Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Contribution à l'enseignement d'ATM
utilisation combinée de Java et de VRML

Kassa, Aimé; Paulus, Valérie

Award date:
1998

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 03. Jul. 2025

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 1997-1998

Contribution à l'enseignement d'ATM :

Utilisation combinée de Java et de VRML

Aimé Kassa
Valérie Paulus

Mémoire présenté en vue de l'obtention du grade de Licencié et Maître en Informatique.

Résumé

Dans ce mémoire, nous présentons le développement d'un didacticiel de "simulation" du protocole ATM. Nous abordons d'abord le cadre dans lequel ce didacticiel a été mis au point. Nous avons en effet participé à un vaste projet appelé P2-Verso. L'objectif de ce projet était principalement d'évaluer la technologie ATM lors d'un travail de collaboration. Ce projet comportait des membres issus de divers pays et de multiples disciplines. Nous décrirons ce projet et l'impact qu'il a eu sur notre didacticiel. Nous décrivons par ailleurs les deux langages utilisés pour y parvenir : Java et VRML. Ensuite, nous donnons au lecteur une idée du travail de conception (pédagogique et fonctionnelle) qu'a nécessité la création de ce didacticiel. Enfin, nous abordons les aspects plus techniques avant de terminer par une évaluation du travail fourni.

Abstract

In this master thesis, we present the development of a simulation software. We approach first the setting in which this software was designed. We have indeed taken part in a vast project called P2-Verso. The object of this project was principally to evaluate the ATM technology during a collaboration work. The project's members came from diverse countries and from multiple disciplines. We describe this project and its impact on our software. We also describe the two languages that have been used to implement it : VRML and Java. Then, we give an idea about the design (educational and functional) needed for creating this software. Finally, we approach the more technical aspects before ending with an evaluation of the work.

Remerciements

Nous tenons à remercier

Philippe van Bastelaer, le directeur de notre mémoire, pour sa stimulation intellectuelle et sa confiance,

Daniel Muller, le maître de notre stage, pour son implication et, à travers lui, le Centre de Génie Electrique de Lyon pour avoir mis à notre disposition un très bon environnement de travail.

et tous les collègues du projet P2-Verso qui nous ont bien aidé : Franck Tarpin Bernard, René Chalon et Pascale Primet.

Nous voudrions aussi remercier nos familles et amis respectifs qui ont su supporter notre effacement. Merci pour leur patience.

Table des matières

RÉSUMÉ **3**

TABLE DES MATIÈRES **7**

TABLE DES FIGURES **11**

INTRODUCTION **13**

CHAPITRE I : PROJET P2-VERSO **17**

I. INTRODUCTION 17
II. PROBLÉMATIQUE ET OBJECTIFS 17
III. ARCHITECTURE DU PROJET 18
III.1. Présentation des institutions *18*
III.1.1. CNET, Centre National d'Etude des Télécommunications 18
III.1.2. ECL, Ecole Centrale de Lyon 19
III.1.3. LICEF, Laboratoire en Informatique Cognitive et Environnement de Formation 20
III.2. Structure en niveaux *20*
III.3. Equipes et tâches *21*
III.3.1. Niveau 1: Technique 22
III.3.2. Niveau 2 : Conception/médiatisation 22
III.3.3. Niveau 3 : Observation/analyse 22
III.4. Déroulement du projet *23*
III.5. Partenaires *23*
IV. MODULE DE TÉLÉFORMATION 23
V. LES CONFIGURATIONS ATM 25
V.1. La configuration globale *25*
V.2. La configuration de Lyon *26*
V.3. Encapsulation par ATM *27*
VI. PARTICIPATIONS ET CONTRIBUTIONS PERSONNELLES 31
VII. CONCLUSION 31

CHAPITRE II : TECHNOLOGIE ATM **33**

I. INTRODUCTION 33
II. ATM, LA TECHNOLOGIE DE COMMUTATION DU RNIS LB 33
III. FONDEMENTS DE LA TECHNOLOGIE ATM 34
III.1. Hiérarchie des connexions logiques *34*
III.1.1. Connexion VCC 34
III.1.2. Lien VCL 35
III.1.3. Connexion VPC 35
III.1.4. Lien VPL 35
III.2. Cellule ATM *36*
III.3. Commutation de cellules *37*
III.4. Multiplexage de cellules *37*
III.5. Modes de connexion *38*
III.6. Protocoles de routage *38*
III.7. Couches *39*
III.7.1. Couche AAL 39
III.7.2. Couche ATM 39
IV. CONCLUSION 40

CHAPITRE III : JAVA **41**

I. INTRODUCTION 41
II. ELÉMENTS DE BASE DU LANGAGE 41
III. CLASSES ET MÉTHODES 42
III.1. Classe *42*
III.2. Méthode *42*

III.3. Redéfinition et surcharge 44
III.4. Accessibilité 44
III.5. Héritage et accessibilité 44
IV. ARCHITECTURE EN MODULES 45
V. CONCURRENCE DES PROCESSUS 45
V.1. Concept de thread 45
V.2. Cycle de vie d'un processus Java 47
V.3. Moniteur, synchronisation et priorité 48
VI. APPLET 49
VII. AUTRES PARTICULARITÉS 51
VIII. COMPILATION 51
IX. CONCLUSION 52

CHAPITRE IV : LE LANGAGE VRML 53

I. INTRODUCTION 53
II. HISTORIQUE DU LANGAGE 53
III. DESCRIPTION D'UN MONDE VRML 55
IV. LE PILIER DES MONDES VRML : LE NŒUD 56
V. LES TYPES DE VRML 59
V.1 Les types de nœuds 59
V.2 Les types de champs 59
VI. LES CONCEPTS D'ÉVÉNEMENTS ET DE ROUTAGE D'ÉVÉNEMENTS 61
VII. LE DYNAMISME EN VRML 2.0 62
VII.1 Le script interne 62
VII.2 Le script externe 65
VIII. ARCHITECTURE THÉORIQUE D'UN BROWSER VRML 70
IX. CONCLUSION 71

CHAPITRE V : LA CONCEPTION PÉDAGOGIQUE 75

I. INTRODUCTION 75
II. LES ÉLÉMENTS DE BASE DE LA CONCEPTION PÉDAGOGIQUE 75
III. LA CONCEPTION PÉDAGOGIQUE DE NOTRE APPLICATION 77
IV. LES CHOIX 78
IV.1 Généralités 78
IV.4.2 La représentation du réseau dans le monde en trois dimensions 79
IV.3 La couche AAL 80
IV.4 Le mode de connexion 80
IV.5 La cellule ATM 81
IV.6 Les concepts de base 81
V. CONCLUSION 83

CHAPITRE VI : ANALYSE FONCTIONNELLE 85

I. INTRODUCTION 85
II. LES ÉLÉMENTS GÉNÉRAUX 85
II.1 Les restrictions au cahier des charges 86
II.2 L'adaptation du simulateur au public visé 87
II.3 Les contraintes 87
II. LES CONCEPTS À ILLUSTRER 89
III.1 Le concept de couche 91
III.2 Le concept d'unité de donnée 92
III.3 Le concept de commutation 93
IV. LES MAQUETTES DE L'INTERFACE 93
IV.1 Les maquettes générales 93
IV.2 Les maquettes des applets 95
IV.2.1 Le concept de commutation 95
IV.2.2 Le concept d'unités de données 95
IV.2.3 Le concept de couches 96
V. CONCLUSION 96

CHAPITRE VII : ARCHITECTURE PHYSIQUE 99

I. INTRODUCTION 99

II. DÉCOUPE EN MODULES 99
II.1. Module Réseau *100*
II.2. Module Atm *101*
II.3. Module Présentation *105*
III. CONCLUSION 105

CHAPITRE VIII : COUCHE DE TRAITEMENT 107

I. INTRODUCTION 107
II. CHOIX D'IMPLÉMENTATION 107
II.1. Classes d'objets *107*
II.2. Mode de connexion, signalisation et algorithme de routage *108*
II.3. Table de commutation et table d'adresses *108*
II.4. Cellule *109*
II.5. Couches ATM *109*
II.6. Primitives de service *110*
II.7. Gestion de la concurrence *112*
II.8. Connexion avec la couche de présentation *114*
III. INTERFACES DES CLASSES D'OBJETS 115
III.1. Module Réseau *116*
III.2. Module Atm *124*
IV. CONCLUSION 130

CHAPITRE IX : COUCHE DE PRÉSENTATION 133

I. INTRODUCTION 133
II. CHOIX D'IMPLÉMENTATION 133
II.1 Utilisation du script externe *133*
II.2 Description de la page HTML *134*
II.3 L'animation des cellules ATM *136*
II.4 L'applet d'animation *137*
II.5 Les particularités de la méthode RunCommutation() *141*
II.6 Les particularités de la méthode RunDonnées() *143*
II.7 Les particularités de la méthode RunCouches() *145*
III INTERFACES DES CLASSES D'OBJETS 146
III.1 Module Présentation *146*
IV. CONCLUSION 155

CHAPITRE X : EVALUATION ET CRITIQUES 159

I. INTRODUCTION 159
II. LE PROJET P2-VERSO 159
III. L'APPLICATION 160
III.1 L'avis des étudiants *160*
III.2 L'avis d'un spécialiste IHM *161*
III.3 Une auto-évaluation *163*
IV LES EXTENSIONS POSSIBLES 164
V CONCLUSION 165

CONCLUSION 167

ANNEXE : CODES SOURCES 171

I. MODULE *RÉSEAU* 171
II. MODULE *ATM* 182
III. MODULE *PRÉSENTATION* 201
IV L'APPLET JAVA 233
V. PAGE HTML 263
VI. MONDE VRML 263

GLOSSAIRE 267

BIBLIOGRAPHIE 269

Table des figures

FIGURE 1 : *ARCHITECTURE DU PROJET P2-VERSO* 21
FIGURE 2 : *DÉCOUPE DES NIVEAUX EN ÉQUIPES* 21
FIGURE 3 : *SPÉCIFICATION GÉNÉRALE DU MODULE* 24
FIGURE 4 : *CONFIGURATION GLOBALE* 25
FIGURE 5 : *CONFIGURATION DE LYON* 26
FIGURE 6 : *FORMATS DES TRAMES AAL1 ET AAL5* 28
FIGURE 7 : *ARCHITECTURE EN COUCHES* 28
FIGURE 8 : *TRANSPORT DES TRAMES MAC ETHERNET SUR ATM* 29
FIGURE 9 : *FORMAT D'UNE TRAME MAC ETHERNET* 29
FIGURE 10 : *ENCAPSULATION LLC/SNAP* 30
FIGURE 11 : *CONNEXION VCC* 34
FIGURE 12 : *CONNEXION VPC* 35
FIGURE 13 : *FORMAT D'UNE CELLULE ATM* 36
FIGURE 14 : *TABLE DE COMMUTATION ET ACHEMINEMENT DES CELLULES* 37
FIGURE 15 : *DÉCOUPE EN COUCHES* 39
FIGURE 16 : *CYCLE DE VIE D'UN PROCESSUS* 47
FIGURE 17 : *CYCLE DE VIE D'UNE APPLET* 49
FIGURE 18 : UNE SCÈNE VRML SIMPLE 58
FIGURE 19 : UNE ROUTE 62
FIGURE 20 : LE FONCTIONNEMENT DU SCRIPT INTERNE 64
FIGURE 21 : ILLUSTRATION DE L'EAI 66
FIGURE 22 : UNE REPRÉSENTATION DU CYLINDRE AJOUTÉ À LA SCÈNE VRML 69
FIGURE 23 : L'ANATOMIE D'UN BROWSER VRML 71
FIGURE 24 : LE MODÈLE EN COUCHE POUR ATM 79
FIGURE 25 : LE FORMAT DE CELLULE PRÉSENTÉ DANS LE SIMULATEUR 81
FIGURE 26 : LE TABLEAU DES CONCEPTS ILLUSTRÉS 89
FIGURE 27 : LA VUE AÉRIENNE 90
FIGURE 28 : LA VUE EN HAUTEUR 90
FIGURE 29 : UN SCHÉMA SIMPLIFIÉ DU RÉSEAU ATM POUR LE CONCEPT DE COUCHES 91
FIGURE 30 : LA MAQUETTE GÉNÉRALE DE L'APPLICATION 94
FIGURE 31 : LA MAQUETTE DE L'APPLET JAVA 94
FIGURE 32 : LA FORME DE L'APPLET POUR LE CONCEPT DE COMMUTATION 95
FIGURE 33 : LA FORME DE L'APPLET JAVA POUR LE CONCEPT D'UNITÉS DE DONNÉES 96
FIGURE 34 : LA FORME DE L'APPLET POUR LE CONCEPT DE COUCHES 96
FIGURE 35 : *COMPOSANTS DE L'APPLICATION* 99
FIGURE 36 : *GRAPHE DES RELATIONS ENTRE CLASSES DU MODULE RÉSEAU* 100
FIGURE 37 : *GRAPHE DES RELATIONS ENTRE LES CLASSES DU MODULE ATM* 102
FIGURE 38 : *GRAPHES DES RELATIONS D'HÉRITAGE ENTRE CLASSES DES MODULES RÉSEAU ET ATM* 103
FIGURE 39 : *GRAPHE GÉNÉRAL DES RELATIONS ENTRE CLASSES* 105
FIGURE 40 : *TABLE D'ADRESSES ET TABLE DE COMMUTATION* 109
FIGURE 41 : *TRANSFORMATION DANS LA MISE EN OEUVRE DES PRIMITIVES DE SERVICE* 111
FIGURE 42 : *SCÉNARIO D'ÉCHANGE* 111
FIGURE 43 : *SCÉNARIO D'ÉCHANGE EN ASYNCHRONE* 112
FIGURE 44 : *CYCLE DE VIE D'UNE INSTANCE DE LA CLASSE SOUSRESEAU* 113
FIGURE 45 : UNE PREMIÈRE SOLUTION D'ANIMATION DES CELLULES ATM 137
FIGURE 46 : DEUXIÈME SOLUTION POUR L'ANIMATION DES CELLULES ATM 137
FIGURE 47 : LA VUE AÉRIENNE DU RÉSEAU ATM DANS LA SCÈNE VRML 139
FIGURE 48 : LE SIMULATEUR LORS DE SON LANCEMENT 141
FIGURE 49 : L'ÉCRAN PENDANT L'ANIMATION DU CONCEPT DE COMMUTATION 143
FIGURE 50 : L'ÉCRAN PENDANT L'ANIMATION DU CONCEPT D'UNITÉS DE DONNÉES 144
FIGURE 51 : L'ÉCRAN PENDANT L'ANIMATION DU CONCEPT DE COUCHES 145

Introduction

La connaissance théorique est un trésor dont la pratique est la clé (Th. Fuller, 1732)

Le présent mémoire traite de la conception et de la réalisation d'une application susceptible d'être utilisée comme didacticiel dans l'enseignement des concepts de base de la technologie ATM. Toutefois, cette application n'est pas tout à fait un module isolé : elle fait partie d'un vaste module multimédia de téléformation sur la technologie ATM, réalisé dans le cadre d'un projet francophone.

En effet, durant notre stage de fin de cycle à l'Ecole Centrale de Lyon (ECL) et plus particulièrement au Centre de Génie Electrique de Lyon (CEGELY), nous avons participé à un projet dénommé *P2-Verso*, regroupant trois grandes institutions internationales : le Centre National d'Etude de Télécommunication de France Télécom (CNET), l'Ecole Centrale de Lyon (ECL) et le Laboratoire en Informatique Cognitive et Environnement de Formation (LICEF). L'objectif du projet est d'utiliser la même technologie ATM cette fois comme support d'un travail collaboratif et d'en évaluer les capacités à travers la réalisation de ce module multimédia de téléformation. Dans ce mémoire, il est plus question de la réalisation du module multimédia de téléformation que de la vérification de l'efficacité d'ATM. Nous traitons plus spécifiquement de l'implémentation d'une stratégie pédagogique adoptée par le projet, celle des simulations. Notre application est un ensemble des programmes de simulation.

A travers cette dissertation, nous voulons partager cette expérience que nous jugeons enrichissante. Ainsi, le contenu de ce mémoire est tiré de ce que nous avons vécu et réalisé durant notre stage à Lyon. Nous décrivons les grandes étapes du travail fait, du cahier de charges à l'implémentation physique de l'application. Toutefois, afin de ne pas trop charger ce document, certaines étapes sont traitées rapidement. Le document est divisé en quatre parties essentielles.

La première partie traite des généralités. Tour à tour, nous présentons sommairement le projet auquel nous avons participé, de sa configuration organisationnelle à sa configuration technique, quelques concepts de base de la technologie ATM, la raison d'être du projet, et les deux langages utilisés pour réaliser notre application.

Les deux parties suivantes abordent l'application elle-même. Dans la deuxième partie, nous présentons le travail de conception. Nous décrivons plus particulièrement la conception pédagogique et l'analyse fonctionnelle sous forme de cahier de charges.

La troisième partie, quant à elle, est consacrée à l'analyse technique. Notre application est réalisée selon l'approche *object oriented*. Pour chacune des couches composant l'architecture de l'application, nous présentons les choix techniques effectués et les interfaces des classes d'objets implémentées.

La quatrième et la dernière partie du mémoire traite de l'évaluation de l'application produite. En plus d'une évaluation par rapport aux objectifs que nous nous sommes fixés, nous

présentons une série d'autres évaluations faites par des tiers. Ceci nous permet d'évaluer objectivement le travail que nous avons réalisé.

Le lecteur pourra trouver dans la partie bibliographique, juste après la conclusion générale, les références des documents que nous avons utilisés tant dans la réalisation de l'application que dans la rédaction du présent mémoire. La partie glossaire reprend les définitions de certains termes importants utilisés tout au long de ce mémoire. Enfin, dans une partie annexe, le lecteur pourra trouver les codes sources de l'application.

Dans la rédaction de ce mémoire, nous utilisons les conventions suivantes :

- le texte est en fonte Times New Roman;
- le code Java ou VRML est en fonte `Courier`;
- un mot-clé Java ou VRML est écrit ***ainsi***;
- le nom d'une classe ou d'une méthode de notre application est écrit *ainsi*;
- tout autre mot (ou expression) important est également écrit *ainsi*.

Partie 1 - Généralités

Chapitre I : Projet P2-Verso

I. Introduction

Dans le cadre de notre stage de fin d'études au *Centre de Génie Electrique de Lyon* (CEGELY), nous avons participé à un vaste projet francophone, *P2-Verso*, visant à expérimenter l'utilisation d'ATM dans le cadre d'un travail collaboratif tel que la réalisation d'un module multimédia de téléformation sur ATM lui-même.

La section II de ce chapitre décrit la problématique et les objectifs assignés à ce projet. Nous décrivons, dans la section III, les institutions participant au projet, l'architecture du projet et les tâches dévolues à chacune de ses composantes.

La section IV décrit brièvement les objectifs du module multimédia de téléformation et ses spécificités. Comme nous l'avons dit plus haut, il s'agit, dans ce projet, d'évaluer l'utilisation de la technologie ATM. La section V précise la configuration réseau globale mise en place pour supporter le travail collaboratif à réaliser. En outre, nous donnons un bref descriptif de l'architecture matérielle présente à Lyon.

Notre travail dans ce projet consistait en la réalisation des simulations de certaines fonctionnalités de la technologie ATM. Le cahier de charges est abordé dans la deuxième partie de ce mémoire. Néanmoins, dans la section VI de ce chapitre, nous précisons notre niveau d'implication dans le projet.

II. Problématique et objectifs

Entre octobre 1995 et juin 1996, dans le cadre du projet *Franco-Réso*, s'est déroulée une expérimentation visant à évaluer l'utilisation d'un certain nombre de téléservices (en mode synchrone et asynchrone) sur des réseaux à faible/moyen débit entre des personnes réparties en cinq points dans deux pays francophones (la France et le Canada). Parmi les téléservices utilisés, nous pouvons citer le téléphone, le courrier électronique, le transfert des fichiers via le protocole FTP et la téléconférence. Les plates-formes de télécollaboration utilisées étaient *Macintosh* et *PC*. L'une des infrastructures réseaux mises en œuvre fut le RNIS à 2x64 Kbps. Cette expérience permit de mettre en évidence quelques faiblesses liées à ce type de réseau :

- la fonctionnalité d'insertion de fichiers attachés dans un courrier électronique ne permettait pas l'échange des fichiers textes et images supérieurs à une certaine taille;

- les outils de téléconférence se sont avérés inefficaces dans le cas d'un travail de télé-écriture d'une quinzaine de minutes ou plus lorsqu'il s'agit d'un document volumineux ou contenant des images ou des illustrations graphiques. Le transfert de gros fichiers en mode synchrone, la télé-conception et la télé-réalisation visuelle ont aussi montré leur inefficacité;

- le transfert de gros fichiers à l'échelle internationale par FTP s'est avéré inefficace pendant les heures d'important trafic;

- le transfert entre plates-formes différentes par FTP posait des problèmes (disparition, contamination ou détérioration du contenu);

- le temps de connexion au site dédié au projet était souvent long. Il fallait toujours prévoir des créneaux horaires pendant des heures de faible trafic afin de pouvoir travailler avec des fichiers volumineux.

Partant de l'hypothèse que les problèmes susmentionnés étaient principalement dus à la faiblesse de la largeur de bande disponible sur les réseaux conventionnels, le projet *P2-Verso* vise à vérifier la capacité d'un réseau ATM à 2 Mbps à supporter, à distance, des activités de télécollaboration et à évaluer l'amélioration par rapport à un accès RNIS à 2x64 Kbps.

En outre, le projet devra réaliser un module multimédia d'introduction à la technologie ATM, permettant un usage tant en formation qu'en présentation ou démonstration. Après discussion et consensus, les acteurs ont assigné au projet l'objectif ci-après :

> *Evaluation de l'interaction collaborative sur ATM, dans ses dimensions technologiques et humaines, au travers de la réalisation d'un module de téléformation, qui devra être utilisable à l'issue du projet.*

Pour atteindre cet objectif, les institutions participant au projet sont interconnectées via un réseau ATM à 2 Mbps. La configuration réseau mise en place est abordée un peu plus loin, dans ce chapitre. Comme pour le projet *Franco-Réso*, le projet *P2-Verso* vise l'évaluation d'un réseau, ATM cette fois-ci, dans l'utilisation d'une gamme de téléservices tels que le courrier électronique, la téléconférence, la vidéoconférence, le protocole FTP, etc.

III. Architecture du projet

Le projet *P2-Verso* regroupe trois institutions : l'*Ecole Centrale de Lyon* (ECL), le *Centre de recherche en informatique cognitive et environnement de formation de Montréal* (LICEF) et le *Centre de recherche et développement de France Télécom* à Paris et à Lannion (CNET). Nous faisons une brève présentation de ces trois institutions avant d'aborder l'architecture organisationnelle du projet. Pour de plus amples informations sur une de ces institutions, le lecteur pourra consulter le site Web associé. Les adresses des sites Web sont reprises dans la partie bibliographie de ce mémoire. Il faut noter que les informations contenues dans cette partie du mémoire proviennent essentiellement de ces sites.

III.1. Présentation des institutions

III.1.1. CNET, Centre National d'Etude des Télécommunications

Le CNET est le Centre de recherche et développement de France Télécom à Lannion et Paris. Ses activités peuvent se résumer en trois points :

1. le développement et l'évolution des services innovants pour ses clients;

2. l'élaboration et l'intégration des architectures de réseaux (réseaux intelligents, ATM, etc.) pour utiliser au mieux les infrastructures;

3. la mise en place, pour les exploitants de France Télécom, des outils de gestion de réseaux avancés, garants de qualité et de flexibilité.

Le CNET souhaite utiliser le module multimédia à produire comme un outil de démonstration des possibilités de la technologie ATM. Le module pourra être utilisé sur ses stands d'exposition.

III.1.2. ECL, Ecole Centrale de Lyon

L'ECL est une des Grandes Ecoles de France. Elle délivre le diplôme d'ingénieur et joue un rôle déterminant dans la recherche et le développement de technologies nouvelles. Dans le projet *P2-Verso*, l'ECL est représentée par un groupe de recherche spécialisé dans l'étude des *Interactions Collaboratives pour la Téléformation et les Télé-activités*, ICTT. Ce groupe de recherche regroupe des personnes venant des départements *Sciences Pour l'Ingénieur* (SPI) et *Sciences de l'Homme et de la Société* (SHS). En ce qui concerne le projet P2-Verso, ICTT est représenté par le département SPI.

SPI est un des départements du CNRS, le *Centre National de la Recherche Scientifique*. Il a pour objectif le développement des connaissances sur les objets imaginés ou maîtrisés par l'homme dans les trois champs d'application majeurs suivants : moyens et méthodes de production, information et communication et élaboration-transformation-transfert. Il associe des personnes venant essentiellement des unités de recherche suivantes :

1. le *Groupe de Recherche en Apprentissage, Coopération et Interfaces Multimodales pour la Productique* (GRACIMP). Il faut fondamentalement intégrer l'homme dans sa composante individuelle et collective dans les nouvelles technologies de l'information (réseaux, multimédia, ATM, etc.) afin de répondre à ses besoins de communication "naturelle" avec son environnement, sans barrière initiatique (langages, protocoles, procédures, etc.). L'analyse de ces besoins, de ces attentes doit permettre d'anticiper sur les usages afin de concevoir, façonner, produire des outils, méthodes et méthodologies adaptés et donc efficaces. C'est dans ce contexte que GRACIMP se mobilise autour de la conception de systèmes d'apprentissage et de communication pour la conduite d'unités de production;

2. le *Centre de Génie Electrique de Lyon* (CEGELY). Il réunit des compétences de chercheurs en Génie électrique reconnus au niveau tant national qu'international. Le centre est organisé autour de trois thèmes de recherche :

   Thème 1 : Modélisation et C.A.O. en électromagnétisme :
   - Modélisation tridimensionnelle des phénomènes électromagnétiques,
   - C.A.O. des dispositifs électromagnétiques,
   - Contrôle non destructif.

   Thème 2 : Circuits et systèmes :
   - Perturbations conduites et rayonnées,
   - Haute tension, isolation et appareillage,
   - Commande et intégration.

   Thème 3 : Composants :
   - Composants de puissance "haute température",
   - Simulation des composants de puissance,

- Etude du vieillissement des composants.

III.1.3. LICEF, Laboratoire en Informatique Cognitive et Environnement de Formation

La Télé-université est une école supérieure du réseau de l'Université du Québec, fondée en 1972. Elle œuvre spécifiquement à distance : les programmes et les cours qu'elle offre peuvent être suivis entièrement à distance. Pour rejoindre sa clientèle à son domicile ou à son lieu de travail, la Télé-université utilise des technologies de communication les plus modernes telles que la télématique, la téléconférence ou la vidéoconférence. Pour ses recherches, elle a créé un centre dénommé LICEF dont les travaux se regroupent en trois grands processus :

1. la *recherche générique orientée* qui vise à résoudre des problèmes de recherche sur des questions liées aux méthodologies, aux modèles ou aux outils informatisés d'apprentissage, dans le but premier de faire avancer les connaissances dans les divers domaines de l'informatique cognitive et de l'ingénierie didactique;

2. la *recherche-développement pré-concurrentielle* qui vise à développer des systèmes informatiques ou des méthodologies intégrant divers résultats de recherche en vue de les appliquer à la conception, à la réalisation, à la diffusion ou à la gestion des environnements d'apprentissage;

3. les *applications pilotes ou les recherches-action* qui consistent à réaliser un produit ou un service innovateur à partir des résultats de la recherche générique orientée ou de la recherche-développement pré-concurentielle, tout en validant les résultats ou en explorant de nouvelles avenues de recherche.

Parmi ses nombreuses recherches, nous pouvons citer le *Campus virtuel*, l'*Ecole Informatisée Clés En Main* (EICEM), etc. La plupart des projets de recherche de LICEF sont orientés vers une intégration des connaissances, des médias et des réseaux télématiques, de façon à soutenir la démarche d'apprentissage de l'apprenant dans un contexte de formation à distance.

A travers les descriptions ci-dessus, nous voulons surtout souligner la qualité de chacune de trois institutions. Chacune d'elles apporte son expertise ou son savoir-faire pour que le projet arrive à terme.

III.2. Structure en niveaux

Le projet *P2-Verso* est structuré en trois niveaux, chaque niveau étant dirigé par un coordinateur. La Figure 1 présente l'architecture du projet. Pour chaque institution, l'arc le reliant à un niveau du projet indique son degré de participation en nombre d'acteurs. Pour l'équilibre et l'harmonie du projet, ce dernier est piloté par LICEF et ECL, avec l'assistance de CNET. Mais la coordination principale revient à LICEF. En effet, c'est à partir de Montréal que sont initiées toutes les séances synchrones. La coordination générale du côté de la France est assurée par ECL. Par ailleurs, chaque institution s'occupe de la coordination d'un des trois niveaux composant le projet.

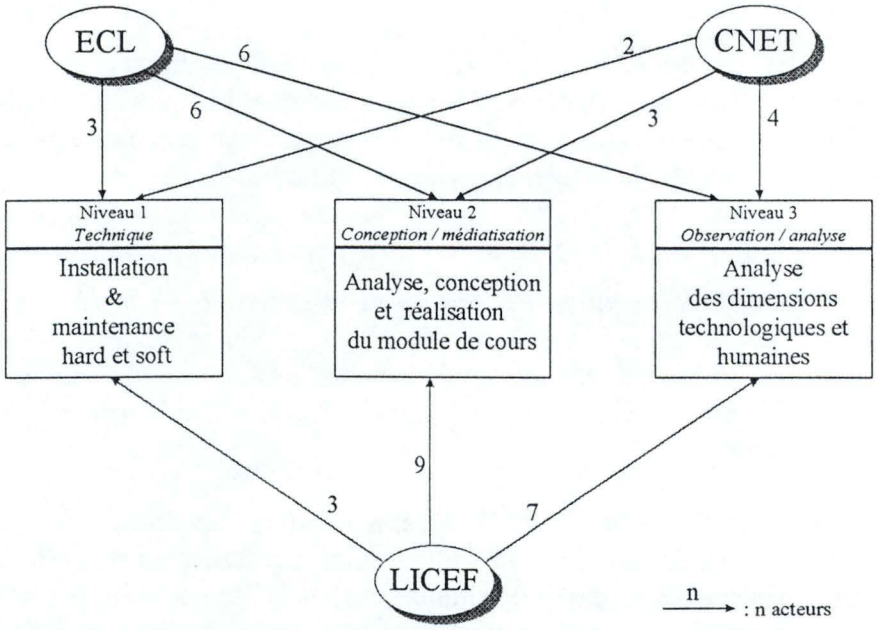


Figure 1 : *Architecture du projet P2-Verso*

Les niveaux sont découpés en équipes afin de réduire la complexité du travail à réaliser. La Figure 2 donne une découpe de ces trois niveaux en équipes. Dans un même niveau, un acteur peut appartenir à plus d'une équipe. De même, un acteur peut intervenir dans plus d'un niveau du projet.

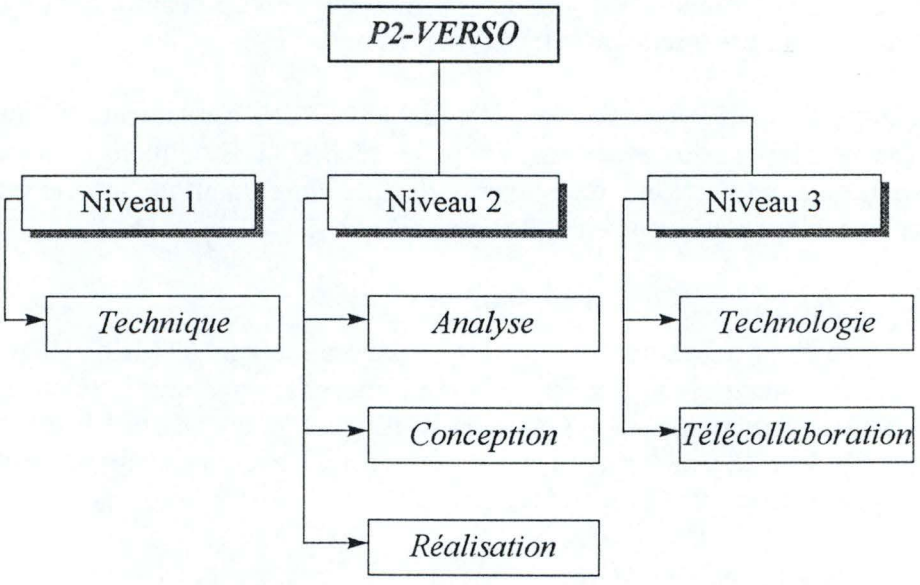


Figure 2 : *Découpe des niveaux en équipes*

III.3. Equipes et tâches

Comme signalé dans la sous-section précédente, les équipes sont regroupées en niveaux. Dans ce qui suit, nous décrivons brièvement chacun des niveaux, en précisant pour chacune des équipes le composant les tâches associées.

III.3.1. Niveau 1: Technique

Le niveau *Technique*, coordonné par le CNET, s'occupe essentiellement du support technique et n'est composé que d'une seule équipe : l'équipe *technique*. Elle s'occupe de la configuration matérielle et logicielle des sites abritant les trois institutions participant au projet. Cette équipe comprend des ingénieurs et des spécialistes en télécommunications.

III.3.2. Niveau 2 : Conception/médiatisation

Le niveau *Conception/médiatisation* est le niveau responsable de la réalisation du module multimédia de téléformation à produire. Ce niveau comprend des ingénieurs et des spécialistes en télécommunications, des programmeurs, des ergonomes, des médiatiseurs, etc. Il est composé de trois équipes dites *équipes d'ingénierie d'activité de téléformation/médiatisation* :

1. une équipe *Analyse* s'occupant de l'identification des contenus du domaine (la technologie ATM), de la définition des objectifs à assigner au module et des stratégies pédagogiques à exploiter (exposés, simulations, etc.), du choix de la métaphore visuelle et sonore, des instruments pédagogiques, du matériel pédagogique, de la plate-forme de développement, de l'établissement des lignes directrices des fonctionnalités de l'interface, etc. Brièvement, sans trop entrer dans les détails, les tâches assignées à cette équipe permettront au projet de se doter de trois modèles suivants : un modèle de connaissance, un modèle pédagogique et un modèle de médiatisation. Ces trois modèles seront raffinés tout au long du projet, au fil des phases;

2. une équipe *Conception* s'occupant de l'établissement du type de clientèle visée, de la définition de la structure pédagogique du module, de la préparation d'un croquis de la première maquette visuelle pour l'interface, etc.;

3. une équipe *Réalisation* s'occupant essentiellement de l'implémentation de la solution adoptée par l'équipe de conception. Parmi ses tâches, nous pouvons citer la télé-écriture des textes, la télé-écriture du scénario détaillé de formation, la télé-réalisation des stratégies pédagogiques choisies, etc.

III.3.3. Niveau 3 : Observation/analyse

L'objectif du projet est d'évaluer l'interaction collaborative sur ATM, dans ses dimensions technologiques et humaines, au travers de la réalisation d'un module de téléformation. Cette évaluation est la raison d'être de ce niveau. Il est coordonné par ICTT, spécialiste en la matière. Pour pouvoir se pencher sur les deux dimensions évoquées dans l'objectif du projet, ce niveau est divisé en deux équipes de recherche :

1. une équipe *Télécollaboration* s'occupant des dimensions humaines. Il est question d'analyser les acteurs impliqués dans la production du module multimédia de téléformation et l'activité de conception en elle-même;

2. une équipe *Technologie* s'occupant des dimensions technologiques, l'objectif étant de mesurer le delta entre RNIS 2x64 Kbps et ATM à 2 Mbps. Il est question de savoir s'il y a persistance ou non des problèmes mis en évidence par le projet *Franco-Réso*. En outre, l'équipe devra identifier les apports et les obstacles technologiques à la télécollaboration. Pour atteindre ses objectifs, l'équipe procède de la manière suivante :

- au fil du projet, elle effectue une étude de l'adéquation des outils utilisés lors des séances synchrones et asynchrones. Pour cela, un certain nombre d'enregistrements (fichier *log* sur *e-mail*, fichier *log* sur Web et enregistrement audio/vidéo des séances synchrones) sont réalisés;
- au fil des séances synchrones, elle effectue une étude des performances et de l'adéquation du système synchrone à la tâche (la télécollaboration du niveau 2);
- en dehors du déroulement de la tâche, elle mesure les caractéristiques et les performances des outils réseaux et informatiques.

III.4. Déroulement du projet

Le projet se déroule d'octobre 1997 à mars 1998 et comporte cinq phases : le démarrage, la logistique, l'analyse, la conception et la réalisation. Certaines phases du projet, plus particulièrement celles concernant le niveau 2, peuvent se chevaucher afin de gagner du temps, le projet étant limité dans le temps.

III.5. Partenaires

Le projet *P2-Verso* bénéficie du concours de plusieurs sociétés et plus particulièrement de sociétés de télécommunication telles que TéléGlobe, General DataComm (GDC), France Télécom, Deutsche Telekom et British Telecom.

IV. Module de téléformation

Pour rappel, dans le cadre du projet *P2 Verso*, le travail de télécollaboration consiste en la réalisation d'un module de téléformation de deux heures sur la technologie ATM. Cette section décrit brièvement les différentes facettes de ce module.

Le module doit permettre à un apprenant de maîtriser les concepts de base de la technologie ATM. Par le terme *apprenant*, nous faisons référence à une catégorie de personnes susceptibles d'utiliser le module. La clientèle visée comprend :

- des ingénieurs de télécommunications ou de réseaux en formation continue,
- des élèves ingénieurs en informatique répartie ou Télécom,
- et des utilisateurs de grands réseaux (Transpac, Internet, etc.).

En outre, il est supposé que cette clientèle maîtrise déjà quelques concepts de télécommunications. L'ingénieur réseau doit être familier avec les modes de transmission, les techniques de commutation et les architectures courantes de réseaux (OSI, TCP/IP). Quant aux étudiants, des connaissances de base en télécommunications et réseaux informatiques sont des prérequis.

Le module multimédia de téléformation à produire possède des objectifs de formation et d'apprentissage. Sur le plan de la formation, l'objectif est de sensibiliser l'apprenant aux avantages des réseaux ATM par rapport aux réseaux à débit moyen et au RNIS bande étroite.
Par contre, au niveau de l'apprentissage, l'objectif principal est de permettre à l'apprenant d'intégrer la technologie ATM comme réponse possible aux besoins de haut débit et de qualité de service des réseaux multi-services et des applications actuelles et futures. Cet objectif principal peut être découpé en objectifs beaucoup plus spécifiques :

1. reconnaître les limites des technologies classiques en regard des nouveaux besoins. Le module multimédia à produire devrait aborder le problème de l'évolution des besoins (les services, le transport, les infrastructures et systèmes de transmission). Les stratégies pédagogiques retenues sont les exposés, les démonstrations et les mises en situation;

2. situer ATM par rapport aux autres technologies (RTC, RNIS bande étroite, X25, IEEE 802, IP, etc.). Ici, il est question d'amener l'apprenant à comprendre les caractéristiques générales de la technologie ATM afin qu'il soit à même de donner des exemples montrant l'universalité d'ATM, de comparer la commutation de cellules par rapport à d'autres types de commutations, d'expliquer les principes de négociation, etc. A ce niveau, la simulation est retenue comme une des stratégies pédagogiques;

3. expliquer les principes de la commutation de cellules et les différentes classes de services associés;

4. mettre en perspective les évolutions possibles de la technologie ATM à partir des grandes tendances actuelles. L'apprenant sera capable de citer les exemples de services, de rappeler les objectifs du RNIS-LB, de nommer les solutions ATM-LAN, de repérer les difficultés (notamment ATM sur IP) et d'estimer les évolutions.

La Figure 3 donne une spécification générale du module multimédia à produire. En résumé, le module multimédia de téléformation disponible sur le Web ou sur CD doit être facile à mettre à jour. Son interface doit pouvoir permettre à la fois une navigation séquentielle et un accès direct. Pour atteindre les objectifs d'apprentissage, des stratégies pédagogiques telles que les simulations sont proposées. Les stratégies médiatiques sont définies afin de supporter et renforcer les objectifs pédagogiques.

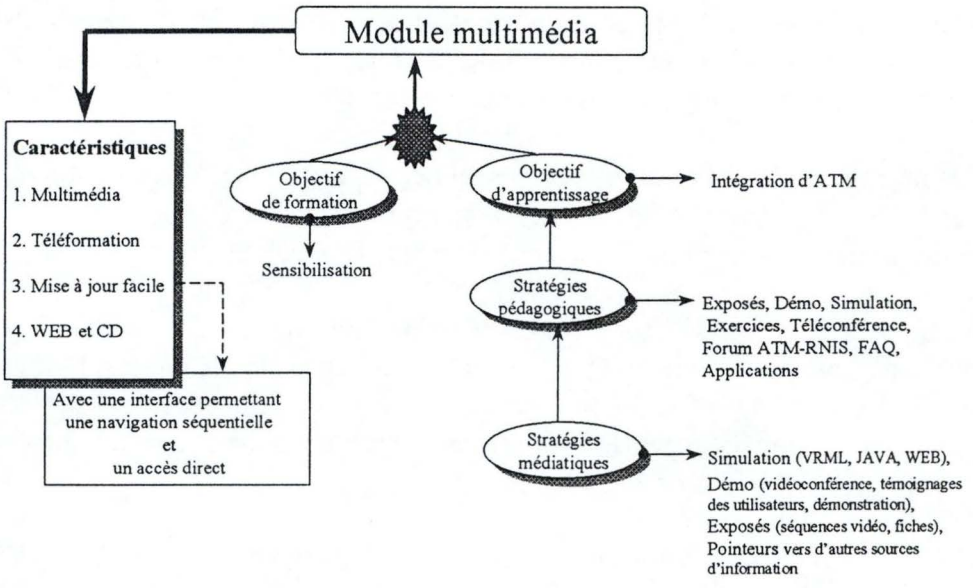


Figure 3 : *Spécification générale du module*

Concernant le contenu du module multimédia de téléformation, les experts de contenu ont retenu le plan suivant :

I. *L'évolution des besoins*

II. *Les caractéristiques générales d'ATM*
III. *Les principes et les traitements*
IV. *La mise en œuvre*
V. *Les services et les applications*
VI. *Les équipements*

Comme précisé plus haut, une des stratégies pédagogiques retenues est la simulation. Les experts de contenu ont prévu des simulations pour des concepts tels que le format de la cellule, le principe de la commutation de cellules, le multiplexage temporel et les couches ATM. Les concepts réellement implémentés par notre application sont décrits dans le chapitre consacré au cahier de charges. Dans ce mémoire, nous ne détaillons pas tout le contenu du module de téléformation à produire.

V. Les configurations ATM

Comme nous l'avons précisé précédemment, les différentes institutions participant au projet *P2-Verso* sont interconnectées via des liaisons ATM. Dans cette section, nous présentons la configuration globale du projet et la configuration mise en place à l'ECL. Nous terminons cette section en décrivant brièvement l'utilisation d'ATM dans le projet.

V.1. La configuration globale

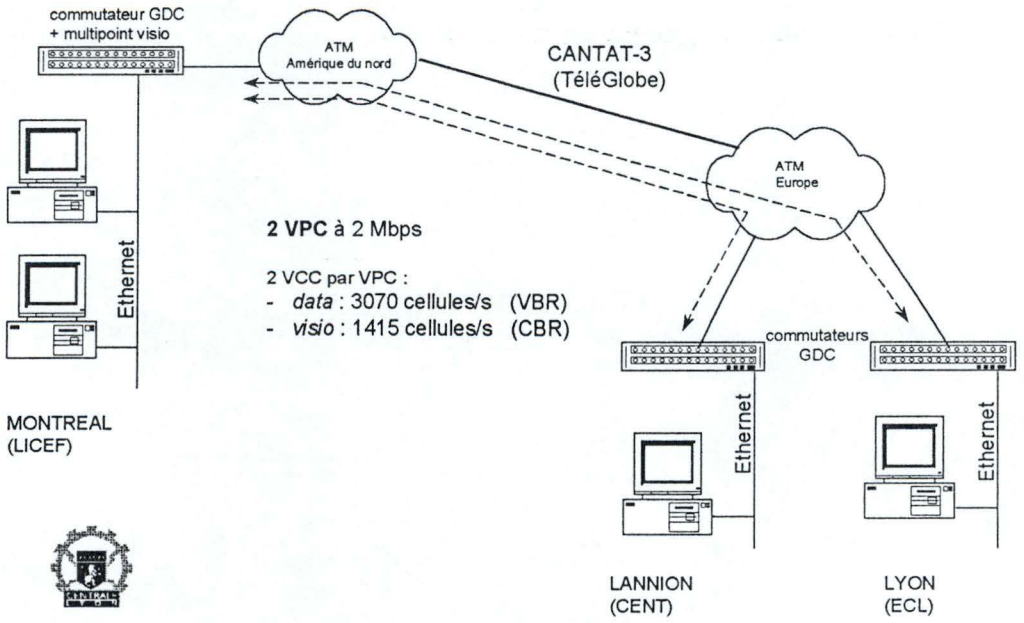


Figure 4 : *Configuration globale*

La Figure 4 donne une description de la configuration globale mise en place pour le projet *P2-Verso*. Entre la France et Montréal, il y a deux sous-réseaux ATM interconnectés par une liaison intercontinentale réalisée via un câble CANTAT-3 de TéléGlobe. L'ensemble {*commutateurs GDC, réseaux ATM*} se comporte comme un pont permettant la communication entre les différents réseaux Ethernet. En ce qui concerne les liaisons ATM, il y en a deux reliant respectivement Montréal à Lannion (CNET) et Montréal (LICEF) à Lyon (ECL). Chacun de ces deux liens est une connexion VPC comprenant deux connexions VCC :

1. une connexion VCC à 3070 cellules/seconde (soit 1.3 Mbps) pour les données fonctionnant en VBR (Variable Bit rate);

2. une connexion VCC à 1415 cellules/seconde (soit 0.6 Mbps) pour la visio fonctionnant en CBR (Constant Bit Rate).

Par connexion VPC, une bande passante de 2 Mbps est réservée, soit un total de 4 Mbps pour le projet.

Les trois commutateurs utilisés dans la configuration globale sont tous d'un même constructeur, *General DataComm*. Ce choix permet bien sûr de réduire le travail de l'équipe Technique. Nous attirons l'attention du lecteur sur le fait que ces commutateurs ne doivent pas être pris comme des commutateurs au sens ATM mais bien des *edge devices* connectant des utilisateurs au réseau ATM. Dans la suite de cette section, pour désigner un tel commutateur, nous utilisons l'expression *commutateur GDC* afin de le distinguer d'un *commutateur ATM*.

A Montréal, un PC est utilisé spécialement pour la coordination principale du projet. Pour rappel, nous avons précisé que la coordination du projet revenait au LICEF. En effet, toute séance synchrone est initiée et coordonnée par LICEF. Nous n'avons pas de plus amples informations sur l'interface *multipoint visio* présente dans le commutateur GDC de Montréal. Toutefois, c'est un pont analogique (audio et vidéo analogiques), piloté grâce au logiciel *Multimedia One*. Nous n'en dirons pas plus.

V.2. La configuration de Lyon

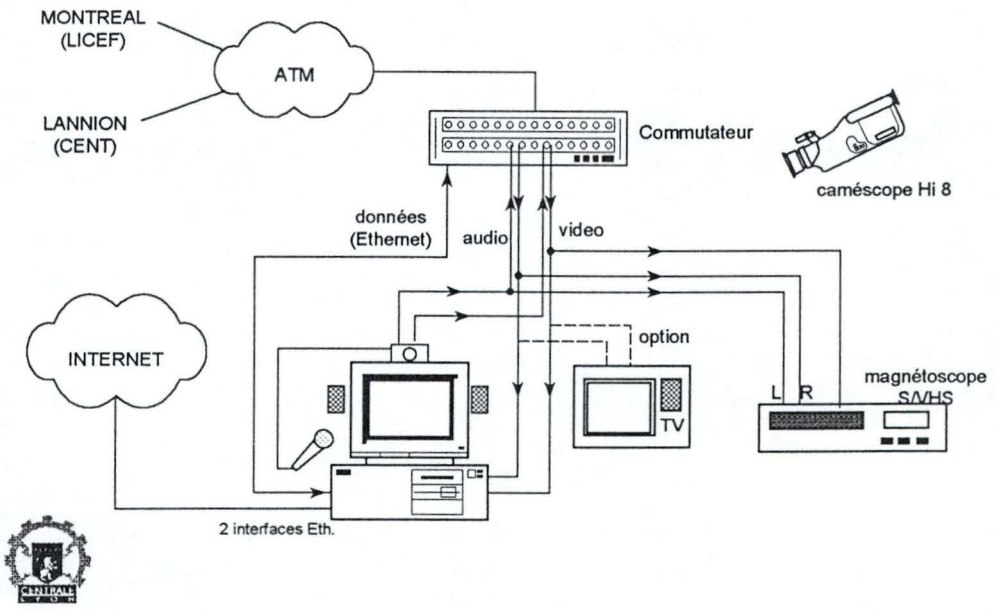


Figure 5 : *Configuration de Lyon*

La Figure 5 décrit la configuration réseau utilisée par les acteurs travaillant à l'ECL. Cette configuration comprend les équipements suivants :

- un commutateur GDC en réseau Ethernet avec un PC et connecté au réseau ATM Europe;

- un PC relié au commutateur GDC via une carte Ethernet pour l'envoi et la réception des données. En outre pour la réception du son et de la vidéo, il y a respectivement un câble audio et un câble vidéo entre le PC et le commutateur GDC. Comme on le voit sur la Figure 5, les deux câbles duplex alimentent aussi un magnétoscope afin de permettre les enregistrements des séances synchrones.

  Le PC possède une deuxième carte Ethernet le reliant au réseau interne de l'ECL. Ceci permet éventuellement de réaliser des connexions Internet;

- un micro relié à une caméra de taille réduite (sur la Figure 5, elle est au-dessus du PC). Le micro et la caméra permettent l'envoi du son et de la vidéo vers le commutateur GDC. En fait, ces deux équipements sont essentiels pour des séances synchrones telles que la vidéoconférence. Ils permettent aux différents intervenants de se voir (vidéo) et de s'entendre (audio) pendant les vidéoconférences;

- une TV pour les vidéoconférences. Afin de permettre, pendant les vidéoconférences, le partage efficace des documents entre les différents acteurs, les sorties audio et vidéo du commutateur GDC sont dirigées vers la TV, le PC étant dédié à la manipulation des logiciels informatiques. C'est donc à partir de cet équipement que Lyon voit les différents intervenants lors d'une vidéoconférence;

- un caméscope HI 8 pour filmer les intervenants présents à Lyon et l'écran de la TV. Ces enregistrements sont utilisés ensuite par le niveau 3 (observation/analyse) pour évaluer l'interaction collaborative sur ATM.

Il faut remarquer que le magnétoscope et le caméscope HI 8 permettent de garder des traces des différentes vidéoconférences. Il est toujours possible de visualiser en différé une séance synchrone.

V.3. Encapsulation par ATM

Un réseau ATM est susceptible de transporter de la voix, des données et des images fixes ou animées grâce au concept de classes d'applications. Ces classes ont été définies en fonction de la nécessité de synchronisation temporelle entre émetteur et récepteur, du débit binaire (constant ou variable) et du type de connexion (orienté connexion ou non orienté connexion) [MELIN97]. A chacune de ces classes d'applications correspond une couche d'adaptation à ATM ou *ATM Adaptation Layer* (AAL).

Dans le projet *P2-Verso*, seules les couches AAL1 et AAL5 sont utilisées et plus particulièrement dans les commutateurs GDC. En effet, chaque commutateur GDC du projet contient :

- une carte d'adaptation *ATM-CODEC H320* à 384 Kbps (soit 6x64 Kbps) pour l'audio et la vidéo. Il faut noter que cette carte étant "boguée" et dépassant parfois son débit théorique, un canal de 566 Kbps (1415 cellules/seconde) est prévu;
- une carte d'adaptation *ATM-Ethernet ETHDOC* pour les données.

Avant de décrire l'architecture en couches correspondant à la configuration globale du projet, il nous semble important de faire un bref rappel sur AAL1, AAL5 et Ethernet. Pour des informations complémentaires, le lecteur peut consulter [MELIN97] et [TANEN97].

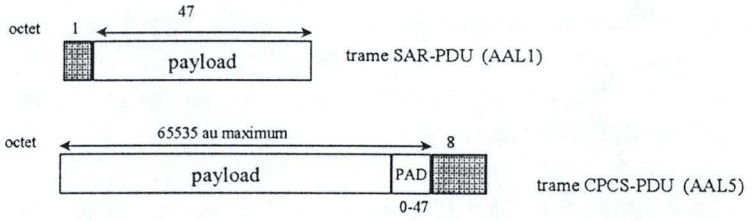


Figure 6 : *Formats des trames AAL1 et AAL5*

AAL1 est prévue pour un trafic de type voix et travaille en émulation de circuit. Dans le projet, elle est utilisée pour la vidéo, avec un débit fixe. La Figure 6 illustre sommairement les formats des trames AAL utilisées dans la configuration du projet. Les informations détaillées sur le format de cette trame peuvent être trouvées dans les deux livres cités précédemment. La trame est transmise comme telle à l'entité couche ATM (sa taille est de 48 octets, correspondant à la taille du champ *Payload* ou charge utile d'une cellule ATM).

Nous utilisons AAL5 pour le transfert des données en mode non connecté. Une trame AAL5 comme illustrée sur la Figure 6, est segmentée en unités de 48 octets (la longueur de la charge utile d'une cellule ATM) par l'entité sous-couche SAR (*Segmentation And Reassembly*) avant d'être passée à l'entité couche ATM. En jouant sur la longueur du champ de remplissage *PADding* (PAD), la taille de cette trame est toujours un multiple de 48.

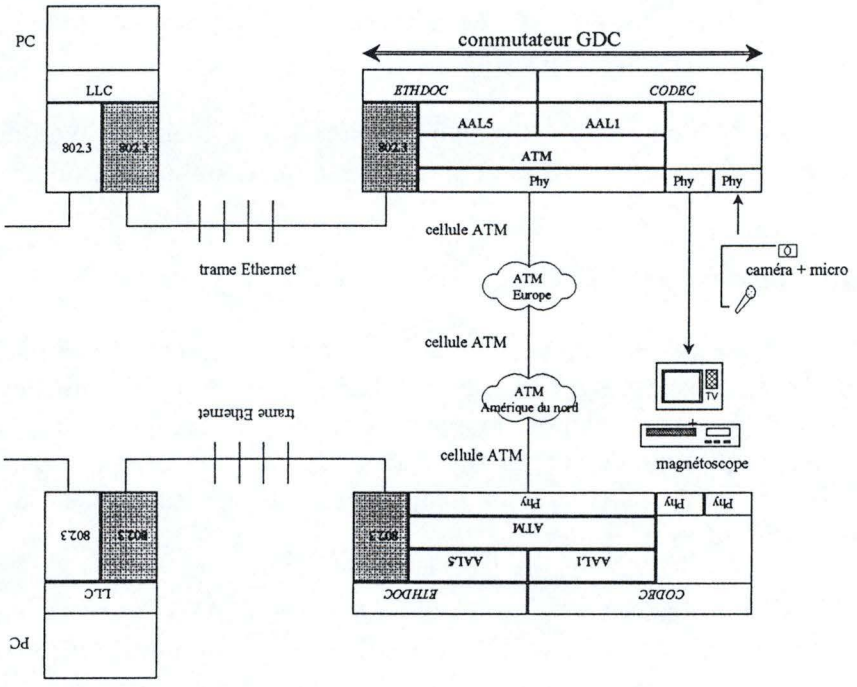


Figure 7 : *Architecture en couches*

La Figure 7 illustre l'architecture en couches correspondant à la configuration globale du projet. Pour rappel, chaque PC possède deux cartes Ethernet, une le reliant au réseau LAN de l'institution et une autre le reliant à un commutateur GDC. Les deux interfaces présentes dans le commutateur GDC permettent respectivement l'adaptation ATM-Ethernet et l'adaptation

ATM-CODEC. Dans ce qui suit, nous illustrons particulièrement et d'une façon brève l'utilisation d'AAL5.

