CONCEPTION ET IMPLÉMENTATION D'UNE COUCHE DE COMMUNICATION ENTRE LES AGENTS INTELLIGENTS ET LES LABORATOIRES VIRTUELS

par

Kaufmann MEUDJA BOUHOM

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

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, avril 2003

Le 22 avril 2003,
Date

le jury a accepté le mémoire de M. Kaufmann Meudja Bouhom dans sa version finale.

Composition du jury

Membre :	M. Roger Nkambou (Direction) Département de mathématiques et d'informatique
Membre :	M. Froduald Kabanza (Codirection) Département de mathématiques et d'informatique
Membre :	Mme Reine Gagnon Département de mathématiques et d'informatique
Membre et président-rapporteur :	M. Sylvain Giroux Département de mathématiques et d'informatique

Sommaire

Dans ce mémoire, nous présentons dans un premier temps les laboratoires virtuels et leur rôle dans un système tutoriel intelligent. Deuxièmement, nous présentons les agents intelligents et leur rôle, suivi troisièmement, du cadre dans lequel ce travail de recherche a été effectué. Pour finir, nous présentons une architecture de communication entre les laboratoires virtuels et les agents. Les instances de cette architecture seront exploitées dans le cadre d'un système tutoriel intelligent mais aussi dans des applications de partage de l'information. Tout au long de ce travail, nous avons développé trois laboratoires virtuels (un laboratoire en digestion de molécules d'acide désoxyribonucléique (ADN) en génie génétique, un microscope ordinaire virtuel et un laboratoire virtuel en mathématiques). De plus, une API (Application Programming Interface) a été conçue et implémentée conformément à cette architecture de communication. L'expérimentation de cette API a été faite dans le cadre du projet Cyberscience et les résultats ont été probants. L'implémentation de cette API et des laboratoires virtuels a été réalisée avec le langage Java et XML (eXtensible Markup Language) avec l'aide de l'API JAXB (Java API for XML Binding).

Remerciements

Je tiens ici à exprimer toute ma gratitude à tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce travail.

C'est avec plaisir que je remercie mon directeur de recherche, le professeur Roger Nkambou, pour la direction de ce travail en général et notamment pour son apport dans la conception de l'architecture globale de cette API (ce « framework »), pour le découpage des grands modules de l'implémentation ainsi que pour les corrections constructives apportées lors de la rédaction du présent mémoire.

C'est également avec un grand plaisir que je remercie mon codirecteur, le professeur Froduald Kabanza, pour sa contribution à la rédaction de ce mémoire, notamment pour les corrections constructives apportées lors de la rédaction du présent mémoire et pour ses judicieux conseils tout au long de ce travail.

J'adresse également mes remerciements à l'ensemble du corps enseignant du département de mathématiques et d'informatique de l'Université de Sherbrooke ainsi qu'à tous les membres de l'équipe du projet CyberSciences. Ceux-ci s'adressent particulièrement à Reine Gagnon et à Claude Déry pour leurs contributions non seulement en tant que membres de l'équipe CyberSciences mais aussi en tant qu'experts et pour leurs apports dans la conception des laboratoires de mathématiques et de biologie.

Mes remerciements vont tout particulièrement à tous les membres de ma famille, mes amis et ceux qui, de près ou de loin, m'ont soutenu par leurs conseils, leurs prières et leurs encouragements.

Table des matières

Liste des acronymes

Dans cette liste d'abréviations, nous donnons les définitions des termes utilisés dans ce mémoire.

3D : Three Dimensions.

ACL : Agent Communication Language.

ADN : Acide désoxyribonucléique.

AI : Agents Intelligents.

API : Application Programming Interface.

ATL : Active Template Libraries.

BD : Bases de Données.

CBR : Case Based Reasonning.

COM : Component Object Model.

CORBA : Common Object Request Broker Architecture.

DCE : Distributed Computing Environment.

DCOM : Distributed Component Object Model.

DM : Data Mining.

DMI : Département de Mathématiques et d'Informatique.

DNS : Domain Name System.

DPL : Distributed Program Link.

DTD : Document Type Definition.

GDAC : Gestion, Diffusion et Acquisition de Connaissances.

IA : Intelligence Artificielle.

IBM : International Business Machines.

IDL : Interface Definition Language.

IP : Internet Protocol.

JAXB : Java API for XML Binding.

JDK : Java Development Kit.

LASIM : Laboratoire des Systèmes Intelligents et du Multimédia.

LVEST : Laboratoires Virtuels pour L'éducation en Sciences et Technologie.

JVM : Java Virtual Machine.

OLE : Object Linking and Embedding.

OMA : Object Management Architecture.

OMG : Object Management Group.

ORB : Object Request Broker.

OS/2 : Operating System / 2.

OSF : Open Software Foundation.

P : Planificateur.

RBR : Rule Based Reasoning.

RMI : Remote Method Invocation.

RPC : Remote Procedure Call.

SGML : Standard Generalized Markup Language.

SMA : Système Multi-Agents.

SQL : Structured Query Language.

STI : Système Tutoriel Intelligent.

TCP : Transmission Control Protocol.

UDP : User Datagram Protocol.

WWW : World Wide Web.

XML : eXtensible Markup Language.

Liste des tableaux

Liste des figures

Introduction

Depuis quelques années, l'enseignement en ligne occupe une part importante dans l'éducation et la formation à distance. De plus, on observe un intérêt croissant d'intégration des technologies dans ce concept. Cette croissance est due au développement de plus en plus rapide dans les domaines de la télématique et de l'informatique [9], [25], [37], [39]. Le nombre de compagnies, d'entreprises et d'universités offrant des programmes de formation sur réseaux internes ou externes ne fait qu'augmenter. Plusieurs travaux ont été faits dans le domaine des cours en ligne et des systèmes concrets en sont découlés (WebCT [45], Blackboard [2], TopClass [41], Bravo ! [42] et bien d'autres). Parallèlement, des laboratoires virtuels ont été développés pour servir de micro monde dans les environnements de formation en ligne, plus particulièrement en science et en génie (LVEST [40], Virtual Enginneering/Science Laboratory [23], Virtual Lab in Statistics [44], Virtual Laser Lab [8], Virolab [6], Pendelium [35] et le laboratoire de physique virtuel [7] pour ne citer que ceux-là).

Le travail présenté dans ce mémoire s'intéresse uniquement aux laboratoires virtuels et aux agents d'assistance et d'encadrement. Dans le but de mieux comprendre ce travail, nous commencerons par cerner le sens des mots "laboratoire" et "virtuel" tout en nous limitant aux définitions qui sont directement en rapport avec l'utilisation de ces mots dans le contexte qui nous concerne. D'après Hachette [17], on entend par :

Laboratoire : "1) Un local spécialement aménagé et équipé pour mener à bien des travaux (notamment de recherche) scientifiques ou techniques. (Exemple : laboratoire de physique,

laboratoire d'analyses bactériologiques, laboratoire d'un photographe, laboratoire pharmaceutique.). 2) Laboratoire de langues : local spécialement aménagé pour enseigner les langues étrangères à l'aide de magnétophones".

Virtuel : "1. Qui existe en puissance seulement ; potentiel. Ant. actuel. 2. Qui n'existe que sous une forme abstraite. – Bureau virtuel : au sein d'une entreprise, local de travail qui, partagé par plusieurs personnes pratiquant par ailleurs le télétravail, n'appartient en fait à aucune d'entre elles. Société de service accessible uniquement sur réseau télématique ou sur Internet. Banque virtuelle. Casino virtuel. 3. INFORM Réalité virtuelle, obtenue par l'enregistrement et le traitement de données en trois dimensions qui permettent de donner l'illusion d'une réalité".

À la lumière de nombreux dictionnaires, y compris celui référencé ici, nous avons constaté que le sens du terme virtuel n'a guère changé au cours des siècles, bien que son usage ait augmenté de façon exponentielle durant cette dernière décennie. Par contre le mot "laboratoire", très spécifique au départ, a progressivement servi à couvrir des concepts parfois assez éloignés de l'origine. Cependant, dans le cadre des laboratoires que l'on trouve sur un campus universitaire, on doit certainement faire des distinctions entre des unités telles qu'un laboratoire de chimie et un laboratoire de langue. Si le terme de laboratoire semblait approprié pour décrire les installations techniques complexes qui furent mises en place pour l'apprentissage interactif des langues modernes dans les années 60, ce n'est peut-être plus le cas aujourd'hui où ce même apprentissage se fait en utilisant des ordinateurs. Ceci dit, appellera-t-on laboratoire toute salle équipée d'ordinateurs, même si l'on n'y fait, par exemple, que de la formation en traitement de texte ? La question pourrait mener à des débats sans fin et, heureusement, il ne nous incombe pas d'y apporter une réponse finale. Dans le cadre de notre étude, nous nous limiterons aux seuls laboratoires liés aux sciences et aux technologies. Cela permettra d'éviter en grande partie les méandres des discussions esquissées ci-dessus. Il est toutefois à noter que la tendance marquée de ces dernières années est de se diriger de plus en plus vers des

travaux de laboratoires scientifiques réalisés au travers de simulations ou de traitement de données sur ordinateurs [1], [11], [20]. C'est là un des signes révélateurs de la multiplicité des dimensions de l'apprentissage qui se fait par ce biais. C'est également un des premiers signes des difficultés croissantes qu'il y a à faire une distinction claire entre ce qui est virtuel et ce qui ne l'est pas. Dans un monde où tout devient digitalisé et est accessible à distance, il est parfois bien délicat de faire la différence entre réalité et virtualité. Cet aspect devrait jouer en faveur des laboratoires virtuels. Nous pouvons donc appeler laboratoire virtuel tout laboratoire ayant au moins un élément virtuel.

Problématique

Malgré l'augmentation de l'intérêt à intégrer les nouvelles technologies dans le concept de formation à distance, il semble encore exister, à ce jour, une lacune importante dans ces univers virtuels de formation en ce qui concerne les laboratoires. Cette lacune consiste en une quasi-absence d'agents virtuels d'assistance et d'encadrement. Dans notre contexte, soit celui de l'enseignement à distance et particulièrement celui de la manipulation directe dans les laboratoires, un agent virtuel est un assistant virtuel qui suit les manipulations de l'utilisateur dans le but de l'assister et de l'encadrer en lui prodiguant des conseils, en lui fournissant des explications, bref en jouant approximativement le même rôle qu'un tuteur humain sans toutefois être en mesure de remplacer ce dernier. Il est cependant important de noter que certains environnements virtuels remédient à cette lacune en intégrant directement les agents spécifiques au domaine dans leurs laboratoires. Ainsi, de nos jours, on développe des agents non génériques qui sont directement relié au laboratoire virtuel (micro monde), donc spécifique au domaine d'application [23]. Cette intégration est à l'origine de plusieurs autres lacunes dans le développement et l'implémentation des laboratoires virtuels. On peut noter entre autres : la lourdeur du système, la quasi abscence quant à la réutilisation de ces agents d'où la nécessité de dissocier le contrôle de l'apprentissage (par les agents d'assistance et d'encadrement) des laboratoires virtuels

qui ne sont que des ressources pour l'apprentissage.
Le problème qui se pose alors est de savoir comment dissocier les agents (intelligents) des laboratoires virtuels et permettre une communication (un échange) des données en tant qu'entités d'un même système ou en tant qu'applications se partageant des données ou des informations (particulièrement dans les STI). La question qui se pose à présent est la suivante : comment doivent cohabiter les agents d'assistance et d'encadrement avec les laboratoires virtuels ? Doivent-ils cohabiter en Système Multi-Agents (SMA), système de composants ou dans un système de couches ?

Objectif

Le but de cette étude est de fournir une API (Application Programming Interface) qui permettra aux développeurs d'implémenter des laboratoires virtuels dans lesquels les activités d'apprentissage peuvent être contrôlées par des agents virtuels d'encadrement (ou agents tutoriels). Pour assurer le contrôle des activités dans les laboratoires, ces agents auront besoin d'un module contenant la structure des connaissances du domaine concerné par le laboratoire. Il sera ainsi possible de modifier plus facilement les connaissances des agents sans avoir à modifier ces derniers mais plutôt en apportant une modification au module expert du domaine.
Dans cette recherche, nous nous concentrons particulièrement sur les laboratoires virtuels et le processus de communication entre les laboratoires virtuels et les agents intelligents. En d'autres termes, l'objectif ici est de concevoir et d'implémenter une couche de communication qui permettra à deux entités d'un même système ou à deux applications de pouvoir s'échanger des données ou des informations. Ceci permettra la communication entre deux entités d'un système ou deux applications développées séparément avec des ontologies différentes. Cette couche devra fournir une ontologie de communication commune ou compréhensible par les deux entités ou les deux applications. Le terme « ontologie » est emprunté à la philosophie où il se réfère au sujet d'existence. Il est aussi souvent

confondu avec l'épistémologie, qui traite de la connaissance et du savoir. Cependant, dans notre contexte, une ontologie est une spécification explicite d'une conceptualisation. En d'autres termes, une ontologie est une spécification employée pour faire des obligations ontologiques qui sont en fait des accords pour l'utilisation d'un vocabulaire, l'acquisition et l'échange des connaissances ou des informations [16], [15].

Méthodologie

Compte tenu du fait que les expériences sur les laboratoires virtuels sont encore au niveau de prototypes, nous préconisons une approche inductive. Cette approche a mené dans un premier temps, à définir différents types de laboratoire (tel que les laboratoires de manipulation à distance et les laboratoires de simulation) en fonction des caractéristiques spécifiques, des champs de connaissances et des domaines couverts. Ensuite, nous avons développé une ontologie pour ces laboratoires. Finalement, nous avons implémenté trois laboratoires virtuels (un en mathématiques et deux en biologie) dans le cadre d'un projet de développement d'un système tutoriel appelé Cyberscience à l'Université de Sherbrooke. Enfin, nous avons développé une couche de communication qui permettra la communication et donc l'échange d'informations entre les laboratoires et les agents tuteurs (les agents tuteurs étant en développement).

Organisation du mémoire

Ce mémoire se subdivise en quatre chapitres. Le chapitre 1 constitue une revue des travaux faits dans les domaines des STI, des agents intelligents (AI) et des laboratoires virtuels. Il est à noter que ce mémoire présente les STI et les AI de façon détaillé. Le chapitre 2 présente le contexte dans lequel cette recherche a été entreprise (le projet Cyberscience) et nous y mettrons en évidence les laboratoires virtuels que nous avons développés dans ce cadre. Nous présentons au chapitre 3 la conception de la couche de communication qui est l'un des buts de la recherche ici présentée. Le chapitre 4 présente

l'implémentation de cette solution et nous concluons avec une présentation des contributions ainsi que des limites de notre système.

Chapitre 1

État de l'art

Les recherches montrent qu'il existe un très grand nombre et une très grande diversité de travaux sur les laboratoires virtuels, les STI et les agents intelligents. Ce chapitre présente donc l'état de l'art.

1.1 Systèmes tutoriels intelligents

Les systèmes tutoriels intelligents ont pour objet de réaliser un enseignement individualisé à l'aide d'un ordinateur. La construction de tels systèmes nécessite une bonne compréhension aussi bien du processus d'enseignement et d'apprentissage, que du comportement des acteurs impliqués dans ce processus (enseignants et apprenants.). Il n'est donc pas surprenant que la conception des STI soit un problème à la frontière de l'éducation, de l'informatique et de la psychologie cognitive. En fait, les STI ont été introduits au début des années 70 dans le but d'expérimenter l'intégration de théories reliées aux domaines de l'intelligence artificielle (IA), des sciences de l'éducation et de la psychologie cognitive. Dans ses travaux sur les STI, Nkambou distingue six types de STI selon leur utilisation : les STI socratiques, les STI procéduraux, les démonstrateurs, les environnements interactifs d'apprentissage, les systèmes critiques et les systèmes sociaux [30].

Les STI socratiques permettent à l'aide d'un dialogue avec l'apprenant, l'enseignement de faits et d'habiletés nécessaires pour inférer d'autres connaissances. Ainsi l'apprenant peut découvrir ses erreurs et ses contradictions grâce à un dialogue avec le système.
Les STI procéduraux permettent l'enseignement d'habiletés et de procédures nécessaires pour accomplir une tâche à travers des exemples, des exercices et des problèmes. Sa forme la plus connue est le « coach », c'est-à-dire un système qui observe la progression de l'apprenant pendant que ce dernier résout un problème et peut l'interrompre pour lui présenter de nouvelles informations ou lui faire des suggestions.
Les démonstrateurs sont des systèmes qui imitent un phénomène dans le but de l'enseigner à l'apprenant. Ils n'enseignent pas de la même façon que les STI socratiques et procéduraux, mais se contentent de montrer des choses à l'étudiant. La simulation donne à l'étudiant la possibilité de pratiquer dans un environnement virtuel mais semblable au réel, ce qui permet de considérer la démonstration comme un outil d'apprentissage et non comme un STI dans le sens réel du terme car l'intelligence ici est encapsulée dans la simulation.
Les environnements interactifs d'apprentissage quant à eux permettent à l'étudiant d'apprendre tout en explorant l'objet d'apprentissage de manière guidée (i.e. le système peut intervenir dans la formation lorsque cela s'avère nécessaire) ou libre (i.e. l'étudiant utilise les outils mis à sa disposition pour construire des modèles mentaux, sans intervention du système).
En ce qui concerne les systèmes critiques, ils ont pour but de critiquer les apprenants dans leur processus de résolution de problèmes. La critique ici consiste à aider, empêcher et réduire les erreurs de jugement et les préjugés chez les apprenants, ceci dans le but d'amener l'utilisateur vers une solution correcte.
Enfin, les systèmes sociaux sont des STI qui font intervenir, en plus des agents classiques d'un STI, d'autres agents tels qu'un ou plusieurs autres étudiants, un ou plusieurs autres

enseignants. Ces différents agents communiquent entre eux lors d'une session d'apprentissage ; par exemple, chaque agent peut donner son point de vue pendant la résolution d'un problème.

Ces différents types de STI ont chacun des caractérisques différentes mais utiles pour un enseignement ou un apprentissage de qualité. Ce sont donc des systèmes complémentaires. Leur intégration s'avère donc très utile pour obtenir des STI plus puissants. C'est dans le but d'atteindre cet objectif que le projet Cyberscience de l'Université de Sherbrooke a vu le jour.

1.2 Agents intelligents

Depuis quelques années, les agents sont à la mode, sans pourtant qu'il y ait unanimité sur leur définition. Chaque chercheur en intelligence artificielle propose sa propre définition d'un « agent intelligent » et critiquent la pertinence de cette appellation pour les applications proposées actuellement par les éditeurs de logiciels [12], [13]. Ils considèrent que ce sont certes là des outils utiles mais qui n'ont souvent pas grand chose à voir avec les systèmes qu'ils développent en laboratoire. On note certainement une très grande confusion entourant le terme. Cette confusion découle aussi du fait que l'appellation « agents » est utilisée à tort et à travers sans aucune précision exacte quant au sens. Il faut donc distinguer les agents « intelligents » des autres. Les agents « intelligents » sont des applications développées par des chercheurs en intelligence artificielle. Ils sont définis de manière assez précise et ils doivent posséder un certain nombre de caractéristiques considérées comme consubstantielles à leur « intelligence ». Par ailleurs le terme « agent » est utilisé pour caractériser toute une gamme de logiciels utilisant des technologies non issues de l'intelligence artificielle mais se référant à l'appellation d'agents au sens de la définition usuelle d'un agent, c'est-à-dire une entité autorisée à agir à la place de quelqu'un et agissant en son nom. Ces « agents » ont pour objectif d'automatiser des tâches répétitives

et pénibles. L'absence d'unanimité dans la définition du terme « agent intelligent » fait qu'on ne le définit que par ses caractéristiques. Alors, nous pouvons définir un agent intelligent comme étant un composant logiciel capable d'être autonome (possibilité de prendre des initiatives et agir sans aucune intervention de l'utilisateur), de communiquer (capacité d'échanger des informations plus ou moins complexes avec d'autres agents, des serveurs ou des utilisateurs) et d'apprendre (capacité de réagir avec un environnement, de s'adapter aux circonstances, de prendre une décision ou d'enrichir eux-mêmes leur propre comportement sur la base d'observations qu'ils effectuent). Un agent intelligent peut contenir un ou plusieurs des éléments suivants : une base de connaissances, un moteur d'inférence lui permettant de tenir des raisonnements plus ou moins complexes, un système d'acquisition de connaissances et un mécanisme d'apprentissage [38]. La communication entre les différents agents se fait généralement par le biais des ACL « Agents Communication Language ».

Dans leurs travaux sur les agents tutoriels, Nkambou et Kabanza distinguent six agents de base, à savoir : l'agent de raisonnement à base de règles (un système expert), l'agent de raisonnement à base de cas, l'agent de « data mining » et l'agent planificateur, l'extracteur de connaissances et l'agent de prise de décision. Les deux derniers sont en fait des combinaisons des quatre premiers [31], [32], [33].

L'agent de raisonnement à base de règles (RBR) est en fait un système expert placé à l'intérieur d'une coquille d'agents. Un système expert est une application capable d'effectuer, dans un domaine, des raisonnements logiques comparables à ceux que feraient des experts humains de ce domaine. Il s'appuie sur des bases de données, de faits et de connaissances, ainsi que sur un moteur d'inférence pour faire des déductions logiques (chaînage avant et arrière). C'est avant tout un système d'aide à la décision car, il utilise des faits et des règles pour identifier les connaissances à utiliser pour la résolution d'un problème donné. Les règles de ce système peuvent être stockées sur le disque (dans une base de données), en mémoire (dans le système) ou en utilisant une combinaison de ces

possibilités.

En ce qui concerne l'agent de raisonnement à base de cas (CBR), l'expérience (basée sur les cas déjà résolus) guide sa compréhension des nouvelles situations. L'interprétation et la mémorisation des cas se font grâce à une indexation basée essentiellement sur des traits de surface. Si une situation nouvelle ne s'apparie pas avec un cas déjà référencé, la base de cas est mise-à-jour par l'ajout de ce nouveau cas dans la base (apprentissage par l'échec). Le raisonnement par cas est bien adapté aux domaines où il n'existe ni théorie ni modèle formalisé, et où le rôle de l'expérience est prédominant. Un des avantages souvent cité en faveur du raisonnement par cas est qu'il opère à un haut niveau pragmatique d'expertise sans s'occuper des modèles théoriques du domaine.

Le rôle de l'agent « data mining » (DM) est avant tout, comme son nom l'indique, la récupération d'information. Il a la capacité de chercher et de récupérer les informations en provenance d'une grande variété de sources (bases de données - (BD), fichiers, internet). Cet agent est souvent utilisé pour arriver à intégrer des éléments externes à un système mais peut aussi permettre de réutiliser et d'exploiter plus facilement le contenu déjà présent dans le système.

L'agent planificateur (P) génère un plan permettant d'atteindre un but donné à partir d'une situation initiale et d'un ensemble de transitions possibles. En général, un planificateur fonctionne à partir d'une collection de transitions possibles et de connaissances ou de stratégies liées au domaine. Il reçoit un but et la situation actuelle, et génère une séquence de transitions ou d'étapes qui sont en fait des actions permettant d'atteindre le but spécifié. La situation actuelle peut être construite en interrogeant la base de connaissances des autres agents.

Pour ce qui est de l'agent de prise de décision, comme nous l'avons indiqué, il sera construit à partir des agents décrits précédemment. Celui-ci contiendra donc un agent de raisonnement à base de règles (RBR), un agent de raisonnement à base de cas (CBR) et un planificateur (P). Le rôle de cet agent est de suivre l'évolution de son environnement

et d'agir (ou intervenir) si nécessaire dans le but d'atteindre son objectif. La figure 1 présente un modèle de l'agent de prise de décision.

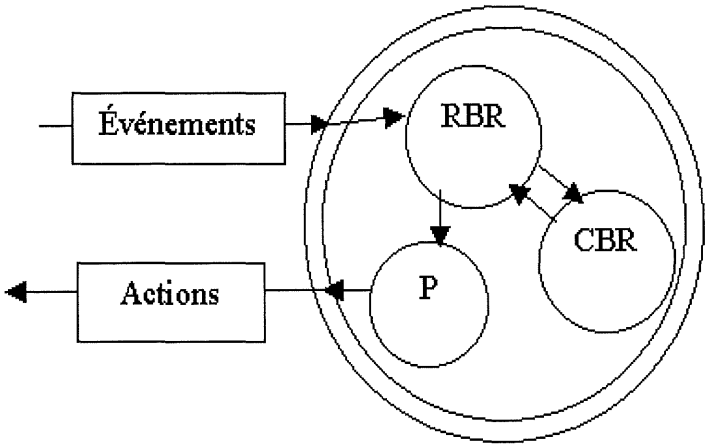

FIG. 1 – *Modèle d'agent de prise de décision*

Le dernier des agents de base est l'agent d'extraction de données. Cet agent est lui aussi une composition d'autres agents. Les agents utilisés ici sont l'agent de raisonnement à base de règles, l'agent de raisonnement à base de cas et l'agent de data mining. La figure 2 montre un modèle de cet agent.

1.3 Laboratoires virtuels

La plupart des travaux sur les laboratoires virtuels ne prennent pas en compte l'intégration de support intelligent à l'apprentissage ou du système conseiller (par exemple : laboratoire virtuel sur l'étude du pendule [10]). Plusieurs institutions, entreprises et individus ont travaillé et travaillent encore à la mise en place de laboratoires virtuels. Ces travaux sont cependant à l'état embryonnaire ou de prototypage (c'est-à-dire que les travaux effectués sont encore à un stade expérimental), ce qui signifie que nous en sommes encore au début de la courbe d'apprentissage sur les objectifs, les stratégies et les technologies.

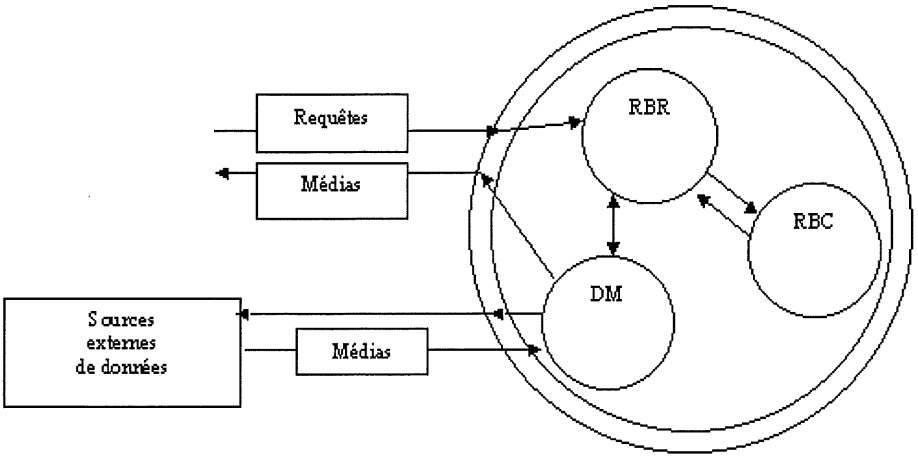

FIG. 2 – *Modèle d'agent d'extraction de connaissances*

Cette section est définie en fonction des articles et des communications sur le sujet et d'un grand nombre de visite des prototypes de laboratoires virtuels disponibles sur Internet (par exemple : virolab [6], laboratoire de physique virtuel [7], virtual laser lab [8], pendulum [10] et [35]).

