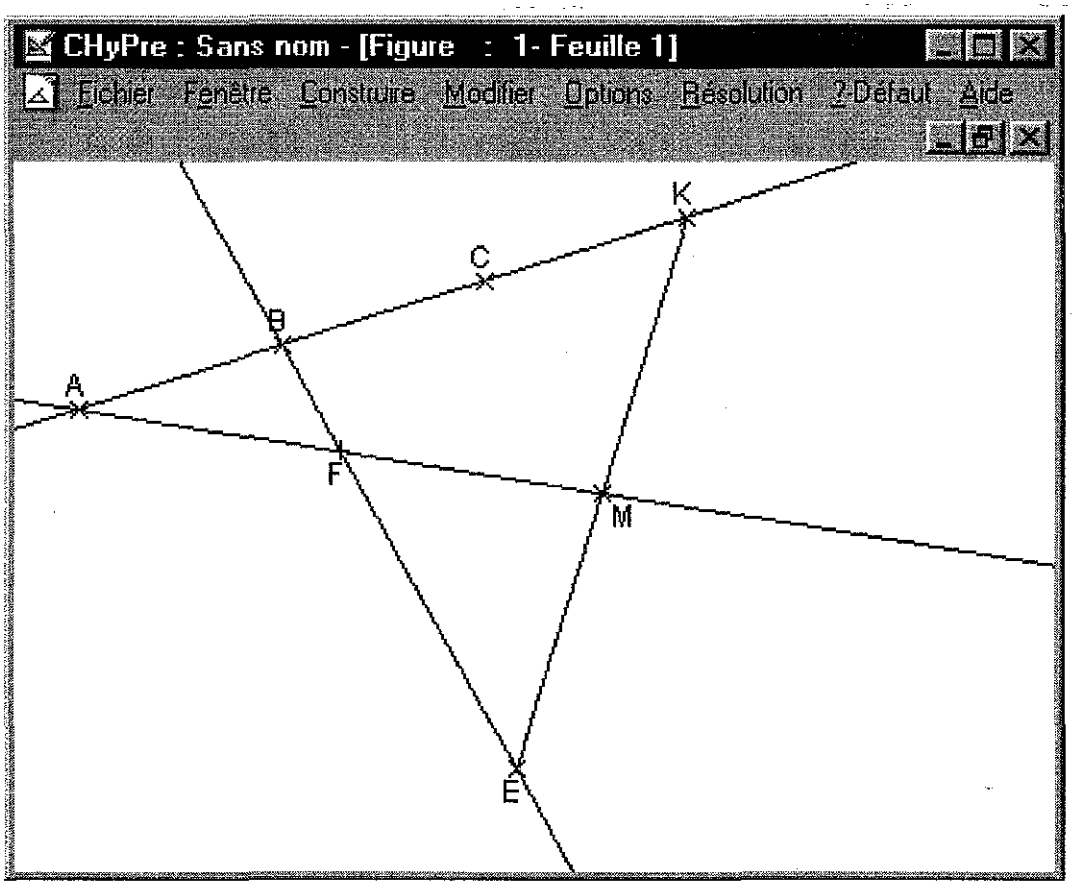


Figure 2 – Figure géométrique correspondant à l'énoncé de l'exemple introductif

En résumé, dans cet exemple, nous souhaitons que l'atelier permette la mise en œuvre des cinq prototypes précités pour la réalisation de l'exercice.

Structure de ce mémoire

Le chapitre 1 est un chapitre préliminaire dont le but est de préciser le définition du système introduit ici et les problématiques de recherche. Sans entrer dans les détails techniques, il permet d'esquisser le contexte d'utilisation de l'atelier. Il détaille les objectifs de recherche pour la conception de l'atelier et la définition des conditions de l'utilisation conjointe de plusieurs prototypes. Il donne une première version du cahier des charges de l'atelier en termes d'objectifs à remplir et de propriétés que doit posséder l'atelier. Du point de vue des concepteurs de logiciels, on y trouve notamment la liste des propriétés qu'il est souhaitable qu'un prototype possède afin de faciliter son utilisation dans l'atelier. Il précise enfin le rôle des différents types d'utilisateurs.

Le chapitre 2 est consacré à l'état des travaux actuels concernant l'utilisation conjointe de logiciels quelconques. Nous présentons tout d'abord quelques prototypes de géométrie. Puis, nous précisons ce que nous désignons par «utilisation conjointe de logiciels». Nous présentons ensuite les produits existants permettant l'utilisation conjointe de logiciels quelconques. Nous illustrons ceux-ci avec les travaux relatifs à la coopération de logiciels éducatifs, en particulier en géométrie. Nous poursuivons en présentant les tentatives de normalisation qui peuvent concerner l'utilisation conjointe de logiciels dans un contexte éducatif. Nous terminons par une synthèse des apports

de ce chapitre et une discussion de ceux-ci pour notre atelier.

À partir de l'analyse des besoins des chercheurs et des tentatives de coopération vues dans le chapitre 2, nous détaillons dans le chapitre 3 les propositions de recherche liées aux besoins identifiés dans le chapitre 1. Ce chapitre présente le détail des propositions pour la conception de l'atelier. Il apporte la spécification détaillée des différents composants retenus.

Le chapitre 4, plus technique, décrit l'implantation des propositions faites dans le chapitre 3. Nous obtenons ainsi la première version de la maquette de l'atelier et son architecture. Du point de vue des concepteurs de logiciels, on y trouve la façon de décrire un prototype pour son utilisation dans l'atelier. Du point de vue du prescripteur, on y trouve les modalités de construction d'une activité pour l'élève faisant appel aux fonctionnalités disponibles sur l'atelier.

La conclusion de ce mémoire reprend l'état d'avancement et les résultats obtenus pour chaque proposition. Elle expose l'évaluation faite de l'atelier. Nous présentons enfin les apports de ce travail et abordons les perspectives de recherche qu'il est permis d'envisager.

Chapitre 1

Un atelier d'expérimentation de logiciels éducatifs

Dans l'introduction nous avons présenté les objectifs scientifiques poursuivis et le système développé dans le cadre de cette thèse. L'objectif de ce chapitre est de proposer à une définition précise de ce système, identifiant des fonctions*[3] et des propriétés.
Nous reprenons d'abord les descriptions du système tel qu'il est présenté dans l'introduction. Nous complétons ensuite ces descriptions afin de raffiner la définition du système (section 1.1). Nous introduisons ensuite les problématiques de recherche liées à la conception et aux besoins de l'atelier (section 1.2). Enfin, nous présentons des concepts formalisés (section 1.3) et des bases de modélisation (section 1.4) sur lesquelles nous nous appuierons dans les chapitres suivants lors de la présentation de nos propositions et de notre implantation.

Description du système

Dans cette section nous reprenons les descriptions du système apparues dans l'introduction. Les termes sont précisés dans la suite du chapitre.

Description 1.1 *Le système est une plate-forme logicielle.*

Le système est un *environnement informatique* dans lequel différents *logiciels* sont mis à la disposition d'un *utilisateur*.

Description 1.2 *Le système permet d'utiliser un ensemble des fonctionnalités*[4] *déjà implantées dans différents prototypes dans un environnement unique lors d'une activité*[5].

Nous considérons un *utilisateur* qui réalise une *activité* nécessitant plusieurs *prototypes*. L'utilisateur réalise son activité dans un environnement unique, c'est-à-dire sur une machine donnée. Chaque prototype peut tourner sur une machine distincte de celle sur laquelle l'utilisateur réalise son activité. Par conséquent, l'*architecture du système est distribuée*. De plus, les logiciels de la plate-forme sont des prototypes dont nous voulons utiliser les fonctionnalités.

Description 1.3 *Le système permet à l'utilisateur de choisir les fonctionnalités dont il a besoin.*

Description 1.4 *Le système permet de réduire la charge cognitive de l'utilisateur, en lui présentant uniquement les informations pertinentes à un moment donné.*

[3] Les mots ou expressions suivis d'un astérisque sont dans le glossaire

[4] Nous parlons de **fonctionnalité** pour les «fonctionnalités» liées au domaine d'application choisi pour l'activité menée avec l'atelier. Tandis que nous parlons de **fonction** pour désigner les «fonctionnalités» offertes par l'atelier

[5] Larousse : Action d'une personne, d'une entreprise, d'une nation dans un domaine défini.

Les descriptions 1.3 et 1.4 débouchent sur l'idée que pour l'utilisateur, tout se passe comme s'il utilisait un «*prototype virtuel*» dont les fonctionnalités sont celles de son choix. Sur la figure

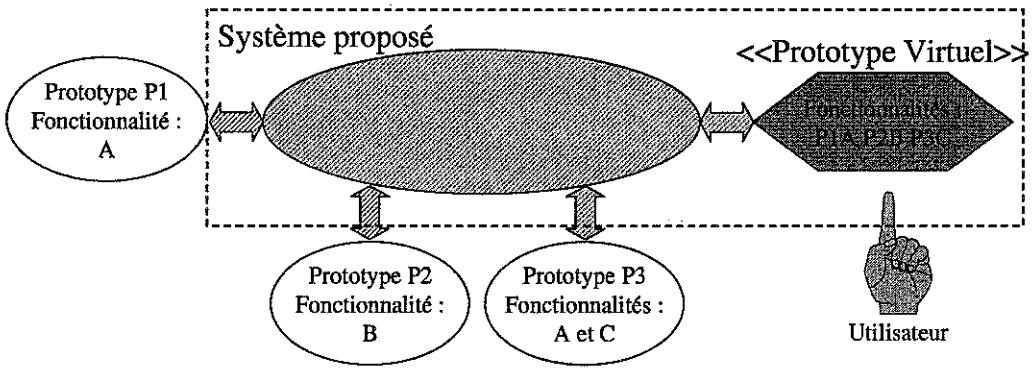


Figure 1.1 – Le système proposé

1.1, nous avons, par exemple, trois prototypes P1, P2 et P3, fournissant respectivement les fonctionnalités A, B et C. L'objectif du système est que pour l'utilisateur tout se passe comme s'il avait sur sa machine toutes les fonctionnalités choisies dans les différents prototypes : ici la fonctionnalité 'A' de P1, la fonctionnalité 'B' de P2 et la fonctionnalité 'C' de P3.
Le système décrit sur la figure 1.1 comprend un équipement qui sert d'intermédiaire (partie hachurée de la figure 1.1) entre les différents prototypes impliqués et le «prototype virtuel» (dans l'hexagone). Les flèches sur cette figure correspondent à une mise en relation entre les divers éléments.

Description 1.5 *Le système offre à des prototypes qui en ont la capacité*[6] *la possibilité d'interagir avec d'autres prototypes.*

Description 1.6 *Le système permet à ces prototypes de partager des ressources et d'échanger des données.*

Les descriptions 1.5 et 1.6 précisent certaines fonctions offertes par le système : supporter l'interaction entre prototypes, le partage de ressources et l'échange de données. De plus, ils introduisent des propriétés qui permettent aux prototypes d'être utilisés par l'atelier : une capacité à interagir et à échanger des données.

1.1 Définition du système

1.1.1 Définitions

Atelier L'atelier est un environnement informatique dans lequel différents prototypes sont mis à la disposition d'un utilisateur pour mener (concevoir, réaliser et exploiter) une activité. L'atelier désigne l'ensemble du système.

Activité Une activité est ensemble de tâches que le sujet doit exécuter, par exemple, une expérimentation ou un exercice. Une activité emploie des prototypes.

[6] Nous précisons cette notion de capacité par la suite. De plus, nous traitons le cas des prototypes existants — qui n'ont pas cette capacité — dans la suite du manuscrit.

Pourquoi appeler notre système un atelier ?

Le système est destiné à un utilisateur qui réalise une activité nécessitant plusieurs prototypes de Logiciels Éducatifs. Historiquement, il est d'abord destiné à un utilisateur qui est un chercheur. Le type d'activité qu'un chercheur réalise avec le système est une expérimentation. Le système est donc d'abord un environnement de réalisation d'expérimentation.
La dénomination du système repose sur l'analogie entre la conception d'une expérimentation de Logiciels Éducatifs et la conception d'une expérimentation d'objets en sciences expérimentales. Pour illustrer notre propos, nous choisissons comme objets des micro-fusées[7]. Une expérimentation de micro-fusées se déroule dans un atelier. Par conséquent nous proposons qu'une expérimentation de prototypes se déroule dans un atelier. Nous l'appelons : **atelier d'expérimentation de logiciels éducatifs**. Nous le désignons par la suite simplement par **atelier***.
Les sections 1.1.2 et 1.1.3 décrivent cette analogie. Elles permettent de mieux cerner le rôle de cet atelier. Nous définissons tout d'abord le concept d'atelier, dans le cas général puis dans le cas d'une activité de conception d'une expérimentation de Logiciels Éducatifs. Cette définition nous permet aussi de voir s'ébaucher l'architecture de l'atelier, c'est-à-dire les différentes parties qui la compose.

1.1.2 Définition générale d'un atelier

Un atelier est une salle où se pratique une activité donnée. L'atelier comporte des postes. Un poste[8] est un coin de l'atelier destiné à une fonction déterminée. Le **poste principal** est celui où a principalement lieu l'activité. Les **autres postes** sont utilisés pour réaliser certaines parties de l'activité. Un **espace de cheminement** permet d'accéder aux différents postes. L'**organisation de l'atelier** dépend de la place réservée aux différents postes et des caractéristiques physiques de l'espace de cheminement.

Par exemple, dans un atelier de conception de micro-fusées, le **poste principal** est celui où la fusée est conçue (figure 1.2). C'est là que le concepteur de la fusée décide des différentes parties à construire et de l'ordre dans lequel la construction est faite. C'est aussi l'endroit où l'activité de construction commence et se termine. À l'issue de la construction, l'utilisateur dispose alors d'une fusée prête pour une campagne de lancement.
Les **autres postes** sont ceux par lesquels il est nécessaire de passer pour obtenir l'objet fini. Le concepteur de la fusée passe, par exemple, par le poste de découpe des ailerons. Les tâches possibles à chaque poste dépendent des outils et matériaux qui s'y trouvent, ainsi que des objets qu'il est permis d'y apporter, des consignes[9] d'utilisation, *etc.*. L'utilisateur du poste peut prendre connaissance de tout cela (tâche possible, des outils et matériaux disponibles, *etc.*) avant de choisir d'utiliser ou non ce poste. Nous appelons l'ensemble de ces informations mises à la disposition de l'utilisateur du poste une «**vitrine***».
L'espace de cheminement permet d'accéder aux divers postes. Il permet aussi le transport d'objets entre les différents postes et jusqu'au poste principal.
L'organisation de l'atelier (donc la disposition des postes) dépend des caractéristiques physiques de la salle où l'atelier se déroule. Cependant l'activité qui s'y déroule n'est pas influencée par les caractéristiques physiques de la salle. En effet, dans une autre salle, les postes seraient disposés autrement et le cheminement pour passer d'un poste à l'autre en serait modifié, sans

[7]Une micro-fusée est un modèle réduit de fusée. Sa taille approximative est de 30 cm. Elle est construite à partir de divers matériaux : bois, carton, *etc.*. Cette fusée dispose d'un moteur à poudre permettant sa propulsion lors de son lancement et pendant le vol.

[8]Larousse : Local, lieu affecté à une destination particulière, où qqn, un groupe remplit une fonction déterminée.

[9]Larousse : instruction formelle donnée à qqn qui est chargé de l'exécuter.

pour cela affecter l'activité de conception d'une fusée. L'architecture d'un tel atelier est présentée à la figure 1.2. Par abus de langage, nous désignons par atelier, la salle et son contenu.

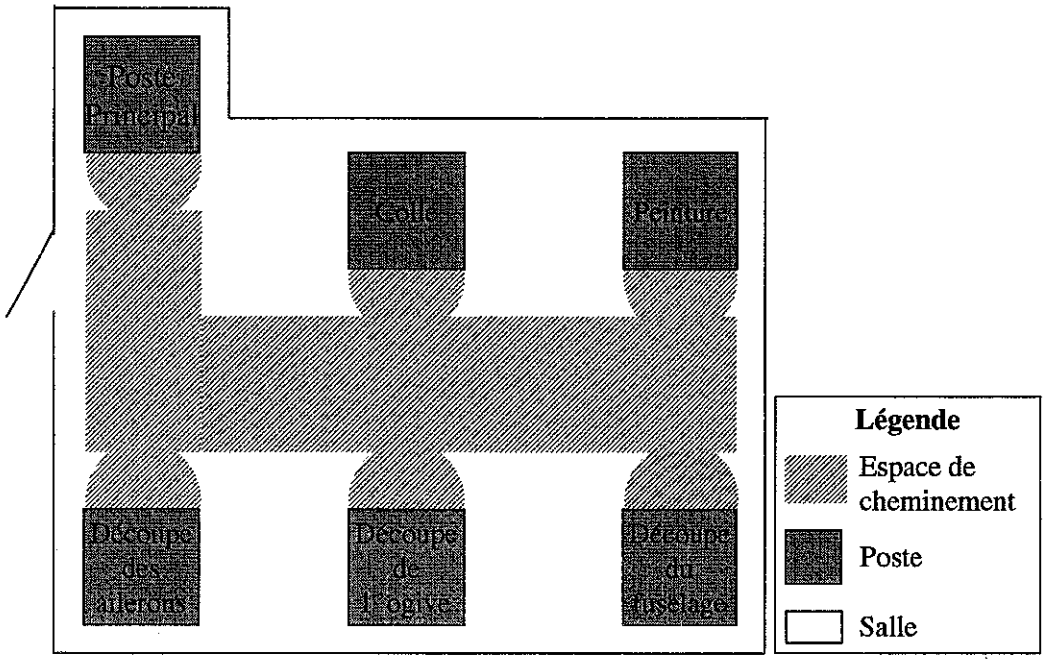


Figure 1.2 – Exemple d'atelier : cas de l'atelier micro-fusées

Du point de vue informatique, un atelier n'est plus une salle, mais un environnement informatique (exemple : AGL — Atelier de Génie Logiciel). Les postes correspondent à des outils logiciels, des composants informatiques ou des machines. L'espace de cheminement est au niveau du système d'exploitation ou du réseau. Nous décrivons ci-dessous notre conception informatique d'un atelier.

1.1.3 Définition de l'atelier d'expérimentation

Par analogie, dans un atelier d'expérimentation de logiciels éducatifs intelligents (figure 1.3), l'analogue d'un poste est un **composant**. Un composant[10] est un équipement de l'atelier destiné à une fonction déterminée.

Le **composant principal** de l'atelier est celui qui permet la conception, la réalisation et l'exploitation d'une expérimentation nécessitant un ensemble de logiciels éducatifs intelligents. C'est là que le concepteur décide de la partie informatique de l'expérimentation qui va avoir lieu : quels sont les prototypes à utiliser, dans quel ordre et pourquoi faire, *etc.*. Nous appelons ce composant le *gestionnaire d'activités*.

Les **autres composants** permettent chacun de réaliser une tâche précise qui constitue une partie de l'activité. Cette tâche consiste à soit assurer un service, dans ce cas, c'est un **composant de service**, soit fournir une fonctionnalité éducative, ici le composant est un Logiciels Éducatifs. Les tâches que permet chaque composant dépendent de la spécification du composant. Cette dernière est rendue publique via une **vitrine**.

Composant Un composant est un équipement de l'atelier destiné à une fonction déterminée, une tâche précise.

[10] Larousse : Élément constitutif d'un ensemble complexe.

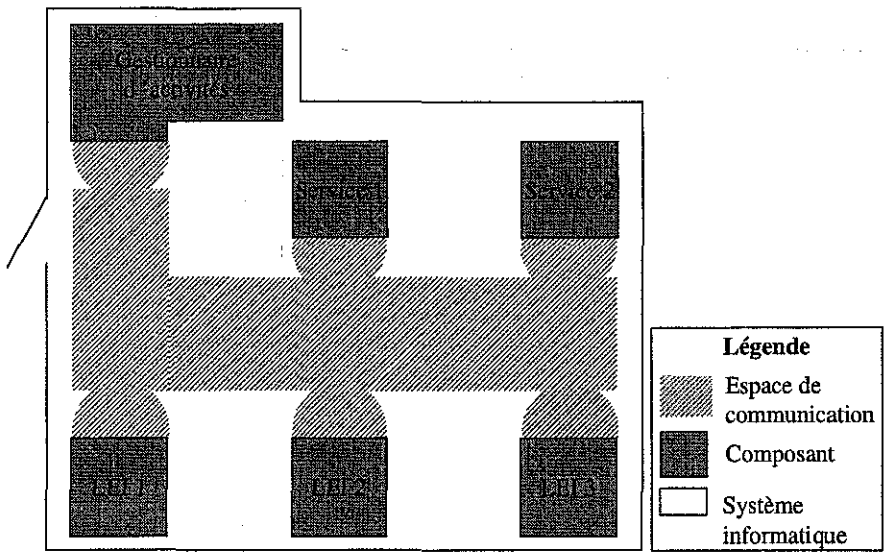


Figure 1.3 – Atelier d'expérimentation de logiciels éducatifs

Vitrine Une vitrine est ce dans quoi est publiée/affichée la spécification publique d'un composant, c'est-à-dire l'ensemble des informations mises à la disposition des autres (composants et utilisateurs).

L'espace de communication permettant d'accéder aux composants est l'analogue de l'espace de cheminement.
Des données sont recueillies lors de l'expérimentation, et en particulier les objets informatiques produits pendant celle-ci (fichiers, par exemple). Elles constituent des résultats à interpréter.
L'espace de communication permet d'accéder aux divers composants. Il permet aussi le transport d'«objets» (donc de données) entre les différents composants.
L'organisation de l'atelier dépend des caractéristiques physiques du système informatique (ordinateur unique, réseau d'ordinateurs, *etc.*).
Par abus de langage, nous appelons atelier, le système informatique concerné incluant les composants et l'espace de communication. La figure 1.3 illustre sans la détailler l'architecture de notre atelier. Reste maintenant à préciser la place de l'utilisateur par rapport à ce dernier.

1.1.4 Rôle des utilisateurs

Trois types d'acteurs utilisent l'atelier : l'administrateur, le prescripteur, le sujet.
L'**administrateur*** est l'utilisateur qui prend en charge tous les aspects techniques de l'atelier. Il règle la configuration et les paramètres de l'atelier en fonction des caractéristiques matérielles : nombre et types de machines, type de réseau, prototypes en présence, *etc.*).
Le **prescripteur*** est l'utilisateur qui conçoit l'activité exécutée par le sujet. Ce prescripteur est soit l'enseignant, soit le chercheur. La mise en place technique étant prise en charge par l'administrateur, le prescripteur se préoccupe uniquement de la mise en place de son activité (expérimentation pour le chercheur ou activité pédagogique —exercice par exemple— pour l'enseignant), c'est-à-dire le déroulement de l'activité.
Le **sujet*** est l'utilisateur qui réalise l'activité conçue par le prescripteur. C'est soit l'apprenant, soit l'enseignant (pour les prototypes qui s'adressent en particulier à eux, comme par exemple *Calques 3D* [Van Labeke 99]).

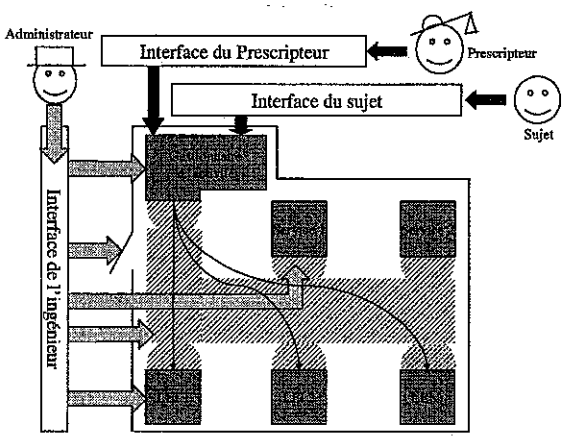


Figure 1.4 – Action des utilisateurs sur l'atelier

Les sections suivantes illustrent les rôles des ces acteurs ainsi que ce qu'ils voient. Nous commençons par montrer ce que voit le sujet. Nous montrons ensuite la préparation que cela suppose du prescripteur. Puis nous montrons la préparation que cela suppose de l'administrateur.

Le sujet

Pour le sujet, le fonctionnement de l'atelier est totalement transparent. Via son interaction avec une interface graphique, le sujet agit sur le gestionnaire d'activités (*cf.* figure 1.4). Ce dernier distribue ses ordres pour que le sujet ait l'impression qu'il agit directement sur les prototypes auxquels il a accès.
L'interface du sujet lui permet de recevoir les consignes qu'il doit suivre pendant l'activité (figure 1.4). Elle lui fournit aussi l'accès aux outils que le prescripteur met à sa disposition.

> **Exemple 1.1** *À un instant donné, le sujet voit apparaître l'écran de la figure 1.5. Une fenêtre en haut et à gauche présente la consigne : le sujet doit dessiner une figure géométrique correspondant à un énoncé. Une autre fenêtre à droite, dans laquelle un prototype fournissant la fonctionnalité d'édition de figure géométrique, s'ouvre sur son écran. Le sujet peut ainsi construire sa figure.*
> *Lorsqu'il estime avoir fini sa construction, il le signale en cliquant sur un bouton «construction finie» dans la fenêtre «transition», en bas à gauche.*
> *Cette partie de l'activité est alors terminée. Un autre écran comportant ces trois fenêtres (consigne, outil et transition) est affiché pour la partie suivante de l'activité.*

L'exemple 1.1 détaille une partie de l'activité du sujet. Nous appelons une telle partie de l'activité une étape*[11].

Étape Une activité est composée de parties successives appelées étapes. Une étape est décrite par une consigne. L'exécution de la consigne est réalisée avec un outil. Une transition signale la fin de l'étape. Au cours d'une étape un outil produit un résultat.

Outil Un outil est un composant constitué à partir d'une fonctionnalité d'un prototype que l'utilisateur peut exécuter. L'outil utilise ou ajoute au prototype les propriétés nécessaires à son utilisation dans l'atelier.

[11] Étape, consigne, outil et transition sont formalisés à la section 3.3

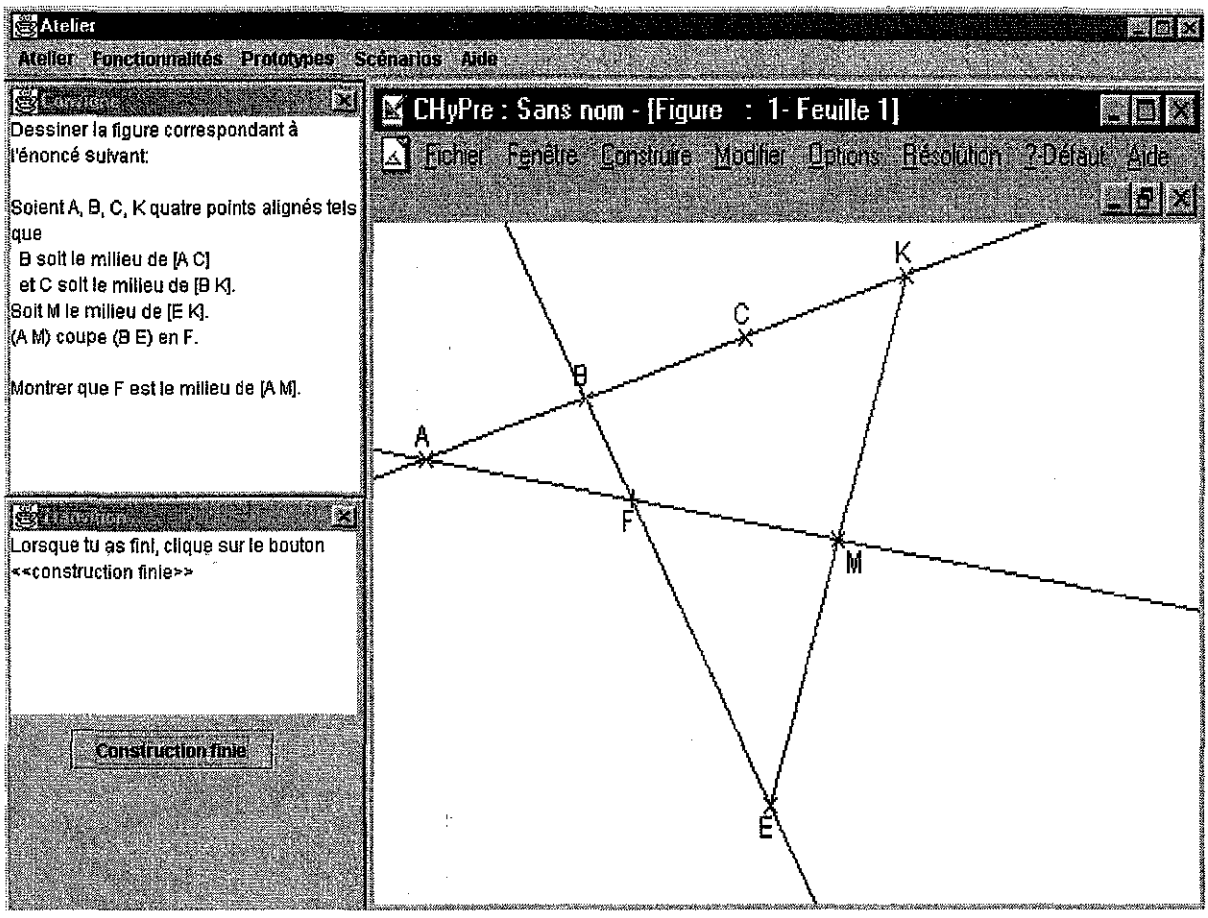


Figure 1.5 – Copie d'écran pour le sujet

Le prescripteur

Pour le prescripteur, le fonctionnement technique de l'atelier est transparent. Il définit le déroulement de l'activité sans modifier la configuration de l'atelier, ceci via une interface (figure 1.4). Dans chaque étape de l'activité, le prescripteur définit

- une **consigne** que le sujet devra suivre ;
- un **outil** à utiliser (telle fonctionnalité de tel prototype) ainsi que le moment où l'outil doit être fermé (à la fin de l'étape courant, à la fin d'une autre étape ou à la fin de l'activité) ;
- les **données à recueillir** ;
- la **présentation** à l'interface de l'écran du sujet. Comme nous venons de le voir, cet écran comporte trois fenêtres : consigne, outil et transition (*cf.* figure 1.5). Pour chacune des fenêtres, il faut préciser la taille (hauteur, largeur en proportion de l'écran de l'atelier) et la position (horizontale et verticale, par rapport à l'écran de l'atelier) de la fenêtre, ainsi que les propriétés (déformable*, déplaçable* et maximisable*).
- la **transition**, c'est-à-dire la condition de passage à l'étape suivante ;

L'exemple suivant permet de mieux comprendre la définition d'une étape par le prescripteur.

Exemple 1.2 *Le prescripteur définit une activité comportant deux étapes. La première correspond à celle de l'exemple 1.1. C'est une étape d'édition de figure. Il la définit, via le formulaire présenté à la figure 1.6 :*

- *la consigne, où* `ex1.txt` *est un énoncé exprimé en langage naturel dans un fichier ;*
- *l'outil, c'est-à-dire la fonctionnalité «édition de figure géométrique» du prototype «*CHyPre*». Ce dernier continue à tourner jusqu'à la fin de l'activité ;*
- *les données à recueillir, ici le résultat produit par le prototype dans un fichier de nom* `ex1.chp` *;*
- *la présentation, (par exemple) la fenêtre contenant la consigne, est positionnée en haut et sur la gauche de l'écran de l'atelier. Elle occupe* 30% *de la largeur de la fenêtre de l'atelier et* 50% *de la hauteur. Sa taille et sa position est fixe. Elle ne peut pas être maximisée.*
- *la transition, ici l'atelier attend que le sujet clique sur un bouton sur lequel est inscrit "Construction finie".*

Consigne

`Dessiner la figure correspondant à l'énoncé suivant:`
>fichier(ex1.txt)

Outils

Fonctionnalité : édition de figure géométrique .....
Prototype : *CHyPre* .....
Moment de fermeture : fin de l'activité .....

Données à recueillir

Résultat produit : oui, fichier `ex1.chp` .....
Connaissances d'interaction : non .....
Autres (informations scrutables) : non .....

Présentation à l'interface

	position Vert	position Horiz	tailleLarg	tailleHaut	déformable	déplaçable	maximisable
Consigne :	haut	gauche	30%	50%	non	non	non
Outil :	haut	droite	70%	100%	non	non	non
Transition :	bas	gauche	30%	50%	non	non	oui

Transition

Étape suivante : .....
Condition de transition à l'étape suivante : bouton «Construction finie» cliqué .....

Figure 1.6 – Caractérisation d'une étape par le prescripteur

L'exemple 1.2 correspond à la définition par le prescripteur d'une portion d'activité à effectuer avec l'atelier.

L'administrateur

L'administrateur agit sur tout l'atelier. Il intègre les prototypes, assure leur communication et met en place les services nécessaires à la réalisation d'une activité. Sa participation est plus technique. Elle dépend de l'implantation de l'atelier. L'exemple 1.3 donne le travail préparatoire de l'administrateur en rapport avec les exemples 1.1 et 1.2 sans préciser l'implantation de l'atelier.

Exemple 1.3 *L'administrateur prépare les prototypes* CHyPre *et* TALC *pour leur coopération avec l'atelier. Il lance l'atelier. Puis il indexe les prototypes* CHyPre *et* TALC. *Il précise les entrées et sorties de chaque prototype (cf. figure 1.3) :*
- *le prototype* CHyPre *reçoit, en entrée, un énoncé dans un fichier nommé* `ex1.exo` ;
- *il produit en sortie une figure stockée dans un fichier nommé* `ex1.ch` *dont le format est bmp ;*
- *il ne produit pas de connaissances d'interactions*[a] ;
- *il ne produit pas d'autres informations scrutables.*

Ces deux prototypes utilisent le même énoncé exprimé dans deux formats différents. L'administrateur s'assure que l'atelier communique cet énoncé au format adéquat à son destinataire. L'atelier est alors prêt pour une prise en main par le prescripteur.

[a]Pendant l'activité, diverses données peuvent être collectées. Les connaissances d'interaction et les informations scrutables en font partie. Elles sont définies à la section 3.4 p 78.

Entrées-sorties du prototype
Flux en entrée : fichier `ex1.txt` formaté en `ex1.exo` .....
Résultat : fichier `ex1.chp` format bmp .....
Flux de connaissances d'interaction : aucun .....
Autres flux en sortie (informations scrutables) : aucun .....

Figure 1.7 – Caractérisation du prototype *CHyPre* par l'administrateur

L'exemple 1.3 correspond à l'utilisation de l'atelier avec deux prototypes et deux étapes, ceux de l'exemple 1.2.

Dans cette section nous avons présenté la partie du travail de l'administrateur pour préparer l'utilisation de l'atelier par le prescripteur et le sujet.

Synthèse

Cette section permet d'identifier des fonctions et des éléments d'architecture de l'atelier. Nous avons introduit les définitions suivantes : atelier, activité, étape, outil, composant, vitrine.

Ces fonctions concernent les prototypes (les mettre à disposition des utilisateurs, permettre l'utilisation de leurs fonctionnalités, supporter leur interaction), l'activité (la mettre en place, recueillir des données pour permettre l'interprétation de celle-ci) ou la technique (gérer les formats, supporter le partage des ressources et l'échange de données). L'atelier doit posséder une architecture distribuée, des composants (des composants-prototypes, des composants de services et un composant principal), un espace de communication et un équipement qui sert d'intermédiaire entre les différents prototypes impliqués. L'organisation de l'atelier dépend de caractéristiques physiques : nombre, type et localisation des machines, type de réseau, ... Les prototypes nécessitent une adaptation pour être utilisés avec l'atelier. Cette section précise les actions des trois

types d'utilisateurs : administrateur, prescripteur et sujet (section 1.1.4). L'atelier présente une interface graphique par utilisateur.

1.2 Étude des objectifs

Le but de cette section est de permettre de cerner ce dont nous avons besoin. Nous avons vu dans l'exemple introductif (page 5) que nous voulons utiliser plusieurs logiciels. Sans l'existence d'une plate-forme pour permettre cette coopération de logiciels, l'utilisateur est confronté à plusieurs difficultés. Nous allons les passer en revue dans cette section. Résoudre chaque difficulté correspond à un ou plusieurs objectifs de recherche et à une ou plusieurs fonctions de l'atelier. Nous regroupons ces objectifs en trois grands «axes» suivant qu'il s'agit d'indexer les ressources, de les utiliser conjointement ou de les gérer.

1.2.1 Objectif : indexer

Pour indexer les prototypes de l'atelier, une base d'information est indispensable. En effet, il est nécessaire de connaître les fonctionnalités intéressantes de chaque logiciel, c'est-à-dire celles qui peuvent être utilisées dans une activité.

Exemple 1.4 *Dans l'exemple introductif (page 5), cinq prototypes sont à indexer :* CABRI-Géomètre, CHyPre, Mentoniezh, PACT *et* TALC. *La figure 1.8 illustre l'indexation de ces cinq prototypes vis-à-vis des fonctionnalités qu'ils implantent.*

Sur la figure 1.8, un élément d'un prototype est soit une base de données (un cylindre) soit un traitement (un rectangle). Une connexion entre une flèche (vers un élément) et une ligne horizontale (vers le nom d'un prototype), indique l'implantation de l'élément dans le prototype. La connexion est foncée si elle est choisie dans l'exemple, elle est claire dans le cas inverse.
Dans cette figure, nous reprenons le regroupement en cinq groupes présenté à la section 2.2. Ici seuls les quatre premiers groupes sont représentés : nous regroupons les fonctionnalités suivant (a) qu'elles relèvent du domaine d'apprentissage, (b) qu'elles constituent une aide ou une fonctionnalité pédagogique, (c) qu'elles fournissent des outils ou des micromondes ou (d) qu'elles relèvent de la dynamique de l'interaction de l'utilisateur avec le prototype.

◇ La *conséquence pour l'atelier* est qu'il doit **disposer d'une base d'informations pour indexer les prototypes et les fonctionnalités disponibles.**

◇ La *conséquence pour chaque prototype* est qu'il devrait **permettre d'accéder indépendamment à ses différentes fonctionnalités.**

◇ La *recherche nécessaire sur cet objectif* concerne la normalisation de la description des prototypes. En effet, comment décrire une ressource pédagogique, en particulier en ce qui concerne les prototypes ? La section 2.4 nous permet de chercher les réponses à cette question dans l'existant.

1.2.2 Objectif : utiliser conjointement

Communication des données

Actuellement, si les utilisateurs réalisaient un exercice de géométrie du début à la fin (de la lecture de l'énoncé à la rédaction d'une réponse) avec les différents prototypes existants, ils se trouveraient dans la nécessité de saisir des données plusieurs fois, sous des formats parfois différents.

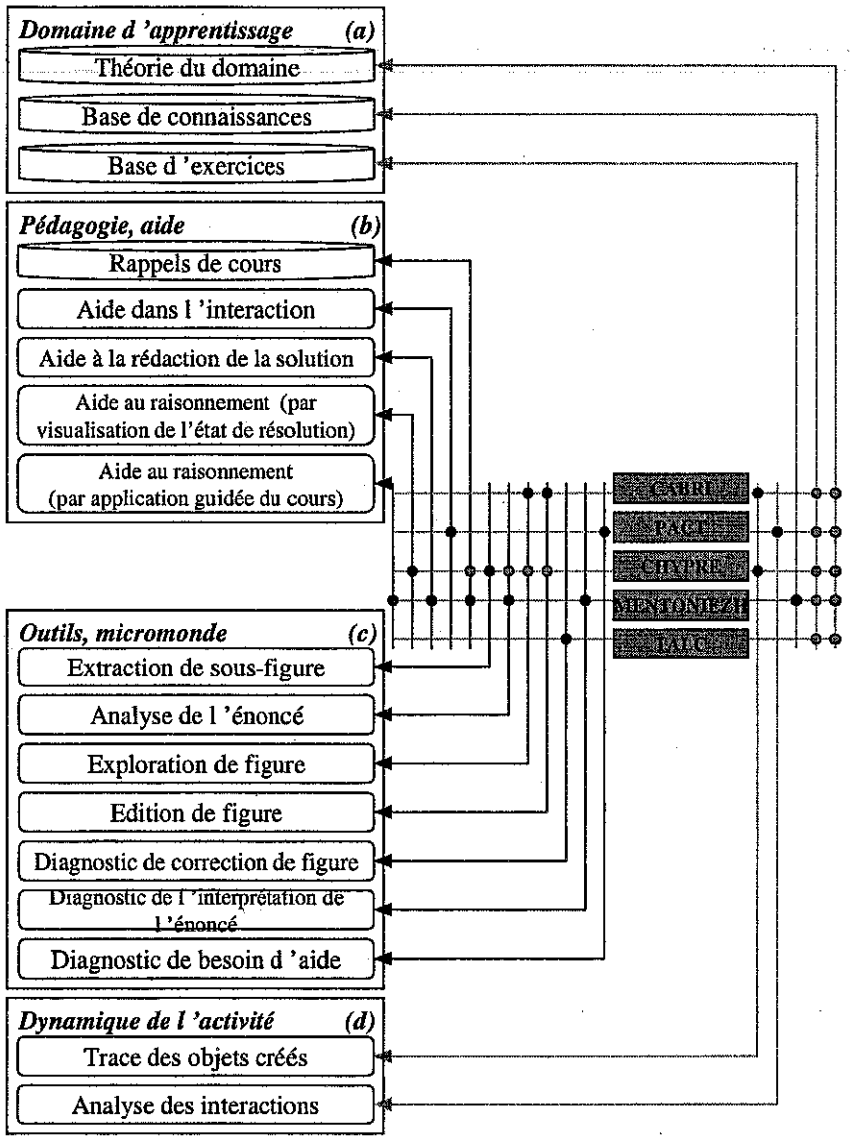


Figure 1.8 – Fonctionnalités des EIAO de l'exemple introductif

Exemple 1.5 *Dans l'exemple introductif (page 5), les prototypes sont utilisés conjointement pour résoudre un problème. Nous disposons initialement d'un énoncé du problème en langage naturel (par exemple, le français).*

Dans l'exemple 1.5, chaque utilisateur interprète cet énoncé afin de pouvoir fournir les informations adéquates aux prototypes dans une forme appropriée (*cf.* figure 1.9). L'**apprenant** (voir les flèches fines sur la figure 1.9), extrait de l'énoncé les faits à partir desquels il peut raisonner et la conjecture qu'il doit prouver. Il utilise ensuite les faits identifiés pour construire la figure avec *CABRI-Géomètre*.

Pour cela, il applique les **outils d'édition** de figure proposé par *CABRI-Géomètre*. Il exploite la conjecture et une nouvelle fois les faits, lorsque *Mentoniezh* demande d'identifier les faits et la conjecture présents dans l'énoncé. Il utilise pour cela les **menus** que lui propose *Mentoniezh*. Il effectue la même tâche (entrée des faits et de la conjecture) avec *CHyPre* grâce à des **menus**. De son côté, l'**enseignant** (les flèches épaisses sur la figure 1.9) identifie les hypothèses

et la conclusion de l'énoncé. Il **traduit** ces informations dans le langage de représentation des connaissances de géométrie que requiert *Mentoniezh*. Il **traduit** une nouvelle fois ces informations dans le langage de représentation des connaissances de géométrie que nécessite *TALC*. Il entre une fois encore ces informations dans *PACT*(par le biais de **menus**). De plus, différents logiciels communiquent les informations (faits, conjectures) directement à d'autres logiciels (voir les flèches dont le trait est constitué de tirets sur la figure 1.9.

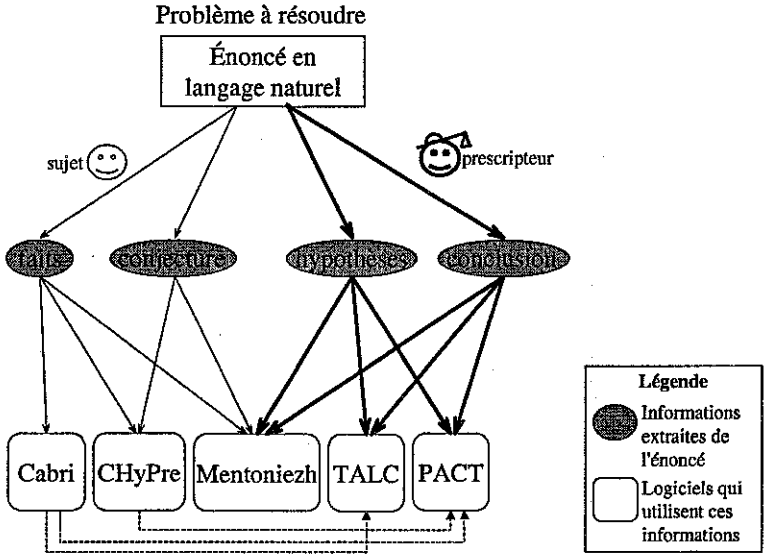


Figure 1.9 – Traitement d'un énoncé par différents prototypes

L'utilisation conjointe des cinq prototypes de notre exemple requiert de chaque utilisateur de multiples saisies sous plusieurs formes des même informations, ainsi qu'une communication inter logiciels de certaines des informations.

Le premier intérêt de l'atelier est de mettre au point un système de communication de données et de gestion des formats afin de supprimer ces saisies multiples.

◇ Dès lors, la *conséquence pour l'atelier* est que celui-ci doit **permettre le transfert de données** à des logiciels divers sans multiplier les saisies. Pour cela, l'atelier doit **assurer la communication entre les prototypes**, d'où la prise en compte d'un média de communication (un équipement) et d'un protocole de transfert des données.

◇ Ainsi, la *conséquence pour chaque prototype*, est qu'il faut **assurer la communication des données au format adéquat** c'est-à-dire que les données soient représentées dans un format compris par tous les logiciels. Deux possibilités sont envisageables :

1. soit il existe un format standard de représentation des données ;
2. soit il existe un moyen de traduire les connaissances à échanger dans le langage de représentation de chaque prototype cible de ces données.

Ces deux possibilités ne sont pas bien évidemment exclusives.

◇ La *recherche nécessaire sur cet objectif* concerne la gestion des formats de données entre différents prototypes. Elle est présenté à la section 3.6.

Monitoring

Le transfert de données est le premier maillon de la solution. La notion même de communication implique sur les prototypes des actions, des inter-actions. En effet, il est parfois nécessaire

de prendre le contrôle de certains prototypes afin de mettre en évidence un objet ou bien de lancer une action.

Exemple 1.6 *Dans l'exemple introductif (page 5),* PACT *agit sur* CABRI-Géomètre *et* CHyPre*(les micromondes d'édition de figures). Pour illustrer notre propos nous prenons le cas de* CABRI-Géomètre.
PACT *a pour but d'aider l'apprenant à partir de l'analyse de ses interactions avec* CABRI-Géomètre. *Dans* CABRI-Géomètre, *l'apprenant est libre d'agir. Il peut ainsi ne jamais atteindre le but qu'il s'est fixé. Lorsque* PACT *détecte que l'apprenant est en difficulté, il peut afficher un message à celui-ci. Toutefois afficher un simple message est parfois insuffisant : par exemple, il est parfois souhaitable de rendre saillants certains objets. Pour cela,* PACT, *met, par exemple, en évidence une propriété de la figure en épaississant les segments d'une sous figure. Ainsi* PACT *doit pouvoir prendre la main sur* CABRI-Géomètre *pour provoquer l'épaississement de certains objets géométriques.*

◇ La *conséquence pour l'atelier* est que l'atelier doit **commander les prototypes** pour agir sur eux : les activer, les désactiver, par exemple.

◇ La *conséquence pour chaque prototype* est qu'il devrait **permettre une certaine forme de prise de contrôle** par un autre prototype. Ritter et Koedinger parlent dans ce cas, de prototype «**scriptable** » ([Ritter 96]).

◇ La *recherche nécessaire sur cet objectif* concerne la définition d'un ensemble de commandes pour agir sur tous les prototypes. Elle est présentée à la section 3.5. Le but est de pouvoir scripter tous les prototypes avec le même langage de scripts.

Protocole de coopération des prototypes

Finalement, lorsque l'on rend possible pour les prototypes des inter-actions, il apparaît indispensable d'ajouter encore une notion de gestion des inter-actions : ordre d'apparition des prototypes, condition de leur clôture, ...

Exemple 1.7 *Dans l'exemple introductif (page 5),* CHyPre *et* Mentoniezh *sont actifs en même temps. De même,* CABRI-Géomètre *et* TALC *sont actifs en même temps, pour que* TALC *puisse récupérer les objets créés avec* CABRI-Géomètre *et ainsi vérifier la conformité de la figure relativement à l'énoncé. Cependant le diagnostic de* TALC *ne doit être lancé que lorsque l'utilisateur a construit une figure avec* CABRI-Géomètre, *avec l'aide de* PACT, *et qu'il demande une validation.*

◇ La *conséquence pour l'atelier* est qu'il doit **posséder un moyen de définir un protocole de coopération entre les prototypes**. Ce protocole de coopération consiste à définir au cours du temps le statut (actifs, inactifs) des participants (les prototypes) et les règles de participation (l'un après l'autre, en parallèle, *etc.*). L'atelier doit aussi être capable de **scruter certains états ou certaines variables d'un prototype** afin de les utiliser dans le protocole de coopération.

◇ La *conséquence pour chaque prototype* est qu'il devrait pouvoir **être rendu actif ou inactif, totalement ou partiellement**. Nous retrouvons ici la propriété de scriptabilité. Le prototype doit aussi **permettre l'observation de certains de ses états ou de ses variables**. Ritter et Koedinger parlent, dans ce cas, d'états ou de variables «**scrutables**» ([Ritter 96]).

◇ La *recherche nécessaire sur cet objectif* concerne deux points :

- la définition de scrutables commun à tous les prototypes. Cet aspect n'est pas approfondi ici.
- la définition d'une protocole de coopération entre les prototypes.

Le dernier point est présenté à la section 3.3.

1.2.3 Objectif : gérer l'activité

Interface

Tous les menus et toutes les fenêtres des prototypes ne sont pas pertinentes à chaque instant. De plus, dans le cas d'un affichage exhaustif de toutes les fenêtres et tous les menus, la charge cognitive dédiée à la gestion de l'interface serait trop lourde. Nous avons donc besoin de savoir comment gérer les informations présentées à l'utilisateur à travers différentes interfaces graphiques (barre de menus, fenêtres, *etc.*) produites par les différents logiciels.

Ainsi, pour diminuer la charge cognitive de l'utilisateur, nous proposons de sélectionner et de faire cohabiter diverses fenêtres de présentation.

Exemple 1.8 *Dans notre exemple les interfaces graphiques de* CABRI-Géomètre, CHyPre, Mentoniezh, PACT *et* TALC *doivent cohabiter.*

◇ La *conséquence pour l'atelier* est qu'il doit **sélectionner ce qui doit être présenté** et **organiser l'écran**.

◇ La *conséquence pour chaque prototype* est qu'il devrait **pouvoir exporter son interface graphique** c'est-à-dire que son interface graphique doit être paramétrable afin de permettre à l'atelier de définir l'endroit où elle sera placé, son statut (actif, inactif), son état (visible, invisible), *etc.*.

◇ La *recherche nécessaire sur cet objectif* concerne la gestion des interfaces graphiques. La recherche sur les interfaces est un domaine en soi, nous n'y avons pas touché. Nous avons adopté une solution *ad hoc* afin de permettre le test des diverses propositions que nous avons faites. Notre contribution ici consiste à définir précisément à la section 4.7 la spécification du gestionnaire d'interface graphique dont nous avons besoin.

Recueillir des données

Pour l'analyse de l'activité après celle-ci, l'atelier doit pouvoir reprendre ou rejouer une séquence d'interaction de l'utilisateur avec l'atelier ou l'un des prototypes qui participe à l'activité. L'atelier doit donc disposer de fonctions propres.

Exemple 1.9 *Dans l'exemple introductif (page 5), nous imaginons que le prescripteur qui a choisi d'utiliser conjointement les cinq EIAO, a pour but d'étudier en quoi l'extraction de sous-figures permet à l'apprenant de mieux résoudre l'exercice. Pour cela, il recueille des données dans diverses conditions expérimentales. Les données dont il a besoin sont principalement liées à l'interaction de l'apprenant avec la fonctionnalité d'extraction de sous-figures offerte par* CHyPre. *Cependant, la sauvegarde de toutes les traces de l'interaction* CHyPre-*apprenant n'est pas pertinente (certaines actions de l'apprenant sur* CHyPre *n'étant pas liées à la fonction de* CHyPre *qui intéresse le prescripteur). De plus, la granularité des informations sauvegardées doit être suffisante, pour que le prescripteur puisse saisir la sémantique de l'action qu'entreprend l'apprenant (sans toutefois être trop fine, car dans ce cas, les informations recueillies risquent d'être inexploitables). Par exemple, il est inutile de mémoriser que l'apprenant clique à tel endroit de l'écran, mais il est utile de savoir que l'apprenant clique sur le bouton qui permet la création d'un nouveau calque (extraction d'une sous figure, visualisée dans une nouvelle fenêtre graphique).*

◇ La *conséquence pour l'atelier* est qu'il doit permettre de **sauvegarder les traces (ou l'historique) de l'interaction** de l'usager avec l'atelier et les divers prototypes et de **choisir les traces à sauvegarder** parmi celles que proposent les différents prototypes.

◇ La *conséquence pour chaque prototype* est qu'il devrait **permettre de sélectionner les interactions utiles** à l'atelier. Pour cela il doit définir la sémantique des interactions de l'utilisateur avec le prototype. Cela permet à la fois de diminuer le nombre de traces à sauvegarder et d'obtenir des traces plus facilement exploitables.

◇ La *recherche nécessaire sur cet objectif* concerne la définition des traces pertinentes à sauvegarder dans le but de collecter des données exploitables sur l'activité. Cet aspect est présenté à la section 3.4.

1.2.4 Synthèse : propriétés de l'atelier et des prototypes

Nous synthétisons les besoins mis en évidence en deux groupes. Nous avons d'une part les propriétés des prototypes pour qu'ils soient utilisés facilement avec d'autres prototypes dans l'atelier. Ils permettent d'ébaucher les recommandations de construction de composants. Nous avons d'autre part les besoins fonctionnels de l'atelier. Ils permettent d'ébaucher un premier cahier des charges macroscopique pour ce dernier.
Nous présentons ces deux groupes dans les sections qui suivent.

Propriétés d'un prototype communicant et coopérant

Nous avons mis en évidence sept propriétés d'un prototype interopérable. Trois de ces fonctionnalités ont été proposés par Ritter et Koedinger [Ritter 96]. Nous avons montré leur intérêt pour l'atelier. Nous les rappelons ci-dessous :

1. scrutable (c'est-à-dire permettre qu'on observe certains de ses états ou de ses variables) ;
2. traçable (c'est-à-dire fournir les traces intelligibles de l'interaction de l'utilisateur avec lui) ;
3. et scriptable (c'est-à-dire permettre une certaine forme de prise de contrôle, dont d'une part être rendu actif ou inactif, totalement ou partiellement et d'autre part permettre de collecter uniquement les interactions jugées utiles).

À ces dernières nous ajoutons :

4. indexable (c'est-à-dire pouvoir être décrit afin d'être retrouvé et d'accéder à ses fonctionnalités) ;
5. interface-exportable (c'est-à-dire pouvoir exporter son interface graphique) ;
6. fonctionnalités-indépendant (c'est-à-dire permettre d'accéder indépendamment à ses différentes fonctionnalités) ;
7. et formats-normalisé (c'est-à-dire assurer la communication au format adéquat des données).

Cette liste de propriétés permet à un prototype de participer pleinement et activement à l'atelier. Elle constitue une liste de recommandations («guidelines») pour favoriser la communication et la coopération inter-logicielle et cela dès la conception des produits.
En ce qui concerne les prototypes existants, il faut se demander ce qu'il faut leur ajouter pour les doter de ces propriétés.

Ébauche d'un cahier des charges - Propriétés d'une plate-forme

Nous avons mis en évidence quelques fonctions ou propriétés de l'atelier. Il doit :

- disposer d'une base de fonctionnalités disponibles pour indexer les prototypes ;
- permettre le transfert de données ;

- assurer la communication des prototypes ;
- commander les prototypes ;
- gérer les différentes interfaces graphiques ;
- posséder un moyen de définir un protocole de coopération entre les prototypes ;
- pouvoir scruter certains états ou certaines variables d'un prototype ;
- sélectionner ce qui doit être présenté ;
- organiser l'écran ;
- sauvegarder des traces d'interaction ;
- permettre le choix des traces à sauvegarder.

Cet inventaire comporte des fonctions et propriétés qui ne sont pas au même niveau. Nous les reprenons et organisons ces fonctions et propriétés dans le chapitre 3, lorsque nous présentons la proposition que nous faisons.
Par ailleurs, remarquons que l'atelier peut aussi posséder les propriétés définies pour les prototypes à la page 23.
Nous présentons l'implantation de ces propriétés au chapitre 4.

1.3 Formalisation

1.3.1 Notations

Il n'y a pas de symbole mathématique pour signifier l'implantation. Nous choisissons de noter que la fonctionnalité F_i est implantée dans le prototype P_j :

$$\mathbf{F_i \prec P_j}$$

Inversement, nous choisissons de noter que le prototype P_j implante la fonctionnalité F_i :

$$\mathbf{P_j \succ F_i}$$

1.3.2 Fonctionnalité

Fonctionnalité

Une fonctionnalité F_i transforme des entrées in_i en sorties out_i :

$$\mathbf{F_i : in_i \longmapsto out_i}$$

Fonctionnalités dans l'atelier

Pour un domaine d'application donné D, l'atelier propose un nombre N_{fonc} fonctionnalités F_i :

$$\exists N_{fonc} \in \mathbb{N} \text{ tel que } \forall i \in \{1, \ldots, N_{fonc}\} \quad \mathbf{F_i}$$

Exemple : en géométrie, nous avons identifié 21 fonctionnalités que l'atelier peut proposer.
D'où $N_{fonc} = 21$ avec $21 \in \mathbb{N}$
Les fonctionnalités sont appelées F_i où $i \in \{1, \ldots, 21\}$.

1.3.3 Prototype

Les fonctionnalités sont implantées dans un nombre N_{proto} de prototypes P_i :

$\exists N_{proto} \in \mathbb{N}$ tel que $\forall i \in \{1, \ldots, N_{proto}\}$.
$\mathbf{P_i}$

Exemple, nous considérons ici 9 prototypes.
D'où $N_{proto} = 9$ où $9 \in \mathbb{N}$
Les prototypes sont appelés P_i où $i \in \{1, \ldots, 9\}$.

1.3.4 Fonctionnalité et Prototype

Une fonctionnalité est implantée dans au moins un prototype

Le nombre de prototypes implantant la fonctionnalité F_i est noté N_{F_i}. Une fonctionnalité est implantée dans au moins un prototype :

$\forall i \in \{1, \ldots, N_{fonc}\}, \exists N_{F_i} \in \mathbb{N}\ N_{F_i} < N_{proto}, \forall j \in \{m_1, \ldots, m_{N_{F_i}}\}$ tel que
$\mathbf{F_i} \prec \mathbf{P_{m_j}}$
avec $m_j \in \{1, \ldots, N_{proto}\}$

Exemple :
La fonctionnalité F_2 est implantée dans les 3 prototypes P_3, P_5 et P_7.
Ici $i = 2$, $N_{F_2} = 3$ et $j \in \{m_1, m_2, m_3\}$
D'où $F_2 \prec P_{m_1}$, $F_2 \prec P_{m_2}$, $F_2 \prec P_{m_3}$
avec $m_1 = 3$, $3 \in \{1, \ldots, 9\}$ ($N_{proto} = 9$),
$m_2 = 5$, $5 \in \{1, \ldots, 9\}$,
et $m_3 = 7$, $7 \in \{1, \ldots, 9\}$.
D'où $F_2 \prec P_3$, $F_2 \prec P_5$, $F_2 \prec P_7$
c'est-à-dire $F_2 \prec (P_3, P_5, P_7)$.