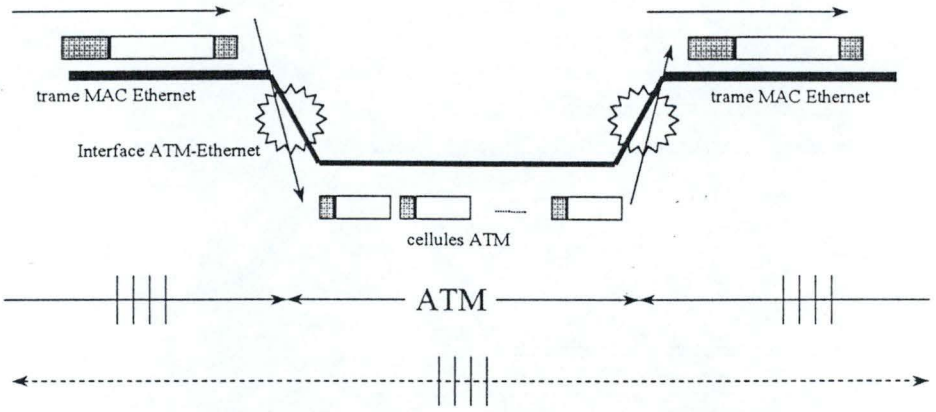


Figure 8 : *Transport des trames MAC Ethernet sur ATM*

L'interface *ATM-Ethernet ETHDOC* s'appuie sur AAL5 pour fournir son service. AAL5 peut être vue comme une sorte de tunnel à travers le réseau ATM, les PC ne voyant que de l'Ethernet (Figure 8).

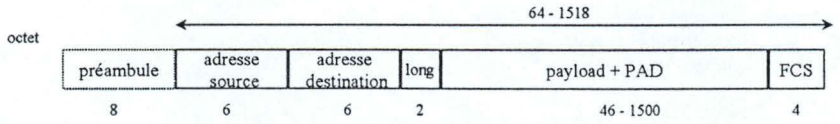


Figure 9 : *Format d'une trame MAC Ethernet*

Pour rappel, comme illustré sur la Figure 9, une trame MAC Ethernet comprend les champs suivants : le préambule, les adresses source et de destination, la longueur du champ *Payload*, le *Payload*, le *PAD* et le *Frame Check Sequence* (FCS). En réalité, le préambule, étant une séquence binaire préétablie transmise en tête afin d'assurer la synchronisation du récepteur sur l'émetteur [PVB97], précède la trame. Nous pouvons considérer que le préambule est de niveau physique. Finalement, l'en-tête de la trame MAC Ethernet comprend 14 octets. Pour des informations complémentaires sur le format de la trame MAC Ethernet, le lecteur peut consulter [TANEN97].

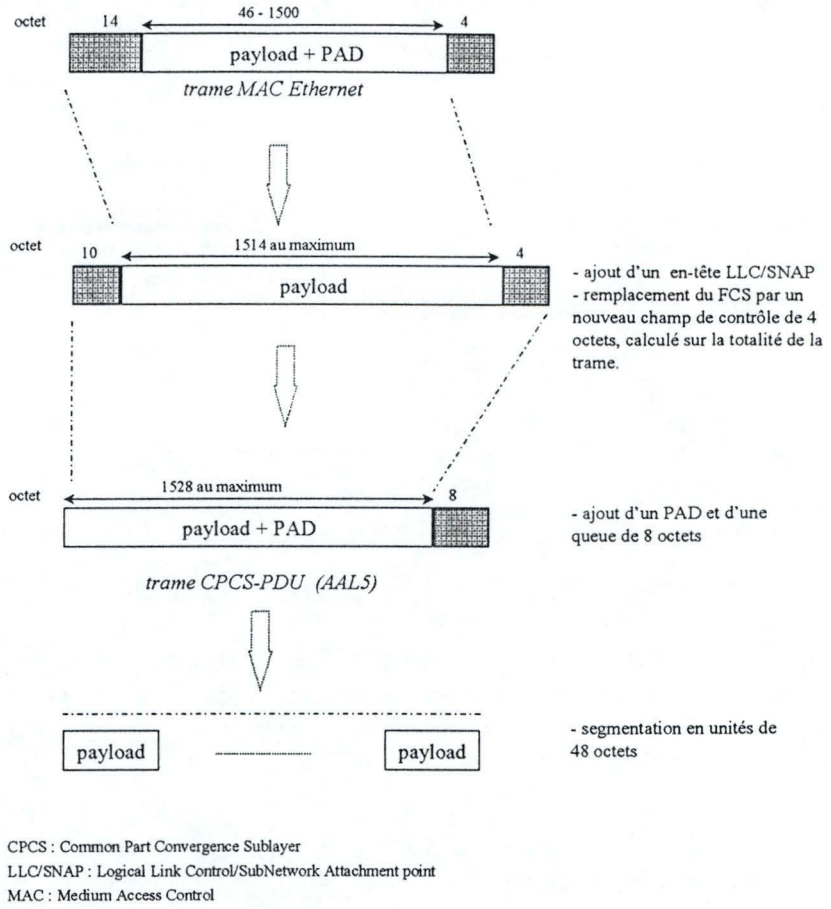


Figure 10 : *Encapsulation LLC/SNAP*

Une trame MAC Ethernet envoyée par le PC de Lyon, d'une façon caricaturale, est encapsulée dans des cellules ATM avant d'être remise à un PC de Montréal (Figure 10). L'encapsulation LLC/SNAP est définie par le RFC 1483. C'est un mécanisme complexe et nous n'en dirons pas plus dans ce mémoire. Concrètement :

- un en-tête LLC/SNAP de 10 octets est ajouté à la trame MAC Ethernet par l'entité ETHDOC, le champ FCS est enlevé et remplacé par un nouveau champ de contrôle de 4 octets calculé sur la totalité de la trame. La taille maximale de la trame ainsi modifiée passe à 1528 octets;

- la trame modifiée est passée à l'entité couche AAL5. Le champ *PAD* (le champ de bourrage) et une queue de 8 octets sont ajoutés afin de constituer un CPCS-PDU (*Common Part Convergence Sublayer Protocol Data Unit*);

- la sous-couche SAR segmente le CPCS-PDU en unités de 48 octets, soit au maximum 32 cellules ATM pour une trame Ethernet. Chacune de ces unités est transmise à l'entité couche ATM pour être envoyée à un commutateur GDC connu. Nous travaillons en PVC (circuit virtuel permanent);

- le commutateur GDC de destination, grâce à la carte d'adaptation *ATM-Ethernet ETHDOC* diffuse la trame Ethernet sur le réseau Ethernet.

VI. Participations et contributions personnelles

Dans la structure du projet *P2-Verso*, nous faisons partie du niveau 2, *Conception/médiatisation*. Pour rappel, ce niveau comprend trois équipes correspondant aux trois phases présentes dans tout processus de réalisation d'un projet informatique : analyse, conception et réalisation (implémentation). Nous sommes présents plus particulièrement dans l'équipe de conception comme acteurs secondaires et dans l'équipe de réalisation comme acteurs de premier plan.

Notre travail consiste à implémenter, sous forme de modules de simulation, certains concepts de la technologie ATM. Dans le projet, une des stratégies pédagogiques adoptées est la simulation sur Web, en Java et VRML (*Virtual Reality Modeling Language*). Il est prévu, dans le module multimédia à produire, d'illustrer certains concepts de la technologie ATM à travers des animations en 3D, très conviviales.

Comme précisé précédemment, le projet comporte cinq phases ayant chacune une date butoir. Normalement, la phase de réalisation devrait avoir lieu pendant le mois de janvier de cette année. Notre stage étant limité dans le temps, à Lyon, nous avons décidé de prendre les devants. Ainsi, nous avons défini un cahier de charges pour pouvoir réaliser déjà quelques simulations. Dans ce mémoire, un chapitre est consacré à la description de ce cahier de charges. Néanmoins, ponctuellement, les autres acteurs du niveau 2 étaient tenus informés de l'avancement de notre travail, surtout en ce qui concerne les interfaces.

VII. Conclusion

Le projet *P2-Verso* associe des expertises diverses afin d'évaluer l'interaction collaborative sur un réseau ATM, dans ses dimensions technologiques et humaines. En ce qui concerne la configuration ATM, le projet a choisi une approche centrée sur l'intégration d'ATM via des ponts. Il s'agit d'interconnecter des réseaux locaux en encapsulant le trafic dans des cellules ATM (RFC 1483). Il est évident que ce choix peut avoir un impact positif ou négatif sur l'expérimentation menée[1]. Toutefois, dans ce mémoire, nous ne présentons pas les résultats du projet.

Dans les deux chapitres qui suivent, nous décrivons quelques concepts de base du JAVA et du VRML. Notons que les premiers mois de notre stage furent consacrés à l'apprentissage de ces deux langages.

---

[1] Les autres formes de mise en oeuvre sont LAN Emulation, MPOA (Multi Protocol Over ATM), etc.

Chapitre II : Technologie ATM

I. Introduction

Tous les spécialistes et opérateurs de télécommunication s'accordent à affirmer que les autoroutes d'information doivent être à large bande et donc à hauts débits pour qu'elles soient à mesure de supporter des applications de plus en plus exigeantes. Associée à la fibre optique, la technologie ATM (*Asynchronous Transfer Mode*) semble être une des solutions. Cette technologie permet de véhiculer sur une même liaison des informations de différents types : données, son, images et vidéo. C'est donc une importante technologie de l'avenir.

Nous commençons par une brève description du contexte d'apparition de la technologie ATM. Dans la suite du chapitre, nous abordons tour à tour certaines caractéristiques essentielles de cette technologie.

Pour ce chapitre, nous supposons que le lecteur possède des connaissances suffisantes sur les concepts ci-après : modes de transmission, techniques de commutation et de multiplexage, architectures et protocoles. Le contenu de ce chapitre est basé sur les documents suivants : [MELIN97], [TANEN97], [FORUM97] et [PVB97]. L'objet de ce chapitre étant de donner une brève description de la technologie ATM, nous invitons le lecteur, pour des informations plus détaillées, à consulter ces documents.

II. ATM, la technologie de commutation du RNIS LB

Vers le début des années 80, les spécialistes et les opérateurs de télécommunication se sont réunis pour définir le *Réseau Numérique à Intégration de Services* ou RNIS (en anglais ISDN, *Integrated Services Digital Network*), un nouveau système téléphonique basé sur une transmission numérique. En bref, le RNIS a été conçu pour intégrer dans un même réseau des services hétérogènes (son, données, images et vidéo lente). Du point de vue architectural, entre l'abonné et le réseau, la liaison analogique est remplacée par une liaison numérique bidirectionnelle susceptible de véhiculer des informations de plusieurs types. La première génération de RNIS, le RNIS bande étroite, supporte des débits faibles (2x64 Kbps en accès de base et 2 Mbps en accès primaire). Pour des plus amples informations, le lecteur pourra consulter [TANEN97].

Mais très vite, il s'est avéré que les applications demandaient de plus en plus des débits importants et des exigences dans la qualité de service. Pour relever ce défi, il fallait définir une deuxième génération de RNIS. Ainsi est né le RNIS large bande ou RNIS LB.

Le RNIS LB est caractérisé par une interface unique, un réseau unique et des hauts débits. Pour son réseau unique, la technologie de commutation retenue est la technologie ATM. C'est une technologie basée essentiellement sur la commutation de paquets en mode circuit virtuel. Toutes les données d'une connexion sont véhiculées sur un même circuit virtuel, garantissant ainsi l'ordre des cellules, une cellule étant un petit paquet de taille fixe (53 octets).

La technologie ATM est conçue pour remplacer la commutation de circuits du téléphone par la commutation de cellules permettant le transport, en plus de la voix, des données et de la

vidéo [TANEN97]. Dans la section qui suit, nous abordons quelques caractéristiques de cette technologie.

III. Fondements de la technologie ATM

Un réseau ATM est constitué d'un ensemble de commutateurs interconnectés par des liaisons ATM point à point, sur lesquelles sont véhiculées des cellules. Tout au tour de ce réseau sont connectés des équipements terminaux susceptibles de s'envoyer des données.

ATM est une technologie orientée connexion. Avant tout échange d'information, il y a une phase d'établissement d'un circuit virtuel de bout en bout entre l'appelant et son correspondant. ATM possède un protocole de signalisation très complexe qui ne sera pas détaillé dans ce mémoire. Il faudrait simplement noter que, pendant la phase d'appel, *la requête de connexion est propagée sur le réseau, établissant la connexion progressivement jusqu'à parvenir au destinataire* [MELIN97]. Notons aussi que lors de l'appel de son correspondant, l'appelant précise son débit moyen, son débit maximal, le type, la durée de ses rafales et la qualité de service qu'il attend du réseau. Ces éléments constituent un contrat de service entre l'appelant et le réseau. L'appelant sera tenu de gérer son débit dans les limites du contrat. A ce sujet, ATM possède un mécanisme de contrôle de trafic protégeant le réseau et l'utilisateur contre les actions pouvant dégrader les performances du réseau.

III.1. Hiérarchie des connexions logiques

Avant tout transfert d'information entre l'appelant et son correspondant, ATM crée un circuit virtuel. Ce dernier est, généralement, une concaténation de plusieurs autres connexions logiques plus simples. Dans cette sous-section, nous décrivons la hiérarchie des connexions logiques supportées par ATM.

III.1.1. Connexion VCC

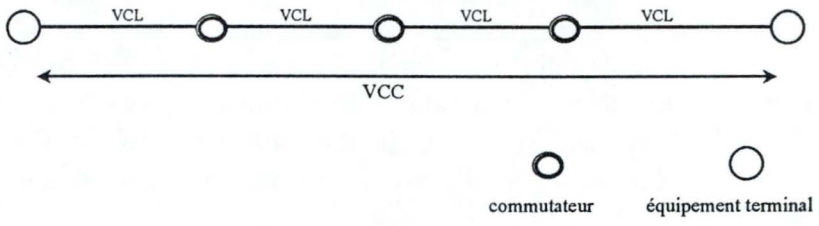


Figure 11 : *Connexion VCC*

Lors d'une demande de connexion, si la destination et le réseau acceptent la requête de connexion, un circuit virtuel est activé entre l'appelant et son correspondant. Ce circuit est appelé *connexion VCC* (*Virtual Channel Connection*).

Une connexion VCC, aussi appelée *ATM connection*, fournit un chemin de bout en bout assurant le transfert unidirectionnel ordonné mais non garanti de cellules entre utilisateurs de la couche ATM. Une connexion VCC peut être constituée d'un ou de plusieurs liens VCL (*Virtual Channel Link*). La Figure 11 illustre le concept de connexion VCC.

III.1.2. Lien VCL

Un lien VCL est un moyen de transfert unidirectionnel de cellules entre un point où un VCI (*Virtual Channel Identifier*) est assigné à une cellule et un point où cette valeur est supprimée ou modifiée. Ces points sont des commutateurs ou des équipements terminaux. Notons qu'au sein d'un lien VCL, le VCI n'est pas modifié.

Plusieurs liens VCL reliant deux équipements terminaux ou commutateurs peuvent être regroupés dans une connexion VPC (*Virtual Path Connection*).

III.1.3. Connexion VPC

Une connexion VPC fournit un chemin assurant un transfert ordonné mais non garanti de cellules entre deux points où les VCI sont créés ou modifiés. Ces points sont des commutateurs ou des équipements terminaux.

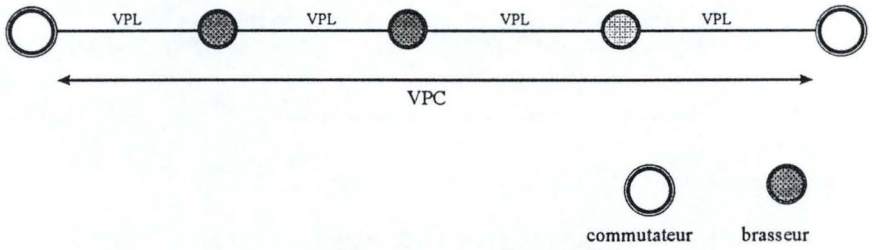


Figure 12 : *Connexion VPC*

En dehors du commutateur, il existe un deuxième type d'équipement capable de réaliser la commutation dans un réseau ATM : le *brasseur*. Comme nous verrons plus loin, le brasseur, pour commuter une cellule, se base sur le champ VPI (*Virtual Path Identifier*) de la partie en-tête de la cellule. Lorsque des brasseurs sont utilisés, une connexion VPC peut être la concaténation de plusieurs liens VPL (*Virtual Path Link*). La Figure 12 illustre le concept de connexion VPC.

Notons que, à la demande de l'utilisateur, une connexion VPC peut être établie de bout en bout. Dans ce cas, la création et la suppression de connexions VCC sont laissées à l'initiative de l'utilisateur.

III.1.4. Lien VPL

Un lien VPL est un moyen de transfert unidirectionnel de cellules entre un point où un VPI est assigné à une cellule et un point où cette valeur est supprimée ou modifiée. Ces points sont des équipements terminaux, des commutateurs ou des brasseurs. A l'intérieur d'un lien VPL, le VPI n'est pas modifié.

Pour des plus amples informations sur cette hiérarchie, le lecteur pourra consulter les manuels cités au début de ce chapitre.

III.2. Cellule ATM

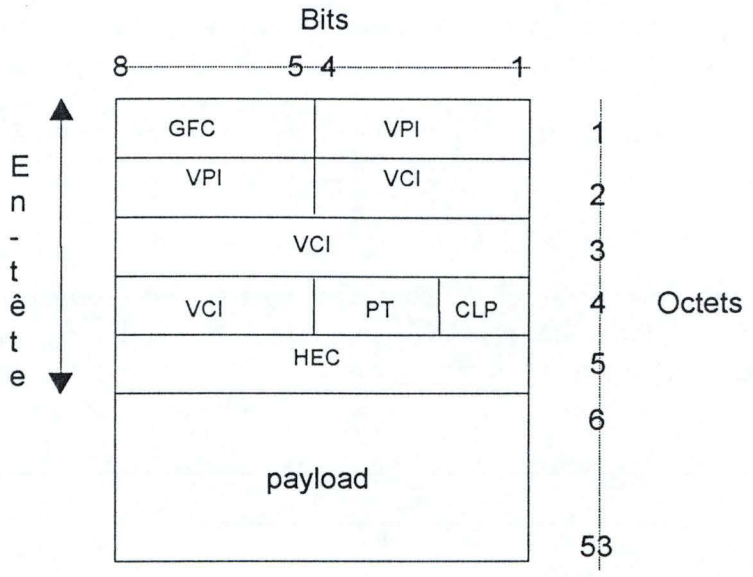


Figure 13 : *Format d'une cellule ATM*

Une cellule possède une taille fixe de 53 octets (Figure 13). Le format d'une cellule diffère légèrement selon que la cellule est entre l'utilisateur et son noeud d'entrée au réseau ATM (dans l'interface UNI, *User to Network Interface*) ou entre deux équipements intermédiaires (commutateurs ou brasseurs) du réseau (dans l'interface NNI, *Network to Network Interface*). Voici brièvement les significations des différents champs formant la structure d'une cellule :

- GFC, *Generic Flow Control* (4 bits) : un champ utilisé à des fins de contrôle de flux et de prévention contre la congestion. Ce champ est présent dans la cellule uniquement dans le cas d'une interface UNI. Dans le cas d'une interface NNI, il fait partie du champ VPI.

- VPI, *Virtual Path Identifier* (8 ou 12 bits) : un champ identifiant le lien VPL. Dans le cas d'une interface UNI, le champ VPI possède 8 bits alors qu'il en a 12 dans le cas d'une interface NNI.

- VCI, *Virtual Channel Identifier* (16 bits) : un champ identifiant le lien VCL sur lequel transite la cellule.

- PT, *Payload Type* (3 bits) : un champ utilisé pour distinguer les cellules des usagers des cellules de service.

- CLP, *Cell Loss Priority* (1 bit) : un champ intervenant dans les mécanismes contre la congestion. Les cellules de priorité haute sont positionnées à 0 alors que celles de priorité basse sont à 1. Les cellules de priorité basse sont détruites en premier lors d'une congestion.

- HEC, *Header Error Control* (8 bits) : un champ permettant de détecter des erreurs pouvant survenir sur la partie en-tête de la cellule.

- *Payload* (48 octets) : un champ contenant le PDU (*Protocole Data Unit*) fourni par l'entité couche AAL de l'émetteur.

III.3. Commutation de cellules

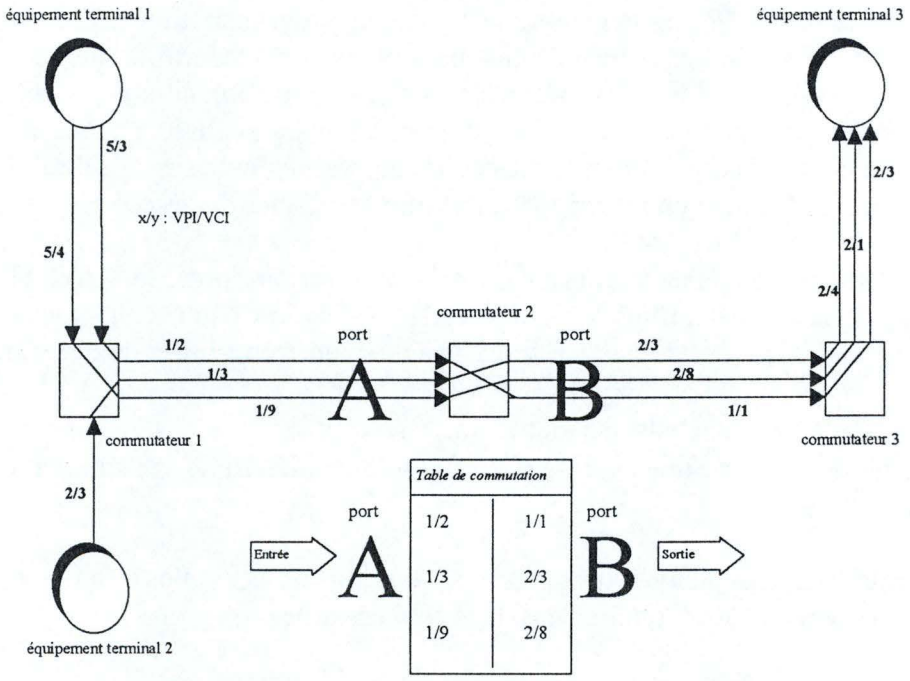


Figure 14 : *Table de commutation et acheminement des cellules*

Pour commuter une cellule, le commutateur ATM se base sur le couple (VPI, VCI) de l'en-tête de la cellule alors que le brasseur ne s'appuie que sur le champ VPI. Dans la suite de ce mémoire, lorsque nous parlons de la commutation ATM, nous faisons allusion aux commutateurs ATM.

Tout commutateur impliqué dans une connexion VCC donnée possède une entrée dans sa table de commutation lui permettant de faire acheminer les cellules appartenant à la connexion (Figure 14). Dans la littérature, cette table est appelée parfois *table de translation* ou *table de routage*. Dans ce mémoire, nous préférons parler de table de commutation. Une table de commutation indique, en fonction du port d'entrée et du couple (VPI, VCI) de la partie en-tête d'une cellule, le port de sortie et le nouveau couple (VPI, VCI) à affecter à la cellule. Notons que, dans la technologie ATM, la commutation est matérielle (non logicielle). Et ceci permet d'atteindre des débits très élevés.

III.4. Multiplexage de cellules

ATM est aussi une technologie de multiplexage. Une liaison ATM est fondamentalement une liaison point à point et unidirectionnelle. Les cellules se suivant sur une liaison ATM peuvent être d'origines diverses. Toutes les cellules d'un circuit virtuel donné possèdent, sur une liaison ATM, un même couple (VPI, VCI) les identifiant par rapport aux autres cellules. C'est

du multiplexage temporel dynamique. Ce mécanisme permet de véhiculer sur une même liaison des informations de différents types. Le fait que la technologie ATM soit asynchrone rend facile son utilisation. Etant donnée qu'il n'existe aucun lien entre l'information véhiculée et le temps, il est possible de multiplexer, dans le temps, plusieurs connexions sur une même liaison ATM. Notons que le flux des cellules émises par une application n'est pas forcement constant. L'émetteur transmet les cellules de façon totalement indépendante du récepteur.

III.5. Modes de connexion

Dans ATM, une connexion est soit permanente soit commutée. Une connexion permanente ou PVC (*Permanent Virtual Connection*) nécessite la programmation manuelle des différents commutateurs pour désigner les routes possibles entre les différents équipements. Les connexions sont donc prédéfinies. Dans le cas d'une connexion commutée ou SVC (*Switched Virtual Connection*), la connexion VCC est établie lors de la phase d'appel. Il est évident que seul le mode de connexion commutée nécessite un protocole de signalisation. Le protocole de signalisation ATM utilise un plan d'adressage afin de permettre des connexions à la demande.

Un plan d'adressage, dans un protocole de signalisation (protocole d'établissement d'une connexion), permet d'identifier la source et la destination d'une connexion. ATM possède plusieurs formats d'adresses mais qui sont, généralement, constitués de deux grandes parties :

- un préfixe réseau : une adresse identifiant le réseau;
- une partie utilisateur : une adresse identifiant de manière unique l'équipement terminal sur un réseau donné.

Les informations supplémentaires sur ce mécanisme complexe sont à trouver dans [MELIN97]. Nous ne les détaillons pas dans ce mémoire.

III.6. Protocoles de routage

Dans la phase d'appel, en plus d'un plan d'adressage, un réseau ATM nécessite un protocole de routage pour pouvoir aiguiller les requêtes de signalisation. En toute généralité, un protocole de routage est un mécanisme permettant, lors d'une demande de connexion, de déterminer une route reliant l'appelant et son correspondant. Une route est une succession de commutateurs. Actuellement, sur des réseaux privés, il existe deux protocoles de routage : IISP (*Interim Inter-Switch protocol*) et PNNI (*Private Network-to-Network Interface*). Normalement, le deuxième devrait remplacer le premier.

IISP est un protocole de routage statique. Les commutateurs ATM possèdent des tables d'adresses préconfigurées. Une table d'adresses indique le port de sortie à emprunter pour atteindre un équipement terminal donné. Ce protocole statique ne supporte pas la qualité de service. En outre, la redondance de liaisons n'est pas envisageable. Dans une table d'adresses donnée, l'adresse d'un équipement terminal n'est reprise qu'une seule fois.

Le protocole PNNI est beaucoup plus souple. Sur base des critères fournis lors de la phase d'appel (la bande passante, le délai de transit, etc.), il est capable de sélectionner une route adéquate permettant de relier l'appelant et son correspondant. Ce protocole supporte la qualité de service.

III.7. Couches

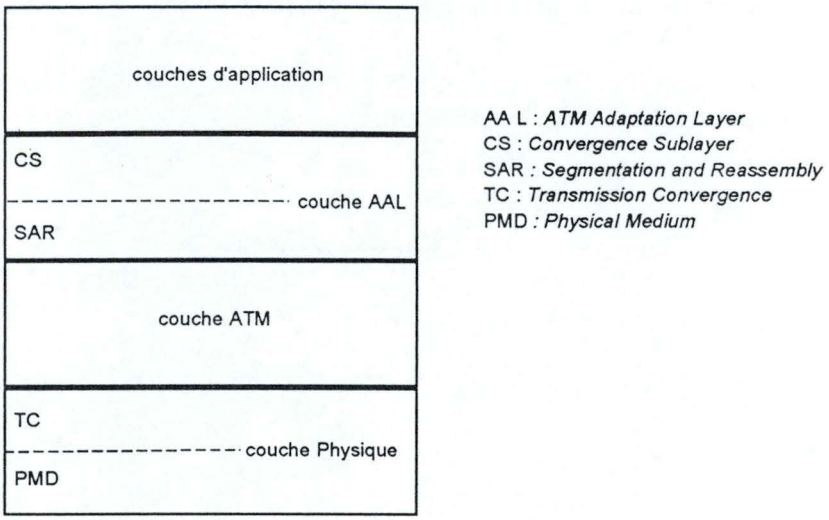


Figure 15 : *Découpe en couches*

Avant de clore ce chapitre, nous rappelons brièvement la découpe en couches d'un réseau ATM. Cette découpe offre quatre couches (Figure 15). La couche la plus basse est la couche physique. Au-dessus d'elle, nous retrouvons la couche ATM, responsable de la commutation. Les services fournis par cette couche sont utilisés par la couche AAL (*ATM Adaptation Layer*), responsable de l'adaptation. Dans la couche la plus haute sont regroupées toutes les applications utilisant les services fournis par un réseau ATM, y compris les protocoles d'interconnexion tels que IP et de transport tels que TCP.

III.7.1. Couche AAL

La couche AAL, dans un réseau ATM, s'occupe d'une part de la segmentation du flux d'information en provenance des applications en fragments de 48 octets, et d'autre part, de réassembler les fragments en flux d'information à envoyer aux applications. Pour supporter plusieurs types d'application (son, données, images et vidéo), il existe quatre types d'AAL, chacun d'eux possédant des spécificités propres. Ces sous-couches rendent chacune un type de service particulier. Les détails sur la couche AAL sont à trouver dans [TANEN97] ou [MELIN97]. Notons que les équipements non terminaux ne possèdent pas une entité couche AAL sauf pour des besoins de la signalisation.

III.7.2. Couche ATM

Au-dessous de la couche AAL se trouve la couche ATM. C'est elle qui s'occupe du transport d'une cellule de bout en bout. Une entité couche ATM assure :

- la commutation et le multiplexage des cellules;
- l'extraction de l'en-tête avant de transmettre la partie donnée à l'entité couche AAL;
- l'ajout ou la modification de l'en-tête avant de transmettre la cellule à l'entité couche physique;
- le contrôle de flux par l'intermédiaire du champ GFC dans le cas d'une interface UNI.

IV. Conclusion

Dans ce chapitre, nous venons de rappeler quelques concepts de la technologie ATM. Nous avons choisi d'insister sur certains concepts plutôt que sur d'autres simplement parce que, dans notre application, nous n'illustrons que quelques fonctionnalités de cette technologie.

Dans les deux chapitres qui suivent, nous présentons brièvement les deux langages qui sont utilisés dans l'implémentation de l'application.

Chapitre III : JAVA

I. Introduction

Dans l'implémentation de l'application, nous utilisons le langage Java. Ce chapitre, sur base de [ANUFF96], [BONJOU96] et [BOUZE97], fait un bref rappel sur quelques éléments du langage Java tel que les types de base, la notion de classe d'objets et d'autres concepts essentiels du langage que nous utilisons dans l'implémentation de l'application. En dehors de ces livres, nous avons exploré quelques sites spécialisés en Java. Une liste de ces sites est reprise dans la partie bibliographie de ce mémoire. L'objet de ce chapitre étant de donner une brève description du langage, nous invitons le lecteur, pour des informations plus détaillées, à consulter la partie bibliographie de ce mémoire.

II. Eléments de base du langage

Java possède les types de données de base suivants :

1. les types d'entiers :
   - ***byte*** (8 bits)
   - ***short*** (16 bits)
   - ***int*** (32 bits)
   - ***long*** (64 bits)

   Il faudrait noter qu'un nombre entier peut s'exprimer soit en décimal (ne doit jamais commencer par un 0) soit en octal (doit commencer par un 0) ou soit en hexadécimal (doit commencer par 0x).

2. les types de nombres flottants :
   - ***float*** (32 bits)
   - ***double*** (64 bits)

3. le type caractère :
   - ***char*** (16 bits non signés c'est-à-dire entre 0 et 65535)

   Le caractère doit être entouré de deux apostrophes. C'est un caractère de 16 bits au format *unicode* c'est-à-dire une valeur entière non signée (comprise ente 0 et 65535).
   Exemples : 'A', '1', '*'

4. le type booléen :
   - ***boolean***

   Le type booléen est non convertible en numérique. Les mots-clés ***true*** et ***false*** définissent respectivement les valeurs de vérité *vrai* et *faux*.

Pour construire un nouveau type, la seule possibilité offerte par Java est celle de définir une classe. Les types dérivés de Java sont donc des classes d'objets. Dans notre module, nous utilisons les types dérivés suivants :

5. le type chaîne de caractères :
   - ***String***

Une chaîne de caractères est une suite de caractères entourés de guillemets. Elle n'est pas un tableau de caractères.

Exemples : "Java ", "Simulateur ATM", "1998"

6. le type vecteur :
   . ***Vector***

   Le type vecteur est une classe de tableaux de taille variable, pouvant contenir n'importe quel objet. L'indice du premier élément d'un vecteur est 0.

Remarques :

- En Java, les conversions de types doivent être exprimées explicitement. Ceci permet d'éviter les erreurs d'inattention au moment de la programmation;
- La taille des types ne dépend pas du système d'exploitation.;
- Une constante est une variable définie avec le modificateur d'accès ***final***. Comme nous le verrons plus loin, un modificateur d'accès précise le degré d'accessibilité d'une variable ou d'une méthode. Le mot-clé ***final*** indique que le contenu de la variable ne pourra être modifié après sa première affectation.

III. Classes et méthodes

Dans la philosophie objet, un programme est constitué d'entités actives composées de structures de données cachées par des fonctions. Ces entités actives ou objets dialoguent via des échanges des messages. Un message est une demande d'exécution d'une opération sur un objet. Toutes les opérations possibles applicables sur un objet sont définies dans sa classe d'appartenance.

III.1. Classe

Formellement, une *classe* est un type abstrait spécifiant les attributs, les opérations applicables à ses instances et le mécanisme de création de ses instances. Les instances d'une classe sont appelées des objets. Tout objet possède une propriété le distinguant des autres objets. C'est son identifiant. Il est généré automatiquement.

En Java, le code source définissant une classe est placé dans un fichier portant le même nom que la classe. Par exemple, la classe *Avion* doit être placée dans le fichier *Avion.java*. Toutefois, il est possible de mettre dans un même fichier, plusieurs classes. Dans ce cas, il ne peut y avoir plus d'une classe déclarée ***public*** (c'est-à-dire accessible de partout). Il faut noter qu'il n'est pas permis de définir une classe à l'intérieur d'une autre classe.

III.2. Méthode

Une *méthode* peut être définie comme une unité de comportement d'un objet définie dans sa classe d'appartenance. Elle est une action élémentaire applicable à un objet et possède une signature. Une signature est l'ensemble formé par le nom de la méthode, le type du résultat et la liste de paramètres. La liste de paramètres peut être composée de 0 ou de plusieurs paramètres. Le type de chaque paramètre doit être spécifié.

Une méthode est :

1. soit *abstraite* : elle ne possède pas de corps d'instructions. Une classe contenant au moins une méthode abstraite est abstraite et toutes ses sous-classes concrètes doivent implémenter toutes ses méthodes abstraites;
2. soit *concrète* : elle possède un corps d'instructions. Une classe est concrète si elle ne contient et n'hérite d'aucune méthode abstraite.

Une classe concrète peut posséder une méthode permettant d'instancier les objets de la classe : le *constructeur* de la classe. Un constructeur ne peut avoir de type de résultat et son nom est identique à celui de la classe. Et il est possible de définir plusieurs constructeurs pour une même classe. Toutefois ces constructeurs porteront le même nom mais auront des paramètres d'appel différents.

Pour créer une instance d'une classe, il faut invoquer le constructeur à l'aide de l'opérateur ***new***. En effet, c'est l'opérateur ***new*** qui crée un objet de la classe et appelle le constructeur dont les types des paramètres correspondent aux types des arguments de la liste.

A titre d'exemple, voici une partie de la liste de méthodes de la classe ***Vector*** :

- ***Vector()*** est le constructeur par défaut de la classe;
- ***addElement(UnObjet)*** ajoute au vecteur l'élément dont la référence est passée en paramètre;
- ***removeElement(UnObjet)*** supprime du vecteur l'élément dont la référence est passée en paramètre;
- ***elementAt(UnIndex)*** accède à l'élément occupant la position passée en paramètre;
- ***contains(UnObjet)*** vérifie l'appartenance d'un objet au vecteur et renvoie ***true*** si l'objet appartient au vecteur et ***false*** dans le cas contraire;
- ***firstElement()*** accède au premier élément du vecteur;
- ***lastElement()*** accède au dernier élément du vecteur;
- ***removeAllElement()*** supprime tous les éléments du vecteur;
- ***removeElementAt(UnIndex)*** supprime l'élément occupant la position passée en paramètre.

En Java, les variables désignant des objets sont des références. C'est le seul mode de désignation. Java n'utilise pas de pointeur (une donnée contenant généralement un entier désignant un emplacement mémoire) pour désigner un objet pour des raisons d'efficacité et de sécurité. Une référence n'est pas modifiable par une méthode. Une référence passée comme paramètre à une méthode désigne toujours le même objet, avant et après exécution de la dite méthode. C'est un passage par adresse.

Dans la définition d'une méthode, il n'est pas permis de spécifier qu'un paramètre est une constante et un paramètre ne peut avoir une valeur par défaut. Le corps de la méthode doit se trouver dans la définition de sa classe d'appartenance. Ne travaillant pas sur des pointeurs, Java permet qu'une méthode et un attribut définis dans une même classe portent un même nom.

Pendant l'exécution, une liste d'objets encore référencés par le processus courant est tenue et tout objet non référencé est supprimé de la mémoire par un processus spécial, *ramasse-miettes*. Le programmeur ne s'occupe donc pas de la destruction des objets. Ce processus spécial s'occupe aussi du compactage de la mémoire. Toutes ces opérations s'effectuent parallèlement au déroulement normal du programme courant.

III.3. Redéfinition et surcharge

Java fait la distinction entre redéfinition et surcharge. Une méthode d'une sous-classe redéfinit une méthode de sa super-classe si les deux possèdent les mêmes paramètres (mêmes signatures). Dans le cas contraire, il s'agit d'une surcharge. La surcharge d'une méthode est le fait de lui associer plusieurs définitions. Dans la redéfinition, seul le corps de la méthode change. Il importe de signaler que Java effectue une liaison dynamique à l'exécution dans le cas d'une redéfinition : il choisit la bonne méthode en fonction de la classe réelle de l'objet et pas en fonction du type déclaré dans le code source. Une telle méthode est dite polymorphe. Le polymorphisme est la faculté d'une méthode de pouvoir s'appliquer à des objets de classes différentes.

III.4. Accessibilité

En Java, il est possible de spécifier le degré d'accessibilité d'un attribut ou d'une méthode à l'aide d'un des mots-clés suivants : ***public***, ***private*** et ***protected***. Ces mots-clés sont appelés *modificateurs d'accès*. L'accès à un attribut ou à une méthode peut être soit limité à sa classe, soit étendu aux autres classes du package, soit étendu à n'importe quelle classe de n'importe quel package (la notion de package est expliquée plus loin, dans la section suivante).

Une méthode peut être déclarée :

- ***public*** : la méthode est accessible à n'importe quelle classe de n'importe quel package;
- ***private*** : la méthode est accessible seulement aux instances de la classe;
- ***private protected*** : la méthode est accessible à toute sous-classe du même package;
- ***protected*** : la méthode est accessible à toute sous-classe.

Remarquons que, par défaut, l'accessibilité de la méthode se limite au package. Comme pour une méthode, un attribut peut être déclaré ***public***, ***private***, ***private protected*** ou ***protected***.

Une classe peut être déclarée publique à l'aide du mot-clé ***public***. Dans ce cas, elle est accessible à n'importe quel package.

III.5. Héritage et accessibilité

L'héritage suppose la présence d'une super-classe et d'une ou plusieurs sous-classes. Une super-classe est une classe étendue par une autre classe appelée sous-classe. Une sous-classe est une classe dérivée à partir d'une autre classe appelée super-classe. Cependant, une sous-classe ne peut étendre qu'une et une seule classe, l'héritage multiple n'étant pas autorisé.

Normalement, les attributs d'une classe sont toujours hérités par ses sous-classes. Toutefois, en Java, l'héritage n'implique pas toujours l'accessibilité. Un attribut déclaré ***private*** reste inaccessible dans les sous-classes. Dans le cas où aucun modificateur d'accès n'aurait été spécifié, les sous-classes définies dans d'autres packages hériteront de l'attribut mais ce dernier restera inaccessible. Mais il peut être accédé via une méthode d'accès définie au niveau de la super-classe.

Une méthode déclarée ***private*** n'est pas héritée par les sous-classes. Si elle est définie sans modificateur d'accès, elle ne sera pas héritée par les sous-classes définies dans d'autres

packages. Une méthode ou un attribut déclaré ***protected*** ou ***private protected*** est accessible dans n'importe quelle sous-classe de n'importe quel package.

Une classe déclarée ***final*** ne peut être étendue et une méthode déclarée ***final*** ne peut être redéfinie dans une sous-classe, avec les mêmes paramètres.

IV. Architecture en modules

Une application peut être organisée en modules. La découpe d'une application en modules peut se faire sur base d'un critère fonctionnel, d'un critère de domaine ou d'un quelconque autre critère. Un module, en Java, est appelé ***package*** et peut contenir une ou plusieurs classes, une classe n'appartenant qu'à un seul package. Le nom du package contenant la classe est inscrit au début de son code source, après le mot-clé ***package***.

Dans jdk1.1.4 (*Java Development Kit*), le compilateur Java mis à notre disposition, les fichiers sources et les fichiers compilés doivent se trouver dans un répertoire ayant le même nom que le package.

V. Concurrence des processus

Cette section décrit brièvement les threads Java. Pour des plus amples informations sur ce concept, nous invitons le lecteur à consulter une documentation appropriée telle que [BONJOU96].

V.1. Concept de thread

Dans un système d'exploitation, nous distinguons deux catégories de processus :

1. les processus lourds qui sont des activités au sein du système d'exploitation;
2. les processus légers ou threads qui sont des sous-activités des processus lourds. Un processus lourd peut être *multithreads*.

Dans ce mémoire, nous faisons référence à des processus légers lorsque nous parlons de processus. Un processus Java est une instance de la classe ***Thread***. Le concept de thread fait partie intégrante du langage Java. Cette classe fournit au programmeur une série de méthodes natives dont voici quelques unes :

- ***Thread()*** est un des constructeurs de la classe;
- ***start()*** lance l'exécution de la méthode ***run()*** du processus;
- ***run()*** contient ce que doit faire le processus et est appelée après le démarrage du processus;
- ***isAlive()*** renvoie ***true*** si le processus est actif;
- ***Suspend()*** suspend le processus;
- ***resume()*** redémarre un processus suspendu.

Java propose deux manières de créer un processus :

1. Définir une sous-classe de la classe ***Thread*** et redéfinir la méthode ***run()***.

Exemple :

```
class ClassAnimation  extends Thread
{
/* déclaration des attributs */
 ...
/* spécification éventuelle d'un ou de plusieurs constructeurs */
 ...
public void run()
   {
   /* ce que doit faire le processus */
      ...
   }
}
```

La classe ClassAnimation hérite de toutes les méthodes de la classe ***Thread*** et seule la méthode ***run()*** est redéfinie. Pour créer une instance de cette sous-classe et lancer son exécution, il suffit de procéder de la manière suivante :

```
ClassAnimation Animation = new ClassAnimation ();
Animation.start(); /* lance le processus Animation qui va
                      exécuter la méthode run() */
```

2. Implémenter une interface ***Runnable***. Une interface Java est une définition de type. Elle déclare un ensemble de constantes (attributs déclarés ***final***) et de méthodes abstraites. Lorsqu'une classe déclare implémenter une interface, elle s'engage à redéfinir toutes les méthodes définies dans l'interface. Pour contourner la restriction selon laquelle une sous-classe ne peut étendre plus d'une super-classe, la solution est d'utiliser les interfaces Java. Une classe peut implémenter plusieurs interfaces. Ainsi, par exemple, une classe peut hériter de la classe ***Thread*** et d'une autre classe telle que la classe ***Applet*** (ce concept est décrit plus loin, dans ce même chapitre).

Exemple :

```
class MonApplet extends Applet implements Runnable
{
/* déclaration des attributs */
Thread Animation; /* attribut Animation de type Thread */
 ...
public void start()
   {
   Animation = new Thread(this); /*création d'une instance de la
                                   classe Thread */
   Animation.start (); /*démarrage du processus Animation */
   }
public void run()
   {
   /*ce que doit faire le processus Animation */
      ...
   }
 ...
```

}

Une applet implémentant l'interface ***Runnable*** doit redéfinir la méthode ***run()*** de la classe ***Thread*** pour le processus qu'elle va activer. Dans l'exemple ci-dessus, le paramètre ***this*** passé au constructeur de la classe ***Thread*** exprime le fait que l'objet cible ***Runnable*** du processus *Animation* est l'applet elle-même. Par conséquent, le processus *Animation* exécutera la méthode ***run()*** de l'applet. On remarquera que la classe *MonApplet* hérite de la classe ***Applet*** et de la classe ***Thread***. Le processus exécutant *MonApplet* et le processus *Animation* s'exécutent d'une manière concurrente.

V.2. Cycle de vie d'un processus Java

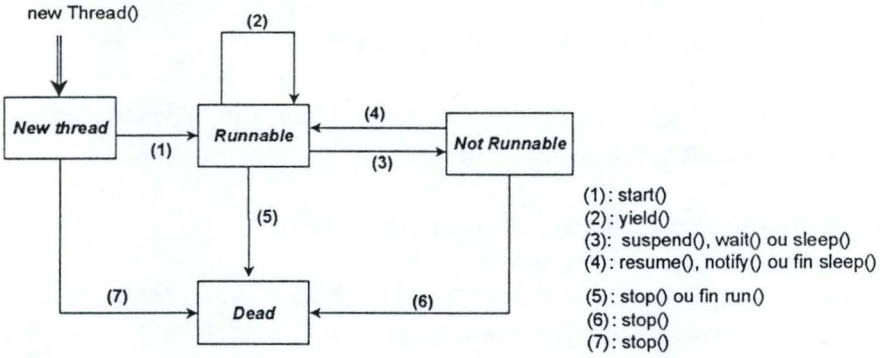


Figure 16 : *Cycle de vie d'un processus*

La Figure 16 trace le cycle de vie d'un processus Java. Il existe quatre états possibles :

- état *New thread* (nouveau né) : le processus vient d'être créé. A ce stade, aucune ressource ne lui est attribuée. Ce processus peut être :
    1. soit démarré par l'application de la méthode ***start()*** qui lui affecte les ressources systèmes nécessaires à son exécution (1);
    2. soit arrêté par l'application de la méthode ***stop()*** (7).

- état *Runnable* (exécutable) : le processus est soit en cours d'exécution soit dans la file d'attente. Lorsqu'il aura le processeur, les instructions contenues dans sa méthode ***run()*** vont être exécutées d'une façon séquentielle. A la fin de son exécution normale ou suite à l'application de la méthode ***stop()***, le processus entre dans l'état *Dead* (5). La méthode ***yield()*** permet à un processus en cours d'exécution d'abandonner provisoirement le processeur et de se mettre à la fin de la file d'attente (2).

- état *Not Runnable* (bloqué) : un processus *Runnable* devient *Not Runnable* dans les quatre cas suivants (3) :

    1. suite à l'application de la méthode ***sleep()***;
    2. suite à l'application de la méthode ***suspend()***;
    3. suite à l'application de la méthode ***wait()*** (pour attendre la réalisation d'un événement);
    4. suite au blocage causé par une opération d'entrée/sortie.

Pour chacun de ces quatre cas, il existe une manière de revenir à l'état *Runnable* :

1. le processus termine le temps passé en paramètre à la méthode ***sleep()*** ;
2. le processus suspendu est réactivé par l'application de la méthode ***resume()*** ;
3. le processus en attente d'un événement est réactivé par la méthode ***notify***(). Cette méthode doit être exécutée par l'objet générateur de l'événement attendu par ce processus;
4. le processus bloqué par une opération d'entrée/sortie est réactivé à la fin de cette opération.

Toutefois, un processus *Not Runnable* peut être stoppé (6) et passé dans un état *Dead*.

- état *Dead* (mort) : un processus meurt soit à la fin de son exécution normale soit après application de la méthode ***stop()***. Mais il est déconseillé d'utiliser la méthode ***stop()*** de peur que le programme se retrouve dans un état inconsistant. Un processus *Dead* libère toutes les ressources mises à sa disposition.

La méthode ***isAlive()*** appliquée à un processus renvoie vrai si le processus est *Runnable* ou *Not Runnable* et faux dans le cas contraire.

V.3. Moniteur, synchronisation et priorité

Le programmeur a la possibilité de gérer efficacement des exécutions concurrentes grâce notamment au mécanisme de synchronisation. La synchronisation, en Java, se réalise au travers de l'utilisation des moniteurs. Une section critique est marquée par le mot-clé ***synchronized***. Généralement, ces sections critiques sont des méthodes. Toutefois, il y a moyen de synchroniser une instruction ou un bloc d'instructions.

Un moniteur est associé à chaque objet possédant au moins une méthode synchronisée. L'accès à un tel objet est exclusif : seul le processus contrôlant le moniteur associé à l'objet peut y accéder. Les autres processus ne peuvent pas appeler une méthode synchronisée de ce même objet. Si l'objet est déjà réservé par un autre processus, ***synchronized*** met les processus demandeurs en attente et lorsque l'objet est libéré, en général, le prochain processus à disposer du moniteur est choisi selon un principe FIFO (*first in first out*). Il faut savoir que l'accès à une méthode non synchronisée d'une classe associée à un moniteur n'est évidemment pas synchronisé.

Un processus nouvellement créé hérite de la priorité du processus qui l'a créé. Cette priorité peut être changée à l'aide de la méthode ***setPriority()***. La valeur de la priorité doit se situer entre ***MIN_PRIORITY*** et ***MAX_PRIORITY***. En effet, la classe ***Thread*** possède trois variables statiques (un seul exemplaire partagé par toutes les instances de la classe ***Thread***) et finales (constantes) : ***MIN_PRIORITY***=1, ***NORM_PRIORITY***=5 et ***MAX_PRIORITY***=10. Le processus qui prendra le processeur au prochain cycle est toujours celui possédant la plus grande priorité. Dans le cas des priorités égales, le principe FIFO est appliqué.

L'exécution d'un processus se déroule d'une façon ininterrompue, de la première instruction jusqu'à la dernière, tant que :

- il n'exécute pas la méthode ***yield()*** ;
- il ne devient pas *Not Runnable* ni *Dead* ;
- il n'y a pas un processus plus prioritaire qui devient *Runnable*.

Pour ne pas bloquer les processus les moins prioritaires, il est recommandé d'insérer la méthode ***yield()*** dans les processus les plus prioritaires afin d'abandonner quelquefois le monopole du processeur. Pour notre application, nous utilisons plutôt la méthode ***sleep()***. Ceci nous permet à la fois de donner le processeur à tous les processus et de ralentir la vitesse d'exécution de l'application.

Il importe de signaler que la synchronisation ne supprime nullement la possibilité qu'il y ait interblocage. Il faudrait donc analyser toutes les dépendances possibles entre les différents processus concurrents.

VI. Applet

Une applet est un petit programme Java devant être invoqué à partir de sa référence incluse dans une page HTML (*Hyper Text Markup Language*). Formellement, une applet Java est une classe et plus particulièrement une sous-classe de la classe ***Applet*** fournie par les API (*Application Programming Interface*) Java. Pour créer une applet, généralement on redéfinit les méthodes ***init()***, ***start()***, et ***paint()***.

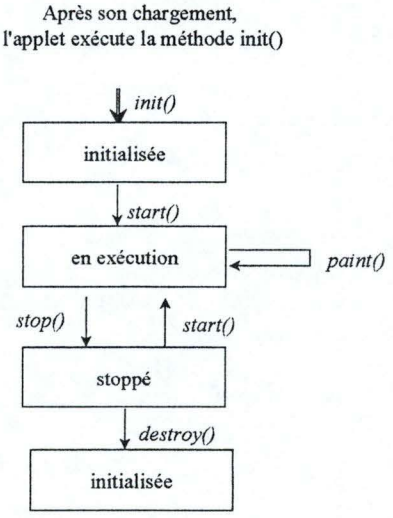


Figure 17 : *Cycle de vie d'une applet*

La méthode ***init()*** contient les initialisations nécessaires à la vie de l'applet. Cette méthode est appelée par le butineur ou browser (plus précisément par un processus contrôlé par le browser) lors du chargement de l'applet. Une fois l'initialisation effectuée, la méthode ***start()*** est exécutée. La Figure 17 décrit le cycle de vie d'une applet.

La méthode ***stop()*** est exécutée chaque fois que la fenêtre dans laquelle s'affiche l'applet n'est plus active. Si la page contenant l'applet redevient visible, la méthode ***start()*** est exécutée automatiquement. La méthode ***destroy()*** est lancée lorsque le browser ferme la page contenant l'applet et toutes les ressources qui lui étaient accordées sont récupérées.

La méthode ***paint()*** renferme toutes les opérations d'affichage à l'écran. Cette méthode est invoquée après l'exécution de la méthode ***start()***.

Une applet Java est contrôlée par un processus du browser. C'est ce processus qui appelle les méthodes gérant l'applet. Mais très souvent, on voudrait qu'une applet puisse réaliser une action de manière continue. Pour effectuer par exemple une animation, la solution consiste à définir un autre processus à l'intérieur même de l'applet. Ce processus va, par exemple, forcer la fenêtre de l'applet à se redessiner toutes les *x* secondes.

Voici le squelette général d'une applet d'animation :

```
/* APPLET d'animation type */
import ... /* Déclaration des packages ou classes importés */
public class MonApplet extends Applet implements Runnable
  {
  Thread Animation ;
  /* déclaration des variables de l'applet */
    ...
  public void init()
  {
  /* les initialisations */
     ...
  }

  public void start()
  {
  /*--- activation du processus d'animation --------*/
  if (Animation == null)
        {
       Animation = new Thread(this) ;
       Animation.start();
        }
  }

  public void stop()
  {
  /* arrêt du processus d'animation */
  if (Animation != null)
        {
      Animation.stop();
      Animation = null ;
      }
  }

  public void run()
  {
  /* déclaration des variables du processus */
     ...
  /* --------- boucle d'animation ------------ */
  while (true)
         {
         /* modification des variables du processus */
            ...
         repaint();     /*demande d'affichage */
           /*-- réglage de la vitesse d'animation -------*/
         try {Thread.sleep(100);}    /* 100 ms */
        catch (InterruptedException signal) {}
        }
  }

  public void paint(Graphics g)
  {
  /* affichage de l'animation */
```

```
        ...
        }
    }
```

Dans l'exemple ci-dessus, en plus des méthodes classiques d'une applet, il y a la méthode ***run()***. Pour rappel, une applet implémentant l'interface ***Runnable*** doit redéfinir la méthode ***run()*** pour le processus qu'elle va activer. Dans la méthode ***run()***, après la modification des variables liées au processus *Animation*, la méthode ***repaint()*** redessine la fenêtre de l'applet. Pour ne pas redessiner continuellement la fenêtre de l'applet, nous avons ajouté la méthode ***sleep()*** de la classe ***Thread***. Elle met le processus *Animation* en veille pendant 100 millisecondes. Le processus exécutant l'applet et le processus *Animation* s'exécutent en parallèle.

VII. Autres particularités

Tiré du C et de C++, Java possède plusieurs spécificités propres. Dans cette partie du mémoire, nous présentons quelques autres différences essentielles par rapport à ces deux langages.

En Java, une variable d'un type simple non initialisée ne contient aucune valeur et son utilisation comme telle provoquera une erreur à la compilation. Ceci est aussi valable pour une variable de type objet. Concernant l'instruction ***for***, elle est considérée comme étant un bloc par Java. Toute variable déclarée dans ce bloc a une portée limitée à ce bloc.

Lors de la définition d'une classe, Java autorise l'initialisation des variables d'instance et de classe. Une variable de classe est une variable commune à toutes les instances de la classe. Elle est déclarée à l'aide du modificateur ***static***. Les variables d'instance sont initialisées, lors de l'instanciation, avant l'appel du constructeur.

Java possède un mécanisme de vérification des bornes des vecteurs. Tout accès en dehors des bornes est refusé.

VIII. Compilation

Java est un langage à la fois compilé et interprété. Le compilateur Java génère du *pseudo-code*, un code intermédiaire représentant le jeu d'instructions d'une machine virtuelle. Le compilateur Java est ***javac***. Pour compiler, par exemple, la classe *trame*, il faudrait lancer la commande suivante :

javac trame.java

La compilation fournit un fichier ayant l'extension **.class**.

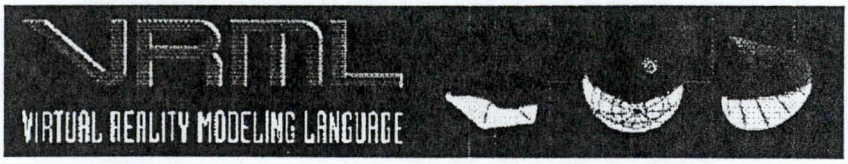
L'interpréteur de Java implémente la machine virtuelle de Java de manière logicielle. Avant d'exécuter un pseudo-code, il le vérifie et analyse ses effets. Cette manière de travailler assure la sécurité et la portabilité (la machine virtuelle permet d'utiliser le code d'un programme sur des plates-formes différentes). En outre, Java possède une édition de liens dynamique : c'est par rapport au besoin de l'application que l'interpréteur charge les classes nécessaires.

IX. Conclusion

Conçu par Sun Mycrosystems, Java est un langage pur objet utilisé généralement pour développer des applications «portables » en réseau. Il est surtout connu par ses applets couramment utilisées sur le Web. Une applet permet de réaliser l'animation des pages, l'interaction entre serveurs et clients, etc. A côté des applets, Java permet aussi la conception d'applications complètes. Comme exemple, nous pouvons citer *HotJava*, le browser de Sun Microsoft.