Compte tenu du fait qu'il s'agit ici d'un domaine naissant, il n'existe pas assez d'articles et de communications sur le sujet. Cependant, il existe plusieurs sites internet qui traitent de la question des laboratoires virtuels. Ceci nous pousse à croire qu'une approche plus expérimentale est adoptée dans ce domaine. Cette approche semble être confirmée par la multitude et la diversité des réalisations qui présentent une grande disparité tant dans les buts que dans les approches des concepteurs. Selon le groupe du projet LVEST[43] (Laboratoires virtuels pour l'éducation en science et technologie) du centre de recherche LICEF et de l'École Polytechnique de Montréal, on peut regrouper les sites traitant des laboratoires virtuels en cinq grandes catégories en fonction de leurs buts explicites ou implicites :

· sites avec expérimentation à distance par la télécommande d'instruments réels et la télémesure synchrone, avec ou sans rétroaction visuelle par canal vidéo ;

· sites basés sur la simulation, que ce soit en mode local, distant (programme de simulation tournant sur le serveur) ou encore au travers de fureteurs par le biais d'applets (Java) ;

· sites orientés vers les procédures de laboratoire, dans lesquels l'apprenant est généralement amené à faire une série d'opérations dans un ordre bien déterminé afin de mener à bien son expérience. Ces sites sont souvent supportés par des animations ;

· sites offrant du matériel de référence en complément au laboratoire, à utiliser avant ou après les séances de laboratoire (préparation des expériences, planification, familiarisation avec le montage et les instruments, etc., puis traitement des données, analyse, interprétation, préparation de rapport, etc.) ;

· sites visant la mise en commun de ressources à des fins de recherche et favorisant le travail collaboratif. Ces sites sont souvent construits autour d'équipements coûteux tels que microscopes électroniques, télescopes, logiciels complexes, etc.

De ces catégories, les quatre premières ont un but didactique alors que la dernière est généralement orientée vers la recherche. À la visite de ces différents sites, on observe les différentes possibilités qu'offrent les nouvelles technologies pour la conception de laboratoires virtuels. Cependant, il existe plusieurs lacunes telles que :

1. l'absence de certaines fonctionnalités essentielles au laboratoire (telle que la manipulation directe ou le partage d'équipement) ;
2. l'absence d'intégration des moyens de support à l'apprentissage par le biais de fonctionnalités de communication entre enseignants et apprenants comme le font les sites offrant des cours en ligne ;
3. la quasi-absence des agents de support à l'apprentissage (aspect intelligent). De plus, ceux qui intègrent ces agents le font directement dans les laboratoires virtuels.

1.3.1 Buts

Les laboratoires visent plusieurs buts dont : les habiletés techniques et psychomotrices, les habiletés cognitives, l'approfondissement de concepts, les lois ou phénomènes scientifiques ou technologiques et bien d'autres [21], [29]. Il faut cependant rester réaliste sur le fait que certains de ces buts sont extrêmement difficiles à atteindre dans un laboratoire virtuel. C'est par exemple le cas de certaines habiletés psychomotrices. Par contre, il est tout à fait possible d'atteindre les buts liés à la connaissance et à l'attitude selon, bien entendu, les domaines et les types de laboratoires.

1.3.2 Potentiels

Les laboratoires virtuels peuvent briser les barrières temporelle, financière et spatiale grâce à la puissance de calcul, la grande capacité de mémoire, la grande capacité des bases de données locales et distantes que leur procurent les systèmes informatiques sur lesquels ils sont fondés. Il ne faut pas oublier que les équipements de laboratoires réels sont généralement assez dispendueux donc très coûteux. Sur le plan temporel, ils permettent une simulation en un laps de temps plus court que dans un laboratoire réel tout en éliminant les tâches routinières. De plus, ils permettent la disponibilité d'équipements en tout temps et à distance donc la possibilité de travailler à partir de chez soi sans avoir besoin de se rendre dans un laboratoire réel où la disponibilité des équipements n'est pas garantie [3]. Sur le plan spatial (distance), ils permettent l'accès à des banques de données expérimentales distribuées ou distantes sans avoir à se déplacer donc, éliminent les barrières géographiques [14], [28].

1.3.3 Fonctionalités

Afin d'atteindre les objectifs des laboratoires conventionnels (habilités techniques, habilités cognitives, approfondissement de concept, etc.), on identifie six éléments clés à la

base d'un laboratoire virtuel (Figure 3 [43]) :

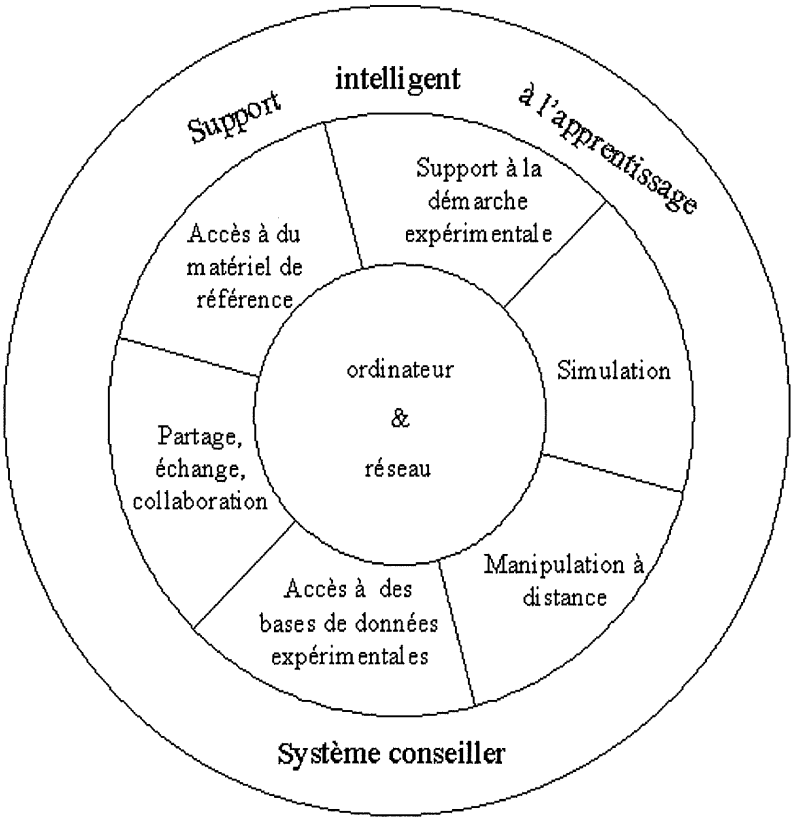

FIG. 3 – *Éléments de base d'un laboratoire virtuel*

· La simulation est une fonctionnalité permettant de reproduire dans les laboratoires virtuels les résultats des laboratoires réels.

· La manipulation à distance quant à elle permet l'utilisation et la manipulation des équipements réels à distance. Elle est considérée comme une fonctionnalité des laboratoires virtuels compte tenu du fait que la manipulation n'est pas directe et se fait à l'aide d'un ordinateur.

· L'accès à des bases de données permet d'avoir à sa disposition une grande quantité de

données, de mesures expérimentales. Celles-ci peuvent faire partie d'une base de données locale ou distante. Ceci permettra donc à un utilisateur de faire des études comparatives, paramétriques ou statistiques sans avoir à refaire toutes les mesures.

· L'accès au matériel de référence permet d'avoir tout le matériel dont on a besoin dans un laboratoire réel intégré au laboratoire virtuel.

· Le support à la démarche expérimentale représente l'accès aux matériaux nécessaires pour une bonne expérience tout en respectant la démarche à suivre.

· Le partage, l'échange et la collaboration sont des fonctionnalités permettant aux différents acteurs d'un laboratoire virtuel d'échanger les résultats de leurs expériences.

· Un système de support intelligent à l'apprentissage permet de pouvoir suivre la démarche de l'utilisateur et de lui apporter une aide au besoin. Il s'agit ici d'un tuteur virtuel.

Après avoir présenté ce qui s'est fait et ce qui se fait en ce moment dans les domaines des STI, des laboratoires virtuels et des agents, nous décrirons dans le prochain chapitre le contexte dans lequel cette recherche a été menée.

Chapitre 2

Contexte de développement

Le projet Cyberscience est un projet en développement au Département de mathématiques et d'informatique de l'Université de Sherbrooke. Ce projet a pour objectif de développer un système tutoriel intelligent intégrant des laboratoires virtuels pour supporter l'apprentissage en ligne. Ce chapitre présente le projet Cyberscience (son interface principale, les différents outils de communication qu'il contient et les laboratoires virtuels qui ont été développés dans ce cadre) et les différentes technologies de distribution d'applications disponibles sur le marché.

Cyberscience est un environnement d'apprentissage à distance. Il s'agit d'un STI générique pour l'enseignement des sciences pures. Les sciences pures se distinguent des autres disciplines par l'expérimentation en laboratoire avec un matériel technique souvent dispendieux et pas toujours disponible, et par l'utilisation d'un langage formel comme les mathématiques. L'interface de l'outil développé dans Cyberscience est un environnement d'apprentissage offrant plusieurs fonctionnalités et outils pour l'apprentissage et le support à l'apprentissage (assistance, communication et bien d'autres). La figure 4 présente l'interface de Cyberscience.

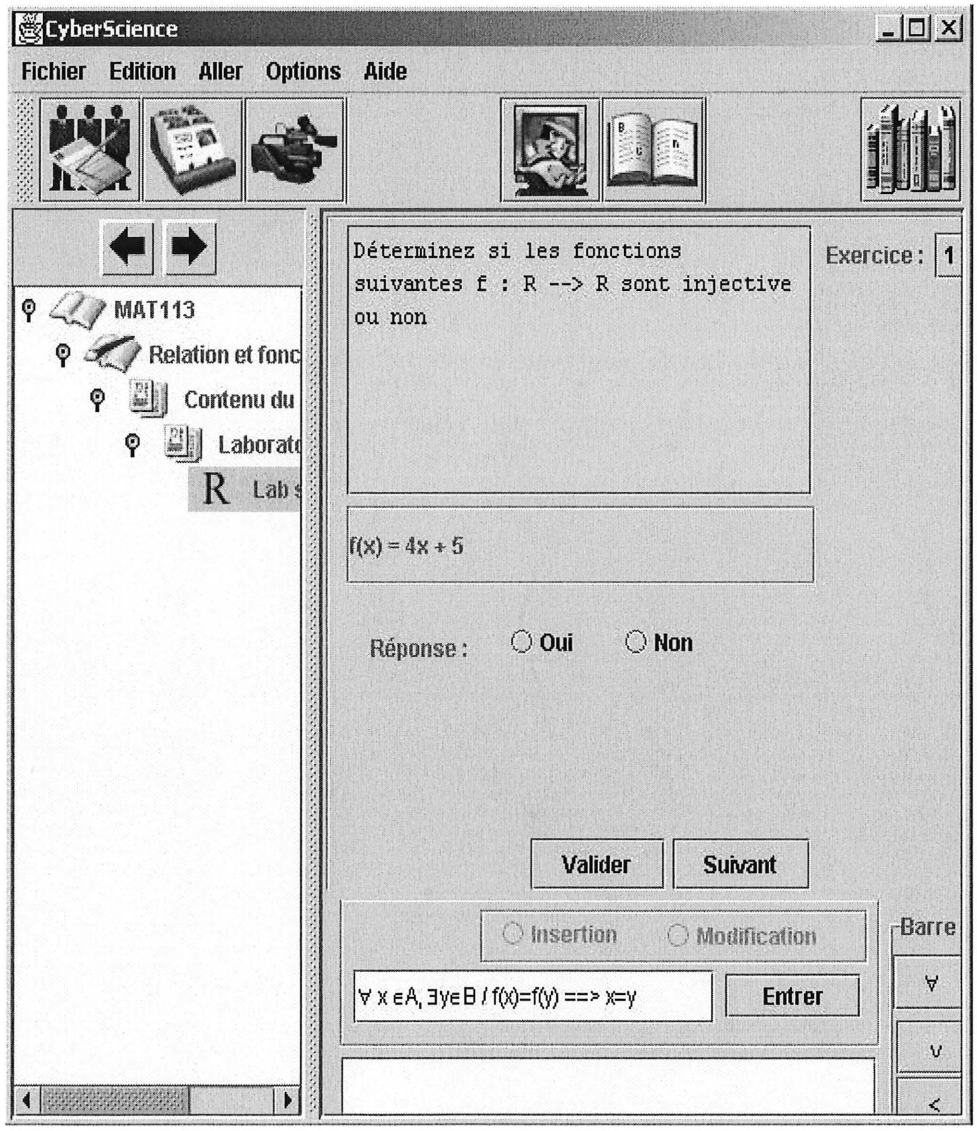

FIG. 4 – *Interface de Cyberscience présentant le laboratoire de MAT113.*

2.1 Outils de communication

Le forum est un outil de communication qui a été développé, implémenté et testé. Il a été développé à l'aide d'un langage de script (Cold fusion) et utilise une base de données Microsoft Access pour le stockage des messages, des informations relatives à son utilisation et des informations sur les utilisateurs. De plus, il est basé sur le modèle client/serveur. Cet outil distingue deux catégories d'utilisateurs (l'administrateur et l'usager ordinaire). L'administrateur du forum peut décider de donner certains droits aux utilisateurs tels que s'inscrire au forum, s'inscrire ou se retirer d'une catégorie, créer de nouveaux thèmes etc. Cet outil donne la possibilité aux usagers de rechercher des messages selon certains critères (auteur, catégorie du message, mot-clé, sujet). Les différentes catégories (par exemple : Agora, Grèce et Rome) ne s'affichent que si l'usager y a accès. Elles sont présentées selon ses habitudes de visite. Il ne peut y avoir plus de quatre catégories visibles à la connexion et si l'utilisateur a accès à plus de quatre catégories, les moins visitées seront disponibles dans le menu déroulant situé après la dernière catégorie. Ce forum donne la possibilité de consulter les mots-clés de groupe et personnel ainsi que les évaluations associées aux messages. Il offre plusieurs autres fonctionnalités qui apportent une valeur ajoutée quant à la gestion des utilisateurs et des messages échangés.

Il est présentement utilisé dans trois départements à l'Université de Sherbrooke (le Département de biologie, le Département de mathématiques et d'informatique et le Département d'histoire). De plus, il est aussi utilisé dans plusieurs autres universités (Université de Moncton, Université de Montréal, Université du Québec à Montréal et Telé-université) et par certains groupes de recherche en système tutoriels intelligents.

Depuis son implémentation et à la suite des commentaires des différents usagers, il a été amélioré afin de mieux répondre aux différents besoins. Parmi les nombreuses améliorations apportées, nous pouvons citer l'ajout des fonctions telles que : l'attachement de fichiers, la personnalisation des catégories, les droits d'accès aux catégories, la

gestion des catégories etc. Les messages non lus sont marqués d'un marqueur rouge en avant et une petite icône en avant d'un message indique qu'il y a un fichier attaché à ce dernier. Les figures 5, 6, 7, 8, 9 et 10 présentent les principaux interfaces du forum.

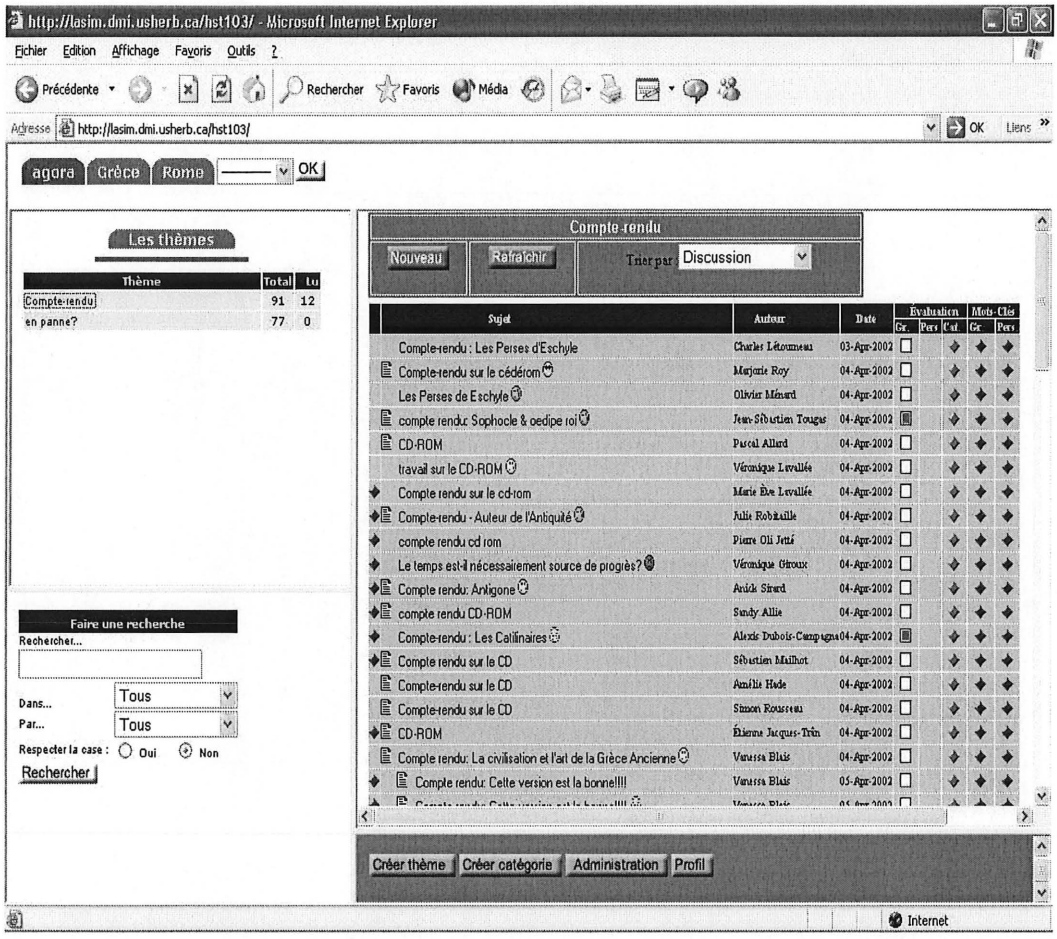

FIG. 5 – *Interface du forum de communication*

Cet outil est présentement en cours d'extension au laboratoire GDAC (Gestion, Diffusion et Acquisition de Connaissances) à l'Université du Québec à Montréal pour une version plus sophistiquée.