Un prototype implante des fonctionnalités

Un prototype implante au moins une fonctionnalité. Le nombre de fonctionnalités implantées par le prototype P_i est noté N_{P_i} :

$\forall i \in \{1, \ldots, N_{proto}\}, \exists N_{P_i} \in \mathbb{N}\ N_{P_i} < N_{fonc}, \forall j \in \{m_1, \ldots, m_{N_{F_i}}\}$ tel que
$\mathbf{P_i} \succ \mathbf{F_{m_j}}$
avec $m_j \in \{1, \ldots, N_{fonc}\}$

Exemple : Le prototype P_9 implante les 3 fonctionnalités F_1, F_2 et F_4.
Ici $i = 9$, $N_{P_9} = 3$ et $j \in \{m_1, m_2, m_3\}$
D'où $P_9 \succ F_{m_1}$, $P_9 \succ F_{m_2}$, $P_9 \succ F_{m_3}$
avec $m_1 = 1$, $1 \in \{1, \ldots, 21\}$ ($N_{fonc} = 21$),
$m_2 = 2$, $2 \in \{1, \ldots, 21\}$,
et $m_3 = 4$, $4 \in \{1, \ldots, 21\}$.
D'où $P_9 \succ F_1$, $P_9 \succ F_2$ et $P_9 \succ F_4$
c'est-à-dire $P_9 \succ (F_1, F_2, F_4)$.

Enchaînement des fonctionnalités dans un prototype

Dans notre contexte, un prototype prêt à fonctionner dans l'atelier enchaîne des fonctionnalités indépendantes, c'est-à-dire qu'une fonctionnalité succède à une autre.

Soit un prototype P_i enchaînant N_{P_i} fonctionnalités. Nous notons g_1 la première, g_j la $j$ième et $g_{N_{P_i}}$ la N_{P_i}ième.
On a alors : $P_i \succ (g_1, \ldots, g_j, \ldots g_{N_{P_i}})$.
Dans un enchaînement, toutes les entrées d'une fonctionnalité sont les sorties de la précédente, c'est-à-dire :
$\forall j \in \{2, \ldots, N_{P_i}\}$ on a $in_j \subset out_{j-1}$ où in_j désigne les entrées de la fonctionnalité g_j et out_{j-1} désigne les sorties de la fonctionnalité (précédente) g_{j-1}.
On a alors :

$P_i \succ (g_1, g_2, \ldots, g_{N_{P_i}})$
$\forall j \in \{2, \ldots, N_{P_i}\}$ et $g_j : in_j \longmapsto out_j$, on a $in_j \subset out_{j-1}$
$P_i : in_1 \xrightarrow{g_1} out_1 \xrightarrow{g_2} \ldots \xrightarrow{g_{N_{P_i}}} out_{N_{P_i}}$
c'est-à-dire
$P_i = g_{N_{P_i}} \circ \cdots \circ g_2 \circ g_1$

Exemple : Le prototype P_9 implante les 3 ($N_{P_9} = 3$) fonctionnalités F_1, F_2 et F_4 :
$P_9 \succ (F_1, F_2, F_4)$
Au lancement du prototype, la fonctionnalité active est F_2 ($g_1 = F_2$). Elle est suivie par F_1 ($g_2 = F_1$) puis par F_4 ($g_{N_{P_9}} = g_3 = F_4$).
$F_2 : in_2 \longmapsto out_2$ puis $F_1 : in_1 \longmapsto out_1$ puis $F_4 : in_4 \longmapsto out_4$
Ici $in_1 = out_2$ (donc $in_1 \subset out_2$) et $in_4 = out_1$ (donc $in_4 \subset out_1$) alors $P_9 : in_2 \xrightarrow{F_2} out_2 \xrightarrow{F_1} out_1 \xrightarrow{F_4} out_4$
c'est-à-dire $P_9 = g_3 \circ g_2 \circ g_1 = F_4 \circ F_1 \circ F_2$

Accès à une fonctionnalité d'un prototype

Dans le cas d'un prototype P_i, nous notons l'accès à la $j$ième fonctionnalité (appelée g_j) :

$\mathbf{P_i.g_j}$

Exemple : dans l'exemple précédent, l'accès à la deuxième fonctionnalité ($g_2 = F_1$) du prototype P_9 est noté $P_9.g_2$ ou encore $P_9.F_1$.

1.3.5 Outil

L'outil O_i est défini par l'accès à une fonctionnalité F_j d'un prototype P_k. Un outil est une fonction qui transforme des entrées in_i en sortie out_i :

$\mathbf{O_i = P_k.F_j}$
$\mathbf{O_i : in_i \longmapsto out_i}$
or $P_k.F_j : in_j \longmapsto out_j$
donc $in_i = in_j$ et $out_i = out_j$.

Exemple : Nous considérons l'outil O_1 défini par l'accès à la fonctionnalité F_1 du prototype P_9. Alors $O_1 = P_9.F_1$.

(F_j et P_k) sont des variables dépendantes. En effet, nous ne pouvons pas choisir n'importe quel j et n'importe quel k pour avoir $P_k.F_j$. De plus, un outil possède les «bonnes propriétés» énoncées ci-dessous.

- traçable : il produit des traces sur l'interaction de l'utilisateur avec lui. Une trace porte la sémantique de l'action de l'utilisateur ;
- scrutable : il permet la consultation de certaines données internes (variables, états) ;
- scriptable : il peut être commandé et paramétré via un script.

1.4 Modélisation des besoins

Dans cette partie, nous déterminons ce qui doit être développé, c'est-à-dire «le quoi». Pour présenter la modélisation, nous employons la notation graphique UML*. Nous rappelons brièvement la signification des éléments de cette notation au fur et à mesure de leur utilisation. Pour une définition complète, vous pouvez consulter le site web UML [UML http].
La méthode de modélisation que nous employons, consiste à nous servir des cas d'utilisations pour spécifier les besoins. Dans cette section, nous présentons :

des acteurs — personnes (ou objets) à l'origine d'une interaction avec le système (l'atelier) ;

des cas d'utilisation — objectifs du système, motivés par un besoin d'un acteur (au moins) ;

des diagrammes de cas d'utilisation — représentations des fonctions du système du point de vue de l'utilisateur. Un diagramme de cas d'utilisation contient des cas d'utilisation. Par abus de «langage», les diagrammes de cas peuvent contenir des diagrammes de cas d'utilisation, c'est-à-dire une collection de cas d'utilisation ;

des scénarios UML[12] — déroulements prévus d'un cas d'utilisation. Chaque déroulement est décrit par un diagramme de séquences ou un diagramme de collaboration ;

des diagrammes de séquences (ou de collaboration) — représentations temporelles (ou spatiales) des objets et de leurs interactions.

Tous ces éléments ne sont pas intégralement décrits dans ce chapitre. Nous décrivons ceux qui permettent de comprendre nos propositions. Des éléments supplémentaires, courants dans les systèmes informatiques, sont fournis en annexe F.

1.4.1 Sélection des acteurs

Cette section présente les acteurs (personnes et objets) retenus pour l'analyse des besoins à l'aide de la technique des cas d'utilisation. Ces acteurs sont utilisés pour la description des interactions acteurs-atelier qui suit. Ils permettent de présenter les fonctions désirées.

Sujet

Cet acteur représente le rôle de l'utilisateur de l'atelier qui exécute une activité.

Prescripteur

Cet acteur représente le rôle de l'utilisateur de l'atelier qui conçoit une activité.

Administrateur

Cet acteur représente le rôle de l'administrateur. Il configure le système.

Usager

Cet acteur représente une généralisation (*cf.* figure 1.10[13]) des trois acteurs précédents : un sujet est un usager, un prescripteur est un usager et un administrateur est un usager.

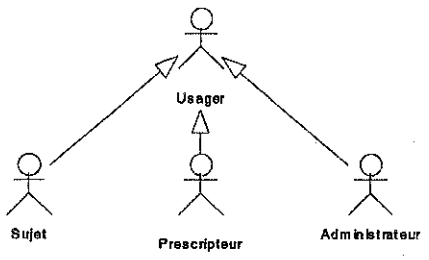


Figure 1.10 – Relations entre les acteurs

Atelier

Cet objet représente le système à développer dans son ensemble. Sa définition a été raffinée depuis l'introduction. Nous avons maintenant une bonne idée du système. Elle sera encore plus précise après cette étape de Modélisation.

Outil

Cet objet représente un outil avec lequel l'atelier ou un usager interagit. Un outil est un prototype particulier. Nous l'avons introduit au chapitre 1. La définition précise d'un outil est formalisée à la section 1.3. Le système et les usagers interagissent avec les outils.

Activité

Cet objet représente l'activité que le prescripteur prépare, à laquelle le sujet participe, et que l'administrateur permet via la paramétrisation de l'atelier. Sa définition a été élaborée à partir du chapitre 1. Elle est formalisée à la section 1.3.

1.4.2 Diagramme principal des cas d'utilisation

Le diagramme principal ne contient que des diagrammes de cas d'utilisation. Ces derniers sont décrits progressivement, dans la suite de ce chapitre ou des suivants. Le diagramme de la figure 1.11 présente les (diagrammes de) cas d'utilisations principaux de l'atelier.

Pour accéder à l'atelier, les usagers (sujet, prescripteur et administrateur) doivent d'abord faire une procédure d'identification. Ceci est décrit par les liens «include» des cas «Définition d'une activité», «Réalisation d'une activité» et «Configuration».

Les cas d'utilisation «Définition d'une activité» et «Réalisation d'une activité» représentent toutes les fonctions en opération normale alors que le cas «Configuration» représente toutes les

[13]La sémantique des flèches UML est rappelé à l'annexe D. Ici il s'agit de l'héritage.

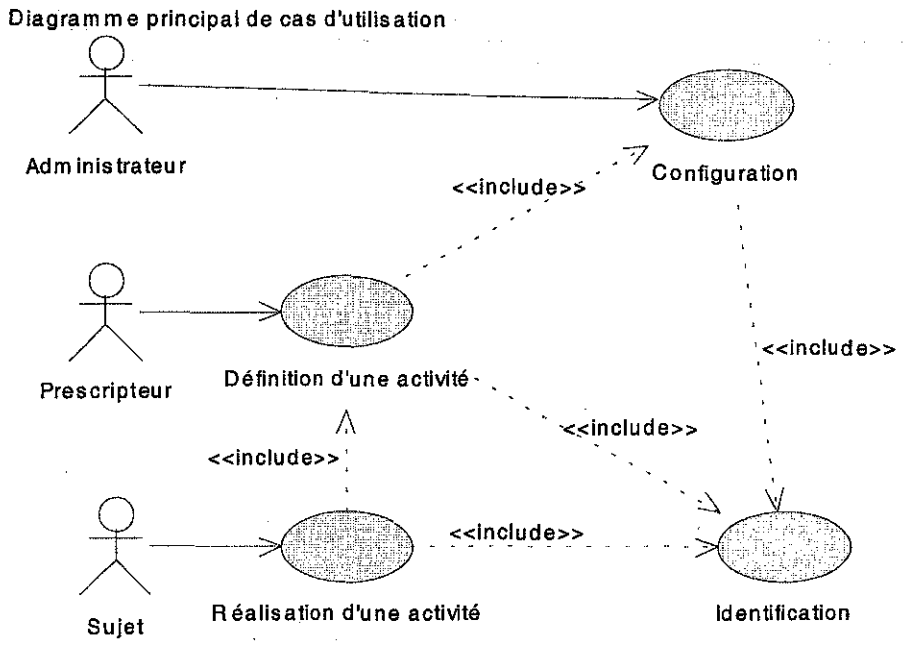


Figure 1.11 – Diagramme principal des cas d'utilisation

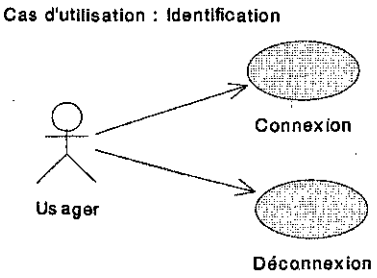
fonctions de gestion de l'atelier. La flèche en pointillé entre le cas d'utilisation «Définition d'une activité» et le cas d'utilisation «Configuration» indique une dépendance. Cette dépendance porte l'étiquette «include» car le cas d'utilisation «Définition d'une activité» utilise le cas d'utilisation «Configuration». De même, le cas d'utilisation «Réalisation d'une activité» dépend (utilise) le cas d'utilisation «Définition d'une activité».

1.4.3 Cas d'utilisation et scénarios UML

Nous détaillons ci-dessous les cas d'utilisation «Identification» et «Configuration». Les cas d'utilisation «Définition d'une activité» et «Réalisation d'une activité» font l'objet de propositions décrites à la section 3.3 p 72.

Identification

Le cas d'utilisation «Identification» recouvre les cas d'utilisation «connexion» et «déconnexion» (*cf.* figure 1.4.3.0). Il sont décrits aux paragraphes suivants.



Cas d'utilisation : Connexion — La table 1.1 décrit les objectifs de l'atelier lors d'une connexion. Trois scénarios sont possibles lors de la connexion. La connexion est acceptée, c'est

Nom :	Connexion
Résumé des responsabilités :	- Le cas d'utilisation «connexion» contient les scénarios décrivant ce qui peut se produire lors de la connexion. L'usager doit fournir son nom et son mot de passe pour se connecter. - L'usager doit exister dans l'atelier, ne pas être connecté, et le mot de passe doit être valide.
Acteurs :	Usager et Atelier.
Pré-conditions	L'usager ne doit pas être connecté
Post-condition	L'usager aura accès à l'atelier s'il a fourni un nom et un mot de passe valide. L'accès sera refusé sinon.
Description des interactions :	– L'usager demande à se connecter. – L'atelier demande les renseignements (nom et mot de passe). – L'usager fournit son nom et son mot de passe. – L'atelier valide les renseignements. – L'atelier accepte la connexion et donne l'accès. Cas variants : – L'usager peut être déjà connecté. L'accès est refusé. – L'usager peut fournir un nom et/ou un mot de passe invalide. L'accès est refusé.
Cas reliés	Les cas «Configuration», «Définition d'une activité» et «Réalisation d'une activité» utilisent le cas «Identification».

Table 1.1 – Description du cas d'utilisation : connexion

le cas normal. La connexion est refusée pour l'une des deux raisons suivantes : soit les renseignements fournis par l'usager sont erronés, soit l'utilisateur est connecté. Nous donnons les diagrammes de séquences pour ces trois scénarios.

Scénario : connexion acceptée Diagramme de séquences (*cf.* figure 1.12) décrivant les interactions de haut niveau pour une connexion lorsqu'elle est acceptée.

Scénario : connexion refusée — cas de renseignements incorrects. Le diagramme de séquences (*cf.* figure 1.13) décrivant les interactions de haut niveau pour une connexion refusée parce que le nom et/ou le mot de passe est incorrect.

Scénario : connexion refusée — cas d'un utilisateur déjà connecté. Diagramme de séquences (*cf.* figure 1.14) décrivant les interactions de haut niveau pour une connexion refusée parce que l'usager est déjà connecté.

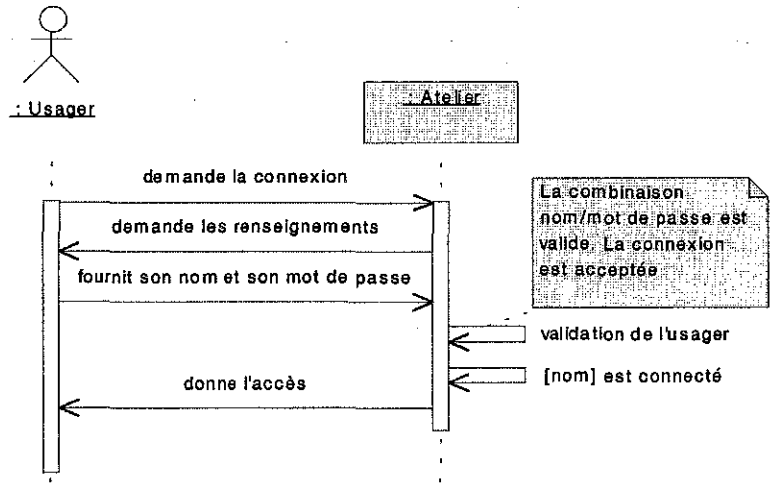


Figure 1.12 – Diagramme de séquences : connexion acceptée

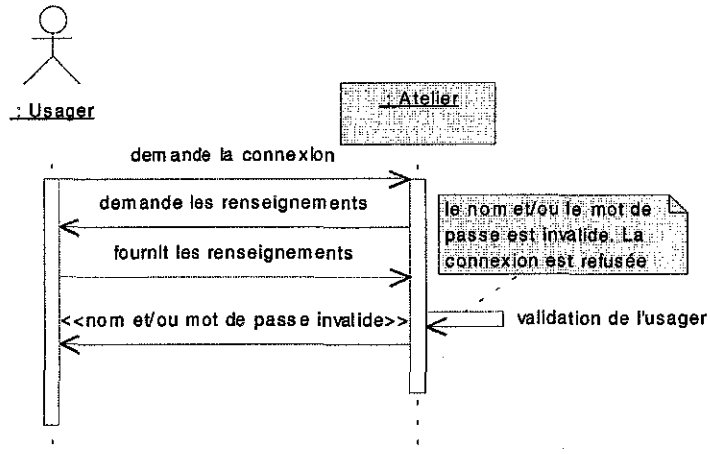


Figure 1.13 – Diagramme de séquences : connexion refusée pour renseignements incorrects

Cas d'utilisation : Déconnexion — La table 1.2 décrit les objectifs de l'atelier lors d'une déconnexion. Le scénario normal de ce cas d'utilisation correspond à l'acceptation de la déconnexion.

Scénario : déconnexion accepté Diagramme de séquences (*cf.* figure 1.15) décrivant en quoi consiste la déconnexion.

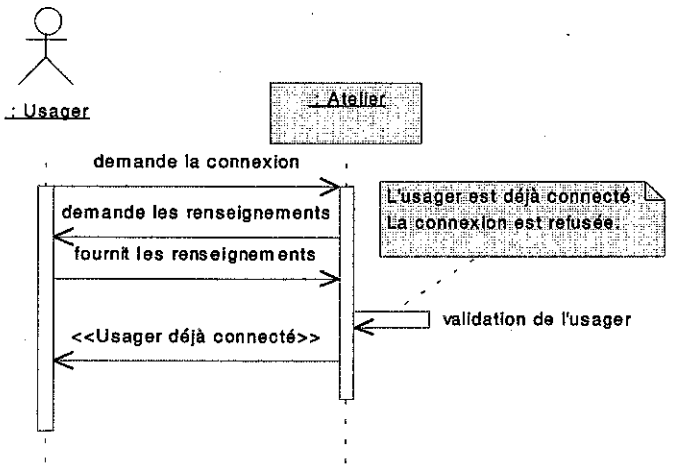


Figure 1.14 – Diagramme de séquences : connexion refusée pour usager déjà connecté

Nom :	Déconnexion
Résumé des responsabilités :	Le cas d'utilisation «Déconnexion» contient les scénarios décrivant ce qui peut se produire lors de la déconnexion.
Acteurs :	Usager et Atelier.
Pré-conditions	L'usager doit être connecté.
Post-condition	L'usager est considéré comme non connecté.
Description des interactions :	– L'usager demande de se déconnecter. – L'atelier déconnecte l'usager.
Cas reliés	Le cas «Connexion».

Table 1.2 – Description du cas d'utilisation : déconnexion

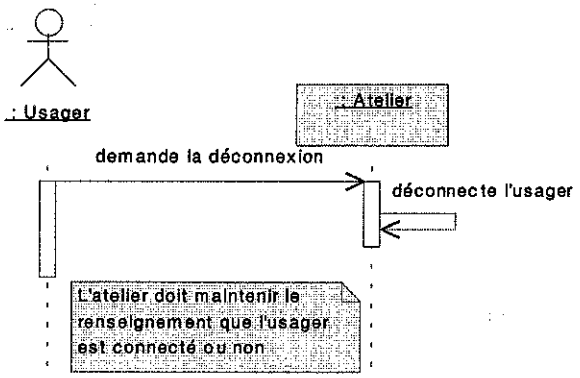


Figure 1.15 – Diagramme de séquences : déconnexion

Configuration

Le cas d'utilisation «Configuration» représente les besoins pour la configuration de l'atelier. Il peut être éclaté en un nouveau diagramme de cas d'utilisation. La figure 1.16 donne le diagramme de cas d'utilisation pour la configuration.
L'atelier implique divers types d'usager. Le cas d'utilisation «Gérer les usagers» comprend toutes

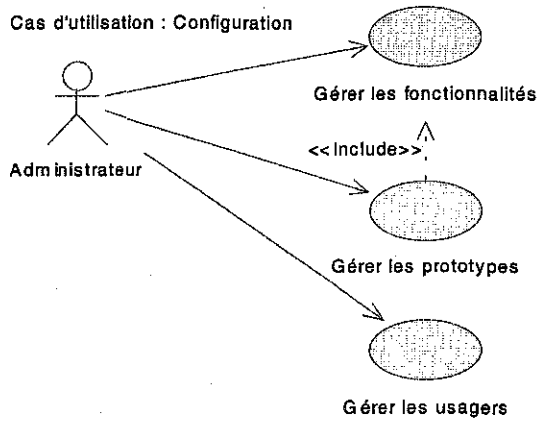


Figure 1.16 – Diagramme de cas d'utilisation : configuration

les fonctions (de l'atelier) nécessaires pour gérer les usagers de l'atelier (*cf.* figure 1.17). Ces cas

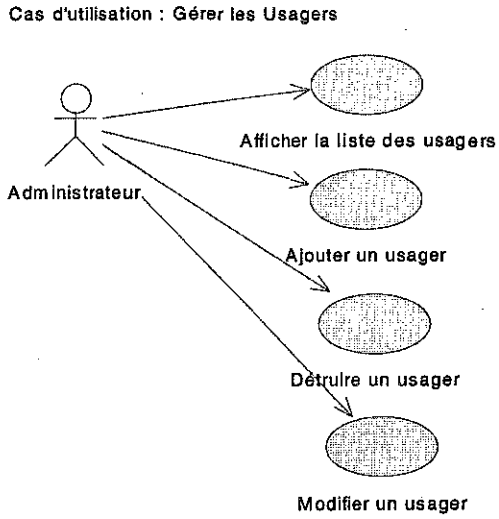


Figure 1.17 – Diagramme de cas d'utilisation : gérer les usagers

d'utilisations sont classiques pour un système comportant différents types d'utilisateurs. Notre recherche ne porte pas sur cet aspect du développement. Par conséquent, nous ne détaillons pas ces cas d'utilisation dans ce chapitre. .

L'atelier implique des prototypes de Logiciels Éducatifs. Le cas d'utilisation «Gérer les prototypes» comprend toutes les fonctions (de l'atelier) nécessaires pour gérer les prototypes. La figure 1.18 donne le diagramme de cas d'utilisation pour gérer les prototypes.

Un prototype implante des fonctionnalités pour un domaine d'application donné (la géométrie par exemple). Le cas d'utilisation «Gérer les fonctionnalités» (du domaine) comprend toutes les

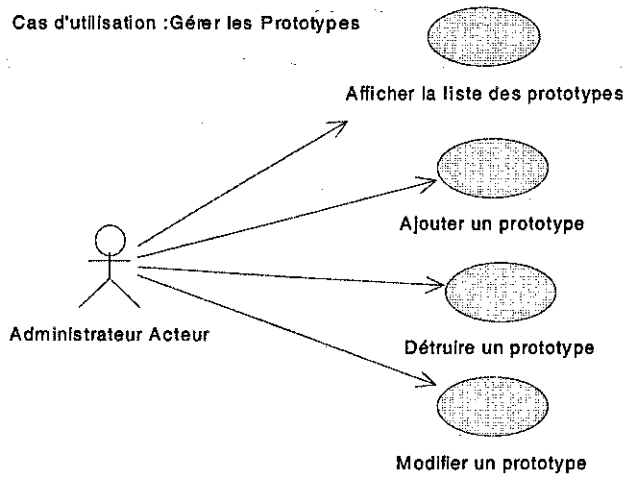


Figure 1.18 – Diagramme de cas d'utilisation : gérer les prototypes

fonctions (de l'atelier) nécessaires pour gérer les fonctionnalités (du domaine) dans l'atelier.

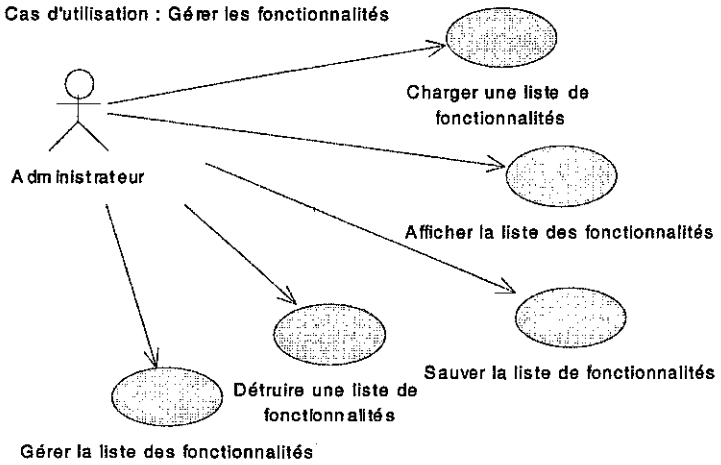


Figure 1.19 – Diagramme de cas d'utilisation : gérer les fonctionnalités

Le cas d'utilisation «Gérer les prototypes» utilise le cas d'utilisation «Gérer les fonctionnalités», d'où le lien de dépendance étiqueté «include» entre ces deux cas. Pour remplir cette fonction, l'atelier maintient une base d'informations pour les fonctionnalités et une autre pour les prototypes. Les fonctions nécessaires pour gérer ces bases d'informations sont présentées dans les sections 2.1 et 2.2 ci-après. Les figures 1.19 p 34 et 1.20 p 35 donnent les diagrammes de cas d'utilisation pour gérer les fonctionnalités.

1.4.4 Synthèse

Liste des cas d'utilisation UML

La table 1.3 récapitule les cas d'utilisations et les scénarios associés, décrits jusqu'à présent. Pour chaque scénario, nous distinguons le cas normal et les cas exceptionnels éventuels.

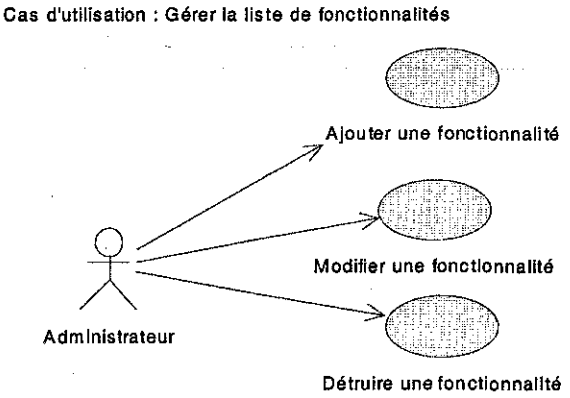


Figure 1.20 – Diagramme de cas d'utilisation : gérer la liste des fonctionnalités

Cas d'utilisation	**Scénarios principaux**
Identification	Connexion
Identification	Déconnexion
Configuration[gérer les usagers]	Afficher la liste des usagers
Configuration[gérer les usagers]	Ajouter un usager
Configuration[gérer les usagers]	Détruire un usager
Configuration[gérer les usagers]	Modifier un usager
Configuration[gérer les fonctionnalités]	Afficher la liste des fonctionnalités
Configuration[gérer les fonctionnalités]	Ajouter une fonctionnalité
Configuration[gérer les fonctionnalités]	Détruire une fonctionnalité
Configuration[gérer les fonctionnalités]	Modifier une fonctionnalité
Configuration[gérer les prototypes]	Afficher la liste des prototypes
Configuration[gérer les prototypes]	Ajouter un prototype
Configuration[gérer les prototypes]	Détruire un prototype
Configuration[gérer les prototypes]	Modifier un prototype
Définition d'une activité	
Réalisation d'une activité	

Table 1.3 – Cas d'utilisation et scénarios associés

Diagramme de classe des objets du domaine du problème

La figure 1.21 présente le diagramme de classe des objets du domaine.
Nous retrouvons ici :

- un `Usager`, avec comme attribut un nom et un mot de passe. Les héritiers (`Sujet`, `Prescripteur` et `Administrateur`)[14] représentent les divers types d'usagers.
- une `Activite`, elle a pour attribut un `Usager` et un `Scenario`. L'`Usager` a une relation d'association $1-1$ avec l'`Activite`. En effet, un usager n'effectue qu'une et une seule activité à la fois dans l'atelier.
- l'`Atelier`, qui a une relation qualifiée avec une `Activite`. En effet, pour un nom donné, il existe une et une seule activité. À partir d'un nom valide, le système peut retrouver la «bonne» `Activite`. Cela permet à l'usager d'interrompre une activité et de la reprendre

[14] À cette étape de la modélisation, nous pourrions nous demander si ces classes sont vraiment nécessaires. Nous reportons cette décision à plus tard.

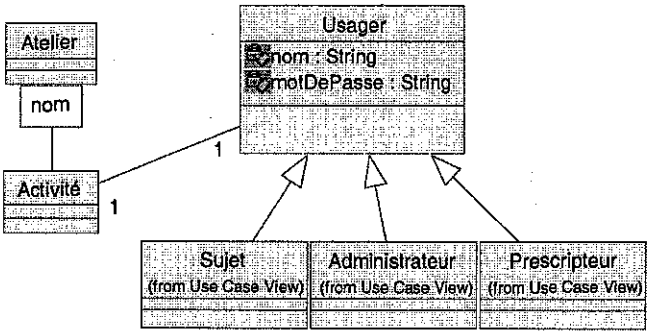


Figure 1.21 – Diagramme de classe des objets du domaine

ultérieurement là où il l'a laissée[15].
Ceci est une présentation simplifiée de modélisation, où nous plaçons les classes connues à partir du domaine du problème. Ces classes sont apparues lors de l'analyse des besoins dans les paragraphes précédents.

Ce chapitre nous a permis de définir l'atelier plus précisément. La présentation à la section 1.2 des difficultés que la définition de l'atelier implique, nous permet d'introduire les problématiques de recherche que nous avons rencontrées. Elle nous permet de situer nos besoins par rapport à l'existant au chapitre suivant, notamment en ce qui concerne les fonctionnalités disponibles en géométrie, les produits de coopération et la normalisation des objets pédagogiques du point de vue de leurs données et de leur architecture.

[15]C'est particulièrement intéressant quand l'activité est pédagogique. Alors qu'il est plus rare d'interrompre une expérimentation (de logiciels) et de la reprendre plus tard.

Chapitre 2

Existant : Utilisation conjointe de logiciels - application en géométrie

2.1 Introduction

Nous souhaitons concevoir un atelier permettant l'utilisation conjointe de prototypes de logiciels éducatifs aux fonctionnalités complémentaires. Ce chapitre a pour but de présenter d'une part les logiciels éducatifs que nous souhaitons utiliser ensemble et d'autre part divers produits existants dont le but est l'utilisation conjointe de logiciels. Ces produits sont dits «produits de coopération» par la suite.
Dans la revue de l'existant, nous présentons tout d'abord des Logiciels Éducatifs de géométrie (section 2.2). Nous cernons ainsi mieux les fonctionnalités à notre disposition dans ce domaine et en particulier les fonctionnalités complémentaires à utiliser ensemble.
Ensuite pour présenter les produits de coopération, nous utilisons une typologie qui s'appuie sur divers paradigmes (section 2.3) : client-serveur, multi-agents, intégrateurs et composants. Chaque paradigme est défini dans la section correspondante.
Enfin, nous présentons (section 2.4) les projets de normalisation pour la description des objets pédagogiques (learning objects) qui se développent depuis quelques années. Ces efforts devraient favoriser l'utilisation conjointe de logiciels.

2.2 Existant : quelques EIAO de géométrie

2.2.1 Définitions

Rappelons que nous avons choisi la dénomination d'EIAO pour désigner notre domaine de recherche et de Logiciels Éducatifs pour les logiciels produits par ce domaine et que, de plus, la dénomination Logiciels Éducatifs regroupe tous les types de logiciels utilisés à des fins éducatives, qu'ils soient intelligents ou non, interactifs ou non. Elle englobe en particulier les logiciels qualifiés classiquement d'EIAO.
Les Logiciels Éducatifs recouvrent divers types de prototypes, avec ou sans capacité de raisonnement ou de résolution de problèmes. Ces capacités, quand elles existent, sont utilisées par le prototype pour fournir une aide à l'apprenant ou pour suivre le raisonnement de ce dernier. Nous distinguons trois types de Logiciels Éducatifs : les micromondes, les tuteurs et les autres outils éducatifs.

Un **micromonde** est un environnement, c'est-à-dire «un monde restreint»

[Dreyfus 81, Minsky 70], qui fournit à l'apprenant des objets, des relations et des outils pour agir sur ce monde (*e.g.* créer et détruire des objets, agir sur eux en les déplaçant, les déformant, *etc.*). Les objets et relations y sont simplifiés. Dans un micromonde l'apprenant est libre d'agir : il n'a aucune consigne à suivre. L'apprenant a toujours l'initiative de ses actions sur le micromonde. Par conséquent, un micromonde est un environnement *ouvert* [Balacheff 94a]. LOGO, conçu par Minsky et Papert [Papert 80], est l'exemple type d'un micromonde : à partir de quelques primitives simples de dessin géométrique dans le plan, l'apprenant construit des procédures de dessin de plus en plus complexes pour répondre à de nouveaux objectifs.

Un **tuteur** est un environnement informatique dans lequel l'apprenant doit exécuter une tâche très précise. Le tuteur s'appuie sur des capacités propres de résolution de problème et/ou de raisonnement pour guider plus ou moins fortement l'apprenant dans sa tâche. L'apprenant n'a donc pas l'initiative de ses actions sur le tuteur.

Beaucoup de Logiciels Éducatifs disposant de capacités de diagnostic n'entrent ni dans la catégorie des micromondes ni dans celle des tuteurs. Nous les rangeons dans la catégorie des autres outils éducatifs. Un **autre outil éducatif** est un environnement informatique qui fournit un support technique à l'apprenant dans la réalisation d'une tâche liée à son apprentissage. L'autre outil éducatif peut disposer de capacités propres de résolution de problème et/ou de raisonnement.

Sur une échelle à peu près continue en fonction de l'initiative laissée à l'apprenant, les autres outils éducatifs se trouvent quelque part entre les tuteurs à une extrémité et les micromondes à l'autre. En effet, l'initiative de l'apprenant peut y être totale, partielle ou absente.

De nombreux travaux de recherche en EIAO présentent l'état de l'art dans ce domaine [Cuppens 90, Ag-Almouloud 92, Ferneda 92, Bazin 93, Bernat 94b, Luengo 97b]. Nous ne présentons pas ici tous ces travaux mais seulement ceux qui permettent de mieux situer les autres, ainsi que ceux dont les fonctionnalités sont intéressantes ou originales dans la perspective d'une réutilisation. Nous avons choisi de présenter les prototypes existants, selon qu'ils sont des micromondes (section 2.2.2), des tuteurs (section 2.2.3) ou d'autres outils éducatifs (section 2.2.4). Notez que les tuteurs et les micromondes sont presque toujours des Logiciels Éducatifs issus de la recherche en EIAO tandis que ce n'est pas le cas de tous les autres outils éducatifs.

2.2.2 Micromondes

Euclide

Euclide [Allard 86] est un langage informatique d'**édition de figure géométrique**. L'environnement informatique défini autour de ce langage est un micromonde.
Euclide, inspiré de et écrit en LOGO, fournit à l'utilisateur un ensemble de primitives pour dessiner. L'apprenant utilise ces primitives en désignant par le biais d'une souris la place géographique des objets à l'écran. L'apprenant peut ainsi éditer une figure correspondant à un énoncé. *Euclide* le conduit à définir ses propres procédures de dessin.
À la différence de LOGO, conçu à partir d'une représentation en coordonnées relatives du plan, *Euclide* ne fait référence à aucun système de coordonnées.

Dans cette section, nous retenons la fonctionnalité **[F_{17}.] édition de figure géométrique**[16] en vue d'une utilisation dans l'atelier.

[16]Nous classons les fonctionnalités par ordre alphabétique et nous les numérotons, *cf.* annexeA

Wimageo

Wimageo [Wimageo http] est un micromonde de construction de figures géométriques planes (**édition de figure géométrique**). Il s'agit de la version pour Windows du programme Imageo (pour DOS). Il fait partie des outils *LILIMATH*[17].
Dans *Wimageo*, écrit avec DELPHI, les figures géométriques sont construites à partir d'un script utilisant un langage simple faisant appel au vocabulaire géométrique usuel. Ce mode de construction est analogue à celui offert par *Euclide*.
Wimageo offre la possibilité de déplacer certains de ses points et d'observer en direct les transformations subies par l'ensemble de la figure. Seuls les points sont déplaçables, on parle de «géométrie du point». Pour déplacer l'un de ces points, il faut amener le curseur de la souris sur sa position puis appuyer sur le bouton gauche pour le sélectionner ; il doit alors changer de couleur. Il suffit ensuite de déplacer le curseur de la souris tout en maintenant le bouton gauche appuyé. Cette possibilité de modifier la figure est nommée «manipulation directe» ([Nanard 90]). La manipulation directe permet, à n'importe quel stade d'une construction, de modifier la position géographique des objets, tout en conservant toutes les propriétés géométriques données durant la construction. Lors de la réalisation traditionnelle de l'exercice (dite papier-crayon), l'apprenant trace une instance de la figure c'est-à-dire un dessin. Ce dernier est statique. Ici, le logiciel permet de tracer une instance de la figure puis de la déformer, donnant ainsi accès à de multiples dessins. C'est à cause de cette possibilité de modifier dynamiquement l'édition, qu'on parle de «géométrie dynamique». En résumé, les figures sont modifiables en conservant toutes leurs propriétés et contraintes. Nous parlons, dans ce cas, d'une fonctionnalité d'**exploration visuelle de figure géométrique.**
Par ailleurs, les figures obtenues constituent des documents imprimables directement ou insérables dans un traitement de texte (format BMP). Il s'agit de la fonctionnalité **exportation de la figure** de *Wimageo*.

Nous retenons les fonctionnalités **[F_{14}.] exploration visuelle de figure géométrique**, **[F_{15}.] exportation de la figure** et **[F_{17}.] édition de figure géométrique** en vue d'une utilisation dans l'atelier.

CABRI-Géomètre, *Chamois* et *Geometer's Sketchpad*

CABRI-Géomètre[18] (acronyme pour **CA**hier de **BR**ouillon Informatique pour le géomètre) [Laborde 86, Laborde 89, Baulac 90, Bellemain 92], est un micromonde de construction de figures géométriques.
CABRI-Géomètre utilise des menus déroulant et la souris pour définir des objets géométriques. Il s'agit de la fonctionnalité d'**édition de figure géométrique.**
Comme *Wimageo*, *CABRI-Géomètre* permet la manipulation directe des figures (fonctionnalité d'**exploration visuelle de figure géométrique**).
De plus, lors de l'interaction de l'apprenant avec *CABRI-Géomètre*, ce dernier sauvegarde une trace des objets créés sous forme d'un historique. Il s'agit de la fonctionnalité de **trace des objets créés à l'interface.**
CABRI-Géomètre intègre aussi une fonctionnalité de **vérification d'une propriété**, appelée oracle. Cet oracle propose la vérification de cinq propriétés : appartenance, parallélisme, perpendicularité, alignement, longueurs égales. Cependant la réponse que donne *CABRI-Géomètre*

[17] *LILIMATH* est un ensemble de programmes écrits par des professeurs de Mathématiques pour répondre aux besoins ressentis dans leurs classes. Ces outils sont rassemblés et diffusés par l'IUFM de Lille.
[18] Une version de ce logiciel est commercialisée par Texas Instrument

ne s'appuie pas sur des capacités de raisonnement. En effet, pour déterminer si une propriété est vraie, *CABRI-Géomètre* déplace légèrement un point libre choisi aléatoirement (les auteurs parlent d'animation de la figure). Si après plusieurs modifications de ce type, la propriété reste vérifiée alors elle est dite visiblement vraie. Cependant rien ne permet d'affirmer en général que la réponse donnée est toujours valide. Elle est uniquement probablement valide. Nous parlons pour cela de vérification empirique (ou probabiliste) de propriété.
Par ailleurs, les figures obtenues constituent des documents graphiques imprimables directement ou insérables dans un traitement de texte. Il s'agit de la fonctionnalité d'**exportation de la figure.**

Nous présentons ci-dessous d'autres logiciels semblables à *CABRI-Géomètre*.
Chamois[19] est un logiciel qui offre les mêmes fonctionnalités que *CABRI-Géomètre*, implantées avec des choix complètement différents [Chamois http]. Nous ne décrivons pas tous ces choix en détail (car ce n'est pas le but de notre propos), mais en voici quelques uns. Initialement, *CABRI-Géomètre* a été développé sur Macintosh. Pour combler le besoin d'un logiciel de ce type tournant sur PC, *Chamois* a été développé. Depuis *CABRI-Géomètre* existe sur PC.
Une autre différence est que la vérification de propriété, qui est probabiliste dans *CABRI-Géomètre*, est analytique dans *Chamois*.
De même, *Geometer's Sketchpad* [Jackiw 95] est un logiciel américain commercialisé, dont les fonctionnalités sont semblables à celles de *CABRI-Géomètre*.

Dans cette section, nous avons mis en évidence les fonctionnalités **[F_{14}.] exploration visuelle de figure géométrique**, **[F_{15}.] exportation de la figure**, **[F_{17}.] édition de figure géométrique**, **[F_{20}.] trace des objets créés à l'interface** et **[F_{21}.] vérification d'une propriété** en vue d'une utilisation dans l'atelier.

Calques 2

Calques 2 [Bernat 94a] est un micromonde de constructions géométriques. *Calques 2* est un produit commercial.
L'apprenant doit tracer une figure (**édition de figure géométrique** et **exploration visuelle de figure géométrique**). *Calques 2* fournit pour ces fonctions un environnement analogue à celui proposé par *CABRI-Géomètre*.
De plus, Calques permet l'**extraction de sous-figures** caractéristiques sous la forme de «calques» afin d'aider l'apprenant à appliquer les théorèmes de cours. Enfin, lors de l'interaction de l'apprenant avec le micromonde de construction, une **trace des objets créés à l'interface** est sauvegardée sous forme d'un historique.

Nous retenons les fonctionnalités **[F_{14}.] exploration visuelle de figure géométrique**, **[F_{16}.] extraction de sous-figures**, **[F_{17}.] édition de figure géométrique** et **[F_{20}.] trace des objets créés à l'interface** en vue d'une utilisation dans l'atelier.

CHyPre

CHyPre [Bernat 94b] (acronyme pour **C**onjecture, **H**ypothèse, **Pre**uve) est un micromonde de constructions géométriques et de démonstation, construit autour de *Calques 2*.
CHyPre fournit une assistance pour la résolution de problèmes lors de certaines étapes de résolution. Dans la première étape, on retrouve toutes les fonctionnalités de *Calques 2* (**édition de figure géométrique**, **exploration visuelle de figure géométrique**, **trace des objets créés à l'interface** et **extraction de sous-figures**). Dans la seconde étape, l'apprenant définit des

[19] © Bourit Cyril 1996

faits géométriques et leur donne un statut d'hypothèse ou de conjecture (**analyse de l'énoncé**). Dans la troisième étape, *CHyPre* fournit un graphe visuel qui représente l'état de résolution du problème par l'apprenant à un moment donnée. Pour ce faire, il présente les hypothèses, les conjectures et les théorèmes que l'apprenant utilise (**aide au raisonnement par visualisation de l'état de résolution**) au fur et à mesure. Dans cette étape, l'utilisateur introduit des faits géométriques (par exemple, B est le milieu de $[AC]$[20]). Il leur attribue un statut : hypothèse ou conjecture. *CHyPre* manipule un ensemble d'implicites pour chaque type de fait (dans notre exemple, $AB = 1/2AC$[21] est un fait implicite). Au moment de l'introduction d'un fait, *CHyPre* associe au fait introduit les implicites qui lui sont associés. Il cherche ensuite à appliquer un des théorèmes dont les prémisses sont les faits existants, et dont la conclusion est le nouveau fait introduit. Si un théorème est applicable à partir des prémisses dont le statut est prouvé, le statut du fait introduit devient lui aussi prouvé. Ce faisant *CHyPre* constitue une chaîne de raisonnement dont le but est de prouver un fait.
De plus, *CHyPre* fournit des **rappels de cours** sous forme de figures caractéristiques liées à l'application des théorèmes du cours.

Nous retenons les fonctionnalités **[F_2.] aide au raisonnement par visualisation de l'état de résolution**, **[F_8.] analyse de l'énoncé**, **[F_{14}.] exploration visuelle de figure géométrique**, **[F_{16}.] extraction de sous-figures**, **[F_{17}.] édition de figure géométrique**, **[F_{19}.] rappels de cours** et **[F_{20}.] trace des objets créés à l'interface** en vue d'une utilisation dans l'atelier.

GéoSpécif

GéoSpécif [Bouhineau 97] est un micromonde de construction de figures géométriques (**édition de figure géométrique**).
Avec *GéoSpécif*, écrit dans le langage Prolog, l'utilisateur peut déplacer (**exploration visuelle de figure géométrique**) un objet géométrique quel que soit l'ordre dans lequel les objets ont été construits (contrairement aux manipulations autorisées par les logiciels de type *CABRI-Géomètre*). Cela est rendu possible en donnant une spécification logique à une figure géométrique, par la capacité de construire automatiquement une figure correspondant à cette spécification donnée à partir de points de base donnés et par la capacité de proposer toutes les animations possibles sur cette figure. Pour faire cela, il manipule des équations sur les coordonnées des points dans le plan.

Nous retenons les fonctionnalités **[F_{14}.] exploration visuelle de figure géométrique** et **[F_{17}.] édition de figure géométrique** en vue d'une utilisation dans l'atelier.

2.2.3 Tuteurs

Geometry Tutor

Geometry Tutor[22] [Anderson 85] est un tuteur intelligent.
La tâche de l'apprenant est de construire la démonstration d'une propriété géométrique. Il dispose d'une figure représentant l'énoncé des hypothèses (**présentation de figure**). Cette figure est agrémentée au cours de la construction de symboles représentant certaines propriétés effectivement démontrées (**aide dans l'élaboration d'une démonstration**). Cependant Geometry Tutor n'a pas les connaissances suffisantes pour faire lui-même la démonstration demandée à

[20] Lire : le point B est le milieu du segment $[AC]$
[21] Lire : la longueur entre A et B vaut la moitié de la longueur entre A et C
[22] Geometry Tutor est un produit commercialisé aux états-Unis.

l'apprenant. En effet, pour pouvoir aider l'apprenant, Geometry Tutor pré-enregistre les démonstrations faites par l'enseignant.

Nous retenons les fonctionnalités **[F_6.] aide dans l'élaboration d'une démonstration** et **[F_{18}.] présentation de figure** en vue d'une utilisation dans l'atelier.

Mentoniezh ou *Tigre*

Mentoniezh [Py 90][23] («géométrie» en Breton) est un tuteur intelligent de géométrie.
C'est un logiciel qui fournit de l'aide à l'apprenant lors de la réalisation de trois étapes : lire, prouver, rédiger. Dans la première étape, «lire - lecture et analyse de l'énoncé», l'apprenant doit distinguer, dans l'énoncé, ce qui est une hypothèse de ce qui est une conjecture (**analyse de l'énoncé** et **aide à l'analyse d'énoncé**). Le logiciel effectue ensuite un diagnostic de correction : toutes les hypothèses sont présentes et correctes, et la conclusion est bien identifiée comme une conjecture (**diagnostic d'analyse de l'énoncé**). Dans la seconde étape, «prouver - construction de la démonstration» l'apprenant élabore une démonstration pour la propriété demandée dans l'énoncé. Il dispose alors d'un ensemble de théorèmes qu'il doit appliquer, pour faire passer ses conjectures à l'état «prouvé». L'aide apportée par *Mentoniezh* dans cette phase consiste à mettre à disposition de l'élève les théorèmes du cours (**rappels de cours**) et à le guider dans l'application méthodique des théorèmes (**aide au raisonnement par application guidée du cours**). Dans la troisième étape, «rédiger : rédaction de la démonstration», *Mentoniezh* lui fournit une aide pour rédiger la démonstration dans le formalisme demandé par l'enseignant pour l'évaluation (**aide à la rédaction d'une démonstration**).
Mentoniezh est plus souple que Geometry tutor car il permet de construire la démonstration par morceaux non nécessairement reliés aux hypothèses et ou au but. D'autre part, il autorise l'apprenant à emprunter plusieurs voies. Cela est possible car *Mentoniezh* calcule tous les plans de démonstrations possibles à un instant donné. Ce principe met en œuvre la reconnaissance de plan ([Kautz 87]). Partant d'une description incomplète des actions effectuées par un sujet, la reconnaissance de plan s'attache à retrouver le but recherché par le sujet, ainsi que le plan qui sous-tend et relie ces actions. *Mentoniezh* cherche ainsi à identifier la démonstration de l'apprenant parmi l'ensemble des possibles et ceci afin de l'aider en cas d'erreur. Comme Geometry Tutor, *Mentoniezh* propose des explications à l'apprenant sur ses erreurs.

Nous retenons les fonctionnalités **[F_1.] aide au raisonnement par application guidée du cours, [F_4.] aide à la rédaction d'une démonstration, [F_5.] aide à l'analyse d'énoncé [F_8.] analyse de l'énoncé, [F_{12}.] diagnostic d'analyse de l'énoncé, rappels de cours** et **[F_{19}.] rappels de cours** en vue d'une utilisation avec l'atelier.

PACT

PACT ([Ritter 95]) est le tuteur proposé part Ritter et Koedinger. Il a pour but d'aider l'apprenant (**aide dans l'interaction**) à partir de l'**analyse des interactions** avec un logiciel donné, par exemple, un micromonde. Cette fonction d'aide implique un fonction de **diagnostic de besoin d'aide.**
Ce tuteur n'est pas dédié à la géométrie. Il s'agit plutôt d'un tuteur qu'on vient «brancher» (il s'agit donc d'un logiciel épiphyte*[24]) sur un autre logiciel (éducatif ou non), afin d'utiliser

[23] Tigre (© IRISA 96) est la version pour Windows de *Mentoniezh*.

[24] Par analogie avec la plante épiphyte en biologie, un logiciel épiphyte est un logiciel qui se greffe à un logiciel existant, afin de recueillir les données dont il a besoin et ce sans gêner le fonctionnement du second logiciel (sinon le logiciel serait parasite)

l'ensemble dans un contexte éducatif. Il est utilisé, par exemple, avec Excel[25] dans [Ritter 96] et avec Geometer's Sketchpad dans [Ritter 96, Ritter 97, Ritter 98].

Nous retenons les fonctionnalités **[F_7.] aide dans l'interaction**, **[F_9.] analyse des interactions** et **[F_{10}.] diagnostic de besoin d'aide** en vue d'une utilisation dans l'atelier.

2.2.4 Autres outils éducatifs

Bien que la catégorie de prototypes que nous caractérisons par «autres outils éducatifs» comprenne des outils variés (tels que calculatrices, tableurs, éditeur de textes, *etc.*), la présentation que nous faisons ici ne contient que des prototypes issus de la recherche sur les EIAO dans le domaine de la géométrie.

TALC

TALC [Desmoulins 94] (acronyme pour **T**uteur d'**A**ide **L**ogique à la **C**onstruction) est un logiciel qui utilise la figure construite avec *CABRI-Géomètre*. Il effectue un **diagnostic de correction de figure** créé par l'apprenant par rapport à la figure spécifiée dans l'énoncé de l'exercice. *TALC*, contrairement à ce que laisse penser son acronyme, n'est pas un tuteur. En effet, bien qu'il s'appuie sur des capacités propres de résolution de problème et/ou de raisonnement pour établir son diagnostic, il n'exploite pas ce dernier pour guider l'apprenant dans sa tâche. *TALC* établit son diagnostic et informe l'apprenant de la correction (ou non) de sa figure. *TALC* n'est pas non plus un micromonde. Nous le rangeons dans la catégorie des outils éducatifs.

Nous retenons la fonctionnalité **[F_{11}.] diagnostic de correction de figure** en vue d'une utilisation dans l'atelier.

DéFI

DéFI [Gras 88] est un tuteur d'**aide dans l'élaboration d'une démonstration** de géométrie.
Il propose deux types d'activités par rapport à une figure donnée. La première activité est l'**exploration conceptuelle de figure géométrique**. Dans cette activité, *DéFI* propose à l'apprenant une **aide à la décomposition d'un problème en sous-problèmes**. L'apprenant choisit la décomposition qui semble convenir. Il doit ensuite déclarer savoir ou non démontrer chacun des sous-problèmes de la décomposition choisie. S'il déclare ne pas savoir démontrer un sous-problème, celui-ci devient le problème courant. Le processus est recommencé jusqu'à ce que l'apprenant ait déclaré savoir démontrer le problème initial. La seconde activité est la démonstration. Dans cette activité, l'apprenant construit la démonstration du problème de but en sous-buts.
Cependant, les capacités d'explorations sont limitées à l'ensemble des possibilités prévues par les concepteurs de *DéFI*. Par conséquent *DéFI* ne permet pas la création par l'enseignant de ses propres exercices, seuls ceux définis par les concepteurs sont disponibles. Pour ces raisons, nous ne considérons pas *DéFI* comme un vrai tuteur, et nous le rangeons dans la catégorie des autres outils éducatifs.

Nous retenons les fonctionnalités **[F_3.] aide à la décomposition d'un problème en sous-problèmes**, **[F_6.] aide dans l'élaboration d'une démonstration** et **[F_{13}.] exploration conceptuelle de figure géométrique** en vue d'une utilisation avec l'atelier.

[25] © Microsoft

CABRI-DéFI

CABRI-DéFI [Giorgiutti 91, Baulac 91] est un système utilisant à la fois *CABRI-Géomètre* et *DéFI*.
Après que l'apprenant ait construit une figure dans *CABRI-Géomètre*, *CABRI-DéFI* l'évalue et guide l'apprenant dans sa tâche de démonstration (**aide dans l'élaboration d'une démonstration**). Le **diagnostic de correction de figure** construite est rudimentaire, obligeant l'apprenant à construire «mot à mot» les objets demandés. Par exemple, si l'apprenant doit construire trois points A, B et C, et s'il ne les construits pas dans cet ordre, sa figure sera dite fausse pour cette raison, même si elle est correcte par ailleurs.
De plus, les capacités d'exploration sont limitées à l'ensemble des possibilités prévues par les concepteurs de *DéFI*. Par conséquent *CABRI-DéFI* ne permet pas la création par l'enseignant de ses propres exercices, seuls ceux définis par les concepteurs sont disponibles. Enfin *CABRI-DéFI* ne possède pas de capacités de raisonnement, ses capacités en sont d'autant limitées.

Nous retenons les fonctionnalités **[F_6.] aide dans l'élaboration d'une démonstration** et **[F_{11}.] diagnostic de correction de figure** en vue d'une utilisation dans l'atelier.

Nous avons présenté des travaux en EIAO permettant de mettre en évidence un ensemble de fonctionnalités que nous voulons utiliser conjointement. Une synthèse est présentée à la section 2.5.1. De plus, nous reprenons en annexe A toutes les fonctionnalités identifiées ici.Dans la section suivante, nous présentons les produits de coopération et d'interopération existants, c'est-à-dire les produits dont le but est de permettre l'utilisation conjointe de logiciels.

2.3 Existant : utilisation conjointe de prototypes

Dans cette partie nous décrivons d'abord les divers types d'utilisation conjointe que nous rencontrons. Nous présentons ensuite les produits (prototypes et logiciels commercialisés) qui permettent de gérer cette utilisation conjointe. Pour les présenter, nous les regroupons suivant différents paradigmes que nous développons dans la suite de cette section.

2.3.1 Définitions : les divers types d'utilisation conjointe de logiciels

La figure 2.1 illustre divers types d'utilisation conjointe que nous rencontrons.
Le premier cas d'utilisation conjointe de logiciels est la **juxtaposition** de logiciels (1). Dans ce cas, les prototypes A et B sont utilisés conjointement pour réaliser la tâche* T de manière indépendante l'un de l'autre. Le prototype A réalise une partie de T tandis que le prototype B en réalise une autre. Aucune transmission d'information n'est nécessaire.

Dans le cas de la **transmission** (2), le prototype A produit un résultat. Ce résultat est fournit au prototype B. Il y a transmission d'informations. Le prototype B exploite ce résultat pour réaliser la tâche T.

Dans le cas de la **coopération** (3), les prototypes A et B participent parallèlement à la réalisation de la tâche T, chacun à leur manière, et peuvent pour cela se communiquer des informations (*cf.* systèmes distribués).
Nous disons que des prototypes **coopèrent** quand ils s'échangent des informations factuelles (des données), que chacun peut prendre en compte à sa manière. Dans ce cas, nous appelons **signaux d'informations**, les signaux échangés par les logiciels. Une transmission de signaux d'informations est symbolisée sur la figure 2.1 par une *flèche pleine courbe.*
Pour illustrer ce type d'utilisation, nous prenons l'exemple de *CABRI-Géomètre* et *TALC*.

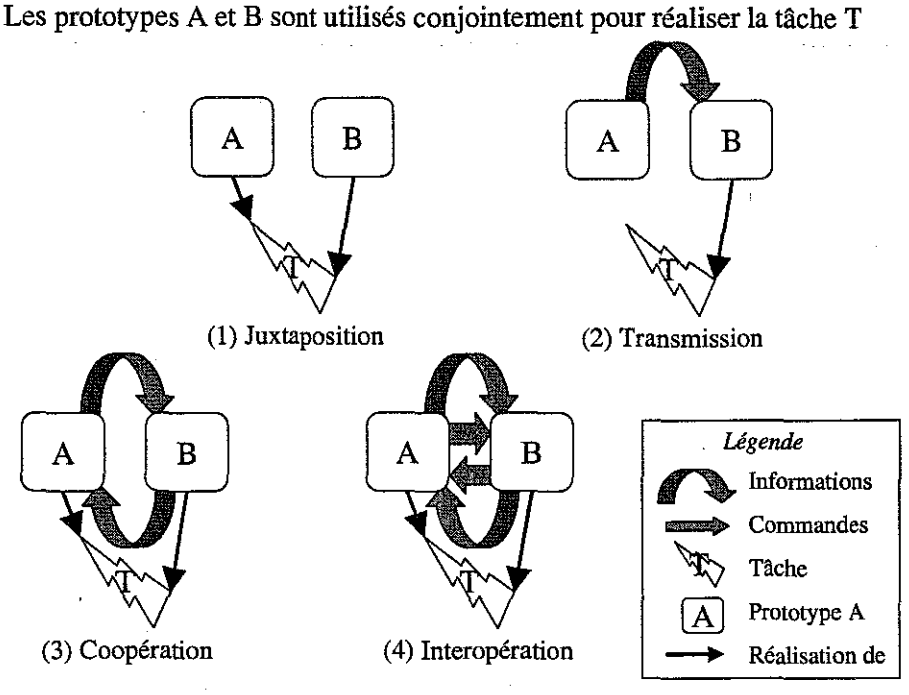


Figure 2.1 – Types d'utilisation conjointe de prototypes

CABRI-Géomètre émet des signaux de création d'objets ou de propriétés géométriques entre objets. *TALC* récupère ces signaux sur lesquels il s'appuie pour faire son diagnostic de correction de figure. *CABRI-Géomètre* et *TALC* participent ainsi ensemble à une tâche qui consiste à permettre à l'apprenant de tracer une figure correcte vis à vis d'un énoncé. Remarquons que les cas (1) et (2) sont des cas particuliers de (3). Nous considérons dans ce qui suit, que la transmission et la juxtaposition sont des formes de coopération et parlons dans ces cas de coopération.