En plus du langage Java, nous utilisons un langage de modélisation en 3D qui lui est compatible et que nous présentons brièvement au chapitre suivant.

Chapitre IV : Le langage VRML



I. Introduction

Un des langages utilisés pour réaliser l'interface de notre simulateur est le langage de modélisation VRML. Ce chapitre présente le langage et ses principales caractéristiques.

Dans la première section de ce chapitre, nous décrivons l'historique du langage VRML. Le lecteur peut ainsi avoir une meilleure idée des raisons qui ont poussé à la création du langage VRML et de celles qui ont permis son évolution.

Ensuite, nous décrivons de façon générale ce qu'est un monde VRML, la façon dont celui-ci est organisé, les standards qui y sont adoptés notamment en ce qui concerne les diverses unités de temps, de mesure linéaire etc.

La troisième section présente les caractéristiques du pilier du monde VRML : le nœud. Dans cette section, nous expliquons également différents mécanismes qui agissent sur les nœuds.

Les différents types de valeurs qui existent dans VRML sont abordés dans la section quatre.

La section cinq présente une première manière d'introduire un certain dynamisme dans une scène VRML. Il s'agit du routage d'événements que nous appellerons script interne.

La sixième section présente les autres types de dynamisme qui existent dans les mondes VRML 2.0. Nous consacrons une section particulière pour ces types car ils ont la particularité de faire interagir le langage VRML avec un langage de programmation[2].

La dernière section présentera l'architecture théorique d'un browser VRML. Un browser VRML peut être considéré comme un interpréteur du langage VRML.

II. Historique du langage

Dans cette partie nous présentons au lecteur un historique du langage VRML, depuis les idées qui ont poussé à sa création, jusqu'à nos jours. Cet historique est repris dans les nombreux livres qui traitent de VRML. Dans le cadre de ce mémoire nous nous sommes principalement inspirés de **[HARTMAN96]** et de **[LEA96]**

Le langage VRML trouve ses sources dans une fusion entre les phénomènes de science fiction, de graphismes en 3D, de réalité virtuelle et du WWW. Le succès des nombreux films et livres actuels nous montrent la place qu'occupe la science fiction dans la vie courante.

[2] Dans le cadre de ce mémoire, le seul langage de programmation qui sera abordé est le langage Java. Ce langage est un des seuls qui, pour l'instant, permettent une interaction avec VRML.

L'argent qui est brassé dans ce domaine peut, en soi, être une preuve. Tout comme la science fiction, la réalité virtuelle occupe une place de choix dans la vie informatique actuelle. Il suffit pour s'en persuader, d'observer le nombre de jeux pour PC commercialisés actuellement et faisant appel à la réalité virtuelle. Ces mêmes jeux nous montrent de plus en plus que les interfaces graphiques ont une tendance à passer dans un cadre tridimensionnel, voir aux images vidéo de personnages insérées dans un monde purement virtuel. Quant au WWW, la preuve de son importance et de son évolution n'est plus à faire de nos jours.

Mettant tous ces éléments ensemble, de nombreuses personnes ont commencé à croire à ce qu'ils ont appelé "le rêve du monde cyber" : un monde virtuel en 3D, où chacun pourrait se promener depuis sa maison et découvrir les gens, les cultures et les pays du monde entier. Tout cela en utilisant le WWW! Le monde cyber serait un monde dans lequel toute personne serait représentée par un petit personnage (souvent appelé "avatar"). Cette personne pourrait voir les gens qu'elle rencontre à travers leur propre avatar. Parmi ces rêveurs du monde cyber, deux personnes ont travaillé de façon plus approfondie sur leur propre vision de ce monde. En s'inspirant des pages HTML (caractérisées par le fait qu'elles existent dans un monde à deux dimensions) Mark Pesce et Tony Parisi ont conçu un browser assez générique pour visualiser tous les environnements tridimensionnels rencontrés sur le WWW. Lors de la conférence de Genève de février 94, consacrée au WWW, ils présentèrent un premier prototype de ce browser. C'est lors de cette conférence que le terme VRML (Virtual Reality Markup Language) fut attribué au langage qui lui était associé. Au lendemain de cette conférence, des milliers de personnes intéressées avaient créé un forum de discussion pour mettre au point les spécifications de VRML. L'évolution de ce langage et l'intérêt croissant que de nombreuses sociétés ont manifesté furent tellement rapides, que lors de la deuxième conférence sur le WWW en octobre 94 un premier brouillon de spécification était proposé. La curiosité envers ce langage ne cessa de s'amplifier après cette conférence. Elle a permis de voir très rapidement apparaître des browsers VRML 1.0 conçus par divers grandes sociétés informatiques, ainsi qu'une description officielle du langage en avril 95, soit 14 mois après la présentation du premier prototype. VRML devient à cette époque Virtual Reality Modeling Language car le mot *"modélisation"* traduisait plus justement les objectifs visés par VRML.

Hélas, cette première version du langage souffrait d'un grand défaut : son caractère statique. Ce langage permettait la création de mondes 3D aussi complexes qu'on le voulait, mais l'utilisateur ne pouvait que se promener autour et les visualiser. Aucune interaction entre l'utilisateur et les objets n'était possible et tous les objets étaient fixes. Les détracteurs de VRML dirent que le M signifiait "Museum", car rien ne changeait jamais dans les scènes et qu'on ne pouvait rien toucher. Pour cette raison plusieurs grandes sociétés (Sony, IBM, SGI,...) ont développé leur propre extension à VRML en y introduisant un certain dynamisme. Mais plutôt que de mettre sur le marché différentes versions de ces extensions, le VRML Architecture Group [3] proposa un vote pour déterminer lequel de tous ces langages deviendrait la seconde version de VRML, assurant ainsi une version standard du langage et évitant une guerre de pouvoir entre les différentes grandes sociétés. C'est en mars 96 que le vote eut lieu. Les sociétés adoptèrent le modèle du "Moving Worlds" de SGI comme début des spécifications de VRML 2.0. Elles décidèrent que ces spécifications devraient être finalisées pour août 96, ce qui n'est pas tout à fait le cas car certains points ne sont pas encore complètement fixés. La seule exception à cette standardisation est l'*External Authoring Interface* dont nous parlerons plus loin. Une fois que cette partie aura été normalisée le

[3] En abrégé VAG. Ce groupe s'occupe essentiellement de tout ce qui est relatif à la standardisation du langage VRML (entre autres)

langage VRML 2.0 changera de nom. Il deviendra VRML 97, et ce conformément aux normes ISO de standardisation des noms.

III. Description d'un monde VRML

Un fichier, ou un monde VRML, est principalement une collection d'objets structurés hiérarchiquement. Chaque objet possède des caractéristiques propres : sa position dans l'espace 3D, sa forme, sa taille, sa couleur etc. Pour créer un monde VRML un simple éditeur de texte suffit. Il faut bien entendu respecter la syntaxe de VRML 2.0. Tous les fichiers décrivant un monde VRML doivent commencer par le même entête, pour permettre une identification plus facile. Cet entête est le suivant :

```
#VRML V2.0 utf8
```

Le terme *"utf8"* fait référence au standard ISO pour les strings. Tous les autres caractères qui seraient écrits entre cet entête et la fin de la ligne (précisée par un "carriage return") ne sont pas pris en compte. VRML est un langage *"case sensitive"*, il faut donc bien prendre garde aux majuscules et aux minuscules lorsque l'on crée un monde.

Dans un fichier (ou un monde) VRML il est possible d'introduire des commentaires. Pour cela il suffit d'insérer le caractère #. Tous les caractères compris entre ce symbole et la fin de la ligne sont ignorés. Il existe cependant une exception à cette règle. Si le caractère # est repris entre des guillemets, alors il fait partie d'un string et on ne le considère pas comme le marqueur d'un commentaire. Les virgules, les blancs, les tabulations et les carriage return sont considérés comme des caractères de séparation pour autant qu'ils apparaissent en dehors de guillemets. On peut utiliser un ou plusieurs d'entre eux pour séparer les entités syntaxiques du fichier VRML là où cela peut s'avérer nécessaire.

Après l'entête, un fichier VRML contient une combinaison quelconque des éléments suivants :

- Un nombre quelconque de prototypes (cfr. section 4);
- Un nombre quelconque de nœuds de regroupement (cfr. section 4);
- Un nombre quelconque de déclarations de routes (cfr. section 6).

Le monde VRML étant un fichier de texte, celui-ci doit être interprété par un moyen particulier. Ce moyen est le browser VRML. Nous détaillerons ce dernier plus en détail dans la dernière section de ce chapitre. Pour l'instant, le lecteur doit juste savoir que sans un browser VRML, aucune visualisation du monde et de son dynamisme n'est possible.

Dans un monde VRML, on ne définit pas d'unités de mesure. Toutes les distances linéaires sont censées être en mètres et tous les angles sont censés être en radians. Les unités de temps sont en seconde et les couleurs en pourcentage de composants RGB[4]. Cependant pour les distances linéaires, le principal est de rester cohérent tout au long de la construction du monde. L'utilisation du mètre comme mesure standard va permettre à des développeurs de mettre leur monde en rapport avec d'autres.

[4] RGB = Red Green Blue. Une couleur particulière est représentée comme un mélange de rouge, de vert et de bleu c'est à dire comme un mélange des trois couleurs primaires. Le pourcentage des composants RGB permet de définir correctement ce mélange pour donner une couleur précise.

Quand on se place à un niveau d'abstraction élevé, un monde VRML est simplement un fichier qui permet de décrire des objets. Ces objets peuvent théoriquement contenir n'importe quoi (des formes géométriques en 3D, des données sonores, des données d'images, des données filmées...). Pour cela VRML fournit un ensemble d'objets de base que l'on appelle des nœuds. Nous allons donc maintenant étudier ces objets en détail.

IV. Le pilier des mondes VRML : le nœud

L'élément de base, pilier de la construction d'un monde VRML, est *le nœud* (***Node***). Un nœud est un type d'objet de VRML. Tout objet d'un monde VRML est décrit grâce à un ou plusieurs nœuds.

Il existe deux grandes catégories de nœuds VRML : les nœuds de regroupement et les nœuds descendants de nœuds regroupement. Un nœud de regroupement est utilisé afin de créer la structure hiérarchique du monde VRML. Il permet également de construire les diagrammes hiérarchiques dont nous parlerons un peu plus bas dans cette section. Les nœuds de regroupement définissent un espace local de coordonnées pour tous les nœuds fils qu'ils contiennent.

Tout nœud peut lui même en contenir d'autres. Un nœud est caractérisé par :

- Son nom qui est aussi le nom du type du nœud;
- Les champs qui le composent. Ces champs sont les paramètres qui distinguent un nœud des autres nœuds du même type. Un nœud peut avoir zéro ou plusieurs champs, mais il précise toujours le type, le nom et la valeur par défaut de ceux-ci. L'ordre dans lequel on lit ou on attribue une valeur aux champs du nœud n'a pas d'importance. Il existe deux sortes de champs : les "***field***" et les "***exposed field***". Un ***field*** définit une valeur initiale pour l'état du nœud et ne peut en aucun cas être changé par un événement. Un ***exposed field*** définit aussi une valeur initiale pour le nœud mais celle-ci pourra être changée par un événement;
- Les événements qu'il génère ou qu'il reçoit. Un événement est un message modifiant la valeur d'un champ. Un événement entrant (***eventIn***) est un événement que le nœud reçoit et qui modifie la valeur d'un de ses ***exposed field***. Un événement sortant (***eventOut***) est un événement envoyé par le nœud pour indiquer qu'un de ses ***exposed field*** a changé de valeur.

Les nœuds contiennent des données élémentaires qui sont stockées dans leurs champs et dans leurs événements. Dans le cas où aucune valeur de champ n'est précisée dans la conception d'un monde VRML, la valeur par défaut de ce champ sera prise en compte par le browser VRML[5] lors de la construction de la scène VRML. Comme nous l'avons précisé plus haut, un monde VRML est un ensemble d'un ou plusieurs fichiers. Dans ce mémoire, le terme monde fait référence aux fichiers décrivant les objets qui le composent. Pour désigner la représentation tridimensionnelle de ce monde, c'est à dire de ce que l'on pourra voir du monde, nous parlerons de la scène VRML. Une scène VRML est donc un affichage des objets contenus dans le monde VRML.

[5] Le browser VRML sera décrit plus loin dans ce document. C'est lui qui interprète les fichiers décrivant les mondes VRML pour les afficher correctement.

Il existe des nœuds prédéfinis. Un nœud prédéfini est un nœud décrit dans la syntaxe du langage VRML et reconnu par le VAG. Ses champs possèdent tous une valeur par défaut. Tous les événements qu'il peut recevoir ou générer sont eux aussi fixés dans sa définition. Cependant, l'utilisateur est libre de créer ses propres nœuds grâce au mécanisme de *prototypage*.

Le prototypage est un mécanisme permettant de définir un nouveau nœud pour un monde VRML donné. Un prototype est toujours conçu à partir de nœuds prédéfinis de VRML. La définition d'un prototype doit respecter la syntaxe suivante :

- Le mot clé ***PROTO***;
- Le nom du nouveau nœud (et donc du nouveau type de nœud);
- La déclaration du prototype qui comprend :
  - une liste publique de ***eventIn*** et ***eventOut*** qui peuvent générer ou recevoir des événements;
  - une liste de champs ***field*** et ***exposed field*** avec leurs valeurs par défaut;
- La définition du prototype qui consiste en une structure d'un ou plusieurs nœuds prédéfinis, une ou plusieurs routes et un ou plusieurs prototypes. Les nœuds qui sont contenus dans cette définition doivent contenir le mot clé ***IS*** autant de fois qu'il y a de champs et d'événements déclarés pour le prototype.

Les noms des ***field***, des ***exposed field***, des ***eventIn*** et des ***eventOut*** doivent être uniques pour un même prototype. L'utilisation de la notion ***IS*** est très importante. Elle permet de faire le lien entre les éléments du prototype et les éléments des nœuds prédéfinis de VRML qui le compose. Il est possible, pour alléger le fichier VRML, de créer les prototypes dans d'autres fichiers que le fichier principal du monde. Pour cela, il faut juste en redonner la déclaration exacte dans le fichier principal et indiquer à quel endroit on peut en trouver la définition. Pour indiquer où se trouve la définition du prototype, on écrit sa localisation entre guillemets, il peut s'agir d'une adresse Internet ou d'une adresse locale sur le disque. On peut même éventuellement en préciser plusieurs et, dans ce cas, la première localisation qui sera trouvée sera prise en compte. Le mot clé utilisé dans cette situation n'est plus ***PROTO*** mais ***EXTERNPROTO***.

Il est fréquent que le champ d'un nœud soit un nœud d'un autre type, on parle alors de nœud fils. Ce nœud fils a lui aussi ses propres champs et ses propres événements. Afin de ne pas se perdre dans la conception d'un monde VRML, il est souvent conseillé d'établir un diagramme hiérarchique. En effet, il faut savoir que toute transformation physique (telle que la translation, la rotation, le changement d'échelle) appliquée à un nœud, l'est aussi à tous ses nœuds fils. Les diagrammes hiérarchiques permettent de mieux voir quels nœuds seront influencés par une de ces transformations.

Il est possible que dans un fichier VRML, un même nœud soit utilisé plusieurs fois. Pour éviter de surcharger le fichier et pour rendre sa lecture plus simple, on peut utiliser l'instanciation. Pour cela il faut utiliser la structure ***DEF - USE.*** Le mot clé ***DEF*** permet d'associer un nom à un nœud et le mot clé ***USE*** permet de l'utiliser à un moment particulier en ne citant que son nom. Dans le cas où plusieurs nœuds auraient reçu le même nom grâce à l'emploi de ***DEF***, ***USE*** prendra alors en compte la déclaration la plus récente. L'utilisation de cette structure est limitée à un fichier. Ce mécanisme ne peut donc pas être utilisé dans le cadre de prototypes externes.

Certains nouveaux nœuds de VRML 2.0 permettent à l'utilisateur d'interagir avec le monde qu'il observe. Tous les nœuds de VRML 2.0 ont un comportement. Ce comportement est rendu possible grâce aux événements qui caractérisent les nœuds. Il est par exemple possible de changer la couleur d'un objet lorsque l'utilisateur clique sur ce dernier. L'interaction est possible également grâce aux possibilités de script qui sont exposées dans la section 7 de ce chapitre.

Afin de permettre au lecteur de mieux comprendre ce qu'est un nœud nous allons illustrer certains des principes que nous venons de citer par un exemple simple.

```
#VRML V2.0 utf8
Transform{
   translation 2 0 0
   children  Shape {
         geometry Box {}
   }
}
Transform{
   translation 4 0 0
   children Shape{
         geometry Cone{}
   }
}
Transform{
   translation 6 0 0
   children Shape {
         geometry Cylinder{}
   }
}
Transform{
   translation 8 0 0
   children Shape {
         geometry Sphere{}
   }
}
```

La scène VRML associée au monde que nous venons décrire est illustrée par la Figure 18.

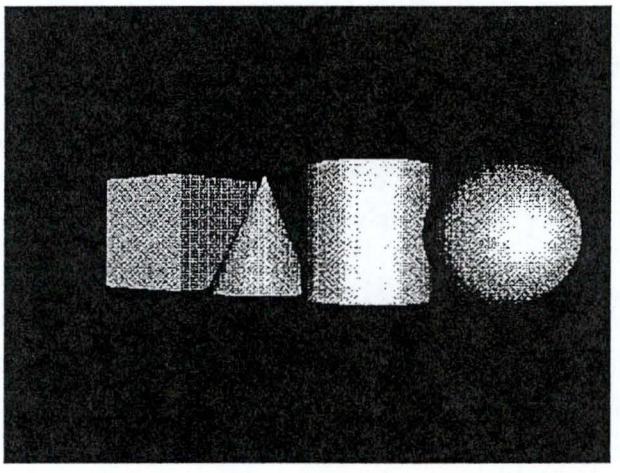


Figure 18 : une scène VRML simple

Nous pouvons donc maintenant expliciter brièvement le contenu du monde VRML. La première ligne du monde est l'entête obligatoire de tout monde VRML.

Le nœud ***Transform*** est un nœud de regroupement qui va permettre d'appliquer des transformations physiques aux objets de la scène VRML (par exemple une translation, une rotation ou un changement d'échelle). Les éventuelles transformations que ce nœud décrit seront appliquées à tous ces nœuds fils. Dans notre exemple, la seule transformation qui sera appliquée est une translation dans le repère de coordonnées à trois dimensions. Les translations sont décrites dans le champ intitulé ***translation.*** Toutes les transformations de notre exemple s'applique à des formes (représentées par les nœuds ***Shape***). Ces formes sont les formes géométriques tridimensionnelles de base, à savoir le cube, le cône, le cylindre et la sphère, qui sont aussi des nœuds prédéfinis de VRML (***Box, Cone, Cylinder, Sphere***). Le lecteur remarquera que pour ces formes aucune caractéristique particulière n'a été décrite dans le monde VRML. Dès lors, le browser VRML va les afficher en utilisant les valeurs par défaut des champs des nœuds qui les décrivent.

V. Les types de VRML

On peut classer les types en VRML selon deux grands axes:

- Les types relatifs aux nœuds;
- Les types relatifs aux champs ou aux événements.

Les types relatifs aux champs ou aux événements sont subdivisés en deux catégories:

- Les types simples;
- Les types multiples.

Nous allons décrire brièvement dans cette section les différentes caractéristiques et particularités de ces types.

V.1 Les types de nœuds

Les types de nœuds correspondent aux nœuds natifs de VRML. Pour ces types, le nom du nœud (qui est repris dans la structure du fichier) correspond aussi à son type. Il est donc évident de savoir de quel type est un nœud natif de VRML, puisque pour cela il suffit de regarder quel est son nom. Comme nous l'avons déjà précisé plus haut, l'utilisateur du langage peut définir ses propres nœuds grâce au prototypage. Comme tout prototype doit avoir un nom, on peut considérer qu'en créant un nouveau nœud, l'utilisateur crée également un nouveau type de nœud.

V.2 Les types de champs

Comme nous l'avons dit dans la section consacrée aux nœuds, tout nœud de VRML est caractérisé par un certain nombre de champs et d'événements. Les champs d'un nœud, au départ, ont toujours une valeur par défaut. Cette valeur est d'un type différent en fonction de ce que le champ représente. De même, comme les événements changent (ou indiquent un changement de) la valeur d'un champ, la valeur qu'ils véhiculent doit être du même type que le champ qu'ils changent. Ces types peuvent être simples ou multiples. Un type multiple est un ensemble de zéro ou plusieurs valeurs d'un même type simple. Nous allons rapidement

décrire les différents types qui existent pour les champs ou les événements de VRML. Les types de champs simples commencent toujours par les lettres *SF* (**S**imple **F**ield). Les types de champs multiples quant à eux, commencent toujours par les lettres *MF*. La valeur d'un type multiple sera toujours écrite selon la syntaxe suivante :

[valeur1, valeur2, valeur3, ...].

En fonction du type multiple qui sera décrit *valeur1, valeur2, valeur3* peuvent être de type réel, booléen ou autre.

1. ***SFBool :***

   Un ***SFBool*** spécifie un type simple contenant une valeur booléenne. Les deux valeurs possibles de ce type sont ***TRUE*** et ***FALSE*** écrits en majuscules.

2. ***SFColor et MFColor :***

   Un ***SFColor*** spécifie un triplet de réels correspondant aux composants RGB d'une couleur. Un ***MFColor*** quant à lui, spécifie un ensemble de zéro ou plusieurs triplets de réels caractérisant chacun les composants RGB de diverses couleurs. Il faut noter que les éléments RGB sont des réels compris entre 0.0 et 1.0 étant donné qu'ils représentent un pourcentage de rouge, de vert et de bleu. Voici un exemple de ***MFColor*** qui contient deux éléments :

   Couleurs [0.1452545 0.1428745 0.66544,
   0.2541 0.022554575 0.2355448].

   Dans cet exemple *Couleurs* est le nom d'un champ. Il représente un ensemble de deux couleurs, la première (définie par le premier triplet de réels) correspond à un bleu tirant vers le violet et la seconde (définie par le second triplet de réels) correspond à un mauve foncé. Les deux triplets sont séparés par une virgule et les éléments de chaque triplet sont séparés par un espace.

3. ***SFFloat et MFFloat :***

   Un ***SFFloat*** spécifie un nombre réel et un ***MFFloat*** précise un ensemble de zéro ou plusieurs réels.

4. ***SFInt32 et MFInt32 :***

   Un ***SFInt32*** spécifie un entier de 32 bits et un ***MFInt32*** spécifie un ensemble de zéro ou plusieurs entiers de 32 bits. Ils peuvent être écrits selon un format décimal ou hexadécimal.

5. ***SFNode et MFNode :***

   Ces types sont un peu particuliers car ils spécifient respectivement un nœud VRML et un ensemble de zéro ou plusieurs nœuds VRML. Le nœud repris peut être un nœud prédéfini de VRML ou un nœud défini par l'utilisateur (c'est à dire un prototype). Les champs de type ***SFNode*** ou ***MFNode*** contiennent le mot clé ***NULL*** pour indiquer qu'ils sont vides.

6. ***SFRotation et MFRotation :***

   Le type ***SFRotation*** spécifie une rotation arbitraire et le type ***MFRotation*** spécifie zéro ou plusieurs rotations arbitraires. Les valeurs de ces types sont décrites grâce à quatre

réels séparés par un ou plusieurs espaces. Les trois premiers indiquent la valeur du vecteur qui détermine l'axe de rotation dans le repère de coordonnées. Le dernier précise en radian l'angle de rotation.

*7. **SFString et MFString :***

Ces types spécifient des chaînes de caractères écrites dans un format qui respecte la norme *UTF-8*. Ces chaînes de caractères sont entourées de guillemets, qui précisent le début et la fin de chaque chaîne.

*8. **SFTime et MFTime :***

Ces deux types spécifient respectivement une seule et zéro ou plusieurs valeurs de temps. Ces valeurs sont en fait des réels. Ces types précisent le nombre de secondes écoulées depuis une date origine. Ce temps origine est le 1 janvier 1970 00:00:00 (GMT).

*9. **SFVec2f et MFVec2f :***

Ces types spécifient respectivement un et zéro ou plusieurs vecteurs à deux dimensions. Les éléments de ces vecteurs sont deux réels séparés par un blanc.

*10. **SFVec3f et MFVec3f :***

Ce types spécifient respectivement un et zéro ou plusieurs vecteurs à trois dimensions. Les éléments de ces vecteurs sont trois réels séparés par un ou plusieurs blancs.

Il existe un dernier type que nous ne décrirons pas dans ce mémoire pour deux raisons. Tout d'abord nous n'avons jamais eu à l'utiliser, et ensuite il est beaucoup plus complexe à décrire que les autres types que nous venons de passer en revue. Il s'agit du type ***SFImage***.

VI. Les concepts d'événements et de routage d'événements

Le grand apport de VRML 2.0 sur VRML 1.0 est la possibilité de faire bouger les objets les uns par rapport aux autres. Ce dynamisme est possible tout d'abord grâce au routage d'événements.

Le routage fait partie du fichier qui décrit une scène VRML. Il indique vers quels nœuds sont envoyés les événements générés que l'on veut traiter dans le monde. La base d'une animation est d'offrir la possibilité de changer le monde à n'importe quel moment. En VRML, changer le monde signifie changer la valeur des champs des objets qui déterminent le monde. Il faut noter que ces changements n'ont en fait lieu que dans la représentation interne du browser VRML et pas dans le fichier lui-même.

Pour changer la valeur d'un champ d'un objet, il faut lui envoyer un événement. Le routage d'un événement précise au browser quel événement il doit générer et vers quel champ il doit l'envoyer. Les événements sont envoyés uniquement vers des champs ***exposed field***. Le routage est donc une partie du monde VRML dans laquelle on décrit une série de couples d'événements (un événement sortant d'un nœud et un événement entrant dans un nœud). Une association d'un événement entrant et d'un événement sortant est appelée *route*. Le principe représenté par une route est illustré par la Figure 19

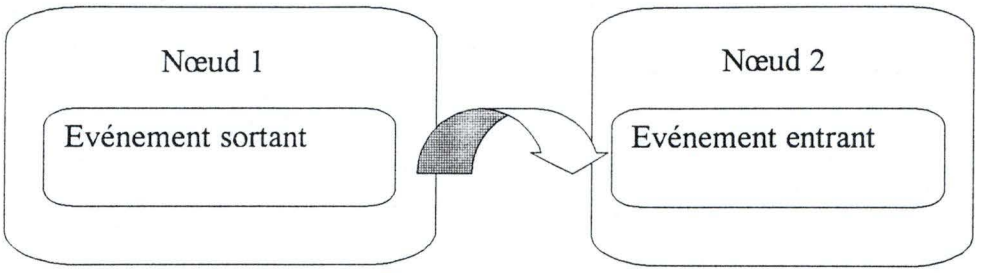


Figure 19 : une route

Voici comment seront déclarées les routes dans un fichier VRML :

```
ROUTE Noeud1.EvénementSortant TO Noeud2.EvénementEntrant
ROUTE Noeud2.EvénementSortant TO Noeud3.EvénementEntrant
ROUTE Noeud4.EvénementSortant TO Noeud3.EvénementEntrant
ROUTE Noeud1.EvénementSortant TO Noeud4.EvénementEntrant
```

L'ensemble de toutes ces routes forme le routage. L'ordre dans lequel les routes sont déclarées n'est pas important. Par contre il faut toujours que l'événement qui se trouve après le mot réservé ***ROUTE*** soit un événement sortant d'un nœud et celui qui est situé après le ***TO*** soit un événement entrant. Ces deux événements doivent être d'un même type légalement admis par VRML[6]. On ne peut pas, par exemple, passer un événement de type "***SFColor***" vers un autre de type "***SFVec3f***". Dans le cas où les types des événements ne coïncident pas, la route est invalide et n'est pas prise en compte par le browser VRML, celui-ci pourrait même éventuellement générer une erreur d'interprétation et ne rien afficher du monde dans la scène VRML.

VII. Le dynamisme en VRML 2.0

L'animation grâce au routage d'événements est assez limitée . En effet, si ce principe avait été le seul prévu pour permettre le dynamisme dans les mondes VRML, nous n'aurions rien vu d'autre que des animations prédéfinies : l'animation des scènes étant réduite à un routage d'événements prédéfinis vers des nœuds prédéfinis. Le dynamisme peut être également géré grâce aux langages de programmation et notamment grâce au langage Java. Cette possibilité offre des avantages très importants si l'on veut décrire des états de la scène VRML plus complexes comme par exemple savoir si une porte est ouverte ou si une lampe est allumée. Ce genre de décision requiert des données plus complexes. Ce type de dynamisme est appelé : *"comportement dynamique des objets"* (en opposition aux comportements statiques du routage d'événements). Le comportement dynamique est basé sur l'état des objets dans le monde VRML.

Cette section présente les deux types d'interaction existant entre Java et VRML. Ces interactions sont possibles grâce au browser VRML que nous décrirons ensuite.

VII.1 Le script interne

La première façon de donner à VRML une ouverture sur un langage de programmation est d'utiliser le nœud prédéfini ***Script.*** Nous avons choisi d'appeler cette façon de travailler l'utilisation du *script interne* car, pour introduire un certain dynamisme dans les mondes

[6] Les types légalement admis par VRML ont été décrits dans la section 4 de ce chapitre.

VRML par ce mécanisme, il n'est pas nécessaire de recourir à un autre langage que le langage VRML.
On peut placer le nœud ***Script*** où l'on veut dans la structure du fichier, il peut être utilisé avec la méthode ***DEF – USE*** et on peut router des événements vers et à partir de ce nœud. Sa principale différence par rapport aux autres nœuds prédéfinis de VRML, réside dans le fait que ses champs sont *"user-extensible"* et que les événements qui arrivent vers ce nœud sont directement passés au programme qui lui est associé. La communauté de VRML a décidé que VRML ne devait être attaché à aucun langage de programmation en particulier et ce, pour éviter d'arriver à une "guerre" des langages de programmation. De plus elle pensait que l'utilisation d'un seul langage ne pouvait remplir toutes ses exigences par rapport à VRML. Cependant la plupart des développeurs VRML utilisent Java à cause de sa puissance et de son intégration étroite avec Internet. Nous allons maintenant détailler un peu plus les deux grandes différences du nœud ***Script*** par rapport aux autres nœuds natifs de VRML.

- Φ Les champs *user-extensible* : la personne qui utilise le nœud ***Script*** peut y déclarer le nombre de champs qu'il veut et qui lui est utile en plus des champs prédéfinis dans la syntaxe VRML. Les seules contraintes sur ces champs sont les suivantes : le type des champs doit être un type légal de VRML et les champs ne peuvent en aucun cas être déclarés comme étant ***exposed field***. Ce nœud sera d'ailleurs le seul cas où un champ non exposé peut changer de valeur.
  Cette caractéristique du nœud ***Script*** permet au développeur de connaître l'état de la scène VRML à un moment donné en fonction des valeurs des champs qu'il aura définis.

- Φ Le lien avec le programme : le nœud ***Script*** possède un champ particulier nommé ***url***. Dans ce champ on peut soit indiquer l'endroit où se trouve le programme, soit insérer le script lui même. Le champ ***url*** peut contenir plusieurs références à différents programmes, dans ce cas le browser exécute le premier script écrit dans un langage de programmation qu'il supporte.

On peut considérer que le nœud ***Script*** fonctionne comme une passerelle entre le monde VRML et le programme (ou script) qui décrit son dynamisme. Le programme seul ne peut cependant pas changer le monde en tant que tel puisque tous les résultats qu'il fournit sont repassés au nœud. Il est donc nécessaire de compléter ce processus par un routage des événements passés au nœud ***Script***. Tous les événements qui sont routés vers le nœud ***Script*** sont immédiatement passés au programme qui lui est associé grâce au browser VRML. La façon dont le browser réalise ce passage du monde VRML au programme dépend du browser et du langage de programmation utilisés. Une fois que l'événement est reçu par le programme, ce dernier le traite comme il veut. Après ce traitement le programme peut éventuellement retourner une valeur au nœud ***Script***. Dans le cas où cette valeur est retournée vers un champ, la seule conséquence est de changer la valeur du champ mais dans le cas où elle est retournée comme valeur d'un événement le traitement est totalement différent. Ces événements sont considérés comme des événements sortants du nœud. Ils peuvent donc être routés vers d'autres nœuds et ainsi permettre la réalisation du dynamisme.

Ce mécanisme du routage des événements du nœud ***Script*** est illustré grâce à la Figure 20 :

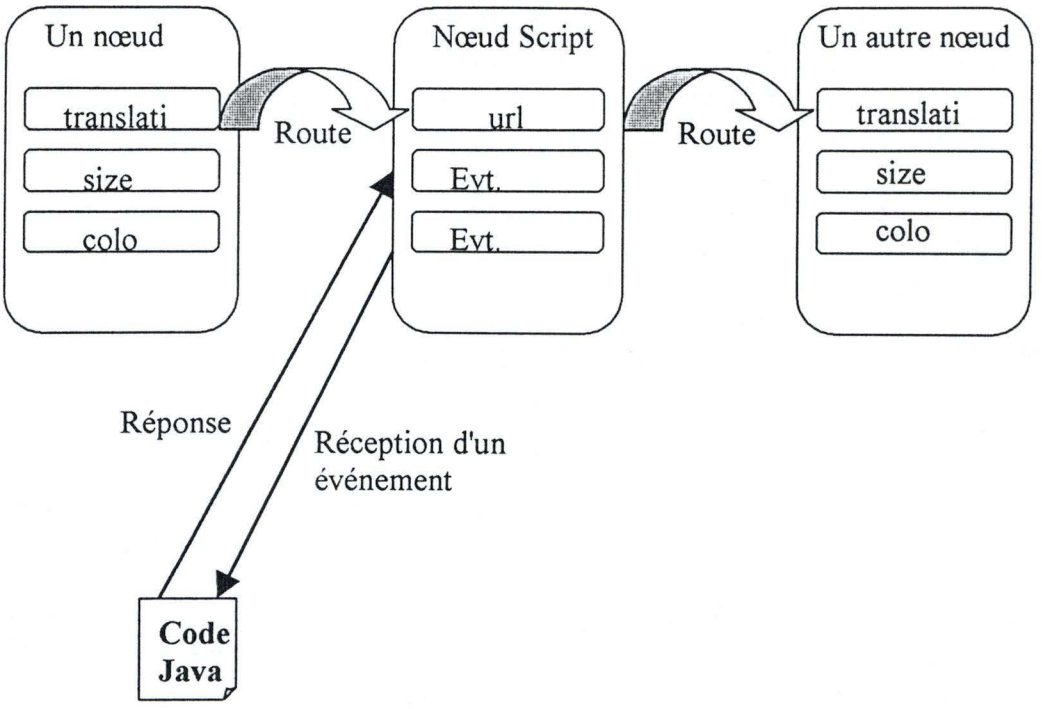


Figure 20 : le fonctionnement du script interne

Le programme qui sera associé au monde VRML par l'intermédiaire du nœud ***Script*** doit pouvoir remplir certaines fonctionnalités. C'est à dire il doit :

- Pouvoir saisir et gérer les événements reçus par le nœud ***Script***;
- Pouvoir consulter et changer la valeur des champs du nœud ***Script***;
- Pouvoir générer et envoyer de nouveaux événements au nœud ***Script*** en fonction de la valeur des champs de ce nœud. C'est à dire en fonction de l'état de la scène VRML.

Nous allons illustrer le fonctionnement de ce script par un petit exemple. Dans celui ci le programme sera écrit en Java. Nous devrons utiliser certaines méthodes qui permettent au programme de réaliser les fonctionnalités décrites ci-dessus. Nous n'entrerons pas dans le détail du fonctionnement de ces méthodes, nous nous contenterons de préciser ce qu'elles font. Pour plus d'informations au sujet des programmes Java associés aux nœuds ***Script*** des mondes VRML, nous proposons au lecteur de se référer à **[LEA96]**.

Supposons que dans un monde VRML nous trouvions un nœud ***Script*** tel que celui-ci :

```
Script{
     url       "Exemple.class"
     eventIn   SFBool start
     eventOut  SFBool on
     field           SFBool state TRUE
}
```

Ce nœud est composé de deux événements (un entrant et un sortant), d'un champ booléen qui contient par défaut la valeur ***TRUE*** et de l'url qui précise le nom du programme Java qui est associé au monde VRML dans lequel ce nœud est défini. Les noms des événements et du champ ont été indiqués dans une police différente du reste du nœud. Voici à quoi pourrait ressembler le programme Java qui lui est associé :

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

class Exemple extends Script {
  private SFBool state;
  private SFBool on;

  public void initialize(){
        state = (SFBool) getField("state");
        on = (SFBool) getEventOut("on");
  }

  public void processEvent(Event e){
        if (state.getValue() == true){
              on.setValue(true);
              state.setValue(false);
        }
        else{
              on.setValue(false);
              state.setValue(true);
        }
  }
}
```

Dans ce programme, toute variable Java a le même nom que le champ ou l'événement du nœud script auquel elle est associée. Le nom de ces variables du programme a été indiqué dans une autre police que celle du reste du programme. La méthode *initialize( )* est invoquée avant toutes les autres, elle permet d'associer les variables du programme Java aux champs ou événements du nœud ***Script*** auquel le programme est lié. La méthode *proccesEvent( )* est invoquée chaque fois que l'événement `start` du nœud ***Script*** reçoit (par le mécanisme de routage) un événement nouveau. En fonction de la valeur de cet événement, le programme va associer une nouvelle valeur au champ `state` du nœud ***Script*** et va générer un nouvel événement qu'il passera à l'événement `on` du nœud ***Script***. Ce qui se passe dans la scène VRML suite à cela dépend du routage et de ce que représente ces événements.

Le script interne est maintenant totalement spécifié grâce au travail du VAG. Il peut donc être utilisé sans problème majeur. On peut trouver toutes les spécifications du script interne sur les sites officiels de VRML.

VII.2 Le script externe

La seconde façon d'ouvrir les mondes VRML sur d'autres langages est d'utiliser le script externe. Pour cela il est nécessaire d'utiliser l'EAI (*External Authoring Interface*) dont les spécifications sont toujours en cours d'élaboration et de proposition. L'EAI est l'interface qui permet à un monde VRML de communiquer avec son environnement extérieur. Elle définit l'ensemble des fonctions que l'environnement externe peut appliquer sur le monde VRML pour affecter celui ci. Le seul exemple pratique de communication qui existe actuellement est le lien entre une applet Java dans une page HTML et un monde enchâssé (c'est à dire repris dans les étiquettes "embed") sur la même page[7]. L'EAI est un pont et un lien entre les

[7] Le consortium WWW est encore en pleine finalisation des propositions sur les objets enchâssés et sur l'introduction des applets Java dans une page HTML. La discussion sur la façon dont les deux communiquent est donc sujette à de nombreux changements.

applications Java et les mondes VRML. Elle permet de construire dynamiquement des fichiers VRML basés sur des données reçues de l'applet Java. Ces mêmes données peuvent elles-mêmes être mises à jour dynamiquement depuis l'interface VRML et ce, toujours grâce à l'EAI. Dans le cadre de ce mémoire, nous décrirons principalement comment utiliser l'EAI du langage Java. La Figure 21 présente la relation qui existe entre une applet Java et un monde VRML.

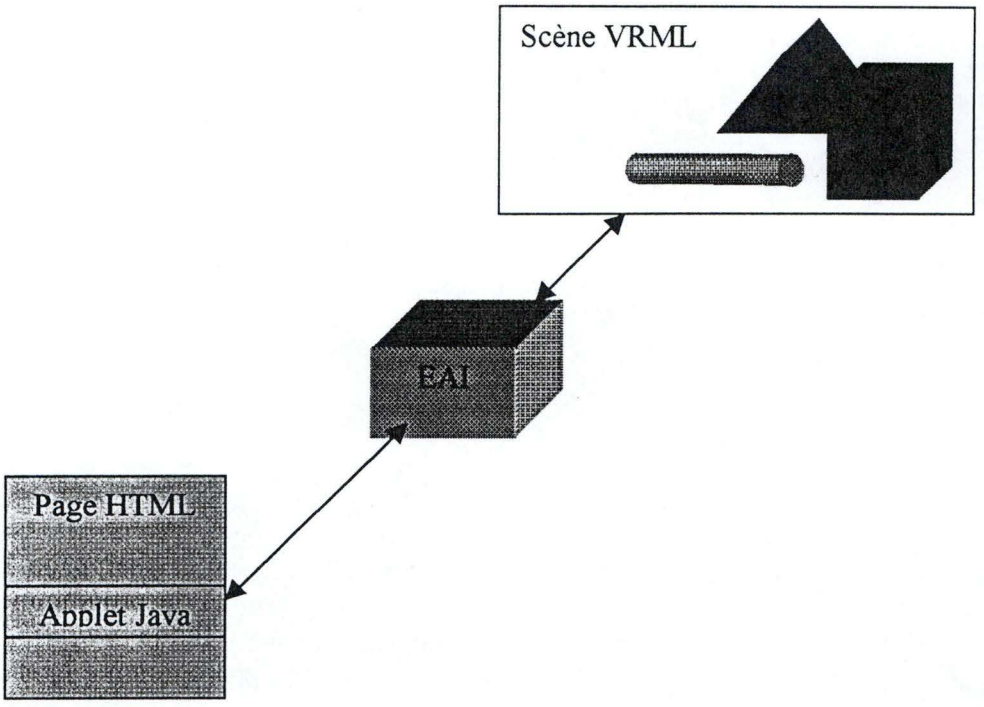


Figure 21 : illustration de l'EAI

L'EAI est une *Application Programmer Interface* (API) qui permet cette interaction entre les deux langages. Une API est une librairie de méthodes disponibles pour un développeur. Cette librairie offre à l'utilisateur les signatures et une description de ce que font les méthodes. L'utilisateur de l'API ne sait donc pas comment sont implémentées les méthodes qu'il va utiliser. L'EAI sera utilisée comme toutes les autres API Java. Cependant l'EAI est fournie avec le browser VRML plutôt que d'être partie intégrante de l'environnement standard du JDK[8].

Les fonctionnalités qui peuvent être remplies grâce à l'utilisation de l'EAI sont les suivantes :

1. ajouter des nœuds au monde VRML. Ce qui peut avoir pour effet d'ajouter des objets dans la scène VRML;
2. supprimer des nœuds du monde VRML. Ce qui peut avoir comme effet de supprimer des objets de la scène VRML;
3. envoyer des événements à des nœuds existants dans le monde VRML;
4. recevoir des événements provenant de nœuds existants dans le monde VRML;
5. interroger le monde VRML sur son état et ses caractéristiques.

[8] JDK est l'abréviation du *Java Developer Kit* conçu par Sun et outil de développement standard du langage Java.

Ces fonctionnalités peuvent également être accomplies par le script interne mais dans le cas particulier du script externe toutes celles-ci peuvent être gérées grâce aux éléments de Java comme les boutons, les potentiomètres, etc.

Avant de pouvoir accéder au monde VRML, il faut d'abord accéder à l'instance de la classe ***Browser*** qui correspond à ce monde. Cette classe est le reflet du monde VRML dans l'applet Java et sera utilisée pour permettre tous les accès au monde. Pour cela il faut définir une variable de type ***Browser*** et lui associer l'instance de la classe. On utilise une méthode particulière illustrée dans l'exemple suivant :

```
(...)
Browser Variable_Browser;

  Variable_Browser = Browser.getBrowser( );
(...)
```

Suite à cette instruction la variable *Variable_Browser* contient l'instance de la classe browser et le lien entre l'applet Java et le monde VRML est créé. Une fois que cette instruction a été réalisée, il est possible de communiquer avec l'applet Java et notamment d'obtenir des références vers des nœuds qui y sont définis. Il existe une condition préalable à cela. Les nœuds dans le monde VRML doivent avoir un nom défini par l'utilisateur. Cela veut dire qu'ils doivent avoir été nommés grâce à la technique ***DEF*** dont nous avons parlé plus haut. Pour lier un nœud du monde VRML à une variable Java il faut, tout comme pour le browser, utiliser une méthode particulière. Cette méthode est illustrée grâce à l'exemple suivant :

```
(...)
Node [ ] Variable_Nœud;
  Variable_Browser = Browser.getBrowser( );
  Variable_Nœud = Variable_Browser.getNode(" NomDuNoeud ")
(...)
```

Le string *NomDuNœud* référencé dans la méthode ***getNode*** est le nom donné au nœud dans le monde VRML. Il faut donc porter une attention particulière aux majuscules et aux minuscules. En général pour éviter de se perdre dans les associations entre les variables Java et les nœuds qu'elles représentent, il est conseillé aux programmeurs de donner le même nom aux variables qu'aux nœuds qui leur sont associés. Ce raisonnement est logique puisque pour travailler sur un nœud depuis l'applet on travaille en réalité sur la variable qui le référence.

Comme nous l'avons précisé un peu plus haut dans cette section, une des fonctionnalités de l'EAI est de permettre la gestion d'événements liés aux nœuds *existants dans le monde VRML* (il n'est pas possible pour l'instant de gérer les événements des nœuds créés dynamiquement). Pour cela il faut lier le browser et le nœud que l'on veut gérer aux variables Java en utilisant les deux méthodes que nous venons d'illustrer. Ensuite chaque événement (entrant ou sortant) que l'on veut gérer doit à son tour être lié à une variable Java grâce aux méthode *getEventIn*( ) et *getEventOut*( ). Voici un exemple de ce lien :

```
(...)
Browser Variable_Browser;
Node[] Material;
Node[] Transform;
EventInSFColor diffuseColor;
EventOutSFVec3f translation;

  Variable_Browser = Browser.getBrowser( );
```

```
   Material = Variable_Browser.getNode("Material");
   Transform = Variable_Browser.getNode("Transform");
   diffuseColor = (EventInSFColor)
Material.getEventIn("diffuseColor");
   translation =(EventOutSFVec3f)
Transform.getEventOut("translation");
```

Ce morceau de l'applet Java permet de créer le lien entre l'applet et le monde VRML. Il permet aussi d'associer deux nœuds du monde VRML à deux variables Java de type Node[]. Et enfin d'associer deux événements de ces nœuds à des variables Java de types "événements".

Comme on peut le remarquer dans cet exemple, dans l'applet Java en plus de préciser si on a affaire à un événement entrant ou sortant il faut également préciser son type VRML. Il faut faire attention à un élément important qui diverge entre Java et VRML. Un EventIn dans Java correspond à un événement entrant dans un nœud VRML et donc à un événement sortant de l'applet. Et réciproquement pour les EventOut. Tous les événements que l'on va chercher dans le monde VRML grâce à la méthode getEvent (In ou Out) doit subir un casting pour être adapté au type Java correspondant. Le langage Java est un langage orienté objet. Lorsqu'on accède à un objet en utilisant une méthode et que l'on place la valeur de cet objet dans une variable, Java ignore le type retourné par la méthode et renvoie un type objet général. Il faut donc convertir cet objet général en l'objet particulier que l'on veut utiliser. Le mécanisme de casting consiste à faire cette conversion. L'exemple suivant illustre ce mécanisme de casting :

translation =(EventOutSFVec3f) Transform.getEventOut("translation");
mécanisme de casting — renvoie un objet général

A chaque type VRML correspond un type Java. Ces types Java sont en fait des classes qui contiennent des méthodes. L'index général de l'EAI reprend toutes ces classes et les expliquent; il peut être consulté à l'url suivante : http://vrml.sgi.com/moving-worlds/spec/ExternalInterface.html.

Comment fonctionne l'ajout ou la suppression dynamique d'une série de nœuds suite à l'utilisation de l'EAI? L'ajout ou la suppression d'objet VRML ne peut se faire que dans des nœuds dits de regroupement qui possèdent les événements "***addChildren***" et "***removeChildren***". Il faut donc qu'un de ces nœuds soit défini dans le monde VRML et qu'il possède un nom. Ensuite la gestion se fait comme celle d'un événement normal. On lie le browser, le nœud et les événements aux variables Java. Et on leur attribue des valeurs.
Pour créer un nouvel objet on utilise la méthode *createVrmlFromString*( ) de la classe Browser. Ensuite, on passera la valeur au browser qui pourra ainsi modifier le monde VRML grâce à la méthode *setValue*( ). Pour la suppression, on travaille de la même façon. Lorsque l'on veut supprimer un nœud fils d'un nœud de regroupement particulier, le browser va comparer tous les fils à celui que l'on voudrait supprimer. S'il trouve un nœud identique, il le supprime sinon il ignore la commande. L'exemple suivant illustre ces deux grands principes.

```
(...)