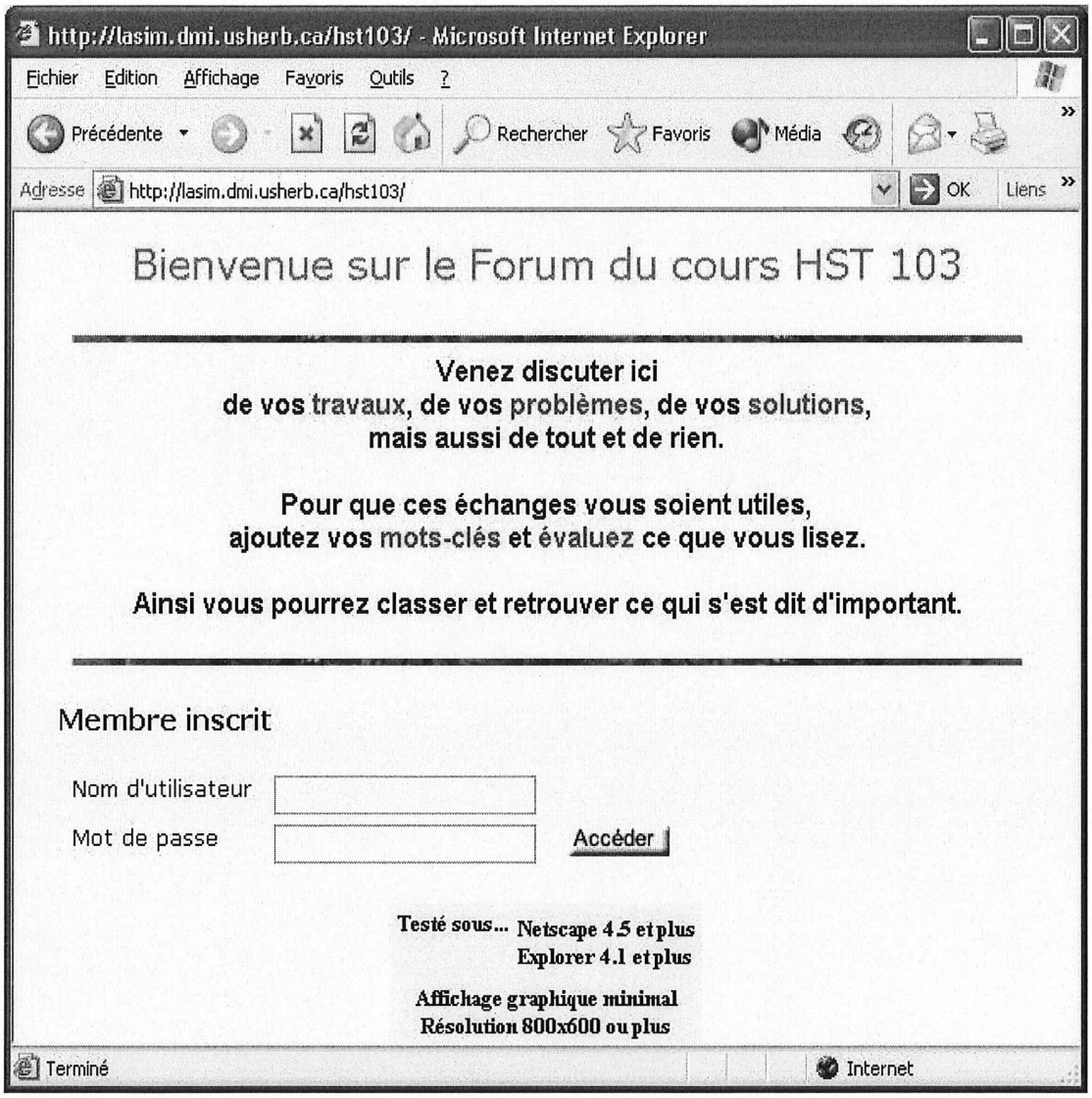

FIG. 6 – *Écran de connexion au forum*

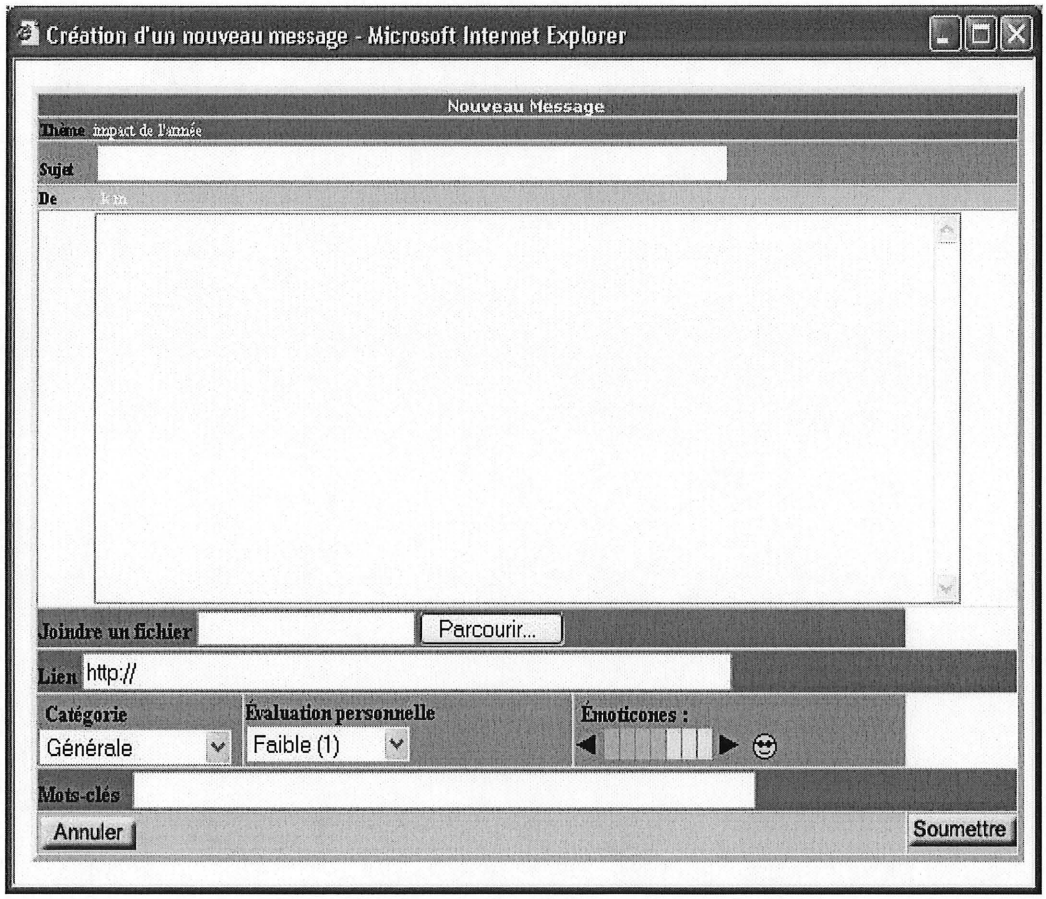

FIG. 7 – *Fenêtre de composition de messages*

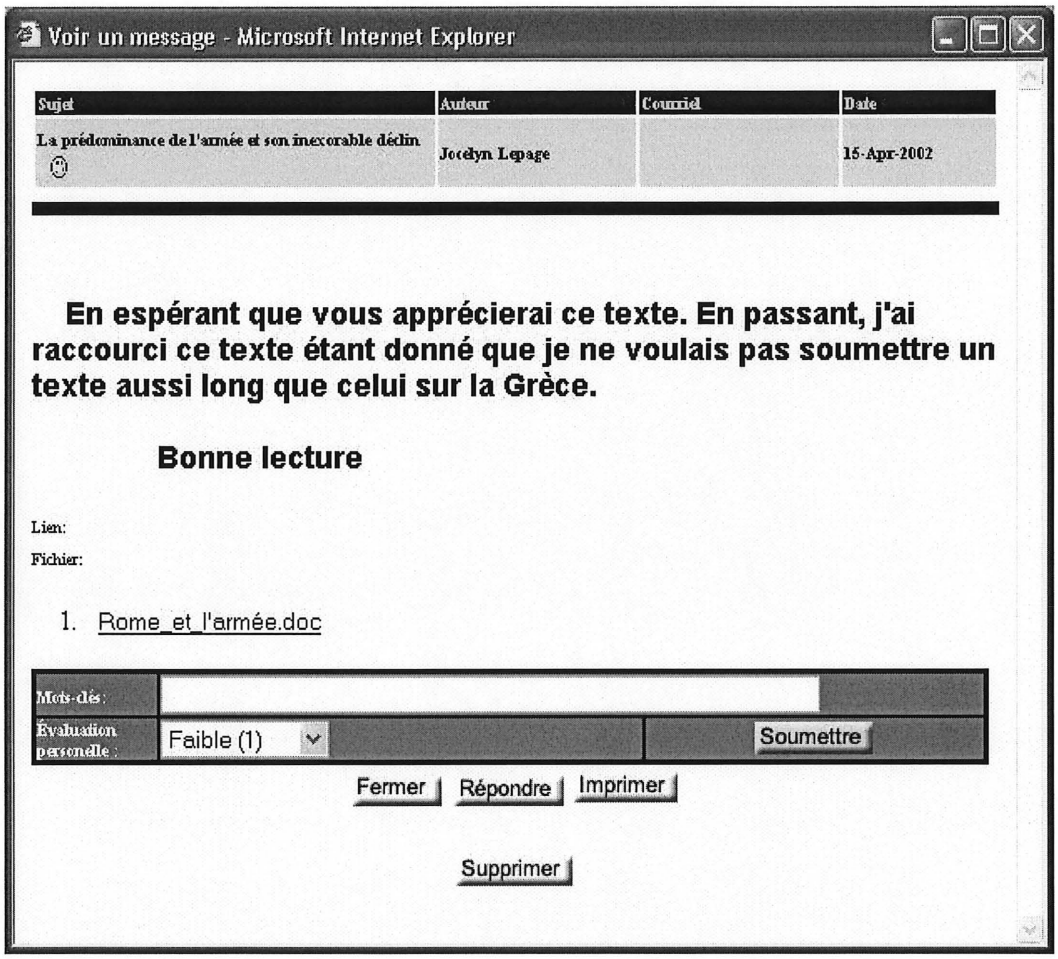

FIG. 8 – *Fenêtre de lecture de messages*

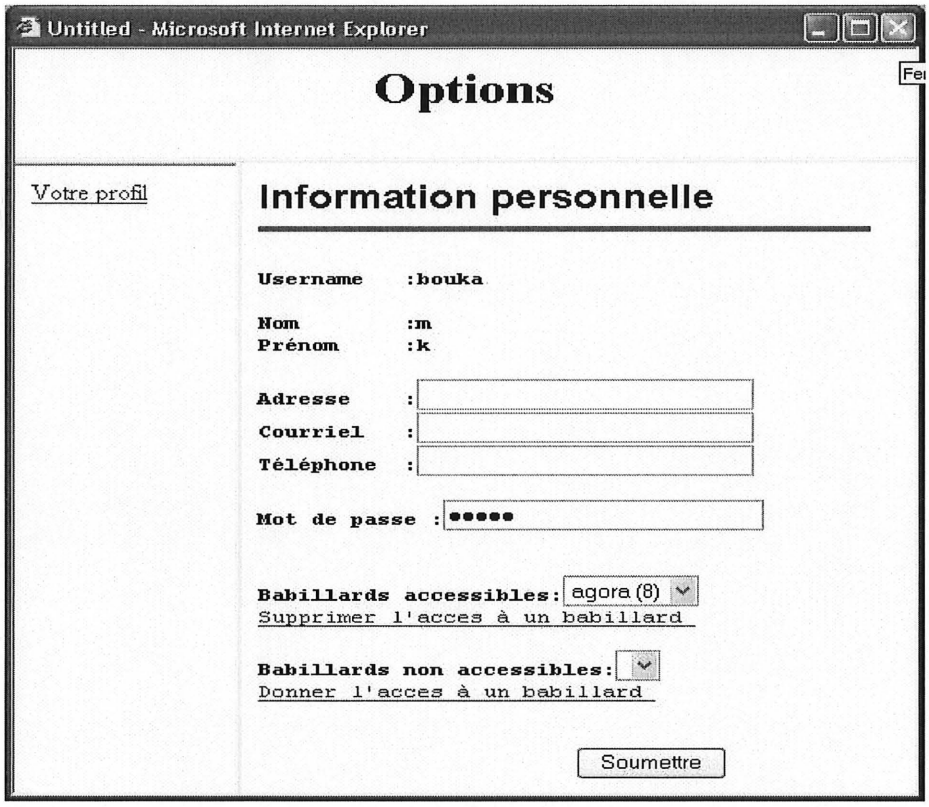

FIG. 9 – *Fenêtre de consultation et de modification du profil personnel*

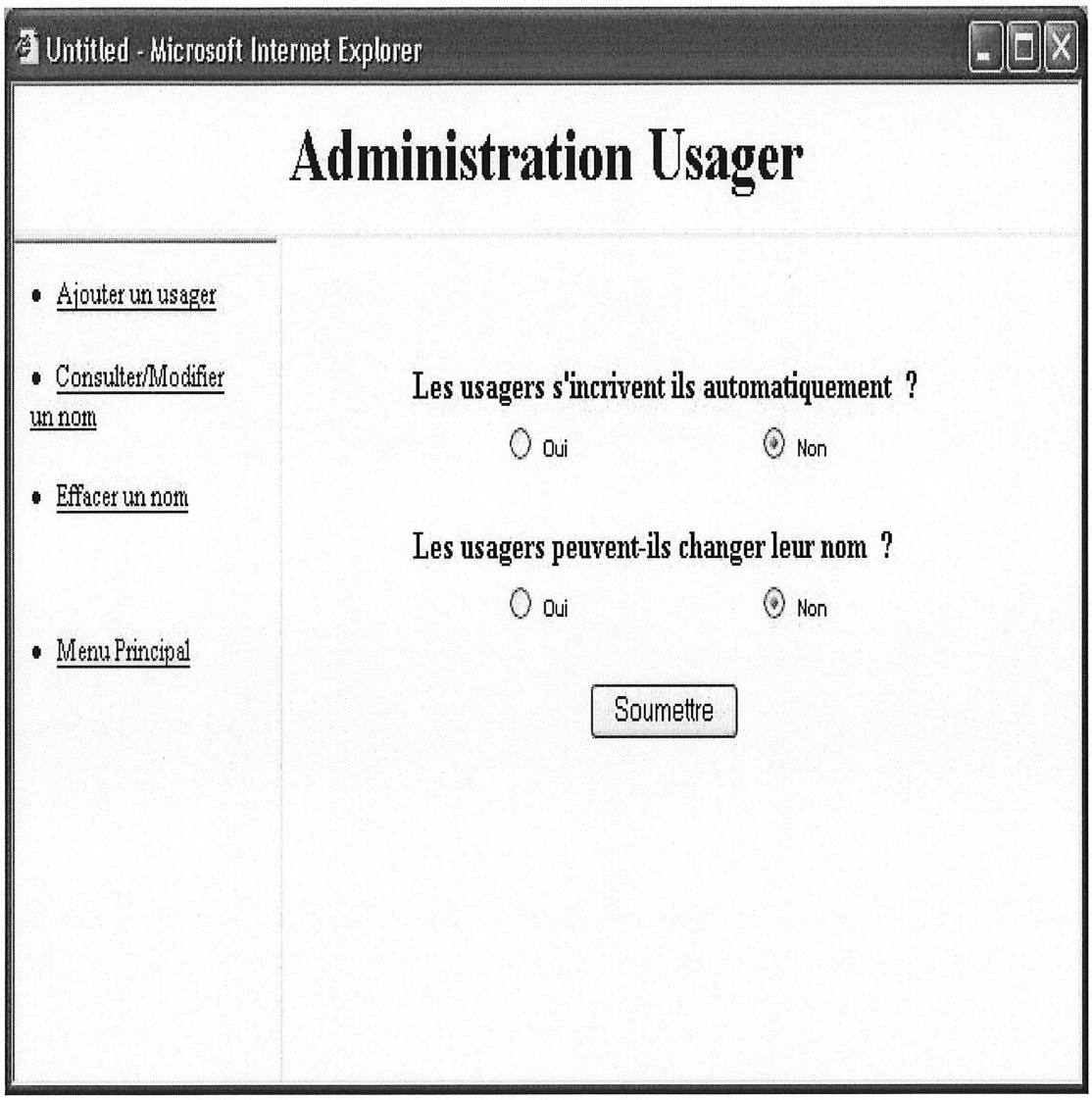

FIG. 10 – *Fenêtre principale d'administration des usagers*

2.2 Laboratoires virtuels

L'intégration des laboratoires virtuels dans les STIs permet aux STIs de suivre l'activité cognitive de l'étudiante ou de l'étudiant. Les manipulations des usagers seront donc accessibles aux agents tutoriels leur permettant ainsi d'avoir une meilleure compréhension des activités de l'étudiant afin de pouvoir l'aider intelligemment dans son apprentissage. Il est à noter que le développement de ces laboratoires peut être considéré comme un travail de cogniticien dans ce sens qu'il nécessite de nombreuses interactions avec les experts. Ceci nécessite donc des stratégies pour permettre aux experts de mettre l'emphase sur les connaissances du domaine concerné pour nous permettre :

1. de comprendre les tâches à exécuter dans le laboratoire cible,
2. d'extraire les connaissances sous-jacentes,
3. de planifier le développement proprement dit du laboratoire.

Il s'agit bien entendu d'une activité itérative qui nécessite la disponibilité des experts tout au long du cycle de développement et une approche scientifique [18], [19]. Nous présentons nos réalisations dans ce qui suit, à savoir le développement d'un laboratoire virtuel en génie génétique et le développement d'un laboratoire virtuel en mathématiques. Nous introduisons aussi notre contribution au développement d'un laboratoire virtuel en biologie (le microscope optique).

2.2.1 Laboratoire virtuel de digestion de molécules d'ADN

Le laboratoire virtuel d'acide désoxyribonucléique est un laboratoire développé pour la biologie moléculaire. Ce laboratoire peut être considéré comme une simulation dans la mesure où il permet de simuler sur ordinateur ou de reproduire les expériences ayant été faites en laboratoire. En d'autres termes, le système reproduit en quelques minutes les résultats d'expériences réelles effectuées en laboratoire pendant plusieurs heures. Nous

traitons ici de la digestion des molécules d'ADN par des enzymes. Le but principal de l'expérience est de déterminer la carte de restriction d'une molécule d'ADN. En d'autres termes, l'étudiant ou l'utilisateur doit déterminer les différentes positions où les enzymes coupent la molécule d'ADN. Pour ce faire, il aura besoin de savoir la taille de la molécule, les différents enzymes qui coupent cette molécule et le nombre de fois que chaque enzyme coupe la dite molécule et effectuer une série de digestions simples et de digestions doubles. Nous entendons ici par digestion simple la digestion ou la coupure d'une molécule d'ADN par un seul enzyme et une digestion double représente la coupure d'une molécule d'ADN par deux enzymes.

En guise d'exemple, considérons une molécule d'ADN linéaire de 10 kbs (10 kilobases). Cette molécule est coupée par deux enzymes: BamI qui la coupe une fois et PstI qui la coupe deux fois. La figure 11 ci-dessous représente notre molécule entière. Quant à la figure 12, elle représente la digestion simple de BamI. La figure 13 représente la digestion simple de PstI. En ce qui concerne la figure 14, elle représente la digestion double de BamI et PstI; la figure 15 illustre la carte de restriction qui devra être analysée par l'étudiant(e) afin de déterminer la ou les positions de coupe des différents enzymes.

FIG. 11 – *Molécule d'ADN non digérée*

FIG. 12 – *Digestion simple de la molécule d'ADN par l'enzyme BamI*

FIG. 13 – *Digestion simple de la molécule d'ADN par l'enzyme PstI*

FIG. 14 – *Digestion double de la molécule d'ADN par les enzymes BamI et PstI*

Dans cet exemple, la seule information supplémentaire mise à la disposition de l'étudiant est la carte de restriction (voir figure 15) résultant des digestions qu'il (l'étudiant) aura demandé. Cette carte de restriction représente les fragments d'ADN superposés les un sur les autres en fonction de l'enzyme, de la taille du fragment et de la distance de migration. Les figures 11, 12, 13 et 14 constituent des étapes du processus interne du système et ne sont pas connues de l'étudiant(e). À la suite de la réception de la carte de restriction, l'étudiant(e) doit imprimer cette carte afin de mieux l'analyser. Ce processus est répété jusqu'à ce que l'étudiant(e) puisse obtenir les informations qu'il ou qu'elle recherchait. Une fois cela fait, il ou elle soumet ses résultats au système qui lui fournira le solutionnaire s'il y est autorisé. Les figures 16, 17, 18 et 19 présentent quelques fenêtres de ce laboratoire.

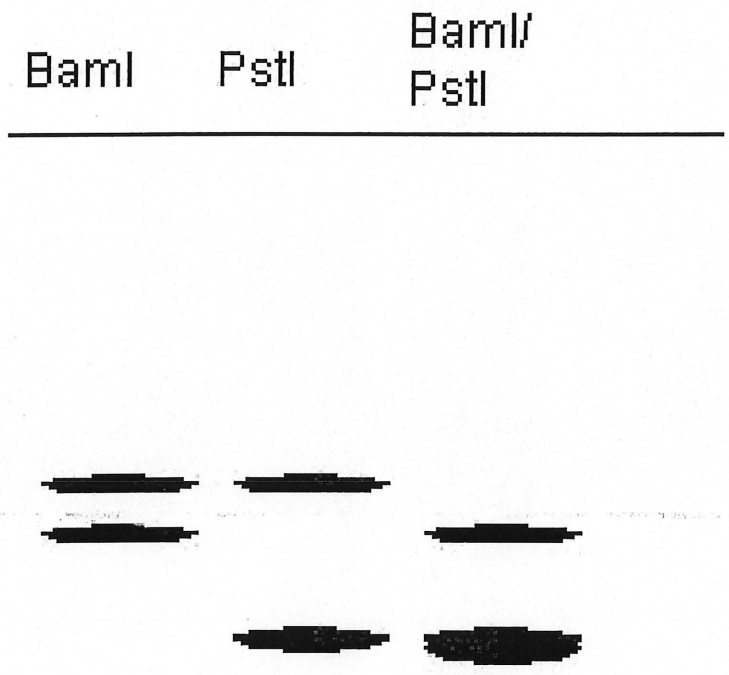

FIG. 15 – *Carte de restriction de la molécule après digestion*

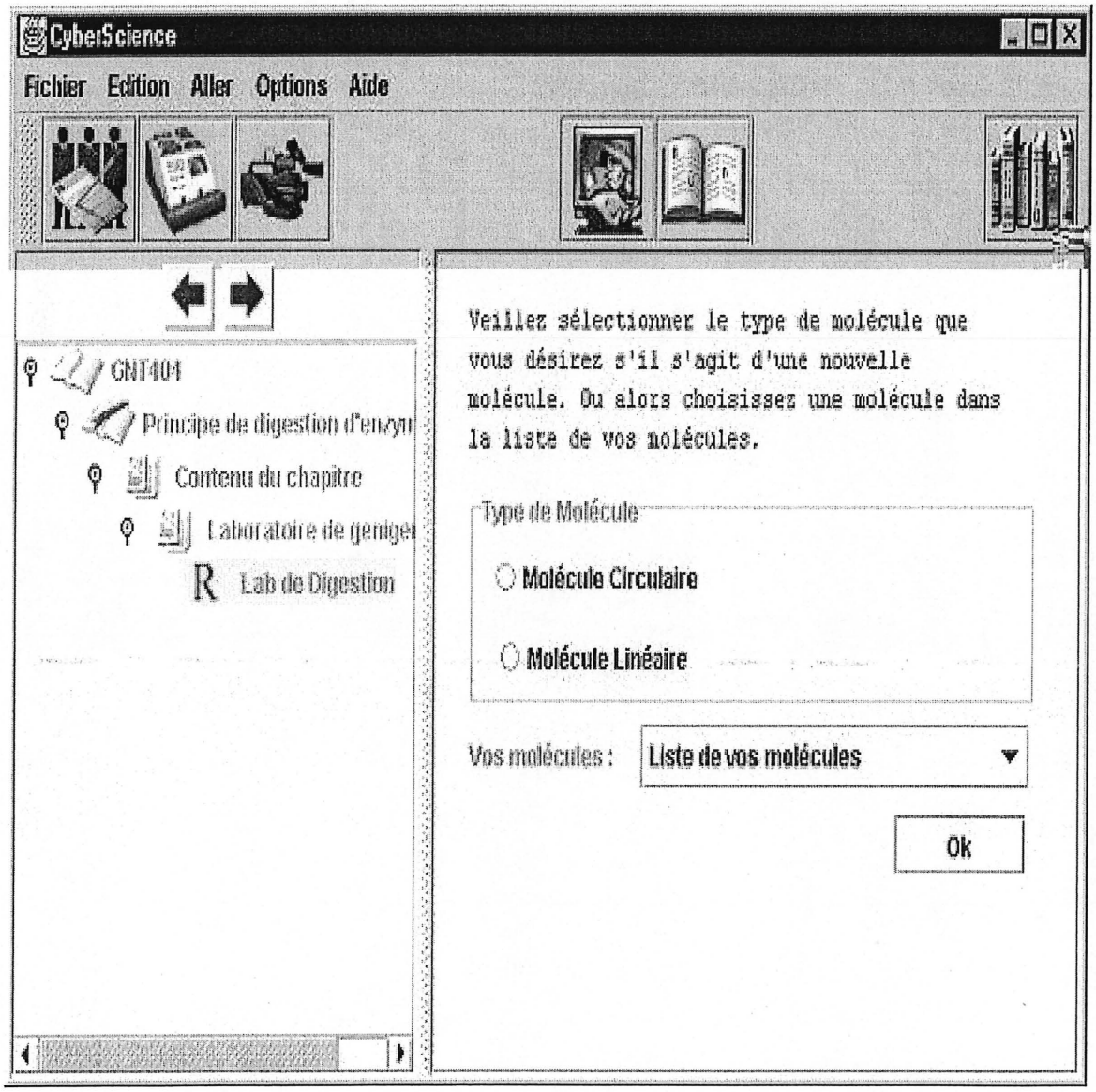

FIG. 16 – *Fenêtre de choix de molécule d'ADN dans le laboratoire de génie génétique*

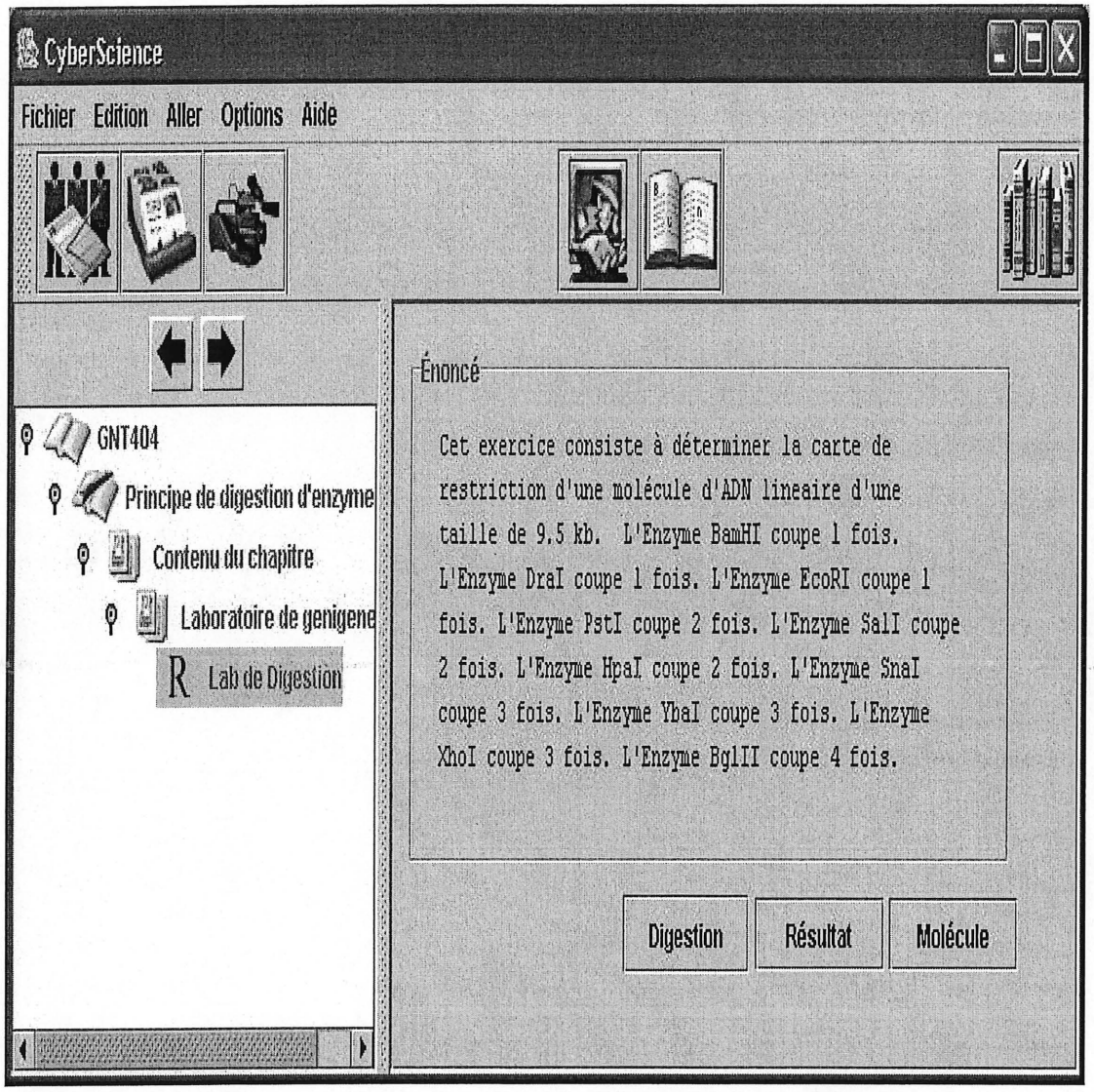

FIG. 17 – *Fenêtre d'énoncé du laboratoire de génie génétique*

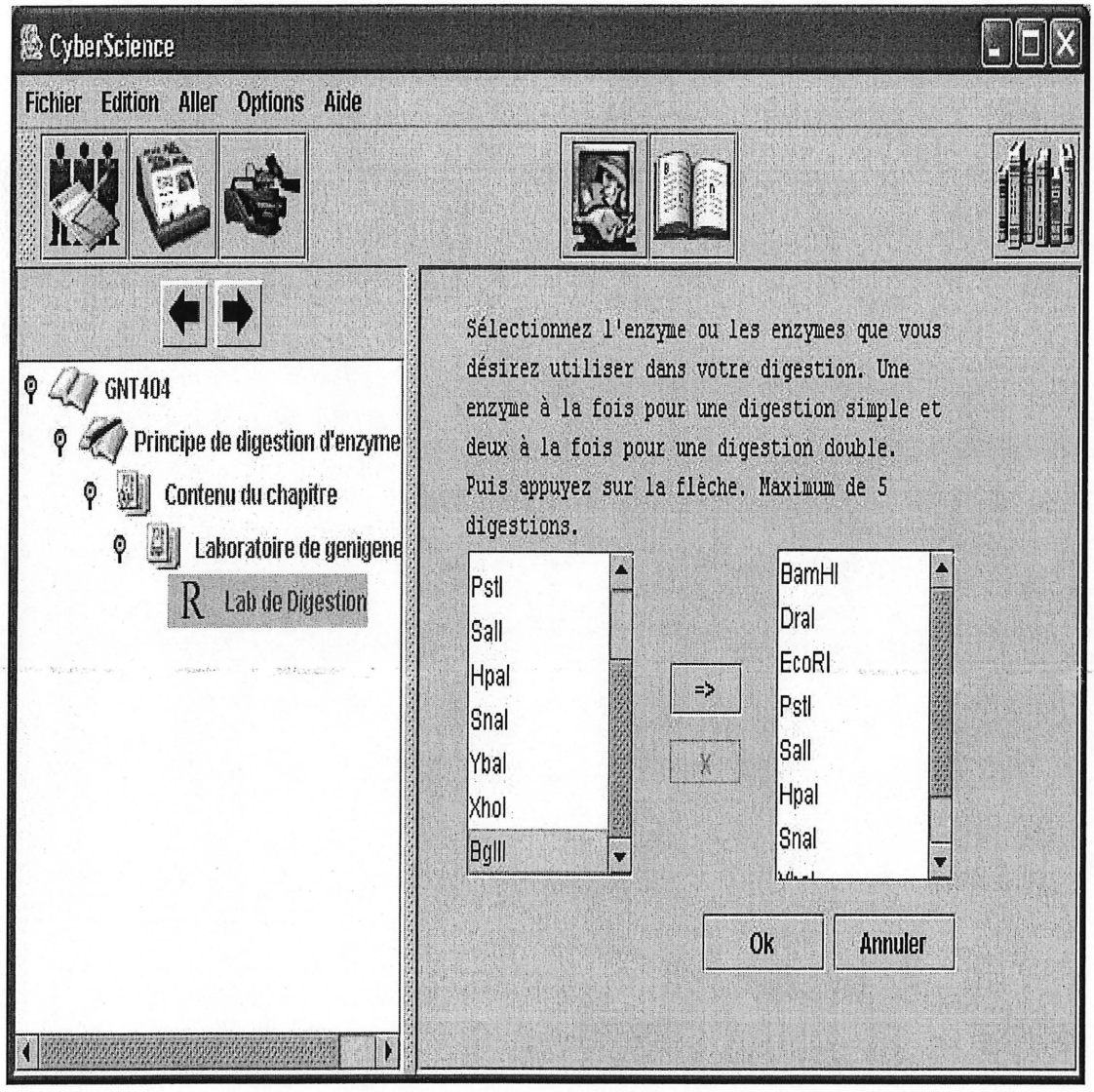

FIG. 18 – *Fenêtre de choix d'enzymes pour la digestion de la molécule d'ADN*

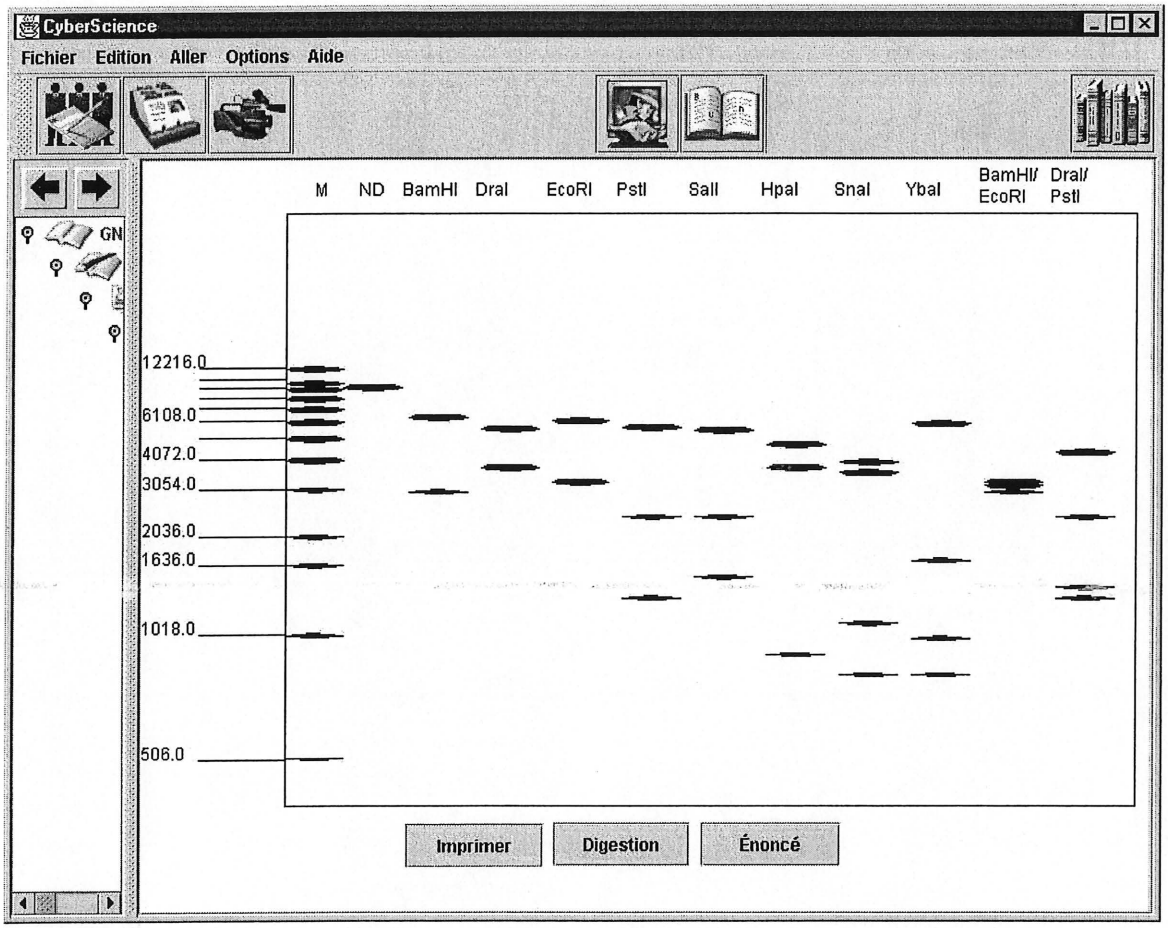

FIG. 19 – *Fenêtre présentant la carte de restriction de la digestion de la molécule*

2.2.2 Microscope virtuel

Toujours en biologie, nous avons participé au développement d'un microscope virtuel en 3D (Three Dimensions). Il s'agit d'un microscope ordinaire ayant exactement les mêmes fonctionnalités qu'un microscope réel. La seule différence est que le microscope virtuel ne fait que simuler les résultats d'une observation microscopique par le traitement de l'image en reduisant sa luminosité. Ce dernier a pour but de donner une formation de base aux étudiant(e)s en ce qui concerne la manipulation d'un microscope ordinaire. La figure 20 présente l'interface de ce laboratoire.