Dans le cas de l'**interopération** (4), les prototypes A et B interagissent l'un sur l'autre et réalisent ensemble la tâche T.
Nous disons que des prototypes **interopèrent** lorsqu'à la fois ils coopèrent et qu'en plus, ils peuvent agir l'un sur l'autre. Dans ce cas, nous appelons **signaux de commandes**, les signaux autres que les signaux d'informations que s'échangent les prototypes qui interopèrent. Une transmission de signaux de commandes est symbolisée sur la figure 2.1 par une *flèche pleine droite.*
Dans notre exemple introductif (page 5), *CABRI-Géomètre* et *PACT* interopèrent. En effet, *CABRI-Géomètre* émet des signaux de création d'objets ou de propriétés géométriques entre objets. *PACT* récupère ces signaux sur lesquels il s'appuie pour faire son diagnostic : est-ce que l'apprenant à besoin d'aide ou non ? Une fois ce diagnostic effectué, si l'apprenant a besoin d'aide *PACT* peut prendre la main sur *CABRI-Géomètre*, pour rendre saillant certains objets en épaississant les segments d'une sous figure. Pour faire cela, *PACT* émet des signaux de commande vers *CABRI-Géomètre*. *CABRI-Géomètre* et *PACT* participent ainsi ensemble à une tâche qui consiste à aider l'apprenant dans la résolution d'un exercice de géométrie.

Remarquons que l'interopération est une forme enrichie de coopération. Ce dernier terme étant plus courant que le premier, nous désignons tous les types d'utilisation conjointe de logiciels par coopération de logiciels. Par conséquent, nous parlons dans la suite de ce mémoire de coopération de logiciels au lieu d'utilisation conjointe de logiciels.

Nous venons de fixer le vocabulaire qui va nous servir dans la suite de cette section à présenter

les produits existants. Pour présenter ces produits, nous avons choisi de les regrouper suivant différents paradigmes qui permettent l'utilisation conjointe de logiciels. Nous présentons d'abord les produits qui relèvent du paradigme «intégrateurs», fort présents sur le marché (2.3.2), puis ceux du paradigme client-serveur (2.3.3), ceux du paradigme composants (2.3.4) et ceux du paradigme multi-agents (2.3.5). Dans ces sections, nous définissons chaque paradigme et nous discutons son adéquation à notre atelier.

2.3.2 Produits de type coopération selon le paradigme intégrateur

Définition et exemples

Une première forme d'utilisation conjointe de logiciels est proposée par les intégrateurs, que nous définissons et illustrons ici. Dans le domaine de la formation, des systèmes ou plates-formes de télé-formation ou d'auto-formation apparaissent régulièrement avec des fonctionnalités qui évoluent à un rythme soutenu. Ainsi il est difficile de faire un inventaire exhaustif des produits actuellement disponibles sur le marché en France et à l'international. Quelques produits sont présentés dans le tableau 2.1. D'autres ressources, régulièrement mises à jour, peuvent être trouvées sur le site du ministère de l'éducation nationale [Educnet http] et sur les sites d'associations professionnelles comme le préau [Préau http] et l'eifel [Eife-l http].

Éditeur	**Produit(s)**
Asymetrix	*Librarian*
Blackborad	*CourseInfo*
CitCom et Cyberion	*WebTutor*
IBM	*DLS* (Distance Learning Systems)
Lotus	*LearningSpace*
Macromédia	*Pathware*
Oracle	*OLA*
SoftArc	*FirstClass*
Sybase	*NGL*
WBT Systems (Dublin, Irlande)	*TopClass* [TopClass http] (anciennement *WEST*)
MEI TECHNOLOGIY CORPORATION	*XAIDA*
WEBCT	*WebCT* [WebCT http]

Table 2.1: Produits intégrateurs

Nous reprenons le terme «intégrateur» introduits dans le rapport du préau ([Préau http]) puis dans celui de l'éducation nationale ([Educnet http]) pour qualifier la catégorie dont relèvent ces produits d'intégration. Un intégrateur est un environnement qui permet de gérer et de diffuser des ressources (en particulier des ressources logicielles). En effet, il a été conçu dans le but d'utiliser des produits divers répartis sur un réseau afin de proposer de la formation à distance pour une organisation donnée (école, université, entreprise publique et privée, *etc.*). Un intégrateur est constitué d'une base de données de ressources pédagogiques (documents, logiciels, *etc.*). Ces ressources sont indépendantes. Elles peuvent être utilisées successivement. Les données concernant la gestion pédagogique sont incorporées au logiciel dans des formats propriétaires et restent propriétés du logiciel. Certaines données sont parfois collectées dans l'intégrateur, et sauvegardées à ce niveau. Ces données peuvent être consultées mais elles ne peuvent pas être communiquées à d'autres ressources de l'intégrateur.
En résumé, un intégrateur est (ici) un environnement informatique qui permet de juxtaposer des

logiciels, sans coopération effective.
Un intégrateur pédagogique propose différentes fonctionnalités dont :

- des services de communication générales : un intégrateur utilise un serveur (éventuellement un serveur Web) pour délivrer des informations et des questionnaires ;
- des fonctions de communication entre les personnes : communication asynchrone (courrier électronique, listes de diffusions, forum de discussion) ou synchrone (chat, ICQ) ;
- l'indexation, la gestion et l'accès aux ressources : des documents d'information (cours, plannings, description des modules), des outils (logiciels éducatifs, logiciels d'évaluation, questionnaires, *etc.*) ;
- la gestion pédagogique et administrative des usagers : inscription, suivis et marquages des modules suivis, mémorisation des résultats obtenus et en particulier la traçabilité des actions des différents usagers ;
- les statistiques automatiques sur l'utilisation des ressources, les résultats aux tests.

La dénomination d'intégrateur ici ne correspond pas à celle définie dans [Wassernan 89] présentée ci-dessous.

Classification des types d'intégration d'outils Dans le domaine du génie logiciel, le désir d'utiliser ensemble des outils prenant en charge des aspects différents du processus de développement est central pour les environnements d'ingénierie du logiciel assistée par ordinateur (en anglais, *CASE environments* pour *Computer-Aided Software Engineering Environments*). Dans ce cadre, [Wassernan 89] a défini cinq dimensions de l'intégration d'outils :

Plate-forme les outils sont exécutés sur le même environnement d'exploitation virtuel (c'est-à-dire les réseaux et les systèmes d'exploitation sont transparents pour les outils) ;

Présentation les outils disposent d'interfaces cohérentes (c'est-à-dire leur «look and feel» est commun) ;

Données les outils se communiquent les données à partager, via un système (appelé «dépôt») qui fait interface entre eux ;

Contrôle les outils se notifient des événements ;

Procédé le procédé (de génie logiciel) est géré du début à la fin, et les outils pertinents à un moment donné y sont associés.

Ces cinq dimensions sont représentées à la figure 2.2

Dans notre contexte La section suivante présente un type de coopération de logiciels plus étendu : le paradigme client-serveur.

2.3.3 Produits de type coopération selon le paradigme client/serveur

Définition et exemples

L'approche client-serveur pour l'utilisation de logiciels consiste à choisir un logiciel qui devient un serveur et un (ou plusieurs) autre logiciel client. Un serveur est une application qui assure en permanence des services à des clients. Il y a pour cela une communication par requête-réponse. Le client qui a besoin d'un service, émet une requête. Le serveur scrute ses clients pour détecter les requêtes produites afin de répondre et de fournir le service demandé.

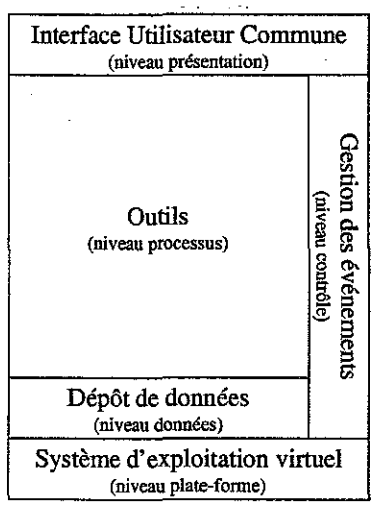


Figure 2.2 – Les cinq dimensions de l'intégration d'outils, selon A. Wasserman

Par exemple, en géométrie *CABRI-DéFI* [Giorgiutti 91, Baulac 91], *CABRI-Euclide* [Luengo 97a], Cabri-*TALC* [Desmoulins 94, Senet 93, Person 95], et *TéLéCABRI* [Tahri 93] sont construits autour d'une version serveur de *CABRI-Géomètre* [Baulac 90, Baulac 92]. Dans ces exemples, chacun des systèmes cités ajoute une fonctionnalité au logiciel *CABRI-Géomètre*, transformé en serveur de figures de géométrie. Autrement dit, la fonctionnalité d'édition et de manipulation d'une figure géométrique est prise en charge par *CABRI-Géomètre*, tandis que le nouveau logiciel prend en charge une nouvelle fonctionnalité.

Cette forme de coopération de logiciels nécessite une adaptation du logiciel choisi comme serveur. Il faut lui permettre de communiquer avec l'autre logiciel. Dans les exemples précédents, les systèmes sont développés sous Macintosh et utilisent les moyens de communication offerts par le système d'exploitation MacOS : les «apple-event» [Apple Computer 93, Tessier 94]. Le principe est que le logiciel client émet des événements (ici apple-event). Ces événements portent une sémantique que le logiciel serveur récupère et interprète. Par exemple, lors de la création d'un point, *CABRI-Géomètre* se charge de son côté, d'afficher ce point sur la figure à l'endroit désigné par la souris. En retour, il émet un événement dont la sémantique est «création d'un point». Cet événement est pris en charge par le système d'exploitation. Le logiciel client peut alors le récupérer et lancer le traitement associé si nécessaire.
On voit ici que l'utilisation de cet événement par le logiciel serveur nécessite une configuration de celui-ci (ou une adaptation pour les prototypes plus anciens). Remarquons que dans le cas de notre exemple, les logiciels clients sont des logiciels conçus spécialement pour implanter une fonctionnalité donnée en utilisant *CABRI-Géomètre* comme serveur de figures. Le lecteur intéressé par une liste plus détaillée de moyens de communication génériques entre micromondes et tuteurs peut se reporter à [Senet 93].

2.3.4 Produits de type interopération selon le paradigme composants

Définition et exemples

La solution proposée ici consiste à considérer chaque prototype comme un composant indépendant, et de composer une application en assemblant plusieurs composants. Par analogie au sens électronique du terme, un composant électronique seul, par exemple, un transistor, ne pro-

duit rien. On assemble différents composants pour constituer un circuit électronique, qui lui sera capable de remplir une fonction donnée. La technologie des composants constitue un mécanisme pour exprimer les entités de logiciels orientés-objets et pour les assembler dans des applications [Orfali 97].

Pour le Logiciels Éducatifs, actuellement, cet assemblage se fait de manière *ad hoc* dans [Cheikes 98, Koedinger 98, Ritter 96, Ritter 98, Suthers 97], parce que les composants entrant dans la composition de l'application sont soit des prototypes existants, soit des logiciels écrits spécialement pour l'application en cours. C'est une première étape indispensable avant de disposer d'un ensemble de composants réutilisables plus facilement.
Dans l'état actuel des développements, les prototypes utilisés dans ces développements nécessitent des modifications afin de permettre leur coopération avec d'autres composants. En effet, ces prototypes n'ont pas été conçus pour coopérer. Il faut donc les modifier pour leur permettre de communiquer avec d'autres logiciels. Par exemple, dans [Ritter 96], les prototypes ont été modifiés pour produire des apple-events, qui eux sont exploités par un tuteur qui leur est greffé. Il a fallu aussi agir sur l'interface du prototype, soit en ajoutant un menu qui donne accès aux fonctionnalités du tuteur, soit quand la solution précédente n'était pas possible, en désactivant provisoirement l'interface du prototype le temps d'afficher une fenêtre contenant un message envoyé par le tuteur. Cette fenêtre disparaissait quand l'utilisateur signalait qu'il l'avait vue. Ensuite l'interface du prototype était réactivée pour permettre à l'utilisateur de reprendre son interaction avec lui. Un autre type de modification du prototype a consisté à permettre au tuteur de prendre le contrôle du prototype le temps soit d'annuler la dernière action de l'utilisateur (fonction undo) afin de lui permettre de repartir d'un état précédent et de suivre les conseils du tuteur, soit de rendre saillants certains éléments de l'interface graphique pour indiquer provisoirement à l'utilisateur les objets importants sur lesquels il devait fixer son attention.
Les exemples cités ici sont implantés chacun sur une machine unique, et utilisent pour communiquer les primitives de communication proposées par le système d'exploitation (exemple : apple-events pour MacOS). Cependant, dans [Koedinger 98], on voit apparaître un ORB (Object Request Broker) qui prend en charge les communications. Un ORB est un équipement logiciel qui se place au milieu d'un ensemble de logiciels (c'est-à-dire un middleware [Wiederhold 92]) pour gérer leur communication. Dans cet exemple, les prototypes tournent encore sur la même machine, mais les communications sont gérées indépendamment du système d'exploitation. Par conséquent, on a ici, une sorte de gestion plus générale des communications, qui peut être exploitée lors de la coopération de prototypes tournant sur des machines distinctes, éventuellement de types distincts (Macintosh, PC et stations de travail sous Unix) et reliées par un réseau local ou via internet.
[Bourguin 01] retient aussi une solution utilisant un ORB. Cependant, le but des auteurs est l'intégration des outils de CSCL[26] à l'intérieur de plus grandes plate-formes comme les campus virtuels. Cette intégration est souvent impossible. C'est pourquoi, ils proposent un cadre théorique commun pour la conception de ces outils. Nous notons que l'architecture de cette proposition comporte des similitudes avec l'architecture de notre atelier. Cependant, nous n'avons pas élaboré ces architecture conjointement. Si elles ont été exprimées au même moment, nous pensons que c'est parce que les besoins étaient urgents et que les technologies actuellement disponibles nous amènent vers ce type d'architecture.

Le paradigme présenté dans la section suivante peut être vu comme une extension de celui-ci dans le sens où une certaine autonomie est donnée aux composants.

[26] *Computer-Supported Collaborative Learning* c'est-à-dire apprentissage collaboratif assisté par ordinateur

2.3.5 Produits de type interopération selon le paradigme multi-agents

Définition et exemples

Dans ce paradigme, les composants ont leur existence propre. Dans ce cas, chaque composant est doté d'une certaine autonomie et d'une capacité à prendre en compte un environnement. On appelle ces composants «des agents». La définition minimale d'un agent d'après Ferber [Ferber 95] est donnée sur la figure 2.3.

Un agent seul n'est d'aucune utilité. Il faut utiliser conjointement plusieurs agents dans une

Un agent est une entité physique ou virtuelle

1. *qui est capable d'agir dans un environnement,*
2. *qui peut communiquer directement avec d'autres agents,*
3. *qui est mue par un ensemble de tendances (sous formes d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),*
4. *qui possède des ressources propres,*
5. *qui est capable de percevoir (mais de manière limitée) son environnement,*
6. *qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),*
7. *qui peut éventuellement se reproduire,*
8. *dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.*

Figure 2.3 – Définition d'agent

«application», appelée Système Multi-Agents (ou SMA). La définition d'un système multi-agents est donnée sur la figure 2.4.

Cette définition des SMA montre que le type d'utilisation conjointe d'agents prévu dans un système multi-agents est au moins une coopération à cause de l'ensemble de relations R. De plus, puisque l'ensemble d'opération Op s'applique à tous les objets O, et donc en particulier aux agents, nous avons affaire à une **interopération**.

Dans le cadre des logiciels intelligents, chaque prototype est un agent qui permet de remplir une fonctionnalité. Par exemple, VISUAL GD ([Ulbricht 97]) est un environnement pour l'apprentissage de la géométrie descriptive développé suivant ce paradigme. Il comprend différents modules : un module de commandes, un module de décision, un groupe de modules tuteurs et une interface graphique qui est structurée à l'intérieur d'un hypermédia[27]. L'interopération entre ces différents modules a été pensée en modélisant ces modules en un système multi-agents.

Cette section termine la présentation des produits de coopération. Ils sont nombreux. Par conséquent, nous avons choisi de les présenter suivant différents paradigmes. Il serait cependant intéressant de disposer d'un standard pour décrire les produits de coopération et les logiciels

[27] Un hypermédia est un document permettant la navigation par le biais de liens hypertextes

Un système multi-agents ou SMA est un système composé des éléments suivants :

1. *un environnement E, c'est-à-dire un espace disposant généralement d'une métrique,*
2. *un ensemble d'objets O. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents,*
3. *un ensemble A d'agents, qui sont des objets particuliers (A inclus dans O ou égal à O), lesquels représentent les entités actives du système,*
4. *un ensemble de relations R qui unissent des objets (et donc des agents) entre eux,*
5. *un ensemble d'opérations Op permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O,*
6. *des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on appellera les lois de l'univers.*

Figure 2.4 – Définition de système multi-agents

éducatifs afin de faciliter leur réutilisation. La section suivante présente justement les tentatives de standardisation qui ont ces buts.

2.4 Existant : tentatives de normalisation des produits

Pour favoriser la diffusion et la réutilisation d'«objets pédagogiques» plusieurs initiatives internationales de standardisation ont été lancées. Pour [Bourda 00], les objets pédagogiques sont «par exemple, des transparents, des notes de cours, des pages Web, des logiciels de simulation, des programmes d'enseignements, des objectifs pédagogiques, *etc.* ». Les Logiciels Éducatifs, tels que nous les avons définis, font donc partie de ces objets pédagogiques.
Le but des initiatives internationales de standardisation des objets pédagogiques (et donc des Logiciels Éducatifs) est de définir des normes pour décrire, implanter et rechercher les composants éducatifs réutilisables (notamment sur le Web).

2.4.1 Présentation des organisations de normalisation

Le besoin de normes a été exprimé au même moment par plusieurs organismes et des propositions ont été élaborées. Les relations entre les différentes organisations sont complexes. Nous les présentons au travers de la figure 2.5.
Ces organismes sont par exemple :

- ADL (Advanced Distributed Learning) fournit des spécifications (requirements) pour AICC et IMS ;
- AICC (Aviation Industry CBT Committee) est un consortium issu de l'industrie de l'aviation. AICC développe des spécifications pour les logiciels éducatifs de l'industrie

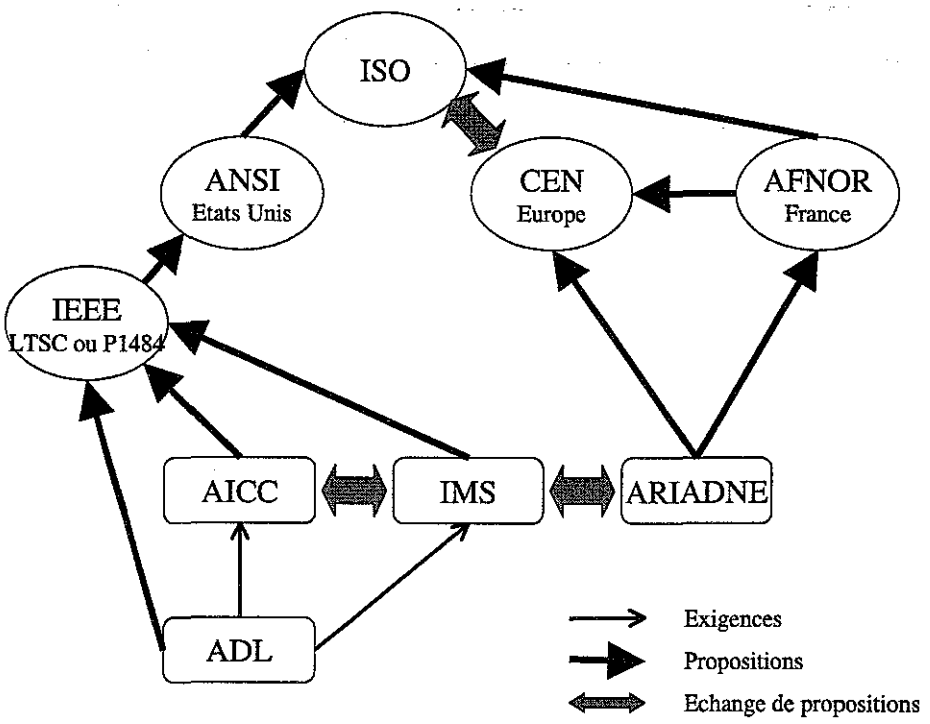


Figure 2.5 – Organisations qui proposent des standards

aéronautique. AICC soumet ses spécifications à IEEE P1484 pour une standardisation. AICC a publié une vingtaine de spécifications et de rapports techniques ;

- IMS (Educom's Instructional Management Systems) est un consortium issu des universités, des institutions, de compagnies commerciales et d'agences gouvernementales. IMS développe la technologie entre les participants, stabilise la technologie via des implantations échantillons («sample implementations») et soumet des spécifications à IEEE P1484 ;
- le projet ARIADNE [ARIADNE http] (Union Européenne) est constitué de participants européens qui développent et étendent des métadonnées (*cf.* section 2.4.3) dans le contexte éducatif. ARIADNE travaille en étroite collaboration avec IMS pour la spécification de métadonnées.

Ces propositions sont ensuite filtrées, validées et proposées pour approbation a des organisations internationales :

- IEEE [28] P1484 (connu aussi en tant que LTSC - Learning Technology Standards Committee [LTSC http]) est un comité de standardisation pour développer des standards techniques dans les technologies éducatives. AICC, IMS et ARIADNE fournissent des spécifications à IEEE P1484 pour standardisation ;
- ANSI (American National Standards Institute). Après approbation des standards développés par IEEE P1484, ils sont soumis à l'ANSI pour évaluation et validation ;
- AFNOR [AFNOR http] (association française de normalisation) est l'homologue français de l'ANSI. L'une de ses missions est de représenter et défendre les intérêts français

[28] IEEE (Institute of Electrical and Electronics Engineers, a pour but de favoriser les processus d'ingénierie, de création, de développement, d'intégration, de partage, et d'application des connaissances dans les domaines des technologies électriques et de l'information [IEEE http]

dans toutes les instances européennes et internationales de normalisation. Elle participe ainsi à l'élaboration d'une norme ISO pour les logiciels éducatifs ;

- CEN [CEN http] est le comité européen de standardisation. Ses membres sont les organismes de normalisation des pays membres de la CEE. La CEN ne siège pas directement à l'ISO. Il y a cependant un échange de propositions entre l'ISO et la CEN.

Dans le domaine des sciences et technologies de l'information et de la communication (STIC), beaucoup de normes émanent de IEEE. La dernière phase de normalisation est réalisé avec l'ISO (International Standards Organization) [ISO http], où contribue chaque pays[29] par le biais d'un organisme le représentant, par exemple :

- BSI (British Standards Institution) pour le Royaume Uni ;
- ANSI pour les États Unis ;
- AFNOR pour la France.

C'est ce qu'illustre la figure 2.6. Les pays soumettent ainsi leurs standards nationaux (pour

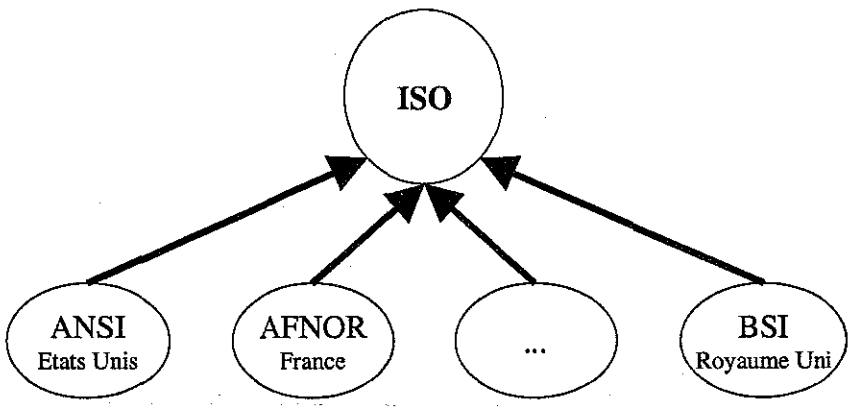


Figure 2.6 – Organisations qui développent des standards

évaluation et validation) à l'ISO par le biais d'un groupe de travail pour parvenir à un consensus international pour les objets pédagogiques. Ce groupe est le sous comité JTC1-SC36 (nommé Information Technology on Learning Technology).

À l'issue du processus de normalisation, un standard international devrait émerger. Pour l'instant, nous présentons le travail produit par les groupes P1484 -LTSC en rapport avec notre travail.

La mission des groupes de travail du LTSC est de faciliter le développement, le déploiement, l'entretien et l'interopération des implantations informatiques de composants et de systèmes de formation et d'éducation. Ils développent des normes techniques, des recommandations concernant les pratiques et éditent des guides relatifs aux composants logiciels, aux outils, aux technologies et aux méthodes de conception.

Ces groupes de travail et d'étude mis en route en 1996 ont pour objectif de déboucher sur des normes internationales. Les résultats des recherches de ces groupes constituent en particulier des propositions pour la norme ISO relatives aux technologies de l'apprentissage (ISO /IEC JTC1-SC36 - Learning Technology) [30].

[29] La liste des organismes membres de l'ISO est consultable à l'adresse : http ://www.iso.ch/addresse/membodies.html

[30] travail en cours encore en 2001

L'ambition de ces groupes de travail et d'étude est de couvrir l'intégralité des thèmes relatifs aux logiciels éducatifs. Ces derniers peuvent être regroupés en cinq types :

1. les groupes généraux ;
2. les groupes centrés sur l'apprenant ;
3. les groupes centrés sur le contenu ;
4. les groupes concernant les données et les métadonnées (métadonnée) ;
5. et les groupes qui se préoccupent de la gestion des systèmes et des applications.

Ces thèmes sont à prendre en compte lors de la création d'un logiciel éducatif intelligent. Dans une perspective de coopération de logiciels éducatifs, nous nous intéressons plus particulièrement aux points 1 (pour l'architecture, voir section 2.4.2) et 4 (pour les méta-données, voir section 2.4.3).

2.4.2 L'architecture LTSA

Le groupe P1484 .1[31] est appelé «Architecture and Reference Model Working Group».
Ce groupe de travail appartient aux groupes généraux. Il a pour but de définir une architecture et un modèle de référence pour les Logiciels Éducatifs à base de composants. Ce standard devrait

1. définir un cadre (framework) avec lequel les architecture(s) des Logiciels Éducatifs peuvent être décrites ;
2. définir un vocabulaire, y compris une notation graphique, pour décrire les architectures des Logiciels Éducatifs à base de composants ;
3. définir des formats, des protocoles et des méthodes pour l'échange d'informations entre composants d'un Logiciels Éducatifs ;
4. définir des interfaces externes (de programmation) requises et facultatives pour les composants des Logiciels Éducatifs ;
5. définir les exigences (requirements), normes et conventions pour le comportement des Logiciels Éducatifs ;
6. et indiquer les services externes que les Logiciels Éducatifs à base de composants doivent utiliser pour établir et assurer l'échange d'informations.

En outre, ce projet compte développer des directives de documentation et de configuration pour des composants de Logiciels Éducatifs.
L'architecture produite par ce groupe est appelée LTSA pour Learning Technology Systems Architecture. Les composants LTSA identifient les interfaces d'interopérabilité minimales pour les logiciels éducatifs. Ils n'identifient pas les interfaces d'interopérabilité pour une application particulière ou un système d'exploitation particulier. Ils n'identifient pas non plus les interfaces d'interopérabilité pour les systèmes apparentés, comme les systèmes de développement des contenus ou de gestion administrative des formations, des formateurs et des apprenants (ce qui est bien dommage).
Les composants spécifiés ici sont aussi génériques que possible. Les Logiciels Éducatifs ne suivent pas forcément strictement cette architecture, mais constituent des variations de leurs implantations. Nous présentons l'architecture LTSA sur la figure 2.7[32]. Sur la figure 2.7, les ovales représentent des processus (*processes*), c'est-à-dire un composant actif qui transforme ses entrées

[31] Des informations sur ce groupe de travail sont disponible à l'adresse : http ://ieee.ltsc.org/wg1
[32] Version provisoire du 14 novembre 2000, susceptible de changer

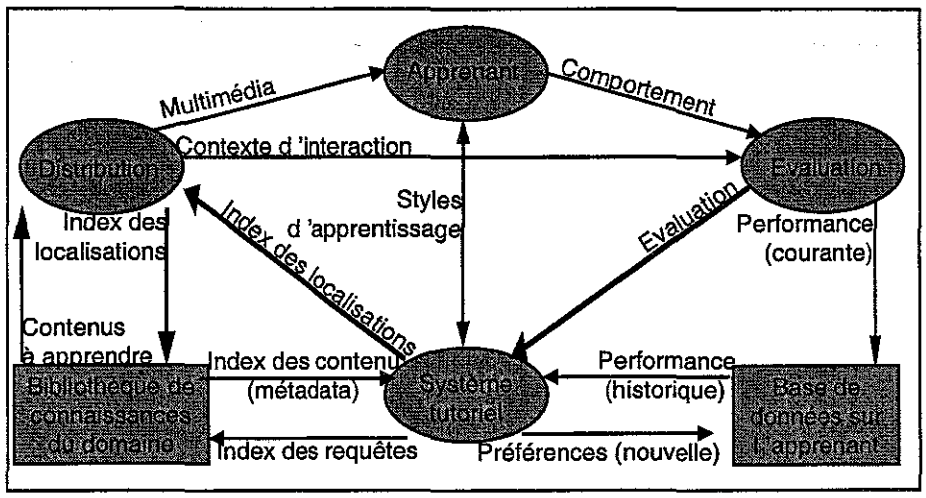


Figure 2.7 – Les composants de l'architecture LTSA

en ses sorties[33]. Ces processus sont nommé «Apprenant» (*Learner Entity*), «Évaluation» (*Evaluation*), «Système tutoriel» (*Coach*) et «Distribution» (*Delivery*).
Les rectangles représentent des mémoires (*Stores*), c'est-à-dire un composant inactif utilisé pour stocker et récupérer des informations. Ces mémoires sont nommées «Base de données sur l'apprenant» (*Learner Records*) et «Bibliothèque de connaissances du domaine» (*Learning Ressources*).
Les flèches représentent des flux d'informations (*flows*), c'est-à-dire le transfer d'information d'un composant à un autre. Les flux d'informations sont nommés «Styles d'apprentissage» (*Learning Preferences*), «Comportement» (*Behavior*), «Évaluation», «Performance» et «Préférences» (*Assessment, Performance and Preference*), «Index des requêtes, des contenus et des localisations» (*Query, Catalog Info and locator*), «Contenus à apprendre» (*Learning Content*), «Multimédia» (*Multimedia*) et «Contexte des interactions» (*Interaction Context*).

Dans l'architecture LTSA chaque processus, mémoire ou flux d'information est décrite précisément. Nous donnons ici l'exemple de la description du processus «Distribution» illustré sur la figure 2.8.
«Distribution» est un processus. Il reçoit des «localisations» de la part du «Système tutoriel» et

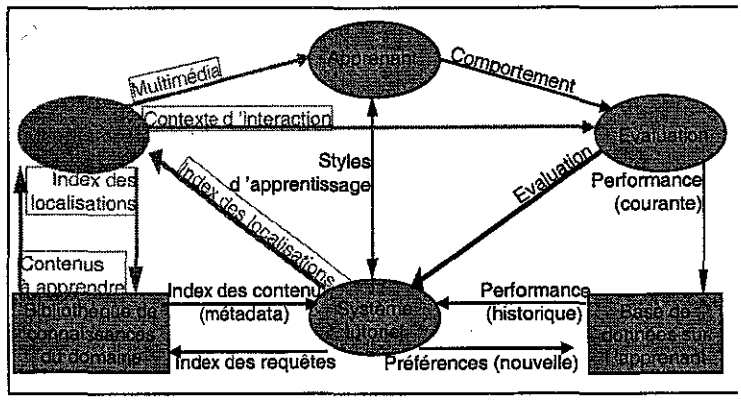


Figure 2.8 – Le processus «Distribution» de l'architecture LTSA

[33]Chaque terme est défini par le groupe de travail (wg3, *working group number* 3) établissant le glossaire des termes pour LTSC, *cf.* http ://ltsc.ieee.org/wg3

récupère des «Contenus à apprendre» depuis la «bibliothèque de connaissances du domaine» Il transforme les «Contenus à apprendre» en une présentation, qui est transférée à l'apprenant via un support «Multimédia». La présentation peut être statique, interactive, collaborative, impliquer une expérimentation, *etc.*. «Distribution» peut être combiné avec le processus d'«Évaluation» (qui récupère le «Comportement» de l'apprenant), afin de fournir des rétroactions adaptées dans l'interactivité. Les méthodes de transformation des «Contenus à apprendre» vers des supports «multimédia» ne sont pas spécifiées. Elles peuvent prendre des formes variées, *e.g.* présentation et questions, TI (donc Logiciels Éducatifs), vidéo-conférence avec un tuteur humain *etc.*.

La section suivante présente le résultat du travail d'un autre sous-groupe de LTSC.

2.4.3 Les métadonnées

Les méta-données ont pour but d'ajouter une information de nature sémantique aux objets pédagogiques de manière à en obtenir une description aussi précise que possible. Étymologiquement, les méta-données sont des données sur les données. Nous retenons une définition plus concrète sur la figure 2.9.

Une métadonnée est une information ajoutée à un texte ou un logiciel, afin de donner des informations sur son contenu.
Par exemple, des métadonnées indiquant un type (logiciel de bureautique, Logiciels Éducatifs, etc.), un domaine d'application (géométrie, économie, etc.), des fonctionnalités, etc. peuvent être ajoutées à un logiciel.
De même, dans un texte au format html, les métadonnées sont signalées par des balises commençant par le mot clé «meta».
Exemple d'une telle métadonnée :
<meta name="GENERATOR" content="Mozilla/4.6 [fr] (Win95; I) [Netscape]">
Cette ligne définit une métadonnée «GENERATOR» (champ «name») à laquelle un contenu (champ «content») est associé.

Figure 2.9 – Définition de méta-données

Nous pensons, comme [Bourda 00], que la capacité des métadonnées à faciliter l'accès aux descriptions des Logiciels Éducatifs dépend grandement de l'existence d'un standard.

ARIADNE , IMS et CEN[34] se sont impliqués au sein du groupe de travail de LTSC sur les métadonnées pour les objets éducatifs : *LOM* pour *Learning Object Metadata*. Ce standard est en cours de spécification. Cependant dans sa version provisoire (version 3), il définit neuf catégories de descripteurs :

1. *general* : caractéristiques indépendantes du contexte, par exemple, un identificateur global unique, le nom de la ressource, le langage, *etc.* ;
2. *lifeCycle* : caractéristiques relatives au cycle de vie de la ressource, par exemple, la version, l'état (Draft, Final, Revised, Unavailable) ;
3. *meta-metadata* : caractéristique de la description elle-même, par exemple, les personnes qui y ont contribué ;

[34] http ://www.cenorm.be/isss/Workshop/lt/lom-localization/LOM-French-v3_8.htm

4. *technical* : caractéristiques techniques ;
5. *educational* : caractéristiques pédagogiques ;
6. *rights* : caractéristiques exprimant les conditions d'utilisation de la ressource ;
7. *relation* : caractéristiques exprimant les liens avec d'autres ressources ;
8. *annotation* : commentaires sur l'utilisation pédagogique de la ressource ;
9. *classification* : caractéristiques de la ressource décrites par des entrées dans les systèmes de classification.

2.5 Discussion : caractéristiques de l'atelier résultant de l'étude de l'existant

Cette section synthétise l'état de l'art présenté ici afin de préciser le contexte de définition de l'atelier. L'étude des principaux Logiciels Éducatifs en géométrie (section 2.2) permet de mettre en évidence les fonctionnalités complémentaires qu'ils offrent. La section 2.5.1 recense les fonctionnalités relevées dans cette étude. En ce qui concerne l'interopérabilité entre logiciels, plusieurs paradigmes ont été proposés pour y parvenir (section 2.3). Tous ne sont pas adaptés à notre projet. La section 2.5.2 extrait de l'existant en matière d'utilisation conjointe de prototypes, ce qui doit être fait, développé ou réutilisé. Enfin, si notre besoin de décrire les Logiciels Éducatifs au sein d'une plate-forme rejoint les préoccupations des organismes de normalisation, les premières propositions de ceux-ci (section 2.4) n'apportent pas de solution à nos besoins de communication de données entre logiciels. La section 2.5.3 extrait des tentatives de normalisation des produits, ce qui est utilisable pour notre atelier.

2.5.1 Un atelier offrant des fonctionnalités des EIAO de géométrie

Une liste de fonctionnalités

L'étude des prototypes de géométrie présentée dans la section 2.2 nous permet d'établir une première liste de vingt et une fonctionnalités disponibles et donc candidates à une utilisation dans l'atelier (*cf.* annexe A)

Ces fonctionnalités sont complémentaires pour la conduite d'un exercice du début à la fin de celui-ci.

Commentaire Cette liste est évidemment provisoire, d'autres viendront la compléter. On peut penser, par exemple, aux affichages de parcours de certains logiciels hypermédia, à la présentation des modèles (du domaine ou de l'apprenant), *etc.*.

Regroupement des fonctionnalités suivant la présentation classique des EIAO

Les fonctionnalités sont regroupées au sein de chaque prototype dans des «modules» particuliers selon les implantations et architectures choisies. De façon générale, dans la présentation classique des EIAO, il y a quatre modules : «domaine», «interface», «modèle de l'apprenant» et «guidage pédagogique» [Quéré 91] (voir figure 2.10).

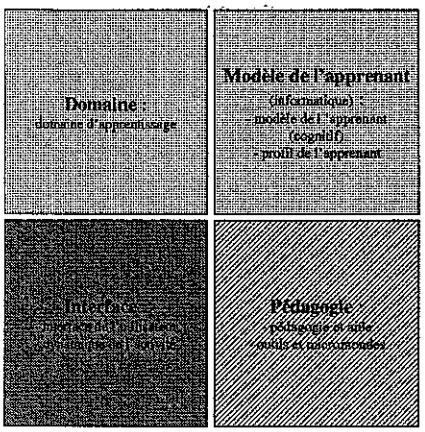


Figure 2.10 – Présentation classique des EIAO

Commentaire Ce découpage en modules ne nous permet pas, par exemple, de distinguer les traitements qui relèvent de la pédagogie de ceux qui sont de simples aides techniques apportées aux apprenants. La figure 2.10 présente les modules de la présentation classique des EIAO. Elle introduit les subdivisions dont nous avons besoin.

Regroupement des fonctionnalités en cinq groupes

Pour regrouper les fonctionnalités listées précédemment en tenant compte des subdivisons précédentes, nous proposons une répartition de ces fonctionnalités dans cinq groupes :

(a) domaine d'apprentissage — Il comprend la théorie du domaine d'apprentissage, des rappels de cours, une base d'exercice et des modèles de l'apprenant. Il inclut le module «domaine» et une partie du module «modèle de l'apprenant»[35] du modèle classique en EIAO.

(b) pédagogie et aide — Il concerne l'implantation de l'aide et de la pédagogie dans le prototype.

(c) outils et micromondes — Il concerne les outils ou micro-mondes fournis à l'utilisateur.

(d) dynamique de l'activité — Il concerne l'interaction de l'utilisateur avec le prototype.

(e) interface de l'utilisateur — Il concerne la présentation des quatre groupes précédents à l'interface des divers utilisateurs (apprenants, enseignants et chercheurs).

Commentaire Les groupes (b) et (c) correspondent à une décomposition du module «Pédagogie», où (b) rassemble les composants relatifs à la pédagogie, tandis que le groupe (c) rassemble tous les outils qui apportent une aide plus technique à l'apprenant.

Les groupes (d) et (e) correspondent à une décomposition au module «Interface» où la gestion de l'interactivité est séparée de la présentation à l'interface.

Nous n'avons pas ajouté un sixième module concernant l'apprenant. En effet, nous considérons que les modèles de l'apprenant ne sont pas des modèles concernant la personne en tant que telle

[35]Le modèle de l'apprenant concerne sa relation au domaine d'apprentissage. C'est le profil de l'apprenant qui concerne un individu donné. C'est donc le profil qui contient des informations qui n'appartiennent pas au domaine d'apprentissage.

mais concernant sa relation aux connaissances qui font l'objet de l'apprentissage. Par conséquent, nous plaçons le modèle de l'apprenant dans le groupe «domaine d'apprentissage», tandis que nous plaçons son profil dans le groupe «dynamique de l'activité». En effet, c'est lors de l'interaction de l'apprenant avec le prototype que sont collectées les informations concernant l'apprenant en rapport avec le modèle cité précédemment.

Recouvrement des regroupements

La figure 2.11 correspond à une vue orientée «fonctionnalité» des EIAO. Dans cette figure,

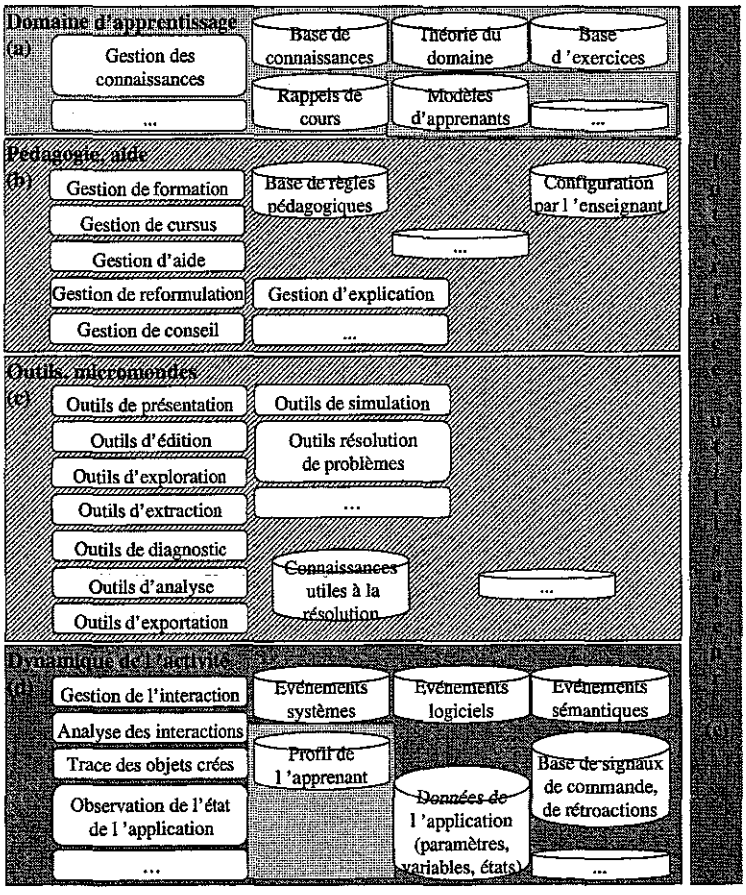


Figure 2.11 – Présentation des EIAO en cinq groupes

chaque groupe (présenté dans un grand rectangle) comprend des bases d'informations (représentées sous forme de cylindres) et des traitements (représentés sous forme de rectangles aux coins arrondis). Chaque traitement implique une ou plusieurs bases d'informations.

Commentaire Nous recouvrons donc grosso modo le modèle classique avec nos cinq groupes suivant la figure 2.12. Le recouvrement n'est pas parfait car, par exemple, la dynamique de l'activité de l'utilisateur est souvent implantée dans le module «Pédagogie».

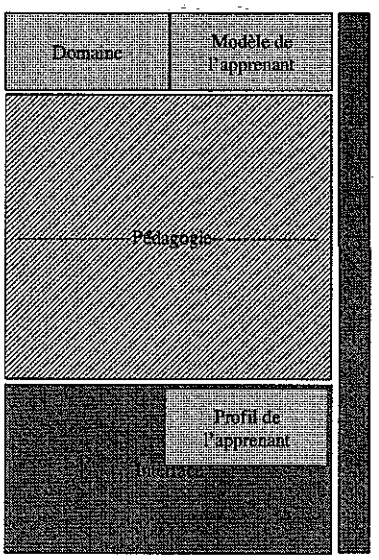


Figure 2.12 – Recouvrement du modèle classique en 4 modules

Localisation des fonctionnalités identifiées dans les cinq groupes

La figure 2.13 fait correspondre les fonctionnalités aux cinq groupes définis. Les cercles sur la partie gauche de la figure correspondent aux fonctionnalités identifiées dans l'état de l'art.

Commentaire Cette figure met en perspective le contraste entre d'une part, le grand nombre des fonctionnalités implantées pour les groupes (b) et (c), et d'autre part, le peu de fonctionnalités relatives aux groupes (a) et (d). La présentation en cinq groupes constitue une vision plus large (différente, si ce n'est meilleure) des EIAO dont le modèle classique ne rendait pas bien compte. Notez qu'aucun EIAO de géométrie existant ne contient tous les éléments présents sur la figure 2.13. Nous caractérisons cependant un EIAO par les éléments qu'il implante. Nous reprenons d'ailleurs cette présentation pour l'indexation des prototypes (voir section 1.2.1).

La suite de la synthèse correspond à l'existant en matière de coopération de prototypes.

2.5.2 Un atelier pour la coopération de prototypes

À la section 2.3.1 p 44, nous avons défini la coopération et l'interopération de logiciels. Notre atelier rentre dans la catégorie des produits d'interopération.

L'atelier peut-il être un système intégrateur (au sens du préau) ?

L'atelier devrait permettre au moins la juxtaposition de prototypes de logiciels éducatifs. Dans ce sens, il devrait au minimum être un intégrateur. Cependant, intégrer des prototypes avec un intégrateur est une forme de coopération qui n'est pas satisfaisante pour notre propos, et ceci pour deux raisons.

La première est que le transfert de données n'est pas suffisamment développé. En effet, nous avons besoin de transférer des données et des résultats entre les ressources de la «base d'outils» pour éviter de ressaisir certaines informations dans les différents logiciels inclus dans l'intégrateur. Cette possibilité est assez limitée actuellement, même si on peut noter des tentatives. Par

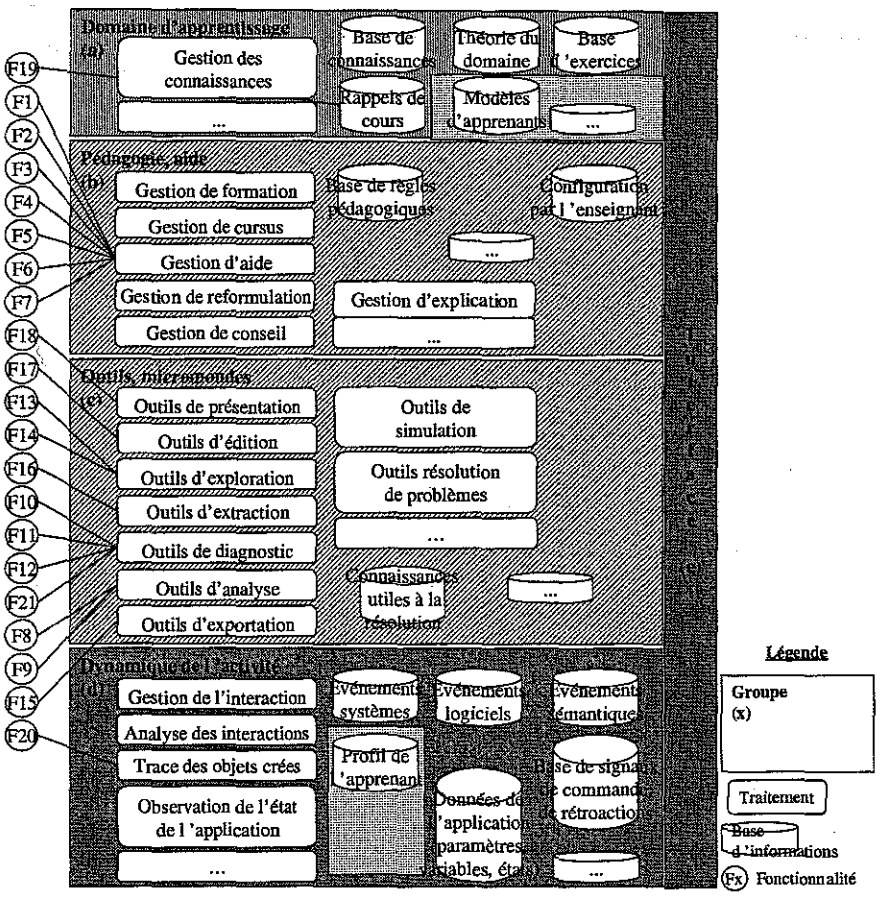


Figure 2.13 – Répartition des fonctionnalités en cinq groupes

exemple, l'intégrateur *WebCT* et l'outil logiciel d'auto-évaluation *Perception* utilisent les mêmes formats. Ainsi les résultats des analyses des tests produits par *Perception* peuvent être exploités par *WebCT*. De même, *Apogée*, l'outil de gestion administrative des étudiants et des cursus à l'université (pour toutes les universités françaises), permet d'identifier les étudiants par un login et un mot de passe. Cette identification est ensuite communiquée et vérifiée lorsque les étudiants se connectent pour des sessions de travail avec les logiciels que l'université met à leur disposition. La seconde raison est que l'interopération n'est pas prévue. En effet, les intégrateurs ne prévoient pas qu'un logiciel puisse agir sur un autre. Cela exclut donc un type de coopération de logiciels nécessaire à notre atelier.

L'atelier peut-il être un système intégrateur (au sens de A. Wasserman) ?

Le procédé à gérer pour nous est une activité. À la section 1.2.2 p 18, nous avons exprimé le besoin d'intégration aux niveaux contrôles et données. À la section 1.2.3 p 22, nous avons exprimé le besoin d'intégration au niveau présentation. Tous ces niveaux d'intégration requièrent l'intégration au niveau plate-forme. Par conséquent, les cinq dimensions de l'intégration d'outils (plate-forme, présentation, données, contrôle et procédés) sont nécessaires pour notre atelier et les outils (éducatifs) que nous voulons utiliser avec lui. Pour chacune des dimensions, il existe des choix d'intégration possibles. Par exemple, l'intégration des données peut avoir lieu via quatre médias ([Wassernan 89]) :

- message
- fichier
- base de données
- objet

Si tous les outils utilisent le même média, leur intégration est facilitée. La prise en compte de tous les choix possibles dans l'atelier constitue une complexité énorme. D'où l'intérêt de la normalisation ces choix (ou au moins la diminution de ceux-ci) pour tous les outils d'un domaine (pour nous, les logiciels éducatifs).

L'atelier peut-il être un système client-serveur ?

L'atelier peut tout à fait être défini comme un système client-serveur.

Le paradigme client-serveur permet de partir d'un logiciel existant, et après modifications, de lui ajouter des fonctionnalités en fabriquant un nouveau logiciel qui fera appel à ses services. Dans la pratique, l'utilisation de logiciels préexistants est difficile. Le paradigme composants présenté dans la section 2.3.4 permet une généralisation de cette approche client-serveur en rendant faisable l'utilisation de plusieurs logiciels préexistants.

L'atelier peut-il être un système à base de composants ?

Ce paradigme :

- permet la communication de données ;
- assure la communication entre prototypes ;
- gère les différentes interfaces graphiques ;
- définit un protocole de coopération entre les prototypes ;
- est capable de commander les prototypes.

Ici, ces exigences pourraient être celles de l'atelier.
Par ailleurs, pour permettre la coopération de prototypes, ce paradigme requiert de la part de chaque prototype :

- d'assurer la communication des données au format adéquat (souvent de manière *ad hoc* tant que des normes n'auront pas été publiées) ;
- de permettre une certaine forme de contrôle (au moins rudimentaire) ;
- de gérer des interfaces graphiques (sans aller jusqu'à exporter son interface graphique) ;
- d'être rendu actif ou inactif, totalement ou partiellement ;
- de permettre qu'on observe certains de ses états ou de ses variables.

Par conséquent, ce paradigme correspond assez à ce qui est recherché dans la mise en œuvre de l'atelier.

L'atelier peut-il être un système multi-agents ?

Pour que l'atelier défini dans ce mémoire soit un système multi-agents, il faut qu'il satisfasse les deux conditions définies par Ferber dans [Ferber 95] (page 63), c'est-à-dire il faut :

1. «qu'il dispose d'agents autonomes fonctionnant en parallèle et cherchant à satisfaire un but ou une fonction de satisfaction ;»

2. «que ces agents possèdent un mécanisme d'interaction de haut niveau indépendant du problème à résoudre (protocoles de communications ou mécanismes d'interaction avec l'environnement).»

Dans le cas de l'atelier le but à satisfaire est de mener une activité comportant plusieurs phases. Certains agents, les «agents-prototypes» proposeraient chacun une fonctionnalité nécessaire au cours d'une phase de l'activité. Pour que ces prototypes deviennent agents autonomes, puissent fonctionner en parallèle avec d'autres agents et possèdent un mécanisme d'interaction, il faudrait adapter les prototypes existants.
Ces adaptations nécessitent un gros travail, que nous pourrions faire uniquement pour les prototypes issus de la recherche selon l'autorisation des auteurs. En revanche, demander l'adaptation de logiciels commercialisés est plus délicat. Par ailleurs, que faire pour les prototypes les plus anciens, dont les auteurs ne sont plus disponibles ?
Par conséquent, l'atelier pourrait être vu comme un système multi-agents, mais nous écartons cette approche.

Synthèse

Parmi les produits de coopération, les intégrateurs sont un concept minimal pour décrire notre atelier. En effet, nous ne voulons pas seulement proposer des services autour d'une juxtaposition de prototypes éducatifs.
Les produits de coopération de type client-serveur permettent de retenir un mode d'articulation des prototypes éducatifs qui coopèrent. La première manière de le voir consiste à définir l'atelier comme un serveur dont les prototypes sont les clients. L'autre manière de le voir est de définir l'atelier comme un client dont tous les prototypes sont des serveurs. Nous y reviendrons dans le chapitre 3.
Les produits de coopération à base de composants permettent de prévoir une évolutivité de l'atelier. En effet, chaque prototype peut être considéré comme un composant. Et ajouter un prototype dans l'atelier correspond à ajouter un composant. De plus, nous avons mis en évidence une liste de propriétés pour ce type de produits d'interopération et pour les prototypes impliqués.
Quant aux systèmes multi-agents, nous n'avons pas choisi ce paradigme pour notre atelier. Cependant, il est possible qu'un prototype de Logiciels Éducatifs développé comme un système multi-agents, puisse être intégré à notre atelier, à la manière d'un composant. De même, il serait possible au prix d'un aménagement d'un composant, d'intégrer à notre atelier un agent particulier issus d'un SMA.
En conclusion, notre atelier a pour vocation d'être plus qu'un intégrateur. Il utilisera les concepts des paradigmes client-serveur et composant. Il n'est pas un SMA, mais n'exclut pas les prototypes de logiciels éducatifs issus de ce paradigme.

2.5.3 Un atelier prenant en compte la normalisation des produits

Comme nous l'avons expliqué dans l'introduction de ce mémoire, notre but, l'utilisation conjointe de logiciels éducatifs, rejoint ceux de la diffusion et de la réutilisation de logiciels. En conséquence, les travaux de normalisation sont une aide précieuse pour nous. En effet, si les prototypes de logiciels éducatifs futurs suivent une norme, nous devons la prendre en compte pour concevoir notre atelier. De plus, un atelier prévu pour utiliser ces nouveaux prototypes normalisés faciliterait le travail des utilisateurs en proposant un processus standard (voire automatique) d'incorporation de ces prototypes.

LTSA

Les Logiciels Éducatifs sont modélisables suivant l'architecture LTSA. Dans cette dernière toutes les fonctionnalités que nous envisageons dans ce travail sont incluses totalement dans le processus nommé «Distribution». Ainsi, il n'est pas possible d'utiliser directement la représentation des Logiciels Éducatifs suivant ce modèle pour l'utilisation que nous voulons en faire. Cependant, les informations dont nous disposons déjà nous permettent de travailler avec des composants «idéaux» respectant l'architecture LTSA présentée dans la section 2.4. Notre travail ne se limite pas seulement aux prototypes idéaux, mais aux prototypes existants. Par conséquent, nous pourrons vérifier que nos propositions sont compatibles à la fois avec les prototypes existants et avec les prototypes dont l'architecture est LTSA.

Métadonnées

Les catégories de descripteurs pour les métadonnées, bien que déjà très abouties, ne permettent pourtant pas de décrire les prototypes de géométrie suivant les fonctionnalités que nous voulons utiliser avec l'atelier. Les fonctionnalités pourraient constituer des propositions à l'intérieurs des groupes 5 ou 9.
De même, la DTD décrite dans [Crampes 99] permet de qualifier «des matériaux pédagogiques». Elle est conforme aux recommandations de description de documents pédagogiques précédentes. Cependant, elle ne permet pas non plus de décrire les prototypes suivant les fonctionnalités implantées.

Positionnement

Les processus de normalisation en cours ne sont pas suffisamment avancés pour qu'on puisse les exploiter. Par conséquent, nous utilisons la liste des fonctionnalités mises en évidence dans ce chapitre pour caractériser le prototype.

Dans ce chapitre nous avons extrait les fonctionnalités complémentaires à utiliser dans l'atelier. De plus, l'étude des produits de coopération nous a permis de décider que l'atelier utilise les paradigmes client-serveur et composant. Enfin, nous avons écarté la piste de la normalisation pour l'instant.

Chapitre 3

Propositions détaillées pour la réalisation de l'atelier

3.1 Introduction

L'atelier est un environnement informatique qui permet de concevoir et de réaliser une activité impliquant des prototypes de Logiciels Éducatifs.

Concevoir une activité avec l'atelier

Pour définir une activité, le prescripteur définit concrètement une hypothèse et détermine les caractéristiques des sujets, ainsi que les paramètres à faire varier, si l'activité est une expérimentation. Il définit ses objectifs pédagogiques s'il s'agit d'une séquence pédagogique. Toute cette première partie de la définition d'une activité a lieu en dehors de l'atelier.
Pour étayer l'hypothèse ou atteindre ses objectifs pédagogiques, il propose une tâche* ou une succession de tâches à faire exécuter par les sujets. Cette tâche ou cette succession de tâches constitue le déroulement de l'activité. L'atelier permet de décrire précisément ce déroulement via la fonction «*définir le déroulement de l'activité*». Cette fonction est décrite à la section 3.3.

Impliquer des prototypes

Dans notre contexte, une activité implique des prototypes. Il faut que ces prototypes soient connus de l'atelier. C'est pourquoi, l'administrateur caractérise et indexe ces prototypes. De plus, les prototypes sont incorporés dans une surcouche logicielle leur ajoutant les propriétés nécessaires à leur utilisation dans l'atelier. Un prototype ainsi modifié est appelé un outil. L'indexation des outils est assurée via la fonction «*indexer les outils*». L'atelier maintient une base d'informations via un «**Index**». Cette dernière n'a pas besoin d'être très élaborée[36]. Nous avons implanté une telle base d'informations (section 4.9) pour valider notre maquette d'atelier.

Réaliser une activité avec l'atelier

Le sujet réalise l'activité (cas d'utilisation : «*réaliser l'activité*»). Pendant la réalisation de l'activité, l'atelier prend en charge tous les aspects techniques nécessaires, énoncés ci-dessous.

[36]Nos choix conceptuels ici sont classiques, par conséquent ils ne sont pas présentés dans ce chapitre.

- la communication entre l'atelier et les outils (fonction «*assurer la communication entre les outils*») : la prise en charge de la communication est assurée par l'architecture choisie (section 3.2) ;
- l'action sur les outils. Il le fait via la fonction «*commander les outils*» : *cf.* section 3.5 ;
- la gestion de la présentation des informations et outils pertinents pour le sujet. C'est la fonction «*gérer les interfaces graphiques*» présentée au chapitre suivant ;
- la circulation des données dans l'atelier et entre les prototypes. L'atelier veille à fournir les données au format adéquat à leur destinataire. C'est la fonction «*gérer les différents formats de données dans l'échange des connaissances*» (section 3.6) ;
- le recueil des données produites, qu'il faut stocker afin de pouvoir les exploiter. C'est la fonction «*recueillir des données*» (section 3.4).