Node[ ] NoeudVRML;
Node [ ] Group;
Browser browser;
EventInMFNode addChildren;
```

```
EventInMFNode removeChildren;

(...) cette partie se fait en général dans l'initialisation de
l'application
browser = Browser.getBrowser( );
Group = browser. getNode("Groupement");
addChildren = Group .getEventIn("addChildren");

(...)
try
{
  NoeudVRML = browser . createVrmlFromString(
        "Transform{ \n"+
        "     translation 2 5.6 8 \n"+
        "     rotation 0 1 0  -0.57 \n"+
        "     children[ \n"+
        "          Shape{ \n"+
        "               appearance Appearance{ \n"+
        "                    material Material{ \n"+
        "                         diffuseColor .5 .5 .5 \n"+
        "                    } \n"+
        "               } \n"+
        "               geometry Cylinder{ \n"+
        "                    radius 1 \n"+
        "                    height 6 \n"+
        "               } \n"+
        "          } \n"+
        "     ] \n"+
        "} \n");
}
catch(InvalidVrmlException e) {}

addChildren .setValue(NoeudVRML); (on ajoute le nœud dans la scène
VRML)
(...)
```

L'exemple précédent permet d'ajouter un ensemble de nœuds dans le monde VRML. Les différents nœuds que l'on ajoute vont modifier la scène VRML. En effet, dans le cas de l'exemple précédent on ajoute une forme géométrique. Ce qui a pour effet, dans la scène VRML, d'ajouter un cylindre de couleur mauve, de longueur 6 et de rayon 1. Ce cylindre aura été placé en position 2 | 5.6 | 8 dans le repère à trois dimensions de la scène et il aura également subi une rotation de 90 degré sur l'axe des Y. Il ressemblera donc à la Figure 22.



Figure 22 : une représentation du cylindre ajouté à la scène VRML

On peut constater que l'utilisation de l'EAI multiplie le nombre de variables Java de façon assez importante. La liste suivante précise les variables qui sont nécessaires dans le cadre d'une utilisation de l'EAI :

- Une variable pour l'instance du browser VRML;

- Une variable par nœud avec lequel on désire travailler. Chacun de ces nœuds doit naturellement exister dans le monde VRML mais il doit aussi avoir un nom dans ce monde;
- Une variable par événement que l'on veut traiter pour chacun des nœuds.

Il est donc très important de choisir correctement le nom de chaque variable Java afin de savoir rapidement à quel objet du monde VRML elle est liée.

VIII. Architecture théorique d'un browser VRML

Dans les sections précédentes nous avons parlé souvent du browser VRML. Cette section présente la structure de celui-ci. Cette architecture est celle présentée par Chris Marrin[9] dans **[MARRIN96]**. Un browser VRML est un interpréteur des fichiers VRML. Il va lire le contenu d'un fichier, l'interpréter et l'afficher dans un visualisateur. C'est le browser VRML qui fait le lien entre le fichier VRML et les fichiers contenant les programmes pour gérer le dynamisme. Dans le cas de routage d'événements, c'est en fait le browser qui reçoit les événements générés et, grâce à la partie routage contenue dans le fichier VRML, peut indiquer au visualisateur quels sont les changements qui doivent avoir lieu dans la scène.

Un browser VRML théorique contient plusieurs points qui nécessitent une interface particulière. On peut le voir sur le schéma de la Figure 23 que nous allons commenter.

Φ Les "Authoring Interface": le *Script Authoring Interface* est l'interface entre le programme (ou script) référencé par un nœud *Script* et les fonctionnalités du browser qui permettent l'interaction entre le nœud et le script lui même. Cette interface est décrite dans les spécifications du langage VRML 2.0. L'*External Authoring Interface* est l'interface qui se trouve entre le browser VRML et un mécanisme externe comme CosmoPlayer[10]. Cette interface ne fait pas encore partie des spécifications de VRML car elle est toujours en voie de standardisation.

Φ Les interfaces programmeurs : le *Browser Programmer's Interface* est l'interface se trouvant entre le browser VRML et le monde extérieur. Il peut simplement s'agir de la connexion vers l'External Authoring Interface si le browser fonctionne seul. Il peut également être l'interface vers une application parent comme un plugin de Netscape. L'*Interpreter Programmer's Interface* permet à un développeur de créer un interpréteur pour un langage qui serait utilisé avec le nœud *Script*. Cela permettrait qu'un composant standard (DLL, DSO,...) fourni par un développeur soit utilisé par le browser pour interpréter correctement le langage. Plusieurs propositions de spécifications pour cette interface commencent à apparaître sur le Web.

[9] Chris Marrin travaille chez SGI et est fort impliqué dans tout ce qui concerne l'EAI. Il a d'ailleurs proposé des spécifications pour celle-ci qui sont pour l'instant soumises au VAG pour être reconnues comme la norme EAI de VRML 2.0. C'est dans le cadre de l'élaboration de ces spécifications qu'il a conçu l'anatomie d'un browser VRML présentée dans ce document.

[10] CosmoPlayer est un outil développé par Silicon Graphics qui permet de visualiser et de se promener dans des mondes VRML. Il s'agit de ce que nous appellerons un visualisateur VRML (traduction du terme "viewer"). Ce visualisateur est distribué sous forme de "plugin", ce qui lui permet d'être intégré au navigateur Internet des utilisateurs.

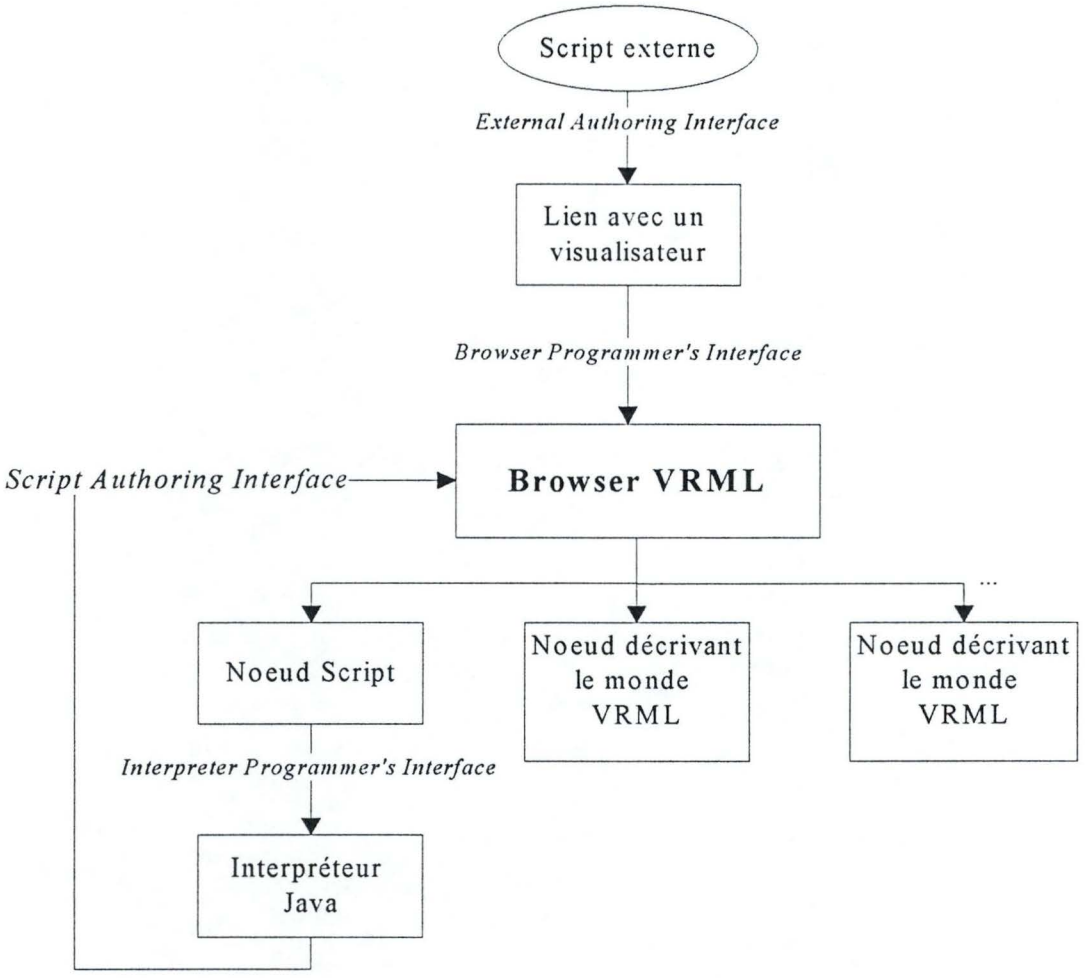


Figure 23 : l'anatomie d'un browser VRML

Cette structure n'est pas un élément qui gèle le design d'un browser VRML. Elle permet surtout de réaliser les endroits qui nécessitent encore d'être normalisés pour que VRML et toutes les possibilités qui l'entourent puissent enfin s'étendre à travers le Web de façon uniforme.

IX. Conclusion

La description du langage VRML clôture la première partie de ce mémoire consacrée aux généralités. Nous pouvons à présent passer à la description de la conception du simulateur et de l'application que nous avons créée.

Partie 2 - Analyse et conception

Chapitre V : La conception pédagogique

I. Introduction

Dans ce chapitre, nous allons décrire les éléments préalables à la conception de notre application. Cette dernière s'inscrivant dans une optique de formation à la technologie ATM, il est important de bien cerner l'application avant de l'implémenter. Nous reprendrons ici des notions décrites dans [CHAVA96] qui s'est lui-même inspiré d'un livre de Kel CROSSLEY et Les GREEN traitant du design des didacticiels.

Dans une première partie de ce chapitre, nous aborderons les éléments de base de la conception pédagogique de tout didacticiel.

Ensuite, nous décrirons la façon dont nous avons répondu à ces critères basiques lors de la conception de notre application.

Troisièmement, nous justifierons certains des choix que nous avons faits dans l'optique que les acteurs du projet P2-Verso souhaitaient donner au module de formation en général et à notre application en particulier.

Nous terminerons par une conclusion.

II. Les éléments de base de la conception pédagogique

Lorsque l'on décide de mettre en place un logiciel devant servir à de l'enseignement assisté par ordinateur (EAO), il est très important de concevoir pédagogiquement ce dernier en ayant recours à une certaine méthode. Les éléments principaux dont il faut tenir compte sont décrits dans les paragraphes suivants.

Tout le monde sera d'accord pour dire que les personnes les plus aptes à établir le canevas que devra suivre le didacticiel sont les professeurs. En effet, non seulement ils peuvent être considérés comme experts du contenu du didacticiel mais en plus, ils peuvent mettre en évidence les points particuliers pour lesquels les élèves rencontrent traditionnellement des problèmes ou des difficultés. Cependant, il est très important de se rendre compte que la situation change du tout au tout. On ne se trouve plus dans le contexte du professeur face à son élève mais d'un élève seul, face à un ordinateur et à une matière qu'il doit mémoriser et, surtout, qu'il doit comprendre. Le contexte d'apprentissage est différent et si l'on souhaite que les résultats obtenus avec l'utilisation du didacticiel soient satisfaisants, il faut impliquer l'élève au maximum dans sa formation et ne pas lui attribuer un rôle passif (comme c'est souvent le cas dans une salle de classe). Il est donc important de considérer l'apprenant comme un sujet actif de son propre apprentissage.

On peut déterminer quatre conditions préalables à la réalisation correcte d'un logiciel d'EAO.

- Les concepteurs de l'application devront certainement travailler en groupe, ce qui nécessite un esprit ouvert à la collaboration et à la négociation. En effet, la création d'un tel logiciel requiert la collaboration d'experts de contenu, d'informaticiens, d'ergonomes et de pédagogues

- Certains membres du groupe de conception doivent posséder une grande expertise de la matière à enseigner. Sans cette expertise, il est impossible de maîtriser correctement le contenu et donc, il est impossible de l'enseigner correctement
- Un membre du groupe de conception au moins devrait avoir enseigné la matière. Il pourra ainsi mettre en évidence les points cruciaux de la matière et les points difficiles pour les étudiants
- Personne dans l'équipe de conception ne devrait mettre sur la table ses convictions quant à ce qui est possible de réaliser ou non avec l'informatique. On peut facilement imaginer qu'une telle personne bloque complètement le processus de développement.

Pour s'assurer que l'application porte ses fruits, il faut essayer d'utiliser certains éléments pédagogiques considérés comme capitaux lors de la réalisation d'un logiciel d'EAO. Les futurs utilisateurs de l'application devront découvrir quelque chose de nouveau. Le logiciel doit leur offrir la possibilité de recommencer plusieurs fois de suite l'animation ou l'expérience reproduite. L'utilisateur peut ainsi acquérir de nouvelles connaissances suite à la compréhension des événements qui lui sont présentés. L'utilisation des graphismes et des couleurs va permettre une mise en évidence des concepts illustrés par l'application. Il est donc capital de les utiliser au mieux en respectant les différentes règles ergonomiques qui leur sont associées.

Le logiciel peut avoir différents rôles en fonction de la manière dont il sera conçu et utilisé par l'apprenant. Il pourrait par exemple servir à introduire de nouvelles notions, à les renforcer, à aider à la compréhension de concepts plus complexes, etc.

Pour qu'un élève apprenne un fait nouveau, nous ne pouvons nous baser sur ce qu'il connaît puisque chaque élève est différent et réfléchit différemment. Le didacticiel doit être le plus ouvert et le plus interactif possible pour permettre à l'étudiant d'en explorer un maximum d'aspects possibles et, par là même, de retenir et de comprendre ce qu'il voit.

Si l'on se place maintenant au point de vue de l'apprenant, on se rend compte que plusieurs éléments peuvent être très importants pour lui. Le rôle qu'il joue, l'environnement dans lequel le logiciel le transpose ou encore le rapport qu'il peut avoir avec le temps sont des éléments à retenir. Plus l'étudiant prend une part active à son propre apprentissage, moins vite il se lassera du didacticiel. Son intérêt pour le contenu qui lui est enseigné a alors beaucoup plus de chances d'être élevé. L'environnement qui sera présenté à l'apprenant sera peut être un élément totalement nouveau pour lui. Le logiciel doit alors offrir la possibilité de définir des angles de vue particuliers et considérés par les concepteurs comme intéressants dans l'apprentissage. Quant aux rapports possibles avec le temps, ils peuvent être multiples. On peut accélérer les choses, les ralentir, voyager dans le passé ou dans le futur, voir même suspendre l'écoulement du temps en offrant un "arrêt sur image".

L'interface que l'on va présenter à l'élève joue un rôle primordial dans son apprentissage. C'est à travers les différents écrans que l'utilisateur rencontre qu'il a la possibilité d'agir sur son environnement ou sur le temps. C'est en utilisant l'interface que l'utilisateur pourra exprimer ses envies et ses besoins, qu'il pourra jouer le rôle qui lui est donné. L'écran principal est le point de départ, la base de la découverte. L'apprenant pourra y agir sur l'information qui lui est présentée et sélectionner l'une ou l'autre des éventuelles options de l'application. Enfin, il permettra d'illustrer l'expérience ou de présenter l'animation à l'étudiant. L'écran principal devra attirer l'attention de l'élève grâce aux animations. On devra y mettre en place certaines couleurs qui représenteront un codage facilement interprétable par

l'élève. En y affichant des textes simples et brefs, l'écran deviendra source d'informations nouvelles ou d'éclaircissement de concepts déjà acquis. Pour concevoir cet écran, il faut prévoir les événements qui seront associés à chacune des actions de l'élève. Il faut donc décider quelles seront les commandes qui lui seront offertes et à quel moment ces dernières seront accessibles.

III. La conception pédagogique de notre application

Cette section décrit la façon dont nous avons utilisé les éléments-clé que nous venons de décrire pour établir la conception pédagogique de notre application.

Dès le début du projet P2-Verso, nous avons choisi de concevoir un simulateur. En général, les simulations permettent de modéliser un processus ou une situation particulière. Dans notre cas, il s'agit de modéliser le fonctionnement de la technologie ATM et plus particulièrement certains concepts de base. Ce simulateur sera intégré à un module de formation plus large sur la technologie ATM. Ce module sera accessible par les étudiants au moyen du Web. Il a donc été conçu dans le sens d'un didacticiel et l'implication de l'étudiant dans son apprentissage sera maximale.

Parmi les participants au projet P2-Verso, nous pouvions compter plusieurs experts de contenu. Certains enseignaient dans des écoles dites "normales", d'autres dans des téléuniversités. De ce fait, l'expertise nécessaire était garantie. Ayant nous même suivi des cours traitant de cette technologie aux FUNDP, nous pouvions indiquer les points difficiles pour les étudiants. Pour le module de simulation, le travail de groupe fut toujours présent. De la conception pédagogique à l'implémentation en passant par l'élaboration du cahier des charges, nous étions toujours au moins deux à travailler sur le simulateur, ce qui nécessite de nombreuses négociations et discussions. En ce qui concerne les connaissances informatiques, les seules conditions qui ont été mises sur pied bien avant la conception pédagogique étaient l'utilisation du Web et des langages Java et VRML. Le fait que nous ne savions pas à l'époque ce qu'il était possible de faire ou non avec ces langages nous a permis de ne pas mettre trop en avant nos convictions d'informaticiens. Nous pouvons donc dire que nous respections les quatre conditions préalables au développement d'un didacticiel.

Nous avons voulu donner au simulateur un rôle d'illustration de concepts de base. Les concepts choisis sont les suivants :

- La commutation
- Le multiplexage
- Les unités de données
- Les couches

Le simulateur servira donc principalement à renforcer les différentes connaissances acquises par l'étudiant dans les chapitres de base du module de formation de la technologie ATM. Il permettra aussi une concrétisation et une animation de concepts jusque là abstraits et statiques.

Nous offrons à l'apprenant la possibilité d'agir sur son environnement et sur l'animation qui lui sera présentée. Le langage VRML va nous permettre de présenter le réseau ATM dans un environnement à trois dimensions. L'utilisateur pourra voyager comme il l'entend dans ce

monde même si nous lui proposons certains points de vue plus "stratégiques". Les actions que l'utilisateur pourra avoir sur l'animation seront présentées grâce à la métaphore du magnétoscope, cela implique qu'il pourra avancer, reculer, démarrer, mettre en pause ou stopper l'animation. Cette façon d'agir nous permet d'introduire une interactivité qui impliquera l'utilisateur dans son propre apprentissage. De plus, nous ne limitons pas le nombre d'animations que l'utilisateur peut voir. Il pourra de la sorte observer les animations autant de fois que nécessaire à sa compréhension des concepts illustrés.

Dans l'interface, les informations fournies par l'animation seront complétées par un texte bref ce qui offre un point de repère supplémentaire à l'apprenant. C'est dans celle-ci que l'utilisateur pourra poser ces choix. Les différents choix qu'il peut faire sont les suivants :

- Utiliser le magnétoscope pour jouer sur l'animation ;
- Changer la vitesse de l'animation ;
- Choisir le concept qu'il veut observer ;
- Naviguer dans le monde en trois dimensions ;
- Accéder aux points de vue définis.

On suppose que la façon d'utiliser le simulateur et de poser les actions que nous venons de décrire aura été présentée de façon succincte dans le module de formation. Dès lors, nous n'avons pas besoin de donner à l'utilisateur d'informations complémentaires à ce sujet.

IV. Les choix

Cette partie décrit les différents choix que nous avons faits lors de l'élaboration du simulateur, ainsi que la justification de ces choix.

IV.1 Généralités

La métaphore du magnétoscope ne sera pas complètement utilisée. En effet, nous estimons que pour comprendre les concepts de base que nous devons décrire, il n'est pas utile de pouvoir revenir en arrière ni même de pouvoir avancer. Chaque animation forme un tout difficilement divisible en parties distinctes pour lesquelles il serait possible de passer de l'une à l'autre. L'utilisation de la "marche arrière" pourrait même créer une certaine confusion dans les concepts. Dès lors, nous offrons à l'utilisateur la possibilité de lancer l'animation, de la stopper quand il veut et de la mettre en état de pause, ce qui lui permet un arrêt sur image et un délai de réflexion plus long sur les informations qu'il peut observer à ce moment précis. Nous donnerons cependant la possibilité de jouer sur la vitesse d'écoulement du temps, ce qui permettra des animations plus ou moins rapides en fonction des desiderata de l'utilisateur.

L'interface qui sera présentée à l'utilisateur sera composée d'un et un seul écran. Celui-ci sera divisé en plusieurs parties. Une d'entre elles servira à présenter à l'élève le réseau ATM dans un environnement tridimensionnel et les autres lui permettront d'agir sur l'environnement ou de poser ses actions sur l'animation. Dans cette interface il faudra offrir à l'utilisateur la possibilité de sélectionner lui-même le concept qu'il souhaite illustrer.

Dans le simulateur, nous utiliserons un maximum de couleurs. Cependant, nous veillerons à limiter celles-ci à un nombre supportable pour l'œil humain. De même, les couleurs seront utilisées pour représenter différents concepts. A chacun de ces concepts sera associée une et

une seule couleur tout au long des différentes animations. En plus d'être cohérentes dans les diverses animations, ces couleurs coïncideront avec les choix de couleur repris dans le module de formation. En utilisant toujours des couleurs identiques pour illustrer des phénomènes identiques, nous faciliterons la capacité de rétention de l'apprenant et nous lui permettrons de trouver des points de repères communs à travers tout le module de formation.

IV.4.2 La représentation du réseau dans le monde en trois dimensions

Nous avons choisi de représenter le réseau ATM dans un environnement tridimensionnel. Cependant, cela implique plusieurs choix possibles. Nous avons pensé présenter toutes les machines sous forme d'un ensemble de couches. Au départ, cette idée n'était pas admise par tous les membres du groupe. En effet, le modèle en couche est assez éloigné de la réalité informatique. Un des experts de contenus avait remarqué que, dans ses cours, peu d'élèves étaient capables de dire quels était les composants matériels de leur ordinateur qui réalisaient le travail de chacune des couches. Ce professeur aurait aimé qu'on associe à la représentation en trois dimensions, la réalité mécanique des ordinateurs.

Si nous avions choisi cette solution, il aurait été difficile de mettre en évidence de façon tout à fait précise et claire les composants machine qui représentaient les couches. De plus, il semblait à la plupart des membres du projet que ce lien pouvait être expliqué aux étudiants en utilisant un autre moyen que la simulation. Nous avons donc décidé de présenter le réseau comme un ensemble de couches dans le simulateur et de consacrer un chapitre du module de formation à l'association "machine - couche".

Nous devons admettre que le modèle en couches utilisé n'est pas le modèle communément admis pour présenter la technologie ATM. En effet, la décomposition standard reconnue par l'UIT-T ne rentre pas telle quelle dans la norme des couches ISO. Selon [PUJOLL97] *"...L'architecture des réseaux à commutation de cellules peut être dite compatible au modèle de référence ISO. Cependant, les fonctionnalités ne sont pas regroupées aux mêmes niveaux que celles de l'ISO. (...)La raison plus profonde au besoin de ce nouveau modèle, dit modèle UIT-T, provient de la nécessité de prendre en charge les applications multimédia, c'est à dire de la superposition de la voix, des données et de l'image."* Ce modèle communément admis dans le domaine des télécommunications est représenté à la Figure 24.

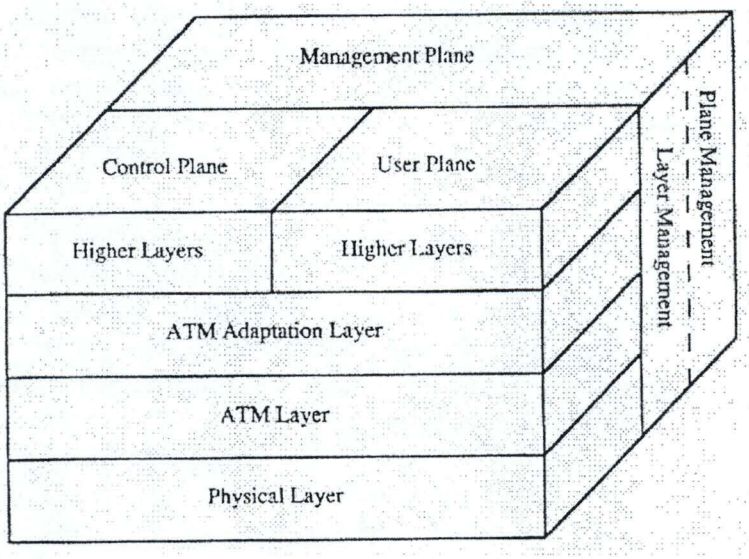


Figure 24 : Le modèle en couche pour ATM

Pour la représentation du réseau ATM dans l'environnement 3D, nous négligerons la notion de plan ("plane" dans la Figure 24). Nous ne présenterons que les trois couches "basses" ainsi qu'une couche supérieure que nous appellerons "couche Application". Nous considérons que cette représentation est suffisante pour illustrer les concepts de base choisis. Le modèle UIT-T présenté à la Figure 24 et son fonctionnement seront décrits longuement dans le module de formation.

Dans la représentation graphique du réseau ATM, les liens sont représentés par une seule liaison physique entre deux machines. Il est possible de voir le support physique comme une juxtaposition de plusieurs VP, eux-mêmes composés de plusieurs VC. Cependant expliquer correctement cette composition et ces juxtapositions ne nécessite pas forcément l'emploi d'un environnement dynamique. Il est possible d'insérer une image en trois dimensions dans le module de formation qui représente correctement cette caractéristique des réseaux ATM. Etant donné qu'il n'y a pas besoin de joindre le moteur de simulation à cette matière pour que l'utilisateur du module la comprenne, dans la représentation tridimensionnelle, nous n'entrerons pas dans le détail de la composition des différents liens physiques et des différents VP.

IV.3 La couche AAL

La couche AAL est probablement la plus complexe des couches dans les réseaux ATM. Au moment du déroulement du projet P2-Verso, elle n'était pas encore complètement définie. Des changements perpétuels y étaient apportés. Ces différents changements ne pouvaient qu'augmenter le niveau de complexité de cette couche. ATM étant en soi une technologie vaste et complexe, il était impossible de tout représenter. Nous avons donc décidé, suite à la complexité qui y est liée, de ne pas développer dans le simulateur la couche AAL. De nombreuses références et informations sont disponibles dans le module de formation, mais aussi dans la bibliothèque de liens Internet qui sera proposée à l'apprenant.

IV.4 Le mode de connexion

Comme nous le rappelons dans le chapitre III, il existe deux modes de connexions dans les réseaux à technologie ATM. Les connexions peuvent être commutées ou permanentes. La PVC (Permanent Virtual Connection) nécessite une programmation manuelle des commutateurs pour créer les circuits entre les différents équipements. Selon [MELIN97],"... *Cette méthode, utilisée au début de l'ATM, était très simple mais assez inefficace pour des réseaux importants (au-delà d'une dizaine de commutateurs ATM) et pour supporter l'allocation dynamique nécessaire à de nombreuses applications."*

Il devient vite très important pour les applications utilisateurs de pouvoir créer des connexions à la demande. C'est ce qu'on appelle le mode de connexion permuté. Les circuits virtuels commutés (Switched Virtual Circuit) sont créés à travers le réseau grâce à des mécanismes de signalisation, d'adressage et de routage bien particuliers et décrits dans [MELIN97]. Nous avons décidé de ne pas aborder ce mécanisme complexe dans le mémoire. De même, lors de l'élaboration de la conception pédagogique du module de formation ATM, les différents membres du projet P2-Verso ont décidé de ne pas développer ce mécanisme dans le simulateur. Une des raisons qui ont poussé à faire ce choix est que la plus grande majorité des circuits ATM existants en France et à Montréal sont des circuits permanents et donc, que l'expérience pratique des membres du projet à propos des SVC est limitée. Les membres du projet ont souhaité illustrer le principe mis le plus souvent en œuvre dans la pratique afin que

les apprenants le maîtrise au mieux. De plus, il est prévu de consacrer un chapitre du module de formation à ce problème. Le simulateur ne doit illustrer que les principes de base du réseau ATM. Le module de formation ne tenant pas compte du protocole de signalisation des SVC, il aurait été assez inopportun de développer celui-ci dans le simulateur.

IV.5 La cellule ATM

La cellule ATM possédera toujours une longueur de 53 octets, cette taille étant le résultat d'un compromis entre ce que souhaitaient les européens et les américains. Cette cellule se décompose en deux grandes parties : l'entête et la charge utile. Comme nous l'avons déjà mentionné plus haut, le format de la cellule varie suivant que l'on se trouve dans l'interface réseau/utilisateur (UNI) ou dans l'interface réseau/réseau (NNI). Cette variation réside principalement dans les champs de l'entête de la cellule. Dans le simulateur, nous ne présenterons que le format présent dans l'interface UNI. Ce choix est purement arbitraire et ne résulte d'aucun niveau de complexité. Le format de cellule qui sera présenté à l'apprenant dans le simulateur est celui présenté à la Figure 25.

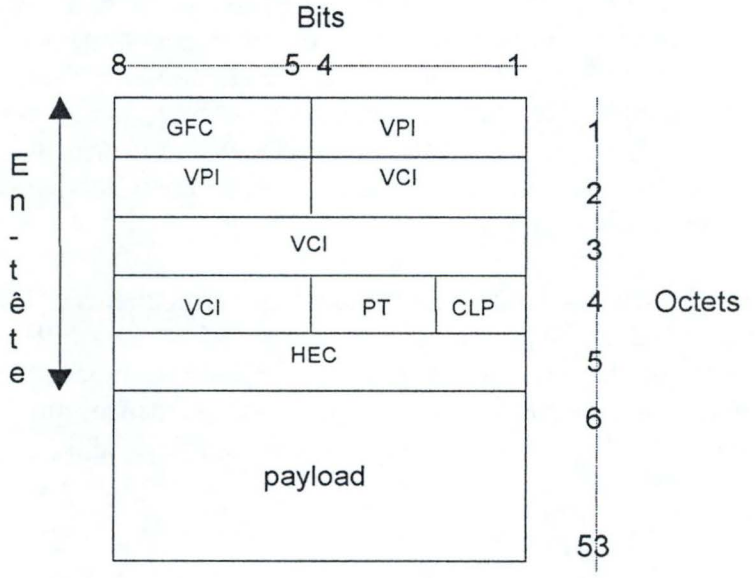


Figure 25 : Le format de cellule présenté dans le simulateur

IV.6 Les concepts de base

Dans cette section, nous allons présenter au lecteur ce que les membres du projet P2-Verso aimeraient voir apparaître dans le simulateur pour illustrer les quatre concepts de base des réseaux à technologie ATM. Il s'agit des concepts de commutation, de couches, d'unités de données et de multiplexage.

- *Le concept de commutation*

  L'objectif principal de l'illustration du concept de commutation est de montrer à l'apprenant comment un commutateur qui reçoit une cellule sait sur quel lien physique ou sur quelle liaison il doit l'envoyer. Pour cela, il est important de montrer à l'apprenant que le commutateur possède des tables de commutation fixes (étant donné

que l'on travaille en mode de connexion permanent) et que l'analyse d'une certaine cellule lui permet de consulter une ligne particulière de cette table. En fonction du résultat de l'analyse de cette ligne, le commutateur sait sur quel lien il doit envoyer la cellule.

Il est nécessaire de bien mettre en évidence le lien existant entre la cellule analysée et la ligne de la table de commutation consultée. Nous pouvons ici choisir d'utiliser la couleur pour exprimer ce lien. Pour que l'élève comprenne correctement cela, il est important de traiter des cellules appartenant à des VCC différents. On suppose que le simulateur présentera à l'apprenant un minimum de 2 VCC.

- *Le concept de couches*

Dans le concept de couches, nous avons choisi de montrer plusieurs éléments bien particuliers.

Le premier objectif est de mettre en évidence l'ordonnancement qui existe dans la façon dont les couches se succèdent pour véhiculer des cellules à travers le réseau. Il faut en fait illustrer que si une couche est de niveau N, elle est utilisatrice des services qui lui sont fournis par la couche N+1 et elle est fournisseur de services à la couche N-1. En travaillant de cette façon, nous tâcherons de montrer comment est construite la cellule ATM par les différentes couches. C'est à dire, nous montrerons entre autre que la couche AAL se charge de segmenter la masse d'information en petits "tas" de 48 octets, que la couche ATM ajoute à ces "tas" un entête les transformant ainsi en cellules ATM et que la couche physique calcule le champ de contrôle d'erreur et adapte le flux d'information au support physique.

Chaque couche remplit des fonctionnalités bien particulières. Comme nous l'avons dit plus haut au sujet de la présentation du réseau ATM, le modèle en couche associé à cette technologie ne correspond pas exactement au modèle ISO. Cependant, le simulateur ne montrera pas à l'apprenant les correspondances que l'on peut trouver entre ces deux modèles. Ce parallélisme sera détaillé dans le module de formation. Il n'est en effet pas nécessaire de disposer d'une animation pour expliquer correctement ce point de la matière.

Dans le simulateur, on essaiera d'une façon ou d'une autre de mettre en évidence la présence de la couche AAL uniquement dans les machines terminales. On peut pour cela utiliser un code de couleur bien particulier qui permettra de réaliser cette mise en évidence.

- *Le concept d'unités de données*

Dans le concept d'unités de données, on veut que le simulateur atteigne principalement deux objectifs.

Tout d'abord, nous voulons montrer à l'utilisateur que la partie "données" des cellules n'est jamais transformée pendant tout le temps que les cellules sont véhiculées à travers le réseau. Nous voulons montrer que la cellule est acheminée dans le réseau un peu comme le serait un paquet à la poste. C'est à dire que seule l'adresse de livraison du paquet (correspondant à l'entête de la cellule) est analysée, voir modifiée et qu'on ne s'intéresse pas au contenu du paquet (correspondant à la partie données). Ce premier point est associé à un autre un peu moins principal. Il faut que le simulateur permette à

l'élève de mémoriser le format et la taille de la cellule. Ces derniers auront déjà été introduits dans le module de formation.

Le deuxième objectif est de montrer quels sont les champs de l'entête de la cellule qui sont modifiés lors du transfert des cellules et à quel moment.

Bien que cela ne relève pas uniquement de l'animation liée au concept d'unité de donnée, il faudra mettre en évidence (dans cette animation ou dans celle du concept de couches) la méthode de construction de la cellule.

- *Le concept de multiplexage*

  Pour ce concept, nous mettrons particulièrement en évidence le multiplexage de flux, le multiplexage "temporel" pouvant être représenté lors de l'animation du concept de commutation. Pour ce faire, nous enverrons d'une seule machine source des trains de cellules appartenant à des flux de types différents (voix, vidéo, données).

  Pour les membres du projet P2-Verso, faire la distinction entre multiplexage de flux et multiplexage "temporel" est important. Personnellement, nous avouons ne pas voir la différence qui peut exister entre ces deux types de multiplexage si ce n'est que dans un cas les trains de cellules émaneraient de machines sources différentes et dans l'autre ces trains émaneraient d'une seule machine. Nous avons donc admis que la différence résidait bien dans le nombre de machines sources.

V. Conclusion

Comme nous venons d'établir les différents objectifs pédagogiques que devrait atteindre notre application, nous pouvons maintenant passer au cahier des charges. C'est dans ce dernier que les choix finaux de l'application vont être fixés tout en tenant compte bien entendu des objectifs qui lui ont été assignés.

Le chapitre suivant présente au lecteur cette analyse fonctionnelle du simulateur. Il présente également les premières maquettes de l'interface grâce auxquelles l'utilisateur de l'application pourra agir sur son environnement.

Chapitre VI : Analyse fonctionnelle

I. Introduction

Pour réaliser notre simulateur, nous nous sommes basés sur le cahier des charges qui avait été élaboré dans le cadre du projet P2 Verso. Cependant, étant donné les limites de temps qui nous furent imposées, nous avons dû limiter ce dernier et faire des choix. Ce chapitre explique le cahier des charges ainsi que les choix que nous avons faits.

Dans un premier temps, nous aborderons les éléments généraux que le simulateur devra traiter. C'est à dire, nous parlerons des restrictions qui ont été posées dès le départ, du public qui est visé par le simulateur et de la façon dont celui-ci doit pouvoir s'adapter à ce public. Enfin, nous aborderons les différentes contraintes que nous devions satisfaire.

Dans une seconde section, nous expliquerons les grands concepts que le simulateur devait illustrer et comment cette illustration devait être réalisée.

Nous terminerons ce chapitre par une brève présentation des maquettes de l'interface du simulateur. Nous y décrirons l'interface générale et les différences qui apparaissent entre les animations distinctes des différents concepts. Nous n'entrerons pas ici dans les détails du fonctionnement et de l'implémentation des applets, ce point étant abordé dans le chapitre relatif à l'analyse technique.

II. Les éléments généraux

Le simulateur que nous avons conçu est inséré dans un module de cours général sur les réseaux ATM. De ce fait, le simulateur va principalement servir de mécanisme d'illustration de certains concepts de base expliqués dans les premiers chapitres de ce module de formation. Une des hypothèses de départ était que le public cible possédait des connaissances de base dans le domaine des télécommunications et des réseaux informatiques. Nous avons supposé que les personnes qui utiliseraient le simulateur possédaient les mêmes connaissances de base. Il ne servait donc à rien de redéfinir les notions de couches, de commutateurs, de machines terminales ou de liens dans ce dernier.

Les principaux concepts que le simulateur devait illustrer au départ étaient au nombre de quatre. Il s'agissait des concepts de commutation, d'unités de données, de couches et de multiplexage. Suite à l'évolution du projet P2 Verso et à la difficulté de définir ce que les acteurs principaux attendaient exactement de l'illustration du concept de multiplexage, celui-ci fut mis de côté et nos efforts se portèrent essentiellement sur les trois autres concepts.

Comme nous le rappelons dans le chapitre relatif à la technologie ATM, dans le simulateur que nous avons conçu, seules certaines fonctionnalités de la technologie ATM ont été implémentées. En effet, la technologie ATM est une technologie vaste et complexe. Il existe de nombreux éléments qui nous auraient permis de mettre en évidence les avantages et inconvénients de cette dernière et de répondre ainsi aux objectifs principaux du projet P2 Verso. Tout illustrer dans les simulations était tout à fait impossible endéans les délais qui nous étaient fixés. Dès le début du projet, il nous a fallu déterminer les limites du simulateur, c'est à dire les éléments que nous ne développerions pas. Toutefois, le simulateur devait être

pensé de façon assez globale pour pouvoir être complété ultérieurement sans difficultés majeures. Nous allons donc présenter maintenant au lecteur les différentes restrictions qui ont été proposées.

II.1 Les restrictions au cahier des charges

La première des restrictions que nous avons décidée concernait le mode de connexion dans lequel nous allions travailler. En effet, comme nous l'avons déjà précisé dans le chapitre concernant la technologie ATM, il existe avant tout échange d'information entre deux machines l'établissement d'un circuit virtuel. Le protocole de signalisation de cet établissement est très complexe et n'est pas détaillé dans ce mémoire. C'est lors de l'établissement de ce circuit que l'appelant va préciser son débit moyen, maximal et d'autres caractéristiques que nous avons déjà citées plus haut. Or, dans les réseaux de technologie ATM, il existe des connexions permanentes et des connexions commutées. Les connexions permanentes permettent, par exemple, à des entreprises qui travaillent très fréquemment ensemble de ne pas avoir à établir un nouveau circuit chaque fois qu'elles veulent communiquer. C'est ce genre de connexions qui est illustré par notre simulateur. En effet, l'utilisateur du module ne pourra pas "voir" lors des animations la façon dont le circuit virtuel est créé entre les deux machines communiquantes. Dès le lancement de l'animation, l'utilisateur du simulateur verra l'échange d'information entre deux machines. Cette restriction résulte, entre autres, d'un choix arbitraire fonction du temps de réalisation qui nous était imparti.

La seconde restriction que nous avons imposée au simulateur (et probablement la principale) concerne la couche AAL. Lorsque l'on analyse le fonctionnement en couche des réseaux supportant la technologie ATM, on peut remarquer que la couche AAL va remplir de nombreuses fonctions. Elle permet notamment :

- De définir, lors de l'établissement d'un circuit virtuel, la classe de service à laquelle appartient ce circuit;
- De traiter les erreurs éventuelles de transmission;
- De traiter la perte ou la duplication de cellules;
- D'établir un contrôle des flux;
- De gérer la synchronisation;
- De segmenter les données en cellules.

Illustrer chaque type de service était déjà quelque chose d'impossible en soi si on voulait éviter de mélanger les différents concepts à illustrer. Nous avons donc décidé de ne détailler aucun de ces services en particulier. Dans le cadre du simulateur, la couche AAL est uniquement utilisatrice des services des couches supérieures et la principale fonctionnalité qu'elle remplit est la segmentation des données en cellules et l'assemblage des charges utiles des cellules pour former des informations. De plus, le simulateur devra mettre en évidence le fait que la couche AAL n'est présente que dans les équipements terminaux. Cette deuxième restriction va impliquer la restriction suivante.

Etant donné que nous ne développions pas le fonctionnement de la couche AAL, le simulateur ne peut donc pas illustrer tout ce qui est relatif au traitement d'erreur. En bref, le simulateur est un petit monde parfait dans lequel aucune erreur ne se produit et dans lequel tous les utilisateurs respectent toujours les contrats qu'ils ont passés lors de la création du circuit virtuel existant entre eux et leur correspondant.

Nous avons également choisi de présenter chaque machine (terminale ou non) comme un ensemble ordonné de couches. Chaque couche est représentée par une et une seule couleur tout au long de la simulation.

Etant donné que nous avons décidé d'utiliser la couleur, il est important de respecter des contraintes ergonomiques et de cohérence. Il faut donc éviter de changer de couleur sans raison. Nous avons choisi d'utiliser des couleurs chaudes pour les objets en mouvement et des couleurs froides pour les objets statiques de façon à attirer le regard de l'apprenant sur les animations et les changements qui avaient lieu pendant celles-ci.

II.2 L'adaptation du simulateur au public visé

Comme nous l'avons déjà précisé dans la présentation du projet P2 Verso, la clientèle visée est la suivante :

- Les ingénieurs télécom ou réseaux;
- Les élèves ingénieurs option informatique;
- Les utilisateurs de grands réseaux.

L'objectif principal du projet étant une sensibilisation des apprenants aux possibilités offertes par les réseaux ATM, le module de formation pourra aussi bien être utilisé sur un stand d'exposition que dans une salle de cours ou de travaux pratiques. De ce fait, les acteurs principaux du projet (et futurs utilisateurs) souhaitaient que le module fonctionne d'une façon différente dans chaque cas. Ils pensaient par exemple différencier une utilisation du module dans une classe de cours avec professeur, d'une utilisation du module par un étudiant seul qui approfondit ses connaissances. De même, on suppose que, pour l'emploi du module sur un stand d'exposition, ce dernier devrait offrir une possibilité de fonctionnement autonome une fois lancé. Cette diversité envisagée dans l'utilisation du module a nécessité d'introduire une très grande modularité dans la conception même du simulateur.

Il nous a été demandé de bien différencier le moteur de simulation de la présentation de la simulation, ceci dans le but de pouvoir "coller" une présentation différente en fonction des besoins sans avoir à concevoir un nouveau simulateur. C'est ce que nous avons fait. Nous nous sommes cependant contentés d'un seul type de présentation des animations, celui qui serait probablement le plus utilisé à savoir une présentation qui serait proposée à un apprenant qui utilise le module de formation seul et à distance. La modularité introduite dans le simulateur sera décrite plus en détail dans la partie traitant de l'analyse technique du simulateur.

II.3 Les contraintes

Nous allons maintenant décrire les contraintes que nous devions principalement respecter pour concevoir notre simulateur. Il nous semble important de décrire ces dernières pour permettre au lecteur de mieux comprendre certains des choix que nous avons faits ou des restrictions que nous avons posées. La description des contraintes permet également de cibler de manière plus approfondie la façon dont nous avons essayé de travailler.

La première des contraintes avec laquelle nous avons dû travailler était la contrainte de temps. Le projet P2-Verso et notre stage étaient tous deux limités dans le temps. Les dates de début

et de fin du projet et du stage ne coïncidaient pas. En effet, le projet P2-Verso commençait en octobre 97 pour se terminer en mars 98 alors que notre stage s'étalait de septembre 97 à janvier 98. Le temps qui nous était imparti pour concevoir le simulateur était donc restreint par ces dates butoirs. Tout le cahier des charges a été pensé en tenant compte de ce critère important. Le premier mois de notre stage fut donc essentiellement consacré à l'apprentissage des langages de programmation que nous devrions utiliser pour réaliser notre simulateur. Il nous permit aussi de faire les premiers choix quant à l'orientation que nous voulions assigner au travail à faire. Grâce aux membres du projet présents à Lyon, nous avons pu rapidement mettre en place les premières idées et les premiers objectifs à atteindre. Le cahier des charges du simulateur fut réalisé en grande partie avant que le projet P2-Verso ne commence réellement.

La deuxième contrainte que nous pourrions décrire ici est l'aspect multi-culturel du projet P2-Verso. Comme nous l'avons déjà mentionné plus haut, le projet P2-Verso rassemble différents acteurs provenants de différents environnements culturels. De ce fait, une des grandes difficultés de ce projet était de parvenir à faire collaborer tous ces acteurs sur un projet aussi vaste que la création d'un module de formation. Pour la conception du simulateur, nous avions cependant un avantage qui était que celui-ci serait entièrement conçu et réalisé à Lyon. Cependant, nous devions à tout moment tenir compte des desiderata des autres acteurs et de leurs remarques. Nous avons donc eu souvent à défendre nos choix et nos idées face à ces derniers. Bien qu'étant une contrainte forte, l'aspect multi-culturel du projet P2-Verso était également d'un grand intérêt, car il nous a permis de mieux comprendre qu'une collaboration et un travail de groupe n'étaient pas toujours quelque chose d'aisé. De même, il fut une expérience très enrichissante pour chacun des acteurs.

Le seul moyen que tous les acteurs avaient pour se rencontrer et pour pouvoir collaborer était d'utiliser les outils informatiques mis à leur disposition. Ceci comprenait les vidéoconférences qui avaient lieu trois après-midi par semaine de 14h00 à 17h30. Les vidéoconférences pour lesquelles la présence d'un membre du groupe "simulation" était nécessaire impliquaient une organisation correcte du planning des différents acteurs car elles étaient particulièrement preneuses de temps.

La contrainte de la modularité du simulateur (dont nous avons parlé plus haut) est aussi une contrainte importante. Etant donné que nous avons travaillé dans une optique de programmation "orientée objet", introduire cette modularité fut moins difficile qu'il n'y paraît en premier lieu. De plus, étant donné le temps qui nous était imparti, nous n'avons développé qu'une seule présentation pour notre simulateur, ce qui limitait la portée de cette contrainte.

Le type même du produit que les acteurs du projet P2-Verso désiraient créer représente une contrainte. En effet, le module de formation doit être le plus autonome possible et doit permettre de faire des mises à jours rapides et efficaces. Les contenus doivent être plus ou moins complexes en fonction du degré de connaissance des apprenants. Le module de formation doit permettre de donner un cours sous forme de vidéoconférence ou en utilisant des séquences pré-enregistrées dans le cas où une vidéoconférence se révèle impossible. Le produit devrait être disponible soit sur un serveur WEB soit sur un CD ROM et devra être paramétrable. Dans le cadre du simulateur ce dernier point est le plus important car il nous orienta sur le choix des langages qui seraient utilisés. En effet, il fallait pouvoir intégrer le simulateur, entre autres, dans des pages HTML. Il devait aussi être accessible indépendamment de la plate-forme que l'apprenant pourrait utiliser. Il nous sembla dès lors

judicieux de nous orienter vers le langage de programmation JAVA. Ce choix fut rapidement accepté par tous les acteurs du projet.

Au CEGELY, un de nos prédécesseurs avait déjà conçu un simulateur de réseaux ATM. Cependant, celui-ci avait été implémenté dans un environnement à deux dimensions et dans un langage difficilement intégrable dans une page WEB. Un des objectifs que nous avons donné à notre simulateur est d'offrir à l'apprenant la possibilité d'avoir une visualisation tridimensionnelle du réseau ATM. Il existe de nombreux outils de modélisation 3D sur le marché. Nous avons donc dû faire un choix. Au moment de la conception du simulateur, le langage VRML 2.0 était déjà en grande partie standardisé. Il permettait entre autres d'intégrer facilement des objets en trois dimensions aux page WEB mais il offrait également une possibilité d'interaction avec le langage JAVA. Nous avons donc choisi de concevoir toute la présentation du simulateur dans ce langage. Ceci nous imposa d'apprendre à le manipuler dans les délais de temps impartis.

Nous venons de résumer au lecteur les principales contraintes que nous avons gérées pour la conception du simulateur. Nous allons maintenant détailler les différents concepts à illustrer.

II. Les concepts à illustrer

Comme nous l'avons déjà précisé plus haut, les concepts à illustrer étaient au départ au nombre de quatre. Suite à de nombreuses discussions, nous avons choisi de ne pas développer le concept de multiplexage. Pour décrire les différents concepts, nous nous sommes basés sur le tableau de la Figure 26. Celui-ci a été réalisé par les membres du groupe simulation de Lyon. Nous allons l'expliquer au lecteur.

Concept illustré	Point de vue	Fonction de la couleur	Animation	Contenu de l'applet
Couche	La vue en hauteur	La couche traversée	Au niveau de la couche visitée	Un texte expliquant le traitement de la couche visitée
Unité de données	La vue aérienne	Une connexion VCC	Au niveau de la cellule	Un changement des champs de la cellule
Commutation	La vue aérienne	Deux connexions VCC	Au niveau de deux cellules issues de deux machines différentes	Une représentation des tables de commutation et une mise en évidence des lignes consultées

Figure 26 : Le tableau des concepts illustrés

Chaque ligne du tableau représente un des concepts à illustrer et permet de mettre en évidence les caractéristiques que devra posséder l'animation. Chaque colonne représente une caractéristique particulière de l'animation.

La première colonne présente les points de vue. Un point de vue est un angle de vue particulier sous lequel le réseau apparaîtra à l'apprenant lorsque celui-ci lancera l'animation illustrant un concept choisi par lui. Nous définissons principalement deux angles de vue pour ce tableau. Le point de vue aérien correspond à une vue aérienne du réseau ATM. Etant donné que toutes les machines sont représentées comme un ensemble de couches, lorsque l'utilisateur les observe sous le point de vue aérien, il peut voir le haut des machines et les

liens qui les unissent. Afin de permettre à l'apprenant de distinguer les machines terminales des commutateurs, nous avons inséré le symbole classique des commutateurs sur le sommet de ceux-ci. Le lecteur peut se faire une meilleure idée de la vue aérienne grâce à la Figure 27.

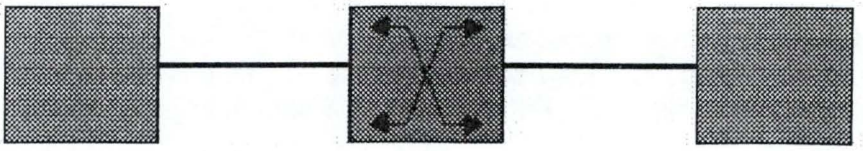


Figure 27 : La vue aérienne

La vue en hauteur permet à l'apprenant de visualiser le réseau ATM comme s'il le regardait de face. Il peut dès lors distinguer les différentes couches (chacune étant représentée dans une couleur différente) des machines du réseau (qu'elles soient terminales ou non). Ce point de vue est illustré à la Figure 28.

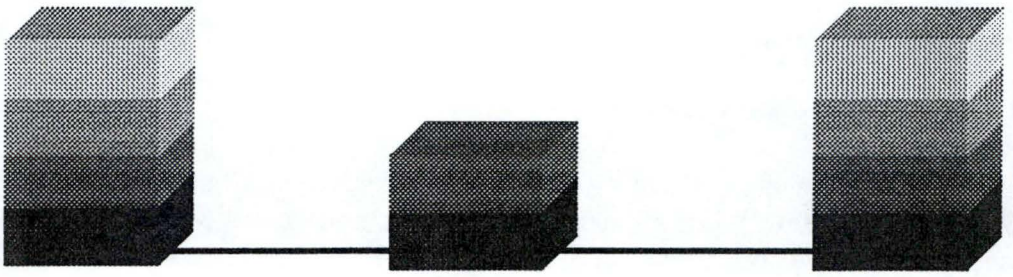


Figure 28 : La vue en hauteur

Lors de la réalisation du simulateur, nous avons introduit un troisième point de vue directement accessible à l'apprenant grâce aux outils de visualisation de VRML 2.0. Ce point de vue correspond à un angle de vue mixte. Il permet de voir à la fois le haut des machines qui composent le réseau (comme dans l'angle de vue aérien) et les entités des couches qui composent chaque machine (comme dans le point de vue en hauteur). Comme ce point de vue "en perpective" est venu s'ajouter après la création du cahier des charges nous n'en dirons rien de plus dans ce chapitre.

Pour l'illustration des concepts, une couleur nouvelle est introduite dans le réseau pour mettre en évidence un élément particulier du réseau. La deuxième colonne indique pour chaque concept le (les) élément(s) que la couleur va permettre de mettre en évidence.

La troisième colonne indique pour l'illustration de chaque concept les éléments qui seront animés.

Enfin, la dernière colonne indique les informations supplémentaires qui seront ajoutées en dehors du monde en 3D, pour offrir à l'apprenant une compréhension plus complète et plus profonde du concept choisi. Ces informations "textuelles" seront donc synchronisées avec ce qui se déroule dans la représentation en trois dimensions du réseau ATM. L'intitulé de la colonne est "*Contenu de l'applet*" car nous avons décidé que ces informations seraient affichées grâce à une applet Java insérée dans les pages WEB qui forment le module.

Nous allons maintenant passer en revue les simulations propres à chaque concept.

III.1 Le concept de couche

Le concept de couche est un concept important dans le domaine des télécommunications. Il permet notamment de mettre en évidence les différentes fonctionnalités qui doivent être couvertes par un réseau pour permettre le transfert de données d'une machine à une autre. Malgré tout, ce concept reste un concept assez théorique et fort distant de l'organisation pratique des réseaux et des ordinateurs. Pour ce concept, il nous semblait important de bien montrer le fait que toute couche est utilisatrice des services offerts par la couche supérieure et fournisseur de services à la couche inférieure. Il nous fallait donc montrer qu'il existe un ordre dans la façon dont les couches travaillent. De plus, chaque couche remplit une ou plusieurs fonctions bien particulières dans le réseau, ce qui permet d'acheminer des informations d'un point à l'autre du réseau. Il fallait donc préciser également quel rôle jouait chaque couche.

Nous allons expliquer brièvement le fonctionnement de cette animation sur un schéma simple. Pour représenter la conversation entre deux machines nous supposons que les cellules ne franchissent qu'un seul commutateur. Ce schéma simplifié est celui de la Figure 29.

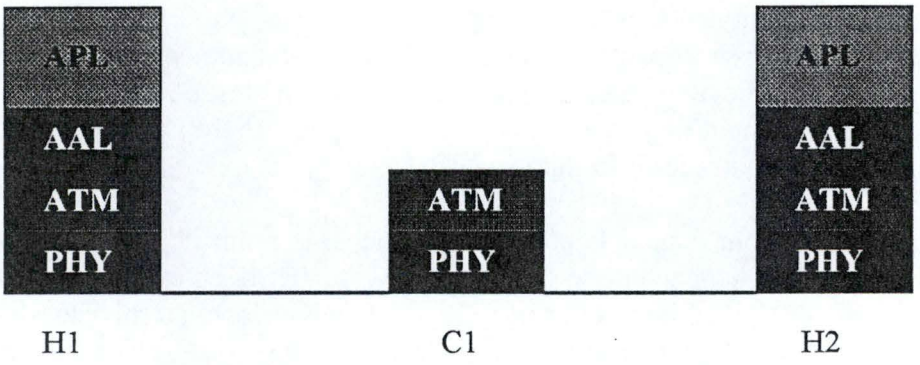


Figure 29 : Un schéma simplifié du réseau ATM pour le concept de couches

Pour illustrer ce concept une seule cellule est nécessaire. Le but de cette animation est de montrer comment les couches théoriques qui composent les différentes machines d'un réseau travaillent sur les cellules. L'animation mettra donc en évidence la couche qui travaille en utilisant un phénomène de changement de couleur de la couche dans la scène VRML.

Nous allons maintenant détailler dans l'ordre ce qui doit se passer dans l'animation de notre petit schéma si on suppose que la machine H1 converse avec la machine H2. Nous désignerons une couche de la façon suivante : H1.AAL représente la couche AAL de la machine H1.

Voici comment devra se dérouler l'animation de façon chronologique :

1. Changement de couleur de H1.APL pendant un certain laps de temps;
2. Retour de H1.APL à sa couleur d'origine et changement de couleur de la couche H1.AAL;
3. Retour de H1.AAL à sa couleur d'origine et changement de couleur de la couche H1.ATM;
4. Retour de H1.ATM à sa couleur d'origine et changement de couleur de la couche H1.PHY;
5. Retour de H1.PHY à sa couleur d'origine. Envoi de la cellule sur la liaison L1;

6. Entrée de la cellule dans le commutateur C1 et changement de couleur de la couche C1.PHY;
7. Retour de la couche C1.PHY à sa couleur d'origine et changement de couleur de la couche C1.ATM;
8. Retour de la couche C1.ATM à sa couleur d'origine et changement de couleur de la couche C1.PHY;
9. Retour de C1.PHY à sa couleur d'origine. Envoi de la cellule sur la liaison L2;
10. Entrée de la cellule dans la machine H2 et changement de couleur de la couche H2.PHY;
11. Retour de H2.PHY à sa couleur d'origine et changement de couleur de la couche H2.ATM;
12. Retour de H2.ATM à sa couleur d'origine et changement de couleur de la couche H2.AAL;
13. Retour de H2.AAL à sa couleur d'origine et changement de couleur de la couche H2.APL;
14. Retour de H2.APL à sa couleur d'origine;
15. Fin de l'animation.

Lors de cette animation il faut tenir compte qu'un commentaire sera affiché simultanément dans l'applet Java qui se trouve sous la scène VRML. Ce commentaire explique brièvement à l'utilisateur de l'application, quelles sont les fonctionnalités de la couche qui change de couleur. Le changement de couleur ne doit dès lors pas être trop rapide pour que l'utilisateur aie le temps de lire toutes les informations affichées.

Il est évident que, pour étudier le concept de couche, le point de vue retenu soit le point de vue en hauteur. Le changement de couleur permet de mettre en évidence la couche traversée (ou la couche qui travaille). L'animation se tiendra donc au niveau des couches qui changeront de couleur lorsqu'elles seront visitées ainsi qu'au niveau des informations affichées dans l'applet Java.

III.2 Le concept d'unité de donnée

Les cellules (unités de données d'un réseau ATM) sont un élément capital dans le réseau. Elles sont d'une taille fixe de 53 octets et sont composées de deux grandes parties : l'entête et la charge utile. Les champs de l'entête qui contiennent le VCI et le VPI sont modifiées lorsque la cellule passe dans un commutateur. L'illustration de ce concept est le plus simple à concevoir. Le but est en effet de rappeler la signification des champs contenus dans l'entête de la cellule et de montrer les changements qui se produisent dans cet entête lorsque la cellule traverse un commutateur.

Comme le simulateur ne présente que des connexions permanentes les différents changements qui vont exister dans l'entête de la cellule sont déterminés bien avant que la cellule ne soit créée. Il existe donc un certain déterminisme qui devra être mis en évidence dans l'applet. Le point de vue qui nous semble judicieux à utiliser est le point de vue aérien. La couleur permet de mettre en évidence une connexion VCC existant entre deux machines terminales. Le scénario sera basé sur l'animation d'une seule cellule. Les différents champs de la cellule seront représentés dans l'applet, ce qui permettra à l'apprenant de remarquer plus facilement les changements qui y ont lieu.

III.3 Le concept de commutation

En illustrant le concept de commutation, nous voulons faire apparaître, entre autres, la transparence de la partie données de la cellule lors de son voyage à travers le réseau. En effet, lorsque les cellules ATM arrivent dans un commutateur, ce dernier n'effectue aucun contrôle sur les données véhiculées. Le seul contrôle qui est effectué se fait sur l'entête des cellules afin de permettre leur routage. Pour faire le routage, les commutateurs utilisent des tables de commutation. Dans le cadre de notre simulateur, ces tables sont fixées puisque nous ne présentons que des connexions permanentes. Nous voulons mettre en évidence la consultation de ces tables et montrer qu'il existe un lien entre l'information contenue dans la cellule analysée par le commutateur et la ligne de sa table qu'il consulte. Le point de vue aérien nous semble suffisant pour que l'apprenant puisse comprendre le concept de commutation. Ce point de vue est d'autant plus important que lors du développement du simulateur nous avons ajouté sur le haut des machines terminales des lettres et sur le haut des commutateurs des chiffres permettant d'identifier aisément les différentes machines. La couleur va permettre de mettre en évidence deux connexions VCC étant donné que l'animation portera sur deux cellules provenant de deux machines différentes et destinées à deux machines différentes mais passant par deux commutateurs identiques au moins. L'applet Java nous permettra d'afficher le contenu des tables de commutation et de mettre en évidence la ligne consultée par ce commutateur lors de l'analyse d'une cellule.

IV. Les maquettes de l'interface

Dans cette partie, nous montrons au lecteur à quoi ressemble visuellement l'application que nous avons conçue. Dans une première section nous présenterons l'interface générale de l'application et dans une seconde partie nous détaillerons les différents contenus de l'applet qui devront illustrer les trois grands concepts du simulateur.

IV.1 Les maquettes générales

Avant que la collaboration internationale ne commence dans le projet P2-Verso, les maquettes de l'interface avaient déjà été pensées et critiquées par certains membres de l'équipe de l'Ecole Centrale de Lyon. Dans cette section, nous présentons au lecteur les maquettes qui ont été proposées aux autres membres du projet P2-Verso. Nous présentons ces maquettes déjà travaillées car ce sont sur ces dernières que s'est réellement basé le travail de conception de l'application. Les figures présentées dans ce chapitre ne sont que des maquettes, le lecteur pourra observer une saisie des écrans dans le chapitre consacré à l'analyse technique du simulateur.

Comme nous l'avons déjà précisé plus haut, l'application s'affichera dans une page HTML divisée en deux grandes parties. La maquette de cette page est illustrée à la Figure 30.

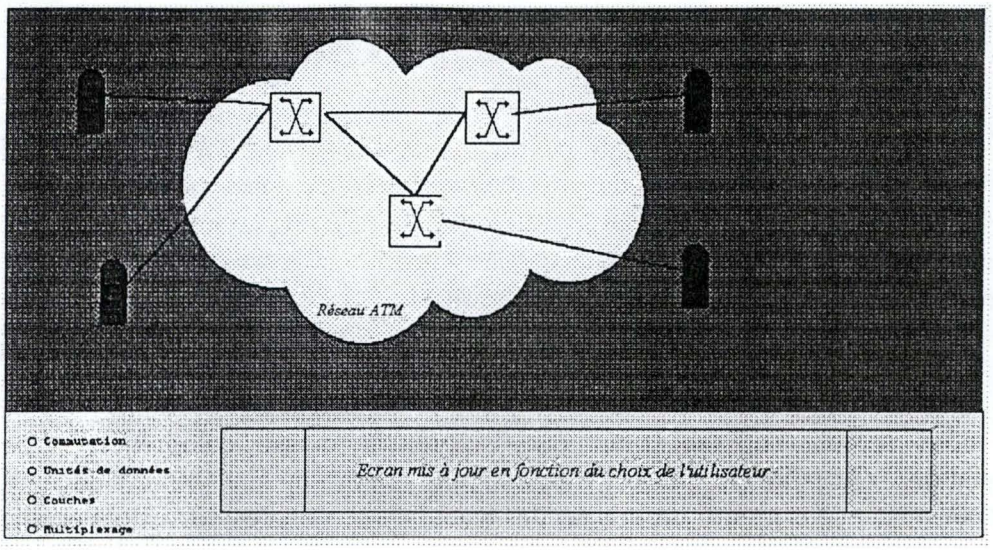


Figure 30 : La maquette générale de l'application

Grâce à la Figure 30, on constate facilement que l'application est bien une page HTML divisée en deux parties de longueur fixe. La largeur des deux parties est égale à la largeur de la page. La hauteur de la scène VRML représente 75% de la hauteur de la page, alors que l'applet Java occupe les 25% restants. C'est dans la zone supérieure qu'apparaîtra la représentation graphique en trois dimensions du réseau ATM. Lors du lancement de l'application, les objets statiques de la scène VRML se dessinent sur un sol plan. Les objets statiques sont :

- Les quatre machines terminales;
- Les trois commutateurs;
- Les sept liaisons physiques entre toutes ces machines.

Leur placement dans la scène VRML est à peu près identique à celui que l'on peut voir dans la partie supérieure de la Figure 30.

La partie inférieure de la page HTML représente l'applet Java. Celle-ci est illustrée en détail par la Figure 31. C'est à partir de l'applet Java que seront réalisés tous les contrôles dans le monde (et donc dans la scène VRML). Seule la navigation interne au monde en 3D relève du tableau de bord de la scène VRML.

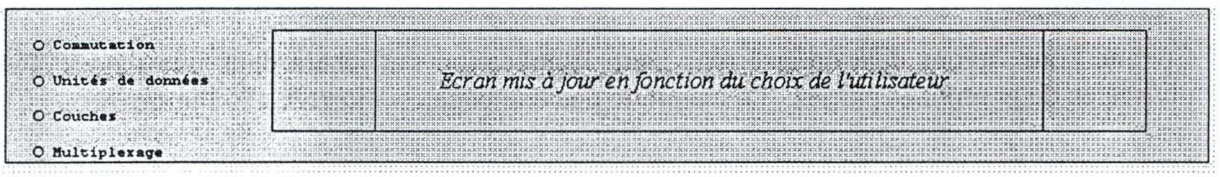


Figure 31 : La maquette de l'applet Java

Suite aux premières critiques qui ont été faites par les membres du projet P2-Verso, il était important que l'utilisateur puisse agir sur l'animation qu'il observait. De ce fait, nous avons introduit dans cette applet la métaphore du magnétoscope. A droite de la zone d'affichage des informations utiles à l'illustration des différents concepts, nous avons placé un bouton intitulé

"Démarrage", un bouton intitulé "Pause" et un bouton intitulé "Stop". Ces derniers ont des actions appropriées en fonction de l'état de l'animation :

- Le bouton "Démarrage" sert à lancer l'animation une fois que l'utilisateur a choisi le concept qu'il veut étudier. Il permet également, avec la bouton "Pause", de relancer l'animation si l'utilisateur a mis celle-ci en pause;
- Le bouton "Pause" met l'animation dans un état de pause si elle était en cours et relance l'animation si elle était dans un état de pause;
- Le bouton "Stop" arrête toute l'animation et remet la scène VRML dans son état initial.

Ces boutons apparaîtront dans les maquettes des applets que nous allons détailler.

IV.2 Les maquettes des applets

Dans cette section, nous allons décrire les trois formes différentes que prendra l'applet Java en fonction du concept que l'utilisateur de l'application aura décidé d'animer.

IV.2.1 Le concept de commutation

Lors de l'illustration du concept de commutation, l'utilisateur de l'application peut voir dans l'applet une représentation de la table de chaque commutateur du réseau ATM. Etant donné qu'il y a trois commutateurs dans le scène VRML, il y aura trois tables de commutation représentées dans l'applet. Cette première forme de l'applet Java est illustrée à la Figure 32.

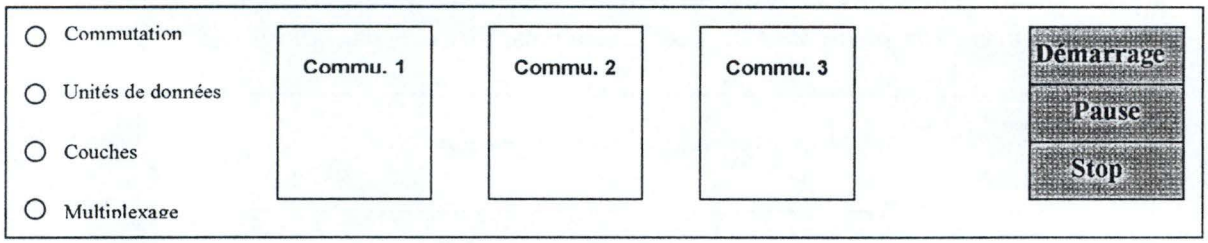


Figure 32 : La forme de l'applet pour le concept de commutation

Les tables de commutation contiennent toutes une ou deux lignes. Chaque ligne est composée de deux parties. Une première qui reprend le numéro de VPI et de VCI d'entrée d'une cellule et une deuxième partie qui contient le numéro de VPI et VCI que la cellule contiendra dans ses champs à la sortie du commutateur.

La Figure 32 ne reprend pas toutes les informations qui se trouveront dans l'applet lors de l'animation. Le rond en face du terme "commutation" (qui représente un bouton radio) sera coché. De plus certaines informations supplémentaires permettront à l'utilisateur de prévoir quelle ligne des commutateurs sera consultée en fonction de la machine source des cellules circulant dans le réseau.

IV.2.2 Le concept d'unités de données

La forme que prendra l'applet Java pour l'illustration du concept d'unités de données est représentée à la Figure 33

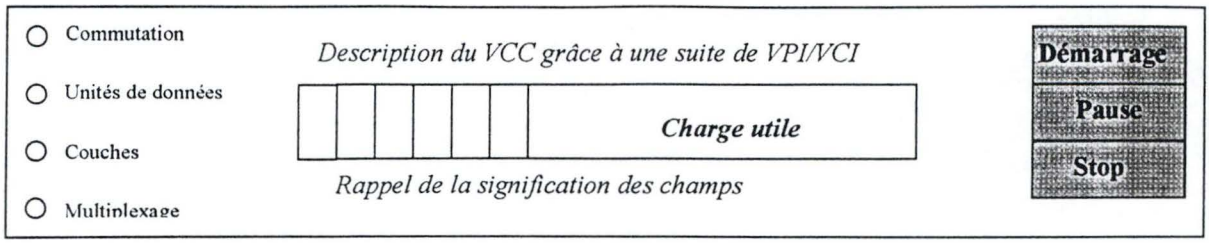


Figure 33 : La forme de l'applet Java pour le concept d'unités de données

La description de la liaison VCC dans laquelle circule la cellule rappelle à l'utilisateur de l'application le fait que le simulateur ne lui présente que le mode de connexion permanent. En fonction de ce chemin fixe, l'utilisateur pourra anticiper les nombres qui devront apparaître dans les champs VPI et VCI de la cellule ATM. La cellule ne change jamais, seuls ces champs VPI et VCI sont modifiés. La signification des champs est aussi un élément constant. Pour des raisons de cohérence, les membres du projet P2 Verso ont préférer que la signification des champs soit donnée en anglais car leurs abréviations viennent des noms anglais des champs. Comme dans la forme de l'applet du concept de commutation, le cercle se trouvant face au terme "unité de données" doit être noirci, traduisant ainsi le fait que le bouton radio qu'il représente a été sélectionné par l'utilisateur.

IV.2.3 Le concept de couches

La forme de l'applet Java pour le concept de couches est la seule forme pour laquelle aucune partie des informations affichées n'est statique. Il existe un changement continu en rapport avec l'animation de la scène VRML. Cette forme est illustrée par la Figure 34 .

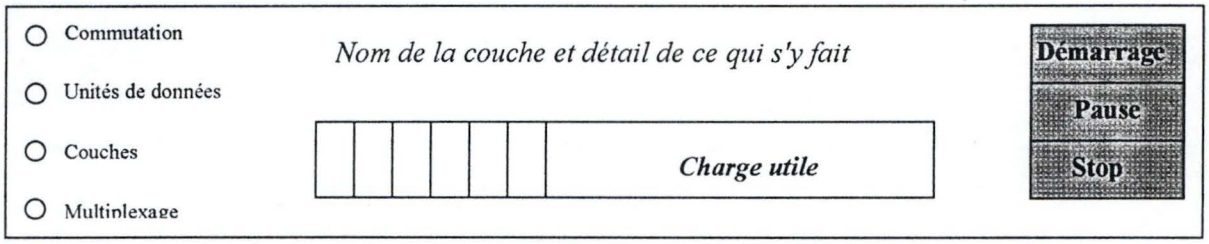


Figure 34 : La forme de l'applet pour le concept de couches

Dans la scène VRML, la couche qui traite la cellule est mise en évidence par un changement de couleur. Lorsque une couche change de couleur, son nom apparaît dans l'applet Java ainsi qu'une brève description de ce qu'elle est en train de faire. Pour illustrer ce concept, une seule cellule circule dans le réseau ATM. Aussi, l'applet pourra mettre en évidence la façon dont cette dernière est construite et modifiée au cours de son voyage à travers le réseau.

Pour permettre une meilleure rétention de ce que fait chaque couche, les champs de la cellule apparaîtront dans la même couleur que celle de la couche qui les crée. Le cercle se trouvant devant le terme "couches" doit être noirci illustrant ainsi le choix de l'utilisateur.

V. Conclusion

Une fois que le cahier des charges a été rédigé, nous sommes passés à la conception de l'application. La partie suivante présente au lecteur l'analyse technique de cette application.

Partie 3 - Analyse technique

Chapitre VII : Architecture physique

I. Introduction

Dans le projet *P2-Verso*, notre travail consiste à implémenter, par le biais des simulations, certains concepts de la technologie ATM. Ce chapitre introduit l'architecture physique de l'application. La section II décrit la découpe de l'application en modules. A ce niveau, nous faisons une découpe logique. Toutefois, l'architecture physique de l'application suit la même découpe que l'architecture logique. A partir des schémas, nous présentons les différentes classes d'objets logiques et les relations existant entre elles. La description de différentes couches de l'architecture physique sera abordée dans les deux chapitres suivants.

Dans l'ensemble, nous avons choisi de travailler selon une approche objet afin de bénéficier des avantages tels que la réduction de la distance sémantique entre le réel et le système informatique, une plus grande cohérence entre l'aspect statique et l'aspect dynamique, une meilleure modularité et une réutilisation des composants du système. Les objets, en plus de leur puissance d'abstraction, de généralisation et d'interaction, procurent une facilité de modification et d'extension. Grâce à cette approche, nous sommes arrivés à réaliser les deux objectifs majeurs que nous nous sommes fixés : représenter directement les entités du monde réel (des réseaux) et permettre une modularité, une réutilisation et une extension du code.

II. Découpe en modules

L'architecture logique de l'application comporte essentiellement deux couches : une couche de présentation et une couche de traitement. La couche de présentation contient un seul module, le module *Présentation*, composé des classes d'objets "graphiques" susceptibles de représenter graphiquement les objets réels du domaine (les télécommunications).

Afin de permettre une réutilisation et une extension du code, nous avons choisi d'appliquer une approche modulaire dans la réalisation de la couche de traitement : elle comprend un module général, le module *Réseau*, et un module spécialisé, le module *Atm*.

Le module *Réseau* contient des classes d'objets généraux du domaine. Il est évident qu'à côté de ce module, il faut des modules spécialisés implémentant les différents protocoles du domaine. Pour le moment, seul le module implémentant quelques fonctionnalités de la technologie ATM est réalisé.

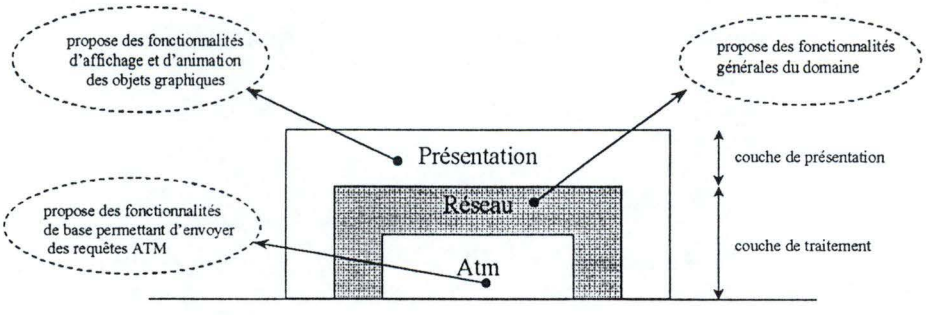


Figure 35 : *Composants de l'application*

La Figure 35 illustre une découpe en niveaux de fonctionnalités. Pour l'application, le module *Atm* constitue le noyau du système. Le module *Présentation* s'appuie sur les modules *Réseau* et *Atm* pour fournir ses services.

Il est important de signaler que les trois modules ne contiennent que des classes d'objets. Au-dessus de ces modules, il faut un programme client tel qu'une *applet Java*. Ainsi, comme il en découle du chapitre traitant de l'analyse fonctionnelle, notre application est un agrégat de simulateurs sous forme d'applets, chacun illustrant un ou plusieurs concepts particuliers de la technologie ATM.

Dans ce qui suit, nous faisons une brève description des différentes classes d'objets de l'application. Pour chaque module, un schéma illustre les différentes classes d'objets et les relations entre elles. Comme souligné plus haut, cette description est orientée logique. Les éléments techniques ne sont pas repris à ce niveau.

II.1. Module *Réseau*

Pour identifier les différentes classes d'objets, nous sommes partis du domaine d'application à couvrir. Chaque classe d'objets représente un concept du monde des réseaux. Toutefois, nous n'avons pas représenté tous les concepts du domaine. Ce choix est influencé par le cahier de charges mis à notre disposition et nous sommes conscients que notre choix n'est pas le seul possible, ni nécessairement le meilleur.

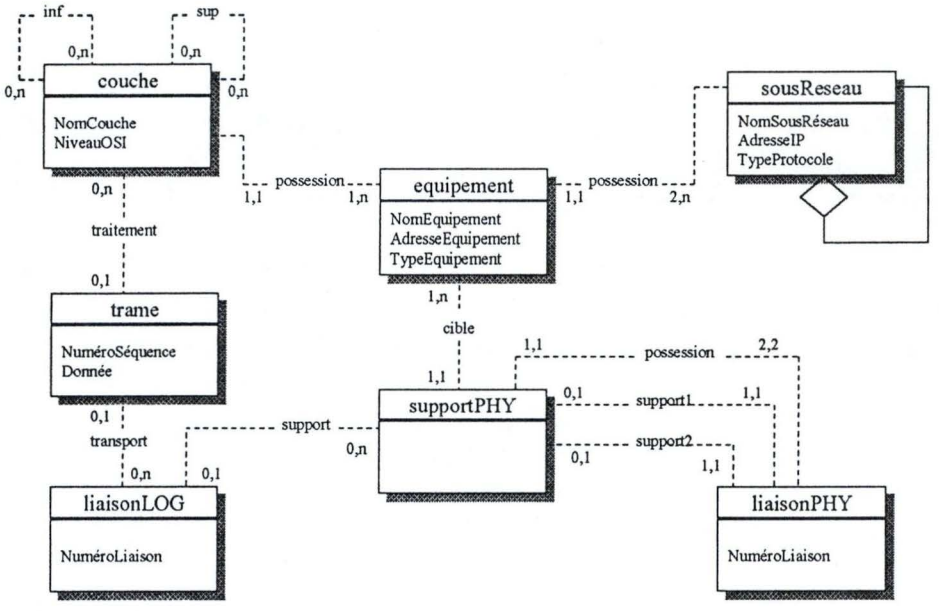


Figure 36 : *Graphe des relations entre classes du module Réseau*

La Figure 36 illustre les classes d'objets du module *Réseau* et les relations qui existent entre elles.

Classe *sousReseau*

Un sous-réseau est caractérisé par un nom, une adresse IP et un type de protocole. Le cas de réseau des réseaux illustré sur la Figure 36 n'est pas traité par l'application. Pour des raisons de simplification, un sous-réseau comprend au moins deux équipements.

Classe *equipement*
Un équipement est vu comme une agrégation d'entités couches. Il est caractérisé par un nom, une adresse et un type. Un équipement est soit terminal soit non terminal. Pour des raisons de simplification, un équipement n'appartient qu'à un seul sous-réseau. Il faut aussi souligner que pour des raisons identiques, nous considérons qu'un sous-réseau comprend non seulement des équipements non terminaux mais aussi des équipements terminaux.

Les équipements, dans un sous-réseau, sont interconnectés par des liaisons physiques bidirectionnelles point à point et un équipement est extrémité d'au moins une liaison physique.

Classe *liaisonPHY*
Une liaison physique est caractérisée par un numéro. Etant bidirectionnelle dans notre application, une liaison physique est une paire de supports physiques, chaque support physique étant unidirectionnel et de sens contraire.

Classe *supportPHY*
Un support physique est un media de transmission entre un équipement source et un équipement cible. Il n'appartient qu'à une seule liaison physique et ne "cible" qu'un seul équipement.

Sur une liaison physique, entre deux équipements, plusieurs communications peuvent avoir lieu. Une communication est matérialisée par un circuit virtuel. Dans la suite de ce chapitre, nous employons plutôt l'expression liaison logique.

Classe *liaisonLOG*
Une liaison logique, caractérisée par un numéro, matérialise une communication unidirectionnelle entre deux équipements adjacents ou non. Une liaison logique entre deux équipements adjacents est portée par un support physique de même sens. Pour des raisons de simplification, une liaison logique entre des équipements non adjacents n'est associée à aucun support physique.

Classe *trame*
Dans le module *Réseau*, nous utilisons le terme *trame* comme un terme générique désignant toute unité d'information échangée par les différentes entités du système. Une trame est caractérisée par un numéro de séquence et par l'information véhiculée. Une trame est soit en cours de traitement dans une entité couche soit en cours de transport sur une liaison logique.

Classe *couche*
Dans le modèle OSI, *Open System Interconnection*, un équipement est une agrégation d'entités couches, chacune d'elles étant bâtie sur la précédente. Une entité couche est caractérisée par un nom et un numéro de niveau OSI. Elle ne peut appartenir qu'à un et un seul équipement.

II.2. Module *Atm*

La Figure 37 illustre les différentes classes d'objets du module. Les classes d'objets générées sont fonction du cahier de charges.

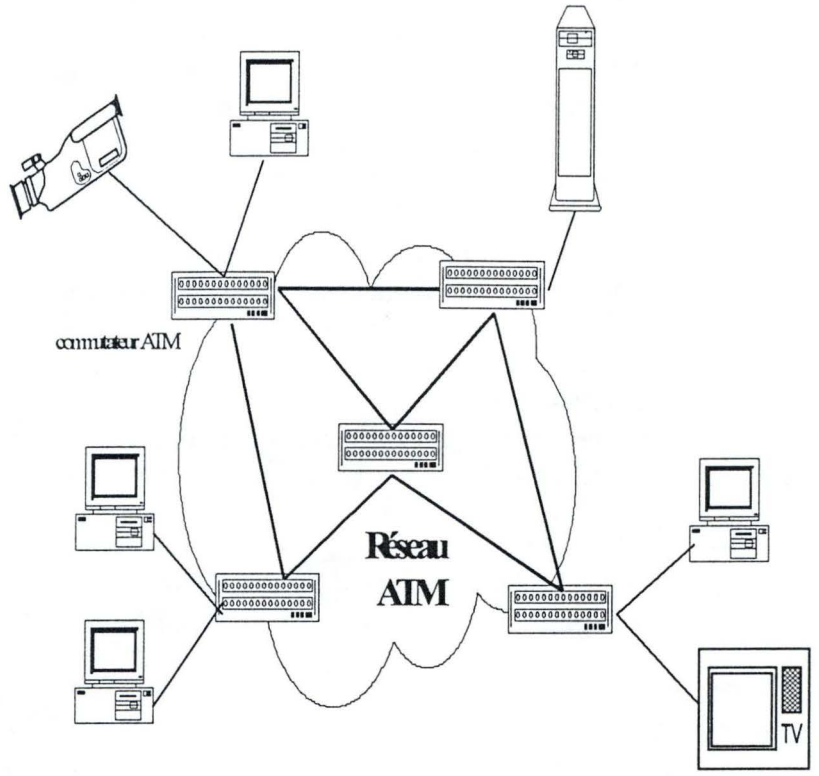


Figure 37 : *Graphe des relations entre les classes du module Atm*

La Figure 38 décrit les relations d'héritage entre les classes du module *Réseau* et celles du présent module.

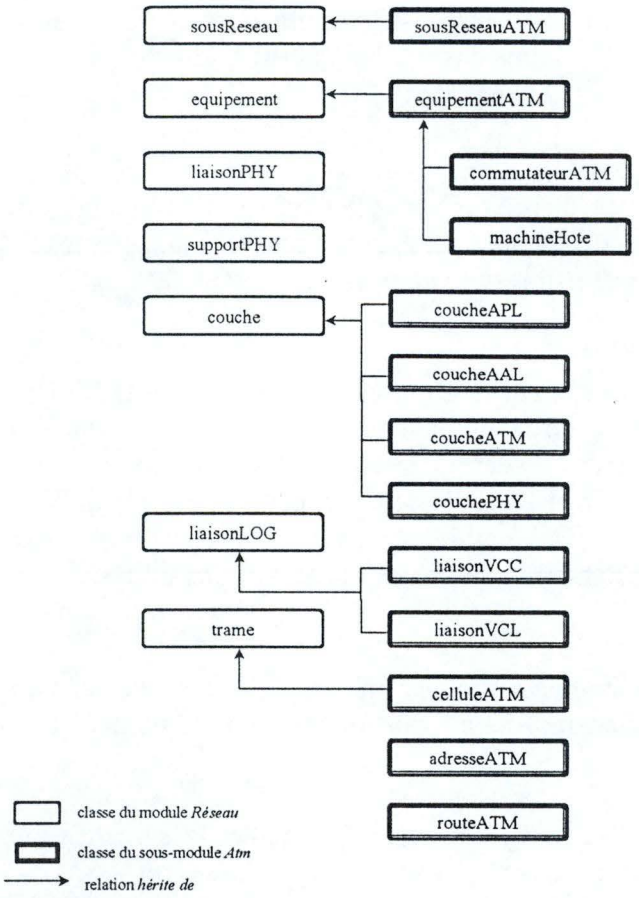


Figure 38 : *Graphes des relations d'héritage entre classes des modules Réseau et Atm*

Classe *sousReseauATM*
La classe *sousReseauATM* hérite de la classe *sousReseau* du module *Réseau*. Un sous-réseau ATM peut supporter plusieurs connexions VCC (*Virtual Channel Connection*).

Classe *liaisonVCC*
La classe *liaisonVCC* hérite de la classe *liaisonLOG* du module *Réseau* et représente le concept de connexion VCC. Etant une liaison logique entre deux équipements non adjacents, une connexion VCC n'est associée à aucun support physique et n'appartient qu'à un seul sous-réseau ATM. Elle est caractérisée, en plus de l'attribut hérité, par un champ indiquant le nombre de cellules déjà reçues par la machine hôte de destination.

Dans l'application, par simplicité, nous considérons qu'une connexion VCC est constituée d'au moins deux liens VCL (*Virtual Channel Link*). Le premier et le dernier liens VCL d'une connexion VCC sont toujours connus.

Classe *liaisonVCL*
La classe *liaisonVCL* hérite de la classe *liaisonLOG* du module *Réseau* et représente le concept de lien VCL. En plus de l'attribut hérité, elle contient un champ contenant le numéro VPI (*Virtual Path Identifier*) la contenant. Un lien VCL, dans notre application, relie deux équipements adjacents. Il est soit le premier soit le dernier soit un quelconque lien VCL d'une connexion VCC et n'appartient qu'à une et une seule connexion VCC.

Classe *celluleATM*
Une cellule est générée par une entité couche ATM d'un équipement terminal. La classe *celluleATM* hérite de la classe *trame* du module *Réseau*. Une cellule n'appartient qu'à une seule connexion VCC et, en plus des attributs hérités, elle possède deux autres champs : CLP (*Cell Loss Priority*) et PT (*Payload Type*).

Classe *equipementATM*
La classe *equipementATM* hérite de la classe *equipement* du module *Réseau* et regroupe deux types d'équipements : les commutateurs ATM et les machines hôtes (équipements terminaux). Les brasseurs ne sont pas traités dans cette application.

Classe *commutateurATM*
La classe *commutateurATM* hérite de la classe *equipementATM*. Un commutateur ATM est l'équipement responsable de la commutation de cellules. Lorsqu'une cellule se présente à l'entrée d'un commutateur, ce dernier, sur base de l'en-tête de la cellule et de sa table de commutation, l'envoie sur la ligne de sortie adéquate. Dans l'application, les éléments de la table de commutation sont représentés par la classe *routeATM*. Il faut noter qu'un commutateur peut être extrémité de plusieurs liaisons physiques.

Classe *routeATM*
Une route ATM est un couple formé par un lien VCL d'entrée dans un commutateur et le lien VCL de sortie correspondant. Elle n'appartient qu'à une seule table de commutation.

Classe *adresseATM*
Pour rappel, un plan d'adressage, dans un protocole de signalisation, permet d'identifier la source et la destination d'une connexion. La classe *adresseATM* représente la notion d'adressage. Une instance de cette classe est un couple formé par l'adresse d'une machine hôte et la liaison physique à emprunter pour l'atteindre. Dans l'application, les instances de la classe *adresseATM* sont regroupées dans des tables d'adresses.

Classe *machineHote*
La classe *machineHote* hérite de la classe *equipementATM*. Une machine hôte est extrémité d'une seule liaison physique.

Classe *coucheATM*
La couche ATM traite du transport d'une cellule de bout en bout. La classe *coucheATM* est une spécialisation de la classe *couche* du module *Réseau*.

Classe *couchePHY*
Dans un sous-réseau ATM, l'entité couche physique s'occupe de l'adaptation d'ATM sur un support physique. La classe *couchePHY* est une spécialisation de la classe *couche* du module *Réseau*.

Classe *coucheAAL*
Le rôle principal de l'entité couche AAL est d'assurer l'interface entre le mode de transmission ATM et les applications. L'entité couche AAL fournit ses services aux applications en utilisant les services qui lui sont offerts par l'entité couche ATM. La classe *coucheAAL* est une spécialisation de la classe *couche* du module *Réseau*.

Classe *coucheAPL*

Une entité couche APL représente toutes les applications susceptibles de recourir aux services de l'entité couche AAL. Elle est présente dans toute machine hôte. La classe *coucheAPL* est une spécialisation de la classe *couche* du module *Réseau*.

II.3. Module *Présentation*

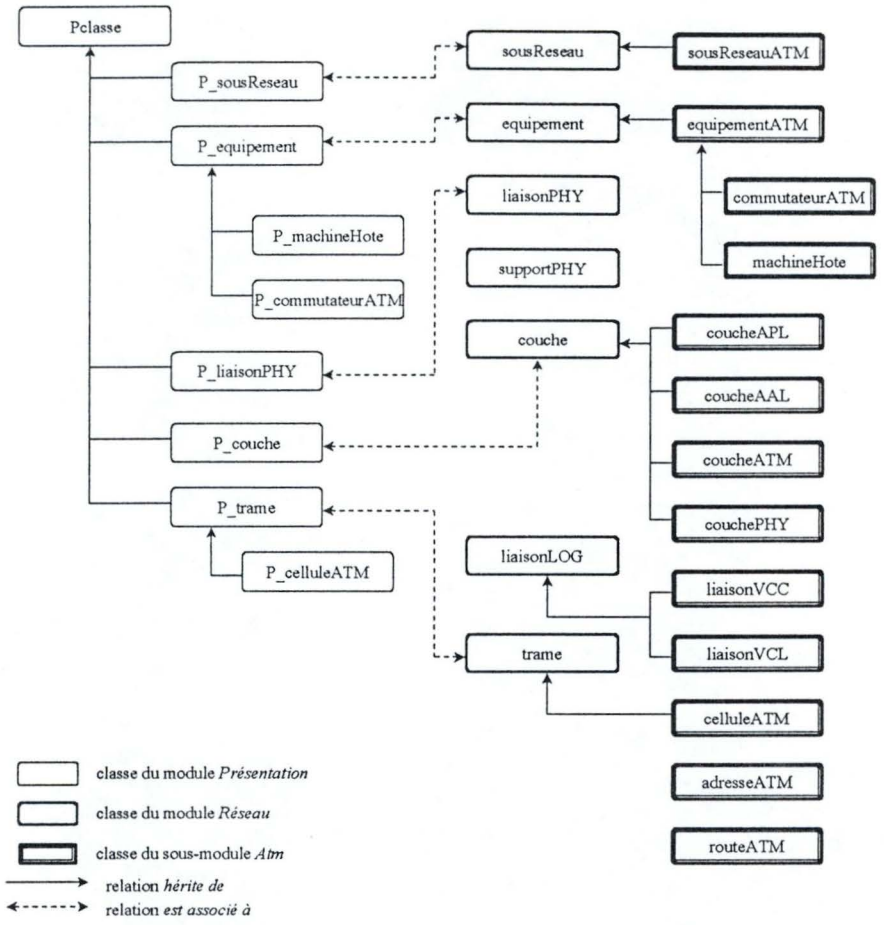


Figure 39 : *Graphe général des Relations entre classes*

Le module *Présentation* est le seul module de la couche de présentation de l'application. Limités par le temps, nous n'avons pas pu appliquer la modularité dans cette couche. Ce module est essentiellement orienté ATM.

La Figure 39 présente le graphe général des relations entre les classes d'objets de trois modules de l'application. Remarquons que toute classe d'objets de la couche de traitement n'est pas associée à une classe d'objets graphiques. Ce choix est fonction du cahier de charges.

III. Conclusion

En adoptons la découpe décrite dans ce chapitre, nous voulions concevoir une couche de traitement qui soit indépendante de la couche de présentation. Tout ce qui concerne la représentation graphique des objets est repoussé vers la couche de présentation. Comme nous le verrons dans le prochain chapitre, la couche de traitement offre à cette dernière des méthodes permettant d'associer un objet abstrait (une instance d'une classe d'objets) à une représentation graphique.

Chapitre VIII : Couche de traitement

I. Introduction

Ce chapitre traite essentiellement de la couche de traitement de notre application. Sur base de la découpe logique présentée au chapitre VII, nous décrivons, d'une part, les choix d'implémentation faits et d'autre part, les interfaces des classes d'objets implémentées. Les codes sources associés sont mis dans la partie annexe de ce mémoire.

II. Choix d'implémentation

Dans cette partie du mémoire, nous explicitons les différents choix effectués lors de l'implémentation de la couche de traitement. Nous rappelons que la couche de traitement de l'application est implémentée en Java. Pour la suite de la section, nous supposons connu le chapitre consacré à ce langage. Sauf pour lever une ambiguïté, dans la suite du chapitre, nous employons le terme *commutateur* pour désigner un commutateur ATM et le terme *sous-réseau* pour un sous-réseau ATM.

II.1. Classes d'objets

Dans l'approche objet, un objet est une entité identifiable à laquelle sont attachées des opérations (méthodes) et un ensemble d'attributs (état de l'objet). Dans les classes d'objets que nous avons définis, nous n'implémentons pas de contrôle pour les attributs identifiants. L'unicité de leurs valeurs doit être vérifiée au niveau du programme client. Les objets, dans l'application sont identifiés sur base de leurs références.

Les relations entre différentes classes d'objets sont implémentées par des variables d'instance. Par exemple, la relation

(classe *equipement*, classe *couche*)

se traduit par :

- un attribut de type ***Vector*** dans la classe *equipement* pour contenir la liste de références des instances de la classe *couche* composant une instance de la classe
- et un attribut de type *equipement* dans la classe *couche* pour référencer l'objet de la classe *equipement* la contenant.

Par simplicité, il n'y a pas d'attributs optionnels. Nous préférons avoir des attributs avec éventuellement une valeur nulle.

En général, pour chaque attribut d'une classe, nous avons prévu une paire d'opérations, pour lire et écrire la valeur de l'attribut. Un attribut n'est accessible que via des méthodes appropriées.

Pour des besoins d'implémentation, des nouvelles classes d'objets sont ajoutées. Elles sont introduites progressivement dans la suite de cette section.

II.2. Mode de connexion, signalisation et algorithme de routage

Normalement, le cahier de charges prévoit un mode de connexion permanente, PVC (Permanent Virtual Connection), car moins complexe à implémenter. Pour rappel, ce mode de connexion nécessite une programmation manuelle de tous les commutateurs afin de désigner les routes possibles entre les différents équipements. Toutefois, pour ne pas rendre figés les modules *Réseau* et *Atm*, nous avons choisi de repousser cette configuration au niveau du programme client.

Pour atteindre cet objectif, nous implémentons un mode de connexion commutée mais transparent pour le programme client. Comme pour un vrai protocole de signalisation, nous implémentons un plan d'adressage : chaque commutateur possède une table d'adresses. Dans l'application, la configuration manuelle concerne le plan d'adressage. La création des tables d'adresses doit être faite par le programme client. Comme signalé précédemment, chaque ligne de cette table précise pour un équipement donné, la liaison physique à emprunter pour l'atteindre. Notons que cette configuration manuelle nécessite au préalable l'exécution des opérations suivantes :

1. la création des machines hôtes;
2. la création des commutateurs;
3. la création des liaisons physiques entre les différents équipements du sous-réseau.

Nous ne prévoyons aucun contrôle d'erreur. A l'initialisation d'une simulation, les tables d'adresses des commutateurs sont supposées correctes.

Lors de la création d'un sous-réseau, les tables de commutation des commutateurs sont vides. Ce n'est que lorsque le programme client crée une connexion VCC que les tables de commutation des commutateurs se trouvant sur la route reliant la machine hôte d'origine à la machine hôte de destination sont mises à jour. Les liens VCL d'une connexion VCC sont générés automatiquement et d'une manière transparente pour le programme client. En effet, sur base des informations contenues dans les différentes tables d'adresses, le sous-réseau est capable de constituer la route reliant deux machines hôtes. Toutefois, il est supposé que cette route existe toujours et qu'elle est unique. L'application travaille sur base d'un *algorithme de routage statique*.

A la fin d'un transfert, la connexion VCC est détruite et les entrées VCL associées sont supprimées des tables de commutation des commutateurs.

II.3. Table de commutation et table d'adresses

Tout commutateur possède une table de commutation et une table d'adresses (Figure 40). Chaque ligne d'une table de commutation est une instance de la classe *routeATM*. Cette table est gérée par la couche de traitement. Par contre, la table d'adresses, regroupant 0 ou plusieurs instances de la classe *adresseATM*, est gérée par le programme client.

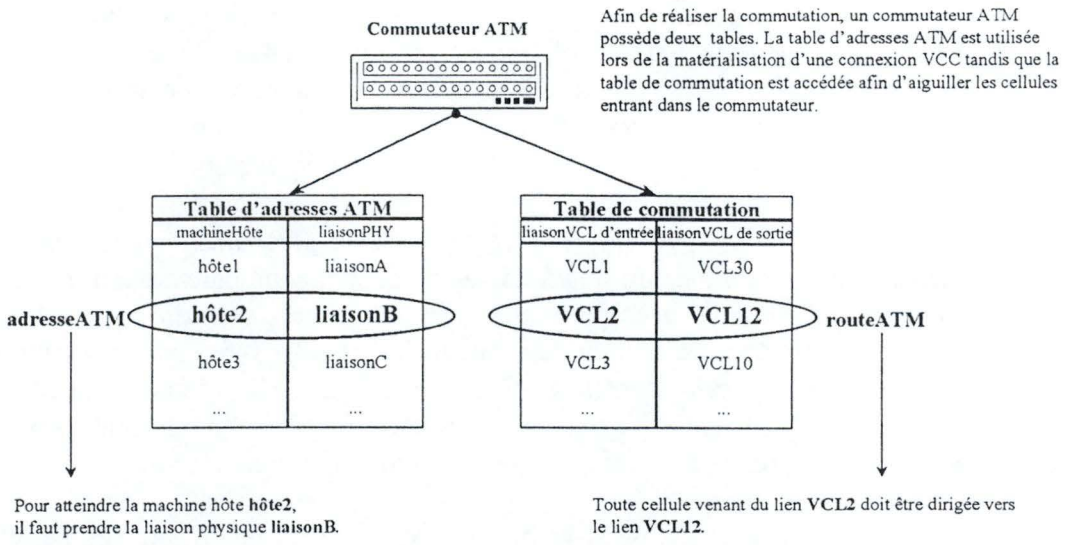


Figure 40 : *Table d'adresses et table de commutation*

Dans la philosophie objet, un objet est identifié par sa référence. Comme souligné plus haut, un lien VCL est porté par un seul support physique. Ainsi lorsque, dans un commutateur, on fait référence à une instance de type *liaisonVCL*, on a implicitement le port d'entrée ou de sortie associé (ce port est, en toute généralité, le point de contact entre le support physique et le commutateur). Un lien VCL est donc unique et associé à un seul port.

II.4. Cellule

Dans l'implémentation de la cellule, nous n'avons pas repris tous ses champs. Le champ *VCI* est remplacé par une référence vers le lien VCL lui-même (*RéfLiaisonVcl*). Dès lors, nous n'avons plus besoin du champ VPI, puisqu'il est un attribut du lien VCL. Le champ *Donnée*, dans l'application, contient la chaîne de caractère "hhhh". Ceci n'a aucune incidence sur le fonctionnement de l'application.

En outre, pour une trame en général (la cellule héritant de la classe *trame*), nous avons un champ contenant une référence vers l'objet la traitant à un instant donné. Pour rappel, une trame est soit en cours de transport sur une liaison logique soit en cours de traitement dans une entité couche. Cette manière de localiser une trame permet de tracer son parcours visuellement (au niveau d'une interface IHM).

II.5. Couches ATM

Dans l'application, une simulation est lancée à partir d'une entité couche AAL. Ne gérant pas la qualité de service, cette entité couche AAL n'est pas divisée en classes. Seuls les équipements terminaux possèdent une entité couche AAL. Par ailleurs, la classe *coucheAPL* est utilisée uniquement par le module *Présentation*. En dehors du constructeur, elle ne possède aucune autre méthode.

Tout équipement possède une entité couche ATM. Les fonctionnalités d'une entité couche ATM sont fonction du type d'équipement la contenant :

- l'entité couche ATM d'une instance de la classe *machineHote* reçoit une donnée : elle génère une cellule et sur base de la référence de la connexion VCC (passée en paramètre), elle enregistre la référence du premier lien VCL de la connexion VCC dans le champ *RéfLiaisonVcl* de la cellule. Comme signalé plus loin, une connexion VCC contient toujours les références de ses premier et dernier liens VCL. Après ces opérations, la cellule est transmise à l'entité couche physique;

- l'entité couche ATM d'une instance de la classe *commutateurATM* reçoit une cellule : sur base de la référence du lien VCL d'entrée contenue dans la cellule (*RéfLiaisonVcl*), elle consulte sa table de commutation afin d'aiguiller la cellule vers la bonne sortie. En effet, en appliquant une méthode spécialisée sur cette référence, elle reçoit la référence du lien VCL de sortie que doit prendre la cellule. Elle remplace la référence du lien VCL d'entrée par celle du lien VCL de sortie obtenu. Après toutes ces opérations, la cellule est transmise à l'entité couche physique du commutateur.

Lorsque l'entité couche ATM de la machine de destination reçoit une cellule, elle met à jour le champ *NbreCellulesReçues* de la connexion VCC concernée et transmet le contenu du champ *Donnée* de la cellule à l'entité couche AAL. Le champ *NbreCellulesReçues* nous permet de couper une connexion VCC dès que toutes les cellules qui devraient être transmises sont reçues par la machine hôte de destination. Il faut noter que, lors du lancement d'une communication entre deux entités couches AAL, le programme client précise la connexion VCC concernée et le nombre de cellules à transférer.

Nous n'avons implémenté aucune fonctionnalité pour la couche physique. Elle joue un rôle passif : elle transmet à l'entité couche ATM des cellules venant de l'extérieur et renvoie vers l'extérieur les cellules venant de l'entité couche ATM. En outre, sur un équipement, pour des raisons de simplification, il n'y a qu'une seule entité couche même si l'équipement est connecté à plusieurs autres.

II.6. Primitives de service

L'objet d'une entité couche est d'offrir des services à l'entité couche immédiatement supérieure. Un service est un ensemble de primitives qu'une entité couche fournit à l'entité couche immédiatement supérieur [TANEN97]. Lorsqu'une entité couche de niveau *n* souhaite dialoguer avec une entité paire, elle utilise un service fourni par une entité couche de niveau *n-1*. A son tour, l'entité couche de niveau *n-1* (si elle n'est pas une entité couche physique) s'appuie sur un ou plusieurs services de l'entité couche de niveau *n-2*, etc. D'une façon générale, un service fourni à l'entité couche *n* par une entité couche *n-1* comprend les primitives suivantes :

- *CONNECT.request* : demande de connexion;
- *CONNECT.indication* : signalisation de la demande de connexion à l'appelé;
- *CONNECT.response* : acceptation ou rejet de la demande de connexion par l'appelé;
- *CONNECT.confirm* : signalisation à l'appelant de la réponse de l'appelé;
- *DATA.request* : envoi de données par l'appelant;
- *DATA.indication* : signalisation à l'appelé de l'arrivée de données;
- *DISCONNECT.request* : demande de déconnexion par l'appelant;
- *DISCONNECT.indication* : signalisation de la demande de déconnexion à l'appelé.

Pour notre application, seules les primitives *DATA.request* et *DATA.indication* sont implémentées, respectivement en *DataREQ()* et *DataIND()*. Cependant, nous avons introduit une simplification (Figure 41). Normalement, les primitives de service sont offertes par l'entité couche inférieure. Comme signalé plus haut, l'objet de la primitive *DATA.indication* est la signalisation à l'appelé de l'arrivée de données. Il nous semble plus facile que l'entité couche inférieure signale à l'entité couche supérieure de cette arrivée et ce, en exécutant la primitive *DATA.indication*. Ainsi, pour notre application, cette primitive de service est fournie à une entité couche inférieure par une entité couche supérieure.

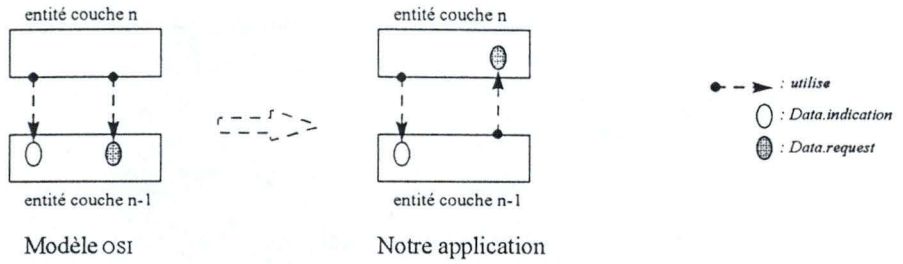


Figure 41 : *Transformation dans la mise en oeuvre des primitives de service*

Toujours pour des raisons de simplification, un mécanisme de primitives de service est ajouté dans les classes *equipement* et *liaisonLOG*.

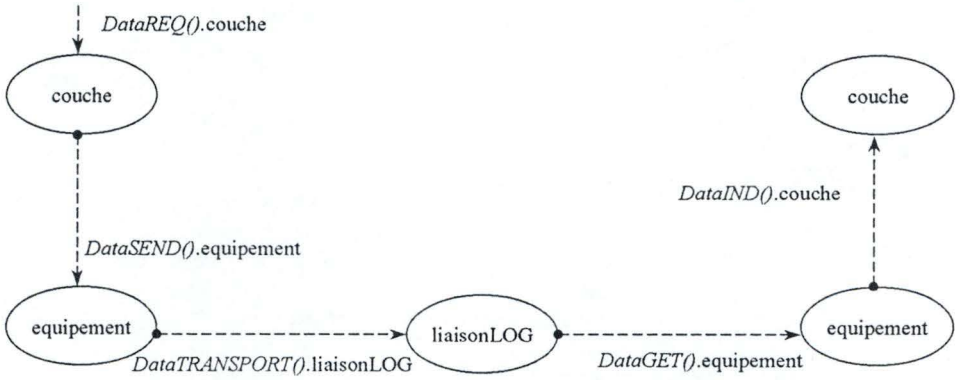


Figure 42 : *Scénario d'échange*

La Figure 42 illustre un scénario d'échange entre deux entités couches physiques. Dans l'application, lorsqu'une entité couche physique reçoit une demande d'envoi (via la méthode *DataREQ()*) sur la ligne d'une trame, elle s'appuie sur la méthode *DataSEND()* de la classe *equipement*. Cette méthode permet à un équipement de réceptionner une trame en vue d'un envoi sur une ligne donnée. Notons que cette même classe *equipement* fournit aux instances de la classe *liaisonLOG* une méthode permettant à un équipement de réceptionner une trame venant d'une ligne de communication : la méthode *DataGET()*.

La méthode *DataTRANSPORT()* de la classe *liaisonLOG* permet de transporter une trame entre deux équipements adjacents. Pour signaler à son entité couche physique l'arrivée d'une trame, un équipement utilise la méthode *DataIND()* de la classe *couche*.

II.7. Gestion de la concurrence

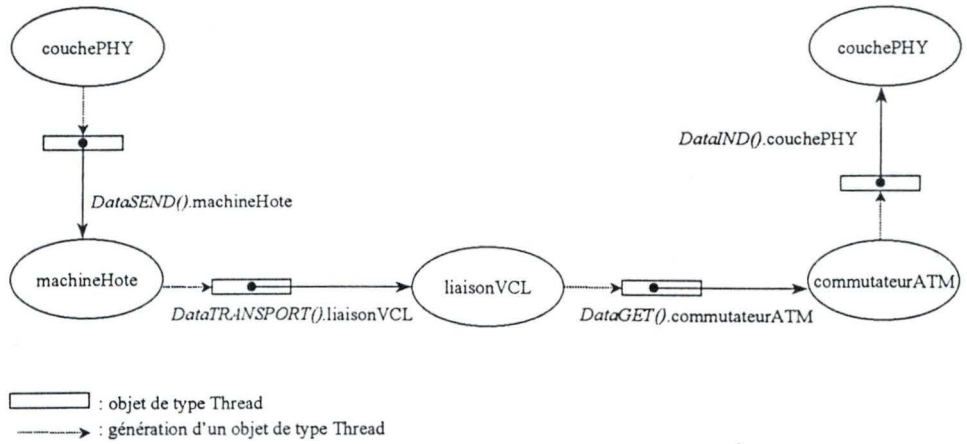


Figure 43 : *Scénario d'échange en asynchrone*

Toutes les primitives de service que nous implémentons sont asynchrones. Nous faisons recours au mécanisme de processus léger ou thread. Toute demande d'exécution d'une primitive de service est encapsulée dans un objet de type ***Thread*** (Figure 43). Ceci permet de libérer rapidement l'objet client. Pour atteindre cet objectif, nous avons ajouté une classe d'objets privée étendant la classe ***Thread*** de Java pour les classes suivantes :

- *coucheAPL*,
- *coucheAAL*,
- *coucheATM*,
- *couchePHY*,
- *equipementATM*,
- *liaisonVCC*.

Toutes ces classes privées ont été conçues sur base du squelette que voici :