2.2.3 Outils d'apprentissage en mathématiques

En mathématiques, plusieurs outils d'apprentissage et de mise à niveau ont été développés. Nous distinguons :

· le traceur de fonctions (Figure 21) qui permet aux usagers de tracer plusieurs types de fonctions (des fonctions linéaires aux fonctions trigonométriques en passant par les polynômes) dans le but de répondre à des questions, d'observer le tracer d'une fonction ou de déterminer le type de la fonction ;

· l'éditeur de tables de vérité (Figure 22) ;

· l'éditeur de diagrammes de Karnaugh ;

· l'éditeur de diagrammes sagittaux (Figure 23) ;

· l'éditeur de circuits logiques ;

· l'éditeur de preuves (Figure 24).

Tous ces laboratoires virtuels ont été développés en Java et exploitent une base de données Oracle pour les ressources et les problèmes.

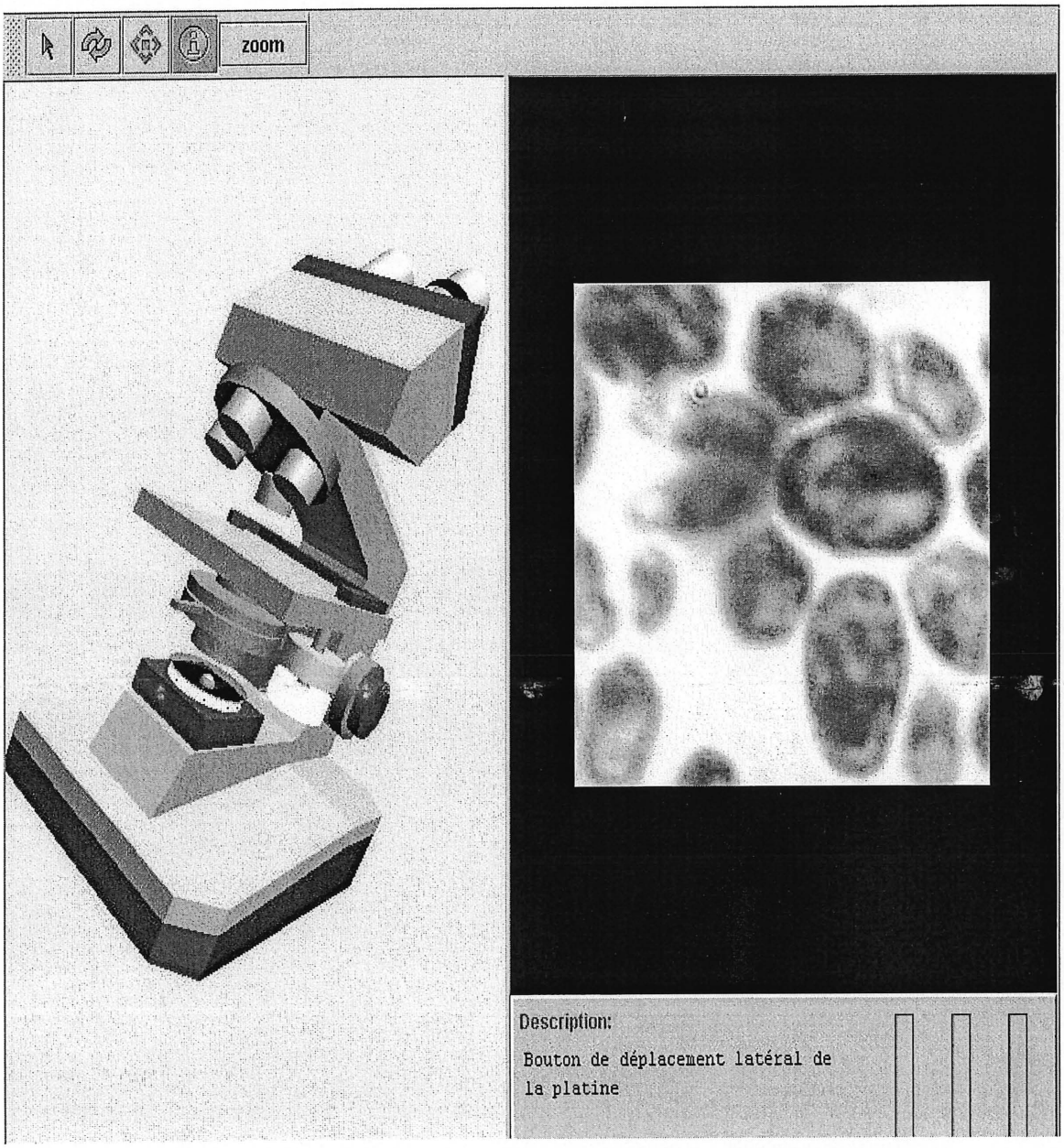

FIG. 20 – *Microscope virtuel*

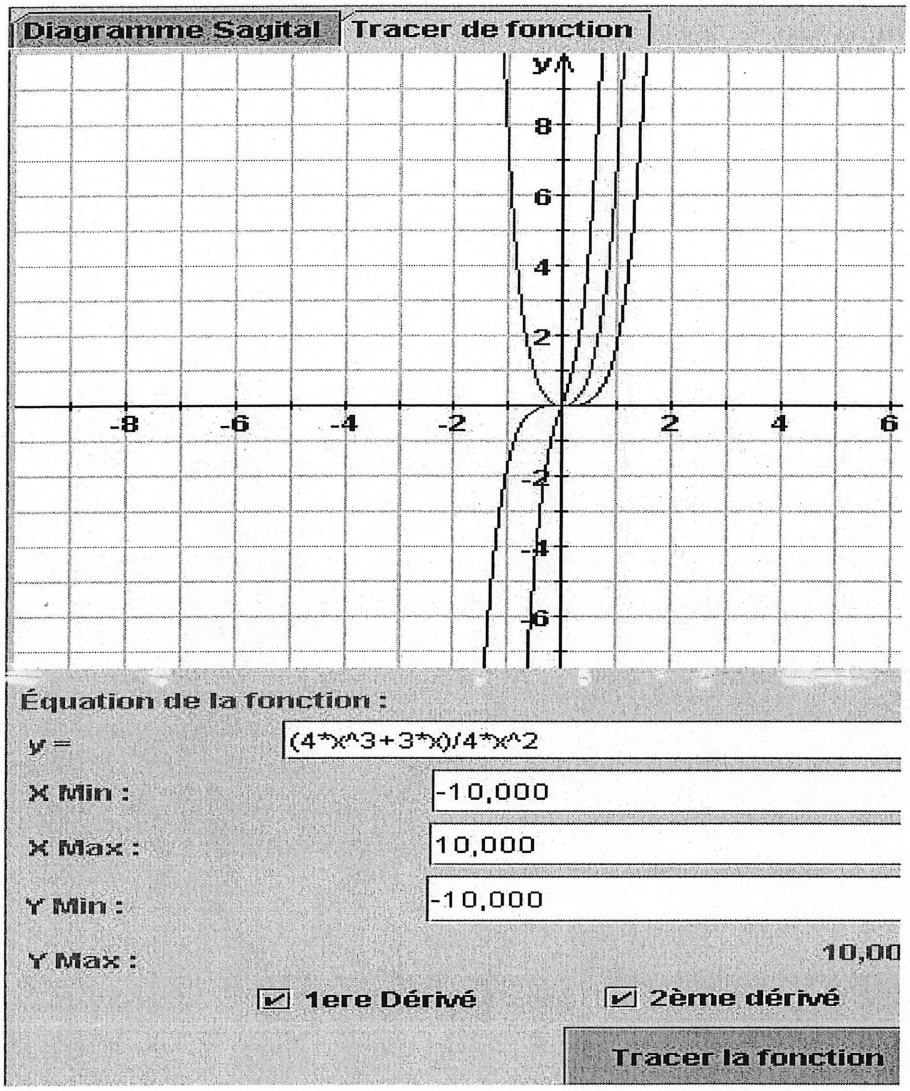

FIG. 21 – *Traceur de fonctions*

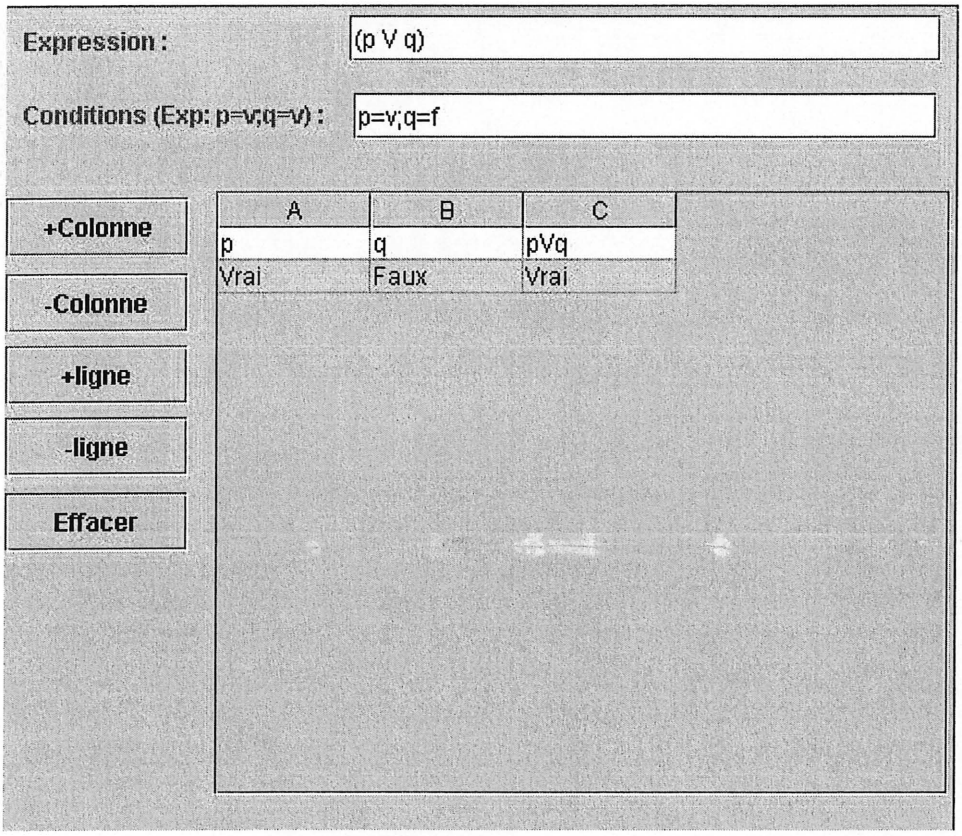

FIG. 22 – *Éditeur de tables de vérité*

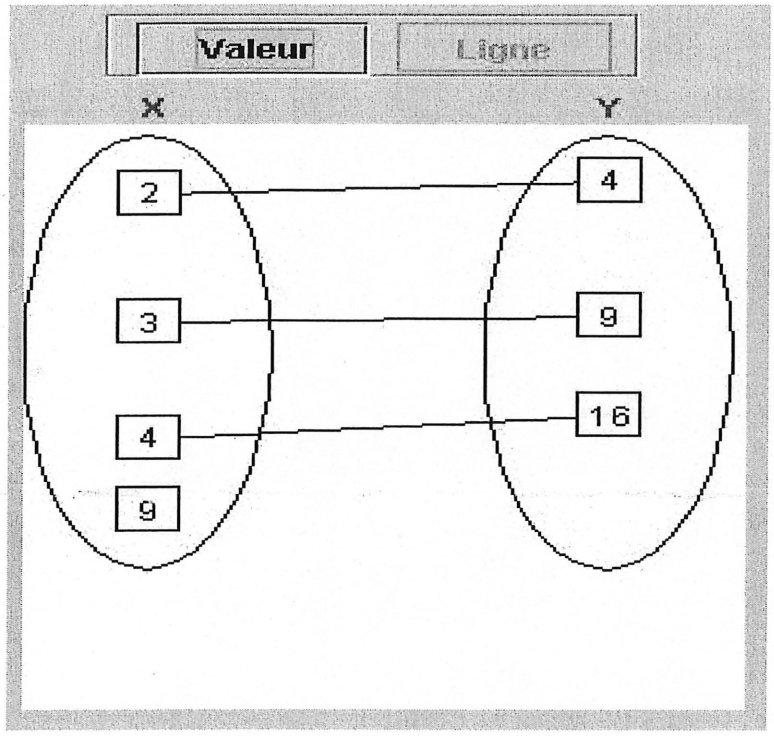

FIG. 23 – *Éditeur de diagrammes sagittaux*

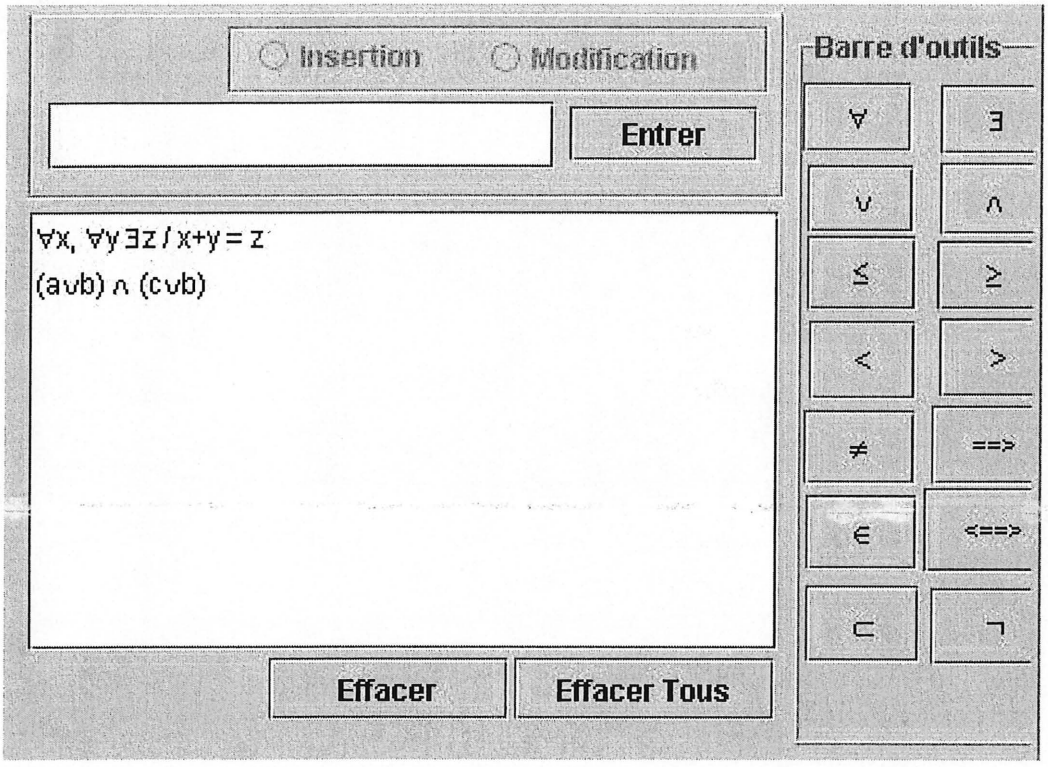

FIG. 24 – *Éditeur de preuves*

2.2.4 Assistance virtuelle dans Cyberscience

Tel que nous l'avons déjà mentionné dans la section 2 (Agents intelligents) du chapitre 1, les systèmes tutoriels intelligents sont des logiciels à base de connaissances dédiés à la formation. De plus en plus, de nombreuses recherches s'intéressent à la conception d'un STI en tant que système multi-agents [31], [34], [22] dans lequel plusieurs agents interagissent dans le but de permettre à un apprenant de réaliser des apprentissages. En fonction du but visé par l'apprentissage et des considérations reliées à l'usager, les agents d'un STI s'appuieront sur des structures de connaissances du domaine enseigné pour guider l'apprenant dans la réalisation de ce but.

Tel que nous l'avons déjà mentionné, on distingue six agents de bases qui sont : un modèle d'agent de raisonnement à base de règles (un système expert), un modèle d'agent de raisonnement à base de cas, un modèle d'agent de « data mining » et un modèle d'agent planificateur. Les deux autres agents sont l'extracteur de connaissances et l'agent de prise de décision, ceux-ci sont en fait des combinaisons des quatre premiers.

La plupart des agents de Cyberscience sont des agents de base. Les variations sont au niveau des règles dans le cas de l'agent de raisonnement à base de règles, des transitions pour l'agent planificateur et des sources de données pour l'agent data mining. Parmi les agents utilisés dans Cyberscience, on retrouve :

- coach : Agent de prise de décision ;
- modèle de l'apprenant : agent de raisonnement à base de règles ;
- système d'aide : agent d'extraction des connaissances ;
- planificateur de cours : agent planificateur.

2.2.5 Technologies de distribution d'applications

Le besoin de communication entre applications informatiques n'a jamais cessé de croître avec la popularisation du réseau Internet. L'architecture client/serveur est née avec la

possibilité de transférer de l'information entre le module serveur et le module client d'une application distribuée.

Les applications distribuées peuvent prendre plusieurs formes. Le système le plus simple est constitué d'un client et d'un serveur échangeant des informations et ce, sans autre interaction avec le reste du ou des réseaux. Le serveur attend et répond aux requêtes des clients, et les clients transmettent leurs requêtes (demande de connexion et autres commandes). Les informations peuvent être échangées de façon bidirectionnelle entre les deux modules.

Le client et le serveur peuvent, à leur tour, devenir simultanément serveur et client pour d'autres applications distantes. Chaque module peut alors jouer le rôle de client, de serveur ou les deux à la fois. Ces modules constituent une application distribuée s'ils interagissent ensemble (exclusivement) pour régler un problème commun. Dans le cas contraire, ils constituent plusieurs applications distribuées (les modules, qui ne se connaissent pas a priori, travaillent à régler plusieurs problèmes qui ne sont pas nécessairement reliés).

La distribution d'applications utilisant les systèmes répartis (sur un même ou sur plusieurs ordinateurs) se fait en utilisant un « middleware », lequel joue le rôle de pont de communication entre les modules de l'application distribuée. En effet, ces derniers (RMI, CORBA, RPC et DCOM sont des exemples bien connus) fournissent tous les outils nécessaires à la gestion de la communication entre les modules. Ils sont appelés « middlewares » puisqu'ils ne sont pas vraiment des exécutables, leur structure est répartie entre les modules de l'application. Les modèles récents de programmation d'applications réparties les plus utilisés sont CORBA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model), RPC (Remote Procedure Call) et RMI (Remote Method Invocation). CORBA, proposée par le Groupe de gestion des objets (OMG), est une architecture pour faciliter les communications entre les applications en environnement hétérogène qui comporte différents langages et systèmes d'exploitation.

DCOM proposée par Microsoft peut être considéré comme une alternative à CORBA ; enfin, RMI de Sun Microsystems vise les communications entre applications programmées en Java seulement. Ces derniers seront élaborés en détail dans les sous-sections suivantes. Nous nous proposons, dans cette section, de présenter les différentes technologies de distribution d'applications existantes. Ceci compte tenu du fait que Cybersciences est un système utilisant l'échange d'informations. Il est donc important dans le cadre de ce travail de présenter les technologies alternatives pouvant permettre la réalisation de ces échanges.

Java RMI

Java RMI propose une méthode relativement simple et puissante pour distribuer des applications sur des machines virtuelles différentes (une machine virtuelle étant une instance de l'interpréteur de bytecode).RMI [27] est un mécanisme permettant d'utiliser sous Java des objets distribués et distants (c'est-à-dire un objet instancié sur une autre machine virtuelle, éventuellement sur une autre machine du réseau) de manière presque transparente pour l'utilisateur, c'est-à-dire de la même façon que si l'objet était sur la même machine virtuelle (JVM) de la machine locale.

Ainsi un serveur permet à un client d'invoquer des méthodes à distance sur un objet qu'il instancie. Deux machines virtuelles sont donc nécessaires (une JVM sur le serveur et une autre sur le client) et l'ensemble des communications se fait en Java. On dit généralement que RMI est une solution « toute Java », contrairement à la norme CORBA de l'OMG permettant de manipuler des objets à distance avec n'importe lequel langage.

Toutes les applications distribuées utilisant RMI sont faites de la même façon. Elles sont divisées en au moins deux modules : le client et le serveur. Le serveur rend disponible des objets aux clients en les enregistrant au gestionnaire de registres. Le client, ayant un lien local ou réseau avec l'ordinateur du serveur, contacte le gestionnaire de registres pour connaître l'emplacement exact de l'objet distant cherché (en spécifiant l'adresse IP

(Internet Protocol) ou DNS (Domain Name System) du serveur, et le nom de l'objet exporté). Une fois l'adresse connue, le client n'a plus à contacter ce service. Il effectue les transactions directement avec le serveur en utilisant l'adresse obtenue.

La communication RMI est essentiellement réalisée en utilisant quatre couches interreliées : la couche application, la couche « Proxy » ou « Stub », la couche « Remote Reference » et la couche Transport (voir la figure 25).

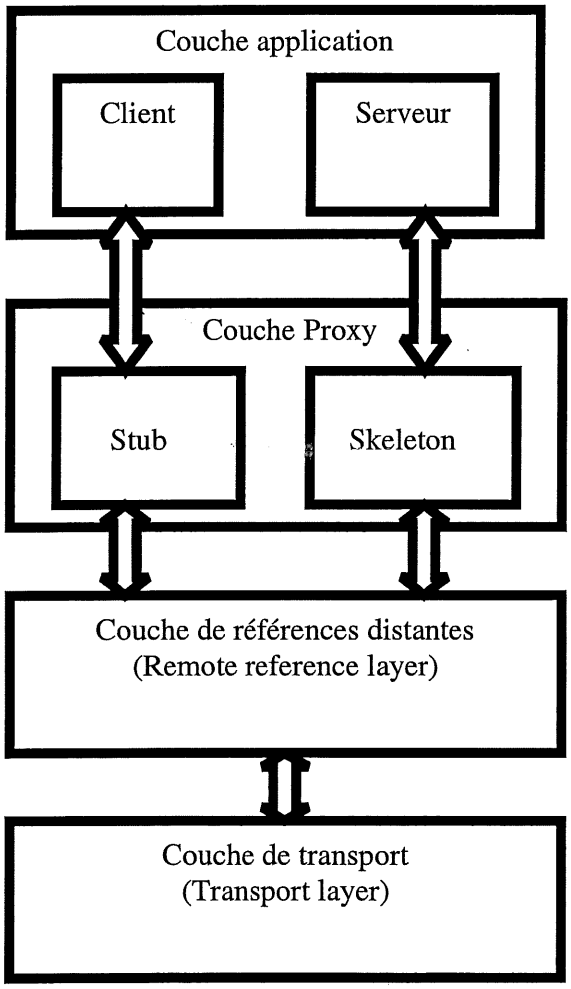

FIG. 25 – *Couches de communication associées à Java RMI*

La première couche est constituée par les modules client et serveur. On y retrouve les appels de haut niveau pour exporter les objets et les accéder.
L'architecture RMI est basée sur le fait que le comportement exporté (les méthodes) est défini dans l'interface (définition des services). L'interface ne contient pas de code exécutable et elle hérite de « java.rmi.Remote ». L'interface « Remote » est essentiellement utilisée pour déclarer aux éventuels clients l'accessibilité à distance d'un objet. Son implémentation est réalisée dans une classe du module serveur. Le client et le serveur sont exécutés en utilisant généralement deux machines virtuelles différentes.
Les deux parties de la couche « proxy » (le « Stub » et le « Skeleton ») sont générées lors de la compilation de l'implémentation et de l'interface. Le « Stub » est le lien entre le client et l'objet distant en d'autres termes, il représente l'objet « remote », une sorte de fantôme qui représente l'objet « remote ». Cette couche intercepte donc les appels faits aux méthodes des objets distants par le client. Elle les redirige ensuite vers le service RMI approprié. Du point de vue du client, l'objet distant est d'abord localisé, créé et il est ensuite retypé (« casted ») du nom de l'interface. Lorsque le client effectue un appel à une des méthodes de l'objet distant, cet appel est effectué directement sur le « Stub ».
La couche des références distantes définit et supporte la sémantique des invocations de la connexion RMI. Elle interprète et gère les références des objets distants. La couche des références distantes est en fait une couche abstraite entre le « Stub », le « Skeleton » et les protocoles de communication (qui sont gérés par la couche de transport). Les interactions entre ces deux couches seront toujours effectuées comme si la connexion était de type connecté (avec un « stream »).
En ce qui concerne la couche de transport, c'est elle qui effectue la connexion entre les machines virtuelles et ce, même en présence d'obstacles réseaux (murs de feu ou « fire wall »). Elle est responsable de la mise en contact des machines virtuelles, de la gestion des connexions, de la vérification des connexions mortes et de la réception de demandes de connexion distante. Toutes les connexions sont des « streams » et utilisent le protocole

TCP de TCP/IP. Il est également possible d'utiliser le protocole UDP à la place de TCP à l'aide d'un service sans connection et sans garantie utilisant l'IP pour le transport de message. Les connexions sont basées sur les adresses IP et les numéros de port des ordinateurs communicants. Un nom DNS peut être utilisé à la place de l'adresse IP.

CORBA

CORBA [36] est la solution proposée par l'OMG (Object Management Group) aux besoins d'interopérabilité d'un nombre grandissant de matériels et logiciels disponibles aujourd'hui. CORBA permet aux applications de communiquer les unes avec les autres où qu'elles soient et quels que soient les concepteurs de ces applications. L'ORB (Object Request Broker) est le logiciel médiateur (« middleware ») orienté objet qui établit la relation client-serveur entre les objets. En utilisant un ORB, une application cliente peut de façon transparente invoquer une méthode sur un serveur d'objets qui peut être sur la même machine ou à travers un réseau. L'ORB intercepte l'appel et il est responsable de trouver un objet pour implémenter la demande, de passer les paramètres, d'invoquer la méthode et de rendre le résultat. L'application cliente n'a pas besoin de savoir où est l'objet invoqué, dans quel langage il a été programmé, et sur quel système d'exploitation il est hébergé ou tout autre aspect du système ne faisait pas partie de l'interface Objet. De fait, l'ORB procure l'interopérabilité entre des applications hébergées sur des machines différentes dans un environnement distribué hétérogène et interconnecte de façon transparente de multiples systèmes à objets [36].

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 différents 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. La figure 26 suivante présente les différentes notions intervenant dans ce modèle objet client/serveur :

· l'application cliente est un programme qui invoque les méthodes des objets à travers le bus CORBA,

· la référence de l'objet est une structure désignant l'objet CORBA et contenant l'information nécessaire pour le localiser,

· l'interface de l'objet est le type abstrait de l'objet CORBA définissant ses opérations et attributs ; celle-ci se définit par l'intermédiaire du langage OMG-IDL (Interface Definition Language),

· la requête est le mécanisme d'invocation d'une opération ou d'accès à un attribut de l'objet,

· le bus CORBA achemine les requêtes de l'application cliente vers l'objet en masquant tous les problèmes d'hétérogénéité (langages, systèmes d'exploitation, matériels, réseaux),

· l'objet CORBA est le composant logiciel cible ; c'est une entité virtuelle gérée par le bus CORBA,

· l'activation est le processus d'association d'un objet d'implantation à un objet CORBA,

· l'implantation de l'objet est l'entité codant l'objet CORBA à un instant donné et gérant un état de l'objet temporaire ; au cours du temps, un même objet CORBA peut se voir associer des implantations différentes,

· le code d'implantation regroupe les traitements associés à l'implantation des opérations de l'objet CORBA ; cela peut être, une classe Java aussi bien qu'un ensemble de fonctions du langage de programmation C,

· l'application serveur est la structure d'accueil des objets d'implantation et des exécutions des opérations ; cela peut être, par exemple, un processus Unix.