Dans ce chapitre, nous nous plaçons au niveau conceptuel des propositions. L'implantation de celles-ci est présentée dans le chapitre 4.

3.2 Architecture de l'atelier

Pour assurer la coopération des prototypes, il faut assurer la connection des prototypes entre eux et avec l'atelier. En théorie, prototypes et atelier peuvent être localisés sur des machines différents sur des sites différents. Nous avons donc choisi une architecture distribuée de type client-serveur dont la modularité est assurée par des composants.

La figure 1.3 p 13 a introduit quatre types de composants :
- les prototypes a partir desquels l'activité est menée ;
- le gestionnaire d'expérimentations qui constitue une sorte de super-prototype «cumulant» les fonctionnalités pédagogiques des prototypes ;
- les composants de services auxquels gestionnaire d'expérimentations fait appel quand cela est nécessaire ;
- et un espace de communication, qui permet de relier les trois types de composants précédents.

L'idée de base est que le gestionnaire d'expérimentations est le client des prototypes et des composants de service. Il fait appel aux services offerts par les prototypes pour gérer une activité. De plus, les prototypes peuvent avoir besoin d'interagir entre eux. Par conséquent une architecture client-serveur de base devient rapidement difficile à mettre en œuvre quand le nombre des prototypes augmente. En effet, chaque prototype est connecté à tous les autres, qui a leur tour lui sont connectés, soit $n(n-1)$ connections[37], soit de l'ordre de n^2.
La solution adoptée pour réduire cette complexité est d'utiliser un médiateur* ou *middleware* [Wiederhold 92] (figure 3.1). Un médiateur est un équipement qui sert d'intermédiaire entre des clients et des serveurs. Le médiateur introduit un niveau logiciel entre les applications et le réseau. Cet intermédiaire simplifie la gestion de la connectivité. Chaque prototype est connecté au médiateur, qui en retour leur est connecté, soit $2n$ connections pour n prototypes. Ce nombre de connections est plus acceptable en terme de complexité.
Le médiateur prend en charge les échanges entre le client et le serveur. Il assure en particulier

- le transport ;
- la sécurité ;
- la synchronisation.

[37] $2 * [n + (n-1) + \cdots + 1] = 2 * [n(n-1)/2]$

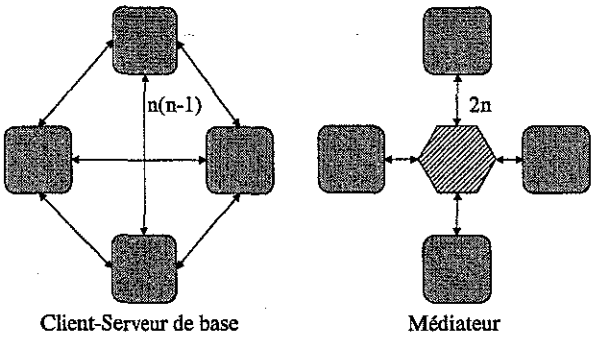


Figure 3.1 – Architecture client-serveur avec et sans médiateur

Nous ne nous préoccupons pas des mécanismes de transport, de sécurité et de synchronisation de bas niveau. Le choix d'un médiateur nous permet de déléguer ces aspects. De plus, les médiateurs existants assurent l'indépendance par rapport aux plate-formes matérielles et aux produits. Ils sont donc tout a fait adaptés à nos besoins. Nous verrons dans la section 3.2.1 quels sont les différents types de médiateurs et celui que nous retenons.

L'architecture que nous proposons est donc une architecture distribuée de type client-serveur utilisant un médiateur. Nous choisissons de définir un médiateur éducatif appelé EduMed (Educational Mediator) regroupant le médiateur proprement dit et les composants de service dont nous avons besoins pour le gestionnaire d'expérimentations puisse exploiter les prototypes. D'où l'architecture présentée à la figure 1.1 p 10. Nous la reprenons dans la figure 3.2, plus précise où le gestionnaire d'expérimentations est généralisé en gestionnaire d'activités, et où l'adaptation des prototypes pré-existants P_i est signalée par un rectangle. Ce rectangle est dans le même

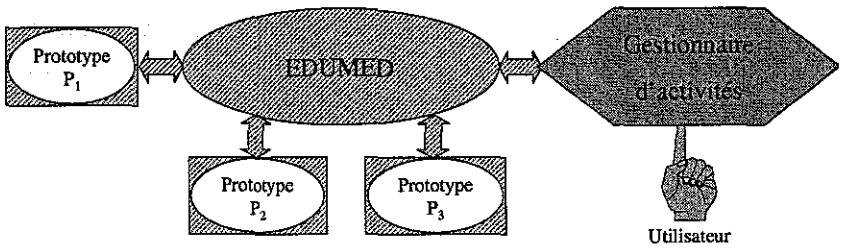


Figure 3.2 – Architecture de l'atelier

motif qu'EduMed car l'adaptation des prototypes dépend des choix faits pour l'implantation d'EduMed.

Ce médiateur assure la communication au sein de l'atelier.

3.2.1 Quel médiateur choisir ?

Il y a quatre types principaux de médiateur.

Les premiers (nommés type 1 par la suite) sont des médiateurs de bases de données (*database* middleware [Wiederhold 92, Roncancio 94]). Ils fournissent des accès transparents à des bases de données relationnelles à travers des environnements et des protocoles hétérogènes. Un exemple est «*Oracle SQL connect*».

Les seconds (nommés type 2) sont des médiateurs basés sur des RPC (*Remote Procedure Call*).

Ce sont les plus employés. Ils permettent la programmation distribuée avec des appels de procédures situées ailleurs sur le réseau. DCE (*Distributed Computing Environment*) est un standard pour ce type d'application. Ils sont utilisés, par exemple, dans ONC RPC de SunSoft.
Ceux du troisième type (nommés type 3) sont des médiateurs à base de messages (*messaging Middleware* [Wiederhold 92]). Ils font communiquer des processus par l'envoi de messages. Ces derniers sont utilisés, par exemple, dans Tooltalk de SunSoft.
Ceux du quatrième type (nommés type 4) sont des médiateurs basés sur des bus objets (*ORB-based Middleware* [Wiederhold 92]). Parmi les applications, les applications orientées objets choisissent principalement ceux-là. Un bus objet ou ORB (Object Request Broker) manipule des objets. Un standard pour les applications basées sur un ORB est CORBA* (Common Object Request Broker Architecture) [Orfali 97]. CORBA spécifie l'interface entre les applications et l'ORB, de telle manière qu'il facilite l'émission de requêtes vers les autres applications (considérées comme des objets ou plutôt comme des composants) et la réception des réponses des autres applications. Les produits conforme à la norme CORBA sont, par exemple, les objets SOM d'IBM et Orbix d'Iona.

Le type 1 ne correspond pas à nos objectifs. Les types 2 et 3 nécessitent beaucoup trop de modifications de toutes les applications. Le type 4 rend possible la conception d'un atelier au niveau application et la réutilisation de prototypes existants. C'est ce type de médiateur que nous retenons. Concernant le standard, notre choix s'est porté sur les ORB conformes à la norme CORBA. En effet, COM/DCOM est un autre standard pour ce type de médiateur avec le paradigme objet, mais principalement avec les applications Microsoft© (produits fermés). Se limiter aux applications Microsoft© est trop restrictif, d'autant que des implantations gratuites d'ORB conformes à la norme CORBA existent. Par exemple, JacORB [JacORB http] est une implantation gratuite que nous avons choisi de déployer.

Les caractéristiques de CORBA, que nous évoquons à la section qui suit, ont orienté notre choix.

3.2.2 Caractéristiques de CORBA

CORBA est une norme pour définir une architecture distribuée de type client/serveur ainsi qu'un certain nombre de services associés. Nous décrivons dans les sections suivantes deux éléments centraux (ORB et IDL) de la technologie CORBA et l'usage que nous en faisons.

ORB

Un ORB est un bus d'objets répartis qui offre un support d'exécution masquant les couches techniques d'un systèmes réparti (système d'exploitation, processeur et réseau). Il prend en charge les communications entre les composants logiciels formant les applications réparties hétérogènes. Le bus CORBA propose un modèle orienté objet client/serveur d'abstraction et de coopération entre les applications réparties. Chaque application peut exporter certaines de ses fonctionnalités (services) sous la forme d'objets CORBA : c'est la composante d'abstraction (structuration) de ce modèle. Les interactions entre les applications sont alors matérialisées par des invocations à distance des méthodes des objets : c'est la partie coopération. La notion client/serveur intervient uniquement lors de l'utilisation d'un objet : l'application implantant l'objet est le serveur, l'application utilisant l'objet est le client. Bien entendu, une application peut tout à fait être à la fois cliente et serveur.

Dans notre atelier, les applications qui doivent exporter des fonctionnalités sous forme d'objet CORBA sont principalement les prototypes. Par conséquent, chaque prototype est encapsulé

dans un objet CORBA. De plus, nous avons montré qu'un service de formatage des connaissances et qu'un service de gestion des interfaces graphiques sont nécessaires. Ces services donnent lieu à deux composants de services implantés dans des objets CORBA. Nous avons par conséquent plusieurs applications implantant les «objets services» : une application par prototype, une pour le gestionnaire de formats et une pour le gestionnaire d'interfaces graphiques. Toutes ces applications sont implantées en tant que serveurs CORBA.
Dans notre atelier l'application cliente est le gestionnaire d'activités.

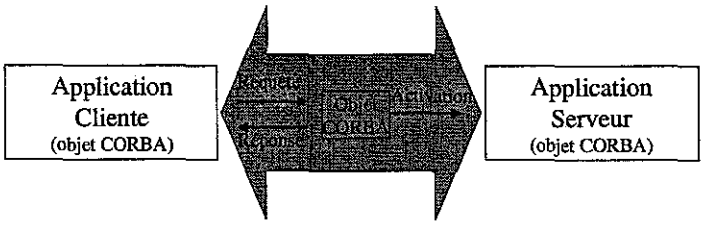


Figure 3.3 – Le modèle objet client/serveur CORBA

La figure 3.3 présente les différentes notions intervenant dans ce modèle objet client/serveur. L'**application cliente** invoque les méthodes objets à travers le bus CORBA. Le **bus CORBA** achemine les requêtes de l'application cliente vers l'objet. La **requête** est le mécanisme d'invocation d'une opération ou d'accès à un attribut de l'objet. L'**activation** est le processus d'association d'un objet d'implantation à un objet CORBA. L'**application serveur** est la structure d'accueil des objets d'implantation et des exécutions des opérations de l'objet CORBA. Chacune de ces notions se traduit par des composantes technologiques fournies par l'implantation de la norme CORBA. En résumé, l'ORB assure la communication entre la partie client et la partie serveur.

IDL

L'ORB achemine les requêtes. Mais il faut pouvoir être compris des différentes parties en présence : le client et le serveur. Pour cela, CORBA définit un langage standard commun à toutes les parties pour exprimer les interfaces de tous les objets CORBA à travers lesquelles l'ORB accède aux composants. Ce langage est appelé IDL* (pour *Interface Description Language*). C'est un langage de spécification indépendant du langage d'implantation. La syntaxe et sémantique complètes d'IDL sont disponibles dans le chapitre 3 de la spécification de l'OMG, sur le site de l'OMG [OMG http].

La technologie CORBA permet d'utiliser différents langages de développement. Les composants peuvent être écrits dans tout langage qui implante les bindings CORBA. Cela signifie à la fois qu'une correspondance (*binding*) IDL–langage est adoptée et que le compilateur intègre les outils nécessaires pour que le composant soit utilisé par un ORB. C'est le cas des langages Ada, C, C++, COBOL orienté objet, JAVA et SmallTalk. Par conséquent, une fois l'interface d'un serveur défini, nous sommes libres de changer le code (ou le langage d'implantation, si la correspondance IDL–langage existe) tant que la spécification des services offerts reste la même. De plus, si les langages actuels deviennent obsolètes, nous pourrons toujours utiliser les composants existants tant que leur langage est conforme à la norme CORBA et développer de nouveaux composants dans le dernier langage «à la mode», si une correspondance IDL–langage est adoptée.

Ainsi, CORBA rend possible la publication (c'est-à-dire le fait de rendre public) des attributs et des méthodes des serveurs de façon à ce qu'ils soient accessibles par les autres composants.

Cette publication est rendue possible via une interface, que nous appelons **vitrine** (*cf.* page 11). Dans sa vitrine, un composant affiche ses services (opérations/méthodes, exceptions et attributs) en masquant les divers problèmes liés à l'interopérabilité, l'hétérogénéité et la localisation de ceux-ci. Toute opération a une signature qui définit son nom, ses paramètres, ses résultats et ses exceptions. La vitrine ne comprend pas l'implantation des opérations ; en effet, IDL n'est qu'un langage pour définir les interfaces. Ainsi, la vitrine va permettre d'afficher ce que fait le composant indépendamment de son implantation. L'utilisation de CORBA assure ainsi une certaine flexibilité.

Projection des vitrines

La traduction d'une vitrine, exprimée en IDL, dans un langage d'implantation s'appelle une projection. En pratique, les vitrines sont projetés d'une part en souches[38] IDL (ou interface d'invocation statique SII) dans l'environnement de programmation du client et et d'autre part en squelettes[39] IDL (ou interface de squelettes statiques SSI) dans l'environnement de programmation du serveur (voir figure 3.4).

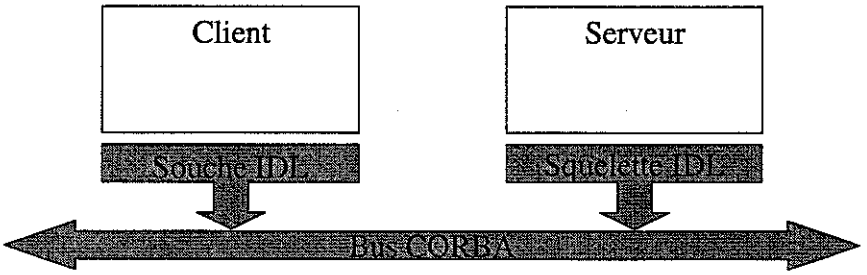


Figure 3.4 – IDL client/serveur CORBA

Le client invoque localement les souches pour accéder aux objets. Les souches IDL construisent des requêtes, qui vont être transportées par l'ORB, puis délivrées par celui-ci aux squelettes IDL qui les délégueront aux objets. Ainsi le langage IDL est la clé de voûte du bus d'objet répartis CORBA. En résumé, CORBA nous permet de faire communiquer des systèmes hétérogènes avec un langage commun.

3.2.3 Description de l'architecture

Notre motivation suit une logique de standardisation que nous trouvons dans les développements actuels en génie logiciel. Dans cette section, nous synthétisons les solutions proposées précédemment. Cette synthèse est présentée sur la figure 3.5.
En haut de la figure 3.5, nous trouvons le gestionnaire d'activités (en grisé). Il est le «chef d'orchestre» qui dirige le comportement d'EduMed (hachuré) et des prototypes Pi (en blanc) plus bas encore. Il définit leurs rôles dans la coopération. En particulier, il appelle les composants de services — par exemple, le gestionnaire de formats— ou les prototypes (en bas) en passant par l'ORB (au centre).
Dans EduMed, nous retrouvons le gestionnaire de formats et le gestionnaire d'interfaces graphiques. Ces deux gestionnaires sont ici des composants de services autour de l'ORB (placés au

[38] *stub*
[39] *skeleton*

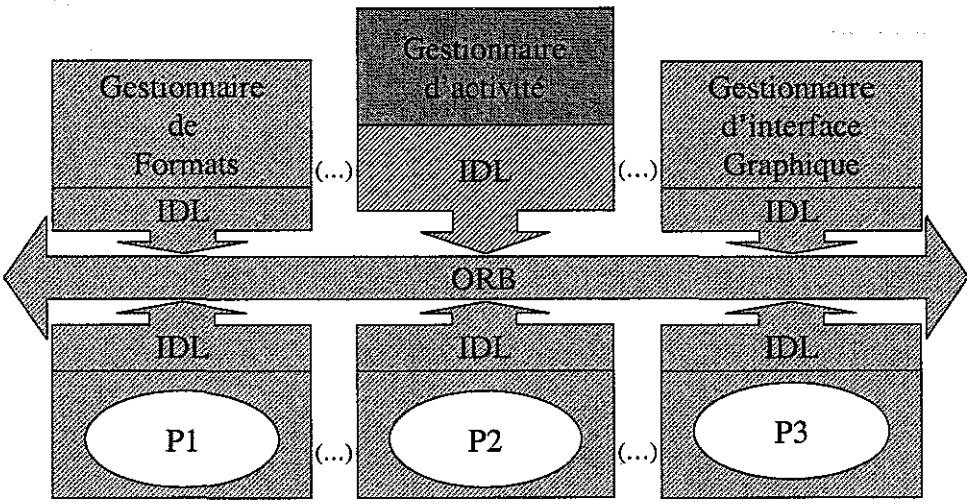


Figure 3.5 – Architecture d'EduMed

dessus de l'ORB). Nous expliquons plus loin en quoi l'ORB et les vitrines définissent un langage de commandes. L'ORB convoie les requêtes et les réponses. Il remplit ainsi le rôle d'interface de communication. Les vitrines permettent à un prototype de publier ses fonctionnalités et ses connaissances. Elles servent aussi de références pour le mécanisme d'exécution des requêtes et le retour des réponses. C'est la vitrine qui permet au gestionnaire d'activités de prendre note des fonctionnalités et des connaissances du prototype.
Nous retrouvons aussi les prototypes ($P1$, $P2$ et $P3$). Ceux-ci sont incorporés dans une surcouche (partie hachurée) permettant leur utilisation par EduMed. L'ensemble constitué du prototype et de sa surcouche est appelé outil (*cf.* page 14).

Dans cette architecture, le gestionnaire d'activités est client des autres composants. En revanche, les autres composants sont des serveurs et éventuellement des clients les uns par rapport aux autres. Le gestionnaire d'activités est le composant le moins «intelligent» de l'atelier. En effet, pour fonctionner, il fait appel aux services offerts par les autres composants. En particulier, il demande les connaissances et les fonctionnalités des autres composants. Seul, il ne peut rien faire. Il n'a pas d'autonomie.
Les fonctions que prend en charge le gestionnaire d'activités sont décrites par diagramme de cas d'utilisation «Définition d'une activité» (*cf.* figure 3.6). Définir une activité comprend 2 cas

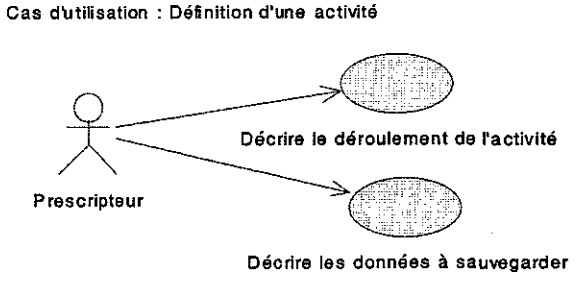


Figure 3.6 – Diagramme de cas d'utilisation : définition d'une activité

d'utilisation :

- décrire le déroulement de l'activité — Ce cas est décrit à la section 3.3 ;

– décrire les données à sauvegarder — Ce cas est décrit à la section section 3.4.

L'architecture que nous proposons ici, nous permet d'avoir un atelier évolutif. En effet, au fur et à mesure que de nouveaux prototypes ou de nouveaux composants de services apparaissent ou disparaissent, l'atelier continue d'exister.

3.3 Définir le déroulement d'une activité

Nous commençons par quelques définitions (section 3.3.1). Ces définitions sont suivies par une formalisation (section 3.3.2) qui permet de déboucher sur la description d'une activité (section 3.3.3).

3.3.1 Définitions

Pour définir le déroulement de l'activité, nous proposons d'utiliser un scénario[40] d'activité, que nous appelons **scénario*** par la suite.
Un scénario peut être découpé en «tronçons» correspondants chacun à une **tâche*** (aussi) élémentaire (que possible) à exécuter pendant une activité. Nous appelons, un tel tronçon, une **étape***. La tâche à exécuter au cours d'une étape est décrite par une **consigne*** donnée au sujet. Elle est exécuté avec un **outil***, c'est-à-dire une fonctionnalité* donnée d'un prototype* donné. Nous appelons **transition*** la condition qui autorise le passage d'une étape à une autre. Un scénario comporte au moins une étape.

Correspondance tâche-étape

Nous faisons correspondre des étapes de granularité plus ou moins fines aux tâches à exécuter. Cette correspondance dépend des fonctionnalités offertes par les prototypes disponibles. L'exemple 3.1 donne une décomposition d'un scénario en étapes. Il est illustré à la figure 3.7.

[40]Déroulement programmé ou prévu d'une action (Petit Larousse Illustré, 1994).

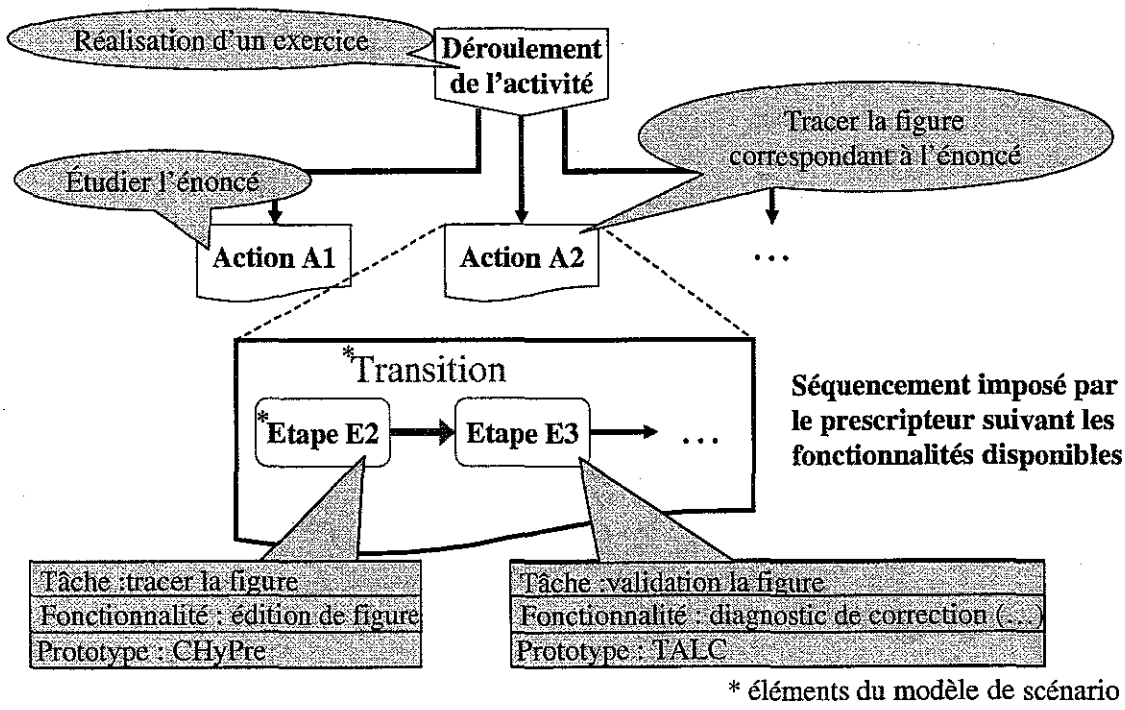


Figure 3.7 – Exemple de déroulement d'une activité

Exemple 3.1 *Dans l'exemple introductif (page 5), il est demandé au sujet de démontrer une propriété à partir d'un énoncé définissant des objets géométriques et des propriétés données en hypothèse. Dans ce cas, indépendamment de ce que le prescripteur souhaite tirer de l'activité, nous identifions différentes actions attendues du sujet :*

Action A1 : étudier l'énoncé ;

Action A2 : tracer la figure correspondant à l'énoncé ;

Action A3 : chercher à démontrer la propriété demandée ;

Action A4 : rédiger la démonstration trouvée.

Ces quatre actions peuvent être supportées par différents prototypes. De plus la validation des différents actions peut être effectuée par un humain ou par un prototype. Il est possible de faire correspondre des étapes de granularité plus ou moins fine à ces actions et leurs validations, suivant les fonctionnalités offertes par les logiciels disponibles. Par exemple, un scénario d'activité possible consisterait en les étapes suivantes :

Tâche de l'étape E1 : étudier l'énoncé ;

Tâche de l'étape E2 : tracer la figure correspondant à l'énoncé ;

Tâche de l'étape E3 : valider la figure (c'est-à-dire vérifier sa correction vis-à-vis de l'énoncé).

Chaque étape est supportée par un prototype particulier. Nous décidons, par exemple, d'étudier l'énoncé via Mentoniezh *(étape E1), de tracer la figure via* CHyPre *(étape E2) et de valider la figure via* TALC *(étape E3).*
La correspondance entre les actions et les étapes s'établie de la façon suivante. À l'action A1 correspond l'étape E1. Il n'y a pas de validation par un prototype de cette action. À l'action A2 correspond l'étape E2. Cette action est validée par un prototype lors de l'étape E3. Enfin actions A3 et A4 ne sont pas prises en charge avec l'atelier dans cet exemple.

Nous voyons que la définition d'une étape repose principalement sur l'élaboration de la tâche à faire exécuter par le sujet (et donc d'une consigne à lui donner). Cette tâche dépend des fonctionnalités offertes par les prototypes.

La formalisation à la section suivante, nous permet de mettre en évidence les caractéristiques d'une étape.

3.3.2 Formalisation

Étape

Le nombre d'étapes est noté n_E. Une étape E_i correspond à une tâche. La description de cette tâche correspond à une consigne C_i donnée à celui qui réalise l'étape. Cette tâche est à réaliser avec un outil O_i (*cf.* formalisation page 26). D'où, une étape est fonction d'une consigne et d'un outil.

$\exists n_E \in \mathbb{N}, \forall i \in \{1, \ldots, n_E\}$,

$$\mathbf{E_i = f(C_i, O_i)}$$

Nous notons

$$\mathbf{E_i(C_i, O_i)}$$

L'outil O_i est défini par une fonctionnalité F_j d'un prototype P_k (c'est-à-dire $P_k.F_j$). D'où, une étape est fonction d'une consigne, d'une fonctionnalité et d'un prototype.
La fonctionnalité $P_k.F_j$ est une fonction qui transforme des entrées in_j en sortie out_j :
$P_k.F_j : in_j \longmapsto out_j$
Une étape transforme donc des entrées in_j en sortie out_j.

$\exists n_E \in \mathbb{N}, \forall i \in \{1, \ldots, n_E\}, \exists k \in \{1, \ldots, N_{proto}\}$ et $j \in \{1, \ldots, N_{P_k}\}$ tel que

$$\mathbf{E_i = f(C_i, F_j, P_k)}$$

avec $P_k.F_j : in_j \longmapsto out_j$

$$\mathbf{E_i : in_j \xrightarrow{P_k.F_j} out_j}$$

or $O_i = P_k.F_j$

donc $E_i : in_i \xrightarrow{O_i} out_i$

Exemple : L'étape E_1 correspond à la consigne C_1 et à l'outil O_1.
L'outil O_1 est défini par l'accès à la fonctionnalité F_1 du prototype P_9. Alors $O_1 = P_9.F_1$
D'où $E_1(C_1, O_1)$. Ou encore $E_1(C_1, F_1, P_9)$.

Quand décider qu'une étape $E_i(C_i, O_i)$ est terminée ? *Une étape est terminée quand la tâche pour laquelle elle a été conçue est terminée.* La tâche est terminée quand la consigne C_i est réalisée en utilisant l'outil O_i. Comment l'atelier peut-il détecter que l'étape est finie ? Quand une condition est réalisée. La transition permet d'exprimer cette condition.

Transition

La terminaison d'une étape E_i est déterminée par la réalisation d'une condition appelée transition T_i.
Ici, il a deux cas.

1. **Transition interne (à l'outil)** : l'outil O_i est à l'initiative de la terminaison ;
2. **Transition externe (à l'outil)** : l'utilisateur est à l'initiative de la terminaison.

Le cas 1 se présente, par exemple, lorsque $O_i = P_i.F_{10}$. La fonctionnalité est «[$\mathbf{F_{10}}$.] diagnostic de besoin d'aide». Ici, lorsque le diagnostic est donné, l'outil n'a plus rien à faire. Dans ce cas, c'est un état donné de l'outil qui détermine la terminaison. Nous disons que la transition est **interne** à l'outil : elle dépend d'un événement provenant de l'outil : ici, un événement qui pourrait s'appeler «RéponseFournie») ou bien guette le passage de l'outil dans un état particulier (ici «état = diagnosticTerminé»).

Le cas 2 se présente, par exemple, lorsque $O_i = P_i.F_{17}$. La fonctionnalité est «[$\mathbf{F_{17}}$.] édition de figure géométrique». Ici, l'utilisateur est libre d'éditer la figure. La fonctionnalité consiste en une boucle sans fin, dans laquelle l'utilisateur peut éditer une figure géométrique. Dans ce cas, l'outil ne peut décider seul que l'édition est terminée. C'est à l'utilisateur de signaler qu'il considère que sa tâche avec cet outil est terminée. Nous disons que la transition est **externe** à l'outil : elle dépend d'un événement dont l'utilisateur est l'initiateur. Notre choix est de dicter à l'utilisateur un comportement qui produira cet événement. Ainsi la transition dépend d'un événement produit par un comportement de l'utilisateur : ici, un événement qui pourrait être un «clic» sur un «bouton» appelé «J'ai fini» ou bien le passage de l'outil dans un état particulier (ici «état = fermé»).

Une transition est donc fortement liée à l'étape à laquelle elle correspond, et en particulier à la consigne et à l'outil de cette étape.

$\forall i\ E_i(C_i, O_i)$, $\exists T_i$ tel que

$$\mathbf{T_i(C_i, O_i)}$$

Exemple : À l'étape $E_1(C_1, O_1)$ correspond la transition $T_1(C_1, O_1)$.

Fermeture d'un outil

Lorsque la tâche pour laquelle une étape a été conçue est terminée, la condition de terminaison de cette étape est réalisée. L'étape peut être fermée. Or pour cette étape, un outil a été lancé. Que faut-il faire de cet outil ? Doit-on le fermer en même temps que l'étape ?

La terminaison d'une étape ne correspond pas forcément la fermeture d'un outil. En effet, pendant l'étape E_i, l'utilisateur a utilisé l'outil O_i pour réaliser une tâche en suivant une consigne C_i. La tâche étant réalisée, l'outil O_i peut rester à la disposition de l'utilisateur.

Le cas se présente, par exemple, lorsque $O_i = P_i.F_{17}$. La fonctionnalité est «[$\mathbf{F_{17}}$.] édition de figure géométrique». Ici, si nous fermons l'outil O_i en même temps que l'étape, l'utilisateur ne peut plus consulter sa figure par la suite. En revanche, si nous laissons l'outil O_i tourner[41], l'utilisateur peut consulter sa figure après la fin de l'étape.

Ainsi nous avons identifié que l'outil O_i peut être fermé

- à la fin de l'étape courante ;
- à un autre moment.

Dans le cas de la fermeture à un autre moment, l'outil continue à tourner. Il reçoit éventuellement d'autres paramètres. La fermeture est alors prévue, à la fin d'une étape postérieure à l'étape courante ou à la fin du scénario.

Scénario

Un scénario est un ensemble structuré d'étapes.

[41] Avec un paramétrage adapté de cet outil O_i, on peut interdire la modification de la figure dès la fin de l'étape, ou autoriser uniquement la manipulation de la figure ([$\mathbf{F_{14}}$.] exploration visuelle de figure géométrique) si le prototype utilisé implante cette fonctionnalité

Point de vue du sujet Du point de vue du sujet, un scénario S est cet ensemble structuré se limite à une suite ordonnée de n_E ($n_E \in \mathbb{N}$) étapes E_i.

Représentation graphique Le sujet réalise d'abord la première étape, puis la seconde, *etc.* (*cf.* figure 3.8). Sur cette figure, nous représentons la succession des étapes sous forme la forme

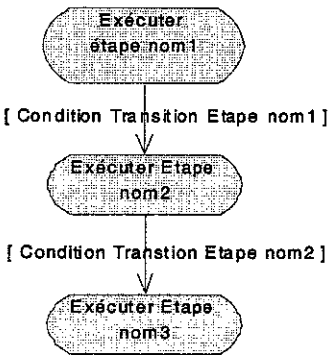


Figure 3.8 – Exemple de scénario du point de vue du sujet

d'un graphe. Pour utiliser un graphe, il faut définir le rôle des noeuds et des des arcs. Les deux possibilités sont :

1. Noeud : étape ; Arc : transition ;
2. Noeud : transition ; Arc : étape.

Nous avons choisit la première solution. De plus, nous avons utilisé les conventions UML :

Élément du scénario	Élément UML	Représentation UML
Étape	Activité UML	Rectangle aux coins arrondis
Transition	Transition gardée UML	Flèche

(Une transition gardée est franchie si la condition entre crochets est remplie.)

Sur la figure 3.8, la première étape (appelée «Exécuter Etape nom 1») est exécutée (par l'atelier). Le sujet réalise alors la tâche décrite par la consigne de cette étape. Quand la condition de terminaison de la première étape (appelée «Condition Terminaison Etape nom 1» est réalisée, la transition est franchie. L'étape suivante est exécutée.

Point de vue du prescripteur Le point de vue du prescripteur est plus complexe. En effet, dans une activité, lors d'une étape réalisée par le sujet des étapes peuvent être exécutées en parallèle.

Représentation graphique C'est le cas dans l'exemple introductif, où *PACT* est lancé en parallèle de *CABRI-Géomètre* et de *CHyPre* pour fournir une aide dans l'interaction au sujet (*cf.* figure 3.9).

Sur cette figure, nous ne faisons plus apparaître les gardes des transitions. Nous avons visualisé les étapes qui se déroulent en parallèle au moyen de barres de synchronisation (lignes horizontales épaisses, sur lesquelles arrivent et partent des flèches). Les transitions au départ d'une barre de synchronisation sont déclenchées simultanément. C'est le cas par exemple au début du scénario. Il commence par une barre de synchronisation. Les transitions de cette barre sont déclenchées en même temps. En conséquence, les étapes «construit une figure» et «analyse des interactions» sont lancées en parallèle par l'atelier. La première est dans le «couloir» de gauche. C'est le sujet

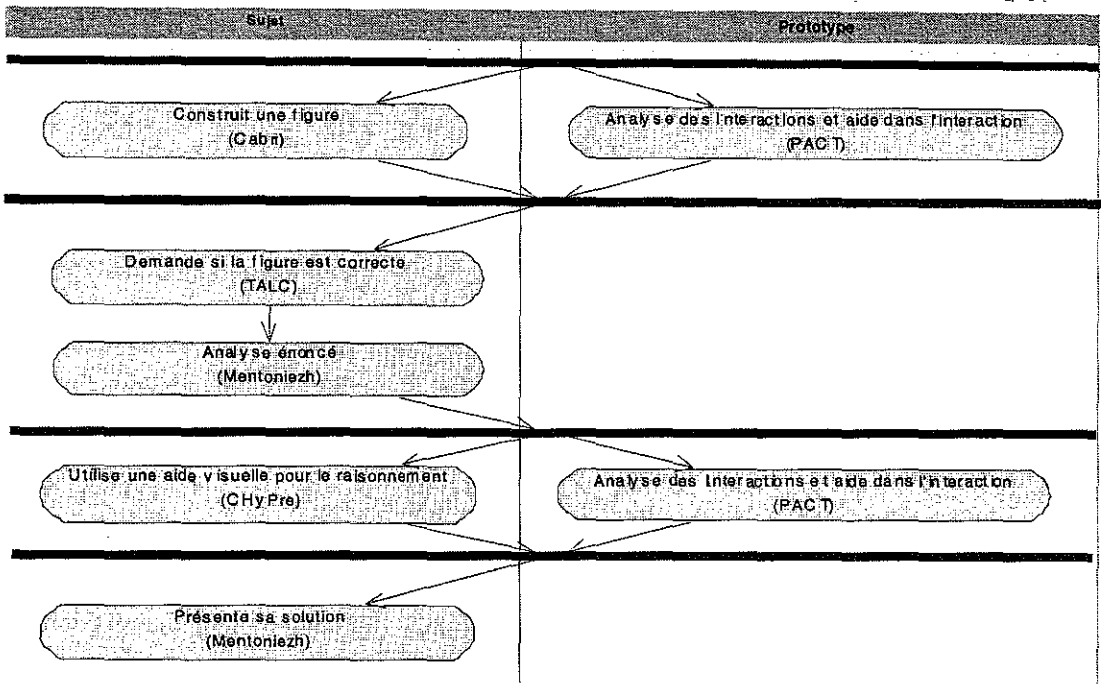


Figure 3.9 – Graphe du scénario de l'exemple introductif

qui la réalise. La seconde est dans le «couloir» de droite. C'est le prototype qui la réalise. Une barre de synchronisation est franchie lorsque toutes les transitions en entrée sur la barre ont été déclenchées. C'est le cas, quand les étapes «construit une figure» et «analyse des interactions» sont toutes les deux terminées.

Quand décider qu'un scénario est terminé ? Un scénario est terminé quand la dernière étape est terminée (ou la dernière barre de synchronisation est franchie). Pour être sûr qu'un scénario se termine, il faut s'assurer que

- chaque étape se termine.
- le scénario ne boucle pas ;

La terminaison d'une étape dépend du sujet. C'est lui qui réalise la tâche lorsque l'atelier exécute l'étape. La tâche a un début et une fin. Donc l'étape a un début et une fin. La fin de la tâche est signalée à l'atelier par la réalisation d'une condition. L'atelier peut aussi ajouter une contrainte supplémentaire pour déclencher à coup sûr la condition (par exemple, un délai au bout duquel l'étape est clôturée. Mais est-ce bien utile ? Et quel intérêt cela peut-il avoir ? Cela dépend de l'activité menée avec l'atelier.

Le scénario est défini comme une succession d'étapes destinées au sujet, sur lesquelles se greffent des étapes supplémentaires (pour apporter une aide par exemple). Cette succession d'étapes destinées au sujet constitue l'activité que le prescripteur définit pour lui. Cette activité a un début et une fin. Donc le scénario a un début et une fin. Cependant, le prescripteur peut commettre une erreur en saisissant les étapes de l'activité qu'il a conçu dans l'atelier. C'est pourquoi, la détection des boucles dans le graphe de l'activité est nécessaire.

La fin du scénario correspond à la fin de l'activité et à la fermeture de tous les outils encore actifs.

3.3.3 Décrire une activité

Décrire une activité, c'est décrire son déroulement (via un scénario) et décrire les données à collecter (*cf.* section suivante).

Décrire un scénario

Un scénario est décrit par un graphe dont les noeuds sont les étapes et les arcs sont les transitions gardées. Des barres de synchronisation permettent d'attendre que toutes les transitions des étapes en cours d'exécution soient franchies pour lancer de nouvelles étapes simultanément. Une transition gardée permet d'exprimer à la fois la condition pour passer d'une étape à une autre et un choix entre plusieurs étapes comme successeur de l'étape qui se termine. Les barres de synchronisation, quant à elles, permettent d'exprimer le parallélisme des étapes. Le graphe permet d'exprimer des scénarios complexes. Créer un scénario, c'est créer les différentes étapes qui le compose.

Description d'une étape

La formalisation précédente introduit les informations nécessaires à la description d'une étape :

- une consigne ;
- un outil ;
- la fermeture de l'outil ;
- les flux de données en entrée (au moins un) et en sortie (au moins un) de l'étape (donc de l'outil). Un flux vient d'un outil et va vers un outil ;
- une transition (par exemple, «c'est le sujet qui signale qu'il a fini») ;

La création d'une étape passe par la définition de ces informations.

Le scénario permet de définir la coopération entre les différents prototypes impliqués. Il constitue une proposition pour définir un protocole de coopération décrivant le déroulement de l'activité. Son implantation est présentée à la section 4.3.

Cette section permet de décrire ce qui est demandé au sujet au cours de l'activité. Cependant, les données à recueillir au cours de l'activité (les flux de données) n'ont pas été évoquées dans cette section. La section suivante remplit ce rôle.

3.4 Définir les données à sauvegarder pendant l'activité

Une activité produit des données. Parmi elles, certaines sont sauvegardables. Cette section permet de les définir (section 3.4.1) et d'étudier leur recueil. Nous introduisons ci-dessous quelques définitions pour préciser les données sauvegardables

3.4.1 Définitions

Quelles sont les données sauvegardables ?

Nous appelons **données sauvegardables** ou simplement **sauvegardables** (nom commun) les données publiques d'un outil. Elles comprennent les données auxquelles un système extérieur à l'outil (l'atelier et les autres outils) a accès. Les données sauvegardables d'un outil sont de trois types :

1. les résultats produits, c'est-à-dire ses sorties ;

2. les informations scrutables, c'est-à-dire les variables et états qu'un système extérieur à l'outil peut scruter (consulter) ;
3. les traces de l'interaction utilisateur-outil, que nous précisons ci-après.

Les paragraphes suivants introduisent des définitions pour aboutir à des traces d'interactions intelligibles.

Qu'est-ce que les traces d'interaction utilisateur-système ?

Les traces d'un programme sont des observations recueillies à propos d'un programme [Haumont 98]. La nature des traces à recueillir est de deux types :

- observation de l'exécution du programme pour suivre pas à pas le déroulement des grandes étapes du programme ;
- observation de l'utilisation de l'interface du système par l'utilisateur.

C'est à ce dernier type de traces que nous nous intéressons. Nous les appelons «traces d'interaction utilisateur-système» ou simplement «traces d'interaction». Les traces d'interaction sont des observations recueillies auprès d'un système lorsque l'utilisateur interagit avec lui. Ces traces d'interaction peuvent être recueillies par différentes techniques. Citons par exemple :

- une caméra (qui filme l'utilisateur du système ou l'écran du système) ;
- une personne (qui joue le rôle d'un scribe et relève elle-même ce qu'elle juge nécessaire) ;
- un dispositif pour récolter les mouvements de l'oeil et de la tête (désignation à l'écran) ;
- un système informatique.

Dans la suite, nous nous intéressons à cette dernière possibilité.

Pour quoi recueillir des traces d'interactions ?

L'analyse de l'activité passe par le recueil des informations concernant les interactions du sujet (de l'utilisateur, en général) avec un outil (un système, en général). Ces interactions, à l'interface du système, permettent de mémoriser les actions de l'utilisateur. Elles peuvent être utilisées pour :

- être rejouées par l'outil source de ses traces (éventuellement associées avec le logiciel-outil adapté) et ainsi observer les comportements de l'utilisateur ;
- être analysées par le prescripteur (analyse asynchrone) ou un système d'aide (analyse synchrone) et ainsi induire les processus cognitifs que l'utilisateur met en œuvre ;
- être exploitées pour fournir des rétroactions pertinentes et ainsi mieux prendre en compte l'utilisateur ;
- être exploitées pour présenter (présentation synchrone ou asynchrone) l'activité de l'utilisateur à un tiers (humain ou système) dans un but de supervision, de tutorat, de maintenance (corrective notamment) ou d'étude (ergonomie de l'interface, par exemple).

En particulier, pour effectuer cette prise en compte ou induction, le prescripteur peut récupérer le maximum d'informations pertinentes sur l'interaction sujet-outil.

Les événements d'interface

Lorsque l'utilisateur interagit avec un outil, il le fait par l'intermédiaire de l'interface fournie par le logiciel. Il utilise le clavier, la souris ou tout autre périphérique mis à sa disposition. Toutes ses actions sont traduites par ce que nous appelons des «événements à l'interface de l'utilisateur et de l'outil», ou plus simplement des **événements d'interface**. Les événements d'interface sont

Nom	Action
WM_LBUTTONDBLCLK	double clic sur bouton gauche
WM_LBUTTONDOWN	pression sur bouton gauche
WM_LBUTTONUP	lâcher du bouton gauche
WM_MBUTTONDBLCLK	double clic sur bouton du milieu
WM_MBUTTONDOWN	pression sur bouton du milieu
WM_MBUTTONUP	lâcher du bouton du milieu
WM_RBUTTONDBLCLK	double clic sur bouton droit
WM_RBUTTONDOWN	pression sur bouton droit
WM_RBUTTONUP	lâcher de bouton droit
WM_MOUSEACTIVATE	activation de la souris
WM_MOUSEMOVE	déplacement de la souris
WM_SETCURSOR	mise à jour de la position du curseur souris
WM_HSCROLL	roulement du bouton de scroll horizontal
WM_VSCROLL	roulement du bouton de scroll vertical

Table 3.1 – Événements souris

générés et gérés par le système d'exploitation. Ils constituent un sous ensemble des événements systèmes. Par exemple, les événements d'interface pour la souris et les actions correspondantes sont données dans la table 3.1.

Sauvegarder tous les événements d'interface ? Sauvegarder tous les événements d'interface produits au cours de l'interaction de l'utilisateur avec l'outil est possible. Cependant, ce type de sauvegarde génère un grand nombre de données dont seul un petit nombre est réellement exploitable. Ces données sont de granularité très fine. Par exemple, un simple clic sur le bouton gauche de la souris, génère une séquence d'événement composés de :

- WM_LBUTTONDOWN ;
- WM_SETCURSOR ;
- WM_MOUSEACTIVATE ;
- WM_RBUTTONUP.

En effet, la séquence commence par un WM_LBUTTONDOWN et se termine par un WM_RBUTTONUP. Entre les deux, suivant la durée pendant laquelle le bouton reste enfoncé, il peut y avoir plusieurs arrivées de l'événement WM_LBUTTONDOWN. De plus, pendant cette même durée, le curseur (WM_SETCURSOR) est mis à jour et activé (WM_MOUSEACTIVATE).
Il est possible de collecter tous les événements d'interface issus du clavier ou de tout autre périphérique permettant à l'utilisateur d'interagir avec l'outil. C'est le cas de l'exemple 3.2, où ce sont tous les événements générés via la souris qui sont collectées.

Exemple 3.2 *Nous utilisons pour cet exemple, le logiciel Winsight qui permet de récupérer tous les événements systèmes lors d'une interaction avec Windows. Il est lancé dans le contexte suivant : le document* `ref.doc` *est ouvert avec Winword. Winsight est configuré pour enregistrer uniquement les événements systèmes de la fenêtre correspondant à l'application Winword. Seul les événements d'interfaces relatifs à la souris sont tracés. La figure 3.10 montre un extrait de la trace sauvegardée pour la séquence suivante : clic dans la fenêtre winword. Sélection d'un mot (double clic). Clic sur la mise en forme d'un mot en gras (bouton G). Saisie d'un mot (non gras).*

```
000769:000004BC {_WwG} WM_SETCURSOR (2004X) Sent  wp=000004BC lp=02000001
000770:000004BC {_WwG} WM_MOUSEMOVE (20004X) Dispatched  wp=00000000 lp=00770031
000771:000004BC {_WwG} WM_MOUSEMOVE (20004X) Dispatched  wp=00000000 lp=00760032
000772:000004BC {_WwG} WM_SETCURSOR (2004X) Sent  wp=000004BC lp=02000001
000773:000004BC {_WwG} WM_MOUSEMOVE (20004X) Dispatched  wp=00000000 lp=00760032
000774:000004BC {_WwG} WM_LBUTTONDOWN (20104X) Dispatched  wp=00000000 lp=00760032
000775:000004BC {_WwG} WM_LBUTTONDOWN (20104X) Dispatched  wp=00000000 lp=00760032
000776:000004BC {_WwG} WM_MOUSEACTIVATE (2104X) Sent  wp=00000E5C lp=02010001
000777:000004BC {_WwG} WM_SETCURSOR (2004X) Sent  wp=000004BC lp=02010001
```

Figure 3.10 – Extrait de la trace des événements d'interfaces lors d'une interaction avec Winword

La séquence de l'exemple 3.2 dure moins de 2 secondes. Elle génère 96 événements d'interfaces de six types : `WM_MOUSEMOVE` (53), `WM_SETCURSOR`, (30) `WM_LBUTONNDBCLK`, (5) `WM_LBUTTONUP` (4) `WM_MOUSEACTIVATE` (2) et `WM_BUTTONDOWN` (2). Ici aucun des mouvements de souris (`WM_MOUSEMOVE`) n'est significatif. De plus, l'interprétation des autres événements n'est pas immédiate. En effet, il faut interpréter les informations de chaque ligne. Pour la première ligne, par exemple, 000769 correspond au numéro de l'événement, $000004BC$ et $wp = 00004BC$ correspond au numéro de la fenêtre. $\{\_WwG\}$ correspond à l'identifiant de la fenêtre ($2004X$) correspond au code hexadécimal de l'événement `WM_SETCURSOR Sent`. Enfin, $lp = 0200001$ correspond à la valeur hexadécimal de la position du curseur.
Une première façon de rendre lisible cette trace consiste à en fournir une autre présentation. Par exemple, la forme «mise à jour du curseur en (x, y)» est déjà plus lisible.

Sauvegarder seulement certains événements d'interface ? Sauvegarder seulement certains événements d'interface réduit le nombre de traces collectées. Cependant, ce n'est pas non plus satisfaisant, car les traces demeurent peu intelligibles. En effet, supposons que nous proposions de sauvegarder tous les doubles-clics de souris. Nous recueillons alors une liste de doubles-clics ainsi que les coordonnées de ces doubles-clics sur l'écran.
Prenons l'exemple de la trace suivante : «double clic gauche en $(368, 64)$». Il n'est possible d'interpréter cette trace, que si «l'objet» présent à cette position sur l'écran est connu (c'est-à-dire ce sur quoi l'utilisateur a «double-cliqué»). Il faut aussi connaître la sémantique de ce type de clic (double clic avec le bouton gauche de la souris) sur «l'objet». Or le prescripteur qui reçoit cette trace ne connaît ni l'objet présent en $(368, 64)$, ni la signification du double clic dans ce contexte.
L'exemple 3.3 illustre l'interprétation d'une séquence d'interaction avec l'outil *CHyPre*.

Exemple 3.3 *Dans* CHyPre, *un double clic gauche à l'endroit où est tracé un point, provoque la sélection de ce point (double clic gauche* $\longmapsto$ *sélection). Soit la séquence d'action suivante :*
- *déplacement de la souris jusqu'au point A.*
- *double clic gauche sur le point A*

Un extrait des traces brutes est donné à la figure 3.11.
Les événements d'interface correspondant à cette séquence sont
- *Série de* `WM_MOUSEMOVE` *et de* `WM_SETCURSOR` *jusqu'à la position* $x = 368$, $y = 64$.
- `WM_LBUTTONDBLCLK` *en* $x = 368$, $y = 64$.[a]

Cependant pour déduire qu'il y a sélection d'un point particulier, il faut aussi savoir qu'à cet endroit il y a un point.

[a] Pour simplifier, nous indiquons les coordonnées mathématiques du point, bien que le logiciel considère une zone de plusieurs pixels autour du point. Nous n'entrons pas dans le détail, ici.

```
000000 WM_SETCURSOR Sent MouseMove in Client
000001 WM_MOUSEMOVE Dispatched (486,91)
000002 WM_MOUSEMOVE Dispatched (472,93)
000003 WM_SETCURSOR Sent MouseMove in Client
000004 WM_MOUSEMOVE Dispatched (472,93)
000005 WM_MOUSEMOVE Dispatched (454,94)
...
000045 WM_SETCURSOR Sent MouseMove in Client
000046 WM_MOUSEMOVE Dispatched (368,65)
000047 WM_MOUSEMOVE Dispatched (368,64)
000048 WM_SETCURSOR Sent MouseMove in Client
...
000059 WM_SETCURSOR Sent MouseMove in Client
000060 WM_MOUSEMOVE Dispatched (368,64)
000061 WM_LBUTTONDBLCLK Dispatched (368,64)
```

Figure 3.11 – Traces pour la sélection du point A

Une fois cette trace filtrée, le prescripteur qui reçoit l'événement d'interface «double clic en $(368, 64)$» ne peut interpréter directement le comportement de l'utilisateur.
Pour donner un sens (une sémantique) à cet événement (en relation avec le comportement de l'apprenant), une solution consiste à rejouer la séquence. En pratique, cette manoeuvre est trop lourde pour être pratiquée à la main. Nous pouvons regretter que peu de logiciels offrent la possibilité de rejouer une séquence.
Une autre solution, consiste à s'appuyer sur les connaissances du logiciel. En effet, dans l'exemple 3.3, *CHyPre* «sait» qu'il y a un point à cet endroit là. La connaissance à exploiter est : «en $(368, 64)$, il y a un point nommé A». Lorsqu'il détecte le double-clic à cet endroit, il applique le traitement associé au double clic, c'est-à-dire la sélection du point. Un double clic à un autre endroit ne provoquerait pas forcément le même comportement. La sémantique de la trace «double clic en $(368, 64)$» est «sélection du point A». *CHyPre* peut alors générer un événement particulier porteur de cette sémantique.
Ainsi, le prescripteur qui reçoit l'événement porteur de la sémantique «sélection du point A», reçoit une information plus intelligible que celle portée par l'événement d'interface «double-clic en $(368, 64)$».

Les événements sémantiques

Nous appelons «**événement sémantique**» un événement porteur d'une sémantique. Cette sémantique permet d'exprimer l'effet de l'événement sur le système. Xavier Dubourg a déjà introduit ce concept dans ([Dubourg 95]) pour modéliser l'interaction en EIAO. Durand fait référence aux «traits sémantiques» ([Durand 97]) pour désigner le même concept dans un contexte multi-agents.
Pour l'exemple 3.3 (et la figure 3.11), les événements sémantiques sont donnés dans la table 3.2.

événement d'interface	**Connaissance associée**	**événement sémantique**
WM_MOUSEMOVE	néant	non pertinent
WM_SETCURSOR	néant	non pertinent
WM_LBUTTONDBLCLK $(368, 64)$	point A en $(368, 64)$	sélection du point A

Table 3.2: Événements sémantiques de l'exemple

Les événements sémantiques sont des traces plus intelligibles que les événements d'interface. Elles sont élaborées à partir des traces brutes. Autrement dit, un **événement sémantique** est une séquence ordonnée d'événements d'interfaces, épurée des événements d'interfaces dénués de signification pour l'objectif visé.
En effet, les événements sémantiques sont composés à partir des événements d'interfaces ayant une signification. L'exemple 3.4 donne des exemples d'événements d'interfaces dénués de signification.

Exemple 3.4 *Le déplacement de la souris n'a pas de signification particulière dans* CHyPre. *Il correspond au déplacement de la souris sur le plan de dessin. Par conséquent, les événements d'interface «déplacement de la souris de la position* $(x1, y1)$ *à la position* $(x2, y2)$*» sont associés à l'événement sémantique «sans signification».*
Par contre le même déplacement de souris dans un logiciel de dessin de type Paint permet (une fois l'outil adéquat sélectionné) de dessiner avec la souris (autant de points que de positions de souris parcourues à l'écran). Dans ce cas, un déplacement de la souris est associé à l'événement d'interface «déplacement de la souris à la position $(x1, y1)$*» l'événement sémantique «tracé d'un point à la position* $(x1, y1)$*».*

La séquence est ordonnée. L'exemple 3.5 illustre l'incidence de l'ordre des événements d'interface dans la fabrication d'un événement sémantique.

Exemple 3.5 *Dans le traitement de texte Word de Microsoft©, la sélection d'un mot (ou d'un bloc de texte) se fait par un double clic sur ce mot. Le déplacement de la souris dans la fenêtre d'édition n'a pas de signification. un clic sur l'icône (portant un caractère G) de mise des caractères en gras provoque deux types de comportement.*
Dans le premier cas, un mot (un bloc de texte) est sélectionné au moment où l'icône est cliqué. En conséquence, le mot (ou le bloc de texte) est mis en gras.
Dans le second cas, l'icône est cliqué alors qu'aucune partie du texte n'est sélectionné (ni mot, ni bloc de texte). En conséquence, toute saisie de caractère est mise en gras jusqu'à ce que l'icône soit de nouveau cliqué.

La définition des événements sémantiques nécessite de connaître les actions que l'utilisateur peut accomplir dans un contexte donné. L'exemple 3.3 montre qu'il est souhaitable que chaque prototype produise les événements sémantiques dont il a la maîtrise.
En effet, lorsqu'un prototype applique une traitement adéquat à une séquence d'interactions, il lui est aisé de composer l'événement sémantique associé au traitement. Dans l'exemple 3.5 il peut composer l'événement sémantique «mise en gras de tel bloc de texte sélectionné» ou «début de saisie en gras» selon le traitement qu'il met en place.

Les événements sémantiques peuvent être produits par tous les types de logiciels, et pas seulement les Logiciels Éducatifs.

Production des événements sémantiques Le recueil des traces d'interaction peut être réalisé de deux façons :

- soit le système est prévu pour une telle utilisation et incorpore donc une possibilité de tracer sa propre exécution [Després 97]
- soit le système n'est pas prévu pour une telle utilisation et c'est un autre système qui va s'en charger.

La première alternative est la plus efficace ([Carraux 99]) car elle implique que les observations sont faites au moment même de l'interaction et avec une connaissance précise du contexte dans lequel est effectuée l'interaction. Ces informations peuvent faire défaut à un programme extérieur chargé de la même besogne dans la deuxième alternative. Malheureusement, la plupart des logiciels n'incorporent pas de possibilité de traces d'interactions. En effet, cela n'a que très peu d'intérêt commercial et n'est pas envisagé dans les logiciels commerciaux. De même, pour les logiciels issus de la recherche, cette possibilité n'est pas implantée lorsque les objectifs de recherche n'impliquent pas la collecte des interactions. C'est donc, dans la plupart des cas, un programme épiphyte* qui se charge de collecter les traces. Citons par exemple les programmes

suivants : pour Unix dans le WOSIT (Widget Observation Scripting and Inspecting Tool)[42] [Cheikes 98], et pour PC dans [Haumont 98, Desmoulins 98] ou dans le système épiphyte décrit dans [Pachet 96]. Nous considérons les événements d'interfaces comme les données brutes à partir desquelles les événements sémantiques sont composés.

Par définition, un outil est traçable. Par conséquent, il produit des événements sémantiques.

Format des événements sémantiques

Format d'un événement Ritter et Koedinger pour leur tuteur PACT [Ritter 96] ont proposé un format de description des événements. Il est également utilisé par Xavier Dubourg [Dubourg 95]. Un événement est un triplet : <objet O ; action A ; paramètres P>[43] où :

- objet désigne le type de l'objet d'interface, par exemple, un point, une droite, un bouton, *etc.* (c'est un nom, éventuellement qualifié) ;
- action désigne l'action appliquée sur l'objet, par exemple, sélectionner, déplacer, actionner, *etc.* (c'est un verbe) ;
- paramètres désigne une liste de paramètres éventuels nécessaires pour préciser l'action, *etc.*.

> **Exemple 3.6** *un double clic sur un point A de coordonnées* (368, 64), *correspond à l'événement <point A (368, 64) ; double cliquer ; aucun>. En effet, l'objet sur lequel porte l'action «double cliquer» est le point A de coordonnées* (368, 64). *De plus, l'action «double cliquer» ne nécessite pas de paramètre.*

Ce format correspond à l'externalisation d'un événement. Il n'augure pas de son implantation. Il est bien adapté pour nos événements sémantiques. Nous le reprenons donc.

Format d'un événement sémantique L'action décrite dans un événement sémantique est plus précise.

> **Exemple 3.7** *Si l'événement de l'exemple 3.6 a lieu dans* CHyPre, *nous savons qu'un double clic correspond à une sélection. L'événement sémantique associé est <point A (368, 64) ; sélectionner ; aucun>*

Ici, sélectionner correspond à la sémantique du double clic sur le point A dans ce logiciel. Dans l'exemple de la figure 3.12, nous reprenons les événements d'interfaces de la figure 3.11 pour lesquels nous avons constitué les événements sémantiques correspondants. Entre < et > apparaît chaque élément du triplet.