```
Private class NomClasse extends Thread
   {
/*déclaration des attributs */

NomClasse(/* paramètres d'appel */ )
      {
      /* début corps du constructeur /*
        ...
/* fin corps du constructeur /*
      }
public run()
      {
       /* ajout de la référence du thread dans ListeThreads */
       RéfSousRéseau.AddRéfThread(this) ;
     /* tant que le sous-réseau est suspendu ... */
       while (RéfSousRéseau.isThreadSuspended())
                  {/* abandon du CPU par le processus courant */
          RéfSousRéseau.AbandonCPU();
                  }
       /* la suspension des processus est levée */
       if (RéfSousRéseau.isThreadActif(this))
            /*requête de service est lancée si et seulement si la
              référence du processus courant est présente dans la
```

```
                liste de références des processus actifs du sous-
                réseau courant */
            {
            /* début corps de la demande d'exécution de la
            primitive de service */
            ...
/* fin du corps de la demande d'exécution de la
            primitive de service */
            }
        }
}
```

Dans le corps de la méthode ***run()***, nous utilisons quatre méthodes fournies par la classe *sousReseau* : *AbandonCPU()*, *AddRéfThread()*, *isThreadActif()* et *isThreadSuspended()*. Elles sont explicitées dans les paragraphes qui suivent.

Pour supporter la concurrence, nous avons modifié la classe *sousReseau*, en y ajoutant des nouveaux attributs et méthodes. *ListeThreads* est la liste de références des processus actifs du sous-réseau. Le booléen *StopThreads* indique si les processus actifs doivent être stoppés ou non. Par contre, le booléen *SuspendThreads* indique si les processus actifs doivent être suspendus ou non.

En ce qui concerne les méthodes ajoutées, elles influent sur le cycle de vie de toute instance de la classe *sousReseau*. Comme illustré sur la Figure 44, un sous-réseau peut se trouver dans trois états possibles.

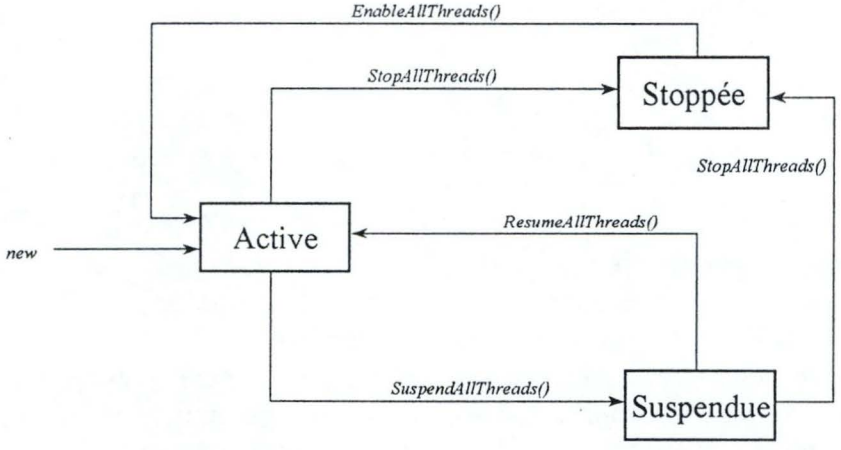


Figure 44 : *Cycle de vie d'une instance de la classe sousReseau*

1. état *actif*

Après sa création, le sous-réseau est dans un état *actif* caractérisé par les éléments suivants :

- *ListeThreads* est vide
- *StopThreads* et *SuspendThreads* ont la valeur ***false***.

Lorsque le sous-réseau est actif, tout processus généré ajoute sa référence dans la liste *ListeThreads* à l'aide de la méthode *AddRéfThread()*. Ainsi, cette liste contient à tout moment les processus en cours d'exécution. Dans la couche de traitement, nous avons ajouté un processus particulier nettoyant continuellement la liste *ListeThreads*, en effaçant la référence de tout processus inactif. Pour rappel, un processus meurt à la fin de son exécution.

Un sous-réseau actif peut être
- soit suspendu via la méthode *SuspendAllThreads()*. Cette méthode affecte à l'attribut *SuspendThreads* la valeur ***true***;
- soit stoppé via la méthode *StopAllThreads()*. Cette méthode initialise la liste *ListeThreads*, affecte à l'attribut *StopThreads* la valeur ***true*** et à l'attribut *SuspendThreads* la valeur ***false***.

2. état *suspendu*
L'état *suspendu* est caractérisé par les éléments suivants :
- *SuspendThreads* a la valeur ***true***
- *StopThreads* a la valeur ***false***.

Lorsque le sous-réseau entre dans cet état, tous les processus actifs sont suspendus. En réalité, tout processus actif s'endort (en exécutant la méthode *AbandonCPU()*) pendant un temps donné avant de vérifier, via la méthode *isThreadSuspended()*, la valeur de l'attribut *SuspendThreads*.

Un sous-réseau suspendu peut être
- soit réactivé via la méthode *ResumeAllThreads()*. Cette méthode affecte à l'attribut *SuspendThreads* la valeur ***false***;
- soit stoppé via la méthode *StopAllThreads()*.

3. état *stoppé*
L'état *stoppé* est caractérisé par les éléments suivants :
- *SuspendThreads* a la valeur ***false***
- *StopThreads* a la valeur ***true***.

Lorsque le sous-réseau entre dans un état *stoppé*, tous les processus actifs sont stoppés. Avant l'exécution effective d'une primitive de service, tout processus effectue, dans l'ordre, les deux opérations suivantes :

1. vérifier que le sous-réseau n'est pas suspendu via la méthode *isThreadSuspended()*. Dans le cas où le réseau est suspendu, le processus entre dans une boucle d'attente. Lorsque le sous-réseau est actif, le processus passe à l'opération suivante;
2. vérifier que le sous-réseau n'est pas stoppé via la méthode *isThreadActif()*. En effet, lorsque le sous-réseau est stoppé, la liste *ListeThreads* est initialisée et, par conséquent, la méthode *isThreadActif()* appliquée au processus courant renvoie ***false***. Dans ce cas, le processus n'exécute pas la primitive de service et meurt.

Un sous-réseau stoppé peut être redémarré via la méthode *EnableAllThreads()*. Cette méthode remet le sous-réseau dans un état *actif*.

II.8. Connexion avec la couche de présentation

Concernant les classes d'objets suivantes, nécessitant une représentation graphique, nous avons ajouté une variable d'instance, *RéfObjetGraphique*, avec deux méthodes associées, *GetRéfObjetGraphique() et SetRéfObjetGraphique()* :

- *sousReseau*,

- *equipement*,
- *liaisonPHY*,
- *couche*,
- *trame*.

Dans le module *Présentation*, chaque classe associée à une de ces classes d'objets possède un attribut pour référencer l'objet représenté.

Un autre élément de la connexion entre les deux couches de l'architecture de l'application est l'implémentation, dans la classe graphique associée à la classe *liaisonPHY* (en l'occurrence, la classe *P_liaisonPHY*), d'une méthode spécialisée dont l'objet est l'animation graphique des trames. Par convention, cette méthode est nommée *AnimationTrame()*. Dans le corps de cette méthode, l'animation effective d'une trame est encapsulée dans un objet de type ***Thread***. Et cette méthode (*AnimationTrame()*) retourne à la couche de traitement la référence du processus généré. Ceci permet à la couche de traitement de s'assurer de la fin de l'animation avant de poursuivre les opérations de transfert proprement dites.

III. Interfaces des classes d'objets

Dans ce qui suit, pour chaque classe d'objets de l'application, nous donnons une brève description de ses attributs et de son interface. Pour ne pas charger ce mémoire, nous avons choisi de ne pas y inclure une représentation en termes d'algorithmes spécifiant les différentes méthodes. En outre, par simplicité, les méthodes privées (intermédiaires) ne sont pas décrits dans ce mémoire. Toutefois, les codes sources des classes d'objets de l'application sont mis dans la partie annexe de ce mémoire. Pour des classes d'objets déclarées ***public*** et ajoutées lors de l'implémentation, nous faisons une brève introduction avant de décrire les attributs et l'interface.

Concernant la description, nous utilisons le formalisme suivant :

1. pour un attribut :

```
<NomAttribut> : <TypeAttribut>
<Commentaire>
```

2. pour une méthode :