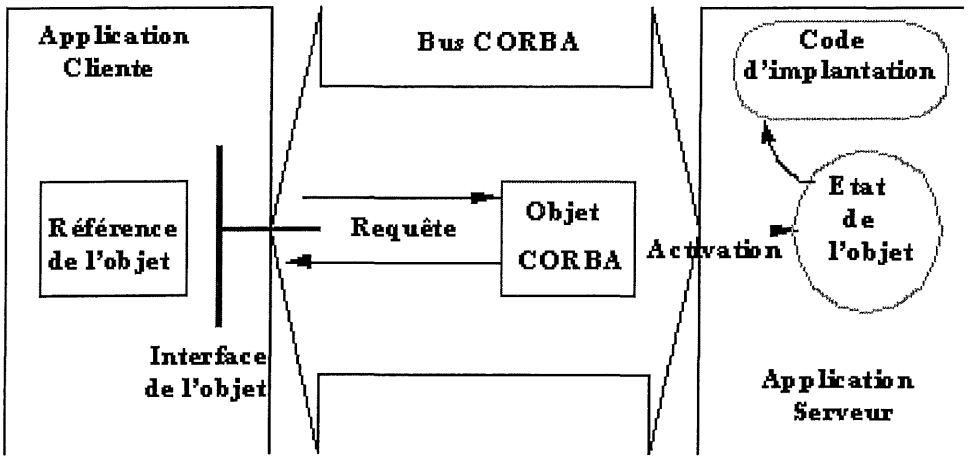

FIG. 26 – *Modèle objet client/serveur*

L'architecture de CORBA dérive de l'OMA (Object Management Architecture). L'OMA est l'architecture globale respectant toutes les spécifications de l'OMG. L'OMG a donc créé l'OMA qui contient plusieurs éléments dont l'ORB, qui en est la partie centrale. CORBA implémente uniquement les fonctionnalités de l'ORB et non l'ensemble du « middleware » OMA. Par abus de langage, on confond OMA et CORBA. CORBA est devenu le nom désignant le logiciel médiateur s'appuyant sur les spécifications de l'OMG. La figure 27 illustre l'architecture de l'OMG.

· Le bus d'objets répartis est la clé de voûte de l'architecture globale de l'OMG. Il assure le transport des requêtes entre tous les objets CORBA. Il offre un environnement d'exécution aux objets masquant l'hétérogénéité liée aux langages de programmation, aux systèmes d'exploitation, aux processeurs et aux réseaux.

· Les services objets communs (« CORBA services ») fournissent sous forme d'objets CORBA, spécifiés grâce au langage OMG-IDL, les fonctions systèmes nécessaires à la plupart des applications réparties. Actuellement, l'OMG a défini des services pour les annuaires (Nommage et Vendeur), le cycle de vie des objets, les relations entre objets, les événements, les transactions, la sécurité, la persistance, etc. Au fur et à mesure des

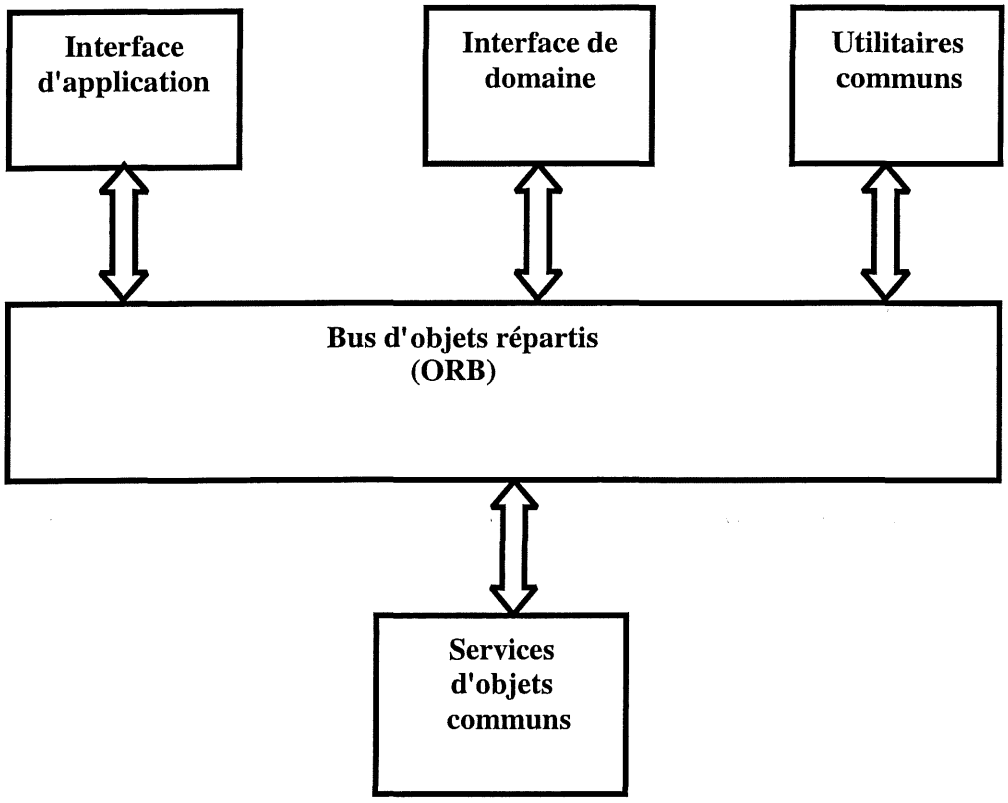

FIG. 27 – *Modèle de l'architecture de l'OMG*

besoins, l'OMG ajoute de nouveaux services communs.

· Les utilitaires communs (« CORBA facilities ») sont des canevas d'objets qui répondent plus particulièrement aux besoins des utilisateurs. Ils standardisent l'interface utilisateur, la gestion de l'information, l'administration, le « workflow ».

· Les interfaces de domaine (« Domain Interfaces ») définissent des objets de métiers spécifiques à des secteurs d'activités comme la finance, la santé et les télécommunications. Leur objectif est d'assurer l'interopérabilité sémantique entre les systèmes d'information d'entreprises d'un même métier.

· Les interfaces d'application (« Application Interfaces ») sont celles qui sont spécifiques à une application répartie et ne sont donc pas standardisés. Toutefois, dès que le rôle de ces objets apparaît dans plus d'une application, ils peuvent alors rentrer dans une des catégories précédentes et donc être standardisés par l'OMG.

Remote Procedure Call

La technologie RPC (Remote Procedure Call) permet à un client d'exécuter une fonction d'un serveur distant. C'est un système essentiellement basé sur la procédure qui n'intègre pas l'orienté objet (exception faite de DCE-RPC Distributed Computing Environment-Remote Procedure Call) [26].

Les caractéristiques principales de RPC sont les suivantes :

· les procédures peuvent être distribuées sur la même machine ou sur un réseau,

· le système est non connecté (« connectionless ») ; La connexion n'existe que pour le temps de l'appel à la procédure,

· les détails du mécanisme de transport utilisés par RPC sont cachés au programmeur ; l'appel à une procédure distante apparaît comme s'il était fait localement,

· RPC supporte la portabilité et l'interopérabilité et ce, sous différents systèmes d'exploitation,

· RPC supporte également le « Multithreading », la sécurité réseau, l'intégrité des

ressources et l'intégrité des données (lors des transferts).

L'application distribuée est divisée en deux modules : le client et le serveur. Comme pour Java RMI, les applications RPC doivent s'enregistrer (au « Port Mapper ») afin d'être visibles pour les clients. Le client demande ensuite un ou des services (de procédures) au « Port Mapper », lequel retourne l'adresse du service demandé. Le « Port Mapper » peut desservir plusieurs services différents et le serveur peut desservir plusieurs clients simultanément.

La communication entre le serveur et le client se fait en passant par le « Stub » côté client et le « Skeleton » côté serveur. C'est au niveau du « Stub » et du « Skeleton » que se fait tout le travail de transaction entre les deux parties de l'application distribuée. Ce système représente une simplification importante par rapport aux « sockets ». Par exemple, le développement de protocoles de communication complexes n'est plus nécessaire avec RPC. Un inconvénient accompagne toutefois cette technologie. On doit créer une interface entre le client et le serveur, et la compiler séparément. RPC existe sous plusieurs bannières. On note entre autres : OSF (Opend Software Foundation) RPC, Sun RPC, Netwise RPC, IBM OS/2 DPL et Sybase RPC.

DCOM

DCOM est un ensemble d'extensions basées sur le DCE-RPC qui intègre l'orienté objet. DCOM est la technologie de composants développée par Microsoft pour Windows. C'est en fait la pierre angulaire de toutes les nouvelles technologies introduites par Microsoft dans Windows, telles qu'OLE (Object Linking and Embedding), DirectX, ActiveX. La grande force de DCOM est le nombre de machines sur lesquelles on peut le trouver, tout simplement en raison des parts de marché de Windows dans le monde. La figure 28 illustre l'agencement des différentes composantes de DCOM. C'est un logiciel médiateur qui ressemble à CORBA à la différence qu'il est fortement adapté aux produits et plates-formes Microsoft. Il est une extension de COM (Component Object Model) en ce sens

qu'il permet de séparer une application en plusieurs parties (composantes ou objets) et de les distribuer sur un réseau [26].

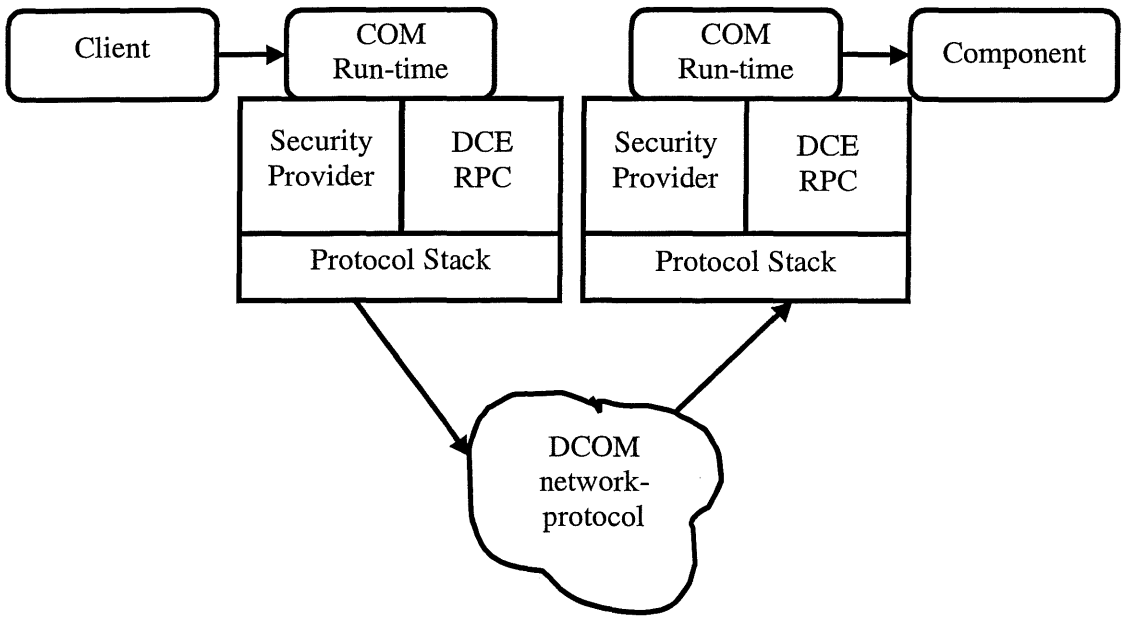

FIG. 28 – *Composantes de DCOM*

DCOM spécifie les interfaces entre les composantes au niveau binaire et non au niveau source. Ceci signifie que la manière d'utiliser les interfaces des composantes n'est pas imposée au niveau des langages de programmation (il n'y a pas de projection du langage de description des interfaces pour chaque langage de programmation). La seule chose qui compte est la structure de ces interfaces ainsi que les conventions d'appel utilisées par ces méthodes. Ce choix technique fournit une certaine liberté dans la manière d'implémenter et d'utiliser les composantes, mais rend les choses techniquement beaucoup plus compliquées. Dans DCOM, les communications et la gestion de la durée de vie des composantes sont prises en charge par le programmeur.

DCOM fournit tous les services de base d'un système à composantes : création et gestion

de la durée de vie des composantes, « marshalling », répartition, gestion des versions, réutilisabilité, extensibilité, gestion des langages de scripts, détection des interblocages, etc. À présent, il est utilisé par la plupart des services de Windows. L'une des manières les plus faciles de l'utiliser est bien entendue de recourir à des outils comme Visual Basic et ses dérivés. Mais il est toujours possible de programmer et d'utiliser des composantes de manière plus classique, en C ou en C++. Toutefois, il est fortement recommandé d'utiliser le C++ et les librairies ATL (Active Template Libraries) de Microsoft, car la programmation directe de DCOM est assez complexe.

DCOM a subi différentes évolutions et changements de nom du fait du marketing de Microsoft. La technologie de DCOM s'est initialement appelée OLE car il permettait d'incorporer directement des objets de d'autres applications ou de réaliser des liens sur ces objets dans un document composite. Techniquement parlant, OLE est un ensemble d'interfaces permettant d'obtenir ce résultat d'intégration et de lien entre les applications. Microsoft a ensuite fait la distinction entre ces interfaces et le modèle objet de composants sur lequel elles se basent. Ce modèle objet a été appelé COM. Dès le départ, Microsoft avait prévu que COM pourra être distribué, mais l'implémentation de DCOM n'est apparue réellement qu'avec Windows NT 4. Par la suite, Microsoft a développé son API de jeux pour Windows 95 nommée DirectX. Cette API utilise exclusivement l'architecture de COM et a inauguré l'introduction de la lettre X dans les noms de technologies de Microsoft. Avec l'arrivée d'Internet, les contrôles ActiveX sont apparus. Ce n'étaient pourtant rien d'autre que des contrôles OLE classiques, mais il fallait trouver un nom spécifique pour les développeurs Internet. Finalement, les ATL (Active Template Libraries) ne sont rien d'autre qu'une librairie C++ permettant de développer des contrôles ActiveX plus facilement. En fait, un contrôle ActiveX n'est rien d'autre qu'une composante OLE ou une composante DCOM ou une composante COM tout court. Tout cela, c'est en fait la même chose, ce sont des composantes ayant plus ou moins d'interfaces à fournir à leurs clients.

Quoi qu'il en soit, DCOM est à présent incontournable. Il est omniprésent dans Windows, et toutes les nouvelles technologies présentes et à venir ne seront pas utilisables sans lui. La programmation classique de Windows est donc en train de subir une mutation profonde et tous les programmeurs auront un jour à s'y mettre sérieusement. Cependant, cela ne va pas être facile car toute la complexité de DCOM resurgira. Mais cette complexité est le prix à payer pour accéder aux nouvelles fonctionnalités de Windows et pour garantir la pérennité et la réutilisabilité des logiciels.

À présent que nous avons fait le tour du contexte de cette recherche, les réalisations du projet Cyberscience et les différents outils disponibles susceptibles de nous permettre de résoudre notre problème, nous allons dans le chapitre suivant présenter et justifier notre choix de solution ainsi que sa conception.

Chapitre 3

Développement d'une architecture pour la couche de communication

3.1 Architecture de la couche de communication

Nous considérons comme protocole tout ensemble de spécifications précisant les moyens logiciels et physiques qu'a un ensemble d'équipements pour partager des informations et, par extension, les logiciels permettant d'y répondre.

Nous entendons par couche de communication une couche intermédiaire entre deux applications ou systèmes qui dans notre cas sont deux composantes d'un même système à savoir les laboratoires virtuels et les agents intelligents. Cette couche permet aux deux composantes de communiquer de façon efficace en utilisant une forme de transmission d'informations par messagerie. La figure 29 présente l'architecture générale de la couche de communication.

L'interface "étudiant" représente la partie visible du système ou encore la partie responsable de l'interaction entre l'étudiant ou l'utilisateur, et le système. Tel que nous l'avons indiqué dans l'introduction, un laboratoire virtuel est un ensemble de laboratoires ayant au moins un élément virtuel. En d'autres termes, il s'agit d'applications permettant aux

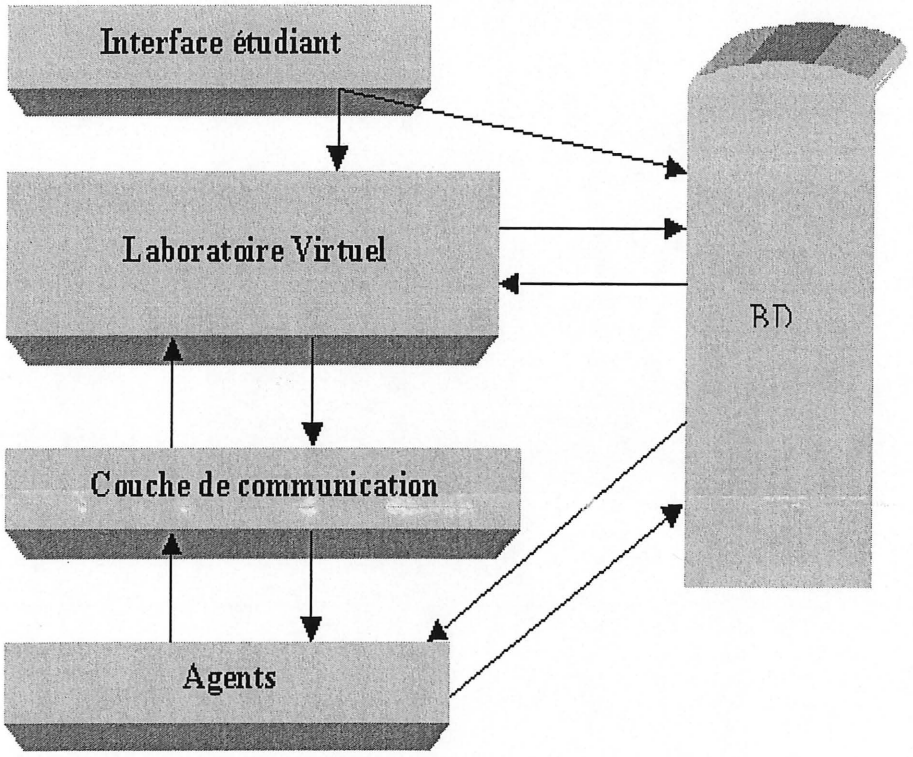

FIG. 29 – *Architecture générale du système*

étudiants de simuler sur ordinateur les expériences qui se font en laboratoire. Quant à la couche de communication, celle qui nous intéresse particulièrement dans cette section, elle sert d'intermédiaire entre les laboratoires et les agents. Elle sert de canal de communication permettant aux deux composantes de communiquer entre eux. Et enfin les agents représentent la composante du système responsable de la validation des travaux des étudiants, des conseils et du « coaching » de ces derniers.

3.1.1 Choix et justification d'une approche

Nous portons notre choix sur un système de couches utilisant une communication par messagerie comme le font les agents. Notre choix a été motivé par le fait que le système utilisé dans le cadre de cette recherche ne pouvait tout à fait être considéré ni comme un système multi-agents (SMA) car, à ce moment, les laboratoires devront être considérés comme des agents, ce qui n'est pas le cas, ni comme un système à composantes donc pas tout à fait un système distribué. Ceci explique la raison pour laquelle nous n'avons pas utilisé des outils de développement d'agents tels MadKit [24] ou Aglet [5] pour ne citer que ceux là. MadKit est une plate-forme multi-agents développé en Java construite sur un modèle organisationnel. Il fournit des équipements d'agents généraux (gestionnaire de cycle de vie, le passage de message, la distribution...) et permet la haute hétérogénéité dans des architectures d'agent et des langues de communication et des personnalisations diverses. En ce qui concerne Aglet, il s'agit là d'une architecture d'agents mobiles sur Internet combinant agent et applet.

Quant à CORBA, RMI et DCOM, leur complexité d'apprentissage et de mise en uvre nous a incités à nous tourner vers une solution plus simple. Alors nous trouvons plus facile de proposer aux développeurs une API qu'ils utiliseront tout en respectant une certaine ontologie prédéfinie et facilement extensible car cette dernière est contenue dans un document de définition de type de document (DTD).

3.1.2 Couche de communication

La couche de communication telle que nous l'avons édemment représente l'outil intermédiaire de communication entre les composants de notre système. Son architecture s'appuie sur le modèle client/serveur classique. Le choix de ce modèle a été motivé par le besoin d'étendre l'utilisation de cette couche à un plus grand éventail de projets car nous aurions bien pu utiliser un modèle basé sur la mémoire partagée. Ce modèle assure la communication entre processus sur une même machine ou sur des machines différentes en utilisant le protocole TCP/IP et un mode de communication point à point asynchrone. L'implémentation du modèle client/serveur s'inspire directement de la philosophie Unix : le moteur (« Back-end ») très clairement séparé de l'interface graphique (« Front-end »). La communication entre les deux composants du système s'effectue grâce à une connexion par canal d'échange (« Socket »). Le choix de « Socket » en mode connecté au lieu du mode non connecté permet de prévenir les pertes de données au cours du transport. La fiabilité des transmissions est assurée par le protocole TCP. Il est néanmoins possible de diffuser à un ensemble de processus (ou « broadcast ») en effectuant des communications point à point sur tous les récepteurs. Avec une telle architecture, il sera possible aux applications de fonctionner aussi bien sur la même machine que sur deux machines distinctes. Dans ce cas précis (celui du projet Cyberscience), les laboratoires virtuels et les agents peuvent, à leur tour, devenir simultanément serveur et client car, ils peuvent, à leur tour, faire des requêtes l'un à l'autre. De plus, cette couche utilise l'API Java de conversion entre Java et XML JAXB(« Java API for XML Binding »). La figure 30 nous présente une architecture de cette couche.

La notion de DTD fait partie intégrante de la spécification XML. Elle permet de définir un modèle de données qui servira ensuite à valider toutes les instances supposées être conformes au modèle. Une DTD se compose d'une liste de déclarations dont chacune définit un module d'un document. Les déclarations de base sont les déclarations d'éléments et d'attributs des déclarations. Les déclarations d'élément définissent ce qu'un ensemble

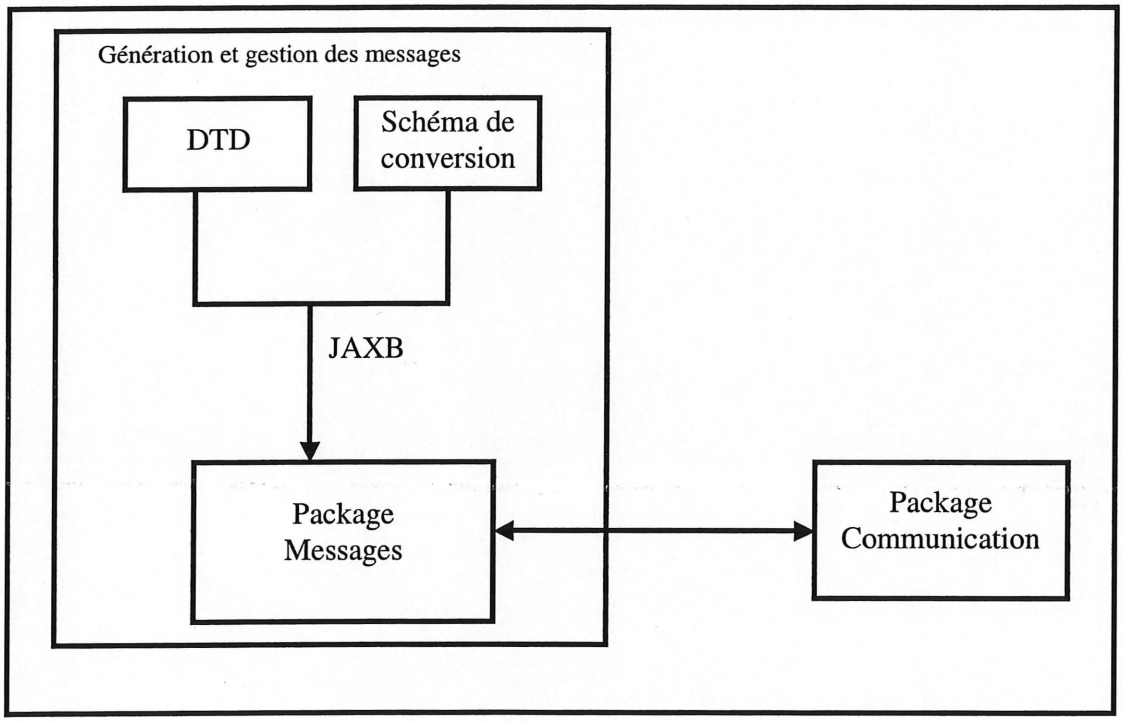

FIG. 30 – *Architecture de la couche de communication*

particulier d'étiquettes dans un document de XML peut contenir. Les déclarations d'attribut accompagnent des déclarations d'élément et fournissent des informations supplémentaires au sujet de l'élément. Le modèle lui-même permet de spécifier une hiérarchisation d'objets typés et valués selon des attributs. Il permet aussi, au travers des entités paramètres, de modulariser l'écriture des DTD. Ce modèle est largement implémenté par tous les outils de création et de modification de documents issus du monde SGML. Dans le cadre de notre projet, la DTD contient les différents éléments qui seront utilisés par JAXB afin de générer les différentes classes Java qui permettront la gestion des différents messages échangés par les différentes applications qui se partagent des données et des variables. Dans ce cas particulier, nous utilisons la DTD dans le but de définir une ontologie qui sera utilisée dans la composition des messages afin de permettre aux agents d'assistance virtuelle et aux laboratoires virtuels de pouvoir communiquer.

Le schéma de conversion est un fichier contenant des instructions sur la façon dont on liera un schéma aux classes. Par exemple, le schéma de conversion pourrait contenir une instruction sur le type primitif qui devra être lié à la valeur d'un attribut dans la classe produite. Ce schéma doit être écrit dans un langage de conversion ou de liaison définie par JAXB. Ce langage est basé sur XML et décrit la relation ou la jointure d'un schéma source à un ensemble de classes dérivées. Un schéma écrit à l'aide de ce langage indique les détails de classes dérivées d'un schéma source particulier.

La librairie de « Messages » contient un ensemble de classes dérivées de la compilation de JAXB en fonction de la DTD et du schéma de conversion puis compilées en Java. La plupart de ces classes dérivées du schéma de conversion sont des classes de contenu mais pas toutes. Une classe de contenu est une classe dérivée d'un composant de production de contenu du schéma de conversion. On y retrouve aussi des classes d'éléments qui sont similaires aux classes de contenus et correspondent directement à une déclaration de type d'éléments dans le schéma. On retrouve aussi une classe d'éléments de racine qui est une classe d'éléments correspondant à un type d'éléments dont les instances peuvent

servir de racine du document. Les instances d'une classe de contenu sont appelées objets de contenu (« content objects ») et les instances d'une classe d'éléments sont appelées objets d'éléments (« element objects »). Cette librairie contient aussi une interface à implémenter par les différentes applications qui désirent communiquer entre elles. Cette interface permettra d'implémenter la fonction qui sera appelée par l'instance de la classe Message pour le traitement d'un nouveau message.
La librairie « Communication » est un ensemble de classes Java qui est utilisé par la classe messages pour acheminer les différents messages qui seront en format XML à travers le protocole TCP/IP. Ce « package » contient les différents « sockets » client et serveur ainsi que toutes les classes traitant ou gérant les différentes connexions. Les « sockets » sont des points de connexion pour une communication. De prime à bord, les modèles de communication qui sont accessibles à travers les « sockets » sont tout à fait analogues à deux outils de la vie courante : le courrier et le téléphone. Dans tous les cas de communication, il y a au moins deux entités qui communiquent ; il faut donc au moins deux points d'entrée.

3.1.3 Architecture détaillée

Après avoir passé en revue l'architecture générale de notre système dans les deux premières sections de ce chapitres, nous allons à présent essayer de le décomposer afin de mieux comprendre les différents objets, leurs rôles et leur fonctionnement.