L'interprétation de la première ligne donne «la souris à été mise à jour». Cette action «ne nécessite pas d'argument». Elle porte sur «la fenêtre de dessin». Autrement dit, elle ne porte pas sur un objet particulier de l'interface graphique.

L'interprétation de la dernière ligne donne «un double clic gauche à été effectué». Cette action «ne nécessite pas d'argument». Elle porte sur le «point A de coordonnées (368,64)».

Les observables

Parmi les événements sémantiques, certains sont pertinents pour la problématique de l'observateur (le prescripteur), en particulier en EIAO. Nicolas Balacheff nomme ce type d'événements

[42] WOSIT est devenu JOSIT depuis, pour JAVA Observation Scripting and Inspecting Tool, disponible donc sur toute plate-forme supportée par JAVA.

[43] C'est nous, qui présentons ce triplet entre < et >, avec ; comme séparateur des éléments du triplet

```
<fenêtre dessin> <mettre à jour souris> <nil>
<fenêtre dessin> <déplacer souris>   <(486,91)>
<fenêtre dessin> <déplacer souris>   <(472,93)>
<fenêtre dessin> <mettre à jour souris>    <nil>
<fenêtre dessin> <déplacer souris>   <(472,93)>
<fenêtre dessin> <déplacer souris>   <(454,94)>
...
<fenêtre dessin> <mettre à jour souris>    <nil>
<fenêtre dessin> <déplacer souris>   <(368,65)>
<fenêtre dessin> <déplacer souris>   <(368,64)>
<fenêtre dessin> <mettre à jour souris>
...
<fenêtre dessin> <mettre à jour souris>    <nil>
<fenêtre dessin> <déplacer souris>   <(368,64)>
<Point A (368,64)> <sélectionner> <nil>
```

Figure 3.12 – Exemple d'événements sémantiques

sémantiques des «**observables**» [Balacheff 94b]. Pour déterminer les observables qui sont utiles dans le calcul des interactions, il propose de distinguer le comportement (lié à cet observable) de son interprétation. C'est un travail d'interprétation comportementale sur les données produites que le prescripteur doit réaliser.

> **Exemple 3.8** *Plaçons nous dans un contexte, où le détail des déplacements d'objets lors de la manipulation d'une figure n'est pas pertinent pour le prescripteur. Dans ce contexte, une série de «déplacement d'un objet» correspond à la «manipulation de la figure». Associer aux événements sémantiques «déplacement d'un point», «déplacement d'une droite» etc. l'interprétation «manipulation de la figure» permet au prescripteur d'obtenir une trace encore plus intelligible.*

Les observables sont élaborés à partir des événements sémantiques pertinents dans un contexte éducatif (*cf.* section 3.4.4). Ils permettent de diminuer le nombre d'informations à collecter : le système sauvegarde des observables au lieu des événements d'interface.

Résumé

Résultats Nous appelons **résultat** la sortie produite par un outil au cours d'une étape.

Scrutables Nous appelons **scrutables** les informations scrutables (adjectif) d'un outil.

Observables Nous appelons **observables** les informations produites à partir des traces de l'interaction de l'utilisateur avec les outils.

3.4.2 Les résultats produits

À la fin de chaque étape du scénario, chaque outil peut produire un résultat. Ce résultat est parfois codé en dur dans l'outil (le prototype). Il peut parfois être externalisé dans un fichier. Plusieurs cas d'externalisation du résultat se présentent :

- l'outil sauvegarde automatiquement le résultat à la fermeture de l'outil dans un fichier de nom connu ;
- l'outil demande à l'utilisateur (le sujet) s'il faut sauvegarder le résultat à la fermeture de l'outil. Il utilise alors un nom par défaut à moins que l'utilisateur (le sujet) ne saisisse un nom particulier ;
- l'outil ne prend pas l'initiative de sauvegarder le résultat ou de le demander, mais l'utilisateur (le sujet) peut dans son interaction, lui demander de sauvegarder le résultat, dans un fichier de nom connu.

Dans ces trois cas, c'est l'outil qui produit le résultat à la fin de l'étape. C'est aussi l'outil qui externalise le résultat. Cependant, c'est l'utilisateur (le sujet) qui a l'initiative du nom à donner au fichier de résultat. C'est gênant, car le sujet n'a pas à gérer les sauvegardes destinées au prescripteur, même s'il peut gérer des sauvegardes personnelles pendant son activité.

Comment s'assurer de l'externalisation du résultat ?

Dans le cas normal, lorsque l'étape se termine, l'outil produit un résultat. Que le sujet ait fait des sauvegardes ou non, il faut qu'une externalisation ait lieu.
Une première solution consiste à mémoriser le nom du fichier résultat, dans une table, lorsque le sujet est à l'initiative de la sauvegarde.
Une deuxième solution consiste à dupliquer ce fichier résultat.
Mais pour ces deux solutions, que faire si le sujet ne fait pas de sauvegarde ?
Une troisième solution consiste à utiliser la scriptabilité de l'outil pour provoquer la sauvegarde, avec par exemple un nom unique construit par rapport à celui de l'étape. Cette troisième solution est indépendante du comportement du sujet. Elle est de plus assez souple pour autoriser le sujet à faire des sauvegardes s'il le souhaite. C'est celle que nous retenons.

Externalisation du résultat

L'outil produit le résultat. Il est stocké dans un fichier. L'externalisation a toujours lieu à la fin d'une étape. Le diagramme de séquence correspondant à la production du résultat est présenté à la figure 3.13.

Que devient le résultat ?

Le résultat est accessible après la fin de l'étape qui l'a produit. Le résultat peut alors être sauvegardé par l'atelier pour l'exploitation de l'activité. Le résultat est «consommé» dans trois situations :

1. il est consulté par le prescripteur— Le diagramme de séquence correspondant à la consultation du résultat par le prescripteur est présenté à la figure 3.14 ;
2. il est utilisé en entrée d'une étape — Dans le diagramme de séquence correspondant à la consultation du résultat par un outil, présenté à la figure 3.15, l'outil a un rôle identique à celui du prescripteur sur la figure 3.14 ;
3. il est consulté par un autre outil — C'est l'activité qui fournit un résultat à une étape. En effet, lors de la définition d'une activité, le résultat d'une étape peut être utilisé comme entrée d'une autre étape. Le diagramme de séquence correspondant au transfert d'un résultat issu d'une étape vers une autre étape est donné à la figure 3.16 .

Dans le cas d'un outil qui n'externaliserait pas son résultat, il est parfois possible d'y avoir accès par scrutation. C'est ce dont nous parlons dans la section suivante.

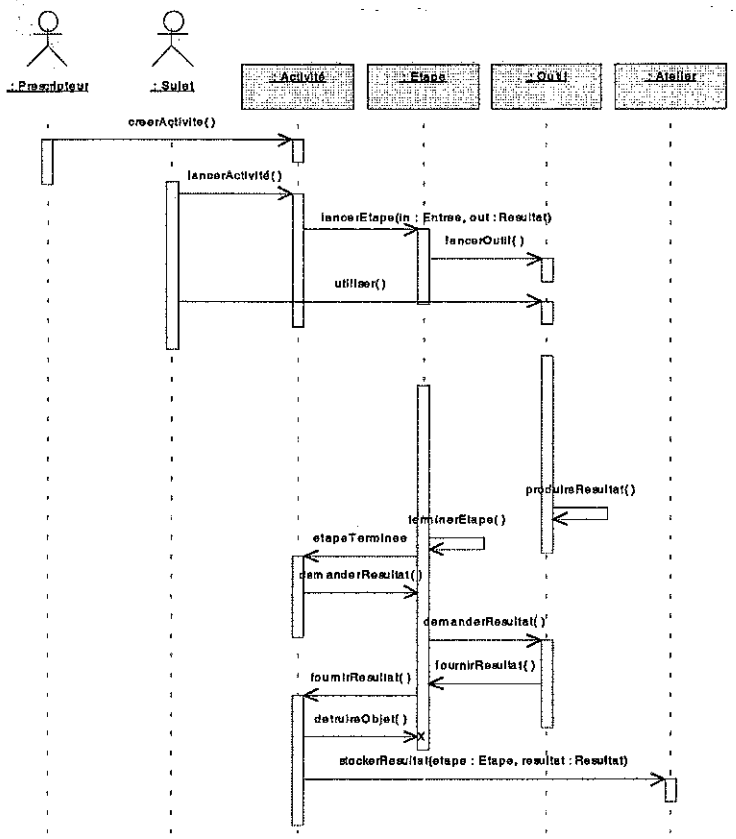


Figure 3.13 – Diagramme de séquence : un outil externalise un résultat

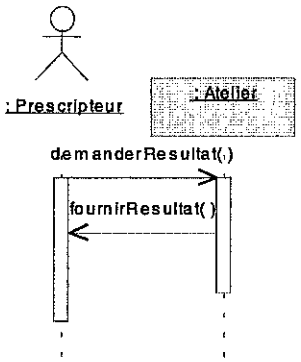


Figure 3.14 – Diagramme de séquence : le prescripteur consulte un résultat

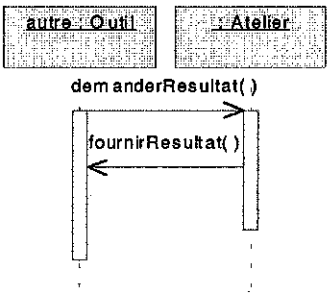


Figure 3.15 – Diagramme de séquence : un objet consulte un résultat

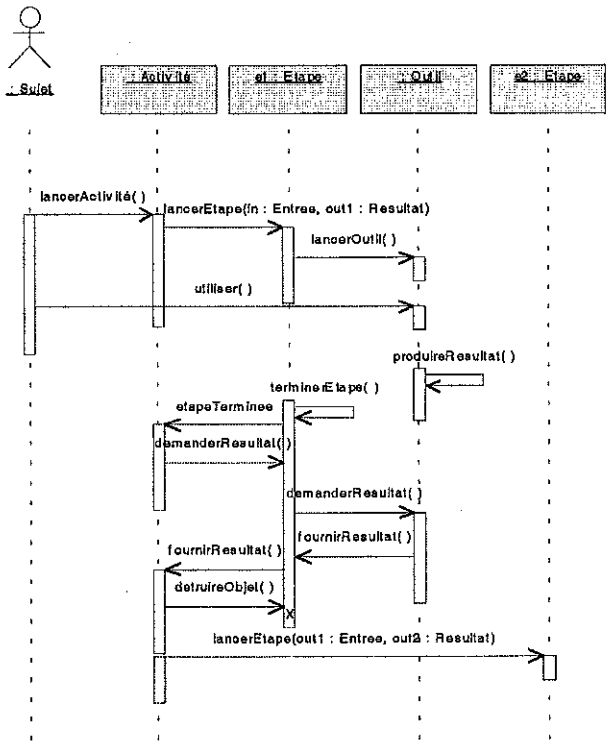


Figure 3.16 – Diagramme de séquence : l'atelier fournit un résultat en entrée d'une étape

3.4.3 Les informations scrutables

Les outils permettent de scruter la valeur de certaines informations.

Recenser les informations scrutables

L'atelier recense les informations scrutables et maintient une base d'informations. Le recensement des informations est décrit par l'algorithme 1. Dans le cas de deux outils (l'un s'appelant un,

Algorithme 1 Recenser les informations scrutables

1: **pour** chaque outil **faire**
2: /* récupérer la liste des informations scrutables */
3: demanderScrutables(scrutables : Scrutables)
4: stockerScrutables(scrutables)
5: **fin pour**

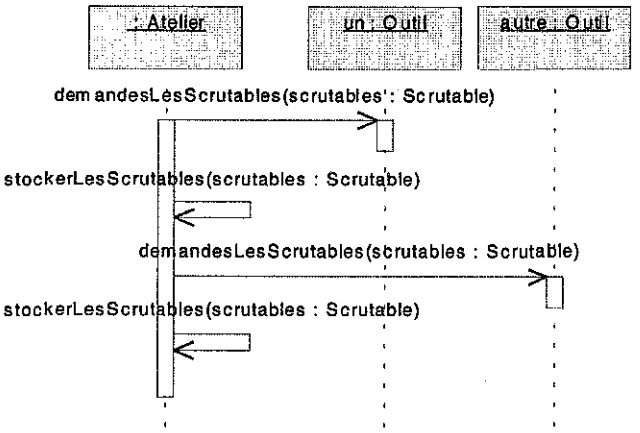


Figure 3.17 – Diagramme de séquence : l'atelier recense les scrutables de deux outils

l'autre s'appelant `autre`), l'algorithme 1 correspond au diagramme de séquence de la figure 3.17. La base d'informations est utilisée pour que les outils et le prescripteur puissent prendre connaissance des scrutables à leur disposition, dans le but de les consulter ultérieurement.

Qui produit le scrutable ? Quand est-il produit ?

Un scrutable est une information interne à l'outil. Cette information est gérée, mise à jour, *etc.* par l'outil. L'atelier n'a pas d'action sur la production du scrutable. En revanche, le scrutable peut être observé par l'atelier ou un outil ou le prescripteur à la demande au cours de l'activité. Donc, nous pouvons considérer que c'est l'outil qui fournit le scrutable.

Le prescripteur choisit des scrutables

Lors de la création d'une activité, le prescripteur peut choisir des scrutables à recueillir périodiquement lors de la réalisation de l'activité. Le diagramme de séquence correspondant au choix des scrutables par le prescripteur est donné à la figure 3.18. Le prescripteur choisit les

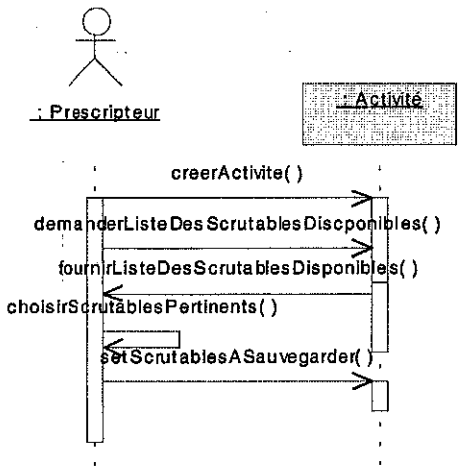


Figure 3.18 – Diagramme de séquence : le prescripteur choisit des scrutables

scrutables qui l'intéressent et détermine leur périodicité de récolte. Cette récolte peut avoir lieu une seule fois lors d'un événement particulier, ou bien régulièrement pendant l'activité. Dans le cas d'une récolte régulière, la période est réglée par le prescripteur. Elle commence avec le début de l'étape à laquelle elle est liée.

Récolte des scrutables pour le prescripteur

Quand le prescripteur a choisi des scrutables à récolter pendant l'expérimentation, le diagramme de séquence correspondant à la récolte périodique est donné à la figure 3.19.

Le prescripteur consulte des scrutables

Après avoir choisi les scrutables qui l'interessent et une fois l'activité réalisée par le sujet, le prescripteur peut consulter les scrutables enregistrés. Le diagramme de séquence correspondant à cette consultation est présenté à la figure 3.20.

Un outil consulte un scrutable

Un autre cas de consultation de scrutables est celui où c'est un outil qui demande un scrutable à un autre scrutable. Le contexte est alors celui où le sujet utilise une outil «b», alors qu'un outil «a» tourne en parallèle de celui-ci et ce, de manière transparente pour le sujet. Le diagramme de séquence correspondant au cas où un outil consulte un scrutable est présenté à la figure 3.21. Ce cas se produit toujours pendant la réalisation de l'activité.

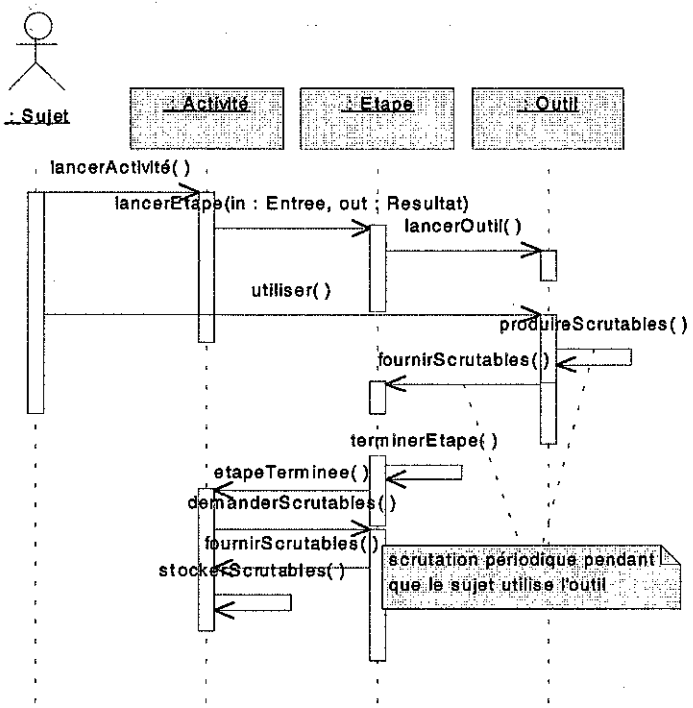


Figure 3.19 – Diagramme de séquence : production périodique des scrutables

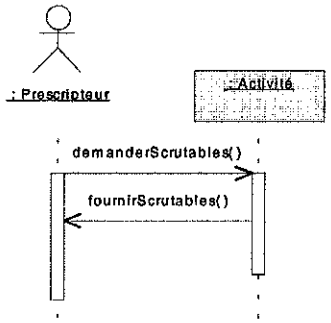


Figure 3.20 – Diagramme de séquence : le prescripteur consulte des scrutables

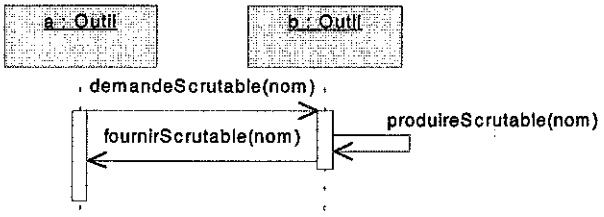


Figure 3.21 – Un outil consulte un scrutable d'un autre outil

3.4.4 Les traces de l'interaction de l'utilisateur avec les outils

Nous utilisons la composition d'événements pour fournir au prescripteur des traces d'interactions (de l'utilisateur avec un outil) qui soient intelligibles.

Recueil des événements sémantiques

Un outil produit des événements sémantiques. Il est capable de les fournir sous la forme d'une pile d'événements. Les événements sémantiques sont les éléments à partir desquels nous composons les observables.

Comment se fait l'épuration des événements inutiles ? Notre approche s'appuie sur des macro-définitions dont la grammaire est donnée ci-dessous.

```
MacroDefinition : := MacroTete constructeurDeMacroDef
                     MacroCorps terminateurDeMacro
MacroTete : := Observable
MacroCorps : : = marqueurDebutDeListe LesElements marqueurFinDeListe
LesElements : : = marqueurElement EvenementSemantique |
                  marqueurElement EvenementSemantique LesElements
```

Cette grammaire spécifie qu'une `MacroDefinition` comprend une partie gauche appelée `MacroTete`. Cette `MacroTete` correspond au nouvel observable défini. La `MacroTete` est associée au `MacroCorps` via un *constructeurDeMacroDef*. Le `MacroCorps` est une liste d'événements sémantiques qui compose le nouvel observable. Cette grammaire est une méta-grammaire car certains de ces termes dépendent des grammaires de définition des observables et des événements sémantiques. La première grammaire dépend de nous, la seconde dépend de l'outil qui implante les événements sémantiques. `Observable` et `EvenementSemantique` sont donc des non-terminaux particuliers que nous appelons «non terminaux à paramétrer».
La grammaire possède aussi :

- des non-terminaux au sens des grammaires BNF (`MacroDefinition`, `MacroTete`, `MacroCorps`, `LesElements` et macroObservable) ;
- des terminaux à paramétrer (*constructeurDeMacroDef*, *terminateurDeMacro*, *marqueurDebutDeListe*, *marqueurFinDeListe*et *marqueurElement*) ;

Pour illuster notre propos nous utilisons les événements sémantiques de figure 3.12 p 86.

Exemple 3.9 *Le prescripteur décide par exemple que les mises à jour de souris ne sont pas significatives pour lui. Il définit alors l'observable vide grâce à la macro-définition suivante :*

```
{vide} ← marqueurDebutDeListe
         marqueurElement [<fenêtre dessin> <mettre à jour souris> <nil>]
         marqueurFinDeListe
```

Le signe ← est le constructeur d'observable (*constructeurDeMacroDef*). L'observable construit est à gauche du constructeur. La liste des événements d'interface qui le compose est à droite du constructeur. C'est un observable particulier puisque c'est l'observable vide. Il est composé d'un seul événement sémantique. Les éléments en gras dépendent des grammaires définis pour eux :

- ici l'observable possède un marqueur de début ({) et un marqueur de fin (}).
- l'événement sémantique est décrit entre un marque de début ([) et un marqueur de fin (]).

Cette macro-définition d'un observable vide, implique que quand l'événement sémantique [<fenêtre dessin><mettre à jour souris><nil>] entre dans la pile d'événements sémantiques, aucun ob-

servable n'est généré.

En revanche, l'exemple 3.10 correspond à la génération d'un observable.

Exemple 3.10 *Le prescripteur décide que dans les déplacements d'objets tels des points, la nature de l'objet déplacé n'est pas pertinent. Il définit la macro-définition suivante :*

```
{<objet><déplacer><nil>} ← marqueurDebutDeListe
                           marqueurElement [<*><déplacer><*>]
                           marqueurFinDeListe
```

Cette macro-définition définit qu'un déplacement de n'importe quel objet (point, par exemple) à n'importe quel endroit correspond à l'observable déplacement d'un objet. Pour tous les événements sémantiques dont le second élément du triplet correspond exactement à <déplacer>, quelque soit la valeur des deux autres éléments du triplet, arrivant dans la pile d'événements sémantiques un observable constitué du triplet <objet><déplacer><nil> est généré.

Exemple 3.11 *Ici, deux événements sémantiques compose un observable :*

```
{<nil><déplacer souris><nil> } ← <Début>
                                 <semI> [<fenêtre dessin><déplacer souris><*>]
                                 <semI> [ <fenêtre dessin><déplacer souris><*>]
                                 <Fin>
```

Pour les deux événements sémantiques, le dernier élément du triplet peut prendre n'importe quelle valeur. Dans l'exemple 3.11, l'observable «déplacer souris» est généré à chaque fois que deux événements sémantiques «déplacer souris» dans la fenêtre de dessin se produit. Cet observable permet de réduire le nombre d'événements significatifs pour constituer la trace.

MacroObservable Nous définissons des `MacroObservables` qui sont composés d'au moins deux observables. Leur grammaire est donnée ci-dessous :

```
macroObservable : := MacroTete constructeurDeMacroDef
                     MacroCorps terminateurDeMacro

MacroTete : := Observable

MacroCorps : := Observable Observable |
                Observable MacroObservables
```

où le non terminal `Observable` est dérivé avec la même grammaire que le non terminal `Observable` de la macro-définition définie 93. L'exemple 3.12 illustre le cas d'un macroObservable.

Exemple 3.12 *Ici, l'expérimentateur ne s'intéresse pas au détail des déplacements d'objets. Pour l'interprétation de l'activité, il veut seulement savoir que le sujet à manipuler la figure. D'où la macro-définition suivante :*

```
{<figure><manipuler><nil> } ← <Début>
                              <semI> {<objet><déplacer><nil>}
                              <semI> {<objet><déplacer><nil>}
                              <Fin>
```

L'exemple 3.12 spécifie que lorsque deux observables «déplacer objet» se suivent dans la pile d'observables, il faut les remplacer par l'observable «manipuler la figure»

Un langage commun d'expression des observables

La grammaire de définition des observables permet de définir facilement les observables pour chaque outil. Nous devons maintenant définir leur format (indépendamment de leur représentation interne dans l'implantation). Le format proposé pour les événements sémantiques est intéressant à reprendre comme base pour définir les observables. Par conséquent, le format minimum d'un observable est un triplet : <objet ; action ; paramètres>. Cependant, nous avons besoin de l'enrichir.
En effet, nos observables ne sont pas acheminés directement à un et un seul logiciel cible comme dans les cas où les événements sémantiques étaient définis ([Dubourg 95, Ritter 96, Pachet 96, Durand 97, Cheikes 98, Haumont 98, Desmoulins 98]). Par conséquent, nous ajoutons à l'observable un identificateur qui permet à l'atelier de déterminer le logiciel qui a généré cet observable. L'attribution de cet identificateur se fait par le biais du gestionnaire de communication. Par ailleurs, les observables acheminés dans les exemples précédents, sont utilisés directement. Or, nous ne savons pas si le logiciel outil qui récupère l'observable, l'utilise immédiatement ou non. Par conséquent, nous ajoutons une information pour dater l'observable. Cette information est souvent utile pour interpréter les résultats de l'activité.
En résumé, nos observables sont des données qui comportent les informations suivantes : identificateur de l'outil (idO), action (A), paramètres (P), objet (O) et date (D). L'interprétation d'un tel observable est : lors de son interaction avec l'outil dont l'identificateur est idO, l'utilisateur a effectué l'action A avec les paramètres P sur l'objet O à la date D. Nous présentons une implantation de ce type de données préservant les triplets définis et utilisés par Ritter *et. al.* à la section 4.4.

Discussion

Nous sommes dans un cadre où
- le nombre d'événements d'interface N_{ei} est borné. En effet, les types d'événements d'interface dépend des interfaces déployées. Et chaque interface gère quelques événements. De plus, le nombre d'événements d'interface est fonction du temps d'utilisation du système lors de l'activité. Comme toute activité a un début et une fin, le nombre d'événements d'interface générés pendant l'activité est borné ;
- le nombre d'événements sémantiques N_{es} produits par un outil est borné. Un événement sémantique est composé d'au moins 1 événement d'interface. Et à tout événement d'interface on associe au plus 1 événement sémantique. Donc $N_{es} \leq N_{ei}$;
- le nombre d'observables N_o est borné. Un observable est composé d'au moins 1 événement sémantique. Et à tout événement sémantique on associe au plus 1 observable. Donc $N_o \leq N_{ei}$. De plus, la réduction de la file des observables permet encore de diminuer le nombre d'observables produit au cours d'une activité.

Donc il est possible d'envisager tous les cas de composition d'événements d'interface, d'événements sémantiques et d'observables. Cependant, le principe de définition des observables exposé dans cette section, ne nécessite pas obligatoirement une définition exhaustive de tous les cas de figure.
La définition des observables est à l'initiative du prescripteur, c'est naturel. Cependant, s'il lui est permis de définir ses observables directement, il est possible qu'il commette des oublis ou des erreurs, par exemple :
- dans la syntaxe des observables en partie gauche de la macro-définition ;
- dans la syntaxe des événements sémantiques en partie droite de la macro-définition ;

– dans la sémantique de son observable lors de la définition de la macro-définition.

Que se passe-t-il alors pendant l'exécution de l'activité ? Si aucun mécanisme de contrôle n'est mis en oeuvre le risque est de ne pas recueillir d'observables intéressants pour le prescripteur. Les paragraphes suivants exposent des solutions.

Mémorisation des données brutes : les événements sémantiques Si l'on veut au moins pouvoir recomposer les observables à partir des événements sémantiques, il faut mémoriser les événements sémantiques. Cette mémorisation s'avère aussi utile pour chercher d'autres observables à partir de motifs qui se répètent par exemple, auxquels le prescripteur n'avait pas pensé, et auxquels il est capable a posteriori de donner une interprétation pertinente par rapport à sa problématique ou (au contraire, pour identifier un observable vide).

Mémoriser la trace de la composition des observables De plus, pour comprendre et vérifier les observables générés, il est utile de mémoriser la trace de composition des observables sous une forme du type : tels événements sémantiques ont provoqué l'application de telle macro-définition et donc la création de tel observable. La constitution d'un tel fichier rappelle la technique mise en oeuvre dans les systèmes experts pour expliquer la production de nouveaux faits, à partir d'une base de faits et de règles de production. Il y a, en effet, une analogie entre les deux processus dont les éléments sont :

- une base de faits — événements sémantiques
- des faits nouveaux — observables
- des règles de production — macro-définition

Tout comme dans un système expert, il est possible de conserver le raisonnement qui a permis de générer un fait nouveaux, il est possible de conserver une trace de la composition d'un observable.

Mise au point d'un mécanisme de détection des erreurs de syntaxe La mise au point d'un mécanisme pour détecter au moins les erreurs de syntaxe s'avère une piste intéressante, vue que nous connaissons la grammaire de définition des macro-définition et la grammaire de définition des observables. De plus pour les outils construits en rajoutant une surcouche autour d'un prototype existant afin de lui ajouter la propriété de traçabilité, nous avons le choix de la grammaire de définition des événements sémantiques. Il nous est alors possible de vérifier la syntaxe du corps des macro-définitions. En revanche pour les outils construits à partir d'un prototype traçable, il y a deux cas :

- soit l'outil permet de faire appel à la grammaire qu'il implante pour ses événements sémantiques et l'atelier peut y recourir pour vérifier la syntaxe des événements sémantiques du corps des macro-définitions ;
- soit il ne le permet et dans ce cas, l'atelier doit implanter cette grammaire pour vérifier la syntaxe des événements sémantiques du corps des macro-définitions.

Par conséquent, il est possible de vérifier le format des événements sémantiques quel que soit l'outil qui les produit.

Construction d'un environnement de saisie des macro-définitions La construction d'un environnement de saisie des macro-définitions permet déjà d'éliminer beaucoup d'erreurs de syntaxe. Cependant elle rejoint la proposition de mettre au point de mécanismes de détection des erreurs de syntaxe. En effet, construire un environnement de saisie conforme à une grammaire, implique l'implantation de cette grammaire et de sa vérification.

Nous avons vu ici une sauvegarde des traces de l'interaction de l'utilisateur s'appuyant sur les sauvegardes gérées par les outils impliqués dans l'atelier. Cette sauvegarde est constituée des observables issus des outils et de ceux produits par l'atelier lui-même. L'intérêt est de pouvoir étudier l'utilisation de l'atelier et son impact sur l'activité.

3.4.5 Externalisation des sauvegardes

Parmi les sauvegardes, les observables constituent une liste. Nous présentons notre choix pour l'externalisation d'une liste, avec le souci de la normalisation des données. De plus, chaque observable est construit à partir d'éléments. Nous présentons notre choix pour externaliser un observable.

Comment présenter une liste de n éléments ?

Une liste de n éléments peut être externalisée (indépendamment de l'implantation choisie) dans un fichier contenant un élément par ligne. Cependant, pour une exploitation efficace de ce fichier, sa structuration est importante. La première structuration possible, consiste à «englober» la liste entre un marqueur de début de liste et un marqueur de fin de liste. Cette structuration à l'intérêt de permettre une détection aisée des fichiers tronqués. Elle permet aussi d'ajouter (avant ou après la liste, des autres informations dans le fichier). Nous choisissons donc d'externaliser une liste entre un marqueur de début de liste et un marqueur de fin de liste. Voici deux façons de présenter une liste :

Proposition 1 : un début, une fin, un séparateur entre les éléments
MarqueurDébutListe
élément$_1$ séparateur
élément$_2$ séparateur
...
élément$_n$
MarqueurFinListe

Proposition 2 : un début, une fin, un marqueur pour chaque élément
MarqueurDébut
marqueurÉlément élément$_1$
marqueurÉlément élément$_2$
...
marqueurÉlément élément$_n$
MarqueurFinListe

Proposition 3 : un début, une fin, un séparateur après chaque élément
MarqueurDébutListe
élément$_1$ séparateur
élément$_2$ séparateur
...
élément$_n$ séparateur
MarqueurFinListe

Choix d'une proposition La proposition 2 correspond à celle choisie pour SGML et ses descendants HTML et XML. Ce choix permet d'utiliser les éditeurs SGML ou XML pour présenter les listes à l'utilisateur. C'est aussi celui qui est fait pour les métadonnées. La présentation est paramétrable (DTD et feuilles de style). Elle permet en particulier la présentation d'une liste sous toutes les autres formes proposées ici ou non.

Comment présenter un triplet ?

Pour les mêmes raisons que précédemment, indépendamment de son implantation, nous externalisons un triplet <objet ; action ; paramètres> sous la forme :
<triplet>
<objet> description de l'objet </objet>
<action> verbe d'action </action>
<paramètres> paramètres de l'action sur l'objet </paramètres>
</triplet>
où triplet est remplacé par «événement sémantique» dans le cas d'un événement sémantique. Ce mode de présentation est généralisable pour un n-uplet. Nous l'utilisons pour le quintuplet qui représente un observable.

Cas d'utilisation des données sauvegardables

Le prescripteur et les outils consultent des données sauvegardables. Dans ce but, l'atelier recense toutes les données sauvegardables. Le prescripteur choisit les données pertinentes pour lui. Pour faire son choix, il consulte le résultat du recensement par l'atelier. Ces situations sont illustrées par le diagramme de cas d'utilisation de la figure 3.22.
Les cas d'utilisation apparaissant ici sont fortement liés les uns aux autres. En effet, le cas

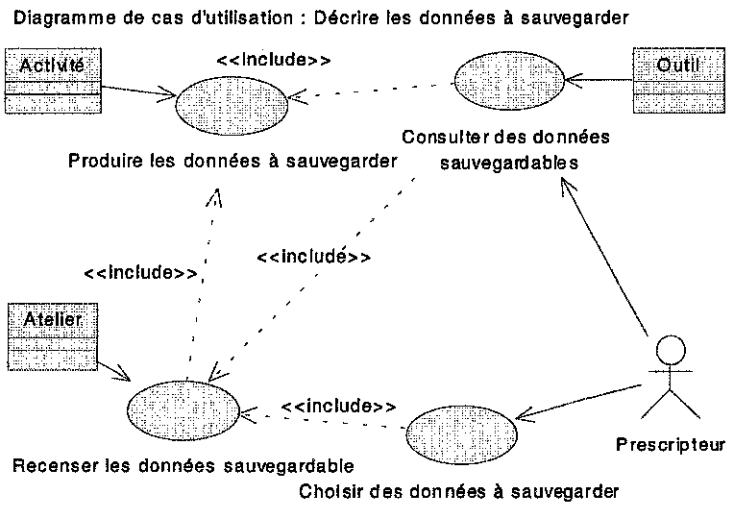


Figure 3.22 – Diagramme de cas d'utilisation : Décrire les données à sauvegarder

«produire les données à sauvegarder» est utilisé par les cas «recenser les données» et «consulter des données». De plus, ce dernier utilise le précédent tout comme le cas «choisir des données à sauvegarder».

Résultats, scrutables et observables permettent l'exploitation des activités de manière synchrone (pendant le temps de l'activité) par un outil ou de manière asynchrone (après le temps de l'activité) par un humain. Les sauvegardes sont un des types de connaissances qui circulent au sein de l'atelier. De même que nous avons définit les données sauvegardables dans l'atelier, la section suivant définit les signaux de commandes indispensables à l'atelier.

3.5 Disposer d'un langage de commande

Rappelons que l'atelier a besoin de :

- demander à un prototype quels sont les observables qu'il implante ;
- sélectionner les observables à récupérer ;
- lire les informations scrutables d'un prototype ;
- lancer/fermer un prototype ;
- activer/désactiver une fonctionnalité ;
- paramétrer l'interface graphique d'un prototype.

Nous traduisons chacun de ces besoins par une ou plusieurs commandes. Le tableau 3.3 rassemble ces commandes.

Action	**Commande**	**Effet**
Récupérer la liste des observables	Lire ObservablesListe	Récupère la liste des noms d'observables (ObservableNom)
Sélectionner les observables à récupérer	Choisir Vrai/Faux ObservableNom	Sélectionne (Vrai) ou ne Sélectionne pas (Faux) l'observable de nom ObservableNom
Lancer/arrêter le flux de observable	Lancer/Stopper ObservableFlux Sortie	Lance/arrête l'enregistrement du Flux d'observables dans Sortie (Fichier ou sortir standard)
Récupère la liste des variables scrutables	Lire VariableScrutableListe	Récupérer la liste de noms de variables scrutables (variableScrutableNom)
Lire les variables scrutables	Lire VariableScrutableNom	Lit la valeur de la variable scrutable de nom variableScrutableNom
Lancer/fermer un prototype	Lancer/Stopper PrototypeNom	Lance/arrête le prototype de nom prototypeName
Récupérer la liste des fonctionnalités accessibles	Lire FonctionnalitesListe	Récupère la liste de noms de fonctionnalités accessibles (fonctionnaliteNom)
Activer/Désactiver une Fonctionnalité	Activer/Desactiver FonctionnaliteNom	Active/désactive la fonctionnalité de nom FonctionnaliteNom
Paramétrer l'interface graphique	Lancer InterfaceGraphiqueScript	Lance le script permettant de positionner tous les paramètres de l'interface graphique

Table 3.3: Commandes pour les prototypes

3.6 Gérer les formats des connaissances

3.6.1 Les différents types de connaissances

L'atelier est amené à gérer différents formats de connaissances. En effet, chaque outil manipule une grande quantité de connaissances complexes, parmi lesquelles :

Connaissances du domaine Elles se rapportent au domaine d'apprentissage ou d'enseignement. Par exemple, en géométrie, ces connaissances sont des objets géométriques (points, droites, *etc.*) et des propriétés (perpendiculaire, parallèle, *etc.*). Nous proposons une solution pour l'échange de ce type de connaissances. Elle consiste à traduire les connaissances d'un format dans un autre (voir section 3.6.2 et annexe C) ;

Connaissances d'interactions Elles se rapportent à l'interaction du sujet avec un outil ou avec l'atelier (*cf.* section 3.4.4) ;

Connaissances pédagogiques Elles concernent les stratégies pédagogiques ou tutorielles ;

Connaissances sur l'utilisateur/apprenant Elles permettent d'adapter le comportement du système à l'utilisateur/apprenant en général (modèle de l'utilisateur/apprenant) ou à un utilisateur/apprenant particulier (profil de l'utilisateur/apprenant).

Tous ces types de connaissances sont susceptibles d'être communiquées à d'autres outils. Dans ce mémoire, nous nous focalisons en particulier sur les deux premiers types de connaissances : les connaissances d'interaction et les connaissances du domaine.

Connaissances d'interaction

En ce qui concerne les connaissances d'interactions, nous avons proposé un format de représentation des observables (section 3.4.4 page 95). Les observables sont un type de connaissances d'interactions que nous récupérons et communiquons à d'autres outils.
Afin que ces observables soient exploitables par ces autres outils, il est souhaitable que ces observables soient formatés de manière adéquate. Or chaque outil utilise sont propre format. Car il n'existe pas (encore) de standard accepté et utilisé dans toute la communauté de recherche sur les logiciels éducatifs.
Le format que nous proposons pour les observables peut nécessiter une transformation préalable à leur communication à un outil utilisant un autre format. Afin de permettre un autre formatage que celui que nous proposons, l'architecture que nous proposons permet d'ajouter un composant de service pour remplir ce rôle. Ce dernier est un composant de services. Les utilisateurs peuvent implanter et ajouter ce composant à l'atelier. Le formateur externe reçoit les observables collectés par l'atelier. Ces observables sont dans notre format. Il les convertit alors dans le format souhaité.
L'échange des connaissances d'interaction est ainsi délégué aux futurs concepteurs d'outils.

Connaissances pédagogiques et connaissances sur l'utilisateur/apprenant

L'échange des connaissances pédagogiques et des connaissances sur l'utilisateur/apprenant (en particulier le modèle de l'utilisateur/apprenant nécessite un travail qui dépasse le cadre de

cette thèse. Il serait en effet intéressant de ne pas devoir saisir plusieurs fois certaines connaissances pédagogiques (par exemple, les choix pédagogiques de l'enseignant) et certaines connaissances sur l'utilisateur (par exemple, les informations nécessaires à l'adaptation de l'interface du type, tel utilisateur est sourd ou malvoyant) Le problème ici est du même ordre que celui qui consiste à éviter la saisie multiple du même énoncé (un exemple de connaissance du domaine). La solution que nous proposons pour échanger les connaissances du domaine et éviter ainsi des saisies multiples pourrait s'avérer intéressante pour échanger les autres types de connaissance. Cependant, pour affirmer cela il faudrait faire une étude des modes de représentation de ces divers types de connaissances et évaluer la faisabilité d'un tel échange. Pour notre part, nous tentons dans un premier temps de résoudre le problème pour les connaissances du domaine.

3.6.2 Échanger les connaissances du domaine

Chaque outil implante les connaissances du domaine à sa façon. Cependant certaines de ces connaissances peuvent être utiles à plusieurs outils. C'est, par exemple, le cas des énoncés des exercices. Dans cette section, nous illustrons notre propos avec l'échange d'un énoncé.
Les énoncés sont souvent stockés de manière extérieure au logiciel dans des fichiers (on parle d'externalisation). Par conséquent, acheminer un énoncé sous forme de fichier à divers outils pourrait être une solution convenable. Or le problème est que chaque outil utilise un format différent pour son fichier d'énoncé.
Si une ontologie pour la géométrie enseignée existait, il serait possible de demander à chaque développeur de faire en sorte que son outil puisse comprendre un énoncé exprimé en suivant cette ontologie. Cependant il n'y a pas d'ontologie pour la géométrie enseignée, et la communauté internationale commence seulement à discuter de telles ontologies[44]. Par conséquent, le problème d'échange de connaissances du domaine, tels des énoncé, se ramène à un problème de format.

Notre objectif est de proposer une approche générale pour l'échange des connaissances du domaine, à travers des «macro-définitions» (page 105), obtenant ainsi un mode de traduction général pour un domaine [3, 6, 5, iv]. Après avoir présenté les principes utilisés de façon *ad hoc* pour l'échange des connaissances avec les outils de la littérature, nous formalisons cette notion de macro-définition et montrons qu'elle constitue une approche générale pour l'échange de connaissances du domaine de différents formats.

Échanger les connaissances du domaine : des solutions spécifiques aux macro-définitions

Afin que l'utilisateur utilise conjointement plusieurs outils pour traiter le même problème en exploitant leur complémentarité, il faut pouvoir échanger les connaissances du domaine. Jusqu'à présent, ceci était réalisé de deux façons : soit «à la main» par l'utilisateur, soit par des traducteurs spécifiques entre deux outils. Dans cette partie, partant de l'analyse des principes de ces deux procédés et des contraintes qui rendent la traduction réalisable, nous définissons les macro-définitions comme un moyen de généralisation de ces principes et de ces contraintes.

Utilisation conjointe de outils «à la main» Le contexte le plus courant actuellement est celui où chaque outil a sa propre représentation des connaissances. L'utilisateur doit alors définir un énoncé différent dans le langage particulier de chaque outil. Considérons que l'utilisateur dispose pour le même domaine d'un *outil source* dont le langage de représentation des connaissances

[44] *Workshop on Ontologies for Intelligent Educational Systems, held in conjunction with Conference on Artificial Intelligence in Education 1999 (AIED'99), Le Mans, France, July 18 & 19, 1999*

est *langage source* et d'un *outil destination* dont le langage de représentation des connaissances est *langage destination* (voir la figure 3.23).
L'exemple 3.13 illustre cette situation.

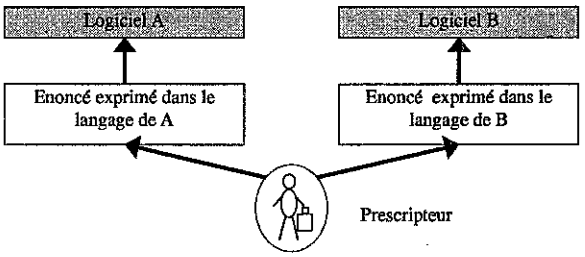


Figure 3.23 – Entrée des énoncés dans deux langages différents

Exemple 3.13 *Le prescripteur veut traiter l'exercice de géométrie dont l'énoncé est : ABC est un triangle isocèle en A et M est le milieu du segment [BC].*
Montrer que MAB est un triangle rectangle en M.
Supposons qu'il dispose des outils Mentoniezh *et* TALC *dont les langages sont HDL et CDL, correspondant chacun à langage source et langage destination. Le prescripteur traduit cet énoncé par les spécifications exprimées dans les langages HDL et CDL. Nous donnons dans le Tableau 3.4 la traduction de cet énoncé ainsi que la sémantique de chaque atome CDL.*

Dans l'exemple 3.13, la définition de l'énoncé dans chacun des deux langages est réalisable sans perte d'informations.
Cette approche présente toutefois l'inconvénient d'obliger l'utilisateur d'une part à connaître plusieurs langages et d'autre part à définir plusieurs fois les mêmes connaissances. Le seul avantage est qu'elle ne nécessite aucun moyen logiciel particulier puisqu'elle repose uniquement sur les compétences et la bonne volonté de l'utilisateur. Cependant, avec cette méthode, nous n'avons aucun moyen de vérifier s'il s'agit du même énoncé, que l'on donne aux deux outils.

énoncé	Langage HDL	Langage CDL	Sémantique CDL
ABC est un triangle isocèle en A	isocèle (A, B, C)	$non(C \in (AB))$	Le point C n'appartient pas à la droite contenant les points A B.
		seg1 = [A, B]	seg1 est un segment d'extrémités A et B.
		seg2 = [B, C]	seg2 est un segment d'extrémités B et C.
		seg3 = [A, C]	seg3 est un segment d'extrémités A et C.
		|A B| = |A C|	La distance entre A et B est égale à la distance entre A et C
M est le milieu du segment [BC]	milieu (M, B, C)	$M \in [BC]$	Le point M appartient à la droite contenant les points B C
		$\|B\ M\| = \frac{1}{2}\ \|B\ C\|$	La distance entre B et M vaut la moitié de la distance entre B C

Montrer que MAB est un triangle rectangle en M	trirect (M, A, B)	$non(M \in (AB))$	Le point M n'appartient pas à la droite contenant les points A B
		$seg4 = [A, B]$,	seg4 est un segment d'extrémités A et B.
		$seg5 = [B, M]$,	seg5 est un segment d'extrémités B et M.
		$seg6 = [A, M]$,	seg6 est un segment d'extrémités A et M.
		$seg5 \perp seg6$	Les segments MA et MB sont perpendiculaires

Table 3.4: Traduction de l'énoncé de l'exemple en HDL et en CDL

Des traducteurs au niveau des connaissances : principes des traducteurs spécifiques
Pour remédier aux inconvénients de l'approche précédente (de saisies multiples dans différents langages sans moyen de vérifier s'il s'agit du même énoncé), les premiers travaux visant à l'utilisation conjointe de deux Logiciels Éducatifs reposaient sur le modèle client-serveur. Cela se traduisait au niveau des connaissances par la traduction du langage du Logiciels Éducatifs client dans le langage du Logiciels Éducatifs serveur. En géométrie, *CABRI-Géomètre* [Baulac 90, Baulac 92] a ainsi bénéficié d'une version spéciale «serveur» qu'ont utilisé des systèmes comme *CABRI-DéFI* [Gras 88, Giorgiutti 91, Gras 96], *HYPERCARRÉ* [Capponi 91], *TéLéCABRI* [Tahri 93] et *TALC* [Desmoulins 94]. Dans ces systèmes, un traducteur spécifique est développé à l'intérieur du client (les communications utilisant les possibilités du système).
En résumé, l'approche utilisée consiste à choisir un langage, par exemple *langage source*, et à le traduire automatiquement dans un autre langage, le *langage destination* (lorsque que cette traduction est possible). Dans ce cas, l'utilisateur entre les connaissances du problème à traiter dans le *langage source*. Un traducteur transforme les connaissances exprimées en *langage source* en un *langage destination*. Les connaissances exprimées dans le *langage destination* sont ensuite transmises à l'*outil destination* (voir Figure 3.24).

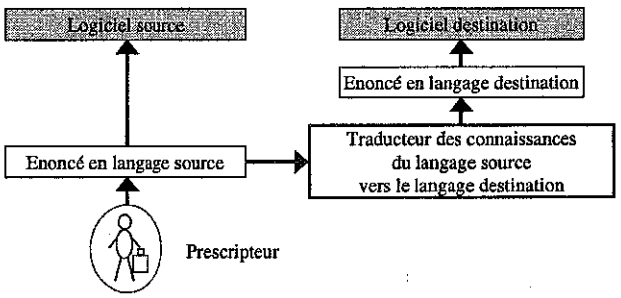


Figure 3.24 – Utilisation d'un traducteur spécifique.

Exemple 3.14 *Avec l'énoncé de l'exemple 3.13, le prescripteur traduit uniquement son énoncé en HDL. La traduction en CDL est faite automatiquement avec un traducteur.*

Cette méthode résout (si la traduction est correcte) le problème précédent de saisies multiples dans différents langages sans moyen de vérifier s'il s'agit du même énoncé, mais présente l'inconvénient d'obliger les concepteurs à implanter un traducteur spécifique pour chaque couple de Logiciels Éducatifs, ce qui n'est pas satisfaisant.

Contraintes de traductibilité

Pour éviter aux concepteurs d'implanter un traducteur pour chaque couple de Logiciels Éducatifs, nous cherchons à définir une approche générale pour la traduction automatique des connaissances. Évidemment, dans le paragraphe précédent le choix du *langage destination* de la traduction automatique ne peut être fait au hasard et dépend de conditions que nous appelons «contraintes de traductibilité». Les définitions préliminaires ci-dessous sont introduites en vue d'aboutir à une définition de «traductibilité».

Définition 3.1 *Atome.*
Un atome d'un langage est le constituant minimal d'un énoncé dans ce langage.

Autrement dit, un énoncé est composé d'une suite d'atomes. Les éléments d'un atome n'ont pas de sens hors de l'atome.

> **Exemple 3.15** *En HDL, égale(A, B, A, C) est un atome, en CDL, /A B/ = /A C/ est un atome mais ni /A B/, ni /A C/, ni A, ni B, ni C n'ont de sens hors de ces atomes.*

Définition 3.2 *Granularité.*
Un langage destination, recouvrant un langage source, est de granularité plus fine qu'un langage source si toute connaissance exprimable par un seul atome dans le langage source peut être décomposée en parties exprimables chacune par un atome de langage destination,
noté langage destination $\leq$ langage source.

Cette notion exprime que deux langages décrivent un domaine à des niveaux de détails plus ou moins grands.

> **Exemple 3.16** *En géométrie, un langage destination représentant par un atome les objets de type point, segment, perpendiculaire, parallèle est de granularité plus fine qu'un langage source représentant par un atome les objets de type triangle, rectangle, triangle rectangle, parallélogramme. Dans cet exemple, langage destination $\leq$ langage source.*
>
> **Contre-exemple 3.17** *En géométrie, un langage destination représentant par un atome les objets de type triangle, quadrilatère, pentagone et la propriété «les longueurs des cotés du polygones sont égales» est de même granularité qu'un langage source représentant par un atome les objets de type triangle équilatéral, et losange. Ici langage destination et langage source ont la même granularité.*

Définition 3.3 *Recouvrement.*
Un langage source est recouvert par un langage destination si tout concept exprimable dans le langage source, est exprimable dans le langage destination,
noté langage source $\subseteq$ langage destination.

Autrement dit, *langage source* est recouvert par *langage destination* signifie que le domaine représenté par le *langage destination* est plus grand que le domaine représenté par le *langage source*.

Exemple 3.18 *En géométrie, un langage source exprimant les concepts de point, droite et cercle est recouvert par un langage destination exprimant les concepts de point, droite, cercle et segment. Dans cet exemple langage source* $\subseteq$ *langage destination.*

Contre-exemple 3.19 *En géométrie, un langage source exprimant les concepts de point, droite et cercle n'est pas recouvert par un langage destination exprimant les concepts de point, droite, segment et demi-droite. Dans cet exemple langage source* $\nsubseteq$ *langage destination.*

Définition 3.4 *Traductibilité.*
Un langage source est traduisible dans langage destination si le domaine du langage source est recouvert par le domaine du langage destination et que de surcroît, la granularité des connaissances est plus fine dans le langage destination que dans le langage source.

Nous avons ainsi obtenu une définition de la traductibilité de *langage source* dans *langage destination* qui permet d'associer à tout atome du *langage source*, un énoncé en *langage destination* qui représente la même connaissance.

Les macro-définitions

Soit un *langage source* traduisible dans un *langage destination* selon la définition de la section précédente. Une façon générale de réaliser un traducteur automatique de *langage source* en *langage destination* est d'utiliser ce que nous appelons des macro-définitions.
Intuitivement, la spécification d'une macro-définition exprime qu'un seul atome d'un *langage source* peut être remplacé par une liste d'atomes d'un *langage destination* (voir exemple 3.20). Si *langage source* est traduisible en *langage destination*, alors l'idée est d'associer une macro-définition exprimée en *langage destination* à chaque atome du *langage source*. Le langage de définition de macro-définition est appelé **macro-langage.**

Exemple 3.20 *Macro-définition*
soit la macro-définition
$isocèle(A, B, C) \leftarrow non(C \in (AB)), seg1 = [A, B], seg2 = [B, C], seg3 = [A, C], |AB| = |AC|.$
où la partie précédant le symbole $\leftarrow$ *est exprimée en HDL et le reste est exprimé en CDL*

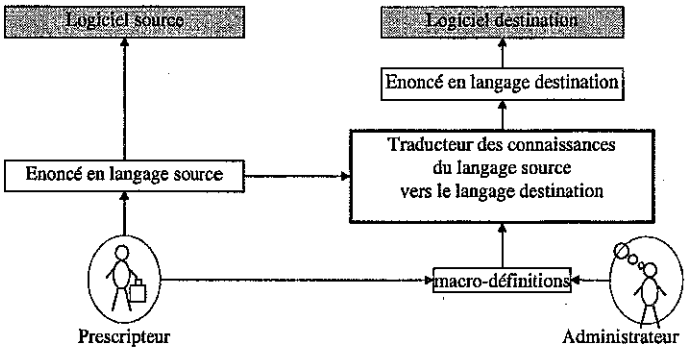


Figure 3.25 – Entrée de l'énoncé en macro-*langage destination*

Sur la figure 3.25, le traducteur est universel dans le sens où il ne dépend pas des langages source et destination. Cela est rendu possible grâce à un certain nombre de paramètres qu'il reçoit

en entrée, et notamment un ensemble de macro-définitions. Ce mécanisme permet de traduire un énoncé exprimé en *langage source* pour un outil en un *langage destination* pour un autre outil, grâce à un ensemble des macro-définitions. Nous appelons ce traducteur un **interprète de macro-définitions**. Cet interprète est intégré dans l'atelier comme un service pour échanger les connaissances.

Du point de vue de l'utilisateur, la situation est au moins celle de la section précédente, c'est-à-dire qu'il n'exprime qu'une fois l'énoncé de son problème pour deux outils (voir Figure 3.25). Elle est même enrichie, car il peut exprimer ses propres macro-définitions, ce qui lui permet d'enrichir les deux langages (source et destination) voire de donner lui-même les macro-définitions permettant la traduction automatique.

Du point de vue de l'administrateur de Logiciels Éducatifs, il n'y a plus besoin de définir un traducteur spécifique pour chaque application, il suffit de donner les macro-définitions exprimées dans le macro-langage de l'*outil destination*. Au lieu d'avoir à définir un traducteur spécifique pour chaque couple de prototypes, il suffit de définir un macro-langage par prototype.

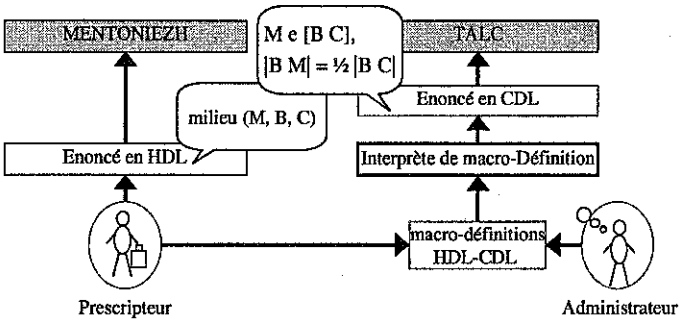


Figure 3.26 – Traduction d'une ligne de la table

Formalisation des macro-définitions

Pour formaliser cette notion de macro-définition , nous donnons la syntaxe et les contraintes syntaxiques d'un macro-langage à partir de l'ensemble de définitions suivantes.

Syntaxe Nous donnons ci-dessous la syntaxe d'un langage de macro-définitions dans le formalisme BNF :

```
MacroTexte : := MacroDefinition terminateurDeMacroTexte
              | MacroDefinition MacroTexte
MacroDefinition : := MacroTete constructeurDeMacroDef MacroCorps terminateurDeMacro
MacroTete : := AtomeEnLangageSource
MacroCorps : := MacroAtome | MacroAtome separateurDAtomeDuLangageDestination MacroCorps
MacroAtome : := AtomeEnLangageDestination | macroUtilisation
```

Cette grammaire spécifie qu'un texte dans un macro-langage est une suite de `MacroDefinitions`. Une `MacroDefinition` est composée d'une partie gauche appelée `MacroTete`, d'une partie droite appelée `MacroCorps`, séparées l'une de l'autre par le *constructeurDeMacroDef*. La `MacroTete` est un atome du *langage source*, le `MacroCorps` est une phrase du *langage destination* (utilisant éventuellement des `MacroDefinitions` via des `macroUtilisations`).

Cette grammaire est une sorte de méta-grammaire, car certains de ces termes dépendent des langages source et destination. Ces termes (en italiques dans la grammaire) ne sont donc pas

stricto sensu des non terminaux (contenant des majuscules en début de mots) ou des terminaux (tout en minuscule). Nous distinguons quatre types de termes dans cette grammaire :

1. «vrais non terminaux»
   = `MacroTexte`, `MacroDefinition`, `MacroTete`, `MacroCorps` et `MacroAtome` ;
2. «non terminaux à paramétrer»
   = *`AtomeEnLangageSource`*, *`AtomeEnLangageDestination`* ;
3. «terminaux à paramétrer»
   = *terminateurDeMacroTexte*, *constructeurDeMacroDef*, *terminateurDeMacro*, *separateurDAtomeDuLangageDestination* ;
4. «terminal particulier»
   = `macroUtilisation`.

Le premier ensemble de non terminaux sont des non terminaux au sens des grammaires BNF. Dans le deuxième ensemble, chaque non terminal est dérivé dans la grammaire du *langage source* ou du *langage destination*. Par conséquent, nous ne donnons pas de règles dans notre grammaire pour dériver ces non terminaux. Dans le troisième ensemble, chaque terminal est un paramètre pour l'interprète de macro-définitions. Par exemple, *separateurDAtomeDuLangageDestination* représente ici le séparateur d'atomes habituel dans le *langage destination*, c'est-à-dire le symbole ',' quand le *langage destination* est CDL. Dans le quatrième ensemble, le terminal, est cherché dans une liste de terminaux qui est maintenue par l'interprète de macro-définitions (cela signifie que les non terminaux reconnus existent déjà). Cela est rendu possible par les règles expliquées page 108. Autrement dit, une `macroUtilisation` est une `MacroTete` qui est déjà définie (c'est-à-dire dont la dérivation est déjà connue).

Nous donnons une instanciation de cette «méta-grammaire» en annexe C.

Exemple 3.21 *Dans l'exemple 3.20, la `MacroTete` est exprimée en HDL (isocèle (A, B, C)), le `MacroCorps` est exprimé en CDL*
$(non(C \in (AB)), seg1 = [A, B], seg2 = [B, C], seg3 = [A, C], |AB| = |AC|.)$.
Le separateurDAtomeDuLangageDestination en CDL est la virgule.
Le terminateurDeMacroTexte utilisé ici est le terminateur de texte en CDL, c'est-à-dire le point.

L'exemple 3.22 en donne une illustration simple, généralisable sans difficulté.