```
<NomOpération> ({<NomParamètre> : <TypeParamètre>}+) : [<TypeRésultat>]
<Commentaire>
```

TypeAttribut, *TypeParamètre* et *TypeRésultat* sont soit des classes d'objets soit des types simples tels que ***int*** ou ***boolean***. Le type du résultat, pour une méthode, est optionnel. Le symbole "+" indique que le nombre de paramètres va de 0 à n.

Par ailleurs, nous supposons que le lecteur maîtrise la terminologie Java. Tous les mots-clés Java utilisés dans cette partie du mémoire sont définis dans le chapitre consacré à ce langage. Nous avons omis volontairement d'insérer des préconditions et des postconditions afin de ne pas trop charger le présent mémoire. Enfin, nous considérons, dans ce mémoire, un constructeur comme une méthode malgré que ceci soit discutable.

III.1. Module *Réseau*

Dans ce module, sauf indication contraire, toutes les méthodes sont déclarées ***public***.

Classe *sousReseau*

public abstract class *sousReseau*

Attributs

- *NomSousRéseau* : ***String***
  une chaîne de caractères désignant le sous-réseau.
- *AdresseIP* : ***String***
  une chaîne de caractères identifiant le sous-réseau dans un réseau des réseaux. Toutefois, nous traitons pas le cas de réseau des réseaux et par conséquent, aucun contrôle n'est prévu quant à l'unicité de cette valeur.
- *TypeProtocole* : ***String***
  une chaîne de caractères ("ATM", par exemple) spécifiant le type de protocole du sous-réseau.
- *ListeEquipements* : ***Vector***
  la liste de références des équipements constituant le sous-réseau.
- *RéfObjetGraphique* : *P_sousReseau* :
  la référence de l'objet graphique associé au sous-réseau.
- *ListeThreads* : ***Vector***
  la liste de références des processus actifs. Tout processus (objet de type ***Thread***) généré dans le sous-réseau a sa référence dans cette liste aussi longtemps qu'il est actif.
- *StopThreads* : ***boolean***
  un booléen indiquant si les processus au sein du sous-réseau doivent être stoppés ou non. Par défaut, sa valeur est ***true***. Cette valeur peut être modifiée par les méthodes *StopAllThreads()* et *EnableAllThreads()* de la classe *sousReseau*.
- *SuspendThreads* : ***boolean***
  un booléen indiquant si les processus au sein du sous-réseau doivent être suspendus ou non. Par défaut, sa valeur est ***false***. Cette valeur peut être modifiée par les méthodes *SuspendAllThreads()*, *EnableAllThreads()*, *StopAllThreads()* et *ResumeAllThreads()*.
- *SleepTime* : ***int***
  le temps d'attente du traitement (en ms) dans une entité couche. Cet attribut permet de gérer la vitesse d'exécution d'une simulation. Plus sa valeur est élevée, plus la simulation est lente. Par défaut, ce temps est nul.

Tous les attributs de la classe *sousReseau* sont déclarés ***protected***.

Méthodes

- *SetNomSousRéseau (NomSousRéseau* : ***String)*** : -
  assigne au sous-réseau le nom passé en paramètre.
- *GetNomSousRéseau ()* : ***String***
  renvoie le nom du sous-réseau.
- *AdapterCouches()* : -
  attribue à tout équipement du sous-réseau ses couches d'accès (les couches basses, spécifiques au type de protocole utilisé). Elle est une méthode abstraite et doit être redéfinie par toute sous-classe étendant la classe *sousReseau*.
- *AddRéfEquipement(RéfEquipement : equipement)* : ***boolean***

ajoute dans la liste *ListeEquipements* l'équipement dont la référence est passée en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.

- *GetListeEquipements() : **Vector***
  renvoie la liste *ListeEquipements* contenant les références des équipements présents dans le sous-réseau.
- *GetAdresseIP() : **String***
  renvoie l'adresse IP du sous-réseau.
- *GetTypeProtocole() : **String***
  renvoie le type de protocole du sous-réseau.
- *SetRéfObjetGraphique(RéfObjetGraphique : P_sousReseau) : -*
  associe l'objet graphique dont la référence est passée en paramètre au sous-réseau.
- *GetRéfObjetGraphique() : P_sousReseau*
  renvoie la référence de l'objet graphique associé au sous-réseau.
- *AddRéfThread(RéfThread : **Thread**) : **boolean***
  ajoute dans la liste *ListeThreads* le processus dont la référence est passée en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *StopAllThreads() : -*
  initialise la liste *ListeThreads*, affecte à l'attribut *StopThreads* la valeur ***true*** et à l'attribut *SuspendThreads* la valeur ***false***. Après l'application de cette méthode, tout processus actif dans le sous-réseau est interrompu.
- *EnableAllThreads() : -*
  initialise la liste *ListeThreads* et affecte aux attributs *StopThreads* et *SuspendThreads* la valeur ***false***. Après l'application de cette méthode, les échanges entre les équipements du sous-réseau sont autorisés.
- *SuspendAllThreads() : -*
  affecte à l'attribut *SuspendThreads* la valeur ***true.*** Après l'application de cette méthode, tout processus actif dans le sous-réseau est suspendu.
- *ResumeAllThreads () : -*
  affecte à l'attribut *SuspendThreads* la valeur ***false***. Après l'application de cette méthode, tout processus suspendu est réactivé.
- *isThreadStopped () : **boolean***
  renvoie la valeur de l'attribut *StopThreads*.
- *isThreadSuspended () : **boolean***
  renvoie la valeur de l'attribut *SuspendThreads*.
- *isThreadActif(RéfThread : **Thread**) : **boolean***
  renvoie la valeur ***true*** si le processus dont la référence est passée en paramètre est contenu dans la liste *ListeThreads*. Tout processus non répertorié dans cette liste doit interrompre son exécution. En effet, dans le corps de tout processus généré par l'application, pour la bonne marche du mécanisme de gestion des processus mis en place, nous utilisons cette méthode avant de lancer toute requête de service. Seuls les processus répertoriés dans cette liste peuvent lancer une requête de service.
- *GetNombreThreads() : **int***
  renvoie le nombre de processus actifs au sein du sous-réseau.
- *SetSleepTime(Temps : **int**) : -*
  affecte à l'attribut *SleepTime* la valeur *Temps* (en ms) passé en paramètre.
- *GoSleep() : -*
  met le processus courant (qui exécute la méthode) en attente pendant un temps égal à *SleepTime*. La vitesse d'exécution d'une simulation est gérée via cette méthode.
- *AbandonCPU() : -*

met le processus courant (qui exécute la méthode) en attente pendant 150 ms. Nous utilisons cette méthode pour gérer la concurrence entre processus. En effet, l'utilisation de cette méthode nous permet de donner à tout processus, la chance de prendre possession du CPU. Il faut noter que tous les processus au sein du sous-réseau ont la même priorité et un processus abandonnant le CPU se place automatiquement en queue de la file d'attente.

- *AbandonCPU(Temps : **int**) : -*
  met le processus courant en attente pendant un temps donné (en ms). Sa fonction est identique à celle de la méthode précédente. Nous l'avons ajoutée afin de nous permettre d'être à mesure de retarder l'exécution de certains processus (le processus d'animation de l'applet par exemple).

Classe *equipement*

***public abstract class** equipement*

Attributs

- *NomEquipement : **String***
  une chaîne de caractères désignant l'équipement.
- *AdresseEquipement : **String***
  une chaîne de caractère identifiant l'équipement dans son sous-réseau d'appartenance. Toutefois, aucun contrôle n'est prévu quant à l'unicité de cette valeur. C'est au programme client de réaliser ce contrôle.
- *TypeEquipement : **int***
  un entier spécifiant le type de l'équipement : *0* pour les équipements terminaux et *1* pour les équipements non terminaux.
- *RéfSousRéseau : sousReseau*
  la référence du sous-réseau contenant l'équipement. Pour rappel, un équipement n'appartient qu'à un et un seul sous-réseau à la fois.
- *ListeCouches : **Vector***
  la liste de références des entités couches appartenant à l'équipement.
- *ListeLiaisonsPhy : **Vector***
  la liste de références des liaisons physiques dont l'équipement est une des extrémités.
- *RéfObjetGraphique : P_equipement*
  la référence de l'objet graphique associé à l'équipement.

Tous les attributs de la classe *equipement* sont déclarés ***protected***.

Méthodes

- *SetNomEquipement(NomEquipement : **String**) : -*
  assigne le nom passé en paramètre à l'équipement.
- *GetNomEquipement() : **String***
  renvoie le nom de l'équipement.
- *GetAdresseEquipement() : **String***
  renvoie l'adresse de l'équipement.
- *GetRéfSousRéseau() : sousReseau*
  renvoie la référence du sous-réseau contenant l'équipement.
- *GetTypeEquipement() : **int***
  renvoie le type de l'équipement.
- *AddRéfCouche(RéfCouche : couche) : **boolean***

ajoute dans la liste *ListeCouches* de l'équipement l'entité couche dont la référence est passée en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.

- *GetListeCouches() : **Vector***
  renvoie la liste *ListeCouches* contenant les références des entités couches de l'équipement.
- *GetCoucheWithNom(NomCouche : **String**) : couche*
  renvoie la référence de la première entité couche de la liste *ListeCouches* dont le nom correspond à *NomCouche*.
- *AddRéfLiaisonPhy(RéfLiaisonPhy : liaisonPHY) : **boolean***
  ajoute dans la liste *ListeLiaisonsPhy* la liaison physique dont la référence est passée en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *GetListeLiaisonsPhy() : **Vector***
  renvoie la liste *ListeLiaisonsPhy* contenant les références des liaisons physiques dont l'équipement est une des extrémités.
- *DataGET(RéfTrame : trame) : **boolean***
  permet à l'équipement de réceptionner la trame dont la référence est passée en paramètre. Elle est une méthode abstraite et doit être redéfinie par toute sous-classe étendant la classe *equipement*. C'est essentiellement un service fourni à la classe *liaisonLOG*. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *DataSEND(RéfTrame : trame, RéfLiaisonLog : liaisonLOG) : **boolean***
  permet à l'équipement de réceptionner en vue de son envoi la trame dont la référence est passée en paramètre. Le deuxième paramètre précise la liaison logique vers laquelle la trame doit être envoyée. Elle est une méthode abstraite et doit être redéfinie par toute sous-classe étendant la classe *equipement*. C'est essentiellement un service fourni à la classe *couche*. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *SetRéfObjetGraphique(RéfObjetGraphique : P_equipement) : -*
  associe à l'équipement l'objet graphique dont la référence est passée en paramètre.
- *GetRéfObjetGraphique() : P_equipement*
  renvoie la référence de l'objet graphique associé à l'équipement.

Classe *couche*

***public abstract class** couche*

Attributs

- *NomCouche : **String***
  une chaîne de caractères désignant l'entité couche.
- *RéfEquipement : equipement*
  la référence de l'équipement contenant l'entité couche, une entité couche n'appartenant qu'à un et un seul équipement.
- *NiveauOSI : **int***
  le niveau OSI de l'entité couche.
- *ListeCouchesInf : **Vector***
  la liste de références des entités couches qui lui sont immédiatement inférieures.
- *ListeCouchesSup **Vector***
  la liste de références des entités couches qui lui sont immédiatement supérieures.
- *RéfObjetGraphique P_couche*
  la référence de l'objet graphique associé à l'entité couche.

Tous les attributs de la classe *couche* sont déclarés ***protected***.

Méthodes

- *GetNomCouche() : **String***
  renvoie le nom de l'entité couche.
- *GetRéfEquipement() : equipement*
  renvoie l'équipement contenant l'entité couche.
- *GetNiveauOSI() : **int***
  renvoie le niveau OSI de l'entité couche.
- *GetListeCouchesInf() : **Vector***
  renvoie la liste *ListeCouchesSup* contenant les références des entités couches qui sont immédiatement inférieures.
- *GetListeCouchesSup() : **Vector***
  renvoie la liste *ListeCouchesSup* contenant les références des entités couches qui sont immédiatement supérieures.
- *GetRéfSousRéseau() : sousReseau*
  renvoie la référence du sous-réseau courant.
- *GetCoucheWithNom(NomCouche : **String**, RéfListeCouches : **Vector**) : couche*
  renvoie la référence de la première entité couche de la liste *RéfListeCouches* ayant *NomCouche* comme nom. *RéfListeCouches* doit être soit *ListeCouchesSup* soit *ListeCouchesInf*.
- *AddRéfVoisineSup(RéfCouche : couche) : **boolean***
  ajoute dans la liste *ListeCouchesSup* la référence de l'entité couche dont la référence est passée en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *AddRéfVoisineInf(RéfCouche : couche) : **boolean***
  ajoute dans la liste *ListeCouchesInf* la référence de l'entité couche dont la référence est passée en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *DataREQ() : **boolean***
  permet à une entité couche de réceptionner une trame en vue de son envoi. A ce niveau, elle est une méthode vide. Elle devra être redéfinie ou surchargée au niveau des sous-classes.
- *DataIND() : **boolean***
  permet à une entité couche de réceptionner une trame. A ce niveau, elle est une méthode vide. Elle devra être redéfinie ou surchargée au niveau des sous-classes.
- *SetRéfObjetGraphique(RéfObjetGraphique : P_couche) : -*
  associe à l'entité couche l'objet graphique dont la référence est passée en paramètre.
- *GetRéfObjetGraphique() : P_couche*
  renvoie la référence de l'objet graphique associé à l'entité couche.

Classe *liaisonPHY*

***public final class** liaisonPHY*

Attributs

- *NuméroLiaison : **int***
  un numéro attribué à la liaison physique.
- *RéfSupport1 : supportPHY*
  la référence du premier objet de type *supportPHY* composant la liaison physique. Pour rappel, une liaison physique est une paire de supports physiques unidirectionnels.
- *RéfSupport2 : supportPHY*
  la référence du deuxième objet de type *supportPHY* composant la liaison physique.
- *RéfObjetGraphique: P_liaisonPHY*
  la référence de l'objet graphique associé à la liaison physique.

L'accès à ces attributs est limité au module *Réseau*.

Méthodes

- *liaisonPHY(NuméroLiaison : **int**, RéfEquipement1: equipement, RéfEquipement2 : equipement) : -*
  est le constructeur de la classe. Elle utilise le constructeur de la classe *supportPHY* afin de créer automatiquement les deux supports physiques composant la liaison physique et reliant les deux équipements dont les références sont passées en paramètre.
- *GetRéfSousRéseau() : sousReseau*
  renvoie la référence du sous-réseau courant.
- *GetRéfSupport1() : supportPHY*
  renvoie la référence du premier support physique de la liaison physique.
- *GetRéfSupport2() : supportPHY*
  renvoie la référence du deuxième support physique de la liaison physique.
- *GetSupportPhyDeSortie(RéfEquipement : equipement) : supportPHY*
  renvoie pour l'équipement dont la référence est passée en paramètre, la référence du support physique dont il est la source. L'équipement référencé par *RéfEquipement* doit être une des extrémités de la liaison physique.
- *SetRéfObjetGraphique(RéfObjetGraphique : P_liaisonPHY)*
  associe à la liaison physique l'objet graphique dont la référence est passée en paramètre.
- *GetRéfObjetGraphique() : P_liaisonPHY*
  renvoie la référence de l'objet graphique associé à la liaison physique.

Classe *supportPHY*

***public final class** supportPHY*

Attributs

- *RéfLiaisonPhy : liaisonPHY*
  la référence de la liaison physique dont le support physique est une des composantes.
- *RéfEquipementCible : equipement*
  la référence de l'équipement cible du support physique, un support physique étant unidirectionnel.
- *ListeLiaisonsLog : **Vector***
  la liste de références des liaisons logiques portées par le support physique. Une liaison logique étant unidirectionnelle est portée par un des supports physiques d'une liaison physique. Un support physique peut porter plusieurs liaisons logiques.

L'accès à ces attributs est limité au module *Réseau*.

Méthodes

- *supportPHY(RéfLiaisonPhy : liaisonPHY, RéfEquipementCible : equipement) : -*
  est le constructeur de la classe. Un support physique n'appartient qu'à une seule liaison physique.
- *GetRéfLiaisonPhy() : liaisonPHY*
  renvoie la référence de la liaison physique contenant le support physique.
- *GetRéfEquipementCible(): equipement*
  renvoie la référence de l'équipement cible du support physique.
- *GetRéfEquipementSource() : equipement*

renvoie la référence de l'équipement source du support physique c'est-à-dire l'équipement cible de l'autre support physique de la paire.

- *GetRéfSousRéseau() : sousReseau*
  renvoie la référence du sous-réseau courant.
- *GetListeLiaisonsLog() : **Vector***
  renvoie la liste *ListeLiaisonsLog* contenant les références des liaisons logiques portées par le support physique.
- *AddRéfLiaisonLog(RéfLiaisonLog : liaisonLOG) : **boolean***
  ajoute dans la liste *ListeLiaisonsLog* la liaison logique dont la référence est passée en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *DelRéfLiaisonLog(RéfLiaisonLog : liaisonLOG) : **boolean***
  enlève de la liste *ListeLiaisonsLog* la liaison logique dont la référence est passée en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.

Classe *liaisonLOG*

***public abstract class** liaisonLOG*

Attributs

- *NuméroLiaison : **int***
  un numéro attribué à la liaison logique.
- *RéfSupportPhy : supportPHY*
  la référence du support physique portant la liaison logique. Ce champ contient une valeur indéterminée pour une liaison logique entre deux équipements non adjacents.
- *ListeTrames : **Vector***
  la liste de références des trames en train d'être véhiculées par la liaison logique.

Tous les attributs de la classe *liaisonLOG* sont déclarés ***protected***.

Méthodes

- *GetNuméroLiaison(): **int***
  renvoie le numéro de la liaison logique.
- *GetRéfLiaisonPhy() : liaisonPHY*
  renvoie la référence de la liaison physique portant la liaison logique.
- *GetRéfSupportPhy() : supportPHY*
  renvoie plus particulièrement la référence du support physique portant la liaison logique.
- *GetRéfEquipementCible() : equipement*
  renvoie la référence de l'équipement cible de la liaison logique.
- *GetRéfEquipementSource() : equipement*
  renvoie la référence de l'équipement source de la liaison logique.
- *GetRéfSousRéseau() : sousReseau*
  renvoie la référence du sous-réseau courant.
- *GetListeTrames() : **Vector***
  renvoie la liste *ListeTrames* contenant les références des trames en train d'être véhiculées par la liaison logique.
- *AddRéfTrame(RéfTrame : trame) : **boolean***
  ajoute la référence d'une trame dans la liste *ListeTrames*. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *DelRéfTrame(RéfTrame : trame) : **boolean***

enlève une trame de la liste *ListeTrames*. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.

- *DataTRANSPORT() : **boolean***
  permet à la liaison logique de réceptionner une trame à envoyer vers son équipement cible. Elle est une méthode vide et doit être redéfinie ou surchargée au niveau des sous-classes. C'est essentiellement un service fourni à la classe *equipement*. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.

Classe *trame*

***public abstract class** trame*

Attributs

- *NuméroSéquence : **int***
  un numéro de séquence attribué à la trame.
- *Donnée: **String***
  la partie information de la trame.
- *RéfLocalisation : **Vector***
  une liste à deux éléments :
  1. le type de la localisation : "0" lorsque la trame se trouve dans une entité couche et plus précisément en cours de traitement dans sa méthode *DataREQ()*, "1" lorsqu'elle est en cours de traitement dans la méthode *DataIND()* d'une entité couche et "2" lorsqu'elle est en cours de traitement dans la méthode *DataTRANSPORT()* d'une liaison logique;
  2. la référence de l'objet traitant la trame (une entité couche ou une liaison logique).

  Cet attribut peut être utilisé pour commenter le parcours effectué par la trame. Nous ne considérons pas le cas où la trame est traitée par un équipement car sans intérêt pour l'application.
- *RéfObjetGraphique : P_trame*
  la référence de l'objet graphique associé à la trame.

Tous les attributs de la classe *trame* sont déclarés ***protected***.

Méthodes

- *GetNuméroSéquence() : **int***
  renvoie le numéro de séquence de la trame.
- *GetDonnée() : **String***
  renvoie le contenu du champ *Donnée* de la trame.
- *SetDonnée(Information : **String**) : -*
  affecte au champ *Donnée* de la trame la chaîne de caractères *Information* passée en paramètre .
- *GetRéfLocalisation() : RéfLocalisation*
  renvoie la liste *RéfLocalisation* de la trame.
- *SetRéfLocalisation(RéfCouche, "0"|"1") : -*
  permet de mettre à jour la localisation de la trame en cours de traitement dans une entité couche. *RéfCouche* est la référence de l'entité couche traitant la trame. Selon la valeur du deuxième paramètre de la méthode, la trame est soit dans un *DataREQ()* soit dans un *DataIND()*.
- *SetRéfLocalisation(RéfLiaisonLog) : -*

permet de mettre à jour la localisation de la trame en train d'être transportée par une liaison logique. *RéfLiaisonLog* est la référence de la liaison logique transportant la trame.

- *SetRéfObjetGraphique(RéfObjetGraphique : P_trame) : -*
  associe à la trame l'objet graphique dont la référence est passée en paramètre.
- *GetRéfObjetGraphique() : P_trame*
  renvoie la référence de l'objet graphique associé à la trame.

III.2. Module Atm

Sauf indication contraire, l'accès aux méthodes est limité aux classes d'objets du module *Atm*. En Java, un constructeur doit toujours être déclaré ***public***.

Classe *sousReseauATM*

public final class *sousReseauATM* ***extends*** *sousReseau*

Attribut

- *ListeLiaisonsVcc : **Vector***
  la liste de références des connexions VCC actives dans le sous-réseau ATM. L'accès à cet attribut est limité au sous-module.

Méthodes

- *sousReseauATM(AdresseIP : **String**) : -*
  est le constructeur de la classe *sousReseauATM*. Elle reçoit en paramètre l'adresse IP du réseau.
- *AdapterCouches() : -*
  affecte à tout équipement du sous-réseau ATM ses entités couches d'accès au sous-réseau ATM. Tout équipement du sous-réseau ATM doit contenir une entité couche physique et une entité couche ATM. En plus de ces entités couche, un équipement terminal (machine hôte) possède une entité couche AAL et une entité couche d'application.
- *AddRéfLiaisonVcc(RéfLiaisonVcc : liaisonVCC) : **boolean***
  ajoute dans la liste *ListeLiaisonsVcc* la connexion VCC dont la référence est passée en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *DelRéfLiaisonVcc(RéfLiaisonVcc : liaisonVCC) : **boolean***
  enlève de la liste *ListeLiaisonsVcc* la connexion VCC dont la référence est passée en paramètre. Cette suppression symbolise la fin du transfert de bout en bout entre les deux machines hôtes concernées par la dite connexion VCC. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *DelAllRéfLiaisonsVcc() : -*
  initialise à vide la liste *ListeLiaisonsVcc*. Elle doit être utilisée après l'utilisation de la méthode *StopAllThreads()* de la classe *sousReseau*. En effet, lors de l'exécution de la méthode *StopAllThreads()*, les connexions VCC encore actives ne sont pas supprimées. Il faudra les enlever pour avoir un système propre.
- *GetListeLiaisonsVcc() : **Vector***
  renvoie la liste *ListeLiaisonsVcc*.

Excepté les méthodes *AddRéfLiaisonVcc()* et *DelRéfLiaisonVcc()* dont l'accès est limité au module *Atm*, toutes les autres méthodes sont déclarées ***public***.

Classe *equipementATM*

public abstract class *equipementATM* ***extends*** *equipement*

Méthodes

- *DataGET(RéfCellule : celluleATM)* : ***boolean***
  permet à l'équipement ATM de réceptionner la cellule dont la référence est passée en paramètre. C'est essentiellement un service fourni à la classe *liaisonVCL*. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *DataSEND(RéfCellule : celluleATM, RéfLiaisonVcl : liaisonVCL)* : ***boolean***
  permet à l'équipement ATM de réceptionner pour son envoi la cellule dont la référence est passée en paramètre. Le deuxième paramètre précise le lien VCL vers lequel la cellule doit être envoyée. C'est essentiellement un service fourni à la classe *couchePHY*. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.

Classe *machineHote*

public final class *machineHote* ***extends*** *equipementATM*

Méthode

- *machineHôte(AdresseMachineHôte :* ***String****, RéfSousRéseau : sousReseau)* : -
  est le constructeur de la classe. Elle reçoit en paramètre l'adresse de la machine hôte et la référence du sous-réseau auquel appartient la machine hôte.

Classe *commutateurATM*

public final class *commutateurATM* ***extends*** *equipementATM*

Attributs

- *TableDeCommutation :* ***Vector***
  la liste de références des routes ATM actives. Chaque élément de la liste est une instance de la classe *routeATM*. Cette liste est utilisée pour aiguiller les cellules au travers du sous-réseau. Elle est représente une table de commutation du commutateur.
- *TableDesAdressesAtm :* ***Vector***
  la liste de références des adresses ATM des machines hôtes du sous-réseau. Chaque élément de la liste est une instance de la classe *routeATM*. Cette liste est utilisée lors de la génération d'une connexion VCC. Elle représente la table d'adresses préconfigurée du commutateur.

Méthodes

- *commutateurATM(RéfSousRéseau : sousReseau)* : -
  est le constructeur de la classe. Elle reçoit en paramètre la référence du sous-réseau auquel appartient le commutateur.
- *AddRéfRoute(RéfLiaisonVclEntrée : liaisonVCL, RéfLiaisonVclSortie : liaisonVCL)* : ***boolean***
  ajoute dans la table de commutation du commutateur la route ATM dont les références des composantes sont passées en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *GetRéfLiaisonVclSortant(RéfLiaisonVclEntrée : liaisonVCL) : liaisonVCL*
  renvoie la référence du lien VCL de sortie correspondant au lien VCL d'entrée dont la référence est passée en paramètre.
- *DelRéfRoute(RéfLiaisonVclEntrée : liaisonVCL)* : ***boolean***

enlève de la table de commutation, la route ATM dont la référence du lien VCL d'entrée correspond à celle passée en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.

- *GetRéfLiaisonPhyDeSortie(AdresseMachineHôte : **String**) : liaisonPHY*
  renvoie la référence de la liaison physique à emprunter pour atteindre la machine hôte dont l'adresse est passée en paramètre.
- *AddRéfAdresseAtm(RéfMachineHôteCible : machineHôte, RéfLiaisonPhy de sortie : liaisonPHY) : **boolean***
  ajoute dans la table d'adresses ATM, l'adresse ATM dont les références des composantes sont passées en paramètre. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.

Excepté les méthodes *AddRéfRoute()* et *DelRéfRoute()* dont l'accès est limité au module *Atm*, toutes les autres méthodes sont déclarées ***public***.

Classe *adresseATM*

***final class** adresseATM*

Attributs

- *AdresseMachineHôte **String***
  l'adresse de la machine hôte à atteindre.
- *RéfLiaisonPhy: liaisonPHY*
  la référence de la liaison physique à emprunter pour atteindre la machine hôte dont l'adresse est *AdresseMachineHôte*.

Méthodes

- *adresseATM(AdresseMachineHôte : **String**, RéfLiaisonPhy : liaisonPHY) : -*
  est le constructeur de la classe. Elle reçoit en paramètre l'adresse de la machine hôte à atteindre et la référence de la liaison physique à emprunter.
- *GetAdresseMachineHôte() : **String***
  renvoie l'adresse de la machine hôte à atteindre.
- *GetRéfLiaisonPhy() : liaisonPHY*
  renvoie la référence de la liaison physique à emprunter.

Classe *routeATM*

***final class** routeATM*

Attributs

- *RéfLiaisonVclEntrée : liaisonVCL*
  la référence du lien VCL d'entrée.
- *RéfLiaisonVclSortie : liaisonVCL*
  la référence du lien VCL de sortie correspondant au lien VCL d'entrée *RéfLiaisonVclEntrée*.

Méthodes

- *routeATM(RéfLiaisonVclEntrée : liaisonVCL, RéfLiaisonVclSortie : liaisonVCL) : -*
  est le constructeur de la classe. Elle reçoit en paramètre les références des liens VCL d'entrée et de sortie correspondant.
- *GetRéfLiaisonVclEntrée() : liaisonVCL*
  renvoie la référence du lien VCL d'entrée de la route ATM.

- *GetRéfLiaisonVclSortie() : liaisonVCL*
  renvoie la référence du lien VCL de sortie de la route ATM.

Classe *liaisonVCL*

public final class *liaisonVCL*

Attributs

- *VPI* : ***int***
  le numéro de la connexion VPC (*Virtual Path Connection*) à laquelle appartient le lien VCL.
- *RéfLiaisonVcc : liaisonVCC*
  la référence de la connexion VCC à laquelle appartient le lien VCL.

Méthodes

- *liaisonVCL(RéfLiaisonVcc : liaisonVCC, RéfSupportPhy : liaisonPHY) : -*
  est le constructeur de la classe. Elle reçoit en paramètre la référence de la connexion VCC à laquelle il appartient et la référence de la liaison physique le portant.
- *GetVPI() :* ***int***
  renvoie le contenu du champ *VPI*.
- *GetRéfLiaisonVcc() : liaisonVCC*
  renvoie la référence de la connexion VCC à laquelle appartient le lien VCL.
- *DataTRANSPORT(RéfEquipementSource : equipementATM, RéfCellule : celluleATM) :* ***boolean***
  permet au lien VCL de réceptionner la cellule dont la référence est passée en paramètre. C'est essentiellement un service fourni à la classe *equipementATM*. *RéfEquipementSource* est la référence de l'équipement source du lien VCL. La méthode renvoie la valeur ***true*** si l'exécution aboutit normalement.

  Excepté les méthodes *liaisonVCL()* et *DataTRANSPORT()* dont l'accès est limité au module *Atm*, toutes les autres méthodes sont déclarées ***public***.

Classe *liaisonVCC*

public final class *liaisonVCC*

Attributs

- *RéfMachineHôteSource : equipement*
  la référence de la machine hôte initiatrice du transfert.
- *RéfMachineHôteCible : equipement*
  la référence de la machine hôte réceptrice du transfert.
- *RéfFirstLiaisonVcl : liaisonVCL*
  la référence du lien VCL créé entre la machine hôte source et son commutateur d'entrée au sous-réseau ATM.
- *RéfLastLiaisonVcl* : *liaisonVCL*
  la référence du lien VCL créé entre la machine hôte de destination et son commutateur d'entrée au sous-réseau ATM. C'est sur base de ces deux références (*RéfFirstLiaisonVcl* et *RéfLastLiaisonVcl*) que les cellules sont acheminées de bout en bout au travers du sous-réseau.
- *NbreCellulesReçues* : ***int***
  le nombre de cellules arrivées à destination. Dans notre application, la perte d'une cellule n'est pas prévue.

Méthodes

- *liaisonVCC(RéfMachineHôteSource : machineHôte, RéfMachineHôteCible : machineHôte) : -*
  est le constructeur de la classe. Elle reçoit en paramètre les références des machines hôtes source et de destination.
- *GetRéfMachineHôteSource() : machineHôte*
  renvoie la référence de la machine hôte source de la connexion VCC.
- *GetRéfMachineHôteCible() : machineHôte*
  renvoie la référence de la machine hôte de destination de la connexion VCC.
- *SetRéfFirstLiaisonVcl(RéfLiaisonVcl : liaisonVCL) : -*
  affecte à la connexion VCC son premier lien VCL.
- *SetRéfLastLiaisonVcl(RéfLiaisonVcl : liaisonVCL) : -*
  affecte à la connexion VCC son dernier lien VCL.
- *GetRéfFirstLiaisonVcl() : liaisonVCL*
  renvoie la référence du premier lien VCL de la connexion VCC.
- *GetRéfLastLiaisonVcl() : liaisonVCL*
  renvoie la référence du dernier lien VCL de la connexion VCC.
- *GetNbreCellulesReçues() : **int***
  renvoie le nombre de cellules déjà arrivées à destination.
- *GetNbreCellulesTransmises() : **int***
  renvoie le nombre de cellules qui doivent être transmises à la machine hôte de destination.
- *CelluleArrivée() : -*
  incrémente le champ *NbreCellulesReçues* d'une unité chaque fois qu'une cellule arrive à destination.
- *GetRéfSousRéseau() : sousReseau*
  renvoie la référence du sous-réseau courant.

Excepté les méthodes *SetRéfFirstLiaisonVcl()*, *SetRéfLastLiaisonVcl()* et *CelluleArrivée()* dont l'accès est limité au module *Atm*, toutes les autres méthodes sont déclarées ***public***.

Classe *celluleATM*

***public final class** celluleATM **extends** trame*

Attributs

- *RéfLiaisonVcl : liaisonVCL*
  la référence du lien VCL de sortie que doit emprunter la cellule.
- *PT : **int***
  un champ indiquant le type de la cellule.
- *CLP : **int***
  un champ précisant la priorité assignée à la cellule.

Méthodes

- *celluleATM(RéfLiaisonVcl : liaisonVCL, PT : **int**, CLP : **int**, Donnée : **String**) : -*
  est le constructeur de la classe. Elle reçoit en paramètre la référence du lien VCL de sortie, le type de la cellule, sa priorité et l'information à contenir. Dans l'application, les valeurs de *PT* et *CLP* sont fixes et n'influent pas sur le déroulement d'une simulation.
- *GetPT() : **int***
  renvoie le contenu du champ *PT*.

- *GetCLP() : **int***
  renvoie le contenu du champ *CLP*.
- *GetRéfLiaisonVcl() : liaisonVCL*
  renvoie la référence du lien VCL de sortie de la cellule.
- *GetRéfLiaisonVcc() : liaisonVCC*
  renvoie la référence de la connexion VCC à laquelle appartient le lien VCL de sortie de la cellule.
- *GetNuméroVcl() : **int***
  renvoie le numéro du lien VCL de sortie de la cellule.
- *GetVPI() : **int***
  renvoie le numéro de la connexion VPC à laquelle appartient le lien VCL de sortie de la cellule.
- *GetRéfMachineHôteSource() : machineHôte*
  renvoie la référence de la machine hôte qui a généré la cellule.
- *GetRéfMachineHôteCible() : machineHôte*
  renvoie la référence de la machine hôte à qui est destinée la cellule.
- *SetCLP(CLP : **int**) : -*
  affecte à la cellule la priorité *CLP* passée en paramètre.
- *SetRéfLiaisonVcl(RéfLiaisonVcl : liaisonVCL) : -*
  affecte à la cellule le lien VCL de sortie dont la référence est passée en paramètre.

Excepté les méthodes *celluleATM()*, *SetRéfLiaisonVcl()* et *SetCLP()* dont l'accès est limité au module *Atm*, toutes les autres méthodes sont déclarées ***public***.

Classe *coucheATM*

***public final class** coucheATM*

Méthodes

- *coucheATM(NomCouche : **String** , RéfEquipement : equipementATM) : -*
  est le constructeur de la classe. Elle reçoit en paramètre le nom de l'entité couche et la référence de l'équipement auquel appartient cette entité couche. Dans l'application, *NomCouche* correspond toujours à la chaîne de caractères "ATM".
- *DataIND(RéfCellule : celluleATM) : **boolean***
  permet à la couche ATM de réceptionner la cellule reçue dont la référence est passée en paramètre. C'est essentiellement un service fourni à la classe *couchePHY*. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *DataREQ(RéfLiaisonVcc : liaisonVCC, Donnée : **String**) : **boolean***
  permet à la couche ATM de réceptionner une donnée à envoyer vers l'entité couche AAL de la machine de destination de la connexion VCC dont la référence est passée en paramètre. C'est essentiellement un service fourni à la classe *coucheAAL*. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.

Classe *couchePHY*

***public final class** couchePHY*

Méthodes

- *couchePHY(NomCouche : **String**, RéfEquipement : equipementATM) : -*
  est le constructeur de la classe. Elle reçoit en paramètre le nom de l'entité couche et la référence de l'équipement auquel appartient cette entité couche. Dans l'application, *NomCouche* correspond toujours à la chaîne de caractères "PHY".

- *DataIND(RéfCellule : celluleATM)* : **boolean**
  permet à la couche physique de réceptionner la cellule dont la référence est passée en paramètre. C'est essentiellement un service fourni à la classe *equipement*. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.
- *DataREQ(RéfLiaisonVcl : liaisonVCL, RéfCellule : celluleATM)* : **boolean**
  permet à la couche physique de réceptionner la cellule dont la référence est passée en paramètre. Cette cellule devra être envoyée sur le lien VCL dont la référence est passée en paramètre. C'est essentiellement un service fourni à la classe *coucheATM*. Elle renvoie la valeur ***true*** si l'exécution aboutit normalement.

Classe *coucheAAL*

public final class *coucheAAL*

Méthodes

- *coucheAAL(NomCouche : **String**, RéfEquipement : machineHôte)* : -
  est le constructeur de la classe. Elle reçoit en paramètre le nom de l'entité couche et la référence de la machine hôte à laquelle appartient cette entité couche. Dans l'application, *NomCouche* correspond toujours à la chaîne de caractères "AAL".
- *DataIND(Donnée : **String**)* : **boolean**
  Cette méthode est fournie à l'entité couche APL. Dans ce module de simulation, cette méthode est vide. Le sous-réseau est attaqué à partir de l'entité couche AAL.
- *DataREQ(RéfLiaisonVcl : liaisonVCL, Volume : **int**)* : **boolean**
  Cette méthode est fournie normalement à l'entité couche APL. Dans ce module de simulation, cette méthode sera utilisée dans le programme principal pour lancer un transfert d'information sur une connexion VCC existante. Le paramètre volume indique le nombre d'unités de données de 48 octets à générer. Elle renvoie la valeur ***true*** si son exécution aboutit normalement.

Seule la méthode *DataREQ()* est déclarée ***public***. C'est par l'entité couche AAL que nous attaquons les simulations.

Classes *coucheAPL*

public final class *coucheAPL*

Méthodes

- *coucheAPL(NomCouche : **String**, RéfEquipement : machineHôte)* : -
  est le constructeur de la classe. Elle reçoit en paramètre le nom de l'entité couche et la référence de la machine hôte à laquelle appartient cette entité couche. Dans l'application, *NomCouche* correspond toujours à la chaîne de caractères "APL".

IV. Conclusion

La couche de traitement renferme tous les traitements de l'application. Elle offre à la couche de présentation et aux programmes clients un ensemble de méthodes leur permettant de fournir les fonctionnalités attendues. La connexion principale entre les deux couches de l'architecture se réalise via les attributs *RéfObjetGraphique* et les méthodes associées. Notons que la couche de traitement telle que nous l'avons conçue peut recevoir d'autres types de protocoles, la classe *sousReseau* étant une classe de base.

Par ailleurs, pour permettre une exécution concurrente dans le système, nous utilisons le mécanisme de thread. Sur base de ce même mécanisme, il est possible à un programme client de gérer l'exécution d'une simulation (démarrage, suspension et arrêt).

Le chapitre suivant aborde la deuxième couche de l'architecture de notre application.

Chapitre IX : Couche de présentation

I. Introduction

Dans ce chapitre, nous essayons d'expliquer au lecteur les choix que nous avons faits lors de la création de la couche de présentation de l'application.

Dans une première section, nous expliquerons les différents choix d'implémentation.

Ensuite, nous présenterons de façon succincte les interfaces des classes d'objets que nous avons dû définir.

Nous terminerons pas une brève conclusion.

II. Choix d'implémentation

La couche de présentation est composée uniquement du module *Presentation.* Cette couche a été implémentée essentiellement en Java. Cependant nous y utilisons de nombreux éléments du script externe de VRML afin de pouvoir présenter à l'utilisateur de l'application une interface en trois dimensions dans une scène VRML. Le lecteur peut donc se référer aux chapitres traitant de ces sujets et nous les supposerons connus.

Nous commencerons par expliquer pourquoi nous avons choisi d'utiliser le script externe pour permettre l'interaction entre les langages Java et VRML. Cette interaction entre les deux langages étant possible dans une page HTML, nous décrivons brièvement cette page HTML et les problèmes qui peuvent y survenir.

Ensuite, nous expliquerons la méthode utilisée pour animer les cellules ATM dans la scène VRML.

Dans une troisième section, nous décrirons les différences qui apparaissent dans l'applet du simulateur en comparaison avec le squelette général d'une applet d'animation présenté dans le chapitre relatif au langage Java.

Enfin, nous décrirons les trois méthodes associées aux trois animations des concepts à illustrer.

II.1 Utilisation du script externe

Les principes de fonctionnement d'un réseau ATM sont des principes assez complexes à comprendre. La quantité d'informations nécessaires à cette compréhension est importante également. Un des souhaits exprimés dans le cahier des charges dont nous avons parlé précédemment concerne les animations mises en œuvre par le simulateur. Celles-ci doivent se dérouler dans un environnement à trois dimensions décrit par le langage VRML. Le moteur de la simulation quant à lui devait être programmé dans un langage portable et adaptable au monde Internet et aux pages Web. Nous savons que le langage de programmation à choisir était Java Il nous restait à connecter les deux langages de la façon la plus appropriée au vu des objectifs que nous voulions atteindre.

L'animation seule ne permettait pas d'expliquer correctement des concepts aussi vastes que la commutation ou les fonctionnalités remplies par chaque couche, il fallait que celle-ci soit accompagnée d'une information plus "manuscrite". Cette information supplémentaire ne nécessitait pas d'apparaître dans un monde en trois dimensions. Le lien entre une applet Java et un monde VRML nous paraissait dès lors être le meilleur moyen de combiner ces deux grands éléments : une animation en 3D accompagnée d'informations "textuelles". Le script à utiliser était donc le script externe puisqu'il permet à un monde VRML de communiquer avec son environnement extérieur et notamment avec une applet Java qui se trouve sur la même page HTML.

Cependant, l'utilisation du script externe et de l'*External Authoring Interface*[11] ne fut pas une chose aisée car, au moment de la réalisation de l'application, cet outil était bien loin d'être standardisé. Une des premières conséquences de cette non standardisation fut le choix des outils informatiques avec lesquels nous allions travailler.

- En ce qui concerne le visualisateur[12] et l'*External Authoring Interface* nous avons pris ceux fournis par SILICON GRAPHICS dans son plugin COSMO PLAYER. Nous avons utilisé ce plugin dans sa version 2 bêta 1. Pourquoi ce plugin plutôt qu'un autre? Tout simplement parce que l'*EAI* fournie avec COSMO PLAYER et utilisée dans le cadre de ce projet est peut être (et même probablement) celle qui deviendra le standard EAI de VRML 2.0. De plus, COSMO PLAYER fait partie des plugins qui permettent d'utiliser l'EAI dans un environnement Netscape (navigateur que nous avons utilisé);
- Comme navigateur WWW nous avons utilisé le navigateur fourni dans le package COMMUNICATOR de NETSCAPE. Et, ce, tout simplement parce que cette version est, à l'époque de notre stage, la plus récente de chez NETSCAPE et que ce navigateur est un des seuls navigateurs avec lequel un browser VRML peut être associé. En effet, outre ce navigateur seuls trois autres permettent une association avec un browser VRML et il s'agit dans tous les cas de navigateurs de la société NETSCAPE.

Comme nous l'avons déjà mentionné dans le chapitre relatif au langage VRML, il est nécessaire de lier certaines variables du monde Java à des objets du monde VRML pour permettre une interaction correcte de ces deux langages. L'EAI contient tous les types nécessaire à la création de ces divers liens. C'est par exemple le cas des types ***EventInMFNode***, ***Browser***, ***Node[]***, etc. que le lecteur retrouvera dans la description des classes d'objets du module de présentation. Pour une description plus détaillée de ces types nous renvoyons le lecteur aux multiples sources qui décrivent l'EAI.

II.2 Description de la page HTML

Pour permettre l'interaction d'une scène VRML et d'une applet Java, il est important que ces dernières se trouvent toutes deux sur la même page HTML. Cette section décrit brièvement au lecteur la page HTML permettant cette collaboration entre les deux langages ainsi que les problèmes que l'on peut rencontrer.

Les personnes qui ont déjà observé les pages sources associées aux pages HTML que l'on peut visualiser à travers le WWW savent que toute page HTML est composée d'un ensemble

[11] En abrégé : EAI

[12] Un visualisateur est un outil qui, une fois associé à un browser Internet, permet à tout observateur de page HTML de voir et d'interagir (au moyen d'un tableau de bord) avec les scènes VRML qu'il rencontre.

d'étiquettes ("tags" en anglais) qui en précisent les caractéristiques (telles que par exemple le format de la police, les insertions d'image etc...).

Pour insérer une applet dans une page HTML, il faut utiliser les étiquettes <APPLET> </APPLET>. Il est important de préciser quelle applet on veut insérer dans la page, cela peut se faire grâce à l'attribut CODE. Ecrire <APPLET CODE = test.class> </APPLET> signifie que l'on veut insérer dans la page HTML l'applet Java qui s'appelle test. Le lecteur remarquera que l'on référence le code objet de l'applet (c'est à dire le code compilé) et pas le code source, ce qui semble assez logique étant donné que cette commande permet de demander au navigateur WEB d'ouvrir le fichier et de l'exécuter sur la machine virtuelle Java.

Cette étiquette possède deux autres options qui permettent de définir la taille que la fenêtre associée à l'applet doit avoir dans la page HTML. Cette taille peut être donnée en pixels ou en pourcentage d'écran. La ligne <APPLET CODE = test.class WIDTH = 100 HEIGHT = 100> </APPLET> fera apparaître dans la page HTML, l'applet test dans un carré de 100 pixels sur 100. Cette façon de travailler peut provoquer des variations de l'affichage en fonction des paramètres de l'ordinateur sur lequel l'utilisateur consulte la page HTML. La ligne <APPLET CODE = test.class WIDTH = 25% HEIGHT = 25%> </APPLET> affichera à l'écran l'applet test dans une zone qui fait 25% de la largeur de la page et 25 % de sa hauteur.

Pour insérer une scène VRML dans une page HTML, on utilise l'étiquette <EMBED></EMBED>. Tout comme dans l'étiquette <APPLET> </APPLET> il fallait préciser quelle applet doit être exécutée, il faut également dans ce cas dire au navigateur quel monde VRML il doit ouvrir et afficher. Pour cela on utilise le mot clé SRC. La ligne <EMBED SRC = test.wrl></EMBED> indique au navigateur que l'on veut insérer une scène VRML décrite par le fichier test.wrl. On peut aussi ajouter différentes options:

- La largeur de la scène VRML dans la page. Elle peut être exprimée en pixels ou en un pourcentage de la largeur de la page HTML;
- La hauteur de la scène VRML dans la page. Elle peut être exprimée en pixels ou en un pourcentage de la hauteur de la page HTML;
- Si on veut définir une bordure autour de la scène VRML et, si oui, donner la largeur de cette bordure en pixels, il faut utiliser le mot clé BORDER.

Dans le cas de notre application, le fichier HTML que nous avons créé est le suivant :

```
<HTML>
<HEAD>
<TITLE>Simulateur ATM</TITLE>
</HEAD>
<BODY>

  <embed src="genere1.wrl" border=0 width=100% height=75%>
  <Applet code="SimuLibre.class" mayscript width=100%
height=25%></Applet>

</BODY>
</HTML>
```

Le fichier genere1.wrl contient quelques éléments définis dans le monde VRML plutôt que de manière dynamique dans les fichiers Java du module de présentation ou dans l'applet Java. Ces quelques éléments sont les suivants :

- Le sol sur lequel le réseau ATM sera dessiné;
- La couleur du fond de la scène VRML;
- Les différents points de vue offerts à l'utilisateur et repris dans le tableau de bord de la scène VRML. Il s'agit du point de vue aérien, du point de vue en hauteur et d'un point de vue intermédiaire que nous avons appelé point de vue en perspective;
- Deux nœuds vides qui seront utilisés par les variables de l'applet Java pour créer les objets statiques et les objets dynamiques dans la scène VRML;
- Une lampe (nœud prédéfini du langage VRML) qui éclairera le réseau ATM une fois ce dernier créé par l'applet Java.

L'instruction "mayscript" signifie uniquement que ce fichier servira entre autres à faire ce que nous avons appelé dans le chapitre consacré au langage VRML du script externe.

Le comportement d'un navigateur Internet qui doit traiter les deux étiquettes que nous venons de décrire n'a pas encore été standardisé par le consortium WWW. De ce fait, tous les navigateurs ne les interprètent pas de la même façon et l'utilisation normale de ces navigateurs peut avoir des conséquences inattendues sur l'affichage de la page HTML à l'écran. Nous avons, par exemple, rapidement constaté qu'il était impossible de recharger les pages contenant ces étiquettes sans provoquer une erreur générale. De même, il est important que le navigateur ne tienne jamais compte, à l'ouverture, du contenu de son cache. Sans quoi, cela revient à recharger la page. Les outils que nous avons choisis nous ont permis de ne pas avoir à traiter trop souvent ce genre d'erreur. Cependant, tant que les étiquettes et la façon de les interpréter n'auront pas été standardisées, les erreurs de ce genre seront fréquentes.

II.3 L'animation des cellules ATM

Lors de l'illustration des différents concepts par le simulateur, l'application crée une animation des cellules ATM sur les liens physiques du réseau ATM dans le monde en trois dimensions. Suite à la non standardisation de l'EAI, il est impossible de gérer les événements que l'on pourrait associer aux objets créés dynamiquement dans le monde VRML. Dès lors, pour déplacer une cellule sur le lien, nous ne pouvions pas simplement changer ses coordonnées comme nous l'aurions fait dans le cas où seule une scène VRML aurait été visible sur la page Web. Il nous a donc fallu trouver une solution pour déplacer (et donc animer) les cellules ATM.

La première solution envisagée fut de créer la représentation graphique d'une cellule à une coordonnée particulière, de la détruire et de la recréer à ses nouvelles coordonnées. Ce mécanisme est illustré par la Figure 45.

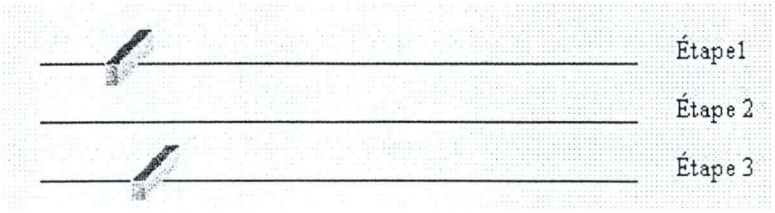


Figure 45 : Une première solution d'animation des cellules ATM

Cette solution posait un problème majeur : dès que le nombre de cellules voyageant à travers le réseau était supérieur à trois, le clignotement provoqué par les destructions et les constructions répétitives devenait trop important pour être supportable par l'œil humain. De plus, ce clignotement risquait d'embrouiller l'apprenant qui utiliserait l'application. D'un point de vue ergonomique, cette solution était loin d'être la meilleure. Il fallait donc supprimer le problème du clignotement.

Très vite, nous avons réalisé que le clignotement des cellules était dû au passage par l'état vide (l'étape 2 sur la Figure 45). Nous avons donc décidé d'associer deux représentations graphiques à la cellule, une à son ancienne position et une à sa nouvelle. La destruction de la représentation graphique de la cellule se trouvant à l'ancienne position ne se fait qu'après avoir affiché la cellule à sa nouvelle position. Ce mécanisme est illustré par la Figure 46.

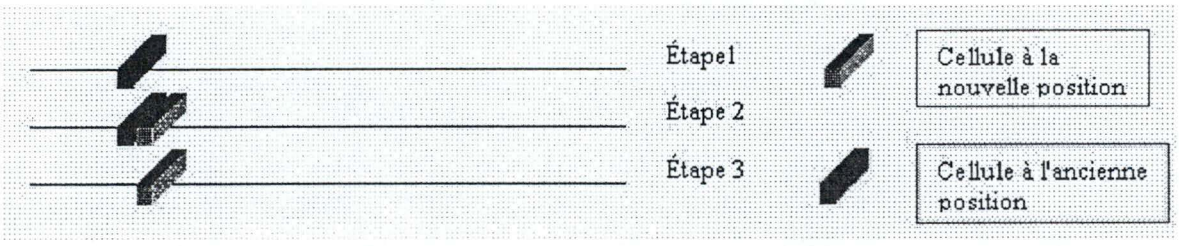


Figure 46 : deuxième solution pour l'animation des cellules ATM

Nous avons gardé cette solution car l'animation des cellules est en général suffisamment rapide pour que l'œil humain ne perçoive pas les deux représentations graphiques des cellules simultanément. L'utilisateur de l'application a, de cette façon, l'impression que la cellule circule sur le lien physique de manière tout à fait normale et fluide. Il nous faut cependant préciser au lecteur de ce mémoire, que par moment sans que nous puissions y donner une explication, l'utilisateur percevra un étirement de la représentation graphique de la cellule ATM. Ce genre de phénomène apparaît entre autres lorsque le processeur de l'ordinateur sur lequel l'application est exécutée est "surchargé". Nous n'avons pas eu la possibilité de voir si ce problème persistait en utilisant un ordinateur avec un processeur plus puissant.

II.4 L'applet d'animation

Dans le chapitre relatif au langage Java, nous avons décrit le squelette général d'une applet d'animation. L'applet exécutée dans l'application suit le même canevas général que ce squelette. Cependant, certains changements ou certaines nouveautés doivent y être intégrés pour permettre le fonctionnement du simulateur.

Rappelons que toutes les animations qui se déroulent dans la scène VRML sont contrôlées par l'applet Java. C'est donc cette dernière qui est l'instigatrice des animations sous la demande de l'utilisateur bien entendu.

Tout d'abord il est important d'importer les packages de l'EAI. Les packages supplémentaires à importer sont les suivants :

```
import vrml.external.field.*;
import vrml.external.Node;
import vrml.external.Browser;
import vrml.external.exception.*;
import netscape.javascript.JSException;
```

Dans l'exemple précédent, les noms des packages importés correspondent aux noms des packages fournis avec le plugin COSMO PLAYER. Si nous avions utilisé le plugin COMMUNITY PLACE de la société SONY, ces noms auraient certainement été un peu différents.

Ces packages permettent entre autres d'utiliser tous les types propres à VRML pour des variables Java. Il n'est donc pas possible de faire le lien entre l'applet Java et la scène VRML sans ces derniers.

Dans la partie de l'applet consacrée à la déclaration des variables, nous avons déclaré plusieurs grands groupes de variables :

- Les variables permettant de décrire le réseau ATM. Leurs types appartiennent aux types définis dans les classes d'objets des modules *Réseau, Atm* et *Présentation;*
- Les variables utiles pour définir l'interface de l'applet Java. Elles permettent de placer dans l'applet Java des éléments d'interaction entre l'utilisateur de l'application et le simulateur (bouton Stop, Démarrage, Pause etc.);
- Les variables que nous appellerons d'informations. Elles permettent d'afficher dans l'applet Java des informations plus textuelles relatives aux concepts illustrés;
- Les variables permettant l'interaction entre l'applet Java et la scène VRML. Ces variables devront être liées aux nœuds du monde VRML dans la partie de l'applet consacrée à l'initialisation.

En plus des méthodes définies dans le squelette de l'applet d'animation, deux nouvelles méthodes sont évoquées dans l'applet de l'application. Il s'agit des méthodes suivantes :

- *public boolean handleEvent (Event evt)*
- *public boolean Action (Event evt, Object arg)*

Ces méthodes permettent de gérer les actions de l'utilisateur sur les objets composant l'interface de l'applet (appui sur le bouton Stoppe, Démarrage etc.). Chaque action de l'utilisateur dans l'interface de l'applet ayant une répercussion sur l'animation qui se déroule dans la scène VRML, ces deux méthodes sont très importantes car elles permettent à l'utilisateur de "contrôler" l'animation.

Dans la partie réservée à l'initialisation nous avons dû trouver un moyen de gérer une erreur fréquente de l'EAI. Comme à l'époque de la réalisation de l'application, cette dernière n'était

pas standardisée, il n'était pas rare que le lien qui aurait dû exister entre l'applet Java et la scène VRML ne soit pas créé. Dans ce cas, l'utilisateur pouvait voir apparaître au bas de son écran une information lui signalant que l'applet était en cours d'exécution mais rien de plus, aucune scène VRML, aucun bouton dans l'applet etc. Ce problème venait du fait que l'assignation à une des variables Java de l'instance du browser VRML utilisé était impossible[13]. Nous avons donc décidé de tenter de faire cette assignation une première fois. Dans le cas où celle-ci se déroulerait sans problème, le reste de l'initialisation peut être exécutée. Dans le cas contraire, nous tentons dix fois d'exécuter l'assignation (en patientant un certain temps entre chaque tentative) et si, au terme de ces dix fois, le lien est toujours inexistant, nous envoyons un message d'erreur à l'utilisateur dans l'applet.

Lors de l'initialisation, les objets statiques du réseau ATM sont affichés dans la scène VRML. Il s'agit des machines hôtes, des commutateurs et des liens physiques entre eux. Pour ne pas surcharger la scène, nous avons décidé de créer un réseau contenant 4 machines hôtes et trois commutateurs. Dans la vue aérienne, les machines hôtes sont identifiées par des lettres et les commutateurs par des chiffres. Le lecteur se fera une idée de la vue aérienne du réseau créée par le simulateur grâce à la Figure 47.

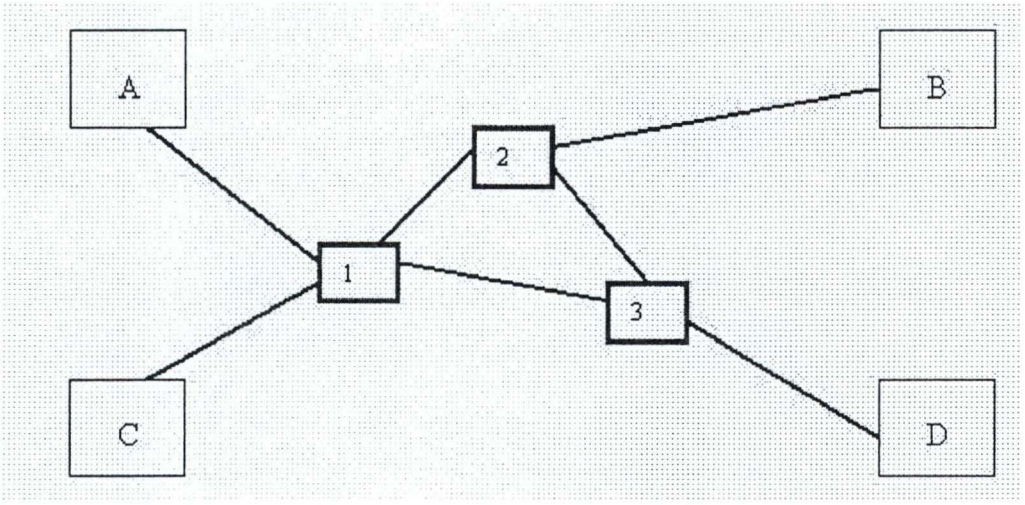


Figure 47 : La vue aérienne du réseau ATM dans la scène VRML

De plus, pour illustrer les différents concepts que prévoit le cahier des charges, nous avons choisi de ne faire circuler les cellules que dans un seul sens. Nous pouvons donner plusieurs explications à cela :

- Nous n'utilisons que le nombre minimal de cellules nécessaire à une bonne compréhension du concept;
- Nous ne faisons pas circuler plusieurs cellules dans des sens différents simultanément pour ne pas surcharger le processeur de l'ordinateur;
- Nous pensons que voir toujours le même type d'animation permet à l'utilisateur de mieux comparer et recouper les différents concepts qu'il étudie. Il peut ainsi mettre les concepts en parallèle plus facilement.

[13] Cette erreur apparaît entre autre dans le cas où l'utilisateur tente de recharger la page HTML sur laquelle se trouve l'applet et la scène VRML en utilisant le bouton ***recharger*** de son navigateur Internet.

Les méthodes propres à l'exécution des différentes animations seront déclarées dans une dernière partie de l'applet. Chacune de ces méthodes pourra être appelée dans la méthode *run( )*. Elles permettent en général de lancer l'animation liée à un concept choisi dans la scène VRML et décrivent les actions qui se dérouleront dans l'applet en synchronisant ces actions avec les événements qui se déroulent dans la scène VRML. Les trois méthodes principales de cette partie sont :

- *public void RunCommutation( )* : qui sera appelée par la méthode *run( )* si l'utilisateur choisit de lancer l'animation relative au concept de commutation
- *public void RunDonnees( )* : qui sera appelée par la méthode *run( )* si l'utilisateur choisit de lancer l'animation relative au concept des unités de données
- *public void RunCouches( )* : qui sera appelée par la méthode *run( )* si l'utilisateur choisit de lancer l'animation relative au concept de couches

Cette dernière partie reprend également les méthodes privées utilisées par les trois méthodes que nous venons de décrire. Les particularités d'implémentation de ces trois méthodes seront décrites un peu plus tard dans cette section.

Pour la méthode *run( )*, nous avons procédé un peu différemment de la façon décrite dans le squelette général du chapitre consacré à l'étude du langage Java. Le méthode ne contient pas de boucle d'animation mais un "switch" vers une des méthodes illustrant un des trois concepts repris dans le cahier des charges. Ce sont ces méthodes qui contiennent chacune une boucle d'animation propre. La méthode *run( )* est donc très simple et s'écrit en Java de la façon suivante :