Génération et gestion des messages

Cette section présente la génération des différentes classes responsables de la gestion des messages ou des données à échanger.
La figure 31 présente le processus de génération de la librairie de gestion des messages à transférer, cette librairie est générée en réponse à la compilation JAXB des fichiers

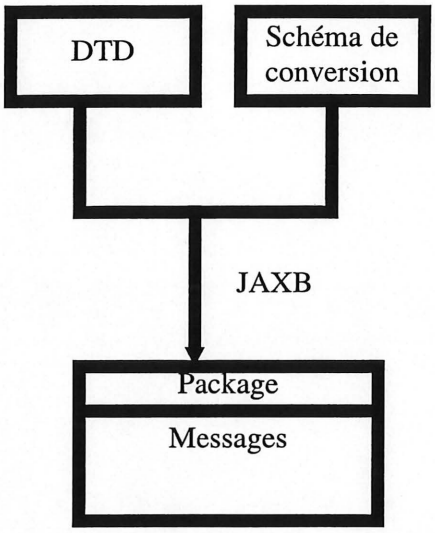

FIG. 31 – *Graphe de génération de la librairie de messages*

contenant la DTD et le schéma de conversion et est constitué de classes java permettant la gestion des messages.

DTD

La notion de DTD fait partie intégrante de la spécification XML. Une DTD définit la structure d'un document XML. Une DTD est caractérisée par un ensemble de règles décrivant les marqueurs autorisés pour ce document, l'ordre dans lequel ils devront apparaître et les relations des marqueurs les uns par rapport aux autres.
L'objectif d'une DTD est de définir un modèle de données formel afin que les outils puissent valider, de façon automatisée, la conformité d'une classe de documents XML à ce modèle. Cette activité est nécessaire pour tous les processus de traitement de l'information XML qui doivent pouvoir présupposer ce qui est contenu dans un document. Par ailleurs, l'objectif d'une DTD est de définir tout ce qui est nécessaire comme ressources à tous les documents d'une classe donnée. Donc elle a pour rôle de définir le vocabulaire

et la structure d'un document. En d'autres termes, elle permet de spécifier les éléments et leurs attributs ainsi que leurs relations, leurs ordres et leurs fréquences d'apparition. Contrairement, à un document XML dans lequel on peut ou non fournir la DTD ce qui permettra de dire d'un document qu'il est valide (donc il comporte une DTD qui servira de modèle de validation) ou qu'il est bien formé (donc ne comporte pas de modèle de validation mais répond aux règles de base de XML), la DTD est obligatoire ici car c'est elle qui fournit les différents attributs qui sont nécessaires à la génération et au bon fonctionnement d'une classe.

Schéma de conversion

Le schéma de conversion est un document non obligatoire mais cependant très important dans la génération des classes résultantes de la compilation à l'aide du compilateur de schéma de JAXB. Ce document est construit à partir de la DTD en utilisant un langage de conversion défini par JAXB qui est lui aussi défini en XML. Ce document nous permet de spécifier :

· le nom de la librairie, des classes dérivées et des différentes méthodes de chaque classe ;

· le type des attributs et des méthodes de chaque classe ;

· les constructeurs, les interfaces et les énumérations ;

· la définition de la conversion des données ou des attributs de type différent de la chaîne de caractères (type « String »).

En d'autres termes, le schéma de conversion permet de spécifier les aspects de la conversion qui n'ont pas été directement ou explicitement faits dans la DTD. On n'est cependant pas tenu de définir toutes les déclarations faites dans la DTD, donc tous les détails ne sont pas nécessaires ici compte tenu du fait que le compilateur de schéma peut générer le code Java des classes sans schéma de conversion. De la même façon, le compilateur de schéma sera en mesure d'utiliser le schéma de conversion spécifié par défaut dans JAXB pour

les composants de la DTD qui n'ont pas été explicitement mentionnés dans le schéma. La déclaration par défaut dont il est question plus haut réduit le verbiage du schéma de conversion et le rend plus robuste à une évolution de la DTD. De plus, les règles de cette dernière sont suffisamment puissantes pour pouvoir convertir la DTD dans plusieurs cas sans nullement avoir besoin d'un schéma de conversion.

Fonctionnement de JAXB

JAXB prend en entrée la DTD et le schéma de conversion pour générer du code Java grâce à son compilateur de schéma. Ce code une fois compilé à l'aide du compilateur Java est utilisable par toute application en générant un arbre d'objets Java représentant les données XML valides par rapport à la DTD par l'instanciation des classes générées. La figure 32 illustre le processus de génération des classes; elle montre clairement que JAXB prend en entrée la DTD et le schéma de conversion, et produit dans un premier temps les fichiers sources de Java après la compilation faite par le compilateur de schéma. Ces fichiers sources sont compilés à leur tour par le compilateur traditionnel de Java et fournissent ainsi les différentes classes à être utilisées par les applications. Ceci ne met cependant pas fin aux fonctionnalités ou au fonctionnement de JAXB. Nous élaborerons le deuxième aspect de ce fonctionnement dans la section du traitement de données. Ceci se fera dans le cas d'utilisation de ce système (voir chapitre 4). Il est à noter qu'il est possible d'obtenir des résultats identique en utilisant l'outil Castor [4].

3.1.4 Communication

Dans cette section, nous nous concentrons sur la façon dont les messages produits par l'encapsulation des données dans un format XML résultant de la section précédente sont gérés (gestion et échange entre les deux applications désirant communiquer ou se transmettre des données ou informations). La communication entre processus ou application

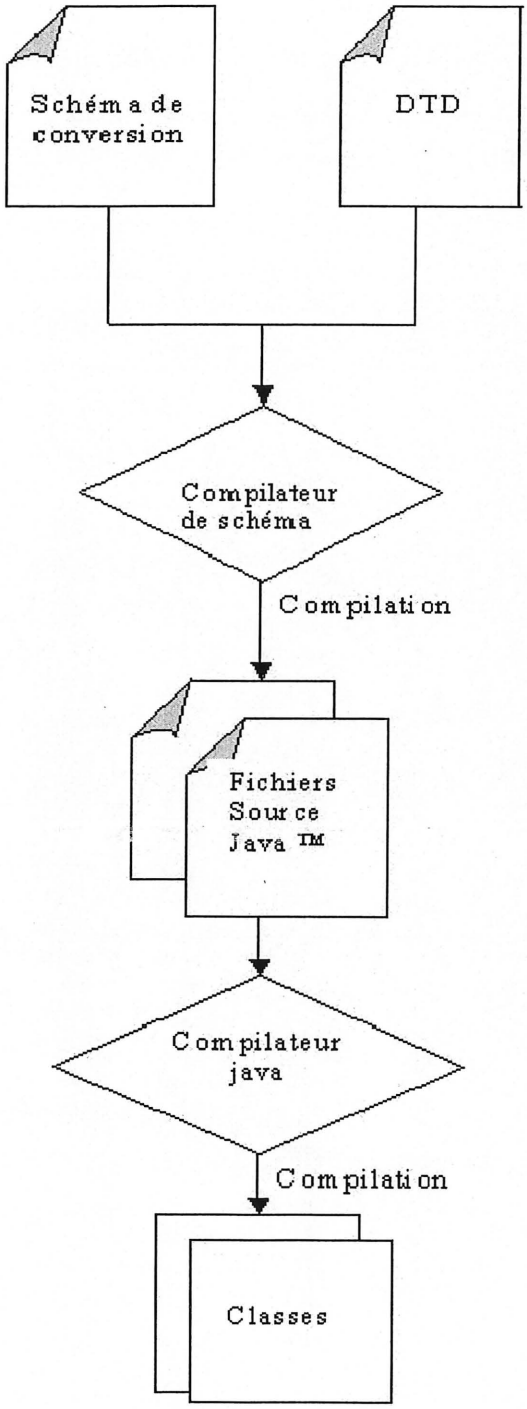

FIG. 32 – *Processus de génération des classes*

sur la même machine ou sur deux machines distinctes reliées en réseau se fait par le protocole TCP/IP avec le modèle point à point asynchrone. Il est néanmoins possible de diffuser à un ensemble de processus (ou « broadcast ») en effectuant des communications point à point sur tous les récepteurs. La figure 33 présente l'architecture à implémenter pour assurer cette communication.

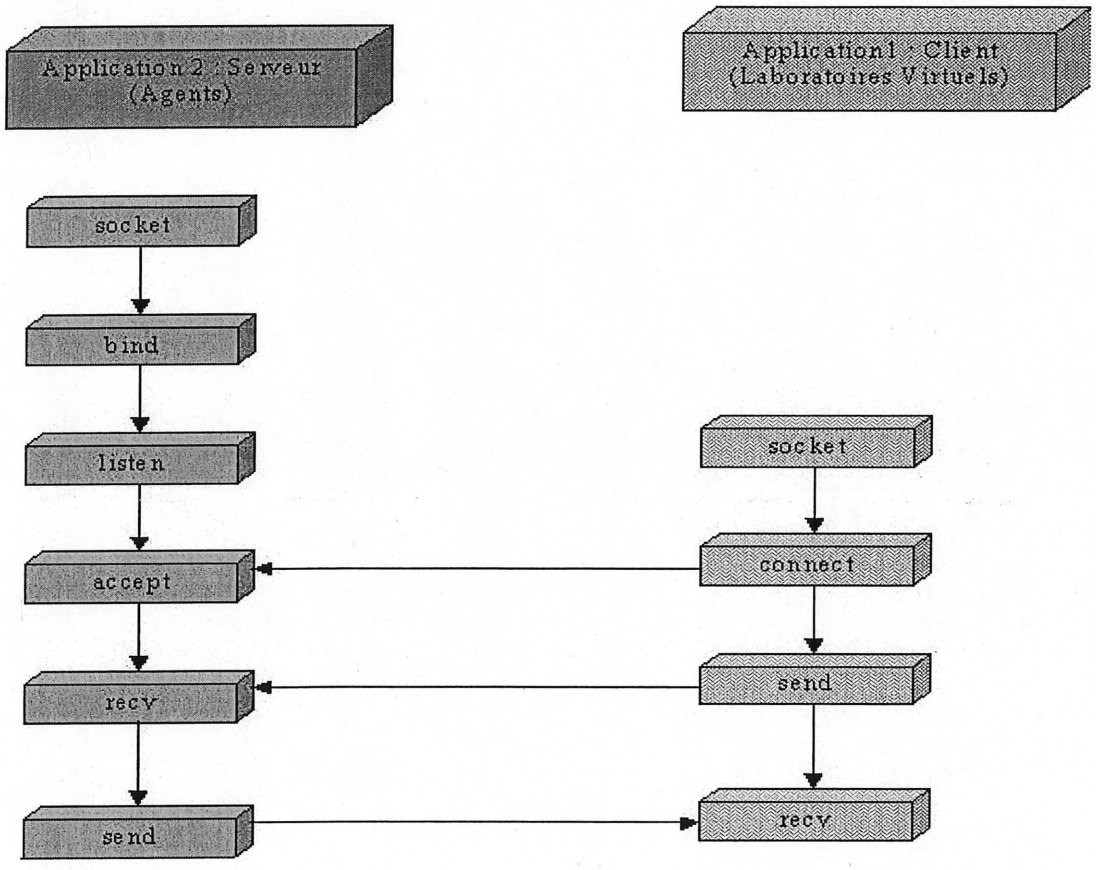

FIG. 33 – *Architecture du module de communication*

L'objet primitif « socket » est le point d'ancrage qui permet à l'application d'obtenir un lien de communication vers la suite du protocole qui servira d'échange et définit le mode de communication utilisé (mode connecté dans notre cas). La méthode « bind » permet

de spécifier le point de terminaison local (essentiellement le port TCP dans l'environnement TCP/IP). La méthode « connect » quant à elle permet à un client d'établir une communication active avec le serveur, le point de terminaison distant (adresse IP + port TCP dans l'environnement TCP/IP) est spécifié lors de cet appel. La méthode « listen » permet à un serveur d'entrer dans un mode d'écoute de communication, dès lors le serveur est « connectable » par un client, le processus est bloqué jusqu'à l'arrivée d'une communication entrante. La méthode « accept » permet à un serveur de recevoir la communication entrante (client), crée un nouveau « socket » et retourne le descripteur associé à l'application. Le serveur utilise ce descripteur pour gérer la communication entrante, il utilise le descripteur de « socket » précédent pour traiter la prochaine communication à venir. Les méthodes « recv » et « send » permettent au client et serveur d'échanger des données afin d'obtenir (client) et transmettre (serveur) le service désiré.

À première vue, il est normal de se demander l'importance ou plus précisément la particularité de cette couche de communication par rapport aux technologies de distribution d'applications. Il est certes vrai que Corba, RMI et DCOM ont déjà fait leurs preuves sur le terrain et ont leurs avantages. Cependant, le système que nous proposons ici a lui aussi du potentiel et il répond à un besoin spécifique. Il constitue un système prêt à être utilisé sans aucune autre programmation. En d'autres termes, ce système fournit des méthodes qui seront utilisées par le développeur et est donc le système le plus facile et rapide à utiliser.

3.2 Modèle des cas d'utilisation

Un cas d'utilisation est une abstraction d'une partie du comportement du système. Le diagramme de la figure 34 suivante présente les cas d'utilisation du système. D'après cette figure, le système comprend quatre acteurs à savoir : l'apprenant, le laboratoire virtuel, la couche de communication et les agents. Un acteur peut exécuter un ou plusieurs cas

TAB. 1 – *Tableau récapitulatif des acteurs et des cas d'utilisation*

Acteurs	Cas d'utilisation
Apprenant	- Identification - Chargement de laboratoire - Manipulation
Laboratoire virtuel	- Collecte d'information sur les manipulations - Information d'agents - Composition de messages - Transfert d'information
Agents	- Chargement des connaissances du domaine - Chargement du profil de l'utilisateur - Assistance - Suivi
Couche de communication	Acteur secondaire

d'utilisation selon le nombre d'arcs partant de chaque acteur. Un acteur représente un rôle joué par une personne ou par toute entité externe qui agit sur le système. Il est à noter ici qu'il n'est pas toujours évident de déterminer les limites d'un système.

3.2.1 Scénario d'identification et de chargement du profil

À chaque nouvelle session de l'étudiant (apprenant) dans l'environnement de Cyberscience, l'étudiant doit s'identifier au système. D'après la figure 35, l'apprenant lance l'application, le système affiche la fenêtre de connexion, puis il entre son nom d'utilisateur, son mot de passe et le système effectue une vérification en interrogeant la BD. Après vérification, le système présente la librairie de cours auquel il est inscrit, informe les agents de l'identité de l'apprenant qui soumet une requête à la base de données afin de charger les informations de l'étudiant.

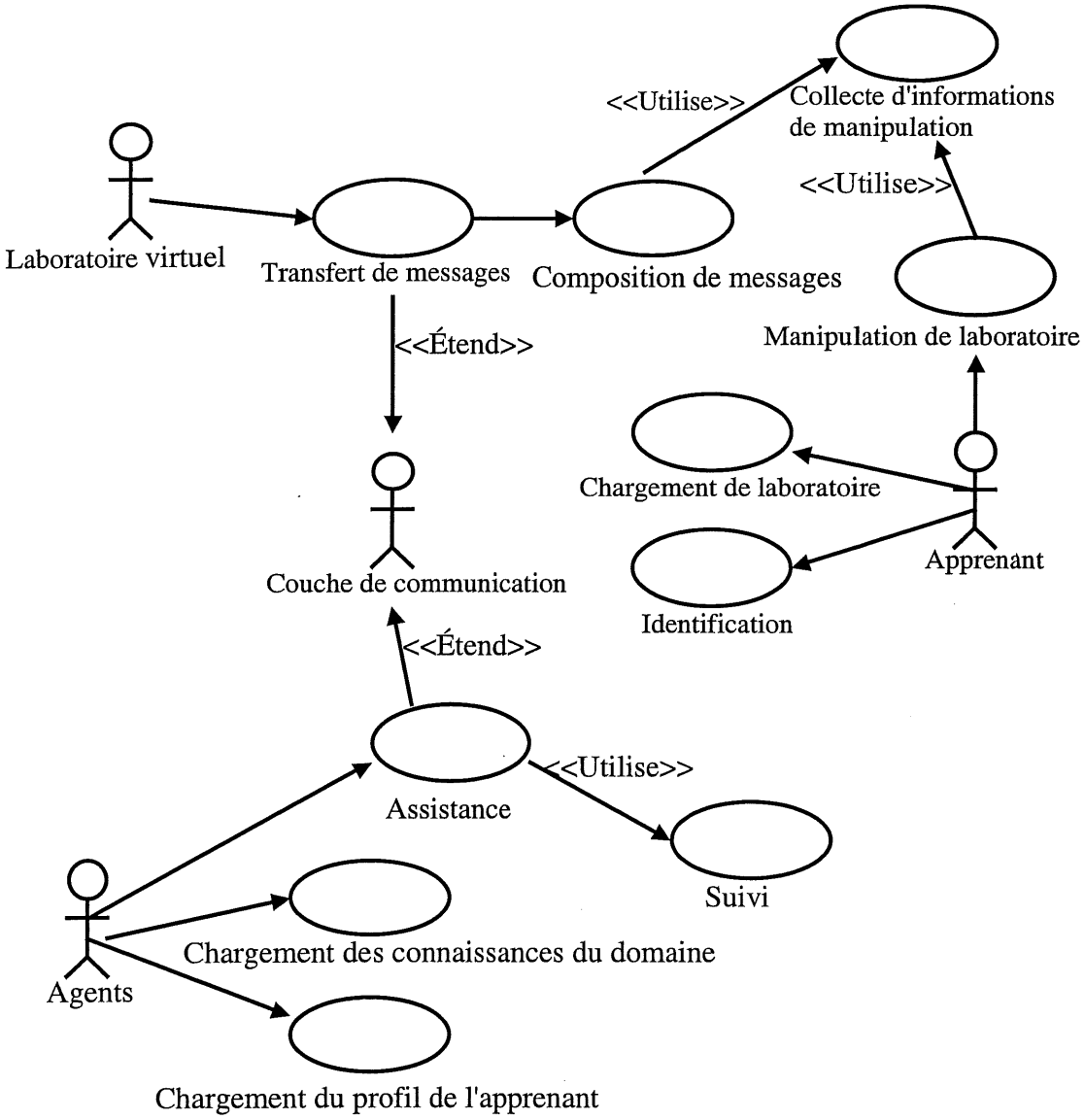

FIG. 34 – *Diagramme des cas d'utilisation du système*

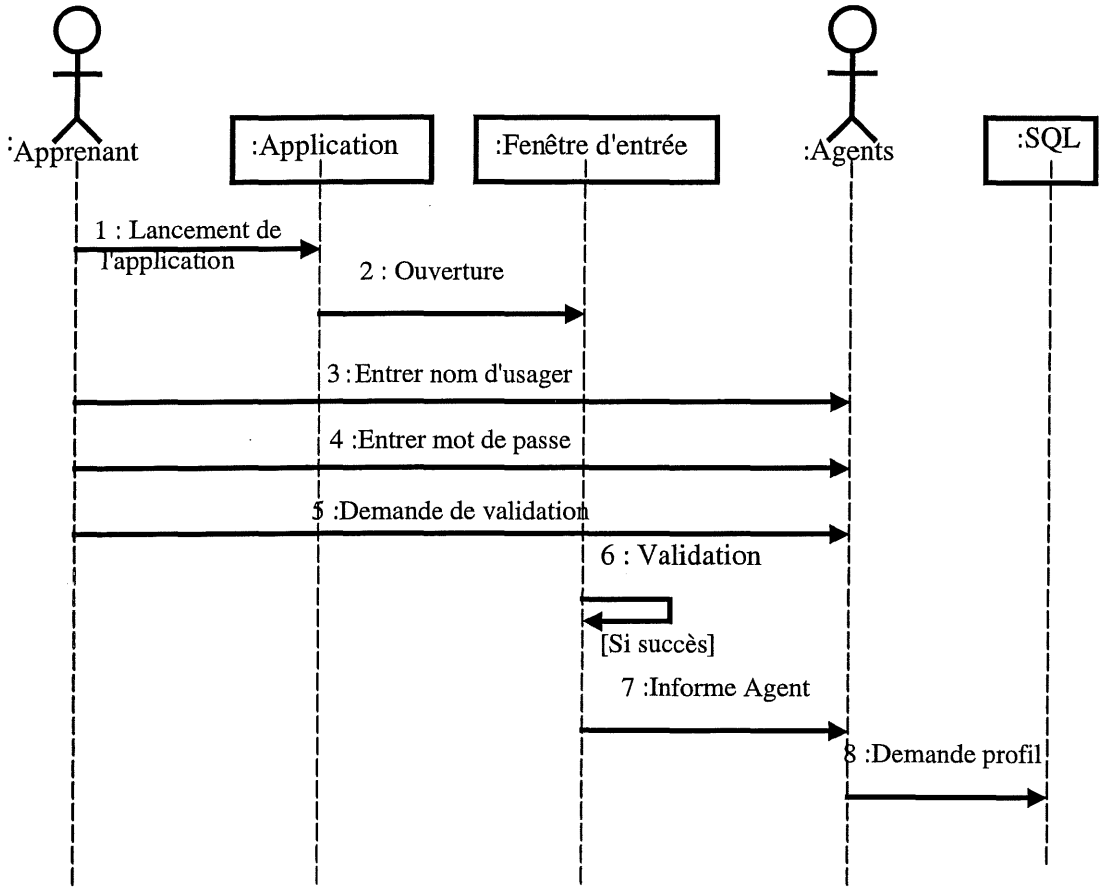

FIG. 35 – *Diagramme de chargement de laboratoires et du profil de l'apprenant*

3.2.2 Scénario de chargement de laboratoires et des connaissances

Après s'être correctement identifié, l'apprenant se voit présenter la bibliothèque des cours auxquels il est inscrit. Il choisit un cours, suivi du chapitre sur lequel il veut travailler, ensuite la section, l'étape puis enfin le laboratoire. Une fois le choix du laboratoire fait, il s'identifie auprès des agents afin que ces derniers puissent charger, de la base de données, les connaissances liées au domaine concerné (voir figure 36).

3.2.3 Manipulation, information et assistance

D'après la figure 37 ci-dessous, une fois le laboratoire chargé, l'étudiant choisit son activité. Une fois l'activité choisie, le laboratoire présente l'énoncé du problème à résoudre et confectionne un message contenant l'énoncé du problème à résoudre par l'apprenant, l'expédie aux agents afin qu'ils puissent y trouver une solution. Par la suite, l'étudiant pose des actions dans le but de résoudre le problème. Une fois l'action posée, le laboratoire utilise la couche de communication pour confectionner un message qui est envoyé aux agents qui l'examinent dans le but de vérifier le raisonnement de l'apprenant. Si cela s'avère nécessaire, les agents construisent un message pour prodiguer des conseils ou poser des questions à l'étudiant via la couche de communication. Le message est par la suite transféré au laboratoire qui le présente à l'étudiant.

3.2.4 Vue d'ensemble du modèle à l'aide d'un exemple

Dans cette section, nous utilisons un exemple pris dans le contexte de Cyberscience afin de montrer les relations existant entre les composants de ce système. Nous utilisons un exemple tiré du laboratoire virtuel de génie génétique.

Soit X, un étudiant ou une étudiante ;

M une molécule d'ADN et E1, E2... , En des enzymes.

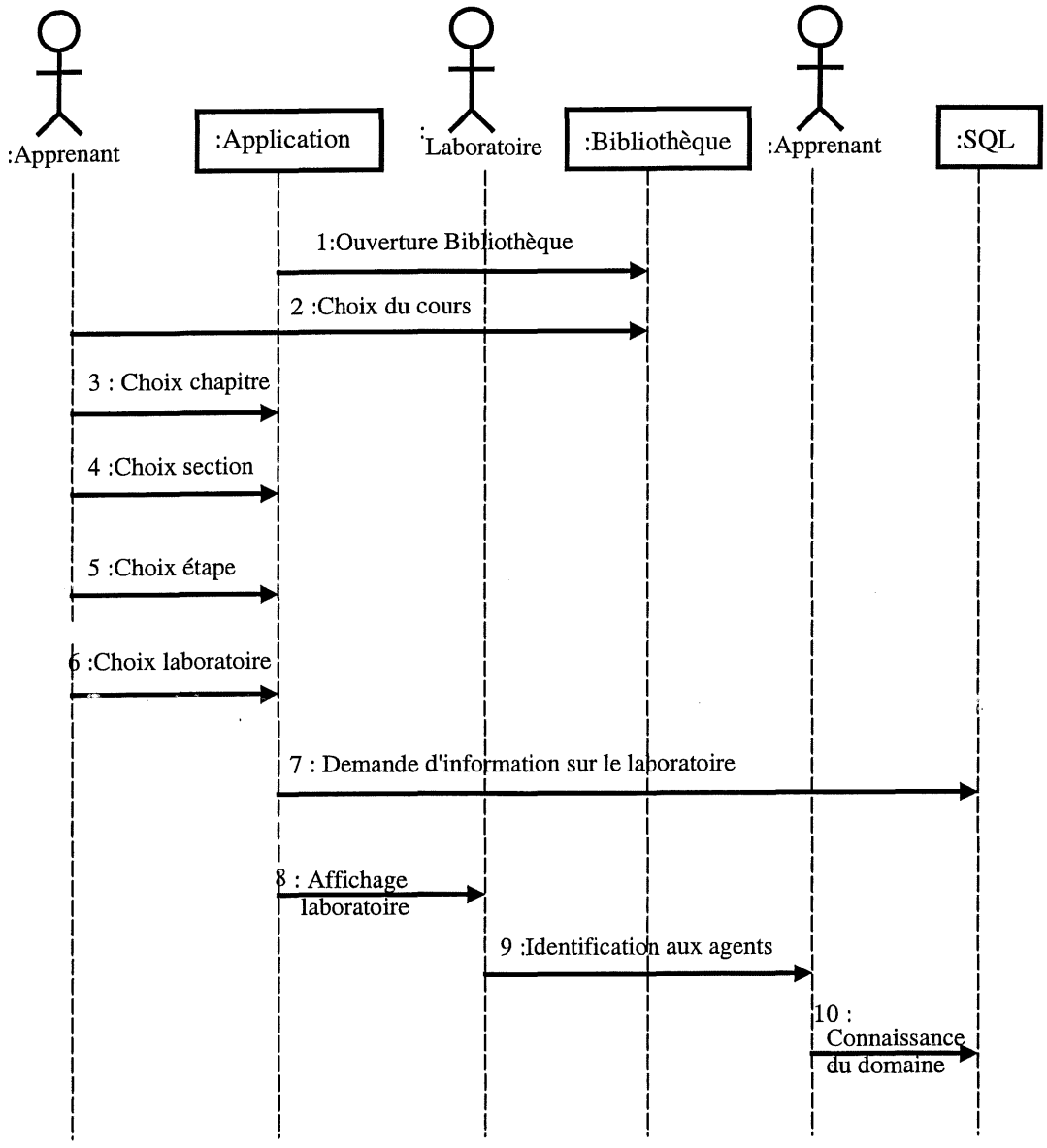

FIG. 36 – *Diagramme de chargement de laboratoires et des connaissances*

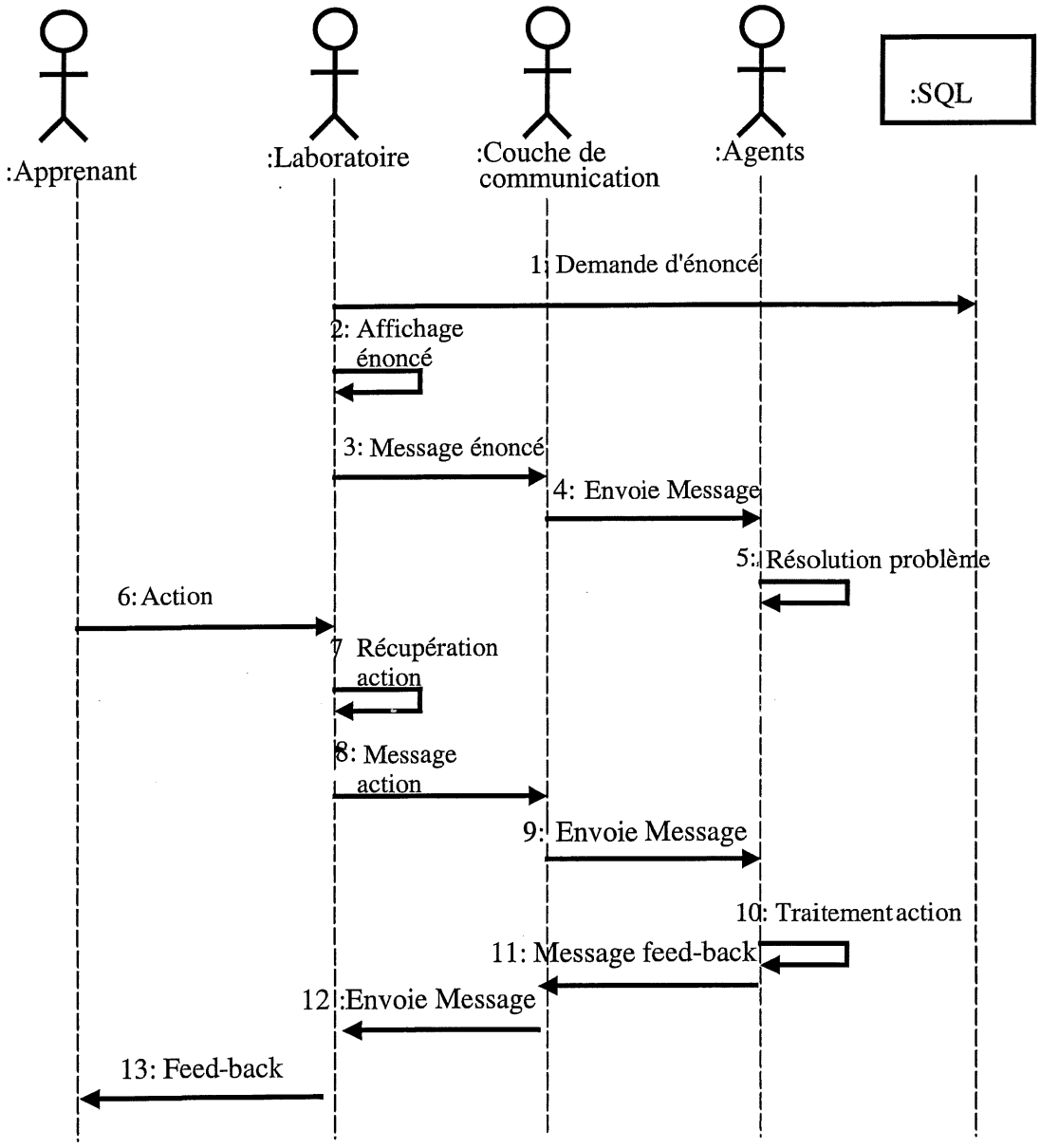

FIG. 37 – *Diagramme de manipulation, d'information et de suivi*

X désire effectuer une digestion de M avec E1, E2, ..., En. Pour ce faire, il devra procéder de la façon suivante :

1. X se connecte à l'environnement Cyberscience ;
2. X charge le laboratoire virtuel de génie génétique ;
3. X choisit ou se voit présenter un problème avec les données M, E1, E2, ..., En ;
4. X effectue une modification dans le laboratoire virtuel, par exemple, X demande une digestion de M avec E1 et E2 ;
5. Retour à l'étape 4 jusqu'à complétion de la tâche ou/et atteinte de son objectif.