Exemple 3.22 *Soit les macro-définitions :*
$triangle(P1, P2, P3) \leftarrow non(P3 \in (P1P2)), seg1 = [P1, P2], seg2 = [P2, P3], seg3 = [P1, P3]$.
$isocèle(A, B, C) \leftarrow triangle(A, B, C), |AB| = |AC|$.
où pour la deuxième macro-définition , la `MacroTete` est exprimée en HDL (isocèle (A, B, C)), le `MacroCorps` est exprimé en CDL et utilise la `macroUtilisation` d'un triangle (triangle(A, B, C)).

Dans cet exemple, le `MacroCorps` de «isocèle (A, B, C)» est composé d'une part d'une `macroUtilisation`, et d'autre part d'un atome du langage source (ici CDL).

Analogie Macro-définition—Procédure Nous illustrons notre propos avec l'exemple 3.22. Nous pouvons faire une analogie entre la définition d'une macro-définition et la définition d'une procédure :

- la macro-définition ($triangle(P1, P2, P3) \leftarrow non(P3 \in (P1P2)), seg1 = [P1, P2], seg2 = [P2, P3], seg3 = [P1, P3].$) est une procédure ;
- sa spécification est $triangle(P1, P2, P3)$;
- son corps est $non(P3 \in (P1P2)), seg1 = [P1, P2], seg2 = [P2, P3], seg3 = [P1, P3].$;
- ses paramètres formels sont $P1$, $P2$ et $P3$.

Nous pouvons faire une analogie entre l'utilisation d'une macro-définition et l'appel d'une procédure :

- la `macroUtilisation` $triangle(A, B, C)$ est un appel à cette procédure ;
- ses paramètres effectifs sont A, B et C ;
- La correspondance entre les paramètres formels et les paramètres effectifs se fait par position.

La traduction de la `macroUtilisation` donne $non(C \in (AB)), seg1 = [A, B], seg2 = [B, C], seg3 = [A, C]$.

Contraintes syntaxiques Pour définir contraintes syntaxiques dans l'utilisation des macro-définitions, nous présentons les règles à respecter et l'algorithme de traduction qui peut être appliqué quand elles sont respectées.

Règles Les règles à respecter sont :

Règle 1 *Tout `MacroCorps` n'utilise que des `macroUtilisations` définies ailleurs.*

Règle 2 *Les macro-définitions récursives sont interdites, même indirectement (récursivité croisée).*

Règle 3 *Deux macro-définitions du même `MacroTexte` ne peuvent pas avoir la même `MacroTete`.*

Règle 4 *Tout paramètre formel de la `MacroTete` est utilisée dans le `MacroCorps`.*

Règle 5 *Les types des paramètres formels d'une définition sont identiques aux types des paramètres effectifs des `macroUtilisations`.*

Ces cinq règles permettent un ordre quelconque des macro-définitions. Cependant il existe toujours un ordre des macro-définitions pour lequel la règle 6 plus simple remplace les règles 1 et 2 :

Règle 6 *Tout `MacroCorps` n'utilise que des `macroUtilisations` définies avant dans l'ordre des macro-définitions.*

Nous retenons donc l'ensemble des règles 6, 3, 4, 5.

Sémantique de la traduction Dans une macro-définition , le symbole «←» (*constructeurDeMacroDef*) exprime la traduction d'une `MacroTete` en un `MacroCorps`. Cette traduction est réalisée en substituant le `MacroCorps` à la `MacroTete` par pour toute `macroUtilisation`, et en substituant les paramètres effectifs aux paramètres formels, dans un énoncé exprimé dans le *langage source*. On peut aussi la concevoir sur le plan de la logique comme une implication, car la phrase du `MacroCorps` étant une des façons de définir la `MacroTete`, le `MacroCorps` implique la `MacroTete` dans le domaine concerné. La traduction d'un énoncé en langage source suivant un `MacroTexte` consiste, pour chaque macro-définition prise dans l'ordre du `MacroTexte`, à remplacer toute occurrence correspondante de `macroUtilisation`. Ce processus de substitution est similaire à celui utilisé par le préprocesseur du compilateur C pour les $\sharp define$ [Kernighan 88]. Cette traduction est exprimée par l'algorithme 7 p 128 qui s'applique si le second ensemble de règles

est respecté. Si l'on veut permettre un ordre quelconque des macro-définitions (premier ensemble de règle), l'algorithme correspondant consiste simplement à ordonner les macro-définitions et à appliquer ensuite l'algorithme 7.

Discussion

En résumé, pour assurer l'échange des connaissances entre deux Logiciels Éducatifs, nous avons quatre possibilités :

1. écrire un traducteur spécifique pour chaque couple de Logiciels Éducatifs E1, E2. Alors pour n Logiciels Éducatifs, il faut $n * (n-1)$ traducteurs (soit presque n^2) ;
2. définir un langage intermédiaire (ou inter-langage [Delevenay 59]), et 2 traducteurs pour chaque couple de Logiciels Éducatifs. Pour n Logiciels Éducatifs, il faut $2n$ traducteurs, et 1 inter-langage ;
3. définir un traducteur utilisant des macro-définitions [Aho 72], et définir autant de `MacroTextes` permettant la traduction des langages de couples de Logiciels Éducatifs E1, E2. Pour n Logiciels Éducatifs, il faut 1 traducteur et $n * (n-1)$ `MacroTextes` (soit presque n^2) ;
4. définir un langage commun, un traducteur utilisant des macro-définitions et autant de `MacroTexte` que de Logiciels Éducatifs. Pour n Logiciels Éducatifs, il faut 1 traducteur, 1 inter-langage et $2n$ `MacroTextes`.

Nous avons éliminé la possibilité 1, trop coûteuse en nombre de traducteurs à implanter. La possibilité 2 est intéressante, mais la difficulté réside dans la définition *a priori* d'un inter-langage pour les Logiciels Éducatifs de chaque domaine d'application. La possibilité 3 est coûteuse en nombre de `MacroTextes` à écrire, mais l'écriture d'un `MacroTexte` est aisée[45]. Cette solution est assez réaliste. La possibilité 4 est alléchante mais rencontre la même difficulté que la possibilité 2. Cependant à partir de la réalisation de la possibilité 3, nous pouvons tendre vers la possibilité 4. En effet, si nous choisissons comme inter-langage, un des langages existants et non un inter-langage inventé *a priori*, le nombre de `MacroTextes` à écrire passe de $2n$ à $2n-2$. Ce gain est minime quand n croît. Cependant cet inter-langage, choisi pour ses caractéristiques constituerait un inter-langage défini *a posteriori*. L'inter-langage (pivot) changerait à chaque fois que cela serait nécessaire (en utilisant le langage recouvrant le plus grand domaine et de granularité la plus fine). Cet inter-langage sera raffiné au fur et à mesure que des Logiciels Éducatifs aux langages plus «recouvrants» et de granularité plus fine inter-opéreront. Nous obtiendrions ainsi un inter-langage défini *a posteriori* de plus en plus général. La définition formelle que nous donnons du concept de macro-définition est une façon générale de réaliser un traducteur automatique d'un langage dans un autre, si le premier est traduisible dans le second. En effet, elle permet d'associer à tout énoncé du premier langage, utilisant des macro-définitions (correctes) d'un `MacroTexte`, un énoncé du second langage, par le principe des substitutions successives.

Cette section achève le développement de nos propositions.

[45] Intuitivement, il est plus aisée d'écrire la traduction de chaque atome d'un *langage source* vers un *langage destination* dans un texte, que d'écrire un nouveau traducteur spécifique.

Chapitre 4

Implantation de l'atelier

4.1 Introduction

L'atelier d'activité de Logiciels Éducatifs est construit dans une architecture distribuée de type client-serveur, impliquant plusieurs composants. Comme nous l'avons expliqué dans la section 2.3.4, les composants sont une extension de la notion d'objet. Par conséquent les langages les plus adaptés pour implanter des composants sont les langages objets ou les langages orientés objets. **Notre choix s'est porté sur JAVA**. En effet, JAVA est un langage objet, qui a pour caractéristique d'être disponible sur des plate-formes matérielles différentes. Or notre atelier, du fait de la diversité des prototypes qu'il est amené à faire coopérer, doit pouvoir tourner sur des plate-formes matérielles différentes telles que des Macintosh, des PC ou des stations de travail sous différents systèmes d'exploitation.

La section 4.2 présente la structuration de l'implantation de ces fonctionnalités dans deux composants de l'atelier : le gestionnaire d'activités et EduMed. Il sont implantés, suivant la hiérarchie de classes de l'atelier présentée sur la figure 4.1 : l'atelier est composé d'un médiateur (classe `EduMed`) et d'un gestionnaire d'activités (classe `GesExpe`). Rappelons que le gestionnaire

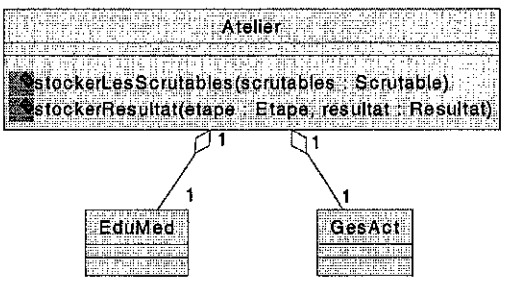


Figure 4.1 – Les classes principales de l'atelier

d'activités prend en charge l'activité proprement dite, tandis qu'EduMed prend en charge les aspects techniques nécessaires à la coopération des prototypes. La figure 3.2 p67 présente l'articulation des trois types de composants de l'atelier : les prototypes, EduMed et le gestionnaire d'activités.
Les prototypes sont pris en charge par EduMed. Chaque prototype (dans un ovale) est encapsulé dans un objet JAVA (des rectangles) accessible via EduMed. Cette encapsulation est présentée dans la section 4.8. L'implantation d'EduMed est présentée dans la section 4.2.2. Celle du gestionnaire d'activités est présentée dans la section 4.2.3.

Les implantations des différentes fonctions prises en charge par EduMed et le gestionnaire d'activités sont présentées aux sections 4.3 à 4.7.

Ce chapitre se poursuit en présentant la mise en œuvre de l'atelier (sections 4.8 à 4.10).

Pour terminer, nous faisons un bilan de l'implantation (section 4.11) permettant de faire le point sur ce qui est implanté, en cours d'implantation ou déployé.

4.2 Architecture générale et présentation des principaux composants

Nous avons vu dans le chapitre précédent (section 3.2.1), que nous choisissons d'implanter l'atelier autour d'un médiateur. Parmi les quatre principaux types de médiateurs, nous avons opté pour un médiateur basé sur un bus logiciel (ORB) suivant la norme CORBA.

4.2.1 Notre utilisation de CORBA

De nombreux produits commerciaux implantent un ORB, comme par exemple VisiBroker [VisiBroker http] ou encore Orbix [Orbix http]. Le lecteur intéressé par un inventaire des produits peut consulter le site suivant : http ://www.lifl.fr/ merle/corba/products.html. Ces produits permettent de développer un projet dans son intégralité en JAVA et de faire le portage sur CORBA. Pour notre implantation, nous avons cherché à utiliser un ORB gratuit. Nous utilisons pour cela JacORB [JacORB http] . Au début de notre travail, JacORB proposait un ORB mis en œuvre par un étudiant allemand. Depuis, plusieurs versions sont sorties et JacORB continue d'évoluer sous licence GPL[46].

Les outils, le gestionnaire de formats et le gestionnaire d'interfaces graphiques offrent leurs services au gestionnaire d'activités. Pour ce faire, les services de chacun sont décrits en IDL dans des vitrines. Ces vitrines servent à la spécification d'objets CORBA. Elles sont projetées et donnent des souches et des squelettes IDL.
Le gestionnaire d'activités est l'application cliente qui hérite des souches IDL des outils, du gestionnaire de formats et du gestionnaire d'interfaces graphiques. Pour leur part, les outils, le gestionnaire de formats et le gestionnaire d'interfaces graphiques sont encapsulés dans des applications serveurs implantant les squelettes IDL.

La projection des vitrines est réalisée par un pré-compilateur IDL. Pour le langage JAVA, ce pré-compilateur s'appelle *IDL2JAVA* ou *IDLTOJAVA* suivant l'implantation de la norme CORBA choisie. Le pré-compilateur utilise génère cinq fichiers, à partir de la spécification. Nous les décrivons sur l'exemple 4.1.

[46]GNU General Public License : http ://www.gnu.org/copyleft/gpl.html

Exemple 4.1 *Notre gestionnaire de formats est composé principalement d'un interprète de macro-définitions. Pour que cet interprète publie ses services, nous les décrivons dans un fichier* `Interpret.idl`*. Le compilateur compile les fichiers (stockant les vitrines exprimées en IDL) en code source en suivant les correspondances IDL-to-Java («IDL-to-Java mappings») définies par l'OMG. Dans notre cas, la projection de ce fichier se fait par la commande*
`idltojava Interpret.idl`
Cette commande crée un répertoire appelé `InterpretApp` *contenant cinq fichiers :*

- `_InterpretImplBase.java` *: définit le squelette IDL du serveur ;*
- `_InterpretInterpret.java` *: définit la souche IDL du client ;*
- `Interpret.java` *: contient l'implantation de la vitrine, et donc le code associé aux méthodes publiées pour l'interprète de macro-définitions ;*
- `InterpretHelper.java` *: fournit des fonctionnalités* CORBA *pour gérer les objets* CORBA *;*
- `InterpretHolder.java` *:* `InterpretHolder.java` *fournit des fonctionnalités pour gérer les structures de données, les opérations et les paramètres de l'interface.*

Une fois la projection obtenue, il reste à compléter dans la partie cliente l'invocation des souches des objets, et dans la partie serveur l'implantation des services.

4.2.2 EduMed

L'atelier est construit autour d'un ensemble de composants : un ORB, des «composants-prototypes», des composants de services et le composant «gestionnaire d'activités ». EduMed est composé de l'ORB déployé et des composants de services.

L'ORB utilise le mécanisme de l'invocation dynamique. Les composants de services implantés par EduMed sont pour l'instant au nombre de deux. Ce nombre est amené à évoluer en fonction des besoins identifiés lors d'activités.
L'un des composants de services a pour rôle d'assurer l'échange de connaissances. En effet, nous avons vu que l'atelier est amené à manipuler des connaissances de formats différents. C'est le gestionnaire de formats qui remplit ce rôle. Nous le décrivons dans la section 4.6.
L'autre composant est chargé de gérer les interfaces graphiques fournies par les différents prototypes afin de présenter au sujet de l'activité une interface aussi unifiée que possible. C'est le gestionnaire d'interfaces graphiques qui remplit ce rôle. Il est présenté à la section 4.7.
La figure 4.2 présente les classes de l'implantation d'EduMed.

4.2.3 GesAct

Le gestionnaire d'activités utilise l'ORB et les vitrines des composants pour gérer l'expérimentation. Il prend en charge les fonctionnalités suivantes :

- le déroulement de l'expérimentation via un gestionnaire de scénario ;
- l'indexation des prototypes et des fonctionnalités ;
- le recueil des données.

Pour ce faire, nous avons créé les objets `GesSce` pour le déroulement de l'expérimentation, `Index` pour l'indexation (il gère par exemple, une liste de fonctionnalités pour un domaine donné) et

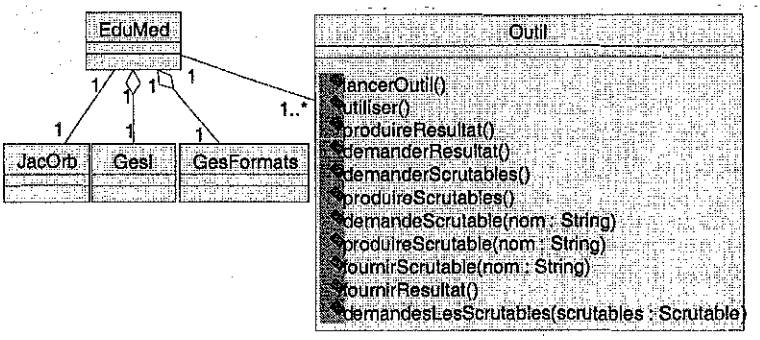


Figure 4.2 – Les classes java d'EduMed

`GesData` pour le recueil de données sauvegardables.
Chacun de ces objets utilise d'autres objets qui sont décrits dans la section appropriée : 4.3 pour `GesSce`, 4.9 pour `Index` et 4.6 pour `GesData`.
La figure 4.3 présente les classes de l'implantation du gestionnaire d'activités.

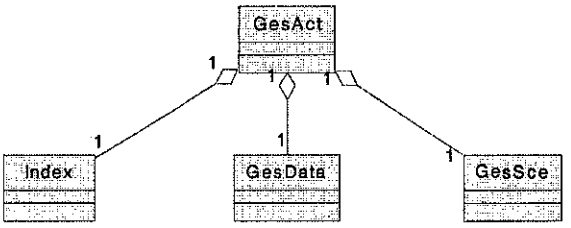


Figure 4.3 – Les classes java du gestionnaire d'activités

4.3 Les scénarios

Le gestionnaire de scénario gère le déroulement de l'activité, que nous appelons scénario (d'où le singulier à scénario). C'est un composant qui permet de créer et d'exécuter un scénario d'activité. Il manipule un objet `Scenario`. Les fonctions qui appartiennent au diagramme de cas d'utilisation «gérer le déroulement l'activité» sont présentées à la section 4.3.1. Nous présentons ensuite l'implantation choisie pour le scénario (section 4.3).

4.3.1 Les fonctions du gestionnaire de scénario

Les fonctions de l'atelier pour gérer le scénario sont les suivantes :

Définir un scénario — Cette fonction consiste à créer un scénario de A à Z. Elle définit à la fois ce qui est demandé au sujet (étapes et transitions) et ce que le prescripteur souhaite extraire de l'activité (données). Nous avons détaillé la création des étapes. Nous détaillons le recueil des données à la section 3.4 ;

Exécuter un scénario — Cette fonction consiste à exécuter étape après étape un scénario ;

Enregistrer un scénario — Cette fonction consiste à enregistrer le scénario courant dans un fichier physique ;

Charger un scénario — Cette fonction consiste à lire un scénario préalablement enregistré physiquement dans un fichier. Le scénario ainsi chargé en mémoire est en lecture seulement ;

Modifier un scénario — Cette fonction consiste à autoriser la modification d'un scénario.

L'interface graphique de l'atelier présente un menu `Scénario` pour accéder à ces fonctions. Ce menu est composé de six items. Les intitulés des menus ont fait l'objet d'une adaptation à l'utilisateur et aux règles de nommage couramment pratiqués dans les logiciels. Nous donnons ci-dessous les intitulés des menus ainsi que la fonction correspondante.

- `chOisir (chOse)` — fonction charger — Cet item permet de choisir et de charger un scénario existant. Le scénario ainsi chargé est en lecture seule — aucune modification n'est autorisée ;
- `Nouveau (New)` — fonction définir — Cet item permet de définir un nouveau scénario ; Le nouveau scénario créé vide lors du lancement de cette commande porte le nom `noname0.sce` jusqu'à ce qu'il soit nommé par l'utilisateur via la commande `sauvegarder`.
- `enregiStrer (Save)` — fonction enregistrer — Cet item permet de sauvegarder un scénario en cours de création ou d'édition sous le nom courant. Il provoque l'ouverture d'une boite de dialogue pour saisir un nom si le nom actuel du scénario commence par `noname`. Cette commande tient compte de l'état du scénario — un message signale l'opération impossible si le scénario est en lecture seule. Les scénarios sauvegardés portent l'extension *.sce*.
- `enregistrer Autre (save As)` — fonction enregistrer — Cet item permet de sauvegarder un scénario sous un autre nom. Cette commande ne tient pas compte de l'état du scénario — il permet de sauvegarder sous un nom n'existant pas dans le repertoire courant un scénario dont l'état courant peut être en lecture seule ou non.
- `Éditer (Edit)` — fonction modifier — Cet item permet d'éditer un scénario en vue de sa modification. Il fait passer le scénario courant dans l'état lecture/écriture.
- `lanceR (Run)` — fonction exécuter — Cet item permet l'exécution d'un scénario préalablement ouvert.

Ce menu est apparaît à l'intérieur de l'interface graphique du gestionnaire d'activités.

4.3.2 Le scénario

Un scénario définit les attributs et méthodes nécessaires pour la création et à l'exécution d'un scénario. Un scénario est composé d'étapes (figure 4.4).

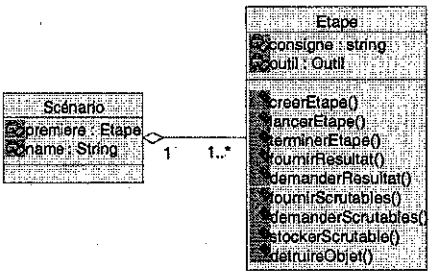


Figure 4.4 – Les classes pour un scénario

Implantation

Comme nous l'avons vu à la section 3.3.3, un graphe comportant des noeuds, des arcs et des barres de synchronisation permet d'exprimer des scénarios complexes. Nous pourrions implanter un tel graphe, à la manière des graphes de Pétri. Cependant, bon nombre d'activités peuvent être décrites comme de simples séquences (on parle, par exemple de séquences pédagogiques). Nous pouvons alors représenter une activité au minimum comme une suite d'étapes. Ici, une liste est un cas particulier d'arbres, qui sont eux-même un cas particulier de graphes. D'où :

$$Scenario ::= \{Etape\ Transition\}+$$

Cette structure séquentielle convient parfaitement dans le cas où les étapes du scénario se succèdent. Cette définition minimale permet de décrire un certain nombre d'activités, mais elle est contraignante. En effet, elle exclut l'exécution «en parallèle» de plusieurs tâches. Nous avons choisi, dans un premier temps, la définition minimale d'un scénario afin de permettre une implantation et un test de l'atelier. Cette première implantation nous permet d'envisager ensuite une structuration plus complexe.

Représentation graphique à l'écran

Pour aider le prescripteur dans la définition d'un scénario, nous proposons une représentation graphique d'une étape sur la figure 4.5. Elle reprend les informations principales concernant

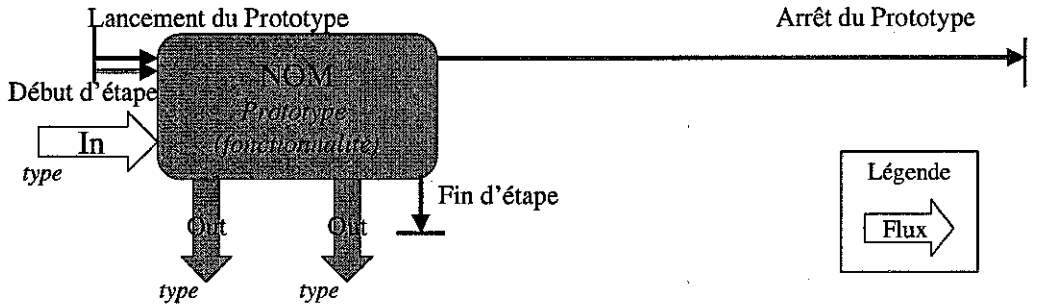


Figure 4.5 – Représentation graphique d'une étape

l'étape. Le lancement et la clôture de l'étape et de l'outil sont marqués par des flèches fines avec butées. Le nom de l'étape, le prototype et la fonctionnalité impliquée sont inscrits dans un rectangle aux coins arrondis. Les flux de données sont représentés par des flèches épaisses étiquetées. Sur cette figure, le temps s'écoule de gauche à droite. Ainsi la butée de fin d'étape est plus à gauche que la butée de fin de prototype car l'étape est finie avant le prototype. L'inverse est impossible.

La représentation graphique d'un scénario est une succession d'éléments correspondants chacun à une étape. Lors de la conception d'un scénario, cette représentation permet de visualiser le début et la fin de chaque étape et de chaque prototype, ainsi que les flux de données.
L'exemple 4.2 présente un extrait d'un scénario issu de l'exemple introductif (page 5).

Exemple 4.2 *Dans notre exemple introductif (page 5),* Mentoniezh *et* CHyPre *sont actifs en même temps. De même,* TALC *et* CABRI-Géomètre *sont actifs en même temps, pour que* TALC *puisse récupérer les objets créés avec* CABRI-Géomètre *et vérifier la conformité de la figure créée relativement à l'énoncé. Cependant, le diagnostic de* TALC *ne peut être lancé que lorsque l'utilisateur a construit une figure avec* CABRI-Géomètre *et avec l'aide de* PACT *et qu'il demande une validation. L'enchaînement de l'étape «tracer figure» avec l'étape «vérifier figure» est présentée sur la figure 4.6 .*

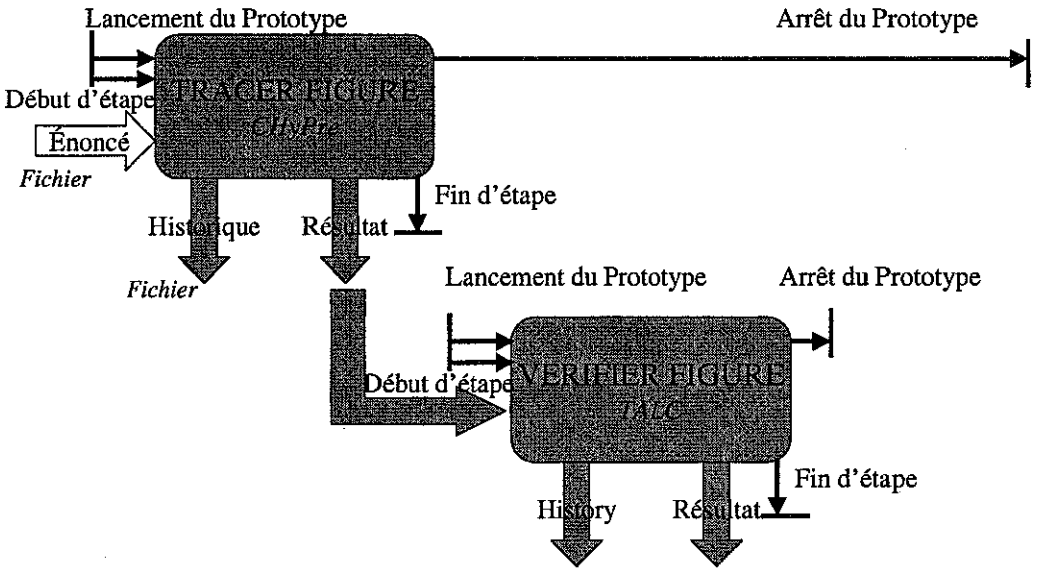


Figure 4.6 – Extrait du scénario de l'exemple introductif

Sur la figure 4.6, l'étape «tracer figure» précède l'étape «vérifier figure». Le prototype *TALC* est fermé dès que l'étape est finie. En effet, une fois que l'étape de diagnostic est terminée et il n'y a pas de raison de garder le prototype en fonctionnement. En revanche, le prototype *CHyPre* lancé pour l'édition de figure n'est pas fermé à la fin de l'étape «tracer figure», ni à la fin de l'étape «vérifier figure» mais ultérieurement (à un moment qui n'apparaît pas sur cet extrait).

Création d'un objet

Créer un scénario, c'est créer les différentes étapes qui le compose. Nous décrivons cette création par l'algorithme 2. Ainsi pour créer un scénario il faut d'abord créer une première étape. L'élaboration du scénario s'effectue alors de manière incrémentale en créant une autre étape et en l'insérant avant ou après une étape existante.

4.3.3 L'étape

Fiche pour caractériser une étape

Nous regroupons toutes les informations nécessaires pour décrire et exécuter une étape dans une **«fiche pour caractériser une étape»**. La fiche vierge est donnée à l'annexe B. Cette fiche constitue un support de discussion utile au prescripteur et l'administrateur, qui ont des vues différentes de l'activité.
La figure figure 4.7 présente cette fiche documentée (chaque champ décrit son contenu). Le

Algorithme 2 Créer un scénario

PRÉREQUIS: les outils sont disponibles dans l'atelier

PRODUIT: un scénario «Scenario sce»

1: EncoreUneEtape = vrai ;
2: **tantque** EncoreUneEtape **faire**
3: /* Définir une étape – (*cf.* algorithme 3) */ creerEtape(out Etape etape) ;
4: /* Insérer l'étape dans le scénario */ insererEtape(in Etape etape, inout Scenario sce) ;
5: /* Demander valeur de EncoreUneEtape */ demanderEtSaisirEncoreUneEtape (out Bouleen encoreUneEtape) ;
6: **fin tantque**

premier cadre (Étape) donne le nom de l'étape. Le second cadre (Consigne) décrit la tâche à accomplir. Le troisième cadre (Outils) décrit avec quoi accomplir la tâche. Le quatrième cadre (Entrées-Sorties du prototype) précise les entrées-sorties récupérables du prototype choisi. Le cinquième cadre (Données à recueillir) précise les données à recueillir au cours de l'activité. Le sixième cadre (Transition) définit la condition pour clôturer l'étape et passer à l'étape suivante. Le septième cadre (Présentation à l'interface) définit la présentation à sur l'interface graphique de l'utilisateur. Nous avons donné un exemple d'utilisation de cette fiche à figure 1.6 p 16.

Implantation

L'idée de l'implantation est de reprendre chaque champ de cette fiche de lui faire correspondre un attribut ou un objet. Nous avons implanté cette fiche en un objet `Etape` et en un objet `Transition` (*cf.* figure 4.8). La transition décrit la condition de terminaison de l'étape. Une transition permet de calculer la fin d'une étape et de désigner l'étape suivante. Elle aurait pu être intégrée dans l'étape pour l'implantation actuelle. Cependant afin de pouvoir évoluer vers l'implantation d'un graphe, nous avons trouvé plus pratique de l'implantée comme un objet indépendant.

Création de l'objet

Nous décrivons cette création par l'algorithme 3.

Exécution une étape

Cette fonction consiste à exécuter étape après étape un scénario (voir algorithme 4). À la fin de l'exécution de cet algorithme, l'atelier lance tous les scripts de fermeture d'outils associés à cette étape. De plus, à la fin de la dernière étape, l'atelier lance tous les scripts de fermeture des outils encore actifs.

La section suivante présente un autre aspect pris en charge par le gestionnaire d'activités : la gestion des données collectées.

Consigne
consigne à donner au sujet

Outils
Fonctionnalité : fonctionnalité informatique avec laquelle le sujet doit effectuer la consigne .....
Prototype : prototype choisi qui propose cette fonctionnalité .....
Moment de fermeture : fin de l'étape ou fin de l'activité ou fin d'une étape donnée .....

Données à recueillir
Résultat produit : oui/non, si oui nom du fichier résultat .....
Connaissances d'interaction : oui/non, si oui lesquelles .....
Autres (informations scrutables) : oui/non, si oui lesquelles .....

Présentation à l'interface

	position Vert	position Horiz	tailleLarg	tailleHaut	déformable	déplaçable	maximisable
Consigne :	haut/bas	gauche/droite	0 – 100%	0 – 100%	oui/non	oui/non	oui/non
Outil :	haut/bas	gauche/droite	0 – 100%	0 – 100%	oui/non	oui/non	oui/non
Transition :	haut/bas	gauche/droite	0 – 100%	0 – 100%	oui/non	oui/non	oui/non

Transition
Étape suivante : le nom de l'étape suivante .....
Condition de transition à l'étape suivante : le sujet signale qu'il a fini, le prototype atteint un état particulier, *etc.* .....

Figure 4.7 – Fiche pour caractériser une étape : fiche documentée

4.4 Le gestionnaire de données

Le gestionnaire de données permet de gérer la collecte des données sauvegardables. Dans la suite, les sauvegardes concernent la collecte des sauvegardables. Dans la définition de l'activité, le prescripteur choisit les sauvegardables mais c'est l'atelier à la fois qui lui propose un formulaire pour les choisir et qui gère les sauvegardes.
Pour ces données nous avons définis une super-classe `Sauvegardable`, spécialisée par les classes `Scrutable`, `Résultat` et `Observable` (*cf.* figure 4.9). L'analyse de l'activité menée dans l'atelier passe par le recueil des informations concernant les interactions du sujet avec chaque outil. Nous avons montré dans le chapitre 3 que dans son interaction avec l'outil, le sujet produit des événements sémantiques que nous structurons en «observables» intégrant une sémantique.

4.4.1 Interprète

Pour associer aux événements sémantiques les observables adéquats nous avons défini une grammaire (page 93).

Principe de l'interprète d'événements sémantiques

Le fonctionnement ici est analogue à celui d'une calculatrice à pile. Les opérandes et opérateurs sont entrés dans un certain ordre. Lorsqu'une opération est possible, les opérateurs et

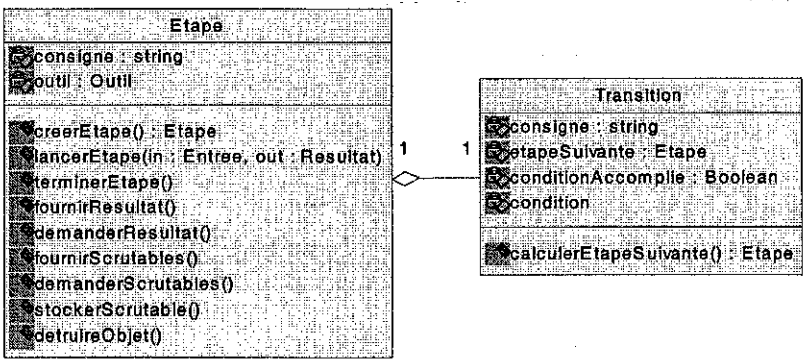


Figure 4.8 – Les classes Etape et Transition

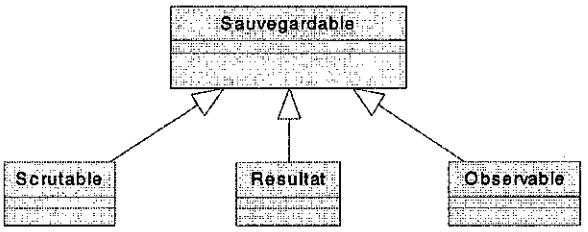


Figure 4.9 – Classes JAVA pour les données sauvegardables

opérandes associés sont dépilés et le résultat empilé. Dans notre cas, les événements sémantiques sont empilés. Ils sont dépilés pour composer un observable. L'observable ainsi composé sont envoyés dans un flux de données afin d'être stocké ou utilisé par un outil. En pratique, ce flux arrive dans une file d'observables. Ainsi à la fin de l'activité, où à la demande, la file d'observables peut être vidée pour stockage ou exploitation par un outil adapté. C'est dans ce dernier cas, qu'une file s'avère plus appropriée qu'une pile.

Consommation des événements sémantiques

Pour illustrer le principe de la consommation[47] des événements sémantiques qui se produit dans la composition des observables, nous donnons un l'algorithme simplifié (*cf.* algorithme 5 p 123) dans le cas où toutes les macro-définitions ne sont composées que d'un seul événement sémantique. L'algorithme 5 est lancé chaque fois qu'un événement sémantique est produit. Il nécessite une pile tampon, pour qu'il n'y ait pas de modification de la pile d'événements sémantiques pendant l'exécution de l'algorithme.
À chaque fois qu'un observable est produit, on essaie de réduire la pile des observables, c'est-à-dire de diminuer le nombre de ses éléments (*cf.* algorithme 6 p 125).
De plus, on vide la pile d'événements sémantiques car il n'y a plus continuité dans la succession des événements. L'exemple 4.3 illustre le problème.

Exemple 4.3 *La pile contient : 1 2. L'élément 3 arrive. L'algorithme 5 est lancé. Il existe une macro-définition : <O ← 2 3>. L'observable O est créé. La pile contient alors : 1. L'élément 4 arrive. La pile contient alors : 1 4.*

[47] Nous parlons de consommation des événements sémantiques pour désigner le fait que des événements sémantiques sont retirés de la pile (c'est-à-dire «dépilés»). Autrement dit, un événement sémantique dépilé est un événement sémantique consommé.

Algorithme 3 Créer une étape

PRÉREQUIS: une liste d'outil «in ListeDOutils lo»
PRODUIT: une étape «out Etape etape»

```
1: /* Saisir le nom de l'étape */ saisirNom(out Nom nomEtape);
2: /* Saisir la consigne */ saisirConsigne(out Consigne cons);
3: /* Choisir l'outil */ choisirOutil(in ListeDOutils lo, out Outil outilEtape)
4: /* Définir moment fermeture de l'outil et type de fermeture */
   parametrerFermetureOutil(out Script fermeture);
5: /* Définir transition */ definirTransition(out Transition transition);
6: /* Définir les flux en entrée */
7: encoreUn = vrai
8: tantque encoreUn faire
9:    definirFluxIn(out FluxEntree nom, out Outil provenance);
10:   ajouterDansListe(inOut ListeFlux lin, in FluxEntree nom);
11:   saisirEncoreUn (out Bouleen encoreUn);
12: fin tantque;
13: /* Définir les flux en sortie */
14: encoreUn = vrai
15: tantque encoreUn faire
16:   definirFluxOut(out FluxEntree nom, out Outil destination);
17:   ajouterDansListe(inOut ListeFlux lout, in FluxEntree nom);
18:   demanderEtSaisirEncoreUn (out Bouleen encoreUn);
19: fin tantque;
20: /* Résultat */
21: Résultat = {Nom nomEtape, Consigne cons, Outil outilEtape, Script fermeture, Transition
    transition, ListeFlux lin, ListeFlux lout}
```

Dans cet exemple, les éléments de la pile sont des chiffres car il n'est pas important de savoir quels événements sémantiques ils représentent. Après la construction de l'observable et l'arrivée de l'élément 4, il serait erroné de pouvoir déduire un observable à partir de la succession des événements 1 et 4. Il faut considérer les éléments à partir de l'arrivée de 4. D'où l'action de vider la pile après la génération d'un observable.

Interprète de macro-définition

Nous implantons la génération des observables dans un interprète de macro-définitions, qui est le même que celui utilisé pour la gestion des formats. Un interprète à pile convient parfaitement pour composer les observables au fur et à mesure. Une méthode spécifique pour l'algorithme donné précédemment est gérée dans l'interprète.

4.4.2 Utilisation des services CORBA

Les résultats, les informations scrutables et les observables sont publiées via la vitrine des outils — dans la fiche de chaque outil, avec leur chemin d'accès et leur type. Pour acheminer les observables créés par les différents outils, nous utilisons les services «Événement» et «Notification» de CORBA.

Algorithme 4 Exécuter un scénario

PRÉREQUIS: un scénario Scenario sce
PRODUIT:

```
1: /* Initialisation par rapport au scénario */
2: sePositionnerAuDebut(in Scenario sce);
3: /* Traitement des étapes */
4: tantque Bouleen scenarioNonFini(in Scenario sce) faire
5:    /* Initialisation par rapport à l'étape */
6:    lireEtapeCourante(in Scenario sce, out Etape etape);
7:    lireConsigne(in Etape etape, out Consigne consigne);
8:    lireOutil(in Etape etape, out Outil outil);
9:    lireTransition(in Etape etape, out Transition transition);
10:   /* Réalisation de l'étape */
11:   afficherConsigne(in Consigne consigne);
12:   lancerOutil(in Outil outil, in Script script);
13:   afficher(in Transition transition);
14:   boucler
15:      attendreNotificationTransition(in Transition transition);
16:   fin boucle
17:   calculerEtapeSuivante(inout Scenario sce, in Transition transition);
18: fin tantque
```

4.4.3 Adaptation d'un prototype ...

Pour un prototype qui n'est pas scrutables ou n'est pas traçable, il faut ajouter une couche logicielle pour l'adapter.

... qui ne génère pas d'événements sémantiques

Lorsque le prototype ne produit pas d'événements sémantiques, il est parfois possible de les calculer en utilisant un **logiciel épiphyte** adapté. Par analogie avec la plante épiphyte en biologie, un logiciel épiphyte est un logiciel qui se greffe sur un logiciel existant, afin de recueillir les données dont il a besoin et ce, sans gêner le fonctionnement du second logiciel (sinon le logiciel serait parasite).
De nombreux environnements de développement de logiciels permettent de gérer directement les événements d'interfaces pour générer des observables. Ils permettent aussi de récupérer ces événements via des outils qui permettent d'«espionner» une application en interceptant les événements. Ces outils sont appelées «crochet» *Hook*, «espion» *spy* ou «vampire». Nous avons utilisé l'un deux (winsight) dans nos exemples lors de la définition des concepts d'événements sémantiques et d'observables à la section 3.4.1. Il existe aussi des bibliothèques (principalement en C) pour implanter soi-même ces outils. Par exemple, *SIMPLE*[48] de l'université de Erlangen-

[48] http ://www7.informatik.uni-erlangen.de/tree/IMMD-VII/Research/Groups/MMB/simple/

Algorithme 5 Consommation des événements sémantiques (algorithme simplifié))

PRÉREQUIS: une pile d'événements sémantiques non vide (nommée eve)

PRODUIT: la pile eve vide

PRODUIT: alimente une pile d'observables (nommée obs)

```
1: modif=vrai,
2: tantque modif and pileNonVide(eve) faire
3:    /* Lire un événement sémantique dans la pile */ eveSem := dePiler(eve);
      modif := faux;
4:    pour chaque MacroDefinition faire
5:       /* Lire le corps de la macro-définition : l'événement sémantique */
         eveSemBut := macroDef.Corps();
6:       si unification(eveSem ;eveSemBut) alors
7:          modif :=vrai;
8:          /* ajouter la tete de la macroDef unifiée dans la pile obs */
            emPiler(macroDef.Tete(),obs);
9:          /* Gérer les MacroObservables */ reduireObservables(obs);
10:         viderPile(eve);
11:         break;
12:      finsi
13:   fin pour
14:   si non(modif) alors
15:      /* Remettre la pile en état avant de sortir de la boucle */
         enPiler(eveSem);
16:   finsi
17: fin tantque
```

Nuremberg en Allemagne ou TracePlus[49] de SST ou encore, *vtrace*[50]
Cédric Haumont a implanté ce type de logiciel pour notre équipe [Haumont 98, Desmoulins 98] dans le cadre d'un DEA. Il l'utilise pour connaître les interactions d'un apprenant avec une interface graphique qui permet de manipuler un équipement électronique (générateur de signal, multimètre, oscilloscope, *etc.*) dans le cadre de travaux pratiques dans un enseignement à distance (les apprenants ont des travaux pratiques à faire chez eux, l'équipement leur est fourni).

... qui ne génère pas de scrutables

Dans le cas des prototypes qui n'ont pas d'informations scrutables, il est possible, si l'on dispose du code, d'ajouter un accès aux informations nécessaires pour l'exploitation de l'activité. Une autre possibilité est offerte par les prototypes qui disposent d'un langage de script ou de programmation. Dans ce dernier cas, il est possible d'écrire le script ou le programme pour accéder

[49] http ://www.sstinc.com/home.html
[50] http ://ifrit.cs.Berkeley.edu/~lorch/vtrace/

aux variables qui sont nécessaires pour l'exploitation de l'activité. Selon les caractéristiques du prototype existant, nous utilisons l'une ou l'autre de ces deux possibilités. Par exemple, le logiciel Excel de Microsoft© dispose d'un langage de programmation (VisualBasic) qui permet de le configurer pour un usage éducatifs. Ce langage de programmation permet en particulier de scruter les variables.

Algorithme 6 Réduction de la pile d'observables (algorithme simplifié)

PRÉREQUIS: une pile d'observable non vide (nommée obs)
PRODUIT: une pile d'observable identique ou de taille inférieure

```
1: modif :=vrai
2: /* Tant qu'on a fait une modification dans la pile d'observables, on essaie de réduire encore
   la pile */
3: tantque modif faire
4:   /* Modif sera remis à vrai pendant l'algo, si on a modifié quelque chose, sinon on sortira
     de la boucle */ modif := faux
5:   /* Lire un observable dans la pile */ obsLu := dePiler(obs);
6:   pour chaque macroObservable faire
7:     /* Lire le dernier observable du corps du macroObservable */
       obsDer := macroObs.dernierElement();
8:     /* S'il existe une macroObservable ayant cet observable comme fin */
9:     si unification(obsLu ;ObsDer) alors
10:      /* Chercher à unifier avec la suite de la Pile */
11:      /* — combien le macroObservable a-t-il d'observables ? */
         nbObs := macroObs.corps.nbObs();
12:      /* — traiter les nbObs-1 autres observables du macroObs ? */ marche := vrai;
13:      pour i :=1 to nbObs-1 faire
14:        /* Réserver l'observable la pile provisoire proviPile */ emPiler(obsLu,proviPile);
15:        /* Lire observable suivant */ obsLu=deFiler(obs);
16:        /* Faire l'unification avec le nbObs-i ièmes observable */
           /* ça marche : on continue le tour – rien à faire de spécial */
17:        si not(unification(obsLu ;macroObs.corps.element(nbObs-i))) alors
18:          /* ça ne marche pas : inutile d'aller voir les autres observables du macroObser-
             vable */ marche := faux; break;
19:        finsi
20:      fin pour
21:      /* Si marche est à vrai c'est qu'on a empiler les nbObs observables */
22:      si marche alors
23:        /* On enPile la macroObs, on vide la pileProvi, et on recommence */
           enPiler(macroObs.Tete(),obs); viderPile(pileProvi); modif := vrai; break;
24:      sinon
25:        /* On remet le contenu de la pile pileProvi dans la pile obs */
           enPiler(dePiler(pileProvi),obs);
26:      finsi
27:    finsi
28:  fin pour
29: fin tantque
```

4.5 Le langage de commandes

EduMed est le médiateur qui permet aux outils de travailler ensemble grâce au chef d'orchestre que constitue le gestionnaire d'activités. Pour agir sur les différents outils, le gestionnaire d'activités utilise des commandes. Ces commandes sont matérialisés par des méthodes implantées dans les serveurs CORBA. Elles sont prises en charge par EduMed. EduMed déploie en effet un ORB qui est le pivot reliant les outils et le gestionnaire d'activités entre eux. Les commandes sont présentées à la section 3.5.
Les commandes sont implantées en tant que méthodes des objets correspondants. Par exemple, pour la méthode de lancement de l'outil *CHyPre*, la vitrine de cet outil comprend une méthode `launch()`. Cette méthode `launch()` correspond à une spécification dans la vitrine de cet outil.
Nous n'avons implanté que les commandes présentées dans le tableau 3.3 afin de montrer la faisabilité de notre approche. Pour ajouter une commande, il suffit de modifier la vitrine des outils concernés et de modifier les fichiers générés pour la correspondance[51] IDL vers JAVA.
Cette méthode pour ajouter les commandes est simple. Elle permet un langage de commandes ouvert. Il est en effet facile d'ajouter des commandes si nécessaire.

4.6 Le gestionnaire de formats

Le gestionnaire de formats est un composant de services utilisé dans l'atelier, pour acheminer les connaissances aux prototypes dans le format approprié. Il est implanté par l'objet `gesFormats`. Dans l'état actuel du développement, nous gérons les connaissances du domaine grâce à un interprète de macro-définitions présenté à la section 3.6.

4.6.1 L'interprète

L'interprète de macro-définitions est une spécialisation du gestionnaire de formats (*cf.* figure 4.10). Les classes de l'interprète sont présentées à la figure 4.11. Le diagramme de séquence

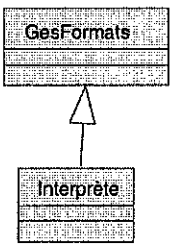


Figure 4.10 – Classes du gestionnaire de formats

de la figure 4.12 présente l'utilisation de l'interprète pour effectuer une traduction. En notes sur la figure 4.12 sont présentés les éléments correspondant à l'exemple 4.4.

[51] *binding*

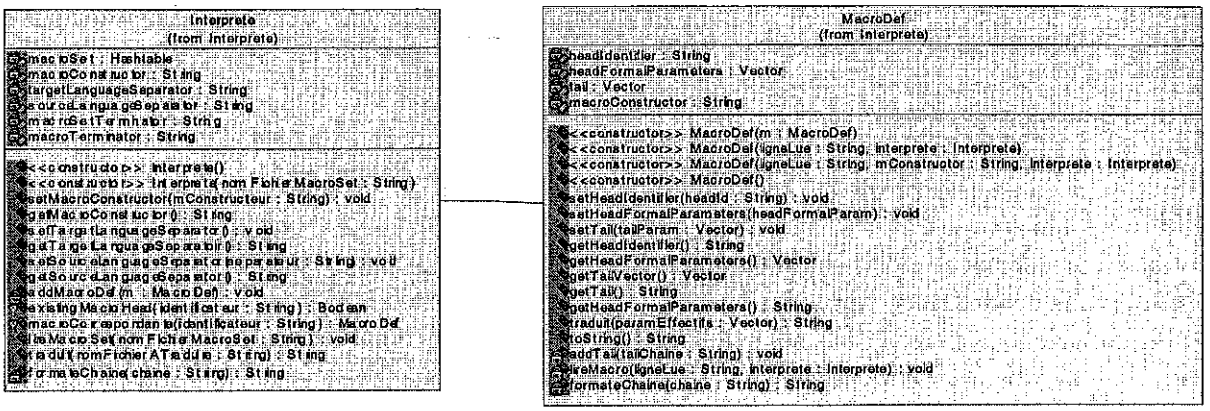


Figure 4.11 – Classes de l'interprète

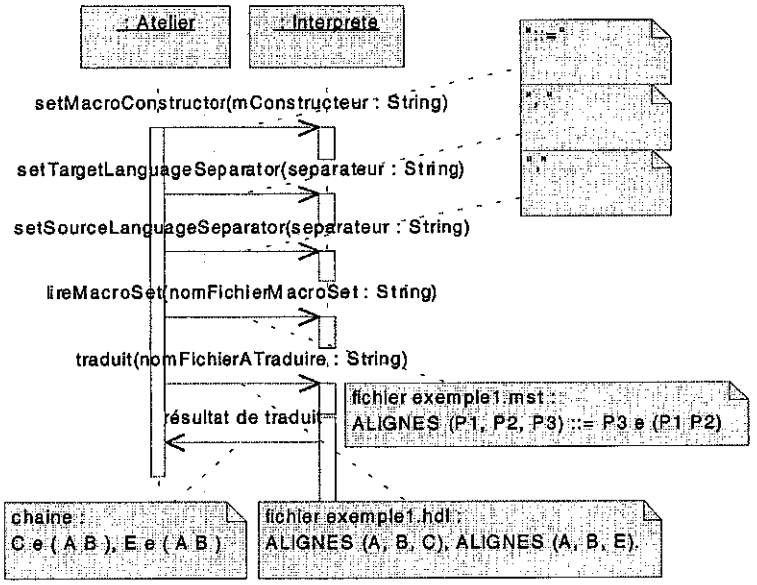


Figure 4.12 – Exemple de traduction

Exemple 4.4 *Le fichier contenant l'énoncé à traduire est* `fichier1.hdl` *:*

```
ALIGNES (A, B, C), ALIGNES (A, B, E).
```

Le fichier contenant les macro-définitions est `fichier1.mst` *:*

```
ALIGNES (P1, P2, P3) ::= P3 e (P1 P2)
```

Le constructeur de macro définition est ::=
Le séparateur d'atomes dans le langage cible est une virgule suivie d'un espace. Le séparateur d'atomes dans le langage source est une virgule. Le terminateur de macro-définitions est un point.

La méthode `lireMacroSet()` consiste à lire le fichier en entrée et à construire une macro-définition pour chaque atome. Le résultat de l'application de la méthode `lireMacroSet()` sur la macro-définition de l'exemple 4.4 donne la table 4.1.

Les macro-définitions ainsi construites sont stockées dans une table de hashage appelée `macroSet` dans l'interprète.

Objet	Nom	Valeur
String	headIdentifier	`ALIGNES`
Vector	headFormalParameters	`[P1, P2, P3]`
Vector	tail	`[3, , e, , (, , 1, , 2, , )]`

Table 4.1 – Exemple de lecture de Macro-définitions.

La méthode «`traduit`» de l'objet `Interprete` consiste à lire le fichier en entrée (l'énoncé), atome par atome et à construire une chaîne contenant le résultat de la traduction. Pour chaque atome correspondant à une macro-définition, il recherche la macro correspondante, puis fabrique le vecteur recevant les paramètres effectifs. Pour le premier atome `ALIGNES (A, B, C)`, il existe une macro-définition correspondant. Les paramètres effectifs sont : `[A, B, C]`. Ensuite, il passe ces paramètres effectifs à la méthode «`traduit`» de l'objet `MacroDef` construit. L'algorithme de cette méthode présente le principe de la traduction (*cf.* algorithme 7 p 128). Pour le premier atome de

Algorithme 7 Méthode «`traduit`» de la classe `MacroDef`

PRÉREQUIS: (paramètre) un vecteur contenant les chaînes des paramètres effectifs : Vector paramEffectifs

PRÉREQUIS: (attribut local) un vecteur contenant la queue de la macro : Vector tail

PRÉREQUIS: (à savoir) le i ième élément d'un Vector est à l'indice i-1

PRODUIT: la traduction : String stringTail

```
1: /* Initialisations */ String stringTail = new String(""); int i, indiceParam;
2: pour i=1;i<=tail.size();i++ faire
3:   /* Pour chaque élément de la tête */
4:   si tail.elementAt(i-1) instanceof String alors
5:     /* l'élément n'est pas un paramètre — on le copie dans le résultat */
       stringTail = stringTail.concat(tail.elementAt(i-1).toString());
6:   sinon
7:     /* L'élément est un entier qui désigne la place du paramètre effectif dans le vecteur
       paramEffectif */
8:     si tail.elementAt(i-1) instanceof Integer alors
9:       indiceParam = tail.elementAt(i-1);
10:      /* On copier le paramètre effectif correspondant dans le résultat */
         stringTail = stringTail.concat(paramEffectifs.elementAt(indiceParam-1).toString());
11:    sinon
12:      /* traitement d'exception */ ...
13:    finsi
14:  finsi
15: fin pour
```

l'exemple 4.4 (ALIGNES (A, B, C)), l'algorithme 7 construit une chaîne composée par la concaténation des chaînes. Ces chaînes sont recopiées depuis le corps de la macro-définition (tail). Si

I	(i-1) ième élément du vecteur tail	Élément ajouté à la chaîne résultat
1	3	C
2	*espace*	*espace*
3	*espace*e	*espace*e
4	*espace*	*espace*
5	*espace*(	*espace*(
6	*espace*	*espace*
7	1	A
8	*espace*	*espace*
9	2	B
10	*espace*	*espace*
11	*espace*)	*espace*)

Table 4.2 – Exemple d'application de la méthode «codetraduit»

l'élément du corps est une chaîne il est recopié tel quel. Si c'est un nombre, il correspond à un paramètre effectif par position. L'application de la méthode «`traduit`» sur la macro-définition de l'exemple 4.4 donne la table 4.2.
La chaîne résultat est donc : `C e ( A B )`.

Aspects techniques

Pour implanter l'interprète, nous avons étudié l'implantation de traducteurs [Bolc 87, Pollack 72, Yellin 88]. L'implantation de l'interprète de macro-définition a d'abord été écrite en Prolog. En effet, au début du projet, nous avons montré l'intérêt de cette notion (macro-définition) dans le domaine de la géométrie pour les logiciels *TALC* et *Mentoniezh* (voir annexe C). *TALC* et *Mentoniezh* ayant été développés sous Prolog sur Macintosh, il était plus aisé d'implanter l'interprète en Prolog. Cependant, afin de pouvoir l'utiliser avec d'autres outils écrits dans d'autres langages et sur d'autres plate-formes matérielles, nous avons cherché à utiliser un interprète de Prolog écrit en JAVA[52], que nous avons déployé. Ce test a été effectué sur PC, sur station unix et sur console JAVA[53] Cette solution s'est avérée désastreuse (beaucoup trop lente). En effet, nous nous retrouvons alors avec trois couches successives d'interprétation : l'interprétation des macro-définitions, via l'interprétation des prédicats Prolog, via l'interprétation du code source dans la machine virtuelle JAVA. Finalement l'interprète de macro-définitions a été réécrit directement en JAVA.

4.6.2 Fichiers

Le tableau 4.3 liste les fichiers JAVA nécessaires pour l'implantation de l'interprète de macro-définition.

Nom	**Description**

[52]http ://compilers.iecc.com/comparch/article/99-11-101

[53]Nous avions obtenu une console java pour le test suite à une demande argumentée, le laboratoire disposant d'une douzaine de machines fournies pas SUN. J'avais passé beaucoup de temps à la configurer. Cependant, au retour d'un déplacement, la machine avait été récupérée. Cela m'a conduit à abandonner les test sur cette console JAVA.

Interprete.class	implante l'interprète
MacroDef.class	implante les macro-définitions
Entree.class	implante les entrées depuis le clavier ou un fichier
EntreeException.class	implante les exceptions lancées par la classe précédente

Table 4.3: Fichiers JAVA pour l'interprète

Les classes `MacroDef` et `Interprete` sont incluses dans un package `Interprete`. Les classes `Entree` et `EntreeException` sont incluses dans un package `IO`. Ce dernier est un package utilitaire, qui regroupe toutes les fonctions permettant de gérer les entrées et les sorties à partir de fichiers ou des entrées et sorties standards. Il est utilisé dans le cas de l'interprète, pour lire des lignes, pour lire des atomes, ... dans les fichiers d'énoncé et de macro-définitions.

Dépendances

Le tableau 4.4 donne les dépendances entre les classes nécessaires pour l'implantation de l'interprète de macro-définitions.

Nom de la classe	**Classes utilisées**
Interprete	MacroDef Entree EntreException
MacroDef	Interprete
Entree	EntreeException
EntreeException	.

Table 4.4: Dépendances des classes JAVA pour l'interprète

4.6.3 Interfaces graphiques

L'interprète est manipulable par des commandes en ligne ou placées dans un fichier de script. Cependant, une interface graphique permet de manipuler l'interprète d'une manière plus conviviale. La figure 4.13 présente les menus de cette interface graphique [Henninger 00][54].
Le menu `Fichier` permet de sauver ou de charger une configuration et de quitter l'interface graphique de l'interprète. La configuration contient tous les paramètres de configuration de l'interprète. Il s'agit du terminateurDeMacroTexte, du constructeurDeMacroDefinition, du terminateurDeMacroDefinition et du separateurDePhraseDuLangageDestination.
Le menu `MacroDef` permet de positionner les paramètres liés aux macro-définitions.
Le menu `Interprète` permet de positionner les paramètres liés à l'interprète et de lancer l'interprétation. L'interprétation consiste à traduire l'énoncé du langage source vers le langage cible en tenant compte de tous les paramètres positionnés.
Le menu `Option` permet de choisir la langue de présentation de l'interface graphique. Il permet aussi de choisir les informations à tracer : les actions de l'utilisateur sur cette interface graphique

[54]Cette interface graphique a été développée par Sandrine Soudant et Danny Henninger, étudiants de maîtrise au cours d'un projet d'initiation à la recherche que j'ai encadré.

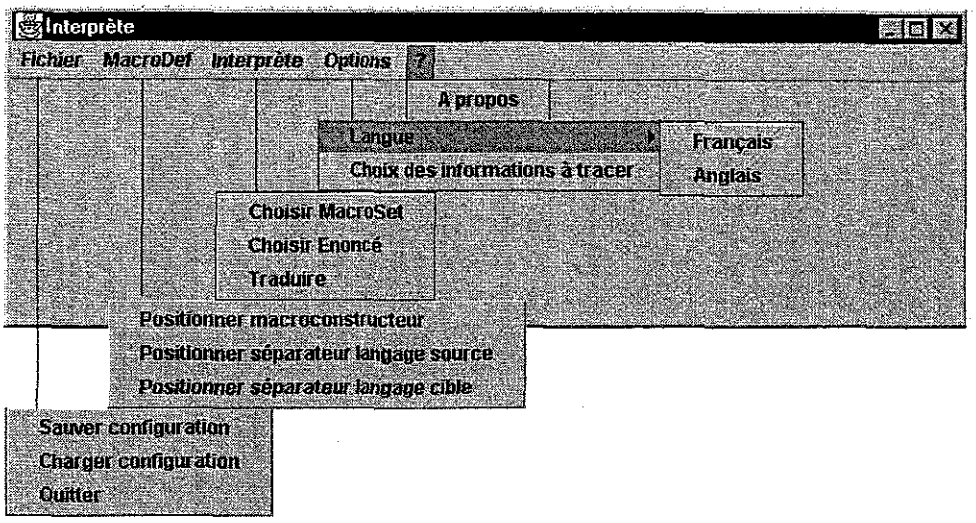


Figure 4.13 – Menus de l'interface graphique de l'interprète de macro-définitions

peuvent être sauvegardées. Enfin le ? permet d'accéder aux informations concernant l'interprète et son interface graphique.