```
public void run( )
  {
        if(Passage==1)
        {
              Passage++;
              Pause=true;
              while(Pause)
              {
                    MonRéseau.AbandonCPU(50);
              }
        }
        switch(TypeSimu)
        {
              case 1:
                    RunCommutation();
                    break;

              case 2:
                    RunDonnees();
                    break;

              case 3:
                    RunCouches();
                    break;

              default:
                    box1.setState(true); //pour l'interface de
l'applet
                    RunCommutation();
                    break;
```

```
        }
    }
```

Comme le lecteur pourra le remarquer, par défaut, le simulateur lance l'animation du concept de commutation. La première partie détermine ce qui se passe lorsque l'utilisateur lance l'application pour la première fois. Afin que l'animation ne commence pas immédiatement, le simulateur se met automatiquement en pause et attend le signal de l'utilisateur pour lancer l'animation choisie (qui, par défaut, est celle de la commutation). Le simulateur tel qu'il se trouve avant tout lancement d'animation est représenté à la Figure 48.

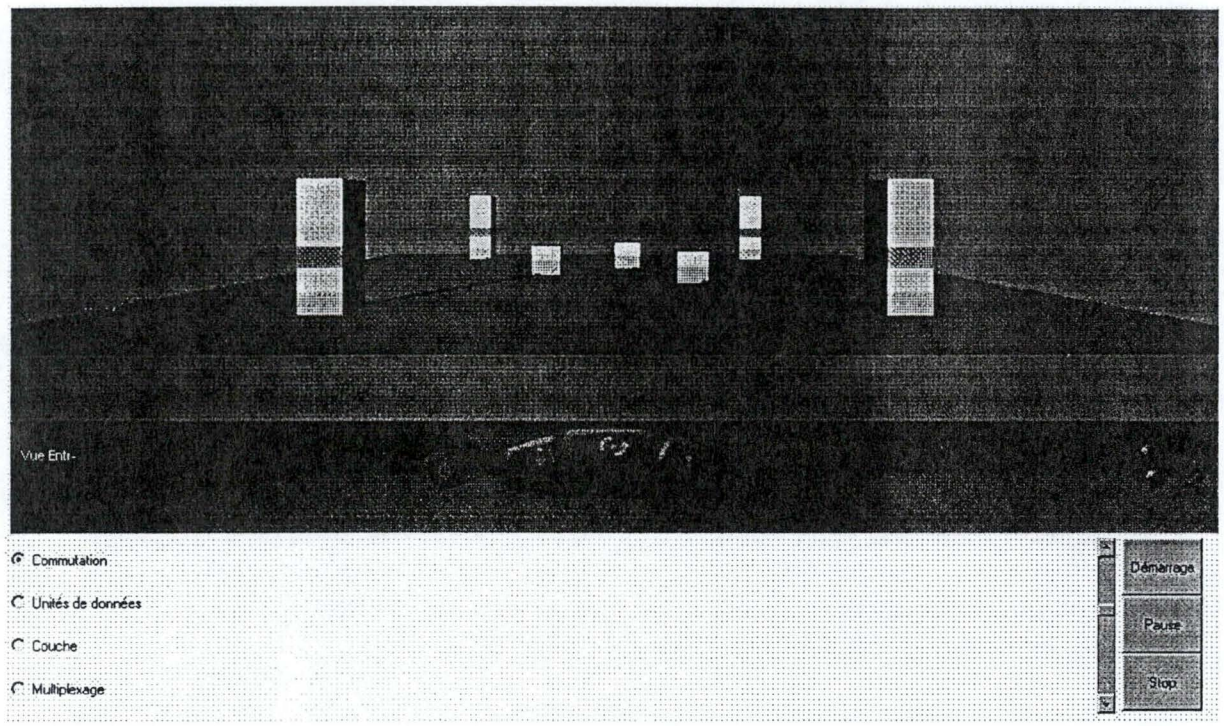


Figure 48 : Le simulateur lors de son lancement

Pour permettre l'affichage des informations dans l'applet Java, nous avons défini une zone tampon. A chaque appel de la méthode *repaint( )*, les parties de cette zone d'affichage qui auront subi des modifications seront réaffichées. C'est grâce à cette méthode que nous avons pu définir des animation fluides dans l'applet. Sans l'utilisation de cette zone tampon et de la méthode *repaint( )*, l'utilisateur aurait eu à supporter un clignotement de toutes les informations affichées dans l'applet Java, ce qui se révèle être très vite désagréable pour l'œil humain. Dès lors, aussi tôt que l'on fait subir une modification à la zone d'affichage de l'applet, il faut également invoquer la méthode *repaint( )* pour que ces changements deviennent effectifs.

II.5 Les particularités de la méthode *RunCommutation( )*

Dans cette section, nous montrons au lecteur les particularités de la méthode *RunCommutation( )*. Elle permet tout d'abord de démarrer l'animation dans la scène VRML. Ensuite, la méthode va afficher des informations complémentaires dans l'applet Java tout en synchronisant ces dernières avec l'action qui se déroule dans le monde en trois dimensions.

Cette méthode va tout d'abord changer le point de vue proposé à l'utilisateur dans la scène VRML. En effet, pour peu que l'utilisateur ait utilisé les possibilités de navigation offertes

par le tableau de bord de la scène VRML, la vision qu'il a du réseau n'est pas forcément la meilleure pour comprendre le concept qu'il veut animer. Il s'agit ici d'illustrer le concept de commutation. Le cahier des charges a prévu dans ce cas de présenter l'animation sous une vue aérienne. Dès que l'utilisateur choisit d'animer le concept de commutation, il pourra observer la scène d'un point de vue aérien. Bien sur, il est toujours libre de naviguer comme il l'entend à travers le monde en trois dimensions en utilisant le tableau de bord. Cette vue n'est qu'une "proposition" que lui fait le simulateur.

Ensuite, la méthode va créer divers objets du réseau ATM et notamment deux liaisons VCC partant et ayant pour destination deux machines hôtes distinctes mais passant par une liaison physique commune. Sur chacune de ces liaisons va être envoyée une cellule. Nous avons limité le nombre de cellules circulant à travers le réseau ATM pour deux raisons :

- Le cahier des charges prévoit d'utiliser le nombre minimum de cellules pour permettre d'illustrer correctement le concept en cours d'animation. Dans le cas de la commutation, il s'agit de deux cellules issues de deux machines différentes;
- Le nombre de cellules en cours d'animation est un des facteurs qui joue sur le taux d'occupation du processeur de la machine. Plus nous avions tendance à animer de cellules, plus la charge du processeur était élevée. Pour éviter d'avoir des animations trop lentes et saccadées, nous avons préféré nous limiter au strict minimum conseillé dans le cahier des charges.

Pour pouvoir afficher le contenu des tables de commutation, nous devons rechercher les informations regroupées dans les objets Java représentant le réseau ATM (entre autres, dans les commutateurs). Ces informations sont stockées par la méthode dans des variables de travail, ce qui permet d'éviter d'avoir à les rechercher au moment où l'on veut les afficher. Toutes ces informations sont ensuite affichées dans des tableaux.

Les lignes des tableaux consultées par les commutateurs pour router une cellule particulière changeront de couleur au moment de la consultation, c'est à dire au moment où, dans la scène VRML, la cellule "entre" dans un commutateur. Pour synchroniser ce changement de couleur des lignes des tableaux de commutation avec l'animation du réseau ATM, nous testons l'endroit où se trouve la cellule. En effet, les objets *celluleATM* du module *Atm* héritent des attributs de la classe *trame*. Les objets de cette classe contiennent un attribut RéfLocalisation (un entier) qui vaudra 0 ou 1 si la cellule est dans une couche et qui vaudra 2 si la cellule est dans une liaison logique (c'est à dire quand la représentation graphique de la cellule se trouve sur la représentation graphique des liens physiques). On peut dès lors très facilement savoir à quel moment, dans l'animation, la cellule se trouve dans un commutateur (RéfLocalisation = 0 ou 1) et quand elle circule sur les liens physiques (RéfLocalisation = 2).

Changer la couleur des lignes consultées n'est pas un problème en soi, savoir dans quel tableau ce changement doit être effectué l'est. Pour y arriver, nous nous sommes basés sur la position qu'occupaient les trois commutateurs dans le repère de coordonnées du monde VRML. Une fois que la cellule est dans un commutateur, on analyse la position de la représentation graphique de celui-ci (grâce aux attributs des objets Java auxquels les commutateurs sont associés) et il est ainsi possible de connaître le commutateur visité et le tableau qui devra être modifié dans l'applet Java.

Le lecteur pourra se faire une idée de l'animation de ce concept grâce à la saisie d'écran présentée à la Figure 49.

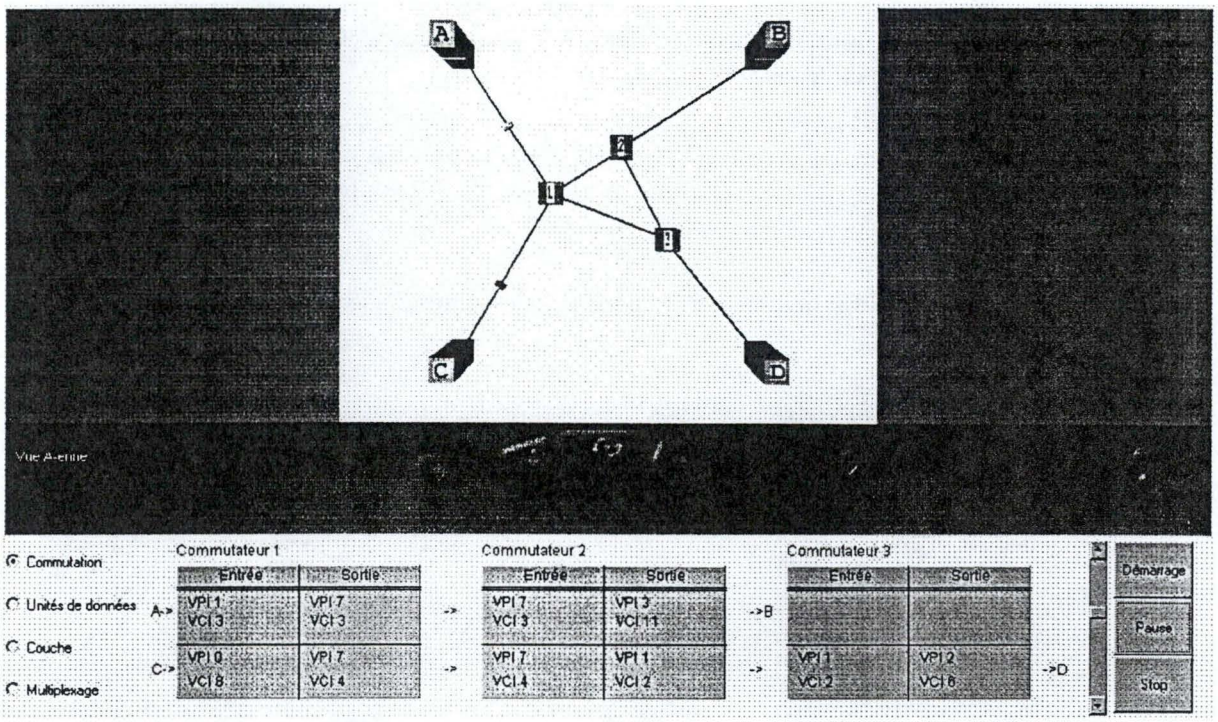


Figure 49 : L'écran pendant l'animation du concept de commutation

II.6 Les particularités de la méthode *RunDonnées( )*

Dans cette section, nous montrons au lecteur les particularités de la méthode *RunDonnées( )*. Tout comme la méthode *RunCommutation( )*, elle permet de lancer l'animation dans la scène VRML et d'afficher des informations complémentaires dans l'applet Java tout en les synchronisant avec l'action qui se déroule dans le monde en trois dimensions.

Tout comme dans la méthode précédente, la méthode *RunDonnées( )* va proposer un angle de vue à l'utilisateur de l'application. Ce dernier n'est naturellement pas tenu d'observer l'animation sous cet angle de vue particulier. Dans ce cas, l'angle proposé par le cahier des charges est la vue aérienne.

Ensuite, les objets particuliers du réseau ATM sont créés et l'animation est lancée. Comme le cahier des charges prévoit de ne montrer que des connexions permanentes entre les machines terminales, il est important de mettre en évidence cet aspect statique, ce qui peut se faire lors de l'illustration du concept d'unités de données. En effet, il est possible d'indiquer le chemin que suivra l'unité de donnée étudiée à travers le réseau ATM. Pour cela, il est important de trouver les informations à afficher et de les placer dans des variables temporaires avant de lancer l'animation.

Une fois l'animation démarrée, l'applet Java contiendra trois grands éléments. Ces éléments sont les suivants :

- Le chemin que suivra la cellule à travers le réseau ATM. Il s'agit dans ce cas d'une information statique qui ne sera jamais modifiée pendant toute la durée de l'animation du concept d'unités de données;

- La signification des abréviations reprises dans les différents champs de la cellule. Tout comme pour le chemin, cette information est statique et ne sera jamais modifiée pendant toute la durée de l'animation;
- La représentation de la cellule et de ses champs. Les champs VCI et VPI de cette dernière seront les seuls à être modifiés lors de l'animation. Pour mettre en évidence ce changement, ces deux champs sont affichés dans une couleur chaude comme tous les objets dynamiques de la scène VRML.

Une représentation de l'interface du simulateur(et donc de l'applet Java) pour ce concept est donnée à la Figure 50.

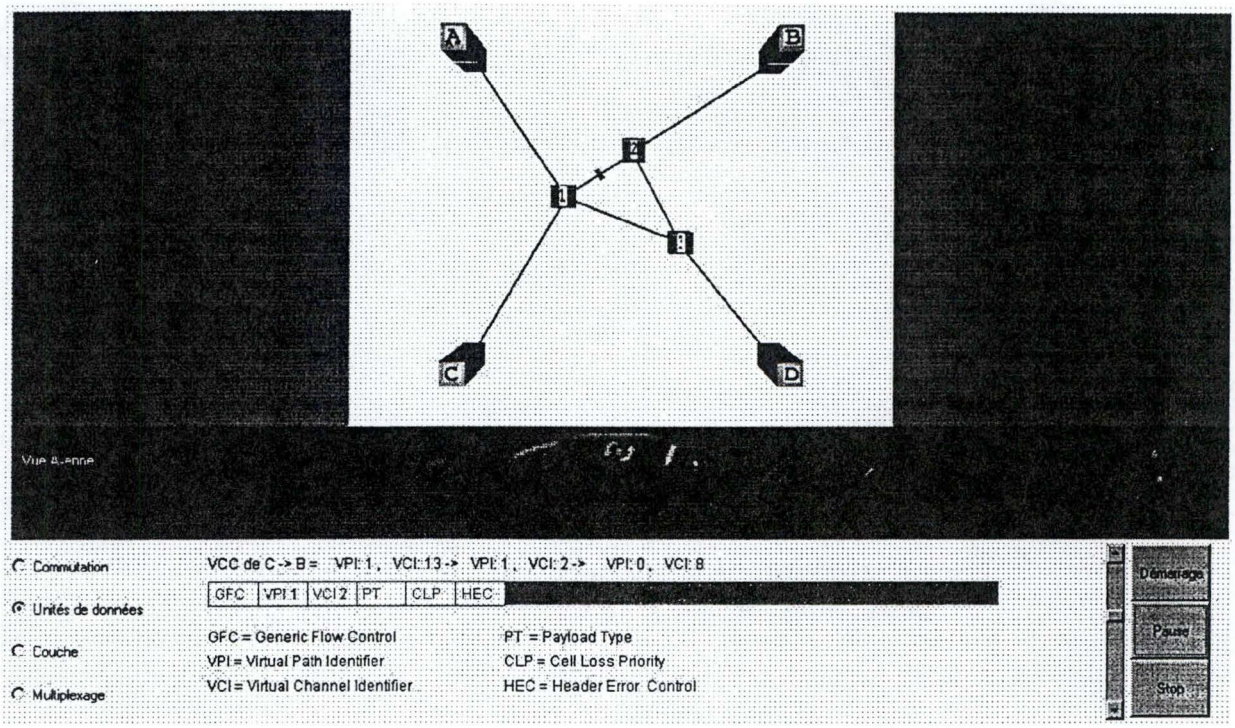


Figure 50 : L'écran pendant l'animation du concept d'unités de données

L'applet Java liée à cette animation est probablement la plus simple que nous avons eue à concevoir pour l'application. La seule difficulté, si on peut considérer cela comme une difficulté, résidait dans le besoin d'afficher la route suivie par la cellule à travers le réseau. En utilisant les attributs des objets *LiaisonVCC* et *CommutateurATM*, ces informations pouvaient être retrouvées selon les étapes suivantes :

- Prendre la référence de la liaison VCL créée entre la machine hôte source et le commutateur d'entrée dans le réseau ATM;
- Prendre la référence de la liaison VCL sortant du premier commutateur traversé par la cellule;
- Prendre la référence de la liaison VCL sortant du second commutateur traversé par la cellule;
- Afficher la suite de VPI, VCI qui apparaîtront dans les champs de la cellule, si la machine source est la machine C et la machine destination la machine B (choix arbitraire) en utilisant les attributs *GetVPI( )* et *GetNuméroLiaison( )* de l'objet *LiaisonVCL*.

Les trois premières étapes sont réalisées une seule fois après la création de la *LiaisonVCC* et après le lancement de l'animation. La dernière étape est réalisée pendant toute la durée de l'animation.

II.7 Les particularités de la méthode *RunCouches( )*

Dans cette section, nous montrons au lecteur les particularités de la méthode *RunCouches( )*. Tout comme les méthodes *RunCommutation( )* et *RunDonnées( )*, elle permet de lancer l'animation dans la scène VRML et d'afficher des informations complémentaires dans l'applet Java tout en synchronisant ces dernières avec l'action qui se déroule dans le monde en trois dimensions.

L'angle de vue proposé par la méthode est ici l'angle que nous avons appelé vue en hauteur dans le cahier des charges. En effet, cet angle est le plus approprié pour illustrer le concept de couche. Il permet de voir les machines terminales et les commutateurs comme un ensemble de couches et pas comme des monoblocs distincts (comme c'est le cas dans la vue aérienne). Cependant, comme dans les deux autres méthodes, l'utilisateur de l'application n'est pas tenu de se limiter au point de vue qui lui est proposé. Il peut toujours naviguer dans la scène VRML grâce au tableau de bord de cette dernière.

L'illustration de ce concept est, dans sa conception, assez simple. Il s'agit de changer la couleur de la couche en activité dans la scène VRML et d'afficher simultanément dans l'applet Java une brève description de ce que cette couche est en train de faire. Le lecteur pourra se faire une idée de l'interface de l'application grâce à la Figure 51. Cependant, suite aux restrictions que nous avons posées dans le cahier des charges, réaliser cette animation posait quelques problèmes.

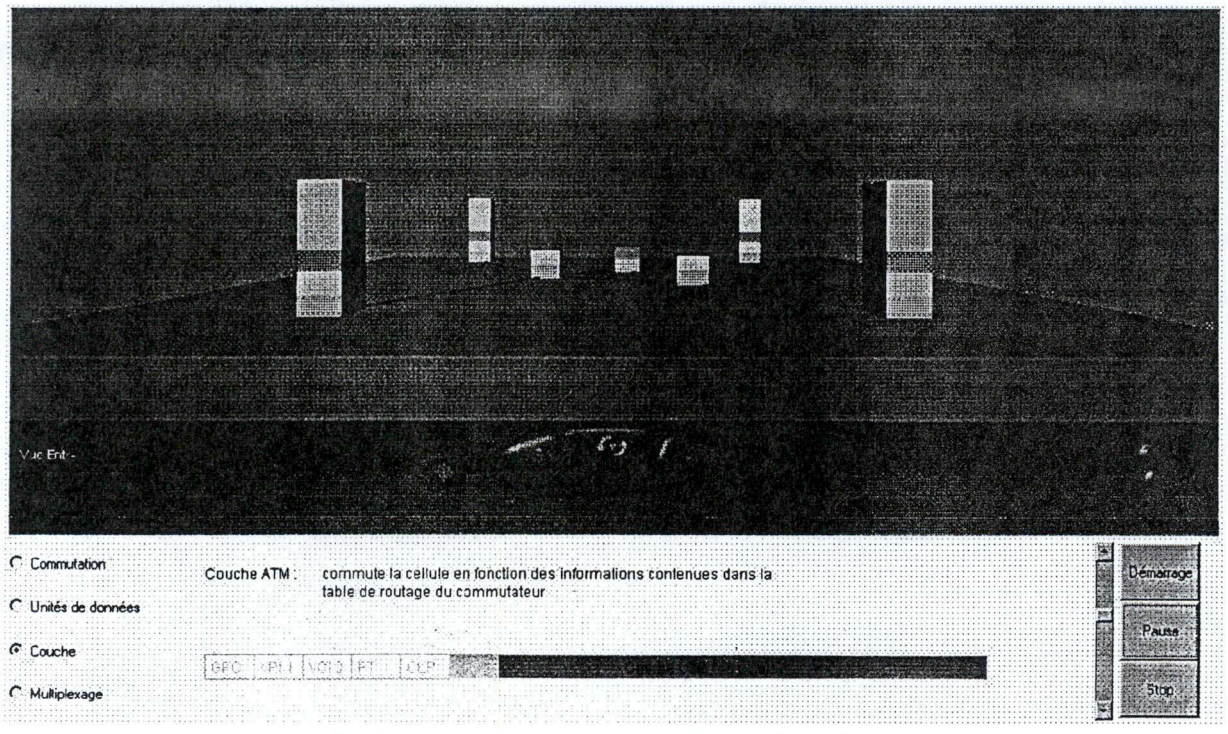


Figure 51 : L'écran pendant l'animation du concept de couches

Tout d'abord les deux seules couches que nous avions développées de manière approfondie étaient les couches physiques et ATM. Il fallait donc trouver un moyen d'animer également les couches AAL et supérieures.

L'animation de ces couches particulières devait se faire en dehors de la boucle principale. Comme les couches AAL et supérieures n'apparaissent que dans les machines terminales, nous devions les animer avant d'entrer dans la boucle ainsi qu'à la sortie de la boucle. A ces deux moments précis, nous changeons la couleur de la représentation graphique de la couche "visitée" et nous affichons dans l'applet Java les informations la concernant. C'est à dire, pour les couches supérieures (que nous appelons couches d'application) et avant d'entrer dans la boucle, l'applet Java précise tout simplement que ces couches ont des données à envoyer. Pour la couche AAL, l'applet Java précise que cette couche segmente et adapte les flux d'information en fonction de la classe de service. On peut voir alors apparaître dans l'applet le champ "charge utile" de 48 octets de la cellule, et ce dans la même couleur que la couche AAL. A la sortie de la boucle, nous recommençons cette animation mais les informations qui apparaissent dans l'applet Java changent en fonction du fait que les couches se trouvent dans la machine destination et plus dans la machine source. Pour la couche AAL, l'information affichée précise que cette couche met en œuvre le processus de réassamblage des données et pour la couche application l'information affichée mentionne que cette couche reçoit les données envoyées.

Dans la boucle principale, l'animation est basée sur l'étude de la cellule et de la couche que celle-ci traverse. Les couches animées à l'intérieur de cette boucle sont uniquement les couches physiques et ATM, ce qui traduit bien les restrictions que nous avons posées dans le cahier des charges. Pour savoir où se trouve la cellule, nous analysons son attribut RéfLocalisation. Si ce dernier vaut 2, cela signifie que la cellule se trouve sur un lien physique. Nous vérifions alors si la couche que la cellule vient de quitter a retrouvé sa couleur d'origine. Si c'est le cas, on ne fait rien si ce n'est attendre que la cellule pénètre dans la couche physique d'une machine. Sinon on restitue à cette couche visitée sa couleur d'origine et on change la valeur de la variable qui nous avait permis de faire cette analyse pour ne pas effectuer le travail plusieurs fois. Dans le cas où l'attribut RéfLocalisation est différent de 2, cela signifie que la cellule se trouve dans une couche. A ce moment on regarde si nous avons déjà effectué le changement de couleur de la couche. Si ce n'est pas fait, on change la couleur de la couche, on affiche dans l'applet Java l'information propre à la couche visitée et on met à jour certaines variables. Si le changement de couleur a déjà été fait, on ne change rien et on attend patiemment que la cellule quitte cette couche.

III Interfaces des classes d'objets

Il est important, pour assurer un changement facile et peu preneur de temps de l'interface de l'application, d'associer aux objets des modules *Réseau* et *Atm* qui ont une représentation dans un monde réel des représentation graphiques. Toutes ces représentations graphiques sont elles aussi des classes d'objets décrites dans le module de *Présentation.* Nous allons donc à présent présenter au lecteur les interfaces de ces classes d'objets.

III.1 Module *Présentation*

Dans cette partie, nous décrivons les classes d'objets liées au module *Présentation* de l'application. Ce module permet d'associer une interface particulière à l'application. Tout comme dans la section décrivant les interfaces des modules *Réseau* et *Atm*, nous ne décrirons

dans ce mémoire que les méthodes et les attributs publiques de chaque classe (les codes sources étant disponible en annexe du mémoire). Nous présenterons les attributs et les interfaces des classes d'objets de la façon suivante :

1. Pour un attribut :
   <NomAttribut> : <TypeAttribut>
   <Commentaire>

2. Pour une méthode :
   <NomOpération>({*<NomParamètre > : <TypeParamètre>*}$^{+}$) : [*<TypeRésultat>*]

Les types *TypeAttributs, TypeParamètre et TypeRésultat* peuvent être des classes d'objets ou des types simples tels que ***int*** ou ***boolean***. Le type de résultat est optionnel pour une méthode. Le nombre de paramètres d'une méthode est compris entre 0 et n (ce qui est traduit ci-dessus par le signe "+"). Nous supposons que le lecteur possède une connaissance suffisante de la terminologie Java pour comprendre celle-ci. Les mots-clés Java utilisés dans cette partie du mémoire sont définis dans le chapitre consacré à ce langage.

Classe *AttributsFormes*
public class *AttributsFormes*

La classe *AttributsFormes* reprend toutes les caractéristiques communes aux formes géométriques simples qui peuvent apparaître dans la scène VRML.

Attributs

- *AxeX : **float***
  une position sur l'axe des X dans le repère à trois dimensions de la scène VRML
- *AxeY : **float***
  une position sur l'axe des Y dans le repère à trois dimensions de la scène VRML
- *AxeZ : **float***
  une position sur l'axe des Z dans le repère à trois dimensions de la scène VRML
- *Couleur : **Color***
  la couleur d'un objet dans la scène VRML
- *RotationAxeX : **int***
  un entier égal à 0 dans le cas où l'objet décrit dans la scène VRML ne subit aucune rotation sur l'axe des X, égal à 1 sinon. Les trois attributs *RotationAxeX, RotationAxeY* ou *RotationAxeZ* ne peuvent être égaux à 1 simultanément, ils sont soit tous égaux à 0 soit il y en a un qui vaut 1 et les deux autres valent 0
- *RotationAxeY : **int***
  un entier égal à 0 dans le cas où l'objet décrit dans la scène VRML ne subit aucune rotation sur l'axe des X, égal à 1 sinon. Les trois attributs *RotationAxeX, RotationAxeY* ou *RotationAxeZ* ne peuvent être égaux à 1 simultanément, ils sont soit tous égaux à 0 soit il y en a un qui vaut 1 et les deux autres valent 0
- *RotationAxeZ : **int***
  un entier égal à 0 dans le cas où l'objet décrit dans la scène VRML ne subit aucune rotation sur l'axe des X, égal à 1 sinon. Les trois attributs *RotationAxeX, RotationAxeY* ou *RotationAxeZ* ne peuvent être égaux à 1 simultanément, ils sont soit tous égaux à 0 soit il y en a un qui vaut 1 et les deux autres valent 0

- *Inclinaison : **float***
  le degré de rotation subi par l'objet dans la scène VRML si un des trois attributs *RotationAxeX*, *RotationAxeY* ou *RotationAxeZ* vaut 1.

Méthodes

- *SetAxeX (AxeX : **float**) : -*
  permet d'attribuer la valeur passée en paramètre à l'attribut *AxeX*
- *SetAxeY (AxeY : **float**) : -*
  permet d'attribuer la valeur passée en paramètre à l'attribut *AxeY*
- *SetAxeZ (AxeZ : **float**) : -*
  permet d'attribuer la valeur passée en paramètre à l'attribut *AxeZ*
- *SetCouleur (Couleur : **Color**) : -*
  permet d'attribuer la valeur passée en paramètre à l'attribut *Couleur*
- *SetRotationAxeX (RotationAxeX : **int**) : -*
  permet d'attribuer la valeur passée en paramètre à l'attribut *RotationAxeX* ; la valeur passée en paramètre doit être égale à 0 ou à 1
- *SetRotationAxeY (RotationAxeY : **int**) : -*
  permet d'attribuer la valeur passée en paramètre à l'attribut *RotationAxe* ; la valeur passée en paramètre doit être égale à 0 ou à 1
- *SetRotationAxeZ (RotationAxeZ : **int**) : -*
  permet d'attribuer la valeur passée en paramètre à l'attribut *RotationAxeZ* ; la valeur passée en paramètre doit être égale à 0 ou à 1
- *SetInclinaison (Inclinaison : **float**) : -*
  permet d'attribuer la valeur passée en paramètre (exprimée en radians) à l'attribut *Inclinaison*
- *GetAxeX ( ) : **float***
  renvoie la position de l'objet sur l'axe des X dans la scène VRML, c'est à dire renvoie l'attribut *AxeX*
- *GetAxeY ( ) : **float***
  renvoie la position de l'objet sur l'axe des Y dans la scène VRML, c'est à dire renvoie l'attribut *AxeY*
- *GetAxeZ ( ) : **float***
  renvoie la position de l'objet sur l'axe des Z dans la scène VRML, c'est à dire renvoie l'attribut *AxeZ*
- *GetRouge ( ) : **float***
  renvoie le pourcentage de couleur rouge contenu dans la couleur de l'objet dans la scène VRML
- *GetVert ( ) : **float***
  renvoie le pourcentage de couleur verte contenu dans la couleur de l'objet dans la scène VRML
- *GetBleu ( ) : **float***
  renvoie le pourcentage de couleur bleue contenu dans la couleur de l'objet dans la scène VRML
- *GetCouleur ( ) : **Color***
  renvoie la couleur de l'objet dans la scène VRML
- *GetRotationAxeX ( ) : **int***
  renvoie 1 si l'objet dans la scène VRML a subi une rotation sur l'axe des X, 0 sinon

- *GetRotationAxeY ( ) : **int***
  renvoie 1 si l'objet dans la scène VRML a subi une rotation sur l'axe des Y, 0 sinon
- *GetRotationAxeZ ( ) : **int***
  renvoie 1 si l'objet dans la scène VRML a subi une rotation sur l'axe des Z, 0 sinon
- *GetInclinaison ( ) : **float***
  renvoie l'angle de rotation de l'objet dans la scène VRML exprimé en radians. Si aucun des attributs *RotationAxeX*, *RotationAxeY* ou *RotationAxeZ* ne vaut 1, alors par défaut l'angle de rotation (et donc l'attribut *Inclinaison*) vaut 0.0

Classe *AttributsBoite*
***public class** AttributsBoite*

La classe *AttributsBoite* reprend les caractéristiques particulières des boîtes de la scène VRML (les nœuds ***Box*** du monde VRML). En plus des attributs et méthodes décrits ici, cette classe hérite de tous les attributs et de toutes les méthodes définies dans la classe *AttributsFormes*, ce qui est normal puisque les boîtes sont des formes géométriques simples.

Attributs

- *TailleX : **float***
  la longueur de la boîte sur l'axe des X du repère de coordonnées de la scène VRML
- *TailleY : **float***
  la longueur de la boîte sur l'axe des Y du repère de coordonnées de la scène VRML
- *TailleZ : **float***
  la longueur de la boîte sur l'axe des Z du repère de coordonnées de la scène VRML

Méthodes

- *SetTailleX (TailleX : **float**) : -*
  permet d'attribuer la valeur passée en paramètre à l'attribut *TailleX*
- *SetTailleY (TailleX : **float**) : -*
  permet d'attribuer la valeur passée en paramètre à l'attribut *TailleY*
- *SetTailleZ (TailleX : **float**) : -*
  permet d'attribuer la valeur passée en paramètre à l'attribut *TailleZ*
- *GetTailleX ( ) : **float***
  renvoie la valeur de l'attribut *TailleX*
- *GetTailleY ( ) : **float***
  renvoie la valeur de l'attribut *TailleY*
- *GetTailleZ ( ) : **float***
  renvoie la valeur de l'attribut *TailleZ*

Classe *AttributsLiens*
***public class** AttributsLiens*

La classe *AttributsLiens* reprend les caractéristiques particulières des cylindres de la scène VRML (les nœuds ***Cylinder*** du monde VRML qui permettent de représenter les liens physiques entre les machines). En plus des attributs et méthodes décrits ici, cette classe hérite de tous les attributs et de toutes les méthodes définies dans la classe *AttributsFormes*, ce qui est normal puisque les cylindres sont des formes géométriques simples.

Attributs

- *Rayon : **float***
  le rayon du cylindre décrit dans la scène VRML
- *Longueur : **float***
  la longueur du cylindre décrit dans la scène VRML

Méthodes

- *SetRayon (Rayon : **float**) : -*
  permet d'attribuer à l'attribut *Rayon* la valeur passée en paramètre
- *SetLongueur (Longueur : **float**) : -*
  permet d'attribuer à l'attribut *Longueur* la valeur passée en paramètre
- *GetRayon ( ) : **float***
  renvoie la valeur de l'attribut *Rayon*
- *GetLongueur ( ) : **float***
  renvoie la valeur de l'attribut *Longueur*

Classe *P_Couche*
***public class** P_couche*

La classe *P_Couche* définit l'objet graphique qui sera associé aux objets de type *couche* du module *Réseau*.

Attributs

- *RéfCouche : couche*
  une référence vers la couche liée à l'objet graphique courant
- *NoeudVRML : **Node[ ]***
  l'objet VRML associé à la représentation graphique de la couche dans la scène VRML
- *Ajout : **EventInMFNode***
  une variable associée au nœud du monde VRML qui permet d'ajouter dynamiquement des objets dans la scène VRML
- *Suppr : **EventInMFNode***
  une variable associée au nœud du monde VRML qui permet de supprimer dynamiquement des objets dans la scène VRML
- *browser : **Browser***
  une variable associée à l'instance du browser VRML utilisé et qui permet donc de faire le lien entre les objets Java et la scène VRML
- *RéfAttributs : AttributsBoite*
  une référence vers les caractéristiques de la représentation graphique de la couche dans la scène VRML
- *AncienneCouleur : **Color***
  une variable qui contient toujours la couleur d'origine de la représentation graphique de la couche dans la scène VRML pour le cas où cette dernière serait modifiée

Méthodes

- *DessineToi ( ) : -*
  dessine la représentation graphique de la couche dans la scène VRML

- *ChangeCouleurCouche (RéfCouche : couche, Couleur : **Color**) : -*
  change la couleur de la couche référencée pour la couleur passée en paramètre
- *SetRéfCouche (RéfCouche : couche) : -*
  permet de faire le lien entre une couche donnée et son objet graphique dans la scène VRML
- *SetRéfAttributs (RéfAttributs : AttributsBoite) : -*
  attribue une valeur aux caractéristiques de la représentation graphique de la couche dans la scène VRML
- *GetRéfCouche ( ) : couche*
  renvoie la couche associée à l'objet graphique courant
- *GetRéfAttributs ( ) : AttributsBoite*
  renvoie les caractéristiques de la représentation graphique de la couche
- *GetAncienneCouleur ( ) : **Color***
  renvoie la couleur d'origine de la couche

Classe *P_Equipement*
***public abstract class** P_Equipement*

La classe *P_Equipement* définit les caractéristiques générales de la représentation graphique qui sera associée aux objets de type *equipement* du module *Réseau*. Elle est abstraite, ce qui implique que certaines de ces méthodes devront être redéfinies dans ses sous-classes.

Attributs

- *RéfEquipement : equipement*
  une référence vers l'équipement associé à l'objet graphique courant
- *PositionEquipement : **float[ ]***
  un tableau de trois réels contenant la position de la représentation graphique de l'équipement dans le repère de coordonnées de la scène VRML

Méthodes

- *SetRéfEquipement (RéfEquipement : equipement) : -*
  permet d'associer un équipement et sa représentation graphique
- *SetPositionX (PositionX : **float**) : -*
  assigne au premier élément du tableau de l'attribut *PositionX,* la valeur passée en paramètre, c'est à dire donne une position sur l'axe des X à la représentation graphique de l'équipement dans le repère de coordonnées de la scène VRML
- *SetPositionY (PositionY : **float**) : -*
  assigne au deuxième élément du tableau de l'attribut *PositionY,* la valeur passée en paramètre, c'est à dire donne une position sur l'axe des Y à la représentation graphique de l'équipement dans le repère de coordonnées de la scène VRML
- *SetPositionZ (PositionZ : **float**) : -*
  assigne au troisième élément du tableau de l'attribut *PositionZ,* la valeur passée en paramètre, c'est à dire donne une position sur l'axe des Z à la représentation graphique de l'équipement dans le repère de coordonnées de la scène VRML
- *SetPositionEquipement (PositionEquipement : **float [ ]**) : -*
  assigne à l'attribut *PositionX* la valeur passée en paramètre, c'est à dire donne une position à la représentation graphique de l'équipement dans le repère de coordonnées de la scène VRML

- *GetRéfEquipement ( ) : equipement*
  renvoie l'équipement associé à la représentation graphique courante
- *GetPositionX ( ) : **float***
  renvoie la position de la représentation graphique de l'équipement sur l'axe des X dans le repère de coordonnées de la scène VRML
- *GetPositionY ( ) : **float***
  renvoie la position de la représentation graphique de l'équipement sur l'axe des Y dans le repère de coordonnées de la scène VRML
- *GetPositionZ ( ) : **float***
  renvoie la position de la représentation graphique de l'équipement sur l'axe des Z dans le repère de coordonnées de la scène VRML
- *GetPositionEquipement : **float [ ]***
  renvoie la position de la représentation graphique de l'équipement dans le repère de coordonnées de la scène VRML
- *DessineToi ( ) : -*
  est une méthode abstraite qui devra être redéfinie dans les sous-classes de la classe *P_equipement* et qui dessine l'équipement dans la scène VRML

Classe *P_commutateur ATM*
***public class** P_commutateurATM*

La classe *P_commutateur ATM* définit la représentation graphique des objets de type *commutateurATM* du module *Atm*. En plus des attributs et méthodes décrits ici, cette classe hérite de tous les attributs et de toutes les méthodes définies dans la classe *P_equipement*

Attributs

Méthodes

- *DessineToi ( ) : -*
  dessine le commutateur ATM dans la scène VRML

Classe *P_machineHote*
***public class** P_machineHote*

La classe *P_machineHote* définit la représentation graphique des objets *machineHote* du module *Atm*. En plus des attributs et méthodes décrits ici, cette classe hérite de tous les attributs et de toutes les méthodes définies dans la classe *P_equipement*

Attributs

Méthodes

- *DessineToi ( ) : -*
  dessine les machines terminales dans la scène VRML

Classe *P_trame*
***public class** P_trame*

La classe *P_trame* définit l'objet graphique qui sera associé aux objets de type *trame* du module *Réseau*.

Attributs

- *RéfTrame : trame*
  une référence vers l'objet *sousReseau* associé à la représentation graphique courante
- *NoeudActuel : **Node [ ]***
  l'objet du monde VRML qui représente actuellement la cellule ATM dans la scène VRML
- *NoeudPrécédent : **Node [ ]***
  l'objet du monde VRML qui représente la cellule ATM dans la scène VRML avant son déplacement d'un pas sur la liaison physique
- *Animation : **int***
  un entier qui permet de savoir si la représentation graphique de la cellule est déjà en mouvement dans la scène VRML
- *RéfAttributs : AttributsBoite*
  une référence vers les caractéristiques de la représentation graphique de la cellule ATM

Méthodes

- *MiseAJour ( ) : -*
  permet d'anticiper le fait que le nœud représentant la cellule ATM dans le monde VRML va changer
- *MiseAJourAnimation ( ) : -*
  permet d'augmenter de 1 l'attribut *Animation* une fois que la représentation graphique de la cellule ATM s'anime dans la scène VRML
- *SetRéfAttributs (RéfAttributs : AttributsBoite) : -*
  permet d'attribuer les attributs passés en paramètres aux caractéristiques de la représentation graphique de la cellule ATM
- *GetRéfAttributs ( ) : AttributsBoite*
  renvoie les caractéristiques de la représentation graphique de la cellule ATM

Classe *P_sousReseau*
***public class** P_sousReseau*

La classe *P_sousReseau* définit les caractéristiques générales de la représentation graphique qui sera associée aux objets de type *sousRéseau* du module *Réseau.* Elle est abstraite, ce qui implique que certaines de ces méthodes devront être redéfinies dans ses sous-classes.

Attributs

- *RéfSousRéseau : sousReseau*
  une référence vers l'objet *sousReseau* associé à la représentation graphique courante
- *Ajout : **EventInMFNode***
  une variable qui permet d'ajouter un élément dans la scène VRML
- *Suppr : **EventInMFNode***
  une variable qui permet de supprimer un élément dans la scène VRML
- *AjoutCell : **EventInMFNode***
  une variable qui permet d'ajouter la représentation d'une cellule dans la scène VRML
- *SupprCell : **EventInMFNode***
  une variable qui permet de supprimer la représentation d'une cellule dans la scène VRML

- *browser : **Browser***
  une variable associée à l'instance du browser VRML utilisé, et qui permet donc de faire le lien entre les objets Java et la scène VRML

Méthodes

- *DessineToi ( ) : -*
  dessine le réseau dans la scène VRML
- *AdapterGraphique ( ) : -*
  permet d'associer à la représentation graphique des équipements la représentation graphique des couches qui les composent. Cette méthode est abstraite et devra donc être redéfinie dans toute sous-classe de la classe *P_sousReseau*
- *SetRéfSousRéseau (RéfSousRéseau : sousRéseau) : -*
  permet d'associer à une représentation graphique l'objet *sousReseau* qu'elle représente
- *GetRéfSousRéseau ( ) : sousRéseau*
  renvoie l'objet *sousReseau* associé à la représentation graphique courante
- *GetAjout ( ) : **EventInMFNode***
  renvoie la variable permettant d'ajouter un élément dans la scène VRML
- *GetSuppr ( ) : **EventInMFNode***
  renvoie la variable permettant de supprimer un élément de la scène VRML
- *GetAjoutCell ( ) : **EventInMFNode***
  renvoie la variable permettant d'ajouter à la scène VRML la représentation graphique d'une cellule ATM
- *GetSupprCell ( ) : **EventInMFNode***
  renvoie la variable permettant de supprimer de la scène VRML la représentation graphique d'une cellule
- *GetBrowser : **Browser***
  renvoie la variable contenant l'instance du browser VRML utilisé

Classe *P_sousReseauATM*
***public class** P_sousReseauATM*

La classe *P_sousReseauATM* définit les caractéristiques de la présentation graphique qui sera associée aux objets *sousReseauATM* du module *Atm.* En plus des attributs et méthodes décrits ici, cette classe hérite de tous les attributs et de toutes les méthodes définies dans la classe *P_sousReseau*

Attributs

Méthodes

- *TrameGraphique (Trame : celluleATM) : -*
  crée l'objet graphique associé à la cellule véhiculée dans le réseau et passée en paramètre. Cette méthode associe également une couleur chaude à la représentation graphique de la cellule
- *AdapterGraphique ( ) : -*
  permet d'attribuer à toute représentation graphique d'un équipement du réseau ATM la représentation des couches qui le composent
- *DessineToi ( ) : -*
  dessine le réseau ATM

Classe *P_liaisonPHY*
public class *P_liaisonPHY*

La classe *P_liaisonPHY* définit les caractéristiques de la représentation graphique des liens physiques associée aux objets *liaisonPHY* du module *Atm.*

Attributs

- *RéfLiaisonPhy : liaisonPHY*
  une référence vers l'objet *liaisonPHY* associé à l'objet graphique courant
- *NoeudVRML :* ***Node [ ]***
  l'objet VRML représentant la liaison physique dans le monde VRML
- *Ajout :* ***EventInMFNode***
  une variable qui permet d'ajouter un élément dans la scène VRML
- *AjoutCell :* ***EventInMFNode***
  une variable qui permet d'ajouter la représentation graphique d'une cellule ATM dans la scène VRML
- *SupprCell :* ***EventInMFNode***
  une variable qui permet de supprimer la représentation graphique d'une cellule ATM dans la scène VRML
- *Browser :* ***Browser***
  une variable associée à l'instance du browser VRML utilisé, et qui permet donc de faire le lien entre les objets Java et la scène VRML
- *RéfAttributs : AttributsLiens*
  une référence vers les caractéristiques de la représentation graphique de la liaison physique courante dans la scène VRML
- *ListeAnimables :* ***Vector***
  un tableau contenant les objets VRML associés aux cellules animées dans la scène VRML

Méthodes

- *DessineToi ( ) : -*
  dessine la représentation graphique de la liaison physique dans la scène VRML
- *AnimationTrame ( ) : -*
  permet d'animer, dans la scène VRML, la représentation graphique des cellules qui se trouvent sur la liaison physique courante

IV. Conclusion

Pour la création de la couche de présentation, nous avons dû faire face à de grands problèmes de non standardisation. La plupart des erreurs qui pourraient apparaître avant l'exécution de l'animation ou lors d'actions sur le navigateur Internet sont dues à ce manque de standardisation.

A l'heure de la rédaction de ce mémoire, les efforts du consortium WWW et du VAG devraient être terminés pour assurer la standardisation des éléments qui ne le sont pas encore. Le langage VRML 97 devrait exister et les étiquettes <APPLET></APPLET> et <EMBED></EMBED> devraient être interprétées de façon identique par tous les navigateurs.

Cependant, nous ne développerons pas dans ce mémoire les nouvelles normes standardisées afin de mettre en évidence les causes des faiblesses de notre application.

La couche de présentation termine cette partie consacrée à l'analyse technique. La partie suivante sera, elle, consacrée à l'évaluation de ce dernier.

Partie 4 - Evaluation

Chapitre X : Evaluation et critiques

I. Introduction

Ce chapitre tente de prendre du recul par rapport à tout ce qui a été dit précédemment dans ce mémoire. Nous posons ici un diagnostic de l'application et du projet dans lequel elle fut conçue.

Nous présenterons d'abord une évaluation personnelle du projet P2-Verso auquel nous avons participé et pour lequel nous avons développé notre application. Nous pensons que faire une critique personnelle du projet est indispensable afin que le lecteur prenne conscience de l'impact que ce dernier a eu sur notre travail.

Nous passerons ensuite à une évaluation de l'application elle même. Pour cela nous avons demandé à diverses personnes d'observer le fonctionnement de l'application et de la critiquer. Il s'agit tout d'abord d'étudiants de notre cours qui ont la même formation en matière de télécommunications que la nôtre. Nous avons également demandé à un spécialiste IHM de nous donner son avis sur l'application. Enfin, nous prendrons du recul pour réaliser nous même une analyse du travail que nous avons fourni.

Dans l'avant-dernière partie de ce chapitre, nous montrerons au lecteur les nombreuses possibilités d'extension de notre simulateur celles que nous avons déjà envisagées lors de la conception et celles qui nous sont apparues intéressantes par la suite.

Nous terminerons par une très brève conclusion.

II. Le projet P2-Verso

Nous aimerions profiter de ce chapitre consacré aux tests et aux évaluations pour donner au lecteur nos impressions sur le projet P2-Verso. Pourquoi? L'application que nous avons conçue est bien différente de ce que nous pensions faire en commençant notre stage au CEGELY. C'est suite aux objectifs du projet que notre maître de stage nous a proposé d'intégrer le groupe de Lyon et de développer le simulateur. Le projet P2-Verso a donc eu de nombreux impacts sur notre stage et sur notre travail.

Comme nous l'avons déjà mentionné plus haut, le projet P2-Verso s'inscrit dans la continuité d'un premier projet appelé Franco-Réso. La plupart des différents acteurs formant le groupe de travail du projet P2-Verso se connaissaient et avaient déjà eu l'opportunité de collaborer. Il nous a donc fallu nous intégrer à un groupe général assez fermé. Cette intégration se fit brutalement sans aucune préparation. Les premières discussions qui eurent lieu étaient basées sur les défauts remarqués dans le projet Franco-Réso. Elles nous étaient dès lors difficiles d'accès car nous ne pouvions y prendre une part aussi active que les autres membres du groupe.

Tout au long de la durée du projet, nous nous sommes sentis peu concernés par les différentes vidéoconférences qui pouvaient avoir lieu. Les seuls moments qui nous semblaient vraiment intéressants étaient ceux concernant le module de simulation sur lequel nous travaillions.

Mais ces moments n'étaient jamais de longue durée et se perdaient souvent dans une masse d'informations inutiles pour notre travail.

Les discussions tenues entre les équipes de Lyon, Montréal, Lannion et Paris échappaient donc à notre intérêt la plupart du temps. Nous ne pouvions alors n'avoir qu'une connaissance superficielle du projet. C'est par la suite, et en particulier lors de la rédaction de ce mémoire, que nous avons appris le plus de choses relatives à ce projet.

La période de notre stage ne coïncidant pas pleinement avec celle du projet, nous avons dû anticiper beaucoup d'éléments pour parvenir à concevoir complètement l'application. Cette anticipation eu pour conséquence que nous travaillions de manière isolée. Les moments de tests et de démonstration étaient rares et ce n'est que quelques jours avant notre départ que certains acteurs ont réalisé qu'ils n'avaient pas encore critiqué le cahier des charges. Dès le premier mois de notre arrivée, nous avons réalisé qu'il serait impossible de terminer l'application si nous n'anticipions pas le projet. Ce manque de temps par rapport au travail à fournir n'a pas toujours été pris en compte par les membres du projet et il fallait parfois le leur rappeler pour ne pas voir tomber les restrictions que nous avions posées dans le cahier des charges.

Cependant, le projet nous a permis de rencontrer des gens et de découvrir d'autres cultures et mentalités. Nous avons aussi très vite pris conscience des difficultés que pouvait engendrer un travail de collaboration de cette envergure. Nous avons pu ainsi observer un côté pratique de la gestion de grands projets que nous ne connaissions pas. Nous avons découvert tous les bénéfices d'une bonne coordination des différentes équipes et le besoin capital d'avoir un leader qui met en place et qui gère toute cette coordination. Nous avons découvert ce que serait probablement notre future vie active. L'envergure et le côté multi-culturel du projet P2-Verso valaient la peine que nous y prenions part.

III. L'application

Dans le projet P2-Verso, notre plus grand et pour ainsi dire seul centre d'intérêt fut le développement de notre application. Suite au travail fourni, il est parfois difficile d'arriver à prendre un recul suffisant pour critiquer objectivement le résultat. Nous avons donc demandé l'avis d'étudiants et d'un spécialiste IHM au sujet de notre application. Avant de leur faire une démonstration, nous leur avons expliqué dans quel cadre s'insérait le simulateur. Nous leur avons précisé que le simulateur permettait d'illustrer différents concepts de la technologie ATM et qu'il faisait partie d'un module de formation plus large traitant de cette technologie. Nous leur avons aussi expliqué en quelques mots ce qu'ils voyaient apparaître à l'écran. Dans cette section, nous allons essayer de reprendre toutes leurs critiques et suggestions.

III.1 L'avis des étudiants

En général, les réactions des étudiants auxquels nous avons présenté le simulateur furent positives. Tous ont trouvé l'interface simple et cohérente. La présentation des éléments du réseau ATM dans un environnement à trois dimensions les impressionnait beaucoup. Cependant, la manipulation du tableau de bord de la scène VRML est souvent complexe pour un novice et l'impression de ne plus savoir où on est, ni comment revenir au point de départ peut être frustrante.

En faisant un zoom sur les cellules circulant à travers le réseau, un étudiant nous a fait remarquer qu'elles ressemblaient à des wagons circulant sur des rails. Cette remarque nous étonna un peu. Lors de notre étude du langage VRML, nous avions pensé représenter des trains de cellules et donc utiliser la métaphore ferroviaire. Nous avons dû abandonner cette idée à cause du taux de charge du processeur qui atteignait de très grandes proportions. Des voies de chemin de fer et des trains ne sont restés qu'un rail qui représentait les liaisons physiques entre machines et que la forme des wagons pour représenter les cellules. En montrant le simulateur aux étudiants, nous ne nous attendions pas à ce qu'ils relèvent cette métaphore.

D'un point de vue pédagogique, les étudiants ont trouvé qu'utiliser le simulateur seul ne leur permettrait certainement pas de comprendre et de mémoriser les différents concepts. Cependant, à titre d'illustration complémentaire à un cours plus ample, le simulateur leur aurait permis de mieux saisir le dynamisme des concepts et d'avoir facilement un niveau de rétention plus élevé qu'avec ce seul cours.

La métaphore du magnétoscope a été très vite assimilée et les étudiants estiment qu'elle permettrait à un professeur de présenter lui même son cours d'une façon plus dynamique s'il utilisait le simulateur. La possibilité de mettre l'animation en état de pause, notamment, peut être très pratique pour donner des explications supplémentaires par rapport à ce qui paraît à l'écran et offre ainsi une possibilité de faire le lien avec ce qui a été dit précédemment par le professeur.

Certains étudiants auraient souhaité que les animations soient décomposées en plus d'étapes bien précises et qu'il soit possible de passer d'une étape à la suivante au rythme de chacun. Le simulateur tel qu'il fonctionne ne leur offre pas cette possibilité et il peut être nécessaire d'effectuer plusieurs fois de suite la même animation pour observer toutes les informations affichées. Une des solutions qu'ils ont suggérées, serait d'offrir la possibilité de jouer sur la vitesse dans les trois animations.

Un autre souhait exprimé par les étudiants serait d'avoir la possibilité de lancer une animation et de passer d'un concept à l'autre sans que celle-ci ne s'arrête. Ils veulent en fait pouvoir mettre les choses en parallèle à n'importe quel moment. Nous leur avons expliqué les raisons de notre choix de stopper l'animation si l'utilisateur changeait de concept. Tout d'abord chaque animation peut être appelée séparément par l'apprenant uniquement pour illustrer ce qu'il vient d'acquérir en consultant le module de formation. Ensuite, il nous semblait important que les trois concepts, bien que très liés, soient clairement distingués les uns des autres. Cependant, il est vrai que ce choix fut essentiellement un choix arbitraire et que nous ne pensions pas que des utilisateurs potentiels pourraient formuler une telle requête.

III.2 L'avis d'un spécialiste IHM

Nous avons demandé à Monsieur Jean Vanderdonckt, chargé de cours et spécialiste IHM (Interface **H**omme/**M**achine) de l'Institut d'Informatique des Facultés Universitaires Notre-Dame de la Paix à Namur, d'évaluer notre application. Nous aimerions ici le remercier pour le temps qu'il nous a consacré et pour les conseils qu'il nous a prodigués.
Tout comme les étudiants, l'avis du spécialiste IHM au sujet de notre application fut globalement positif. Il nous a cependant signalé quelques erreurs ergonomiques.

Une des critiques formulées concerne le potentiomètre que nous faisons apparaître dans l'applet Java. Aucun libellé ne permet à un utilisateur qui découvrirait le simulateur seul de savoir que ce potentiomètre sert à régler la vitesse lors de l'animation du concept de couches. De même aucune borne supérieure ou inférieure ne lui est attribuée et l'utilisateur ignore quelles sont les unités de mesure associées à ce potentiomètre. Etant donné que la possibilité de modifier la vitesse n'est offerte à l'utilisateur que dans le cas de l'illustration du concept de couches, il serait judicieux, selon le spécialiste, de supprimer ou de masquer la représentation du potentiomètre dans les deux autres animations. Il s'agit là d'un erreur de cohérence. Nous aurions également pu ajouter un sigle qui permette à l'utilisateur de savoir dans quel sens il doit faire varier le potentiomètre pour augmenter (et respectivement diminuer) la vitesse. Le spécialiste IHM nous a suggéré dans ce cas d'utiliser l'image d'un lapin pour montrer comment augmenter la vitesse et celle d'une tortue pour la diminuer.

Du point de vue de l'animation, tout est cohérent. Les couleurs ne sont pas les plus ergonomiques qu'il soit mais, leur association unique à un seul objet est une bonne idée. Cette association se remarque surtout aux niveaux suivants :

- Dans l'animation du concept de couches, la cellule construite dans l'applet reprend pour chaque partie la couleur de la couche qui l'a construite
- Dans l'animation du concept de commutation, les lignes changeant de couleur passent du noir à la couleur de la cellule analysée par le commutateur
- Dans les trois animations, nous avons systématiquement choisi de représenter les objets statiques par des couleurs froides et les objets dynamiques par des couleurs chaudes

Lors de l'animation du concept de commutation, le spécialiste IHM pense que le temps de changement de couleur de la ligne consultée est un peu court. Il reconnaît cependant que ce temps respecte le processus, c'est à dire le temps pendant lequel la cellule reste dans le commutateur, et que cette cohérence est un bénéfice.

Nous avons toujours respecté toutes les dimensions ergonomiques des axes, notamment dans l'applet Java lors de l'animation du concept de commutation. Dans un univers bidimensionnel, l'axe horizontal représente souvent le temps. La façon dont nous avons placé les tables de commutations nous permet d'associer à l'axe horizontal le suivi temporel d'évolution de la cellule à travers le réseau ATM.

Dans la scène VRML, il est important de définir des points de repère facilement accessibles à l'utilisateur. En effet, en utilisant les possibilités de navigation du tableau de bord de la scène, un utilisateur peut rapidement se trouver bloqué et ne plus très bien savoir où il se trouve par rapport à son angle de vue d'origine. Grâce aux vues prédéfinies, l'utilisateur peut oser se promener à travers le réseau. Il pourra toujours revenir très facilement à son point de départ. Les vues permettent aussi d'éviter des manipulations inutiles.

La métaphore du magnétoscope va permettre à l'utilisateur de devenir actif dans l'animation. Il peut prendre, dans une certaine mesure, les commandes de l'animation et peut ainsi s'impliquer au mieux dans son apprentissage. Cette interactivité va lui permettre un taux de rétention plus grand des concepts qu'il observe. Dans notre cas, le spécialiste IHM a approuvé ce choix mais aurait préféré qu'il existe plus de paramètres à faire varier. Il aurait peut être même fallu créer des profils utilisateur en fonctions desquels l'animation aurait été

très simple avec peu de paramètres variables ou aurait pu être plus complexe avec plus de paramètres modifiables.

La scène VRML est assez simple pour ne pas poser de problèmes d'interface. A l'heure actuelle, il existe peu de règles ergonomiques propres à la réalité virtuelle. Les quelques règles existantes sont essentiellement basées sur des niveaux de performance, de taille et de décomposition de fichiers etc. Pour notre part, le fichier composant la scène VRML de base est extrêmement simple et de petite taille. Les performances ne sont pas trop mauvaises. Pour peu que le processeur utilisé soit assez puissant (à partir d'un Pentium 90), l'animation est fluide. Cette fluidité est principalement due au fait que nous utilisons le nombre minimal de cellules nécessaires à la compréhension du concept illustré.

III.3 Une auto-évaluation

Dans cette section, nous allons critiquer notre propre travail, ce qui n'est jamais une chose aisée. Nous espérons pouvoir prendre assez de recul pour rester suffisamment objectif dans ce que nous allons présenter au lecteur.

La première impression que nous avons eue une fois le simulateur terminé, fut de penser que personne ne le prendrait pour un logiciel "sérieux". La scène VRML en trois dimensions donne, au premier abord, un aspect ludique à notre application. Cependant, loin d'être un désavantage, nous pensons que ce côté des choses pourrait peut-être inciter l'utilisateur à approfondir ses connaissances sur les trois concepts illustrés. L'utilisation de la réalité virtuelle pourrait elle-même attirer des utilisateurs nouveaux.

Nous devons bien reconnaître que notre application ne correspond pas vraiment à ce que l'on appelle communément un simulateur. L'utilisateur ne peut faire varier aucun paramètre. Les seules possibilités d'interaction qui lui sont offertes sont les suivantes :

- La possibilité de démarrer, mettre en pause ou stopper l'animation
- La possibilité de choisir le concept qu'il veut illustrer
- Les possibilités de navigation dans la scène VRML qui lui sont offertes par le tableau de bord de la scène VRML

Nous pensons que l'application telle quelle serait bien plus utile à un professeur pour illustrer les concepts qu'il enseigne sur un plan théorique, qu'à un étudiant qui voudrait approfondir ses connaissances du fonctionnement des réseaux ATM.

Le manque de standardisation de l'EAI peut entraîner une lenteur de l'animation et une éventuelle surcharge du processeur si celui-ci n'est pas assez puissant. Cette lenteur est un peu paradoxale pour illustrer le fonctionnement d'un réseau ATM qui fait partie des réseaux hauts débits. Comme nous l'avons déjà mentionné, la non standardisation de l'EAI ne nous permet pas de gérer les événements associés aux objets VRML créés dynamiquement. La solution que nous avons trouvée pour animer les cellules engendre parfois dans la scène VRML un étirement de ces cellules que nous ne pouvons expliquer. Cet étirement de la cellule est assez inopportun pour représenter un objet qui est toujours d'une taille fixe de 53 octets. D'un point de vue pédagogique cela pourrait se révéler assez ennuyeux. Nous trouvons également que cette lenteur peut être particulièrement désagréable par moment et peut même gêner l'utilisation du tableau de bord de la scène VRML en provoquant des déplacements dans la scène 3D saccadés et désordonnés.

Nous avons configuré le tableau de bord de la scène VRML pour qu'il ne contienne que les boutons vraiment nécessaires à la navigation de l'utilisateur dans le monde en 3D. Etant donné que nous sommes habitués à manipuler ce tableau de bord, nous oublions parfois qu'il peut être assez complexe pour une personne qui vient de le découvrir. Nous avons été forcés de constater cet état des choses en proposant aux étudiants d'utiliser notre application. Il aurait peut-être été opportun d'en expliquer un peu le fonctionnement. En plus de cette difficulté de manipulation, le choix des couleurs ton sur ton provoque parfois des difficultés de lecture des informations affichées sous les boutons du tableau de bord. Cependant, nous n'avons pas conçu ce dernier, nous ne faisons que l'utiliser. Le problème de la manipulation serait probablement identique si nous avions utiliser le plugin VRML proposé par une autre société mais les couleurs choisies auraient peut être faciliter la lecture.

IV Les extensions possibles

Après avoir évalué notre application en tentant de prendre le maximum de recul, nous allons présenter au lecteur les différentes extensions auxquelles nous avons songé. La puissance du moteur du simulateur et le niveau de modularité que nous y avons introduit devraient permettre de mener ces extensions à terme sans grandes difficultés.

Les premières extensions que l'on pourrait apporter à l'application consistent à lever les différentes restrictions que nous avons posées dans le cahier des charges. Il serait par exemple possible d'illustrer le mode de connexion commuté en plus du mode de connexion permanent. Pour ce simple cas on peut envisager deux évolutions différentes.

La première évolution consisterait à montrer à l'utilisateur de l'application comment se créent les circuits virtuels au sein du réseau ATM en ne lui offrant aucune autre possibilité d'interaction sur l'animation que celles existant actuellement. On illustrerait seulement une étape supplémentaire dans l'animation. Dans ce cas, on pourrait par exemple choisir de montrer ce principe dans l'animation relative au concept de commutation. Les tables affichées ne contiendraient aucune information et c'est seulement lors du protocole de création du circuit que les commutateurs les mettraient à jour.

La seconde évolution envisageable engendrerait des modifications plus profondes de l'application. Il s'agirait d'offrir à l'utilisateur la possibilité d'introduire lui-même les paramètres de création de circuit, un peu comme le ferait un administrateur réseau dans la vie courante. Cette extension suppose que l'on implémente le fonctionnement de la couche AAL afin de permettre le traitement des différentes classes de service.

En plus d'implémenter la couche AAL, on pourrait mettre en œuvre d'autres extensions au simulateur. Il sera par exemple possible d'illustrer le traitement des erreurs. Une des propositions que nous faisons pour mettre en place le traitement d'erreur serait de laisser à l'utilisateur le choix du type d'erreur qu'il voudrait voir traitée par le réseau. Nous proposons ici quelques exemples :

- La destruction d'une liaison physique (les câbles n'étant pas à l'abri de bulldozers mal intentionnés)
- L'insertion erronée de cellules
- La destruction de cellules

- La déformation de l'entête de la cellule
- La congestion du réseau

Naturellement, en donnant ces exemples, nous ne faisons qu'introduire quelques pistes possibles d'extensions.

Après la discussion que nous avons eue avec le spécialiste IHM, ce dernier nous a proposé, dans le cas où nous proposerions des extensions au simulateur, de tenir compte du profil de l'utilisateur. On pourrait supposer qu'un utilisateur novice travaillerait sur l'application telle qu'elle existe actuellement et que, au fur et à mesure qu'il acquiert de l'expérience, le simulateur se complexifie et lui offre la possibilité d'intervenir sur de plus en plus de paramètres différents et, ce, jusqu'à un niveau qui serait considéré comme un niveau d'expert.

Les diverses extensions que nous venons de décrire sont liées aux fonctionnalités propres du simulateur. Plusieurs extensions et améliorations peuvent également être faites au niveau de l'interface et plus particulièrement de l'interaction entre Java et VRML (dès que l'EAI aura été standardisée). Par exemple pour l'instant, le nombre de cellules circulant dans le réseau est limité au strict minimum pour éviter une surcharge du processeur. La standardisation de l'EAI permettrait de gérer les événements associés aux objets VRML créés dynamiquement. De ce fait, pour déplacer les cellules, il ne serait plus nécessaire d'avoir recours à la solution que nous avons décrite dans le chapitre X. Il suffirait d'envoyer à la cellule un événement lui indiquant ses nouvelles coordonnées. L'animation gagnerait en fluidité et en réalisme. Il serait en effet possible de gérer la vitesse facilement quel que soit le concept qu'on illustre, on pourrait envoyer un nombre de cellules beaucoup plus important, etc.

Dans la scène VRML elle-même, on pourrait introduire des nouveautés comme la possibilité "d'entrer" dans les commutateurs voir comment sont traitées les cellules. Il faut reconnaître que ce genre d'extension augmenterait probablement la charge du processeur. Il est aussi possible que l'aspect ludique dont nous avons parlé soit renforcé et que l'attrait de l'utilisateur soit ainsi plus grand. Envisager une plus grande marge d'action sur la scène elle-même pourrait être agréable pour mieux impliquer l'utilisateur dans son apprentissage des grands concepts des réseaux ATM. On pourrait envisager la possibilité de le laisser dessiner lui-même son réseau en lui proposant les objets 3D représentant les commutateurs, les machines hôtes et les liaisons physiques. Il pourrait à tout moment décider de supprimer une machine ou une liaison.

En bref, nous estimons que notre application peut être complétée et améliorée de nombreuses façons. La forte modularité que nous avons introduite dans la conception et la puissance du moteur de simulation sont probablement parmi les causes de cette possible évolution.

V Conclusion

Bien qu'imparfait sur certains points, notre simulateur semble plaire aux nombreuses personnes qui l'ont testé. La nouveauté des langages, l'utilisation de la réalité virtuelle et du WWW ne sont certainement pas étrangers à cet attrait. Cependant, le premier de nos objectifs était de réaliser un produit fini et utilisable dans les délais de temps qui nous étaient impartis et en respectant un cahier des charges assez conséquent. Au vu des critiques qui ont été formulées, il nous semble que le but soit atteint.

Ce chapitre clôture la quatrième et dernière partie de ce mémoire.

Conclusion

Partant de la description du projet auquel nous étions associés durant notre stage de fin de cycle, nous avons situé et décrit l'application dont il est question dans ce mémoire. Cette application représente notre modeste contribution à l'enseignement de la technologie ATM. Nous avons utilisé deux langages actuellement en vogue sur le Web, le langage Java et le langage de modélisation VRML. Le pari était de combiner ces deux langages afin de faire profiter l'enseignement de l'essor du multimédia. Faute d'une documentation adéquate, nous nous avons eu du mal à atteindre cet objectif. Par ailleurs, le langage de modélisation VRML n'étant pas encore "stabilisé", il était difficile pour nous d'utiliser toute sa puissance. Tout ceci explique ce manque d'interactivité invoquée dans la partie consacrée à l'évaluation de l'application. VRML n'est pas un langage de programmation.

En ce qui concerne Java, le mécanisme de thread nous a permis de mettre en oeuvre une exécution concurrente dans l'application. Notre période de stage étant limitée, nous n'avons pas pu utiliser des techniques formelles telles que les réseaux de pétri afin d'éliminer toute situation d'interblocage. Néanmoins, par rapport aux fonctionnalités attendues, le risque d'interblocage est nul. Notre couche de traitement peut supporter d'une manière concurrente plus d'une communication, contrairement à ce qui était prévu.

L'application que nous avons réalisée fait partie d'un vaste module multimédia de téléformation sur la technologie ATM appelée CIVIC, *Cité Virtuelle Connaissante*. Toutefois, notre application peut être utilisée d'une façon autonome, notamment dans le cadre d'un cours sur les réseaux. Concernant les autres résultats du projet (notamment le delta entre un accès à 2x64 Kbps et une liaison ATM à 2 Mbps dans le cadre d'un travail collaboratif), nous n'avons pas pu obtenir une quelconque information.

Nous pensons que l'application produite et le présent mémoire pourront être utiles d'une façon ou d'une autre au développement du concept d'EAO (*Enseignement Assisté par Ordinateur*).

Annexes

Annexe : Codes sources

I. Module *Réseau*