Nous pouvons regarder à présent comment le système réagit face aux différentes actions de X (voir les figures 38 et 39). Après la connexion de l'étudiant X dans l'environnement de Cyberscience, les agents sont mis au courant de l'environnement de l'étudiant afin qu'ils puissent se procurer le modèle étudiant de X de la base de données, c'est-à-dire les connaissances antérieures de X. Par la suite, X demande le chargement d'un laboratoire après avoir choisi le cours. À la suite de cette action, un message de type « Information » est transmis aux agents afin qu'ils puissent prendre conscience du domaine dans lequel X s'apprête à travailler et qu'ils chargent les connaissances appropriées du dit domaine (« 1 » dans les figures 38 et 39). Dans le laboratoire nous servant d'exemple ici, s'il s'agit de la première entrée de X, il devra choisir le type de molécule avec lequel il aimerait travailler sinon il choisit une nouvelle molécule ou une molécule sur laquelle il travaille déjà. Suite au choix de X, le système lui présentera les informations relatives à la molécule afin qu'il puisse procéder aux différentes digestions qui modifieront l'environnement du laboratoire (« 2 » et « 3 » pour la présentation des données du problème et la modification de l'environnement). Un message est envoyé aux agents afin qu'ils prennent connaissance du problème de l'étudiant ainsi ils pourront mieux le suivre grâce aux messages qu'ils recevront à la suite des manipulations de X dans son environnement de travail. Ils peuvent aussi envoyer des messages soit pour modifier l'environnement de l'étudiant dans le cas d'une démonstration, soit pour fournir de l'information à l'étudiant concernant ses

manipulations en cas d'erreur (« 9 » dans les figures 38 et 39). Tous ces messages sont construits avec l'objet *message* qui est une instance de la classe *Message* de la couche de communication. Cet objet doit être créé par chacune des applications car c'est lui qui bénéficie de la capacité d'accéder et de générer les messages en format XML en utilisant les classes de la librairie de *Messages* et, d'acheminer ces derniers via la librairie de *Communication* qui contient les différents « sockets » de communication (respectivement « 4 », « 7 » et « 5 », « 6 » des figures 38 et 39). À la réception d'un message, ce même objet *message* utilise une fois de plus la librairie de *Messages* pour décoder le message XML et fait donc appel à la méthode de traitement des messages qui est l'implémentation de l'interface « Reception » de la couche de communication (« 8 » et « 10 » des figures 38 et 39).

La figure 38 présente l'initialisation et l'utilisation de la couche de communication dans le cas où le client et le serveur tournent sur une même machine. En ce qui concerne la figure 39, elle présente l'initialisation et l'utilisation de cette couche dans le cas où le client et le serveur fonctionnent sur des machines distantes. Dans ce cas, chacune des machines se doit d'avoir une copie identique de la couche de communication. En effet, les laboratoires virtuels et les agents pouvant respectivement jouer le rôle de client et de serveur, la librairie de communication contient les classes du client et celles du serveur. Cependant, il est possible de supprimer de cette librairie les classes spécifiques au client ou au serveur selon le cas si nous désirons établir une distintion entre le contenu du module client et celui du module serveur. Il est à noter que dans cette deuxième figure, il y a une connexion qui s'établie entre les deux machines à travers la couche de communication (« 11 » de la figure 39).

À présent que nous avons pris connaissance de l'architecture de notre système, nous allons aborder dans le chapitre suivant l'implémentation de cette dernière.

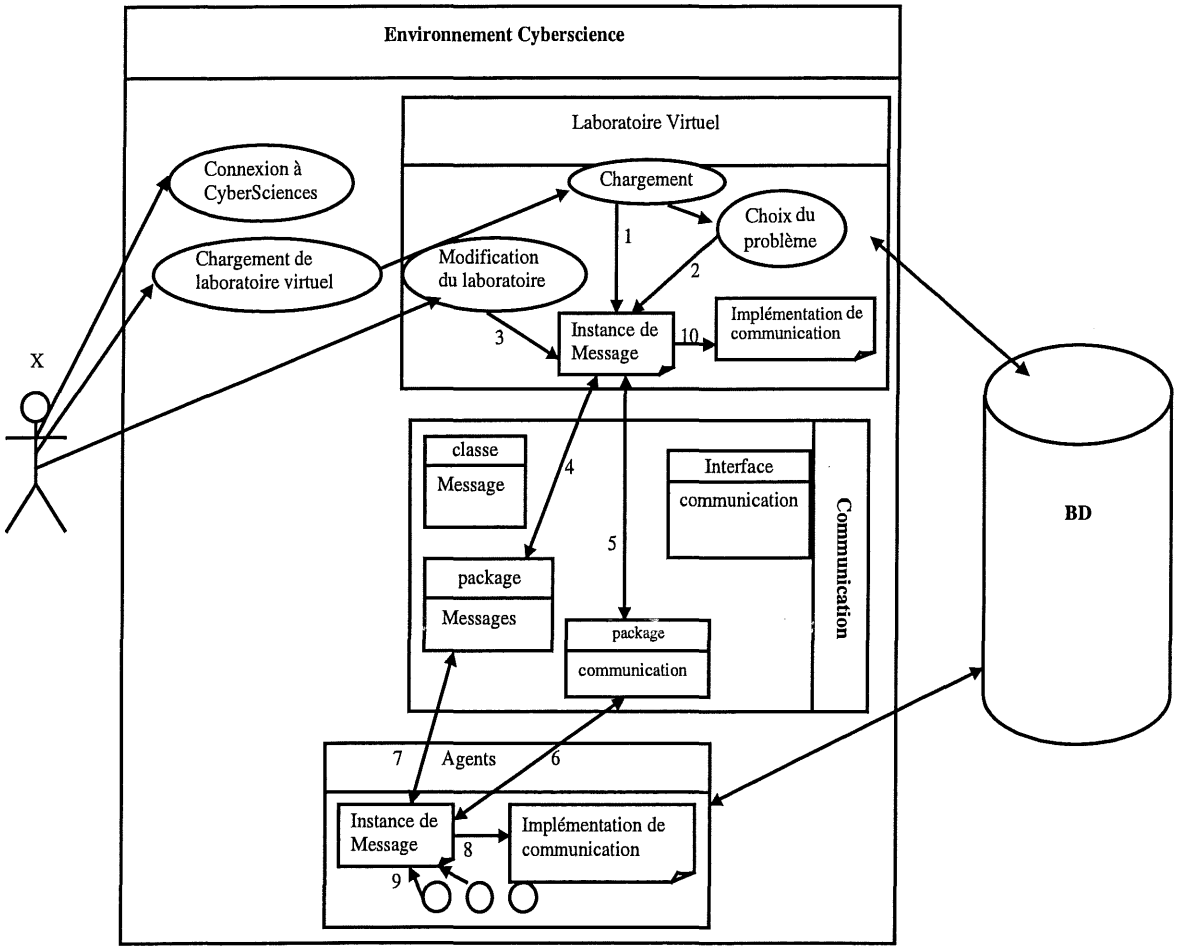

FIG. 38 – *Interaction entre les différents composants de la couche de communication sur machine unique*

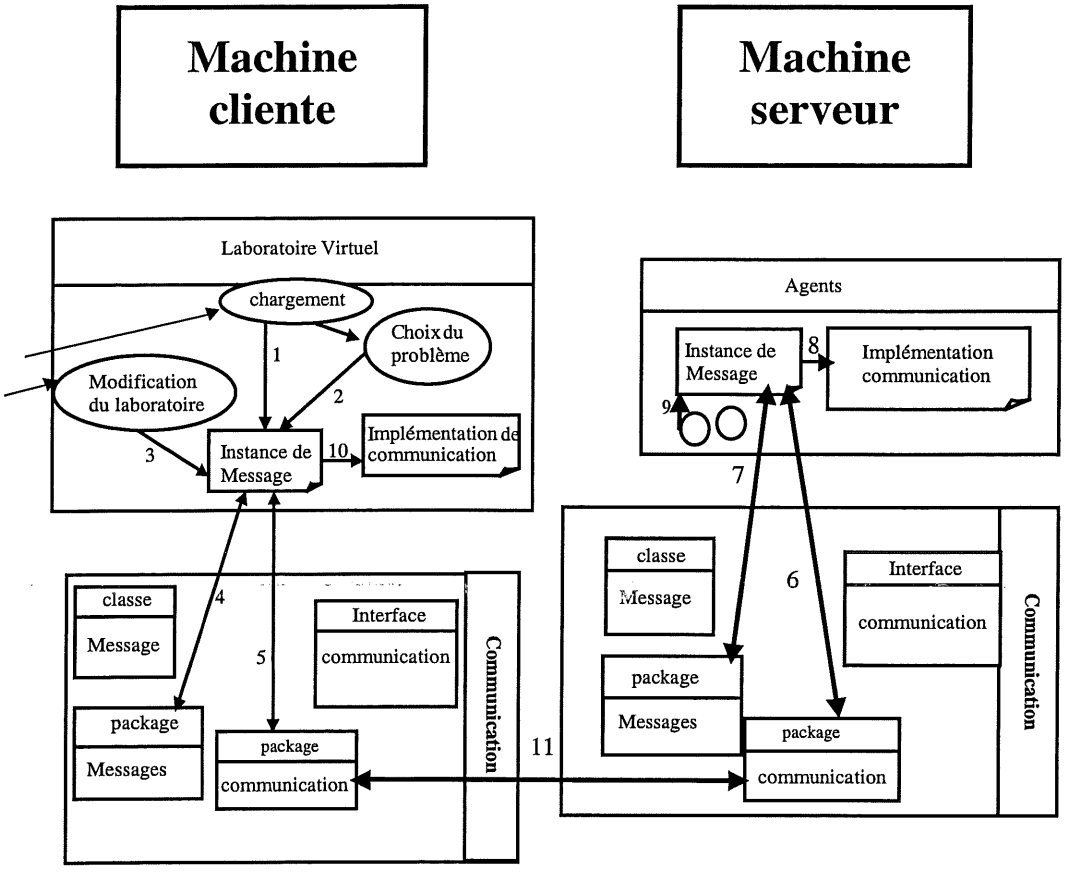

FIG. 39 – *Interaction entre les différents composants de la couche de communication sur machines distantes*

Chapitre 4

Réalisation

Dans le chapitre précédent, nous avons présenté l'architecture de notre couche de communication. Dans ce chapitre, nous présentons sa réalisation.

4.1 Moyens et logiciels utilisés

4.1.1 Le langage de programmation Java

Nous avons utilisé le langage Java qui est un langage de programmation orienté objet et qui s'exécute quels que soient l'architecture, le processeur et le système d'exploitation. En d'autres termes, toute application développée en Java est portable et, c'est cette portabilité qui a motivé le choix de ce langage. Une des caractéristiques principales de ce langage est que le code Java sera compilé par le compilateur Java pour être interpreté par une machine virtuelle : le code machine résultant est nommé ByteCode (code octet). Lors de l'exécution, le code octet produit sera transformé en un code machine compréhensible par le microprocesseur utilisé, ce qui explique la portabilité de ce langage.

4.1.2 Langage XML

XML est un langage de balises extensible qui peut être qualifié de méta-langage car, il permet de définir de nouvelles balises pour décrire la présentation d'un texte. Ce langage est particulièrement adapté pour l'échange de données de documents. La force de XML réside dans sa capacité à pouvoir décrire n'importe quel domaine de données grâce à son extensibilité. Il permet de structurer et de définir le vocabulaire et la syntaxe des données. Un document suivant les règles de XML est appelé « document bien formé ». Un document XML possédant une DTD et étant conforme à celle-ci est appelé « document valide ». Le choix de XML a été motivé par le potentiel qu'il nous fournissait quant à la structuration des données à échanger, sa portabilité, sa popularité et sa facilité d'utilisation.

4.1.3 JAXB

Jaxb est une API qui traduit les classes Java, c'est-à-dire le code de structuration des données manipulées par les composants, dans des formats de description XML équivalents et inversement. Cette traduction se fait grâce à des DTD ou des Schémas XML.

4.1.4 Plate-forme

Nous avons travaillé dans les environnements Windows 98, Windows NT et Linux. Windows NT était le cadre expérimental de ce travail. Cependant, l'application étant développée en Java, les clients travaillant sur toute autre plate forme pourront exploiter et implémenter cette API.

4.2 Description de la librairie de messages

Tel que nous l'avons déjà vu, ce travail a pour but fournir un outil d'échange de données et d'informations entre les agents intelligents et les laboratoires virtuels. Bien que nous utilisons le modèle client/serveur, le module le plus important est celui des messages que nous présentons dans cette section.

La librairie de messages est constituée des classes Java générées à l'aide de l'API JAXB de Sun. Cette dernière utilise un fichier de spécification et un fichier de correspondance. JAXB est une API qui simplifie la création d'applications Java pouvant lire, manipuler et recréer des documents XML contenant les données manipulées par les composants. Donc en traduisant les objets Java en format XML et vice-versa, son compilateur génère automatiquement des classes Java à partir de schémas XML sans que le développeur ait à écrire des programmes complexes de « passage ». De plus, le compilateur gère des messages d'erreurs et de vérification de la validité des documents XML permettant de s'assurer qu'uniquement des messages valides sans erreur seront acceptés et traités par un système. Le fichier de spécification est en fait une DTD (annexe A.1) qui contient l'ontologie de communication, soit le vocabulaire que devront utiliser les composantes pour s'échanger de l'information. Le fichier de schéma de conversion est un fichier de format « xjs » qui contient les informations de typage et de correspondance du contenu du DTD (annexe A.2). Un message comprend une catégorie et le message proprement dit (tableau 2 et l'annexe A.1).

Nous avons prévu dans notre ontologie cinq types ou catégories de message :

· « *Identification* » - cette catégorie est utilisée pour informer les agents de l'identité de l'apprenant ou du laboratoire virtuel afin que ces derniers puissent charger le bon profil ou le bon domaine de connaissance nécessaire au suivi des activités de l'utilisateur ;

· « *Questions* » - cette catégorie est utilisée pour spécifier soit une question de l'apprenant aux agents, soit une question des agents à l'apprenant ;

· « *Demo* » - cette catégorie est utilisée soit pour spécifier au serveur qu'il s'agit d'une démonstration faite par l'usager ou encore pour spécifier au laboratoire qu'il s'agit d'une démonstration en provenance des agents ;

· « *Reponses* » - cette catégorie spécifie tout simplement la réponse de l'apprenant à une question qui lui a été posée ;

· « *Information* » - enfin, la catégorie d'information est utilisée soit par les laboratoires pour informer les agents d'un changement d'état ou d'une manipulation dans l'environnement par l'usager, soit par les agents pour informer l'usager d'un fait ou encore attirer son attention. Cette catégorie est utilisée lorsqu'on ne désire pas avoir une réaction de l'autre partie.

En ce qui concerne le message en soi, il peut contenir soit un contenu, soit une manipulation. Nous faisons une différence ici entre contenu et manipulation car le contenu s'applique uniquement aux catégories ne pouvant contenir qu'une seule instruction (par exemple : l'« *Identification* », la « *Réponse* » et la « *Question* »). Quant à la manipulation, elle réfère aux catégories pouvant avoir plus d'une itération donc une ou plusieurs instructions (tel est le cas de « *Demo* » et « *Information* »). Ici, la catégorie est une chaîne de caractères alors que la manipulation est soit une action, soit une démonstration. Une démonstration (« *Demo* ») est constituée de plusieurs étapes. On distingue six classes d'actions possibles : « *Tourner* » (qui consiste à tourner un objet dans le laboratoire virtuel), « *Deplacer* » (qui consiste à déplacer un objet dans le laboratoire virtuel), « *Tracer* » (qui consiste à tracer une fonction ou une ligne dans le laboratoire virtuel), « *Joindre* » (qui consiste à joindre deux ou plusieurs objets dans le laboratoire virtuel), « *Digerer* » (qui consiste à faire des digestions simples et doubles sur une molécule d'ADN dans le laboratoire virtuel) et « *Inserer* » (qui consiste à inserer un objet dans le laboratoire virtuel).

TAB. 2 – *Tableau récapitulatif de la composition d'un message*

Éléments	Composants
Messages	Categorie, message
ICategorie	Question | Reponse | Information | Identification | Demo
Message	contenu | manipulation
Manipulation	Action | demo
Action	Tourner | Tracer | Inserer | Deplacer | Digerer | Joindre
Demo	etape+
Tracer	fonction
Tourner	objet, angle
Deplacer	objet, depart, arrivee
Inserer	Donnees, emplacement
Joindre	pointA, pointB
Digerer	type+
Type	simple | doubles
Simple	Enzyme
Doubles	enzyme1, enzyme2

4.3 Description de la librairie de communication

La librairie de communication est le module responsable du transfert de données et d'informations des agents aux laboratoires virtuels et vice-versa. Ce module est constitué de trois librairies à savoir : les librairies d'événements, d'exceptions et de filtres.

4.3.1 Librairie d'événements

La librairie est constituée de l'interface d'écoute du serveur, de l'interface d'écoute du client, de la classe d'événement de serveur et de la classe d'événement du client.

L'interface d'écoute du serveur

Cette interface permet d'écouter toutes les informations qui sont transférées sur le réseau à travers le serveur. Toutes les classes qui l'implémentent recevront tous les messages

destinés au serveur. Ces classes devront implémenter la méthode « *server_Event* » qui prend comme paramètre un objet de type « *Server_Event* » correspondant à une instance de la classe « *Server_Event* » qui est décrite plus bas et qui contient le message ainsi que son expéditeur.

L'interface d'écoute du client

En ce qui concerne l'interface d'écoute du client, toutes les classes qui l'utilisent implémentent la méthode « *network_Event* » qui a comme paramètre un objet dérivant de l'instanciation de la classe « *Network_Event* » et contenant le message en provenance du serveur. Donc tous les messages en provenance du serveur arrivent dans cette méthode.

La classe d'événements du serveur

La classe d'événements du serveur « *Server_Event* » prend en entrée deux paramètres à savoir une chaîne de caractères et un objet de type « *Com_Socket* » correspondant respectivement au message reçu et au client expéditeur. L'événement de serveur est lancé par l'agent qui ici joue le rôle du serveur lorsqu'il reçoit un nouveau message du client donc du laboratoire virtuel.

La classe d'événements du client

La classe d'événements du client « *Network_Event* » prend un objet en paramètre d'entrée permettant la création d'une nouvelle instance de la classe qui sera utilisée par l'application. L'événement de client est lancé par le laboratoire afin de pouvoir recevoir et gérer l'objet en provenance des agents.

4.3.2 Librairie d'exceptions

Cette librairie contient deux classes permettant de gérer les erreurs et les exceptions du coté serveur et du coté client. Il s'agit des classes « *ComServerException* » pour le serveur et « *ComSocketException* » pour le client. Ces deux classes prennent en entrée deux paramètres qui permettent l'initialisation et la création d'une nouvelle instance de la classe ou de l'exception. Ces paramètres sont une chaîne de caractères correspondant au message émis soit par le client, soit par le serveur, et un objet correspondant au lanceur soit l'objet « *ComServer* » pour le serveur ou « *Socket* » pour le client.

4.3.3 Librairie de filtres

Cette librairie contient deux interfaces et une classe. L'interface « *ComFilter* » qui définit deux méthodes à implémenter à savoir « *FilterIn(String s)* » et « *FilterOut(Object o)* » correspondant respectivement à la transformation d'une chaîne de caractères en un objet et d'un objet en une chaîne de caractères ceci afin de se conformer au format de données à utiliser soit dans l'application soit dans le transfert de données. Quant à l'interface « *ComListFilter* », elle définit deux méthodes « *filterStringIn(String s)* » et « *filterStringOut(String s)* » qui pourront permettre d'apporter quelques modifications telles que le cryptage d'une chaîne de caractères. La classe « *DefaultComFilter* » est l'implémentation par défaut de l'interface « *ComFilter* » donc un exemple de l'implémentation de cette interface. Elle permet tout simplement d'envoyer la description d'un objet en appelant la méthode « *toString* » de la classe parent qui l'utilise.
À présent que nous avons une description détaillée des différentes librairies de notre API, nous allons présenter son implémentation dans la section suivante.

4.4 Exploitation de l'API dans une application

Pour implémenter cette API, le développeur se doit d'importer la bibliothèque « *apicom* ». Les annexes B.4 et B.5 présente un exemple d'intégration de cette API dans une application.

Une fois cette bibliothèque importée, il devra créer une instance des classes « *APICom* » et, « *APIClient* » pour le client (ou Laboratoire virtuel) ou « *APIServeur* » pour le serveur (ou Agents). Il est à noter que toute classe ou application utilisant cette API se doit d'implémenter l'interface de communication « *Reception* » qui lui permettra de recevoir les messages en provenance de son interlocuteur (voir les annexes B.1, B.2 et B.3 pour une coquille des classes « *APICom* », « *APIClient* » et « *APIServeur* »).

La classe « *APICom* » est la classe permettant de confectionner les messages qui seront acheminés au correspondant. Ces messages seront transformés en format « xml » et stockés dans un fichier nommé « Message.xml ». Une fois le message confectionné à l'aide de la classe « *APICom* », il est donc possible de l'expédier au destinataire à l'aide de la méthode « *send* » des classes « *APIServeur* » et « *APIClient* ». Ces deux dernières classes implémentent respectivement les interfaces « *NetworkListener* » et « *ServerListener* ». En d'autres termes, l'« *APIServeur* » implémente l'interface « *ServerListener* » et l'« *APIClient* » implémente l'interface « *NetworkListener* » ce qui permet à l'application serveur d'être à l'écoute de l'information en provenance des laboratoires ou des agents à travers le réseau et de traiter cette information à l'aide la classe « *APICom* » qui transforme le format « xml » en objet Java. Cependant il faut noter que l'utilisateur peut aussi vouloir travailler directement avec les données en format xml. Dans ce cas, il pourra récupérer l'information dans le fichier « messageR.xml ».

4.4.1 Exemple d'implémentation

L'exemple que nous avons utilisé pour tester le fonctionnement de cette API est une application permettant de générer un problème de calcul de la puissance ou de la factorielle d'un nombre. Dans cet exemple, l'étudiant choisit le type de fonction avec lequel il veut travailler. À cet instant, un message de type « *Identification* » est confectionné et expédié à l'agent. Sur réception du message, l'agent charge la classe permettant de résoudre les problèmes correspondant au type spécifié par l'usager. Ces classes de calcul de la factorielle et de la puissance représentent une simulation du type d'agent que devra implémenter CyberScience. En d'autres termes, ils correspondent à des connaissances spécifiques au domaine d'étude ou de travail en laboratoire. À chaque problème posé à l'utilisateur, un message est envoyé à l'agent avec les informations sur le problème à résoudre. Une fois le problème résolu par l'étudiant, sa réponse est soumise à l'agent qui le vérifie et retourne un message à l'usager l'informant de l'exactitude ou non de sa réponse. En cas de mauvaise réponse, l'apprenant peut demander à l'agent de lui faire une démonstration de la solution du problème. L'agent lui retourne toutes les étapes de la résolution du problème et demande à l'étudiant de procéder de la même façon pour résoudre un autre problème qu'il lui propose. Les figures 40, 41, 42, 43, 44 présentent les différents écrans illustrant cette démarche.