4.6.4 Composant de service de l'atelier

Pour transformer cet interprète en composant de services, nous avons défini sa vitrine :

`In_statement`
`Out_statement`
`Macro_text`
`Interpret()`

Le composant CORBA correspondant est accessible via EduMed. C'est le gestionnaire d'activités qui fait appel à l'interprète (via le gestionnaire de formats) via EduMed lorsqu'un scénario d'activité nécessite la conversion de formats de connaissances. C'est lors de la définition d'une activité qu'est déterminé le besoin de conversion. C'est lors de l'exécution de l'activité qu'a lieu cette conversion, plus précisément au début de l'étape qui nécessite le fichier converti. Pour créer le composant `Interprete`, nous écrivons tout d'abord la vitrine de ce composant dans le fichier `Interprete.idl`. Le `Interprete` est un composant qui prend en entrée un énoncé `In_statement` et un macro-texte `Macro_text` et produit un énoncé traduit en sortie `Out_statement`. La traduction de cet énoncé est lancée par la méthode `interpret()`. Nous obtenons par conséquent le fichier suivant :

```
String In_statement
String Out_statement
String Macro_text
void interpret(void)
```

Nous faisons abstraction ici des méthodes et attributs à créer pour l'exportation de l'interface graphique du gestionnaire de formats.

4.7 La gestion de l'interface graphique

Au cours d'une activité, l'atelier manipule les interfaces graphiques de différents prototypes. Toutes ces interfaces graphiques ne sont pas utiles pendant tout le temps de l'activité. Notre but est d'afficher sur un écran unique (l'écran hôte sur lequel l'activité a lieu) toutes les fenêtres (ou une parties des fenêtres) issues des différents prototypes. C'est, par exemple, ce que permet un terminal X (appelé TX). TX est une station de travail où des logiciels Unix tournant sur un serveur peuvent être utilisés. Les éléments graphiques nécessaires à l'utilisateur pour interagir avec le logiciel sont affichés sur l'écran du TX. Notre but est d'utiliser ce type de terminal sur la machine hôte du prescripteur, pour gérer les différentes interfaces graphiques produites par les différents prototypes. Cette gestion nécessite de sélectionner ce qui doit être présenté à l'écran et d'organiser l'écran de manière à minimiser la charge cognitive nécessaire à l'utilisateur pour suivre l'activité. Cependant ce type d'interface graphique virtuelle n'existe pas encore. De plus, elle nécessite un travail de recherche conséquent, qui dépasse le cadre de ce travail.
Pour résoudre notre problème, nous utilisons un gestionnaire d'interfaces graphiques qui permet de définir une fenêtre[55] englobant les interfaces graphiques que nous importons des prototypes. Pour les inclure dans notre fenêtre englobante, nous avons besoin de composants qui exportent leur interface graphique. C'est pour cela que nous avons défini une propriété des interfaces graphiques : elles doivent être interface-exportable (section 1.2.3). Dans ce cas, nous affichons l'interface graphique du composant sur l'écran du sujet de l'activité. Ainsi, le sujet peut agir directement sur son écran comme s'il était sur le logiciel distant.
Rappelons que cette propriété est une forme de paramétrisation de l'interface graphique : elle permet par exemple de lancer l'interface graphique à l'endroit désiré. Dans le cas où nous avons un prototype ayant cette propriété, nous utilisons un composant de services pour gérer l'interface graphique du prototype au sein d'une activité. Ce composant de services est appelé gestionnaire d'interfaces graphiques (voir section 4.7.1). Notre but était de spécifier notre besoin, étant donné que nous ne possédions pas de prototypes ayant cette propriété. Pour cela, nous avons utilisé des solutions *ad hoc* afin de tester notre atelier avec les prototypes existants (voir section 4.7.2). Ces solutions utilisent la spécification du gestionnaire d'interfaces graphiques. Cela nous permet de changer la solution adopté en modifiant le moins possible le reste de l'atelier.

4.7.1 Le gestionnaire d'interface graphique

Le gestionnaire d'interface graphique est un composant de services permettant de gérer les interfaces des différents prototypes lancés. Nous considérons ici les prototypes ayant la bonne propriété «interface graphique exportable». Puisque les prototypes peuvent exporter leur interface graphique, le gestionnaire d'interfaces graphiques les importe. Il doit alors pouvoir les positionner dans l'espace de l'écran et dans le temps de l'activité. Nous avons par conséquent défini les fonctionnalités du gestionnaire d'interfaces graphiques en fonction de la vitrine dont nous avions besoin. La vitrine du gestionnaire d'interfaces graphiques est donnée ci-après :

```
AffectIDWindow()
CreateIncludingWindow()
ActivateWindow()
DesactivateWindow()
OrganizeWindow()
```

[55] Cette fenêtre englobante est une première étape vers une interface virtuelle plus élaborée

L'interface graphique d'un prototype étant incluse dans une fenêtre, nous donnons à cette fenêtre un identificateur grâce à la méthode `AffectIDWindow()`. Ensuite nous pouvons insérer cette interface graphique dans une fenêtre qui l'inclut : cette fenêtre est celle qui est positionnée à l'écran du poste où se déroule l'activité. Ceci se fait par la méthode `CreateIncludingWindow()`. Ceci fait, il est possible d'activer, de désactiver et de positionner la fenêtre en fonction des besoins de l'activité grâce aux méthodes `ActivateWindow()`, `DesactivateWindow()` et `OrganizeWindow()`.

4.7.2 Solutions *ad hoc*

N'ayant pas de prototypes ayant les propriétés adéquates, nous avons recherché une solution *ad hoc* dans deux cas. Le premier cas est celui de l'expérimentation de *TALC* et *Mentoniezh*. Le second cas est celui de l'expérimentation des autres prototypes dont nous disposions. La dernière solution est la plus générale.

Première solution

Le premier besoin de gérer les interfaces graphiques est apparu lorsque nous avons voulu faire coopérer *TALC* et *Mentoniezh* tournant sur Macintosh. A l'époque les notions de scénario d'expérimentation et de médiateur n'avaient pas encore été conçues. Notre but étant de montrer la possibilité d'utiliser un interprète de macro-définitions, nous avons choisi d'implanter une première plate-forme sur la même machine (un même Macintosh). *TALC* et *Mentoniezh* étaient alors lancés en même temps et la gestion des interfaces graphiques se faisait manuellement. Nous avons ajouté des boites de dialogues pour donner des consignes à l'utilisateur et pour recevoir des messages de l'utilisateur en particulier pour qu'il signale qu'il a fini une étape.

Cette solution est pratique pour le test. Cependant elle n'est pas envisageable lors d'une expérimentation avec des sujets. En effet, trop de fenêtres sont actives en même temps et le sujet ne sait pas celle qu'il doit prendre en compte à un moment donné sans l'aide d'un guide humain. L'affichage de boites de dialogues pour l'y aider n'était pas suffisants. Cette solution a en outre l'inconvénient de restreindre l'usage des prototypes tournant sur la même machine : soit Macintosh, soit PC, soit Unix.

Solution plus générale

Choix d'un logiciel Pour pouvoir utiliser des prototypes tournant sur des machines différentes, nous avons étudié les logiciels existants, dont *VNC* [VNC http] et *pcAnywhere*[56]. Ces logiciels permettent de visualiser l'écran d'un ordinateur sur un autre et d'agir indifféremment de l'un ou de l'autre. Pour le *pcAnywhere*, comme son nom l'indique, c'est uniquement l'écran d'un PC qui est visualisé. En revanche, il offre des fonctionnalités intéressantes, que n'offre pas *VNC*, en particulier la possibilité de transférer des données entre le PC recevant l'écran et celui l'émettant. C'est pourquoi, dans une activité utilisant des prototypes tournant exclusivement sur PC, l'utilisation de *pcAnywhere* est très intéressante. Cependant, nos prototypes peuvent avoir des systèmes d'exploitations différents. Par conséquent, nous avons utilisé le logiciel *VNC*.

Notre utilisation de *VNC* L'exemple 4.5 illustre une utilisation possible de *VNC*.

[56] © 1995-2001 Symantec Corporation.

Exemple 4.5 *Nous lançons d'une part* CABRI-Géomètre[a] *sur un Macintosh et nous configurons ce Macintosh en tant que serveur* VNC. *Nous lançons d'autre part* Mentoniezh *sur un PC et nous configurons ce PC en tant que serveur* VNC. *Le Macintosh et le PC sont reliés par un réseau*[b] *à une station Unix. La station Unix est configurée en tant que client* VNC. *Nous lançons l'atelier sur la station Unix. Nous pouvons alors visualiser* CABRI-Géomètre *et* Mentoniezh *sur la station Unix et agir sur l'un et l'autre comme si nous étions sur le PC et sur le Macintosh.*

[a]Pour présenter le principe de *VNC* ici, nous ne parlons pas ici des modifications apportées aux prototypes pour fonctionner avec CORBA

[b]intranet ou internet

Cette solution a l'inconvénient de mobiliser trois machines dans le cas de l'exemple 4.5. Cependant elle a l'avantage de ne pas nécessiter d'installer sur la station Unix un émulateur PC pour lancer *Mentoniezh* et un émulateur Macintosh pour lancer *CABRI-Géomètre*. L'intérêt de *VNC* par rapport à d'autres produits existants à l'époque de l'expérimentation est double. D'une part, *VNC* est gratuit. D'autre part, *VNC* permet toutes les combinaisons possibles : la machine cliente (qui reçoit les interfaces graphiques) peut être un PC[57], un Macintosh ou une station Unix, tandis que la machine «serveur» peuvent aussi être un PC, un Macintosh ou une station Unix.

Ce paragraphe achève la description de l'implantation de nos composants. Les paragraphes suivant décrivent la mise en œuvre de l'atelier.

4.8 Mise en œuvre : ajouter un composant

Les prototypes sont des Logiciels Éducatifs (existants ou futurs). Les Logiciels Éducatifs existants n'ont pas été conçus pour coopérer dans l'atelier. Nous avons résumés dans la section 1.2.4, les propriétés nécessaires à un prototype pour coopérer dans l'atelier.
Les prototypes existants ne possèdent pas toutes les propriétés minimales. C'est pourquoi, nous présentons des propositions pour modifier les prototypes existants. Nous proposons en particulier d'encapsuler les prototypes existants dans une surcouche implantant les propriétés nécessaires. Cette encapsulation est présentée dans la section 4.8. Elle permet en particulier d'assurer la communication avec EduMed et, dans la mesure du possible, de rendre le prototype scriptable, scrutable, traçable et interface-exportable par lui. Dans cette section, nous expliquons comment ajouter un composant à l'atelier. Nous manipulons deux types de composants : les composants de services et les composants constitués autour d'un prototype.
Les composants de services sont actuellement au nombre de deux : le gestionnaire de formats et le gestionnaire d'interfaces graphiques. Nous avons montré dans la section 4.6, l'intégration de l'interprète de macro-définitions en tant que composant CORBA. L'intégration de tout nouveau composant (de services ou prototype) suit le même processus. La section 4.8.1 décrit ce processus. Cependant, l'intégration d'un prototype existant nécessite un travail préparatoire que nous présentons dans la section 4.8.2.

4.8.1 Intégration d'un nouveau composant

Nous expliquons ici le portage CORBA d'un composant écrit dans un langage comptatible CORBA. Nous avons illustré le processus de portage CORBA avec l'interprète de macro-

[57] *VNC* marche même avec des applications DOS. C'est très intéressant dans notre domaine puisqu'il y a encore d'anciens prototypes tournant sous DOS.

définitions dans un composant appelé `gestionnaire de formats` vu à la section 4.6.
Notre travail consiste d'abord à décrire la vitrine du composant. La vitrine du composant rassemble tous les éléments que le composant peut publier. Il peut publier tout ou partie des informations accessibles depuis une autre application. Ces informations sont des données (variable, états, observables , *etc.*) et des traitements (fonctions, procédures, méthodes, *etc.*).
Dans cette section, nous supposons que le composant a toutes les «bonnes propriétés», c'est-à-dire qu'il est scriptable, scrutable, traçable et interface-exportable. Pour les composants qui n'auraient pas ces «bonnes propriétés», voir la section suivante.

Rédaction d'une vitrine

La vitrine est d'abord exprimée en langage naturel par le prescripteur. L'administrateur définit alors la vitrine.
Nous créons le composant pour le prototype *CHyPre*. Alors la vitrine pour ce prototype est stockée dans le fichier : `ChypreC.idl`[58].

```
module ChypreCApp {
  interface ChypreC
  {
    ...
  };
};
```

Le fichier `ChypreC.idl` ci-dessus définit un module nommé ChypreCApp et une interface nommée ChypreC. L'administrateur complète les «... » en fonction des caractéristiques du prototype *CHyPre*. Ce travail est actuellement réalisé à la main. Cependant il est partiellement automatisable.

Génération des fichiers CORBA

À partir de la vitrine écrite, le compilateur approprié génère la correspondance vers le langage choisi. La commande `idl2java Chypre.idl` génère les fichiers nécessaires pour CORBA. Ces fichiers[59], dans notre cas, sont au nombre de cinq :

- `_ChypreCImplBase.java`;
- `_ChypreCStub.java`;
- `ChypreC.java`;
- `ChypreCHelper.java`;
- `ChypreCHolder.java`.

Ils sont placés dans un répertoire portant le nom du module défini dans la vitrine (ici ChypreCApp). Ce répertoire constitue un package JAVA.
Ces fichiers constituent des coquilles vides auxquelles l'administrateur ajoute les liens avec *CHyPre*. Il compile ensuite tous les fichiers correspondants au nouveau module CORBA. Il dispose alors d'un client et d'un serveur CORBA pour *CHyPre*. Le composant (`ChypreC`) est alors prêt pour être utilisé par l'atelier.

[58] Règle de nommage : nom du prototype auquel on accole un C (pour composant CORBA)
[59] Les noms de fichiers sont fonctions de l'implantation.

Utilisation du composant encapsulé

Chaque prototype est inclus dans un objet CORBA, en tant que serveur. Lorsque le gestionnaire d'activités est lancé, tous les objets CORBA sont lancés. Lorsqu'un objet CORBA est lancé, il initialise les services CORBA. Ce faisant, une «implantation» de chaque objet est créée. Par exemple, au lancement de l'objet CORBA encapsulant le prototype *CHyPre*, il est initialisé et une implantation de l'objet `ChypreC` est créée. Nous l'appelons `ChypreC_Impl`. À partir de ce moment là, `ChypreC_Impl` attend une requête du client. Le client dans notre cas est le gestionnaire d'activités. Quand le gestionnaire d'activités est lancé, il initialise à son tour les services CORBA. Ensuite il «établit le lien» (*bind*, en anglais) avec chaque objet CORBA. On dit alors que l'objet est **lié**. Par exemple, il établit le lien avec l'objet *`CHyPre`*. À partir de ce moment là, un proxy est créé pour chaque objet CORBA «lié». Par exemple, le proxy pour *CHyPre* est créé. Nous l'appelons *`CHyPre`*`_Proxy`. Lors du lancement d'une étape où le prototype *CHyPre* est lancé, le gestionnaire d'activités utilise le *CHyPre*_Proxy via sa vitrine de cette manière : *CHyPre*_Proxy.launch(), c'est-à-dire qu'il lance la méthode launch(). Le lancement d'une méthode est exprimé dans le paradigme objet. Remarquez que le gestionnaire d'activités utilise le proxy *CHyPre*_Proxy de la même manière qu'il utilise un objet local. Ensuite, CORBA provoque le comportement du proxy *CHyPre*_Proxy. Il fait la requête à l'implantation *`CHyPre`*`_Impl`. *`CHyPre`*`_Impl` lance alors le code associé à la méthode «launch». Lorsque la méthode «launch» se termine, elle retourne une valeur. Cette valeur est renvoyée au proxy *CHyPre*_Proxy qui la reçoit. Ensuite le proxy *CHyPre*_Proxy renvoie ce résultat au gestionnaire d'activités.

Préparer la récupération de résultat d'un prototype

1: **pour** chaque prototype **faire**
2: **pour** chaque fonctionnalité **faire**
3: /* Administrateur saisit le type du résultat (flux, fichier, string) */
4: /* Administrateur saisit le chemin d'accès du résultat */
5: **fin pour**
6: **fin pour**

4.8.2 Encapsulation d'un prototype existant

Cette section répond à la question suivante : que faut-il ajouter ou modifier dans un prototype existant pour permettre sa coopération avec d'autres prototypes ? Le but est d'obtenir un prototype «commandable» depuis un autre exécutable. Par commandable, nous entendons que le prototype fournisse un point d'accès afin d'agir sur lui. Pour cela le prototype est encapsulé dans un objet CORBA.

Rechercher les caractéristiques du prototype

Le prescripteur qui connaît le prototype (éventuellement avec l'aide de l'administrateur) remplit le formulaire prototype en langage naturel.

Rendre scriptable, scrutable, traçable et interface-exportable

Une fois le formulaire rempli, les points d'accès aux différentes données et au différents traitement sont identifiés. Si le prototype est scriptable, scrutable et interface exportable, les informations précédentes sont suffisantes et la vitrine pourra être définie.

Si, au contraire, le prototype n'a pas ces bonnes propriétés, nous proposons de l'encapsuler dans un objet, en ajoutant autant que faire se peut les propriétés souhaitées. Ces ajouts dépendent de deux paramètres.

Le premier paramètre est la granularité des traitements et des données accessibles. Ça dépend de la manière dont le prototype est programmé. Si la granularité est fine, le source que nous écrivons permet d'ajouter les méthodes pour accéder aux traitements et données. Ensuite nous publions ces méthodes dans la vitrine et nous ajoutons les liens nécessaires pour atteindre les traitements et les données du prototype.
Si la granularité est grande, nous ne pouvons pas faire grand chose. Par exemple, nous lançons le prototype en entier et attendons son résultat. Dans ce cas, la seule chose que nous publions dans la vitrine est la méthode launch() pour lancer le prototype.

Le second paramètre est l'existence d'observables générés par le prototype.
Si des observables existent, nous ajoutons dans notre surcouche, les méthodes permettant de les ajouter à la vitrine.
S'il n'existent pas, nous devons les construire en utilisant une des solutions décrites dans la section 3.4.4.

Bilan

En résumé, pour ajouter un prototype à l'atelier, il faut :

1. rendre le prototype aussi scriptable, scrutable, traçable et interface-exportable que possible ;
2. remplir le formulaire «prototype» ;
3. rédiger la vitrine du composant ;
4. générer les fichiers de la correspondance pour CORBA ;
5. assurer les liens entre les fichiers pour CORBA et le prototype ;
6. compiler les fichiers pour CORBA ;
7. installer le serveur du prototype ;
8. faire les liens dans EduMed avec le client du prototype.

Exemple

La figure 4.14 illustre l'architecture de l'atelier dans le cas de l'utilisation de *TALC* et *Mentoniezh*. Au moment de cette expérimentation, *TALC* et *Mentoniezh* étaient deux logiciels qu'on devait lancer en un seul bloc. Par conséquent, leur vitrine ne contient que la méthode launch() et les énoncés des exercices (`HDL_statement` et `CDL_statement`). De plus, la coopération de ces deux logiciels nécessite une traduction de l'énoncé au format approprié. L'annexe C détaille le raisonnement qui conduit à définir le macro-texte adéquat. Il est appelé `HDL-CDL_macro-text` et est ajouté à la vitrine de *Mentoniezh*. Sur cette figure apparaît aussi le gestionnaire d'activités et les constituants d'EduMed. Ces constituants sont l'ORB et les composants de services (gestionnaire de formats et gestionnaire d'interfaces graphiques) avec leur vitrine.

4.9 Mise en œuvre : la base de données d'indexation des prototypes

Pour intégrer et utiliser un prototype au sein de l'atelier, nous avons besoin de caractériser les prototypes. Nous créons pour cela un index. Cet index est implanté grâce à l'objet `Index`.

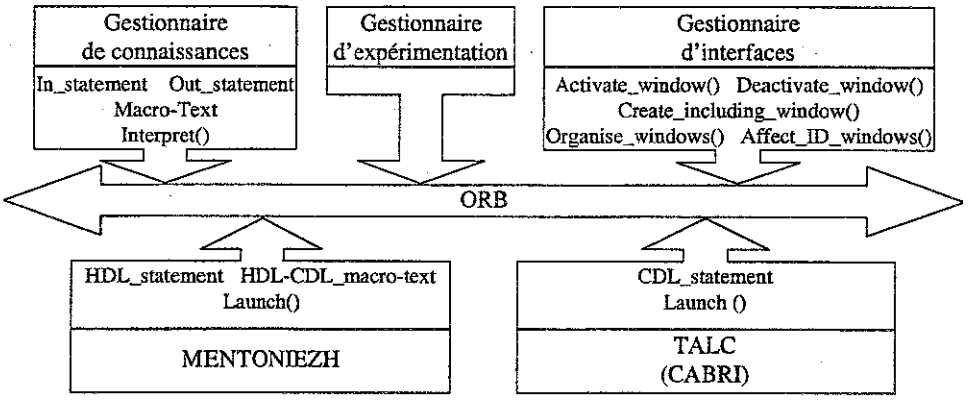


Figure 4.14 – Instanciation de l'atelier pour *TALC* et *Mentoniezh*

Chaque prototype est caractérisé (en particulier) par ses fonctionnalités. Chaque fonctionnalité peut être implantée dans différents prototypes. Nous gérons par conséquent dans cet index les objets `Prototype` et `Fonctionnalité`. Un objet `Fonctionnalité` contient principalement la description d'une fonctionnalité.

C'est le gestionnaire d'activités qui permet de gérer cet index via les menus `Fonctionnalité` et `Prototype`. Il gère une base d'information, implantée sous forme de liste chaînée d'objets, pour les fonctionnalités et une autre pour les prototypes pour un domaine donné. Le nombre d'objets manipulés dans chaque liste étant faible (au plus une vingtaine), l'utilisation d'une structure de données plus complexe (et efficace) n'était pas indispensable.

4.9.1 Menu Fonctionnalité

Le menu `Fonctionnalité` permet d'entrer des fonctionnalités pour un domaine donné. Nous avons établi une liste de fonctionnalités pour notre domaine d'application (la géométrie) d'après les fonctionnalités repérées dans les prototypes existants (chapitre 2). Nous définissons cette liste par défaut dans l'atelier. Le gestionnaire d'activité permet de :

- charger une liste de fonctionnalités en mémoire ;
- consulter la liste des fonctionnalités en mémoire ;
- modifier la liste des fonctionnalités en mémoire ;
- créer une liste des fonctionnalités en mémoire ;
- enregistrer la liste des fonctionnalités dans un fichier physique.

Ces éléments constituent les items du menu `Fonctionnalités`. Ils permettent la création et la modification de la liste de fonctionnalités (*cf.* figure 4.15) en fonction des besoins et de l'évolution des travaux dans le domaine.

4.9.2 Menu Prototype

La figure 4.16 présente un extrait de la fiche de description du prototype *CHyPre*. Le prototype présenté sur la figure 4.16 (*CHyPre*) possède quelques fonctionnalités parmi celles de la figure 4.15. Cependant le prototype n'ayant pas été conçu pour être utilisé dans un contexte de coopération, toutes les fonctionnalités qu'il implante ne sont accessibles que lorsque le prototype a été lancé. Il n'est donc pas possible de lancer l'une d'elles seule et directement. Nous disons, dans ce cas, que chaque fonctionnalité est «interne» au prototype. Chacune est accessible par

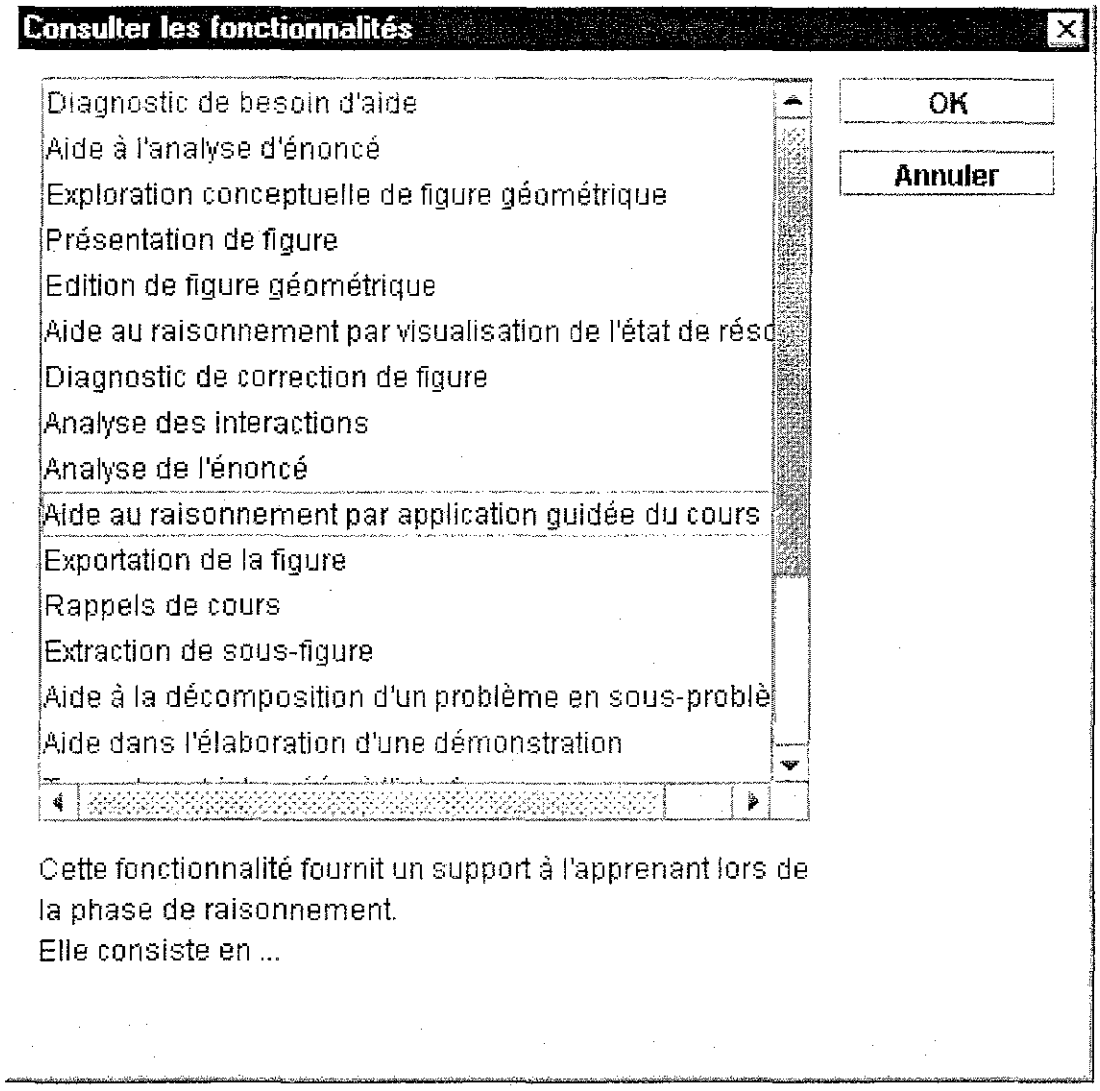


Figure 4.15 – Fenêtre de consultation des fonctionnalités

des menus. Il n'est pas non plus possible de choisir quelles fonctionnalités doivent être activées/désactivées.
La figure 4.17 illustre le choix des fonctionnalités pour un prototype.

Grâce à l'indexation présentée dans cette section, le gestionnaire d'activités connaît maintenant les prototypes disponibles et leurs fonctionnalités. Il peut alors les utiliser dans l'élaboration et la réalisation d'un scénario.

4.10 Mise en œuvre : choisir les données à sauvegarder

Le prescripteur, lors de la définition d'un scénario d'activité, doit définir quelles sont les données à mémoriser. C'est le gestionnaire d'activités qui fournit cette fonctionnalité via l'objet `gesData`. Cet objet récupère sur les vitrines de tous les composants présents, la liste de toutes données publiées par les composants. Ces données sont des informations scrutables, des résultats et des observables. À partir de ces données il fabrique un formulaire qu'il présente au prescripteur.

- **Nom :** *CHyPre*
- **Liste des fonctionnalités implantées :**
  1. Aide au raisonnement par visualisation de l'état de résolution
  2. Extraction de sous figure
  3. Trace des objets créés
  4. Analyse de l'énoncé
  5. Exploration visuelle de figure géométrique
- **Accès à chacune des fonctionnalités :**

Fonctionnalité n°	Type de points d'accès	Point d'accès
1	Interne	Menus
2	Interne	Menus
3	Interne	Menus
4	Interne	Menus
5	Interne	Menus

- **Fonctionnalités activables-désactivable :**

Fonctionnalité n°	Activable-Désactivable	Commande d'activation	Commande de désactivation
1	Non	Néant	Néant
2	Non	Néant	Néant
3	Non	Néant	Néant
4	Non	Néant	Néant
5	Non	Néant	Néant

- ...

Figure 4.16 – Extrait de la fiche de description du prototype *CHyPre*

Ce dernier choisit alors les informations à prendre en compte.
Lors de la réalisation du scénario, cet objet capte les données demandées et fabrique des flux de données. Ces derniers peuvent alors être dirigés soit vers un prototype particulier, soit vers une sortie standard, soit vers un fichier.

Avec cette section, nous avons terminé d'expliquer l'implantation des propositions et la mise en œuvre de l'atelier. La section suivant fait le point sur l'état d'avancement du projet.

4.11 Bilan

Nous avons implanté l'atelier à partir des classes principales présentées dans la figure 4.18. Ces classes sont écrites avec le *JDK*1.2. Cette figure décrit les classes que nous avons implantées, celles que nous avons adaptées et celles que nous avons spécifiées. En particulier, nous avons téléchargé JacORB [JacORB http], puis installé et configuré ce dernier pour un fonctionnement dans le cadre d'EduMed. Par ailleurs, nous avons spécifié le gestionnaire d'interfaces graphiques sans l'implanter totalement. En effet, nous avons expliqué dans la section 4.7 que le développement de ce composant est très lourd. Pour pouvoir tester l'atelier, nous avons implanté des solutions *ad*

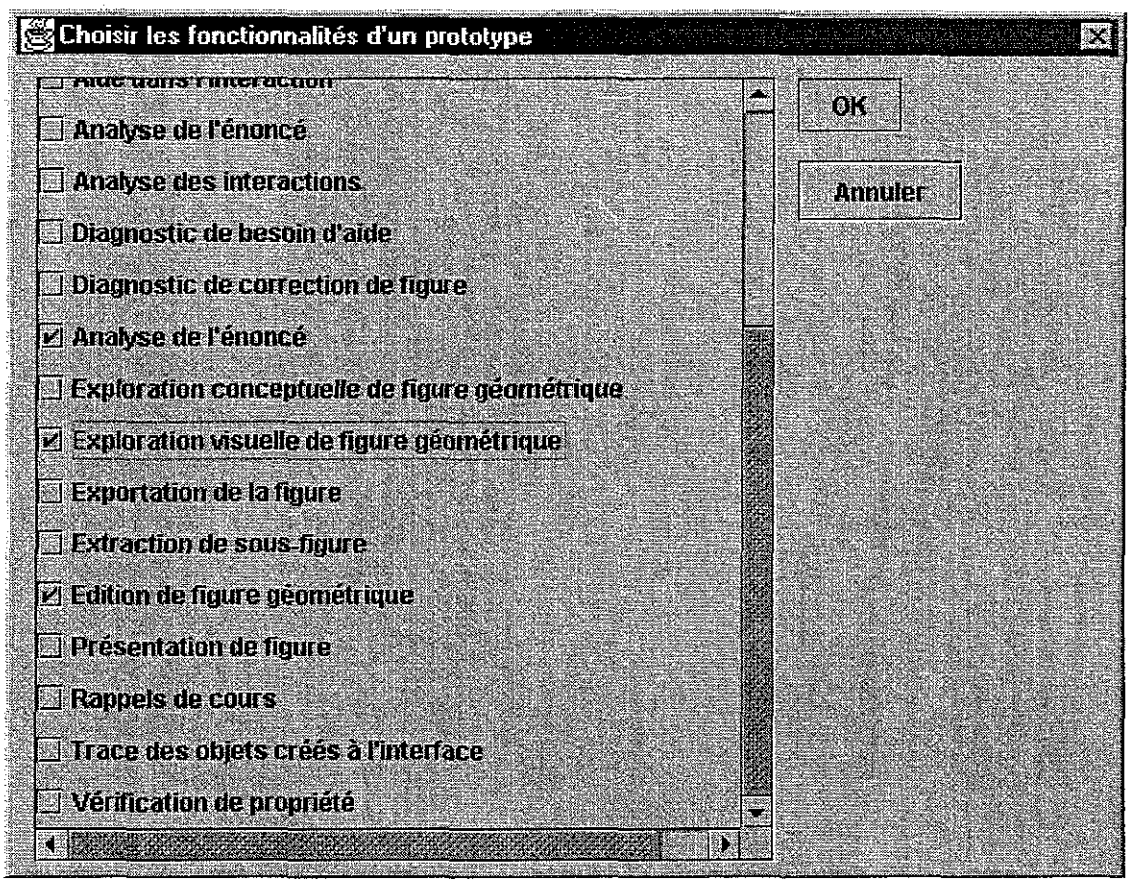


Figure 4.17 – Fenêtre de choix des fonctionnalités d'un prototype

solutions *ad hoc* remplissant la spécification de ce gestionnaire d'interfaces graphiques. Elles ont été testées sur PC (sous windows 95 et sous DOS) et sur station unix (sous SunOS[60]).
L'atelier n'a pas été testé récemment sur Macintosh. Cependant, la première version de l'atelier avait été implantée sur Macintosh et testée avec les prototypes *TALC* et *Mentoniezh*. Or la machine sur laquelle tout avait été configuré à été changée deux fois. La configuration nécessitait en particulier de télécharger et d'installer de nombreux logiciels et mises à jour du systèmes. La seconde fois, nous avons récupéré un Macintosh vierge, sur lequel il n'y avait ni *TALC*, ni *Mentoniezh*, ni *CABRI-Géomètre*. Dans ces conditions, nous avons décidé d'abandonner l'expérimentation sur Macintosh.

L'encapsulation des prototypes à été testée pour *CHyPre* et *Mentoniezh* sur PC et sur station Unix.
Le test de l'atelier nécessitait aussi l'installation du logiciel *VNC* : un serveur *VNC* sur PC, un client *VNC* sur Macintosh.

Nous terminons maintenant ce mémoire en rappelant dans le chapitre suivant, le contexte, les apports principaux et les perspectives de notre travail.

[60]Release 5.6 Version $Generic_1 05181 - 16[UNIX(R)SystemV Release 4.0]$

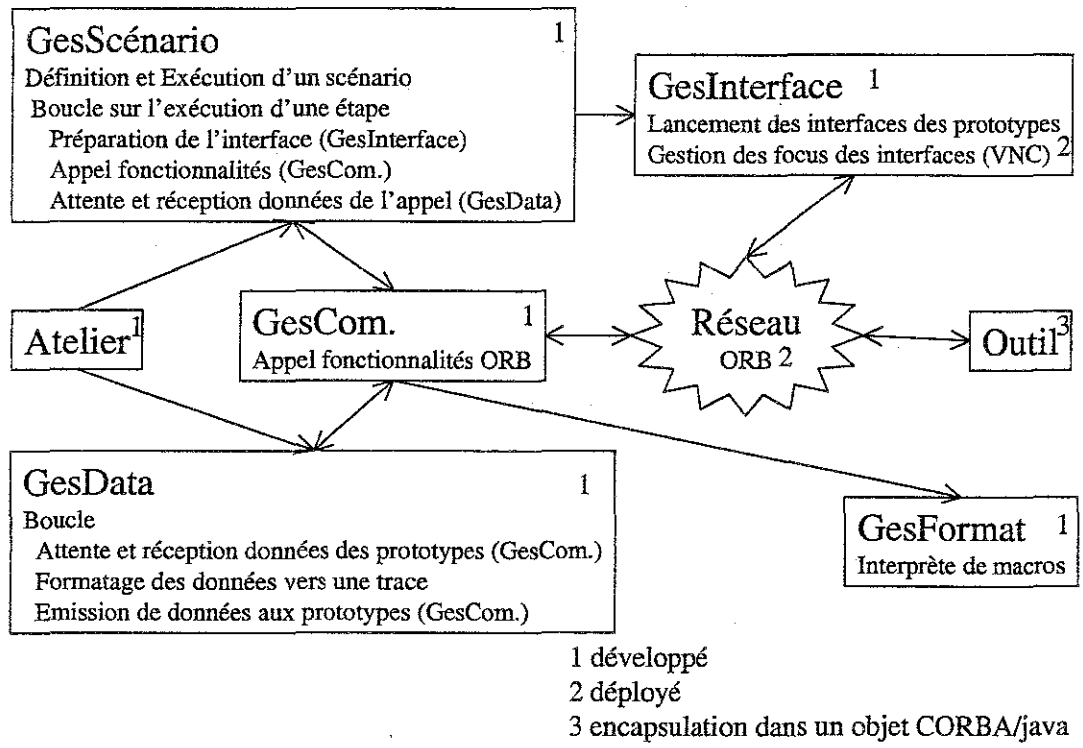


Figure 4.18 – État d'avancement de l'implantation (sur une description fonctionnelle de l'atelier)

Conclusion

Le contexte

L'objectif de ce travail est de concevoir une plate-forme permettant d'expérimenter des logiciels complémentaires issus de la recherche en EIAO. Cette plate-forme est nécessaire pour trois raisons :

- mettre au point les produits en adéquation avec les besoins et les nouvelles technologies ;
- favoriser une utilisation effective des produits de la recherche en EIAO dans les classes ou sur Internet ;
- offrir un environnement plus riche aux apprenants et aux enseignants.

Actuellement, de nombreux prototypes ou logiciels ont étés développés pour implanter des fonctionnalités visant une amélioration de l'apprentissage et de l'enseignement dans un domaine. Cependant, chaque prototype n'offre qu'une partie des fonctionnalités souhaitables (simulation, résolution de problème, diagnostic ou explication). Par conséquent, les utilisateurs rencontrent de nombreuses difficultés pour utiliser conjointement, à la main, des fonctionnalités complémentaires. En effet, il n'est aisé

- ni pour l'enseignant de proposer une activité pédagogique impliquant des fonctionnalités complémentaires ;
- ni pour l'apprenant de repérer la fonctionnalité et le prototype à utiliser dans les tâches à accomplir ;
- ni pour les chercheurs d'évaluer les prototypes ou d'expérimenter une fonctionnalité en utilisant l'existant.

La plate-forme que nous proposons, pour remédier à ces difficultés, facilite l'utilisation conjointe de prototypes. Elle permet la coopération des fonctionnalités complémentaires déjà implantées. Ceci se met en œuvre dans un environnement adapté et avant tout nouveau développement.
La coopération entre composants logiciels existants et complémentaires est une nécessité pour «passer à la vitesse supérieure» (ne plus tout refaire) dans beaucoup de domaines d'application de l'informatique. Ce travail a mis en évidence les difficultés, les informations à échanger, les services nécessaires, dans le cadre des environnements d'apprentissage; il propose des solutions utilisant les technologies objets normalisées. En ce sens il est une contribution intéressante à la vaste question de la coopération.
Notre proposition s'appuie sur l'analogie entre la conception d'une expérimentation de prototypes et la conception d'une expérimentation d'objets comme des micro-fusées. La première se déroule dans un espace appelé «atelier d'expérimentation». Nous proposons de garder le même terme pour la seconde et d'appeler «atelier» la plate-forme informatique qui permet l'expérimentation de prototypes.
Au cours de cette recherche, nous avons mis l'accent sur les quatre axes de travail suivants :

- la conception de l'atelier ;
- la réalisation d'une maquette ;
- la définition des conditions de l'utilisation conjointe de plusieurs prototypes ;
- la formulation de recommandations pour favoriser la communication et la coopération inter-logiciels, et cela dès la conception des produits.

Définir et développer un tel atelier est un projet ambitieux qui nécessite de prendre en compte à la fois plusieurs types d'utilisateurs (enseignant, apprenant et chercheur) et de nombreux aspects techniques.
En ce qui concerne les utilisateurs, l'atelier doit permettre à l'enseignant de choisir les fonctionnalités dont il a besoin en fonction d'une activité pédagogique. Il doit diminuer la charge cognitive de l'apprenant, en lui présentant uniquement les informations et outils nécessaires à un moment donné. Il doit permettre au chercheur de monter une expérimentation pour recueillir des données à exploiter. Mener ces trois projets de front est ambitieux. Par conséquent, nous développons prioritairement les fonctionnalités dédiées aux chercheurs, sans finaliser une interface ergonomique destinée à l'enseignant ou à l'apprenant.
En ce qui concerne les aspects techniques, l'atelier d'expérimentation de logiciels éducatifs est un environnement construit autour de trois types de composants :

- les prototypes : ce sont les composants à partir desquels l'activité est menée. Ils nécessitent une adaptation et une encapsulation dans un composant CORBA afin de pouvoir interagir entre eux et avec les deux autres types de composants ;
- le gestionnaire d'activités (ou GesAct) : c'est le composant qui permet de définir et de gérer l'activité proprement dite en faisant intervenir les prototypes ;
- le médiateur éducatif (ou EduMed) : c'est un ensemble de composants qui prend en charge tous les aspects techniques nécessaires pour que le gestionnaire d'activités exploite les prototypes.

Il est implanté en JAVA. Nous reprenons chacun des aspects techniques un par un dans la section suivante.

Principaux apports

À propos des prototypes

Typologie des fonctionnalités en géométrie

Cet aspect concerne la typologie des fonctionnalités disponibles pour un domaine. Il n'y a pas de typologie avec laquelle une communauté de chercheurs et d'utilisateurs soit en accord. Par conséquent, nous avons élaboré une liste de fonctionnalités concernant notre domaine d'application (la géométrie) à partir de l'étude des logiciels existant. Cette liste, composée de vingt et un éléments, est rappelée en annexe A.
Nous avons ensuite réparti ces fonctionnalités suivant cinq groupes :

(a) domaine d'apprentissage ;
(b) pédagogie et aide ;
(c) outils et micromondes ;
(d) dynamique de l'activité ;
(e) interface de l'utilisateur.

Ce regroupement nous permet de distinguer

- les fonctionnalités qui relèvent de la pédagogie de celles qui sont des aides techniques ;
- les fonctionnalités qui relèvent de la gestion de la dynamique de l'activité de celles qui constituent une présentation à l'interface.

Ces distinctions ne sont pas possibles dans la présentation classique des EIAO en quatre modules («domaine», «interface», «modèle de l'apprenant» et «guidage pédagogique» [Quéré 91]).
Le regroupement des fonctionnalités est présenté sur la figure 2.11 (page 59). Il met en perspective le contraste entre d'une part, le grand nombre des fonctionnalités implantées pour les groupes (b) et (c), et d'autre part, le peu de fonctionnalités relatives aux groupes (a) et (d). Ce contraste est révélateur du petit nombre de recherches portant sur les fonctionnalités liées à la dynamique de l'activité et au domaine d'apprentissage.
Nous reprenons la présentation des EIAO en cinq groupes pour l'indexation des prototypes.

Indexation des prototypes et des fonctionnalités

Cet aspect concerne l'indexation des prototypes. L'atelier dispose d'une base d'informations pour indexer les prototypes disponibles

- pour identifier les prototypes disponibles ;
- pour permettre à l'utilisateur de choisir les prototypes qu'il souhaite utiliser ;
- pour permettre un accès aisé à ceux-ci ;
- pour faciliter la sélection et l'enchaînement de ceux-ci.

Bien que des travaux sur l'indexation des «objets pédagogiques» progressent, il n'y a pas de standard qui permette de décrire les prototypes issus de la recherche en EIAO de manière assez fine et propre au domaine enseigné. Nous avons présenté les tentatives de normalisation à la section 2.4. De celles-ci, nous avons extrait l'architecture LTSA (Learning Technology Systems Architecture) et les métadonnées concernant les objets pédagogiques (LOM, Learning Objects Metadata). Cependant, le niveau de description atteint par l'architecture ne permet pas de différencier les prototypes selon les fonctionnalités qu'ils implantent. De même, les LOM ne permettent pas non plus d'atteindre le niveau de description permettant de décrire les fonctionnalités des prototypes. Par conséquent, nous utilisons la liste de fonctionnalités pour le domaine de la géométrie décrite précédemment. Nous utilisons cette fiche pour indexer chaque prototype et pour créer les accès via l'atelier. Cette indexation introduit un niveau de granularité supplémentaire à celle proposée dans l'architecture LTSA.

Propriété des prototypes

Cet aspect concerne la gestion de l'interopération entre les prototypes grâce à la définition d'un *langage de commande* pour l'atelier et de *propriétés souhaitables* pour les prototypes. Ces propriétés sont la scriptabilité, la scrutabilité, la traçabilité ainsi que l'exportabilité de l'interface. La scriptabilité permet une certaine forme de prise de contrôle sur un prototype grâce à un script. La scrutabilité permet l'observation d'informations concernant le prototype (états ou variables principalement) par scrutation. La traçabilité permet la mémorisation des actions de l'utilisateur sur le prototype (pour pouvoir rejouer la séquence par exemple). Enfin l'exportabilité de l'interface permet à l'atelier d'agir sur l'interface graphique du prototype.

À propos des données

Communication des données

Au sein de l'atelier et en particulier entre les prototypes, nous avons besoin d'un média de communication pour

- éviter à l'utilisateur de ressaisir certaines données de multiples fois ;
- acheminer les données vers un prototype.

Ces données à communiquer sont de différents types :

- les connaissances du domaine ;
- les connaissances d'interaction ;
- les connaissances pédagogiques ;
- les connaissances sur l'apprenant ;
- les résultats produits ;
- les variables scrutables.

Pour assurer la communication entre les prototypes et avec l'atelier, nous utilisons un médiateur (*middleware*). Ce médiateur est un bus objet (ORB) suivant la technologie CORBA. Le médiateur que nous avons déployé est JacORB [JacORB http] .

Gestion des formats de connaissances du domaine

L'atelier permet la communication de données. Cependant, il ne suffit pas d'acheminer les données. Il faut, de plus, qu'elles soient exploitables par le prototype qui les reçoit. Puisqu'il n'existe pas de norme, il faut utiliser le format adéquat. Nous nous sommes focalisé, dans un premier temps, sur les connaissances du domaine.
L'échange de connaissances du domaine est assuré par un interprète de macro-définitions. L'interprète est un logiciel indépendant. Il est encapsulé dans un composant CORBA afin de constituer un composant de service pour EduMed. Les macro-définitions constituent un moyen de réaliser un traducteur automatique d'un langage à un autre, si le premier est traduisible vers le second. Nous avons donné une formalisation des concepts de traductibilité et de macro-définition. La traductibilité d'un langage vers un autre se décompose en deux contraintes successives (couverture du domaine et granularité plus fine). Le concept de macro-définition, similaire à celui utilisé dans le préprocesseur du langage C, permet la traduction atome par atome d'un langage dans un autre. Une macro-définition permet d'associer à tout énoncé d'un premier langage un énoncé d'un second langage, par le principe des substitutions successives.
Nous avons ensuite montré la faisabilité de notre approche en l'implantant en géométrie entre les langages HDL et CDL. Une implantation des macro-définitions peut être facilement et rapidement effectuée pour d'autres domaines où les conditions sont assurées (nous pensons en particuliers aux domaines déjà très formalisés comme en sciences physiques : électricité, optique, mécanique, *etc.*).

À propos de l'activité

Gestion de l'interface graphique

Pendant l'activité, tous les menus et toutes les fenêtres de tous les prototypes mis en œuvre ne sont pas pertinents à chaque instant. De plus, dans le cas d'un affichage exhaustif de toutes les

fenêtres et tous les menus, la charge cognitive ne serait pas supportable par l'utilisateur. Ainsi, pour diminuer la charge cognitive de l'utilisateur, nous proposons de sélectionner et de faire cohabiter diverses fenêtres de présentation. Le gestionnaire d'interfaces graphiques a pour but la sélection et l'organisation de ce qui doit être affiché à l'écran au fur et à mesure de l'activité. Nous avons spécifié un composant de service. Cependant, dans l'implantation actuelle, nous avons géré l'interface graphique en utilisant des solutions logicielles existantes : solution *ad hoc* dans un cas, logiciel de prise de contrôle (*VNC*) dans l'autre cas.

Préparation de l'exploitation de l'activité

Cet aspect concerne la préparation de l'exploitation de l'activité menée avec l'atelier. Cette exploitation au cours de l'activité (ou de manière différée) nécessite de sauvegarder des données. Ces données sont de trois types :

- les résultats produits par le prototype ;
- les informations scrutables du prototype ;
- les traces de l'interaction de l'utilisateur avec le prototype.

Nous proposons pour ce faire d'enregistrer ces données. Cependant, toutes ces données ne sont pas pertinentes pour le prescripteur. C'est pourquoi, nous proposons de sélectionner celles qui sont pertinentes. En revanche, les traces de l'interaction de l'utilisateur avec le prototype sont très nombreuses et de bas-niveau. Nous proposons alors de les agréger afin de construire des observables c'est-à-dire des traces d'interaction pertinentes pour le prescripteur. Ils permettent de se placer à un niveau communication des connaissances plus abstraites (moins proches des événements traités par le système d'exploitation). Nous avons proposé d'une part un mécanisme pour cette composition et d'autre part un format de représentation de ces observables. Le mécanisme d'agrégation des traces d'interaction en observable repose sur le même principe que les macro-définitions. Le test de l'utilisation de l'interprète de macro-définitions pour construire des observables fait partie des perspectives.

L'atelier que nous avons ainsi spécifié constitue une plate-forme généraliste, la plus portable possible. Il constitue une base pour le développement de composants à reprendre par d'autres chercheurs ou développeurs.

Perspectives

Le domaine de la coopération des prototypes de recherche en EIAO n'avait pas été étudié de façon systématique. Le présent mémoire a permis de cerner les besoins et d'avancer quelques propositions. Beaucoup de questions n'ont pas été approfondies. Nous proposons ci-après quelques perspectives.

Perspectives Techniques

Faciliter la mise en œuvre

Une étape suivante dans le développement de l'atelier consiste à fabriquer des outils pour faciliter le travail de l'administrateur dans la préparation de l'atelier et des activités. Cela consiste à proposer une automatisation du passage de la fiche de description d'un prototype à la vitrine (exprimée en IDL) et au composant CORBA, par exemple.

Améliorer l'interface

Ensuite, il faudrait développer des interfaces aussi conviviales et ergonomiques que possible. Après l'activité de cet atelier amélioré sur la forme, un travail est nécessaire pour adapter ces interfaces à un contexte éducatif. La transition du contexte de recherche au contexte éducation devrait se faire assez simplement. L'utilisation conjointe de fonctionnalités complémentaires permettrait ainsi d'offrir un environnement plus riche aux apprenants et aux enseignants.

Étendre la notion de scénario

Définir le déroulement de l'activité consiste à créer un scénario. Nous avons défini un scénario d'activité comme un ensemble structuré d'étapes via une suite d'étapes. C'est le gestionnaire de scénario à l'intérieur du gestionnaire d'activités qui permet de définir et d'exécuter un scénario. Cette première implantation nous permet d'envisager ensuite une structuration plus complexe, gouvernée par un automate.

Perspectives d'évaluation

Expérimenter l'atelier via internet

L'atelier a été expérimenté dans notre laboratoire entre des ordinateurs reliés par un réseau Ethernet. Par activité, nous entendons ici, le lancement de l'atelier et la manipulation des prototypes via l'atelier.
Toutefois la conception de l'atelier permettrait le test de l'atelier via l'internet. En effet, grâce à la technologie CORBA, il n'y a théoriquement pas de différence entre deux ordinateurs distants sur un réseau local ou sur internet. Le test devrait cependant prendre en compte la distance et le débit des liaisons.

Évaluer avec les utilisateurs finaux

L'atelier n'a pas été expérimenté avec des enseignants et des apprenants. Un telle activité est un travail intéressant à effectuer dans la suite de notre travail. Nous projetons de mettre nos sources et notre documentation en ligne afin de permettre à d'autres chercheurs de s'approprier l'atelier pour mener des activités et si possible pour nous envoyer leurs commentaires, appréciations et résultats dans le but de faire évoluer l'atelier.

Appliquer l'atelier à un autre domaine

L'atelier tel qu'il est défini ici n'est pas spécifique d'un domaine d'enseignement et d'apprentissage : il est généralisable. Nous l'avons mis en œuvre pour la géométrie. L'atelier permet de créer une nouvelle liste de fonctionnalités pour un autre domaine. Cette liste peut alors être utilisée pour indexer des prototypes. Ces prototypes peuvent être utilisés pour construire d'autres scénarios d'activité. La richesse des logiciels dans le domaine permet de déterminer des fonctionnalités diverses. C'est, par exemple, le cas dans les domaines suivants : comptabilité/gestion, physique (dont électronique), langues vivantes, géographie, chimie, *etc.*. Notez cependant que la conduite de certains exercices recouvre un certain nombre d'étapes qui sont indépendantes du domaine d'application choisi. Par exemple, un exercice comprend les étapes suivantes : analyse de l'énoncé, résolution et rédaction de la réponse. Et pendant cet exercice dans le cadre de l'atelier, divers Logiciels Éducatifs proposent des formes d'aide, d'explication ou de rappels de cours. Ces fonctionnalités «génériques» sont instanciées par le domaine en question. En tenant compte

de cela, il serait possible de préparer des scénarios types servant de base aux chercheurs pour monter une activité sans tout redéfinir.

Perspectives de recherche

Spécifier des composants logiciels à vocation pédagogique

Par ailleurs, la spécification de composants logiciels à vocation pédagogique est également nécessaire pour leur mise en œuvre au sein des portails et plate-formes de formation qui sont en développement sur le Web. Les études sont encore balbutiantes, bien que des commissions de normalisation se mettent en place au niveau international à cause des enjeux commerciaux et de la forte demande. Le présent travail est une contribution aux nécessaires études sur la spécification des logiciels éducatifs en vue de leur utilisation dans des parcours de formation sur le Web. En particulier, la typologie des fonctionnalités des logiciels pour la géométrie élaborée à partir de l'état de l'art des logiciels existants pourrait servir de base de discussion pour des groupes pluridisciplinaires afin de normaliser les termes de cette typologie.

Étendre l'échange des connaissances

L'atelier dans l'état actuel permet l'échange de connaissances du domaine en évitant ainsi des saisies multiples. La solution que nous proposons (l'interprète de macro-définitions) pourrait s'avérer intéressante pour échanger d'autres types de connaissances, par exemple :

- les connaissances d'interactions ;
- les connaissances pédagogiques ;
- les connaissances sur l'apprenant.

Cependant, pour affirmer cela il faudrait faire une étude des modes de représentation des ces divers types de connaissances et évaluer la faisabilité d'un tel échange.

Proposer un standard pour la représentation des connaissances en géométrie

L'exemple de la coopération de *TALC* et *Mentoniezh* nous a amené à utiliser le langage CDL comme inter-langage pour exprimer les connaissances en géométrie. Ce langage pourrait servir de base pour définir une ontologie des connaissances géométriques dans un cadre pédagogique. Ce langage, pivot de la traduction, changerait à chaque fois que cela serait nécessaire (en utilisant le langage recouvrant le plus grand domaine et de granularité la plus fine). Cet inter-langage sera raffiné au fur et à mesure que des Logiciels Éducatifs aux langages plus «recouvrants» et de granularité plus fines inter-opéreront. Nous obtiendrions ainsi un inter-langage défini *a posteriori* de plus en plus général.

Nous espérons que cet atelier sera une aide pour mieux évaluer et pérenniser les réalisations issues de la recherche en EIAO. Il devrait contribuer à rendre les logiciels éducatifs plus conviviaux à la fois pour les chercheurs (pour tester et valider des idées) et pour les utilisateur finaux (c'est-à-dire les enseignants et les apprenants).

Bibliographie

[AFNOR http] AFNOR. Association Française de NORmalisation. http ://www.afnor.fr/

[Ag-Almouloud 92] S. Ag-Almouloud. *L'ordinateur, outil d'aide à l'apprentissage de la démonstration et de traitement de données didactiques.* Thèse de Doctorat, Université de Rennes, 1992.

[Aho 72] A. V. Aho et J. D. Ullman. *The Theory of Parsing, Translation and compiling. Volume 1 Parsing.* Automatic Computation. Prentice-Hall, Englewood Cliffs (N. J.), 1972.

[Allard 86] J.C. Allard et C. Pascal. *Euclide, un langage pour la géométrie plane, logiciel et manuel.* Cédic-Nathan, 1986.

[Allen 90] R. Allen, P. Nicolas et L. Trilling. Figure correctness in an expert system for teaching geometry. *8th biannual conference of the Canadian society for computational studies of intelligence*, pages 154–160, Ottawa, 1990.

[Anderson 85] J.R. Anderson, C.F. Boyle et G. Yost. The geometry tutor. *9th International Joint Conference on Artifical Intelligence*, Los Altos, 1985. Morgan Kaufmann Publishers.

[Apple Computer 93] Inc. Apple Computer. Chapter 3 - Introduction to Apple Events. *Inside Machintosh : Interapplication Communication*, Apple Technical Library. Addison-Wesley, New York, 1993.

[ARIADNE http] ARIADNE. Alliance of Remote Instructional Authoring and Distribution Networks for Europe. http ://ariadne.unil.ch/.

[Authier 98] A. Authier, C. Grolleau, M.L. Hocquenghem, S. Hocquenghem, F. Monnet, Y. Paquelier, P. Sérès, A.M. Serfati et A. Varoquaux. Géospace : logiciel de construction mathématique dans l'espace. CREEM (CNAM) - CRDP de Champagne-Ardenne, 1998.

[Balacheff 94a] N. Balacheff. *Learning through Computers : Mathematics and Educational*

Technology, chapitre Artificial Intelligence and Real Teaching. Springer Verlag, Berlin, 1994.

[Balacheff 94b] N. Balacheff et M. Vivet. *Didactique et intelligence artificielle*, chapitre Didactique et Intelligence Artificielle, pages 9–42. La Pensée Sauvage, Grenoble, 1994.

[Balacheff 97] N. Balacheff, M. Baron, C. Desmoulins, M. Grandbastien et M. Vivet. Conception d'environnements interactifs d'apprentissage avec ordinateurs. tendances et perspectives. Sylvie Pesty et Pierre Siegel, éditeurs, *6e journées du PRC-GDR Intelligence Artificielle, Grenoble*, pages 315–338. Hermès, avril 1997.

[Balbiani 94] P. Balbiani, V. Dugat, L. Fariñas del Cerro et A. Lopez. *Éléments de géométrie mécanique.* Langue - Raisonnement - Calcul. Hermès, Paris, 1994.

[Baulac 90] Y. Baulac. *Un micromonde de géométrie dynamique, Cabri-géomètre.* Thèse de Doctorat, Université Joseph Fourier, Grenoble I, Grenoble, 1990.

[Baulac 91] Y. Baulac et I. Giorgiutti. Interaction micromonde/tuteur, le cas de cabri-géomètre et DéFI. *2es Journées Environnement Interactif d'Apprentissage avec Ordinateur, EIAO'91*, pages 11–18, Cachan, 1991. Les Éditions de l'École Normale Supérieure de Cachan.

[Baulac 92] Y. Baulac, F. Bellemain et J.-M. Laborde. Cabri : the Interactive Geometry Notebook, 1992.

[Bazin 93] J.-M. Bazin. Un modèle d'expert en résolution de problème de géométrie. *3es Journées Environnement Interactif d'Apprentissage avec Ordinateur, EIAO'93*, Cachan, 1993.

[Bellemain 92] F. Bellemain. *Conception, réalisation et expérimentation d'un logiciel d'aide à l'enseignement de la géométrie, Cabri-géomètre.* Thèse de Doctorat, Université Joseph Fourier, Grenoble I, Grenoble, 1992.

[Bernat 89] P. Bernat. Dessiner l'Espace. Topiques éditions, 1989.

[Bernat 91] P. Bernat. Pratiquer l'Espace. Topiques éditions, 1991.

[Bernat 94a] P. Bernat. Calques 2. Topiques éditions, 1994.

[Bernat 94b] P. Bernat. *Conception et réalisation d'un environnement interactif d'aide à la résolution de problèmes. CHYPRE : un exemple pour la démonstration*

en géométrie. Thèse de Doctorat, Université Henri Poincaré, Nancy I, 1994.

[Bernat 95] P. Bernat. Spécificités et modélisation de l'interaction dans un EIAO. *4es Journées Environnement Interactif d'Apprentissage avec Ordinateur, EIAO'95*, pages 208–220, Cachan, 1995. Hermès.

[Bernat 96a] P. Bernat. Approche didactique pour la modélisation informatique des connaissances et de l'interaction dans CHYPRE. Rapport interne, Centre de Recherche en Informatique de Nancy, Vandoeuvre-lès-Nancy, 1996.

[Bernat 96b] P. Bernat. Modélisation des connaissances et de l'interaction dans un logiciel de résolution de problèmes en géométrie : CHYPRE. *Sciences et Techniques éducatives*, 3(2), 1996.

[Bernat 97] P. Bernat. Représenter et manipuler des connaissances dans un environnement d'apprentissage de résolution de problèmes. *Revue d'intelligence artificielle*, 11(2) :213–238, 1997.

[Bolc 87] L. Bolc. *Natural Language Parsing System.* Symbolic computation. Springer, 1987.

[Botquelen 97] B. Botquelen. Les systèmes d'information client-serveur et leurs outils. une tendance. *Génie logiciel*, 44 :2–9, 1997.

[Bouhineau 97] D. Bouhineau. *Construction automatique de figures géométriques et programmation logique avec contraintes.* Thèse de Doctorat, Université Joseph Fourier, Grenoble I, juin 1997.

[Bourda 00] Y. Bourda et M. Hélier. Métadonnées et XML : applications aux «objets pédagogiques». *Technologie de l'Information et de la Communication dans les enseignement d'ingénieurs et dans l'industrie, TICE'2000*, pages 135–141, Troyes, october 2000.

[Bourguin 01] G. Bourguin et A. Derycke. Integrating the CSCL Activities into Virtual Campuses : Foundations of a new Infrastructure for Distributed Collective Activities. *1st European Conference on Computer-Supported Collaborative Learning, Euro-CSCL 2001*, Maastricht, mar 2001.

[Calmet 97] J. Calmet, P. Kullmann, S. Jekutsch et J. Schü. Un logiciel multi-agents basé sur le concept de médiateur. Joël Quinqueton, Marie-Claude Thomas et Brigitte Trousse, éditeurs, *5es journées francophones Intelligence Artificielle et Systèmes Multi-Agents, JFIADSMA'97,*, La Colle sur Loup, 1997. Hermès.

[Capponi 91] B. Capponi. Hypercarré : problème tutoriel en géométrie. *2es journées Environnements Interactifs d'Apprentissage avec Ordinateur, EIAO'91*, pages 11–18, Cachan, 1991. Les Éditions de l'École Normale Supérieure de Cachan.

[Carraux 99] E. Carraux. Support informatique pour l'analyse de l'interaction homme/machine. Rapport de DEA en sciences cognitives, Université Joseph Fourrier, Grenoble I, 199.

[CEN http] CEN. Commité Européen de Normalisation. http ://www.cenorm.be/

[Chamois http] Chamois. Logiciel shareware d'édition et de manipulation de figures géométriques. http ://www.multimania.com/bourit/

[Cheikes 98] B. A. Cheikes, M. Geier, R. Hyland, F. Linton, L. Rodi et H.-P. Schaefer. Embedded Training for Complex Information Systems. Henry M. Goettl, Barry P. anf Halff, Carol L. Redfield et Valerie J. Shute, éditeurs, *4th International Conference Intelligent Tutoring Systems, ITS'98, San Antonio, Texas, USA*, volume 1452, série *LNCS*, pages 36–45. Springer, august 1998.

[Crampes 99] M. Crampes, L. Bayard, A. Gelly et P. Uny. Spécification et proposition d'une DTD pour la qualification de matériaux pédagogiques adaptatifs. *STE*, 6(2) :343–374, 1999.

[Cuppens 90] R. Cuppens, éditeur. *Actes de l'université d'été "Informatique et enseignement de la géométrie*, Toulouse, 1990. IREM.

[Delevenay 59] E. Delevenay. *La machine à traduire*, volume 834, série *Que sais-je ?* P.U.F., Paris, 1re édition, 1959.

[Després 97] C. Després et P. Leroux. Raisonner sur la trace : analyse de sessions avec l'application roboteach. *5es Journées Environnement Interactif d'Apprentissage avec Ordinateur, EIAO'91*, pages 277–288, Cachan, 1997. Les Éditions de l'École Normale Supérieure de Cachan.

[Desmoulins 94] C. Desmoulins. *Étude et réalisation d'un système tuteur pour la construction de figures géométriques.* Thèse de Doctorat, Université Joseph Fourier, Grenoble, 1994.

[Desmoulins 98] C. Desmoulins et V. Liberatore. Une formation de techniciens en électronique utilisant Internet à domicile. *Conférence européenne sur les usages pédagogiques d'Internet et sur la construction de l'identité européenne, INTELE'98*, page 54, 1998.

[Dreyfus 81] H. L. Dreyfus. *From Micro-Worlds to Knowledge Representation : AI at Impasse.* Mind Design. MIT Press, Cambridge, Massachusset, 1981.

[Dubourg 95] X. Dubourg. *Modélisation de l'Interaction en EIAO, une Approche événementielle pour la Réalisation du Système REPÈRE.* Thèse de Doctorat, Université de Caen, 1995.

[Durand 97] S. Durand, F. Lesage et C. Moulin. Utilisation des systèmes multi-agents dans la modélisation des systèmes adaptatifs. *International Symposium of Economics and Informatics*, page 4, Bucarest, mai 1997.

[Educnet http] Educnet. Étude comparative technique et pédagogique des plates-formes pour la formation ouverte et à distance. http ://www.educnet.education.fr/superieur/plateforme.htm.

[Eife-l http] Eife-l. Europeen Institute for E-Learning http ://www.eife-l.org/Fr/

[Ferber 95] J. Ferber. *Les systèmes multi-agents. Vers une intelligence collective.* Informatique Intelligence Artificielle (IIA). InterEditions, 1995.

[Ferneda 92] E. Ferneda, M. Py, P. Reitz et J. Sallantin. L'agent rationel SAID : une application en géométrie. *1st European Colloquim on Cognitive Sciences*, pages 175–192, Orsay, 1992.

[Giorgiutti 91] I. Giorgiutti et Y. Baulac. Interaction micromonde/tuteur en géométrie, le cas de cabri-géomètre et de defi. Monique Baron, Régis Gras et Jean-François Nicaud, éditeurs, *2es journées Environnement Interactif d'Apprentissage avec Ordinateur, EIAO'91*, pages 11–18, Cachan, 1991. Les Editions de l'Ecole Normale Supérieure de Cachan.

[Grandbastien 96] M. Grandbastien. Introduction du numéro spécial consacré aux recherches sur les logiciels d'apprentissage de la géométrie et leur usage. *Sciences et Techniques Éducatives*, 3(2) :145–156, 1996.

[Grandbastien 98] M. Grandbastien. Developing Knowledge Systems for Training in the Workplace : a Challenge for the coming Years. *IT&KNOWS, IFIP World Computer Congress '98, Vienna, Austria.* Austrian Computer Society, août 1998.

[Gras 88] R. Gras. *Aide logicielle aux problèmes de démonstration géométriques dans l'enseignement secondaire*, volume 17. IREM, Grenoble, 1988.

[Gras 96] R. Gras et I. Giorgiutti. Computer Aided Proofs in School Geometry. Intelligent Learning Environments : the Case of Geometry. Jean-Marie

Laborde, éditeur, *NATO Advanced Research Workshop on Intelligent Learning Environnements : the Case of Geometry, Grenoble, 1989*, volume 117, série *Series F : Computer and Systems Sciences.*, pages 63–81. NATO ASI Series, 1996.

[Gruber 93] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Nicola Guarino et Roberto Poli, éditeurs, *Formal Ontology in Conceptual Analysis and Knowledge Representation.* Kluwer Academic Publisher, Dordrecht (The Netherland), 1993.

[Haumont 98] C. Haumont. Aide au formateur pour le suivi de travaux pratiques à distance. Rapport de DEA informatique, Université Henri Poincaré, Nancy I, 1998.

[Henninger 00] D. Henninger et Soudant S. Définition d'une interface pour un interprète de macro-définitions. Rapport de Maîtrise informatique - module d'initiation à la recherche, Université Henri Poincaré, Nancy I, 2000.

[IEEE http] IEEE. Institute of Electrical and Electronics Engineers. http ://www.ieee.org/

[ISO http] ISO. International Standards Organization. http ://www.iso.ch/

[Jackiw 95] N. Jackiw. User Handbook for the Geometer's Sketchpad. Visual Geometry Project, Key curriculum Press, 1995.

[JacORB http] JacORB. The free Java implementation of the OMG's CORBA standard. http ://jacorb.inf.fu-berlin.de/

[Kautz 87] H. A. Kautz. *A formal theory of plan recognition.* Thèse de Doctorat, University of Rochester, 1987.

[Kernighan 88] B. Kernighan et D. Ritchie. *The C Programming Language.* Prentice Hall, Upper Saddle River (NJ), 1988.

[Koedinger 98] K. R. Koedinger, D. D. Suthers et K. D. Forbus. Component-Based Construction of a Science Learning Space. Henry M. Goettl, Barry P. anf Halff, Carol L. Redfield et Valerie J. Shute, éditeurs, *4th International Conference Intelligent Tutoring Systems, ITS'98, San Antonio, Texas, USA*, volume 1452, série *LNCS*, pages 166–167. Springer, august 1998.

[Konstantas 93] D. Konstantas. Object oriented interoperability. Oscar M Nierstrasz, éditeur, *7th European Conference on Object-Oriented Programming, ECOOP'93*, Lecture Notes in Computer Science, pages 80–102, Kaiserslautern (Germany), 1993. Springer-Verlag.

[Laborde 86] J.-M. Laborde. Projet d'un système intelligent d'apprentissage de la géométrie. Présentation de projet, LSDD-IMAG, Grenoble, 1986.

[Laborde 89] J.-M. Laborde et L. Trilling. Conception et réalisation d'un système intelligent d'apprentissage de la géométrie. Présentation de projet, LSDD-IMAG, Grenoble, 1989.

[Le Huitouze 88] S. Le Huitouze. *Mise en oeuvre de PrologII/MALI.* Thèse de Doctorat, Université de Rennes I, 1988.

[Lepine 97] J. Lepine et S. Wallerand. Atelier de géométrie 3D. TLC-Edusoft, 1997.

[Lester 97] J. C. Lester, S. A. Converse, B. A. Stone, S. E. Kahler et S. Todd Barlow. Animated Pedagoggical Agents and Problem-Solving Effectiveness : A Large-Scale Empirical Evaluation. Ben Du Boulay et Riichiro Mizogushi, éditeurs, *International Conference on Artificial Intelligence in Education, AIED'97, Kobe, Japan*, pages 23–30. IOS Press, 1997.

[LTSC http] LTSC. Learning Technology Standards Committee http ://ltsc.ieee.org/

[Luengo 97a] V. Luengo. *CABRI-EUCLIDE : Un Micromonde de Preuve intégrant la Réfutation, Principes Didactiques et Informatiques, Réalisation.* Thèse de Doctorat, Université Joseph Fourier, Grenoble I, 1997.

[Luengo 97b] V. Luengo. Un micromonde de preuve intégrant la réfutation : Cabri-euclide. Jean-François Nicaud Monique Baron, Patrick Mendelsohn, éditeur, *5es Journées Environnement Interactif d'Apprentissage avec Ordinateur, EIAO'97*, pages 85–97, Cachan, 1997. Hermès.

[Mei Technology Corporation 97] Mei Technology Corporation. XAIDA. Mei Technology Corporation, 8930 FourWinds Drive, Suite 450, San Antonio, Texas 78239, 1997.

[Minsky 70] M. Minsky et S. Papert. Draft on a Proposal to ARPA : for Research on Artificial Intelligence. Rapport, MIT, Cambridge, Massachusset, 1970.

[Nanard 90] J. Nanard. *La manipulation directe en interface homme-machine.* Thèse de Doctorat, Université des sciences et techniques du Languedoc, Montpellier, 1990.

[OMG http] OMG. Object Management Group. http ://www.omg.org/

[Orbix http] Orbix. ORB CORBA de IONA©. http ://www.iona.com/products/orbhome.htm

[Orfali 97] R. Orfali, D. Harkey et J. Edwards. *Instant CORBA*. Wiley Computer Publishing, New York, 1997.

[Pachet 96] F. Pachet, P.-Y. Djamen, C. Frasson et M. Kaltrenbach. Un mécanisme de production de conseils exploitant les relations de composition et de précédence dans un arbre de tâche. *Sciences et Techniques Éducatives*, 7(3/4), 1996.

[Papert 80] S. Papert. *Jaillissement de l'esprit, ordinateurs et apprentissge*. Flammarion, Paris, 1980.

[Person 95] M. Person. Génération de diagnostic dans le tuteur de construction géométriques TALC. Rapport de DEA informatique, Université Henri Poincaré, Nancy I, 10 1995.

[Pollack 72] B. W. Pollack. *Compilers techniques*. Auerbach Publishers, Princeton (USA), 1972.

[Préau http] Le Préau. Association, nouvelles technologies Éducatives. http ://www.preau.asso.fr/.

[PrologIA 95] PrologIA. Manuel de référence pour PrologII+, 1995.

[Py 90] D. Py. *Reconnaissance de plan pour l'aide à la démonstration dans un tuteur intelligent de la géométrie*. Thèse de Doctorat, Université de Rennes I - Institut de Formation Supérieur en Informatique et Communication, 1990.

[Qasem 97] S. Qasem. *Conception et réalisation d'une interface 3D pour Cabri-Géomètre*. Thèse de Doctorat, Université Joseph Fourier, Grenoble I, 1997.

[Quaife 89] A. Quaife. Automated Development of Tarsky's Geometry. *Journal of Automated Reasoning*, 5 :97–118, 1989.

[Quéré 91] M. Quéré et al. *Systèmes experts et enseignement assisté par ordinateur*, volume 7, série *Collection Autoformation et Enseignement Multimédia*. Ophrys, 1991.

[Ritter 95] S. Ritter et K. R Koedinger. Toward lightweight tutoring agents. *World Conference on Artificial Intelligence in Education, AIED'95*, pages 91–98, 1995.

[Ritter 96] S. Ritter et K. R Koedinger. An Architecture for Plug-in Tutor Agents. *Journal of Artificial Intelligence in Education*, 7(3/4) :315–347, 1996.

[Ritter 97] S. Ritter. Communication, Cooperation and Competition among Multiple Tutor Agents. Ben Du Boulay et Riichiro Mizogushi, éditeurs, *International Conference on Artificial Intelligence in Education, AIED'97, Kobe, Japan*, pages 31–38. IOS Press, 1997.

[Ritter 98] S. Ritter, P. Brusilovsky et O. Medvedeva. Creating more versatile intelligent learning environments with a component-based architecture. Henry M. Goettl, Barry P. anf Halff, Carol L. Redfield et Valerie J. Shute, éditeurs, *4th International Conference Intelligent Tutoring Systems, ITS'98, San Antonio, Texas, USA*, volume 1452, série *LNCS*, pages 554–563. Springer, august 1998.

[Roncancio 94] C. Roncancio. Interopérabilité entre SGBD : systèmes fédérés et systèmes multibases. *Technique et Science Informatiques*, 13(3) :385–419, 1994.

[Senet 93] C. Senet. Communication entre TALC et CABRI-Géomètre. Rapport de stage de maîtrise des sciences et techniques, expert en systèmes informatiques, Université Joseph Fourier, Grenoble I, 1993.

[Suthers 97] D. Suthers et D. Jones. An Architecture for Intelligent Collaborative Educational Systems. Ben Du Boulay et Riichiro Mizogushi, éditeurs, *International Conference on Artificial Intelligence in Education, AIED'97, Kobe, Japan*. IOS Press, 1997.

[Tahri 93] S. Tahri. *Modélisation de l'interaction didactique : un tuteur hybride sur Cabri-géomètre.* Thèse de Doctorat, Université Joseph Fourier, Grenoble I, 1993.

[Tessier 94] S. Tessier et J.-M. Laborde. Descriptions des événements apple acceptés par cabri-géomètre. Rapport technique no. RT 105, IMAG, 1994.

[TopClass http] TopClass. Produit e-learning de WBTsystems. http ://www.wbtsystems.com/

[Trilling 96] L. Trilling. Rétrospective du projet mentoniezh. *Sciences et Technologies Éducatives*, 3(2) :157–162, 1996.

[Ulbricht 97] V. R. Ulbricht, N. dos Santos et R. S. Wazlawick. VISUAL GD : hypermedia environment for Descriptive Geometry. *Graf & Tec*, 2(1) :9–38, dec 1997.

[UML http] UML. Unified Modeling Language. http ://uml.free.fr/

[VisiBroker http] VisiBroker. ORB CORBA de Borland©. http ://www.borland.fr/produits/VisiBroker/index.asp

[Van Labeke 99] N. Van Labeke. *Prise en compte de l'usager enseignant dans la conception des EIAO. Illustration dans Calques 3D.* Thèse d'université, Université Henri Poincaré, Nancy I, décembre 1999.

[VNC http] VNC. Virtual Network Computing. http ://www.uk.research.att.com/vnc/

[Wassernan 89] A. I. Wassernan. Tool integration in software engineering environments. Fred Long, éditeur, *Software Engineering Environments*, volume 467, série *LNCS*, pages 137–149. Springer-Verlag, 1989.

[WebCT http] WebCT. Produit e-learning de WebCT Inc. http ://www.webct.com/

[Wiederhold 92] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3) :38–49, 1992.

[Wimageo http] Wimageo. Imageo pour Windows http ://www.lille.iufm.fr/lilimath

[Yellin 88] D. M. Yellin. *Attribute Grammar Inversion and Source-to-Source Translation*, volume 302, série *LNCS*. Springer Verlag, 1988.

Liste des publications

Article dans une revue nationale avec comité de lecture

[1] M. Macrelle. Du formateur vers l'EIAO : étudier des stratégies de communication. *Revue Informations In Cognito, Grenoble, France*, 9, août 1997.

Conférences internationales avec comité de lecture

Articles

[2] M. Rosselle et M. Grandbastien. Experimenting Features from Distinct Software Components on a Single Platform. Gilles Gauthier, Claude Frasson et Kurt VanLehn, éditeurs, 5^{th} *International Conference Intelligent Tutoring Systems, ITS'2000*, volume 1839, série *LNCS*, pages 163–172, Montreal, Canada, juin 2000. Springer.

[3] M. Macrelle et C. Desmoulins. Macro-Definitions, a Basic Component for Interoperability between ILEs at the Knowledge Level : Application to Geometry ILEs. 4^{th} *International Conference on Intelligent Tutoring Systems, ITS'98*, volume 1452, série *LNCS*, pages 46–55, San Antonio, Texas, USA, août 1998. Springer Verlag.

Posters

[4] M. Macrelle. Specification of the Educational Mediator Architecture : an Inter-operation Architecture for Educational Software. *Conference on Artificial Intelligence in Education, AIED'99*, pages 735–737, Le Mans, France, septembre 1999. International AIED Society. IJAIED publi, IOS Press.

Conférences nationales avec comité de lecture

Articles

[5] M. Macrelle. Partager des connaissances via macro-Définitions : un outil pour l'interopération entre EIAHs. *4ième Rencontres des Jeunes Chercheurs en Intelligence Artificielle - RJCIA'98*, Toulouse, France, 1998.

[6] C. Desmoulins et M. Macrelle. Interopérer via des macro-définitions pour partager des connaissances Application aux EIAHs de géométrie. *Ingénierie des connaissances, IC'98*, pages 221–230, Pont-à-Mousson, France, mai 1998. Loria.

[7] M. Macrelle. Un étayage portant sur des connaissances limitées, expérimentation en aérodynamique. *1ières Journées Francophones : Sciences de la Cognition vers les Applications, Villeneuve d'Ascq, France. Association Scicoia, juillet 1997.*

Posters

[8] M. Macrelle. Du formateur vers l'EIAO : un étayage portant sur des connaissances limitées. *5ièmes journées Environnements interactifs d'apprentissage avec Ordinateur, EIAO'97,* pages 294–295, Cachan, France, mai 1997. Hermès.

Conférences nationales sans comité de lecture

Articles

[9] M. Macrelle. Présentation de CHyPre : un logiciel d'aide à la résolution de problèmes en géométrie. *colloque inter IREM de géométrie*, Bussang, France, juin 1998. IREM de lorraine.

[10] J. Morinet-Lambert, M. Macrelle, A. Bronner et J. Cochet. Évaluation des enseignements à l'Université Henri Poincaré. *Colloque Interuniversitaire sur les Méthodes d'Évaluation - CIME'98*, Poitiers, France, juillet 1998.

Support de cours

[i] J. Morinet-Lambert, M. Macrelle et Joseph Rouyer. Informatique - Algorithmique. Applications en Pascal, septembre 1998.

Mémoires et rapports de recherche

[ii] M. Rosselle et M. Grandbastien. Expérimenter des fonctionnalités issues d'EIAO différents sur une plate-forme unique. Rapport interne, LORIA-UHP, 2001.

[iii] M. Macrelle. Specification of EMA (Educational Mediator Architecture) - an Inter-operation Architecture for Educational Software. Rapport interne, LORIA-UHP, 1998.

[iv] M. Macrelle. HDL to CDL Macro-Definition Set. Rapport interne, LORIA-UHP, 1998.

[v] M. Macrelle. Du formateur vers l'EIAO : un étayage portant sur des connaissances limitées. Rapport de DEA de sciences cognitives, LORIA-UHP, 1996.

Annexes

Annexe A

Liste de fonctionnalités pour la géométrie

Nous rappelons ci-dessous, la liste de fonctionnalités relevées dans l'état de l'art pour le domaine de la géométrie :

F1. aide au raisonnement par application guidée du cours ;
F2. aide au raisonnement par visualisation de l'état de résolution ;
F3. aide à la décomposition d'un problème en sous-problèmes ;
F4. aide à la rédaction d'une démonstration ;
F5. aide à l'analyse d'énoncé ;
F6. aide dans l'élaboration d'une démonstration ;
F7. aide dans l'interaction ;
F8. analyse de l'énoncé ;
F9. analyse des interactions ;
F10. diagnostic de besoin d'aide ;
F11. diagnostic de correction de figure ;
F12. diagnostic d'analyse de l'énoncé ;
F13. exploration conceptuelle de figure géométrique ;
F14. exploration visuelle de figure géométrique ;
F15. exportation de la figure ;
F16. extraction de sous-figures ;
F17. édition de figure géométrique ;
F18. présentation de figure ;
F19. rappels de cours ;
F20. trace des objets créés à l'interface ;
F21. vérification d'une propriété.

Cette liste est évidemment provisoire, d'autres fonctionnalités viendront la compléter.

Annexe B

Fiche pour caractériser une étape

Nous présentons ci-dessous le formulaire à remplir pour décrire une étape.

Consigne

Outils

Fonctionnalité : .....

Prototype : .....

Moment de fermeture : .....

Données à recueillir

Résultat produit : .....

Connaissances d'interaction : .....

Autres (informations scrutables) : .....

Présentation à l'interface

	position Vert	position Horiz	tailleLarg	tailleHaut	déformable	déplaçable	maximisable
Consigne :			%	%			
Outil :			%	%			
Transition :			%	%			

Transition

Étape suivante : .....

Condition de transition à l'étape suivante : .....

Chaque champ est documenté sur la figure 4.7 (page 119).

Nous avons illustré l'utilisation de cette fiche avec la première étape de l'exemple introductif (voir figure 1.6 p 16).

Annexe C

Exemple d'application de l'interprète de macro-définitions

Traduction de HDL vers CDL

Nous avons présenté dans la section 3.6 le concept de macro-définition et son implantation (section 4.6). Rappelons qu'une macro-définition permet la traduction d'une connaissance exprimée dans un langage source vers une connaissance exprimée en langage cible. Elle est composée d'une partie gauche appelée `MacroTete`, d'une partie droite appelée `MacroCorps`, séparées l'une de l'autre par le *constructeurDeMacroDef* (le symbole ← ici). La `MacroTete` est un atome du *langage source*, le `MacroCorps` est une phrase du *langage destination* (utilisant éventuellement des macro-définitions via des `macroUtilisations`).
Pour montrer la faisabilité du concept de macro-définition, nous avons choisi de l'implanter dans le domaine de la géométrie, pour lequel nous avons une expérience importante et où un nombre important de Logiciels Éducatifs aux fonctionnalités diverses a été développé dans la communauté de recherche française. Après avoir justifié le choix des deux outils *TALC* [Desmoulins 94] et *Mentoniezh* [Py 90] pour implanter une première interopérabilité au niveau connaissances, nous présentons comment un langage de macro-définitions en géométrie, macro-CDL, nous a permis de réaliser très simplement cette interopérabilité. Nous montrons ensuite comment les contraintes de traductibilité (définies page 104) sont respectées par les langages de représentation des connaissances de *TALC* et *Mentoniezh*. Nous présentons alors le langage macro-CDL obtenu et la façon dont nous avons résolu les problèmes de non-conformité avec les contraintes de traductibilité.

C.1 *TALC* et *Mentoniezh*, deux Logiciels Éducatifs à faire interopérer au niveau connaissances

Le plus efficace étant de n'avoir à traiter que le problème de l'échange des connaissances en géométrie, il fallait trouver deux Logiciels Éducatifs tournant sur le même matériel, le même système d'exploitation et programmés dans le même langage. Un couple de tels Logiciels Éducatifs n'existait pas. Cependant, pour des raisons historiques [Trilling 96], les outils *TALC* et *Mentoniezh* étaient tous deux programmés dans le même langage, Prolog *II* [Le Huitouze 88, PrologIA 95]. Ce langage possède le grand avantage d'être compilé dans une représentation intermédiaire que des interprètes exécutent de façon identiques sur de nombreuses machines et

systèmes. Il a suffit alors de reprogrammer les aspects interface de l'un des deux outils (ce fut *Mentoniezh*), pour que l'interopérabilité ne pose plus de problème au niveau matériel (Macintosh), système d'exploitation (MacOS) et langage de programmation, mais seulement au niveau de l'échange des connaissances. Le but de *TALC* est de vérifier que la construction faite par un apprenant correspond à l'énoncé d'une figure donnée par un enseignant. Pour que l'apprenant réalise la construction de sa figure, *TALC* inter-opère avec *CABRI-Géomètre* via les primitives de communications de MacOS appelées Apple-events [Apple Computer 93, Tessier 94], de la même manière que Ritter & Koedinger [Ritter 96]. L'interopérabilité au niveau connaissances est réalisée comme décrit plus haut par un traducteur spécifique d'énoncés *CABRI-Géomètre* directement vers le langage interne de *TALC* (nommé LDL pour Logical Description Language). *Mentoniezh* est un tuteur intelligent pour la démonstration en géométrie euclidienne plane. Il a pour objet, à partir de l'énoncé d'une figure représentant des hypothèses et d'une propriété à démontrer, d'aider l'apprenant à élaborer et à rédiger une démonstration de cette propriété dans ces hypothèses. Il est programmé sur un PC sous DOS. Malgré l'objectif initial du projet *Mentoniezh* [Allen 90, Trilling 96], aucune interface de construction et de vérification de figure n'est disponible. Ces fonctionnalités sont de ce fait actuellement réalisées à la main sur un support papier. Ainsi l'interopérabilité entre *TALC* et *Mentoniezh* est très intéressante car elle permet d'utiliser la complémentarité des deux Logiciels Éducatifs et d'obtenir un Logiciels Éducatifs répondant à la définition d'origine du projet *Mentoniezh*.

C.2 Conformité des langages de *TALC* et *Mentoniezh* aux contraintes de traductibilité

Dans cette partie nous présentons les langages CDL et HDL, qui représentent tous deux des connaissances géométriques, c'est-à-dire des objets et des relations entre ces objets. Nous expliquons ensuite dans quelle mesure les contraintes de traductibilité sont respectées par ces langages.

C.2.1 Le langage CDL

L'outil *TALC* permet à l'enseignant de décrire une figure géométrique qu'il veut que l'apprenant réalise. Pour cela, l'enseignant dispose d'un langage logique CDL (Classroom Description Language), proche de celui des manuels scolaire en France. CDL permet de décrire des énoncés géométriques comportant des objets (point, droite, demi-droite, cercle et distance) et des relations les liant (appartenance, parallélisme, orthogonalité, égalité, *etc.*). CDL est un langage déclaratif où les atomes sont chacun une propriété portant sur des objets décrits par des termes (comme [A, B] pour un segment) ou des identificateurs (voir exemple C.6).

Exemple C.6 *Dans cet exemple, l'enseignant définit un parallélogramme de sommets A, B, C et D, une droite L1 passant par B et perpendiculaire au segment [B C], une droite L2 passant par D et perpendiculaire au segment [A D] (voir figure C.1).*

C.2.2 Le langage HDL

L'outil *Mentoniezh* permet à l'enseignant de décrire les hypothèses et la conclusion du théorème qu'il veut que l'apprenant démontre (*cf.* exemple C.7). Pour cela, l'enseignant dispose d'un langage logique HDL (Hypothesis Description Language). HDL permet de décrire des énoncés géométrique comportant des objets de base (point, droite et cercle) et des objets composés

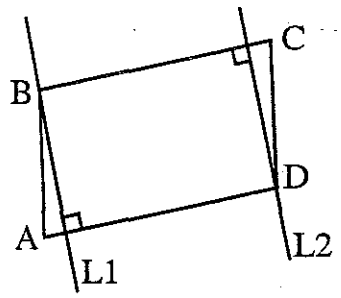


Figure C.1 – Figure illustrant l'exemple C.6

Langage CDL	**Sémantique CDL**
$[AB]//[CD]$	Le segment d'extrémités A et B et celui d'extrémités C et D sont parallèles
$[BC]//[AD]$	Le segment d'extrémités B et C et celui d'extrémités A et D sont parallèles
$L1 \perp (BC)$	La droite L1 est perpendiculaire à la droite passant par les points B et C
$L2 \perp (AD)$	La droite L2 est perpendiculaire à la droite passant par les points A et D
$B \in L1$	Le point B appartient à la droite L1
$D \in L2$	Le point D appartient à la droite L2

Table C.1 – Énoncé de l'exemple en CDL

(triangle, rectangle, triangle-rectangle, parallélogramme, *etc.*) et des relations les liant (appartenance, parallélisme, orthogonalités, quadrilatère non croisé, *etc.*). HDL est un langage déclaratif où les atomes sont chacun un prédicat de propriété dont les paramètres sont des identificateurs représentant des objets de base.

Exemple C.7 *L'énoncé suivant définit comme hypothèses l'énoncé de l'exemple C.6, et comme conclusion le fait que les droites L1 et L2 sont parallèles.*
Hypothèses : parallélogramme (A, B, C, D), perpendiculaire (L1, B, C), perpendiculaire (L2, A, D), appdroite(B, L1), appdroite(D, L2).
Conclusion : parallèle (L1, L2).

C.2.3 Conformité des langages de *TALC* et *Mentoniezh* par rapport aux contraintes de traductibilité

Seule la propriété HDL «quadrilatère non-croisé» ne permet pas à CDL de recouvrir HDL car elle n'est pas exprimable en CDL. Nous considérons dans un premier temps HDL privé de cette propriété, ce qui permet de dire qu'il est couvert par CDL et nous traitons le problème de cette propriété plus loin. La granularité du langage CDL est plus fine que celle d'HDL. En effet, le langage CDL représente par un atome des objets de type point et segment et HDL représente par un atome des objets de type triangle, rectangle et parallélogramme, qui se décomposent en point et segment. Ainsi le langage HDL est traduisible dans CDL. Pour tout atome du langage HDL, il est donc possible de trouver un énoncé en CDL qui représente la même connaissance. Nous définissons donc un langage de macro-définitions pour la traduction de HDL en CDL, que nous appelons macro-CDL.
Pour obtenir une généralisation totale pour le domaine de la géométrie plane, il nous faudrait un inter-langage dont la granularité soit la plus fine et qui recouvre tout le domaine. Dans ce contexte, tout langage géométrique serait traduisible dans ce langage. Notons qu'il existe de tels

langages couvrant toute la géométrie et dont la granularité est plus fine que CDL, par exemple, celui utilisé dans l'axiomatique de Tarski [Quaife 89]. Cependant ce langage n'est pas adapté à l'enseignement et ne peut être utilisé dans le contexte des Logiciels Éducatifs. Définir un tel langage de représentation de la géométrie plane pour l'enseignement, est précisément un des objectifs de notre plate-forme. Il pourrait constituer une ontologie [Gruber 93] pour l'enseignement de la géométrie plane.

C.3 Des macro-CDL pour HDL

Pour un atome HDL (c'est-à-dire un prédicat et ses arguments), nous devons définir une macro-CDL correspondante. Plusieurs cas se présentent suivant le nombre de façons de définir cette macro-CDL. Le cas le plus simple est celui où une seule macro-CDL correspond à un prédicat. Un autre cas est celui où plusieurs façons d'exprimer le corps de la macro-CDL sont possibles. Enfin le dernier cas est celui où les contraintes de traductibilité ne sont pas respectées, alors il n'existe *a priori* pas de macro-définition qui corresponde (la traductibilité n'étant plus assurée). Pour chacun de ces trois cas, nous présentons un exemple dans ce qui suit (l'ensemble des macro-définitions est donnée dans [iv]).

C.3.1 Une seule macro-définition possible

La propriété parallèle est un exemple de propriété HDL définissable par une seule macro-CDL :

Exemple C.8 $Parallèle(L1, L2) \leftarrow droite(L1), droite(L2), L1//L2.$

C.3.2 Plusieurs macro-définitions possibles

La propriété aligné est un exemple de propriété HDL définissable par plusieurs macro-CDL.

Exemple C.9 *Traduction du prédicat HDL «alignés (P1, P2, P3)». Le prédicat HDL «alignés (P1, P2, P3)» peut être défini par au moins trois macro-CDL différentes (sans compter toutes les permutations de P1, P2 et P3) dont les corps sont les suivants :*

- $P3 \in (P1P2)$
  Le point P3 appartient à la droite (P1 P2) ;
- $(P1P2) = (P1P3)$
  Les droites (P1 P2) et (P1 P3) sont égales ;
- $(P1P2)//(P1P3)$
  Les droites (P1 P2) et (P1 P3) sont parallèles et elles ont au moins un point en commun.

L'enseignant a plusieurs choix possibles. Selon le contexte, il peut vouloir définir des macro-définitions les plus générales possibles, ou des macro-définitions faisant intervenir les objets de la partie de cours actuelle, ou faisant référence aux connaissances supposées de l'apprenant, ou tout autre choix. Dans le contexte de *TALC*, il doit s'assurer que la sémantique de l'énoncé CDL qu'il choisit correspond bien à son idée.

C.3.3 Aucune macro-définition possible

Le prédicat HDL non-croisé (P1, P2, P3, P4) n'est pas traduisible en CDL. Il définit qu'un quadrilatère est non croisé, c'est-à-dire qu'aucun de ses quatre côtés ne coupe un des autres. Sur la Figure C.2, le quadrilatère de gauche est non croisé, les deux quadrilatères de droite sont

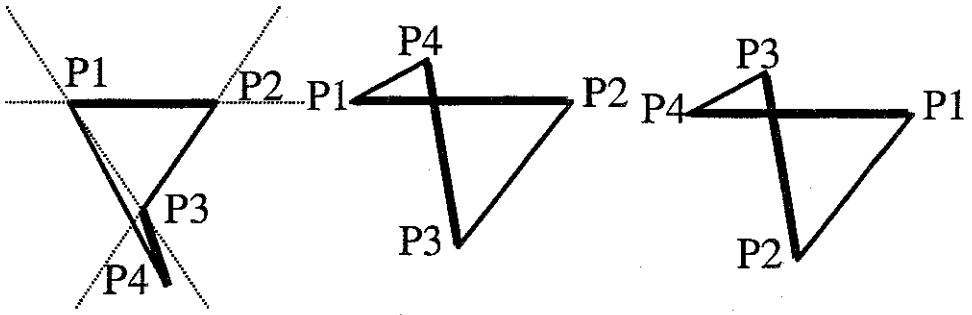


Figure C.2 – Quadrilatères non croisé et croisés

croisés.
Pour exprimer géométriquement, qu'un quadrilatère est non croisé, nous exprimons soit directement qu'un de ses côtés coupe le côté opposé soit que les points C et D sont dans le demi-plan délimité par la droite (AB). Ces deux solutions ne sont pas exprimables en CDL. La première solution n'est pas exprimable en CDL car la négation d'une conjonction est une disjonction, qui ne sont pas exprimables en CDL. En effet, les disjonctions permettraient à l'enseignant de définir un énoncé pour lequel deux constructions exclusives seraient nécessaires et non une seule (par exemple, un énoncé où deux cercles sont sécants ou bien tangents). La seconde solution n'est pas non plus exprimable en CDL parce que l'objet demi-plan n'y est pas défini.
Dans le contexte des Logiciels Éducatifs, un outil peut exprimer certaines connaissances qui ne sont pas traitées directement par le système à base de connaissance. L'outil fait alors appel à des connaissances extérieures à ce système à base de connaissances. Cette méthode peut aussi être utilisée pour permettre à l'enseignant d'exprimer qu'un quadrilatère est non croisé en effectuant un calcul si l'outil gère une représentation géométrique analytique, sinon en demandant à l'apprenant lui-même si un quadrilatère est croisé ou non.
C'est d'ailleurs cette dernière méthode qu'emploie *Mentoniezh* pour le prédicat HDL «non croisé». Ce prédicat n'est donc pas un prédicat du langage au sens strict. Pour cette raison et parce que les trois autres solutions ne sont pas réalisable sans modifier *TALC*, nous ne les avons pas implantées. Nous avons choisi de ne pas tenir compte de l'intégralité de la sémantique de ce prédicat HDL et de l'exprimer simplement avec la macro-définition :
Noncroisé$(P1, P2, P3, P4) \leftarrow non(C \in (AB)), non(D \in (BC)), non(A \in (DC)), non(B \in (DA))$.
Nous supposons donc que cette connaissance, qui est généralement implicite dans les énoncés de géométrie, est aussi implicitement respectée par l'apprenant.

C.4 Implantation

Comme nous l'avons expliqué précédemment, une version de *Mentoniezh* a été préalablement implantée sur PROLOGII+ pour éviter les difficultés d'interopérabilité au niveau matériel, système et outil et pouvoir se focaliser sur le niveau connaissances. Notre objectif est que l'apprenant puisse construire dans *TALC* une figure correspondant aux hypothèses de *Mentoniezh* et qu'en

cas de vérification positive, il puisse enchaîner directement l'élaboration de la démonstration de la conclusion par rapport à ses hypothèses. L'interaction *TALC-Mentoniezh* est donc de la forme «l'un puis l'autre», via un transfert de connaissances. Pour remplir cet objectif, nous avons défini un interprète de macro-CDL qui prend en entrée un fichier contenant les macro-définitions pour *Mentoniezh* écrites en macro-CDL et un énoncé en HDL et produit l'énoncé CDL correspondant. L'énoncé CDL est ensuite fourni à *TALC* et l'énoncé HDL à *Mentoniezh*. En pratique, l'interprète macro-CDL suit la définition de la partie 3, c'est-à-dire qu'il prend les macro-définitions dans l'ordre du fichier des macro-définitions. Il opère successivement les substitutions des occurrences d'utilisation de macro-définition dans le fichier contenant l'énoncé en HDL. Pour cela, l'interprète est construit classiquement par la succession d'un analyseur lexical, d'un analyseur syntaxique et d'un analyseur sémantique suivant la grammaire :

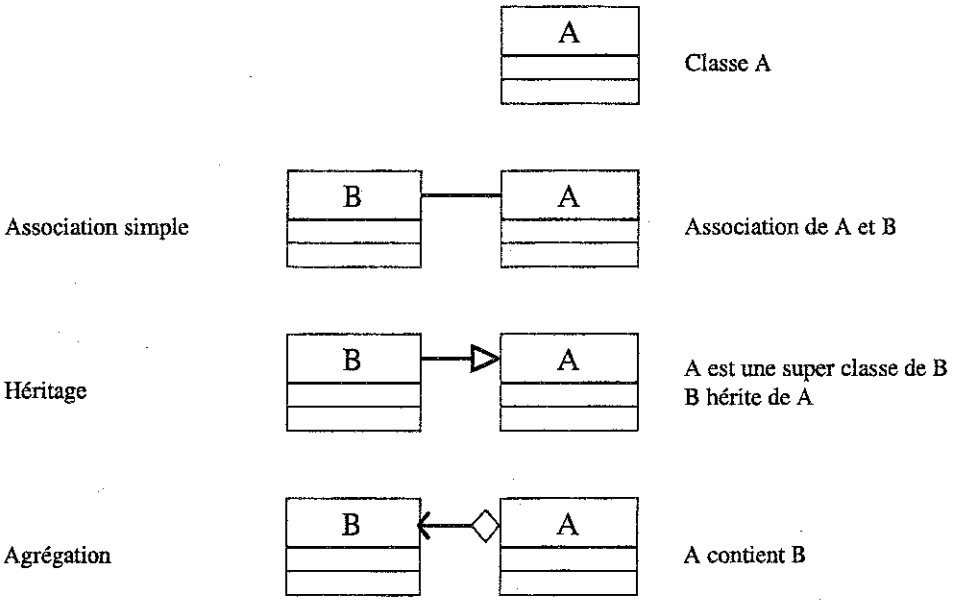
macroTexte : := macroDéfinition *finDeFichier*
| macroDéfinition macroTexte
macroDéfinition : := macroTête ← macroCorps .
macroTête : := *PhraseEnLangageSource*
macroCorps : := macroPhrase
| macroPhrase , macroCorps
macroPhrase : := *PhraseEnLangageDestination*
| *macroUtilisation*

Pour dériver les non terminaux PhraseEnLangageSource et PhraseEnLangageDestination nous utilisons les fonctions qui dérivent ces atomes dans chaque outil. Cette implantation ne nécessite donc qu'un faible travail.

Maintenant que notre interprète de macro-CDL est implanté, l'interopérabilité entre *TALC* et un autre système, dont le langage serait traduisible en CDL, peut être réalisée sans intervention du concepteur pour modifier *TALC*, par simple définition dans un fichier texte des macro-définition adaptées à ce langage.

Annexe D

Rappel des notations UML utilisées



Annexe E

Les classes java de l'atelier

La figure E.1 présentent les classes principales de l'atelier sans attribut et sans méthode. Elle

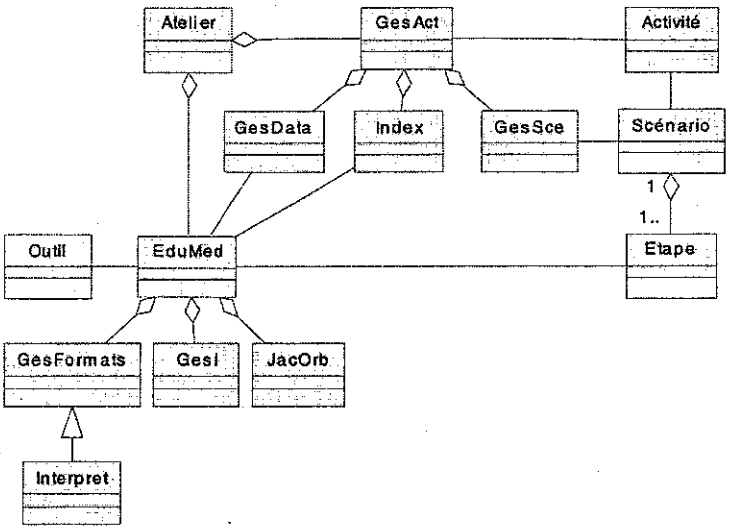


Figure E.1 – Classes Principales non détaillées

permet d'avoir une vue globale de l'atelier.

La figure E.2 présentent les classes principales de l'atelier avec attributs et avec méthodes

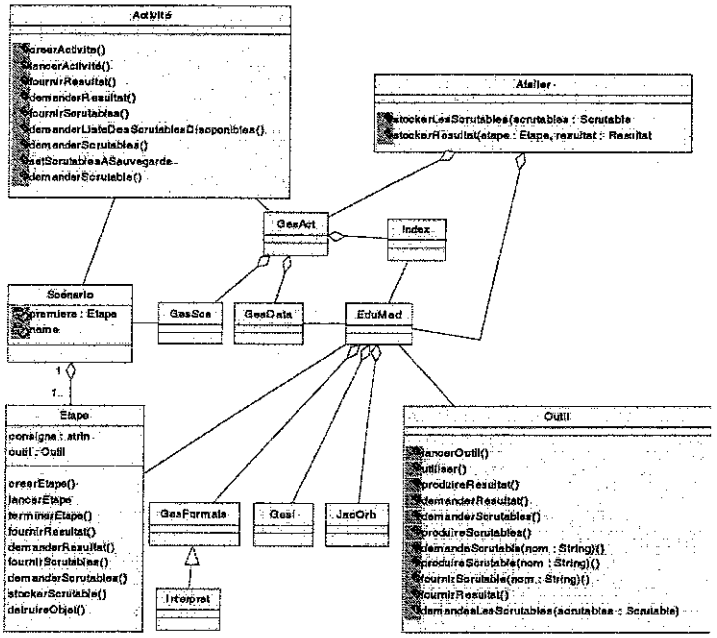


Figure E.2 – Classes Principales détaillées

Annexe F

Cas d'utilisation et scénarios

Nous détaillons ici tous les cas d'utilisations et les diagrammes de séquences correspondant qui ne sont pas décrits dans la chapitre 3.

F.1 Gérer les fonctionnalités

Les paragraphes suivants reprennent chacun des cas d'utilisation de la figure 1.19 p 34.

Cas d'utilisation : Afficher la liste des fonctionnalités

Nom :	Afficher la liste des fonctionnalités
Résumé des responsabilités :	Ce cas d'utilisation permet à l'administrateur d'obtenir la liste de toutes les fonctionnalités chargées dans l'atelier pour le domaine d'application choisi.
Acteurs :	L'administrateur et l'atelier.
Pré-conditions	L'usager doit être un administrateur et avoir effectué la procédure d'identification.
Post-condition	La liste des fonctionnalités est obtenue
Description des interactions :	– L'administrateur demande la liste des fonctionnalités. – L'atelier lui retourne la liste des fonctionnalités chargées.
Cas reliés	

Scénario : afficher la liste des fonctionnalités

Cas normal.
Diagramme de séquences (*cf.* figure F.1) décrivant les interactions de haut niveau pour afficher la liste des fonctionnalités :

F.2 Gérer les prototypes

Les paragraphes suivants reprennent chacun des cas d'utilisation de la figure 1.18 p 34.

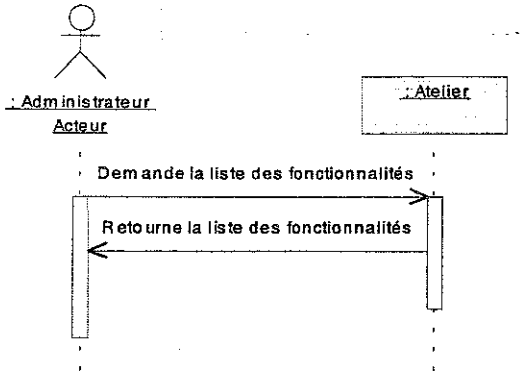


Figure F.1 – Diagramme de séquences : afficher la liste des fonctionnalités

Cas d'utilisation : Afficher la liste des prototypes

Nom :	Afficher la liste des prototypes
Résumé des responsabilités :	Ce cas d'utilisation permet à l'administrateur d'obtenir la liste de toutes les prototypes chargés dans l'atelier pour le domaine d'application choisi.
Acteurs :	L'administrateur et l'atelier.
Pré-conditions	L'usager doit être un administrateur et avoir effectué la procédure d'identification.
Post-condition	La liste des prototypes est obtenue
Description des interactions :	– L'administrateur demande la liste des prototypes. – L'atelier lui retourne la liste des prototypes chargés.
Cas reliés	

F.3 Gérer les usages

Les paragraphes suivants reprennent chacun des cas d'utilisation de la figure 1.17 p 33.

Cas d'utilisation : Afficher la liste des usagers

Nom :	Afficher la liste des usagers.
Résumé des responsabilités :	Ce cas d'utilisation permet à l'administrateur d'obtenir la liste de toutes les usagers du système.
Acteurs :	L'administrateur et l'atelier.
Pré-conditions	L'usager doit être un administrateur et avoir effectué la procédure d'identification.
Post-condition	La liste des usagers est obtenue
Description des interactions :	– L'administrateur demande la liste des usagers. – L'atelier lui retourne la liste des usagers chargés.
Cas reliés	

Index

Glossaire

Acteur : Personne ou composant à l'origine d'une interaction avec le système.

Activité : (Larousse) action d'une personne, d'une entreprise, d'une nation dans un domaine défini.
(Ici) ensemble de tâches que le sujet doit exécuter. C'est une expérimentation si le prescripteur est chercheur. C'est une activité pédagogique (un exercice) si le prescripteur est un enseignant.

ADL : Advanced Distributed Learning.

Administrateur : Utilisateur qui prend en charge tous les aspects techniques de l'atelier.

AICC : Aviation Industry CBT Committee.

ANSI : American National Standards Institute.

API : Application Program Interface.

ARIADNE : Alliance of Remote Instructional Authoring and Distribution Networks for Europe.

Atelier : (Ici) Atelier d'expérimentation de logiciels éducatifs intelligents.

BOA : Basic Object Adapter.

CAI : Computer Aided Instruction.

CAL : Computer Aided Learning.

Cas d'utilisation : Objectif su système, motivé par un besoin d'un acteur (au moins). Par abus de language désigne aussi un groupe de cas d'utilisation.

CBT : Computer Aided Teaching.

CDL : Classroom Description Language.

Composant : (Larousse) élément constitutif d'un ensemble complexe.
(Atelier) équipement de l'atelier destiné à une fonction déterminée.

Composant de service : Composant qui assure un service pour l'atelier, cad une fonction non éducative (par opposition à composant Logiciels Éducatifs).

Composant Logiciels Éducatifs : Composant qui assure une fonctionnalité éducative pour l'atelier.

Conjecture : Fait à prouver.

Connaissances d'interactions : Connaissances sur l'interaction du sujet avec un outil ou avec l'atelier.

Connaissances du domaine : Connaissances se rapportant au domaine d'apprentissage ou d'enseignement.

Connaissances pédagogiques : Connaissances sur les stratégies pédagogiques ou tutorielles.

Connaissances sur l'utilisateur/apprenant : Connaissances qui permettent d'adapter le comportement du système à l'utilisateur/apprenant en général (modèle de l'utilisateur/apprenant) ou à un utilisateur/apprenant particulier (profil de l'utilisateur/apprenant).

Consigne : (Larousse) instruction formelle donnée à qqn qui est chargé de l'exécuter.
(Ici) description de la tâche à réaliser donnée au sujet de l'activité.
CORBA : Common Object Request Broker Architecture.
CSCL : Computer-Supported Collaborative Learning c'est-à-dire apprentissage collaboratif assisté par ordinateur.

Déformable : Propriété d'une fenêtre dont la taille (largeur, hauteur) peut être changé.
Déplaçable : Propriété d'une fenêtre dont la position peut être changé (horizontalement et verticalement).
DCE : Data Circuit-terminated Equipment.
DCE : Data Communication Equipment.
DCE : Distributed Computing Environment (from OSF).
DCE : Distributed Computing Environment.
Diagramme de cas d'utilisation : Représentation des fonctions du système du point de vue de l'utilisateur.
Diagramme de collaboration : Représentation spatiale des objets et de leurs interactions.
Diagramme de séquences : Représentation temporelle des objets et de leurs interactions.
DII : Dynamic Invocation Interface.

EAO : Enseignement Assisté par Ordinateur.
EAT : Enseigner et Apprendre avec les Technologies nouvelles.
EduMed : Educational Mediator (defined in this work).
EduMed : Médiateur Éducatif (défini dans ce mémoire).
EIAH : Environnements Interactifs d'Apprentissage Humain.
EIAO : Enseignement Intelligemment Assisté par Ordinateur.
EIAO : Environnements Intelligents d'Apprentissage avec Ordinateur.
EIAO : Environnements Interactifs d'Apprentissage avec Ordinateur.
Epiphyte : Un logiciel épiphyte est un logiciel qui se greffe à un logiciel existant, afin de recueillir les données dont il a besoin et ce sans gêner le fonctionnement de l'autre logiciel (sinon le logiciel serait parasite au lieu d'épiphyte).

Fonction : (Ici) fonction offerte par l'atelier.
Fonctionnalité : (Ici) fonction implantée dans un Logiciels Éducatifs. Exemples : explication, simulation, résolution de problème, diagnostic, support dans la réalisation d'une tâche élémentaire.

GNU : Gnu is Not Unix.
GUI : Globally Unique Identifier.

HDL : Hypothèse (?) Description Language.
HOD : Head of Delegation.

ICAI : Intelligent Computer Aided Instruction.
IDL : Interface Definition Language, le langage du standard de l'OMG pour définir les interfaces de tous les objets CORBA. La syntaxe et sémantique completes d'IDL sont disponibles dans le chapitre 3 de la spécification de l'OMG, sur le site de l'OMG..
IEC : International Electrotechnical Commission.
IEEE : Institute for Electrical and Electronic Engineers.
IES : Intelligent Educational Software.

IMS : Educom's Instructional Management Systems.
ISO : International Organization for Standardization.
ITS : Intelligent Tutoring Systems.

JTC1 : Joint Technical Committee number 1.

KQML : Knowledge Query and Manipulation Language.

Langage naturel : Langage utilisé par les humains, par opposition aux langages informatiques.
LTSA : Learning Technology Systems Architecture.
LTSC : Learning Technology Standards Committee.

Maximisable : Propriété d'une fenêtre qui peut être occupé tout l'écran disponible. Une fenêtre qui a été maximisée, peut reprendre sa taille d'origine.
Middleware : Équipement qui se place au milieu d'un ensemble de logiciels, gérant la communication de ces logiciels.

NB : National Body.

OMG : Object Management Group, an international organization with over 700 members that establishes industry guidelines and object management specifications in order to provide a common framework for object-oriented application development. Its members include platform vendors, object-oriented database vendors, software tool developers, corporate developers, and software application vendors. The OMG Common Object Request Broker Architecture specifies the CORBA object model. See www.omg.org for more information..
ORB : Object Request Broker, un ORB est un équipement logiciel qui se place au milieu d'un ensemble de logiciels (c'est-à-dire un type de middleware), pour gérer leur communication..
OSF : Open Software Foundation.
Outils : Objet fabriqué, utilisé manuellement ou sur une machine pour réaliser une opération déterminée (Larousse en ligne).Ici, fonctionnalité d'un prototype utilisée par le sujet au cours de l'activité.

Prescripteur : Utilisateur de l'atelier qui prescrit l'activité exécutée par le sujet avec l'atelier. Il est soit chercheur soit enseignant.
Prototype : (Ici) logiciel ou produit issu la recherche en EIAO.

RPC : Remote Procedure Call.

SC : Subcommittee.
Scénario UML : Instance d'un cas d'utilisation. Déroulement prévu d'un cas d'utilisation. Chaque déroulement est décrit par un diagramme de séquences ou un diagramme de collaboration.
SEC : IEEE LTSC Sponsor Executive Committee.
SG : Study Group.

TAG : Technical Advisory Group.
TI : Tuteurs Intelligents.
Trace : Élément d'un historique.

Traces d'interaction : Observations recueillies auprès d'un système lorsque l'utilisateur interagit avec lui.

UML : Unified Modeling Langage.

Vitrine : Lieu de présentation publique. (Ici) Ensemble d'informations (objets, prédicats, méthodes) présentées publiquement. Un composant publie les informations accessibles pour les autres composant dans sa vitrine. Spécification publique d'un composant.

WBT : Web-Based Training.
WG : Working Group, groupe de travail.
WOSIT : Widget Observation Scripting and Inspecting Tool.

Résumé

Cette recherche se situe dans le contexte des EIAO (Environnements Interactifs d'Apprentissage avec Ordinateur). Les nombreux prototypes développés en recherche implantent une ou plusieurs fonctionnalités requises pour la formation (simulation, explication, etc.) mais jamais l'ensemble de ces fonctionnalités. Au lieu de chercher à développer un nouvel outil qui proposerait cet ensemble, nous proposons de faire coopérer divers prototypes offrant des fonctionnalités complémentaires. L'objectif du travail est donc de définir des critères, une architecture et des outils permettant cette coopération. Nous focalisons notre proposition sur la coopération de prototypes existants et nous restreignons l'application à l'enseignement de la géométrie plane.

Nous proposons un atelier logiciel qui permet à un enseignant ou un chercheur d'utiliser des fonctionnalités implantées dans des prototypes différents, à travers une interface unificatrice, à peu près comme s'ils étaient disponibles dans le même logiciel. Pour le définir, nous avons modélisé une activité d'apprentissage du point de vue de l'exécution de logiciels et de l'échanges de données. Notre modèle comprend des scénarios découpés en étapes munies de transitions, des fonctionnalités offertes par un prototype, et la notion d'observable construite à partir de traces d'interaction et d'événements sémantiques. Nous proposons la notion de macro-définition avec les grammaires et interprètes associés pour adapter aussi bien des données du domaine que des observables. Notre atelier est implanté dans une maquette en Java et toutes les propositions sont faites avec un objectif de généralité qui confère aux propositions un caractère générique.

Mots-clés: coopération, EIAO, atelier, architecture, logiciel éducatif, enseignement, géométrie.

Abstract

This research domain is about Intelligent Educational Software (IES). The many research prototypes developed implement one or more of the necessary formation functionalities but never the whole of these functionalities. Instead of developing a new tool which would propose all functionalities, we make complementary prototypes co-operate in an integrated environment. The work objective is thus to define criteria, an architecture and tools allowing this co-operation. We carry out this co-operation within a software workshop which allows a teacher or a researcher to use functionalities implemented in different prototypes, through a unifying interface, about as if they were available in the same software. The diversity of the problems to be solved exceeds outlines of a thesis. We focus our proposal on the co-operation of existing prototypes and we restrict the apply-domain to geometry teaching and learning.

The workshop allows a teacher or a researcher to use functionalities implemented in different prototypes, through a unifying interface, about as if they were available in the same piece of software. To define it, we modeled an activity from different points of view. Our model includes scenarios cut out in stages provided with transitions, functionalities offered by a prototype, and the concept of observable built up starting from interaction traces and semantic events. We propose the macro-definition concept, associated with grammars and interpreters to format domain knowledge as well as observables. Our workshop is implemented in a model in Java. All the proposals are made with an objective of being generic.

Keywords: co-operation, EIAO, workshop, architecture, educational software, geometry.