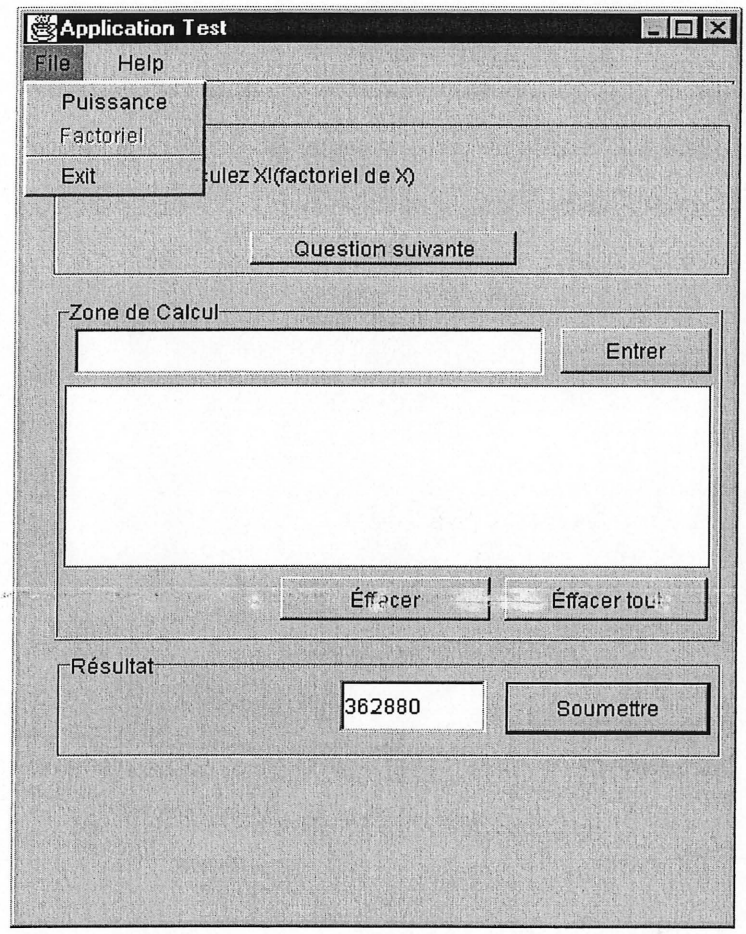

FIG. 40 – *Écran du laboratoire de calcul de la puissance et de la factorielle*

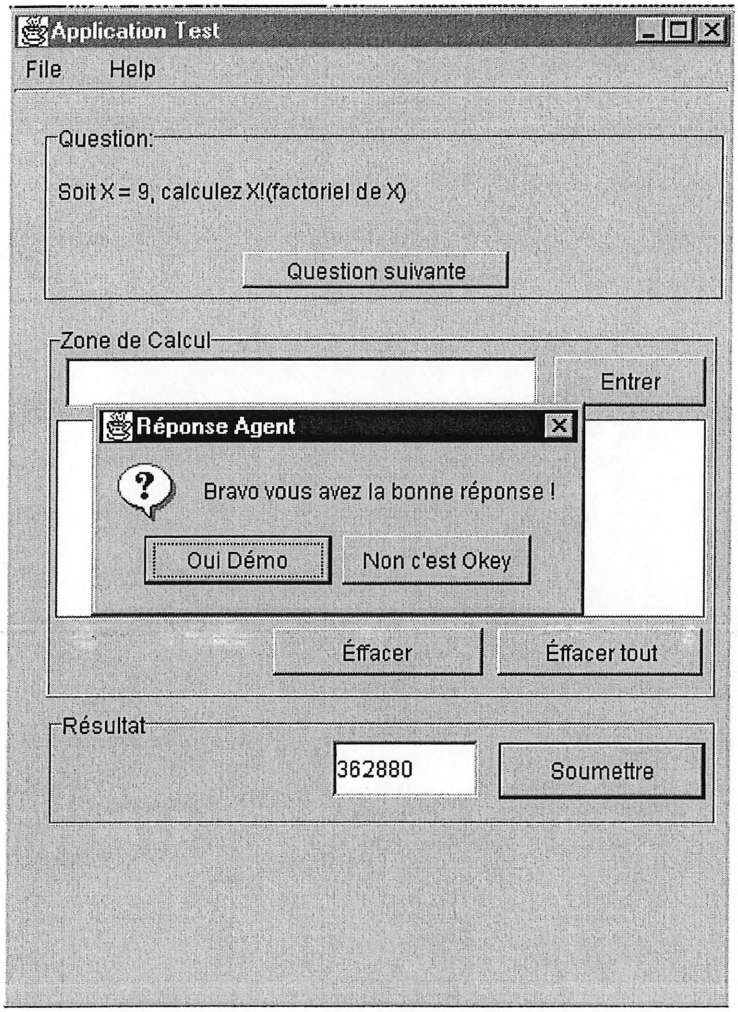

FIG. 41 – *Écran de laboratoire avec feed-back positif de l'agent*

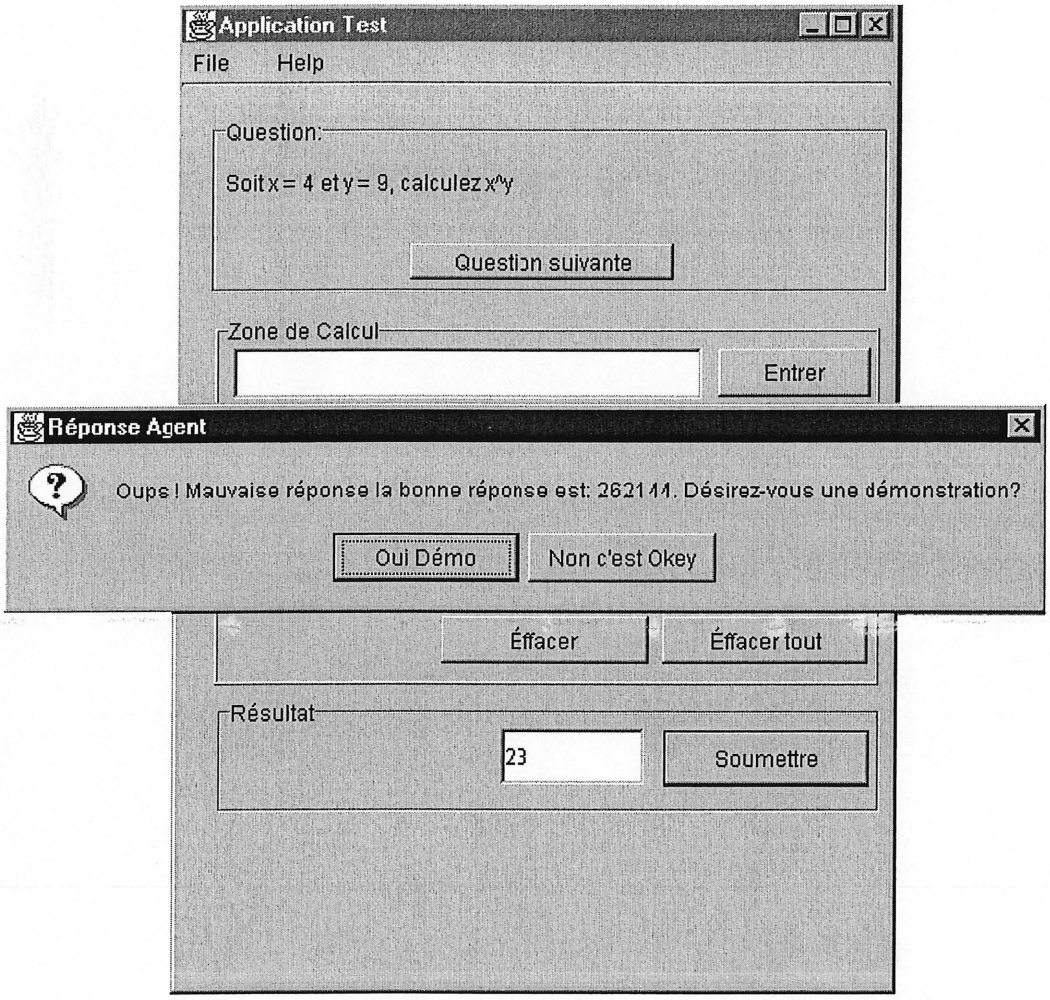

FIG. 42 – *Écran de laboratoire avec feed-back négatif de l'agent*

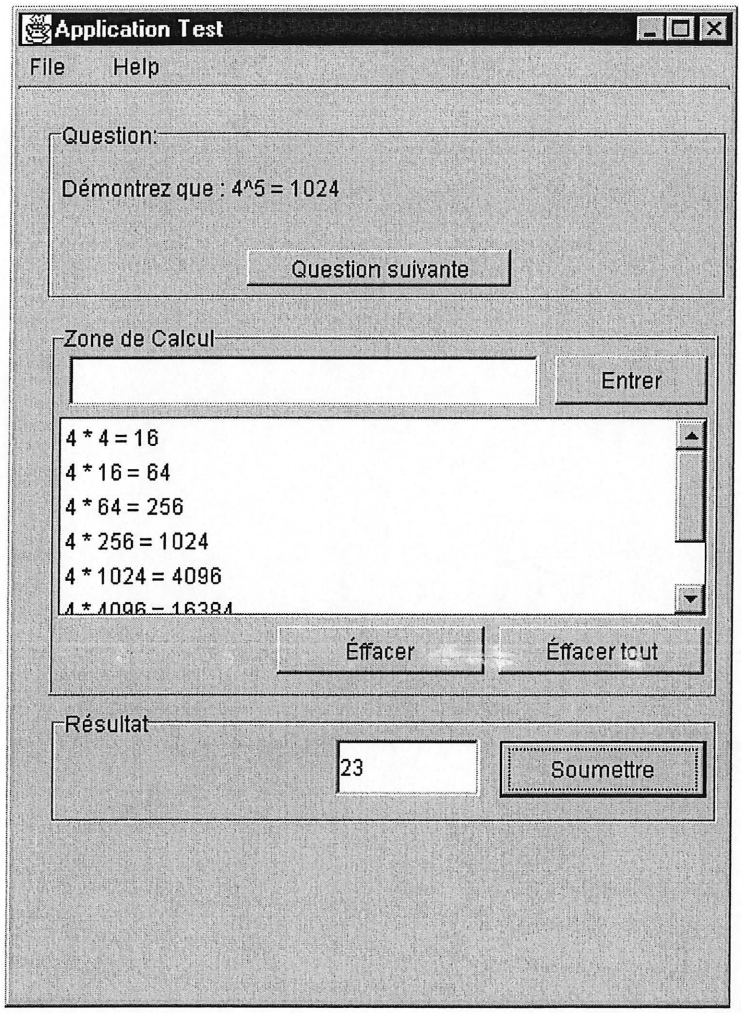

FIG. 43 – *Réaction de l'agent face à la demande de démonstration de son résultat par l'usager*

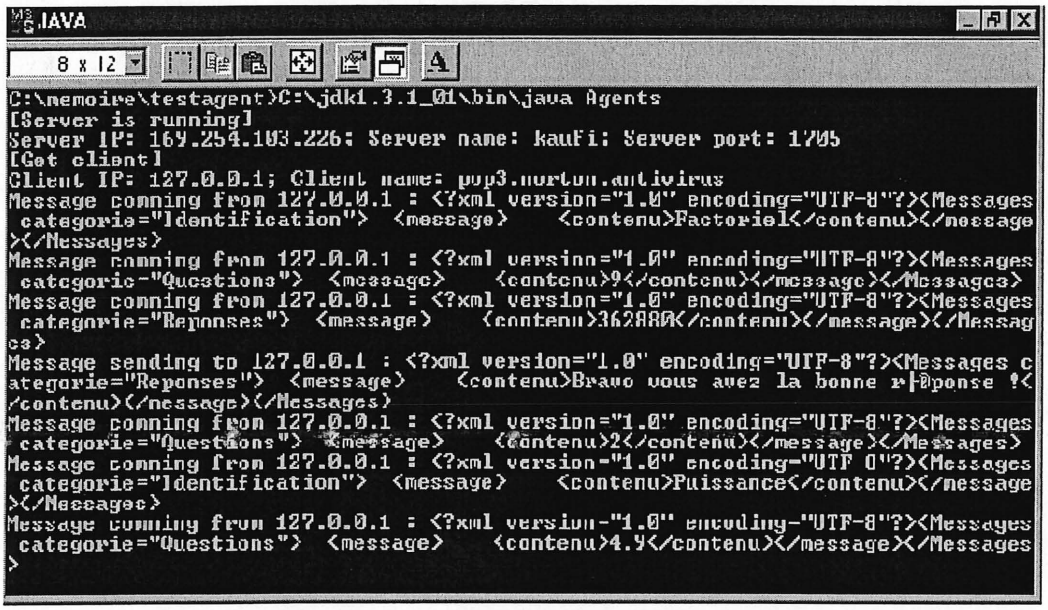

FIG. 44 – *Aperçu des activités du serveur (messages reçus et émis par l'agent)*

Conclusion

Dans ce mémoire, nous avons présenté une architecture de communication et de messagerie entre les laboratoires et les agents intelligents. Nous avons aussi présenté l'architecture et l'implémentation d'une API permettant l'intégration de ces outils dans une application d'aide à l'enseignement de STI. L'architecture de communication est basée sur le modèle client/serveur avec canaux de communication. En ce qui concerne l'architecture de messagerie, elle fournit une ontologie commune de communication aux différents acteurs. Elle est obtenue à l'aide d'une API de Java JAXB qui permet de faire la conversion d'un objet Java en un document xml et vice-versa. Les applications implémentant cette API ont le choix de travailler avec les messages en format xml ou en objet Java.
Dans ce qui suit, nous présentons l'originalité de notre solution, les limites de notre travail et quelques idées de travaux futurs pouvant permettre d'améliorer le système.

Originalité

La contribution de ce travail réside en la création d'une API constituée d'un ensemble portable de classes en Java pour la création et gestion de messages en format xml et pour la communication entre les laboratoires virtuels et les agents intelligents dans une application de STI. La particularité de la librairie de messages relève du fait qu'elle est constituée dans un format portable et qu'elle est extensible grâce à une compilation de fichier de DTD et de schéma de conversion à l'aide de l'API JAXB. Ceci met donc en évidence la possibilité d'extension et d'adaptation de la librairie de messagerie afin de

pouvoir faciliter l'implémentation de ce système dans d'autres applications.

Limites

La pricipale limite de notre système réside dans l'incomplétude de l'ontologie de communication. En effet, l'ajout d'un nouveau laboratoire virtuel va exiger la modification de l'ontologie pour y inclure les nouvelles actions primitives. En conséquence, il faudra modifier manuellement la DTD afin de les incorporer et d'en spécifier les paramètres. Une fois cette modification effectuée, il faut recompiler les fichiers contenant la DTD et le shémas de conversion ; ce qui mettra automatiquement à jour les classes de la librairie de gestion des messages. De plus, cet outil n'a pas été testé suffisamment.

Travaux futurs

Dans cette section, nous présentons les travaux futurs envisagés. Il s'agit ici d'énoncer quelques idées de recherche qui pourront permettre de combler les lacunes de notre système. Ces travaux pourront consister à :

· offrir une interface qui permettra l'utilisation de ce système dans plusieurs types d'application et ce malgré le langage de programmation utilisé ;

· généraliser l'utilisation du système en enrichissant l'ontologie de manière à répertorier la plupart des actions communément utilisées dans les laboratoires virtuels ;

· effectuer des tests supplémentaires auprès des utilisateurs ;

· ajouter un moteur qui lui permettra d'être autonome pour capter les actions et les manipulations de l'usager dans le laboratoire virtuel et dans le micro monde puis pour confectionner un message qui sera achéminé aux agents.

Annexe A

Ontologie et schéma de conversion

A.1 Ontologie et schéma de conversion

<!ELEMENT Messages (message) >

<!ATTLIST Messages categorie (Questions | Reponses | Information | Identification | Demo) #REQUIRED >

<!ELEMENT message (contenu | manipulation) >

<!ELEMENT manipulation (Actions | demo) >

<!ELEMENT demo (etape+) >

<!ELEMENT Actions (Tourner | Tracer | Inserer | Deplacer | Digerer | Joindre) >

<!ELEMENT etape (#PCDATA) >

<!ELEMENT Tracer (fonction) >

<!ELEMENT fonction (#PCDATA) >

<!ELEMENT Inserer (Donnees,emplacement) >

<!ELEMENT emplacement (#PCDATA) >

<!ELEMENT Deplacer (objet,depart,arrivee) >

<!ELEMENT objet (#PCDATA) >

<!ELEMENT depart (#PCDATA) >

<!ELEMENT arrivee (#PCDATA) >

<!ELEMENT Tourner (objet,angle) >

<!ELEMENT angle (#PCDATA) >

<!ELEMENT Digerer (simple | doubles)+ >

<!ELEMENT simple (enzyme_1) >

<!ELEMENT doubles (enzyme_1, enzyme_2) >

<!ELEMENT enzyme_1 (#PCDATA) >

<!ELEMENT enzyme_2 (#PCDATA) >

<!ELEMENT Joindre (pointA,pointB) >

<!ELEMENT pointA (#PCDATA) >

<!ELEMENT pointB (#PCDATA) >

<!ELEMENT contenu (#PCDATA) >

<!ELEMENT Questions (#PCDATA) >

<!ELEMENT Reponses (#PCDATA) >

<!ELEMENT Donnees (#PCDATA) >

<!ELEMENT Information (#PCDATA) >

<!ELEMENT Identification (#PCDATA) >

<!ELEMENT Demo (#PCDATA) >

A.2 Schéma de conversion de la DTD

<?xml version="1.0" encoding="ISO-8859-1" ?>

<xml-java-binding-schema version="1.0ea">

<options package="Messages" />

<element name="Messages" type="class" root="true">

<attribute name="categorie" convert="categorieTypes"/>

</element>

```
<element name="message" type="class" />
<element name="manipulation" type="class" />
<element name="Actions" type="class" />
<element name="demo" type="class" />
<element name="Tourner" type="class"/>
<element name="Tracer" type="class"/>
<element name="Inserer" type="class"/>
<element name="Deplacer" type="class"/>
<element name="Digerer" type="class"/>
<element name="Joindre" type="class"/>
<element name="angle" type="value" convert="BigDecimal"/>
<conversion name="BigDecimal" type="java.math.BigDecimal"/>
<enumeration name="categorieTypes" members="Questions Reponses Information
Identification Demo"/>
</xml-java-binding-schema>
```

Annexe B

Code d'implémentation

B.1 Coquille de l'APICom

```
...public class APICom {
public APICom() {
}
public void setCategorie(String s){
}
public String getCategorie(){
}
public void setTracer(String s) throws Exception{
}
public Tracer getFonctionTracer(){
}
public void setDeplacer(String _objet, String _depart, String _arrivee){
}
public Deplacer getDeplacer() {
}
```

```
public void setTourner(String _objet, BigDecimal angle){
}
public Tourner getTourner(){
}
public void setInserer(String donnee, String emplacement){
}
public Inserer getInserer(){
}
public void setJoindre(String pointA, String pointB){
}
public Joindre getPointJoind(){
}
public void setContenu(String c){
}
public void setQuestion(String s) {
}
public String getQuestion(){
}
public void setReponses(String s){
}
public String getReponses(){
}
public void setIdentification(String s){
}
public String getIdentification(){
}
public void addEtapeDemo(String s) {
```

```
}
public List getEtape(){
}
public void addEnzymeSimple(String en){
}
public void addEnzymeDouble(String e1, String e2){
}
public Digerer getEnzyme(){
}
public void ConstructionArbre() throws Exception {
}
public void accesContenu() {
}
}
```

B.2 Coquille de l'APIClient

```
...
public class APIClient extends ComClient implements NetworkListener{
public APIClient() {
}
public APIClient(int port, String IP) {
}
public void networkEvent(NetworkEvent evt){
}
public void addReception(Reception r){
}
```

```
public void send(){
}
}
```

B.3 Coquille de l'APIServeur

```
...
public class APIServeur extends ComServer implements ServerListener {
public APIServeur() {
}
public APIServeur(int port, String nom) {
}
public APIServeur(int port, String nom, String MotPasse) {
}
public APIServeur(int port, String nom, String MotPasse, int MaxConnexion) {
}
public void serverEvent(ServerEvent evt){
}
public ComSocket getSender (){
}
public void addReception(Reception r){
}
public void send(){
}
}
}
```

B.4 Code source de l'exemple - Client (Laboratoire)

```
package testclients;
import apicom.*;
import apicom.APICom;
import java.util.List;
import java.util.ListIterator;
import java.util.Vector;
public class Clients extends JFrame implements Reception{
Puissance p;
Factoriel f;
Object obj;
APICom apicom1;
APIClient apiclient = new APIClient();
Vector demo;
String fonction=null;
public Clients() {
try {
apiclient.addReception(this);
jbInit();
}
catch(Exception e) {
}
}
private void jbInit() throws Exception {
}
private void Feed_Back(String fd){
```

```
}
public void puissance_actionPerformed(ActionEvent e) {
try {
...
p = new Puissance(apiclient);
apicom1=new APICom();
fonction = "Puissance";
apicom1.setCategorie("Identification");
apicom1.setIdentification(fonction);
apiclient.send();
apicom1=null;
...
}catch(Exception ex){}
}
public void factoriel_actionPerformed(ActionEvent e) {
try {
...
f = new Factoriel(apiclient);
apicom1=new APICom();
fonction = "Factoriel";
apicom1.setCategorie("Identification");
apicom1.setIdentification(fonction);
apiclient.send();
apicom1=null;
...
}catch (Exception ex){}
}
```

```
public void reception(APICom _apicom){
if (_apicom.getCategorie().compareTo("Reponses")== 0)
Feed_Back(_apicom.getReponses());
else if (_apicom.getCategorie().compareTo("Demo")== 0)
Demo(_apicom.getEtape());
else if (_apicom.getCategorie().compareTo("Questions")==0)
setQuestion(_apicom.getQuestion());
}
private void setQuestion(String question){
}
private void Demo(List dem){
}
}
```

B.5 Code source de l'exemple - Agents

```
package testagent;
import apicom.*;
import java.util.*;
public class Agents extends APIServeur implements Reception{
private PuissanceE puissance;
private FactorielE factoriel;
private long resultat;
long x,y;
String fonction;
Vector v;
APICom apicom1;
```

```
public Agents()
{
super.addReception(this);
}
public void reception(APICom _apicom)
{
apicom1=_apicom;
if (apicom1.getCategorie().compareTo("Questions")==0){
setQuestion(apicom1.getQuestion(),0);
}
else if (apicom1.getCategorie().compareTo("Reponses")==0){
checkAnswer(Long.parseLong(apicom1.getReponses()));
}
else if (apicom1.getCategorie().compareTo("Identification")==0){
fonction = apicom1.getIdentification();
}
if(fonction.compareTo("Puissance") == 0) {
puissance = new PuissanceE();
factoriel = null;
}
else {
factoriel = new FactorielE();
puissance = null;
}
}
private void checkAnswer(long answer){
}
```

```
private void setQuestion(String question, int s){
try{
...
String questions = "Démontrez que: " ;
if (fonction.compareTo("Puissance") == 0)
questions = questions + "4^5 = 1024" ;
else
questions = questions + "10! = 3628800" ;
apicom1.setCategorie("Questions") ;
apicom1.setQuestion(questions) ;
super.send() ;
...
}catch(Exception e){
}
}
}
private void Demo(){
}
public static void main(String args[])
{
Agents ag = new Agents() ;
}
}
```

Bibliographie

[1] D.A. Bender. Combining a computer simulation with a laboratory class - the best of both worlds? *Computers Education*, 13(3) :235–243, 1989.

[2] Blackboard. Blackboard, courseinfo. http://www.blackboard.com.

[3] R.G. Bruce. Distance delivery and laboratory courses. *ASEE/IEEE Frontiers in Education Conference*, 1997. http://fairway.ecn.purdue.edu/~fie/fie97/.

[4] Castor. Castor project. http://castor.exolab.org.

[5] Dongbin Tao Chong Xu. Aglet. http://www.cs.duke.edu/~chang/aglet.

[6] Université de Genève. Virolab. http://udrem.unige.ch/virolab.

[7] Collège de Maisonneuve. Laboratoire de physique virtuelle. Site Web: http://www.cmaisonneuve.qc.ca/~phy/dossierLPV/LPV.htm.

[8] Dalhousie University Physics Departement. Virtual laser lab. Site Web: http://vll.phys.dal.ca/.

[9] D. Dion, A. Escobar, J. Tremblay, and D. Laurendeau. Development of educational tools: a web-oriented approach. *ASEE/IEEE Frontiers in Education Conference*, 1997. http://fairway.ecn.purdue.edu/~fie/fie97/.

[10] F.-J. Elmer. The pendulum lab, 1998. http://monet.physik.unibas.ch/~elmer/pendulum/.

[11] J. Fischman. Wroking the web with a virtual lab and some java. *Science*, 273(2) :591–593, 1989.

[12] L. Foner. What's an agent? http://www.media.mit.edu/people/foner/agents.html.

[13] S. Franklin and A. Graesser. It is an agent or just a program? : A taxonomy for autonomous agents. http://www.msci.memphis.edu/~franklin/AgentProg.html.

[14] M.W. Gertz, D.B. Stewart, and P.K. Khosla. A human-machine interface for distributed virtual laboratories. *IEEE Robotics & Automation Magazine*, pages 5–13, 1994.

[15] T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *the Padua workshop on Formal Ontology*, 1993.

[16] T.R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 2(5) :199–220, 1993.

[17] Hachette/Edicef. Dictionnaire universel francophone, 1997.

[18] R.S. Haun. Steps to build the virtual laboratory. *Nature Biotechnology*, 15(7) :683–684, 1997.

[19] D. Hodson. Laboratory work as scientific method : three decades of confusion and distorsion. *Journal of curriculum studies*, 28(2) :115–135, 1996.

[20] C.M. Hoffman. Soft lab - a virtual laboratory for computational science. *Mathematics and Computers in Simulation*, 36(4) :479–491, 1994.

[21] A. Inglis. The virtual laboratory : A commic book approach to teaching laboraotry skills at a distance. developing open courses. *Centre for Distance Learning*, pages 165–186, 1993.

[22] W.L. Johnson, J.W. Rickel, and J.C. Lester. Animated pedagogical agents : Face-to-face interaction in interactive learning environment. *International Journal of Intelligence Artificial in Education*, 2000.

[23] M. Karweit. A virtual engineering/science laboratory. Site Web : http://www.jhu.edu/virtlab/virtlab.html.

[24] Madkit. The madkit project. http://www.madkit.org.

[25] S.A. Mengel, W.J. Adams, and M. Hagler. Using a hypertext instructional design methodology in engineering education. *ASEE/IEEE Frontiers in Education Conference*, 1997. http://fairway.ecn.purdue.edu/~fie/fie97/.

[26] Microsoft. http://msdn.microsoft.com.

[27] Sun Microsystem. http://java.sun.com.

[28] D. Naber and G. Leblanc. Providing a human biology laboratory for distant learners. *The American Journal of Distance Education*, 8(4), 1994.

[29] M.B. Nakhleh. Chemical education reserach in the laboratory environment. *Journal of Chemical Education*, 71(3):201–205, 1994.

[30] R. Nkambou. *Modélisation des connaissances de la matire dans un systme tutoriel intelligent: modles, outils et applications.* PhD thesis, Université de Montréal, 1996.

[31] R. Nkambou and F. Kabanza. Designing intelligent tutoring systems: A multiagent planning approach. *ACM SIGCUE Outlook*, 27(2):46–60, 2001.

[32] R. Nkambou and Y. Laporte. Interating learning agents in virtual laboratory. *World Conference on Educational Multimedia, Hypermedia & Telecommunication*, pages 1669–1671, 2000.

[33] R. Nkambou, Y. Laporte, and A. Mayers. Cooperating agents in a virtual laboratory for sypporting learning in engineering and science. *the 5th Annual World Conference on the WWW and Internet*, pages 789–792, 2000.

[34] H.S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(3):1–40, 1996.

[35] University of Western Ontario. Pendelium. http://www.cogsci.uwo.ca/VirtualLabs.

[36] OMG. http://www.omg.com.

[37] M. Petre, L. Carswell, B. Price, and P. Thomas. Innovations in large scale supported distance teaching: Tranformation for the internet, not just translation. *ASEE/IEEE Frontiers in Education Conference*, 1997. http://fairway.ecn.purdue.edu/~fie/fie97/.

[38] B. Piechaczyk. Les agents intelligents et les services d'information personnalisée sur internet. Master's thesis, Institut d'Etudes Politiques, 1996.

[39] D.R. Rehak. Carnigie mellon online : Web-mediated education. *ASEE/IEEE Frontiers in Education Conference*, 1997. http://fairway.ecn.purdue.edu/~fie/fie97/.

[40] J.J. Sparkes. Distance education and distance learning, developing open courses. *Centre for Distance Learning*, 13(3) :15–38, 1993.

[41] WBT Systems. Topclass. http://www.wbtsystems.com.

[42] Universal Learning Technology. Bravo ! http://ult.net/tour.

[43] Télé-université. Laboratoires virtuels pour l'éducation en sciences et en technologie (lvest). http://www.licef.teluq.uquebec.ca/lvest.

[44] Rice University. Virtual lab in statistics. http://www.ruf.rice.edu/~lane/vrls.html.

[45] WebCT. Webct. http://www.webct.com.