```
package Reseau;
import java.util.*;
import java.awt.*;
import Presentation.P_sousReseau;
/*============================================================*/
/* Classe sousReseau                    */
public abstract class sousReseau
      {
      // les champs d'un objet sousRéseau
      protected String NomSousRéseau;
      protected String TypeProtocole;
      protected String AdresseIP;
      protected Vector ListeEquipements;
      protected P_sousReseau RéfObjetGraphique;
      protected Vector ListeThreads;
      protected boolean StopThreads;
      protected boolean SuspendThreads;
      protected int SleepTime=0;
      /*
============================================================================== */
/*/*/
      public synchronized String GetNomSousRéseau()
            {
            return NomSousRéseau;
            }
/*/*/
      public synchronized String GetTypeProtocole()
            {
            return TypeProtocole;
            }
/*/*/
      public synchronized String GetAdresseIP()
            {
            return AdresseIP;
            }
/*/*/
      public synchronized Vector GetListeEquipements()
            {
            return ListeEquipements;
            }
/*/*/
      public void SetNomSousRéseau(String f_NomSousRéseau)
            {
            NomSousRéseau=f_NomSousRéseau;
            }
/*/*/
      public abstract void AdapterCouches();

/*/*/
      public synchronized boolean AddRéfEquipement(equipement
                  f_RéfEquipement)
            {
            if (ListeEquipements.contains(f_RéfEquipement))
```

```
                {
                return false;
                }
        else
                {
                ListeEquipements.addElement(f_RéfEquipement);
                return true;
                }
        }

/*/*/
    public synchronized boolean AddRéfThread(Thread f_RéfThread)
        {
        if (ListeThreads.contains(f_RéfThread))
                {
                return false;
                }
        else
                {
                ListeThreads.addElement(f_RéfThread);
                if (ListeThreads.size()==1)
                        {
                        GestionThreads w_Proc=new GestionThreads(this);
                        w_Proc.start();
                        }
                return true;
                }
        }
/*/*/
    public void StopAllThreads()
        {
        ListeThreads.removeAllElements();
        StopThreads=true;
        SuspendThreads=false;
        DelAllLiaisonsLog();
        }
/*/*/
    public void EnableAllThreads()
        {
        ListeThreads.removeAllElements();
        StopThreads=false;
        SuspendThreads=false;
        }
/*/*/
    public synchronized void SuspendAllThreads()
        {
        SuspendThreads=true;
        }
/*/*/
    public synchronized void ResumeAllThreads()
        {
        SuspendThreads=false;
        }
/*/*/
    public synchronized boolean isThreadsStopped()
        {
        return StopThreads;
        }
/*/*/
    public synchronized boolean isThreadSuspended()
        {
```

```
            return SuspendThreads;
            }
/*/*/
      public synchronized boolean isThreadActif(Thread f_RéfThread)
            {
            return ListeThreads.contains(f_RéfThread);
            }
/*/*/
      public synchronized int NombreThreads()
            {
            return ListeThreads.size();
            }
/*/*/
      public synchronized void SetSleepTime(int f_Temps)
            {
            SleepTime=f_Temps;
            }
      protected void DelAllLiaisonsLog()
            {
            Vector RéfListeEquipements=GetListeEquipements();
            int i=0;
            while (i<RéfListeEquipements.size())
                  {
                  Vector RéfListeLiaisonsPhy =
                        ((equipement)

      RéfListeEquipements.elementAt(i)).GetListeLiaisonsPhy();
                  int j=0;
                  while (j<RéfListeLiaisonsPhy.size())
                        {
                        liaisonPHY RéfLiaisonPhy =
                              (liaisonPHY) RéfListeLiaisonsPhy.elementAt(j);
      RéfLiaisonPhy.GetRéfSupport1().GetListeLiaisonsLog().removeAllElement
s();
      RéfLiaisonPhy.GetRéfSupport2().GetListeLiaisonsLog().removeAllElement
s();
                        j++;
                        }
                  i++;
                  }
            }
      protected synchronized void MajRéfThreads()
            {
            for (int i=0;i<ListeThreads.size();i++)
                  {
                  if (!(((Thread) ListeThreads.elementAt(i)).isAlive()))
                        {
                        ListeThreads.removeElementAt(i);
                        }
                  }
            }
      public void AbandonCPU()
            {
            try {Thread.sleep(150);}      // en millisecondes!!!!
            catch(InterruptedException signal) {return;}
            }
      public void AbandonCPU(int f_Temps)
            {
            try {Thread.sleep(f_Temps);}        // en millisecondes!!!!
            catch(InterruptedException signal) {return;}
            }
```

```
	public void GoSleep()
		{
		try {Thread.sleep(SleepTime);}
		catch(InterruptedException signal) {return;}
		}
/*/*/
	public synchronized P_sousReseau GetRéfObjetGraphique()
		{
		return RéfObjetGraphique;
		}
/*/*/
	public synchronized void SetRéfObjetGraphique(P_sousReseau
			f_RéfObjetGraphique)
		{
		RéfObjetGraphique=f_RéfObjetGraphique;
		}
	}
/*/*/
/*/*/
class GestionThreads extends Thread
	{
	sousReseau RéfSousRéseau;
/* constructeur */
	public GestionThreads(sousReseau f_RéfSousRéseau)
		{
		RéfSousRéseau=f_RéfSousRéseau;
		}
	public void run()
		{
		this.setPriority(NORM_PRIORITY);
		while (!RéfSousRéseau.ListeThreads.isEmpty())
			{
			RéfSousRéseau.MajRéfThreads();
			RéfSousRéseau.AbandonCPU();
			}
		}
	}
/*=====================================================================*/

package Reseau;
import java.util.*;
import java.awt.*;
import Presentation.P_equipement;
/*=============================================================*/
public abstract class equipement
	{
	// les champs d'un objet equipement
	protected String NomEquipement;
	protected String AdresseEquipement;
	protected sousReseau RéfSousRéseau;
	protected int TypeEquipement;
 //0=machineHôte, 1=commutateur, 2=routeur
	protected Vector ListeCouches;
	protected Vector ListeLiaisonsPhy;
	protected P_equipement RéfObjetGraphique;
	/*
============================================================================ */
	//
	public synchronized void SetNomEquipement(String f_NomEquipement)
		{
		NomEquipement=f_NomEquipement;
```

```
        }
public synchronized String GetNomEquipement()
        {
        return NomEquipement;
        }
public synchronized String GetAdresseEquipement()
        {
        return AdresseEquipement;
        }
public synchronized sousReseau GetRéfSousRéseau()
        {
        return RéfSousRéseau;
        }
public synchronized int GetTypeEquipement()
        {
        return TypeEquipement;
        }
public synchronized Vector GetListeCouches()
        {
        return ListeCouches;
        }
public synchronized Vector GetListeLiaisonsPhy()
        {
        return ListeLiaisonsPhy;
        }
public synchronized P_equipement GetRéfObjetGraphique()
        {
        return RéfObjetGraphique;
        }
public synchronized void SetRéfObjetGraphique(P_equipement
              f_RéfObjetGraphique)
        {
        RéfObjetGraphique=f_RéfObjetGraphique;
        }
public synchronized boolean AddRéfLiaisonPhy(liaisonPHY
              f_RéfLiaisonPhy)
        {
        if (ListeLiaisonsPhy.contains(f_RéfLiaisonPhy))
              {
              return false;
              }
        else
              {
              ListeLiaisonsPhy.addElement(f_RéfLiaisonPhy);
              return true;
              }
        }
public synchronized boolean AddRéfCouche(couche f_RéfCouche)
        /* on suppose que la couche a été déjà créée */
        {
        if (ListeCouches.contains(f_RéfCouche))
              {
              return false;
              }
        else
              {
              ListeCouches.addElement(f_RéfCouche);
              return true;
              }
        }
public synchronized couche GetCoucheWithNom(String f_NomCouche)
```

```
            {
            for (int i=0;i<ListeCouches.size();i++)
                  {
                  if (((couche)
                  ListeCouches.elementAt(i)).GetNomCouche()==f_NomCouche)
                        {
                        return (couche) ListeCouches.elementAt(i);
                        }
                  }
            return null;
            }

      public boolean DataGET(trame f_RéfTrame)
            {
            System.out.println("equipement.java : DataGET est une METHODE
VIDE");
            return false;
            };
      /* service fourni à un lien logique entrant. Port sortant pour la
ligne logique. L'équipement accepte une trame venant d'un liaison de
transmission logique */

      public boolean DataSEND(trame f_RéfTrame, liaisonLOG f_RéfLiaisonLog)
            {
            System.out.println("equipement.java : DataGET est une METHODE
VIDE");
            return false;
            };
      /* service fourni à sa couche physique
      La couche physique "demande" à l'équipement d'envoyer la trame */

}
/*=====================================================================*/


package Reseau;
/* Classe couche                    */
import java.util.Vector;
import java.awt.*;
import Presentation.P_couche;
/*=============================================================*/
public abstract class couche
      {
      // les champs d'un objet couche
      protected String NomCouche;
      protected equipement RéfEquipement;
      protected int NiveauOSI;
      protected Vector ListeCouchesSup;
      protected Vector ListeCouchesInf;
      protected P_couche RéfObjetGraphique;
      /* les méthodes de la classe */
      //
      public synchronized void AddRéfVoisineSup(couche f_RéfCouche)
            {
            ListeCouchesSup.addElement(f_RéfCouche);
            }
      public synchronized void AddRéfVoisineInf(couche f_RéfCouche)
            {
            ListeCouchesInf.addElement(f_RéfCouche);
            }
      public synchronized String GetNomCouche()
```

```
            {
            return NomCouche;
            }
      public synchronized int GetNiveauOSI()
            {
            return NiveauOSI;
            }
      public synchronized equipement GetRéfEquipement()
            {
            return RéfEquipement;
            }
      public synchronized Vector GetListeCouchesSup()
            {
            return ListeCouchesSup;
            }
      public synchronized Vector GetListeCouchesInf()
            {
            return ListeCouchesInf;
            }
      public synchronized sousReseau GetRéfSousRéseau()
            {
            return this.RéfEquipement.GetRéfSousRéseau();
            }
      public synchronized couche GetCoucheWithNom(Vector f_RéfListeCouches,
            String f_LabelCouche)
            {
            int j=-1;
            for (int i=0;i<f_RéfListeCouches.size();i++)
                  {
                  if (((couche)
                   f_RéfListeCouches.elementAt(i)).GetNomCouche()==
                              f_LabelCouche)
                        {
                        j=i;
                        }
                  }
            if (j!=-1)
                  {
                  return (couche) f_RéfListeCouches.elementAt(j);
                  }
            else
                  {
                  return null;
                  }
            }
      public synchronized P_couche GetRéfObjetGraphique()
            {
            return RéfObjetGraphique;
            }
      public synchronized void SetRéfObjetGraphique(P_couche
                  f_RéfObjetGraphique)
            {
            RéfObjetGraphique=f_RéfObjetGraphique;
            }
      }
/*=====================================================================*/

package Reseau;
import java.util.*;
import java.awt.*;
import Presentation.P_liaisonPHY;
```

```
/*=============================================================*/
public final class liaisonPHY
      {
      // les champs d'un objet liaisonPHY
      int NuméroLiaison;
      supportPHY RéfSupport1;
      supportPHY RéfSupport2;
      P_liaisonPHY RéfObjetGraphique;
      /* les méthodes de la classe */
      //
      public liaisonPHY(int f_NuméroLiaison, equipement f_RéfEquipement1,
                        equipement f_RéfEquipement2)
            {
            /* on suppose que les deux équipements ont été déjà créés */
            NuméroLiaison=NuméroLiaison;
            RéfSupport1=new supportPHY(this, f_RéfEquipement2); //1--->2
            RéfSupport2=new supportPHY(this, f_RéfEquipement1); //2--->1
            f_RéfEquipement1.AddRéfLiaisonPhy(this);
            f_RéfEquipement2.AddRéfLiaisonPhy(this);
            RéfObjetGraphique=null;
            }
      public synchronized sousReseau GetRéfSousRéseau()
            {
            return RéfSupport1.GetRéfSousRéseau();
            }
      public synchronized supportPHY GetRéfSupport1()
            {
            return RéfSupport1;
            }
      public synchronized supportPHY GetRéfSupport2()
            {
            return RéfSupport2;
            }
      public synchronized supportPHY GetSupportPhyDeSortie(equipement
                  f_RéfEquipement)
            {
            if (RéfSupport2.GetRéfEquipementCible()==f_RéfEquipement)
                  {
                  return RéfSupport1;
                  }
            else
                  {
                  return RéfSupport2;
                  }
            }
      public synchronized P_liaisonPHY GetRéfObjetGraphique()
            {
            return RéfObjetGraphique;
            }
      public synchronized void SetRéfObjetGraphique(P_liaisonPHY
                  f_RéfObjetGraphique)
            {
            RéfObjetGraphique=f_RéfObjetGraphique;
            }
      }

/*=============================================================*/
package Reseau;
import java.util.*;
import java.awt.*;
/*=============================================================*/
```

```
public final class supportPHY
      {
      // les champs d'un objet supportPHY
      liaisonPHY RéfLiaisonPhy;
      equipement RéfEquipementCible;
      Vector ListeLiaisonsLog;
      /* les méthodes de la classe */
      //
      public supportPHY(liaisonPHY f_RéfLiaisonPhy, equipement
            f_RéfEquipementCible)
            {
            RéfEquipementCible=f_RéfEquipementCible;
            ListeLiaisonsLog=new Vector();
            RéfLiaisonPhy=f_RéfLiaisonPhy;
            }
      public synchronized equipement GetRéfEquipementSource()
            {
            if (RéfLiaisonPhy.GetRéfSupport1()==this)
                  {
                  return
                  RéfLiaisonPhy.GetRéfSupport2().GetRéfEquipementCible();
                  }
            else
                  {
                  return
                  RéfLiaisonPhy.GetRéfSupport1().GetRéfEquipementCible();
                  }
            }
      public synchronized equipement GetRéfEquipementCible()
            {
            return RéfEquipementCible;
            }
      public synchronized liaisonPHY GetRéfLiaisonPhy()
            {
            return RéfLiaisonPhy;
            }
      public synchronized sousReseau GetRéfSousRéseau()
            {
            return RéfEquipementCible.GetRéfSousRéseau();
            }
      public synchronized boolean AddRéfLiaisonLog(liaisonLOG
            f_RéfLiaisonLog)
            {
            if (ListeLiaisonsLog.contains(f_RéfLiaisonLog))
                  {
                  return false;
                  }
            else
                  {
                  ListeLiaisonsLog.addElement(f_RéfLiaisonLog);
                  return true;
                  }
            }
      public synchronized boolean DelRéfLiaisonLog(liaisonLOG
            f_RéfLiaisonLog)
            {
            if (ListeLiaisonsLog.contains(f_RéfLiaisonLog))
                  {
                  ListeLiaisonsLog.removeElement(f_RéfLiaisonLog);
                  return true;
                  }
```

```
            else
                  {
                  return false;
                  }
            }
      public synchronized Vector GetListeLiaisonsLog()
            {
            return ListeLiaisonsLog;
            }
      }

/*================================================================*/
package Reseau;
import java.util.*;
import java.awt.*;
import Presentation.P_liaisonPHY;
import Presentation.P_sousReseau;
import Presentation.P_sousReseauATM;
/*================================================================*/
public abstract class liaisonLOG
      {
      // les champs d'un objet liaisonLOG
      protected int NuméroLiaison;
      protected supportPHY RéfSupportPhy;
      protected Vector ListeTrames;
      /* les méthodes de la classe */
      public synchronized int GetNuméroLiaison()
            {
            return NuméroLiaison;
            }
      public synchronized liaisonPHY GetRéfLiaisonPhy()
            {
            return RéfSupportPhy.GetRéfLiaisonPhy();
            }
      public synchronized supportPHY GetRéfSupportPhy()
            {
            return RéfSupportPhy;
            }
      public synchronized equipement GetRéfEquipementCible()
            {
            return RéfSupportPhy.GetRéfEquipementCible();
            }
      public synchronized equipement GetRéfEquipementSource()
            {
            return RéfSupportPhy.GetRéfEquipementSource();
            }
      public synchronized Vector GetListeTrames()
            {
            return ListeTrames;
            }
      public synchronized sousReseau GetRéfSousRéseau()
            {
            return GetRéfEquipementCible().GetRéfSousRéseau();
            }
      public synchronized boolean DataTRANSPORT()
            {
            System.out.println("liaisonLOG.java : DataTRANSPORT est une
METHODE VIDE");
            return false;
            }
      public synchronized boolean DelRéfTrame(trame f_RéfTrame)
```

```
            {
            if (ListeTrames.contains(f_RéfTrame))
                  {
                  ListeTrames.removeElement(f_RéfTrame);
                  return true;
                  }
            else
                  {
                  return false;
                  }
            }
      public synchronized boolean AddRéfTrame(trame f_RéfTrame)
            {
            if (!ListeTrames.contains(f_RéfTrame))
                  {
                  ListeTrames.addElement(f_RéfTrame);
                  return true;
                  }
            else
                  {
                  return false;
                  }
            }

}

/*=====================================================================*/
package Reseau;
/* Classe trame                  */
import java.util.*;
import java.awt.*;
import Presentation.P_trame;
/*=====================================================================*/
public abstract class trame
      {
      // les champs d'un objet trame
      protected int NuméroSéquence;
      protected String Donnée;
      protected Vector RéfLocalisation;
            /* RéfLocalisation est une liste à deux éléments.
Le 1er élément contient le type de la localisation
                  "0"=couche dans DataREQ,
                  "1"=couche dans DataIND,
                  "2"=LiaisonLOG dans DataTRANSPORT
            et le 2ème, une ref. à l'objet qui traite la trame */
      protected P_trame RéfObjetGraphique;
      /* les méthodes de la classe */
      public synchronized void SetDonnée(String f_Donnée)
            {
            Donnée=f_Donnée;
            }
      public synchronized String GetDonnée()
            {
            return Donnée;
            }
      public synchronized int GetNuméroSéquence()
            {
            return NuméroSéquence;
            }
      public synchronized void SetRéfLocalisation(couche f_RéfCouche,
                  String f_TypeLocalisation)
```

```
            {
            RéfLocalisation=new Vector(2);
            RéfLocalisation.insertElementAt(f_TypeLocalisation,0);
            RéfLocalisation.insertElementAt(f_RéfCouche,1);
            }
      public synchronized void SetRéfLocalisation(liaisonLOG
            f_RéfLiaisonLog)
            {
            RéfLocalisation=new Vector(2);
            RéfLocalisation.insertElementAt("2",0);
            RéfLocalisation.insertElementAt(f_RéfLiaisonLog,1);
            }
      public synchronized Vector GetRéfLocalisation()
            {
            return RéfLocalisation;
            }
      public synchronized P_trame GetRéfObjetGraphique()
            {
            return RéfObjetGraphique;
            }
      public synchronized void SetRéfObjetGraphique(P_trame
            f_RéfObjetGraphique)
            {
            RéfObjetGraphique=f_RéfObjetGraphique;
            }
      }
/*=====================================================================*/
```

II. Module *Atm*

```
package Reseau.Atm;
import java.util.*;
import java.awt.*;
import Presentation.P_couche;
import Presentation.P_sousReseauATM;
import Reseau.sousReseau;
import Reseau.liaisonPHY;
import Reseau.supportPHY;
import Reseau.equipement;
/*=============================================================*/
/* Classe sousReseauATM                    */
public final class sousReseauATM extends sousReseau
      {
      Vector ListeLiaisonsVcc;
      /* les méthodes de la classe */
      /*------------------- DEBUT CONSTRUCTEUR ------------------------*/
      public sousReseauATM(String f_AdresseIP)
            {
            AdresseIP=f_AdresseIP;
            TypeProtocole="atm";
            ListeEquipements =new Vector();
            ListeThreads=new Vector();
            ListeLiaisonsVcc=new Vector();
            P_sousReseauATM ObjetGraphique=null;
            // le sous-réseau est collé à son objet de présentation
            (graphique)
            }
      /*------------------- FIN CONSTRUCTEUR --------------------------*/
      public synchronized Vector GetListeLiaisonsVcc()
            {
            return ListeLiaisonsVcc;
```

```
        }
    public synchronized void AdapterCouches()
    // cette méthode ajoute les couches d'accès dans tous les équipements
        // appartenant au sous-réseau.
        {
        for (int i=0;i<ListeEquipements.size();i++)
            {
            equipementATM w_RéfEquipement=
                (equipementATM) ListeEquipements.elementAt(i);
            /*pour tout equipementATM, on a les couches phy et atm */
            couchePHY CouchePhy =
                new couchePHY("PHY", w_RéfEquipement);
            w_RéfEquipement.AddRéfCouche(CouchePhy);
            /* */
            coucheATM CoucheAtm =
                new coucheATM("ATM", w_RéfEquipement);
            w_RéfEquipement.AddRéfCouche(CoucheAtm);
            /* */
            CoucheAtm.AddRéfVoisineInf(CouchePhy);
            CouchePhy.AddRéfVoisineSup(CoucheAtm);
            /* */

            switch (w_RéfEquipement.GetTypeEquipement())
                {
            case 0 : // Une MACHINE HOTE ATM => 3 couches
                    coucheAAL CoucheAal =
                        new coucheAAL("AAL", w_RéfEquipement);
                    w_RéfEquipement.AddRéfCouche(CoucheAal);
                    CoucheAtm.AddRéfVoisineSup(CoucheAal);
                    CoucheAal.AddRéfVoisineInf(CoucheAtm);
                    /*- ajoutée pour le test => à enlever ---*/
                    coucheAPL CoucheApl =
                        new coucheAPL("APL", w_RéfEquipement);
                    w_RéfEquipement.AddRéfCouche(CoucheApl);
                    CoucheAal.AddRéfVoisineSup(CoucheApl);
                    CoucheApl.AddRéfVoisineInf(CoucheAal);
                    /*--------------------------------*/
                    break;
            case 1 : // Un COMMUTATEUR ATM => 2 couches
                    break;
                }
            }
        /*-------------------------------------*/
        }
    synchronized boolean AddRéfLiaisonVcc(liaisonVCC f_RéfLiaisonVcc)
        {
        if (ListeLiaisonsVcc.contains(f_RéfLiaisonVcc))
            {
            return false;
            }
        else
            {
    /*-------------------------------------------------------------------
-------------*/
            equipementATM w_RéfEquipementCourant =
                f_RéfLiaisonVcc.GetRéfMachineHôteSource();
            equipementATM w_RéfMachineHôteCible =
                f_RéfLiaisonVcc.GetRéfMachineHôteCible();
            equipementATM w_RéfMachineHôteSource =
                f_RéfLiaisonVcc.GetRéfMachineHôteSource();
            liaisonPHY w_RéfNextLiaisonPhy;
```

```
                while (w_RéfEquipementCourant!=w_RéfMachineHôteCible)
                      {
                      if (w_RéfEquipementCourant==w_RéfMachineHôteSource)
                            {
                            /* C'est forcement une machine terminale */
                            w_RéfNextLiaisonPhy =(liaisonPHY)
          w_RéfEquipementCourant.GetListeLiaisonsPhy().elementAt(0);
                            }
                      else
                            {
                            /* C'est donc un commutateur! Sur base de la
table des adresse et de l'adresse de la machine de destination, je trouve
la prochaine liaison physique à prendre */
                            w_RéfNextLiaisonPhy=((commutateurATM)
                      w_RéfEquipementCourant).GetRéfLiaisonPhyDeSortie(
                      w_RéfMachineHôteCible.GetAdresseEquipement());
                            }
                            if (w_RéfNextLiaisonPhy==null)
                                  {
                                  System.out.println("(sousRéseauATM -
AddRéfLiaisonVcc) cul de sac!");
                                  break;
                      }
                      /* support1 ? support2 ?*/
                      supportPHY w_RéfNextSupportPhy=
      w_RéfNextLiaisonPhy.GetSupportPhyDeSortie(w_RéfEquipementCourant);
                      /* Création d'un VCL d'entrée sur
                                  w_RéfNextSupportPhy */
                      liaisonVCL w_RéfLiaisonVcl = new
                      liaisonVCL(f_RéfLiaisonVcc, w_RéfNextSupportPhy);
                      /* ce VCL est le premier VCL du VCC courant */
                      if (w_RéfEquipementCourant==w_RéfMachineHôteSource)
                            {
                f_RéfLiaisonVcc.SetRéfFirstLiaisonVcl(w_RéfLiaisonVcl);
                f_RéfLiaisonVcc.SetRéfLastLiaisonVcl(w_RéfLiaisonVcl);
                            }
                      else
                            {
                            ((commutateurATM)
      w_RéfEquipementCourant).AddRéfRoute(f_RéfLiaisonVcc.GetRéfLastLiaison
Vcl(),w_RéfLiaisonVcl);

      f_RéfLiaisonVcc.SetRéfLastLiaisonVcl(w_RéfLiaisonVcl);
                            }
                      w_RéfEquipementCourant=(equipementATM)
w_RéfNextSupportPhy.GetRéfEquipementCible();
                      }
                /*-----------------------------------------*/
                ListeLiaisonsVcc.addElement(f_RéfLiaisonVcc);
                return true;
                }
          }
      public synchronized void DelAllRéfLiaisonsVcc()
          {
          ListeLiaisonsVcc.removeAllElements();
          }
      synchronized boolean DelRéfLiaisonVcc(liaisonVCC f_RéfLiaisonVcc)
          {
          if (!ListeLiaisonsVcc.contains(f_RéfLiaisonVcc))
                {
                return false;
```

```
                    }
              else
                    {
                    /*-------------------------------------------*/
                    liaisonVCL
w_RéfLiaisonVcl=f_RéfLiaisonVcc.GetRéfFirstLiaisonVcl();
                    equipementATM w_RéfEquipementCourant=(equipementATM)
w_RéfLiaisonVcl.GetRéfEquipementCible();
                    equipementATM
w_RéfMachineHôteCible=f_RéfLiaisonVcc.GetRéfMachineHôteCible();
                    //
                    while (w_RéfEquipementCourant!=w_RéfMachineHôteCible)
                          {
                          /* Consultation de la table de commutation pour
                          sauvegarder le prochain VCL */
                          f_RéfLiaisonVcc.SetRéfFirstLiaisonVcl(
                                ((commutateurATM)
w_RéfEquipementCourant).GetRéfLiaisonVclSortant(w_RéfLiaisonVcl));
                          /* suppression de l'entrée de la table de
                          commutation */
                          ((commutateurATM)
w_RéfEquipementCourant).DelRoute(w_RéfLiaisonVcl);
                          /* suppression du VCL dans la liaison physique
                          entrant */
      w_RéfLiaisonVcl.GetRéfSupportPhy().DelRéfLiaisonLog(w_RéfLiaisonVcl);
                          /* */
      w_RéfLiaisonVcl=f_RéfLiaisonVcc.GetRéfFirstLiaisonVcl();
                          w_RéfEquipementCourant=(equipementATM)
                                w_RéfLiaisonVcl.GetRéfEquipementCible();
                          }
                          /* suppression du VCL de la dernière liaison
                          physique */
      w_RéfLiaisonVcl.GetRéfSupportPhy().DelRéfLiaisonLog(w_RéfLiaisonVcl);
                    /*-------------------------------------------*/
                    ListeLiaisonsVcc.removeElement(f_RéfLiaisonVcc);
                    return true;
                    }
              }
      }

/*=================================================================*/
package Reseau.Atm;
import java.util.*;
import java.awt.*;
import  Reseau.equipement;
/*=================================================================*/
/* Classe equipement                    */
public abstract class equipementATM extends equipement
{
      // les champs d'un objet equipement
      synchronized boolean DataGET(celluleATM f_RéfCellule)
      /* service fourni à un lien logique entrant. Port sortant pour la
ligne logique. L'équipement accepte une trame venant d'un liaison de
transmission logique */
            {
            try
                    {
                    traiteDataEqu ActionEQU = new traiteDataEqu(this,
                                                f_RéfCellule);
                    ActionEQU.start();
                    }
```

```
            catch(OutOfMemoryError signal)
                  {
                  System.out.println("** ERREUR ** (équipementATM -
DataGET) Out of memory :");
                  System.out.println("    "+signal);
                  return false;
                  }
            return true;
            }
      synchronized boolean DataSEND(celluleATM f_RéfCellule, liaisonVCL
                  f_RéfLiaisonVcl)
      /* service fourni à sa couche physique
       La couche physique "demande" à l'équipement d'envoyer la trame */

            {
            try
                  {
                  traiteDataEqu ActionEQU = new traiteDataEqu(this,
                                                f_RéfCellule,

                        f_RéfLiaisonVcl);
                  ActionEQU.start();
                  }
            catch(OutOfMemoryError signal)
                  {
                  System.out.println("** ERREUR **
                        (équipementATM - DataSEND) Out of memory :");
                  System.out.println("    "+signal);
                  return false;
                  }
            return true;
            }
private class traiteDataEqu extends Thread
      {
      celluleATM RéfCellule;
      equipement RéfEquipement;
      liaisonVCL RéfLiaisonVcl;
      char TypeTraitement;
/* constructeur lors d'un GET */
      traiteDataEqu(equipement f_RéfEquipement, celluleATM f_RéfCellule)
            {
            RéfEquipement=f_RéfEquipement;
            RéfCellule=f_RéfCellule;
            RéfLiaisonVcl=null;
            TypeTraitement='G';
            }
/* constructeur lors d'un SEND */
      traiteDataEqu(equipement f_RéfEquipement, celluleATM f_RéfCellule,
                              liaisonVCL f_RéfLiaisonVcl)
            {
            RéfEquipement=f_RéfEquipement;
            RéfCellule=f_RéfCellule;
            RéfLiaisonVcl=f_RéfLiaisonVcl;
            TypeTraitement='S';
            }
      public void run()
            {
            this.setPriority(NORM_PRIORITY);
            sousReseauATM w_RéfSousRéseau=(sousReseauATM)
                                    RéfEquipement.GetRéfSousRéseau();
            w_RéfSousRéseau.AddRéfThread(this);
```

```
            if (TypeTraitement=='G')
                  {
                  while (w_RéfSousRéseau.isThreadSuspended())
                        {
                        w_RéfSousRéseau.AbandonCPU();
                        }
                  if (w_RéfSousRéseau.isThreadActif(this))
                        {
                        ((couchePHY)

      RéfEquipement.GetCoucheWithNom("PHY")).DataIND((celluleATM)
                                                      RéfCellule);
                        }
                  }
            else
                  {
                  while (w_RéfSousRéseau.isThreadSuspended())
                        {
                        w_RéfSousRéseau.AbandonCPU();
                        }
                  if (w_RéfSousRéseau.isThreadActif(this))
                        {
                        RéfLiaisonVcl.DataTRANSPORT((equipementATM)
                              RéfEquipement, RéfCellule);
                        }
                  }
            }
      }
}

/*=====================================================================*/

package Reseau.Atm;
import java.util.*;
import java.awt.*;
import Presentation.P_machineHote;
import Reseau.sousReseau;
/*=================================================================*/
/* Classe machineHote                    */
public final class machineHote extends equipementATM
      {
      /* les méthodes de la classe */
      public machineHote(String f_AdresseMachineHôte, sousReseauATM
f_RéfSousRéseau)
            {
            AdresseEquipement=f_AdresseMachineHôte;
            RéfSousRéseau=f_RéfSousRéseau;
            RéfSousRéseau.AddRéfEquipement(this);
            TypeEquipement=0;
            ListeLiaisonsPhy = new Vector();
            ListeCouches = new Vector();
            RéfObjetGraphique=null;
            }
      }
/*=====================================================================*/

package Reseau.Atm;
/* Classe commutateurATM                  */
import java.util.*;
import java.awt.*;
import Reseau.liaisonPHY;
```

```
/*============================================================*/
public final class commutateurATM extends equipementATM
      {
      // les champs d'un objet commutateur
      Vector TableDeCommutation;
      Vector TableDesAdressesAtm;
      /* les méthodes de la classe */
      /* constructeur commutateur ATM  version 1 */
      public commutateurATM(sousReseauATM f_RéfSousRéseau)
            {
            RéfSousRéseau=f_RéfSousRéseau;
            TypeEquipement=1;
            TableDeCommutation = new Vector();
            TableDesAdressesAtm = new Vector();
            ListeLiaisonsPhy = new Vector();
            ListeCouches = new Vector(2);
            f_RéfSousRéseau.AddRéfEquipement(this);
            RéfObjetGraphique=null;
            //
            }
      synchronized void AddRéfRoute(liaisonVCL f_RéfLiaisonVclEntrée,
                                    liaisonVCL f_RéfLiaisonVclSortie)
            {
            routeATM w_RouteAtm=new routeATM(f_RéfLiaisonVclEntrée,
                        f_RéfLiaisonVclSortie);
            TableDeCommutation.addElement(w_RouteAtm);
            }
      public synchronized liaisonVCL GetRéfLiaisonVclSortant(liaisonVCL
                  f_RéfLiaisonVclEntrée)
            {
            for (int i=0; i<TableDeCommutation.size();i++)
                  {
                  if (((routeATM)

      TableDeCommutation.elementAt(i)).GetRéfLiaisonVclEntrée()==
                                    f_RéfLiaisonVclEntrée)
                        {
                        return ((routeATM)
                  TableDeCommutation.elementAt(i)).GetRéfLiaisonVclSortie()
                  ;
                        }
                  }
            System.out.println("** ERREUR **
                        (commutateurATM - GetRéfLiaisonVclsortant)");
            return null;
            }
      synchronized boolean DelRoute(liaisonVCL f_RéfLiaisonVclEntrée)
            {
            for (int i=0; i<TableDeCommutation.size();i++)
                  {
                  if (((routeATM)

      TableDeCommutation.elementAt(i)).GetRéfLiaisonVclEntrée()==
                                    f_RéfLiaisonVclEntrée)
                        {
                        TableDeCommutation.removeElementAt(i);
                        return true;
                        }
                  }
            System.out.println("** ERREUR ** (commutateurATM - DelRoute)");
            return false;
```

```
        }
    public synchronized void AddRéfAdresseAtm(equipementATM
            f_RéfMachineHôteCible, liaisonPHY f_RéfLiaisonPhyOut)
        {
        adresseATM w_AdresseATM=new
         adresseATM(f_RéfMachineHôteCible.GetAdresseEquipement(),
                f_RéfLiaisonPhyOut);
        TableDesAdressesAtm.addElement(w_AdresseATM);
        }
    public synchronized liaisonPHY GetRéfLiaisonPhyDeSortie(String
            f_AdresseMachineHôte)
        {
        for (int i=0; i<TableDesAdressesAtm.size();i++)
            {
            if (((adresseATM)

    TableDesAdressesAtm.elementAt(i)).AdresseMachineHôte==
            f_AdresseMachineHôte)
                {
                return ((adresseATM)
                TableDesAdressesAtm.elementAt(i)).GetRéfLiaisonPhy(
                );
                }
            }
        System.out.println("** ERREUR **
            (commutateurATM - GetRéfLiaisonPhyDeSortie)");
        return null;
        }
    }

/*=====================================================================*/
package Reseau.Atm;
import Reseau.liaisonPHY;
/*=====================================================================*/
/* Classe adresseATM                    */
final class adresseATM
    {
    // les champs d'un objet adresseATM
    String AdresseMachineHôte;
    liaisonPHY RéfLiaisonPhy;
    /* constructeur adresseATM */
    adresseATM(String f_AdresseMachineHôte, liaisonPHY f_RéfLiaisonPhy)
        {
        AdresseMachineHôte=f_AdresseMachineHôte;
        RéfLiaisonPhy=f_RéfLiaisonPhy;
        }
    synchronized String GetAdresseMachineHôte()
        {
        return AdresseMachineHôte;
        }
    synchronized liaisonPHY GetRéfLiaisonPhy()
        {
        return RéfLiaisonPhy;
        }
    }

/*=====================================================================*/
package Reseau.Atm;
/*=============================================================*/
final class routeATM
    {
```

```
        // les champs d'un objet routeATM
        liaisonVCL RéfLiaisonVclEntrée;
        liaisonVCL RéfLiaisonVclSortie;
        /* les méthodes de la classe */
        routeATM(liaisonVCL f_RéfLiaisonVclEntrée, liaisonVCL
                    f_RéfLiaisonVclSortie)
              {
              RéfLiaisonVclEntrée=f_RéfLiaisonVclEntrée;
              RéfLiaisonVclSortie=f_RéfLiaisonVclSortie;
              }
        synchronized liaisonVCL GetRéfLiaisonVclSortie()
              {
              return RéfLiaisonVclSortie;
              }
        synchronized liaisonVCL GetRéfLiaisonVclEntrée()
              {
              return RéfLiaisonVclEntrée;
              }
        }
/*=========================================================================*/

package Reseau.Atm;
import java.util.Vector;
import java.awt.*;
import Reseau.liaisonLOG;
import Reseau.sousReseau;
import Reseau.supportPHY;
/*=========================================================*/
/* Classe liaisonVCC                    */
public class liaisonVCC extends Reseau.liaisonLOG
        {
        machineHote RéfMachineHôteSource;
        machineHote RéfMachineHôteCible;
        liaisonVCL RéfFirstLiaisonVcl;
        liaisonVCL RéfLastLiaisonVcl;
        int NbreCellulesRecues;
        /* les méthodes de la classe */
        //
        public liaisonVCC(machineHote f_RéfMachineHôteSource,
                          machineHote f_RéfMachineHôteCible)
              {
              RéfMachineHôteCible=f_RéfMachineHôteCible;
              RéfMachineHôteSource=f_RéfMachineHôteSource;
              RéfFirstLiaisonVcl=null;
              RéfLastLiaisonVcl=null;
              //RéfSupportPhy=null;
              NuméroLiaison=1 + (int) Math.random();
              ((sousReseauATM) GetRéfSousRéseau()).AddRéfLiaisonVcc(this);
              ListeTrames=new Vector();
              }
        public synchronized machineHote GetRéfMachineHôteSource()
              {
              return RéfMachineHôteSource;
              }
        public synchronized machineHote GetRéfMachineHôteCible()
              {
              return RéfMachineHôteCible;
              }
        public synchronized sousReseau GetRéfSousRéseau()
              {
              return RéfMachineHôteSource.GetRéfSousRéseau();
```

```
        }
    synchronized void SetRéfLastLiaisonVcl(liaisonVCL f_RéfLiaisonVcl)
        {
        RéfLastLiaisonVcl=f_RéfLiaisonVcl;
        }
    synchronized void SetRéfFirstLiaisonVcl(liaisonVCL f_RéfLiaisonVcl)
        {
        RéfFirstLiaisonVcl=f_RéfLiaisonVcl;
        }
    synchronized void CelluleArrivée()
        {
        NbreCellulesRecues++;
        }
    public synchronized int GetNbreCellulesRecues()
        {
        return NbreCellulesRecues;
        }
    public synchronized int GetNbreCellulesTransmises()
        {
        return ListeTrames.size();
        }
    public synchronized liaisonVCL GetRéfFirstLiaisonVcl()
        {
        return RéfFirstLiaisonVcl;
        }
    public synchronized liaisonVCL GetRéfLastLiaisonVcl()
        {
        return RéfLastLiaisonVcl;
        }
    }
/*=====================================================================*/

package Reseau.Atm;
import java.util.Vector;
import Reseau.liaisonLOG;
import Reseau.supportPHY;
import Reseau.trame;
/*=============================================================*/
/* Classe liaisonVC                    */
public final class liaisonVCL extends liaisonLOG
    {
    // les champs d'un objet liaisonVC
    int VPI;
    liaisonVCC RéfLiaisonVcc;
    /* les méthodes de la classe */
    //
    liaisonVCL(liaisonVCC f_RéfLiaisonVcc, supportPHY f_RéfSupportPhy)
        {
        RéfSupportPhy=f_RéfSupportPhy;
        RéfLiaisonVcc=f_RéfLiaisonVcc;
        ListeTrames=new Vector();
        NuméroLiaison=CalculVCI();
        VPI=CalculVPI();
        RéfSupportPhy.AddRéfLiaisonLog(this);
        }
    public synchronized int GetVPI()
        {
        return VPI;
        }
    public synchronized liaisonVCC GetRéfLiaisonVcc()
        {
```

```
            return RéfLiaisonVcc;
            }
      synchronized int CalculVCI()
            {
            if (RéfSupportPhy.GetListeLiaisonsLog().size()==0)
                  {
                  return (int) (2.0/(Math.random()+0.1));
                  }
            else
                  {
                  Vector w_Liste;
                  liaisonVCL w_RéfLiaisonVcl;
                  w_Liste=RéfSupportPhy.GetListeLiaisonsLog();
                  w_RéfLiaisonVcl=(liaisonVCL)
                        w_Liste.elementAt(w_Liste.size()-1);
                  return w_RéfLiaisonVcl.GetNuméroLiaison()+1;
                  }
            }
      synchronized int CalculVPI()
            {
            if (RéfSupportPhy.GetListeLiaisonsLog().size()==0)
                  {
                  return (int) (1.0/(Math.random()+0.1));
                  }
            else
                  {
                  Vector w_Liste;
                  liaisonVCL w_RéfLiaisonVcl;
                  w_Liste=RéfSupportPhy.GetListeLiaisonsLog();
                  w_RéfLiaisonVcl=(liaisonVCL)
                        w_Liste.elementAt(w_Liste.size()-1);
                  return w_RéfLiaisonVcl.GetVPI();
                  }
            }
      synchronized boolean DataTRANSPORT(equipementATM
                  f_RéfEquipementSource, celluleATM f_RéfCellule)
      /* service fourni à un équipement */
            {
            //
            AddRéfTrame(f_RéfCellule);
            //
            Thread w_RéfThread=
                  GetRéfLiaisonPhy().GetRéfObjetGraphique(
                  ).AnimationTrame(f_RéfEquipementSource,
                  f_RéfCellule.GetRéfObjetGraphique());
            try {
                  traiteDataVcl ActionLOG=new traiteDataVcl (this,
                                    f_RéfCellule, w_RéfThread);
                  ActionLOG.start();
                  }
            catch(OutOfMemoryError signal)
                  {
                  System.out.println("** ERREUR **
                        (liaisonVCL - DataTRANSPORT) Out of memory :");
                  System.out.println("    "+signal);
                  return false;
                  }
            return true;
            }
private class traiteDataVcl extends Thread
      {
```

```
      liaisonVCL RéfLiaisonVcl;
      celluleATM RéfCellule;
      Thread RéfThreadAnimation;
/* constructeur */
      traiteDataVcl(liaisonVCL f_RéfLiaisonVcl, celluleATM f_RéfCellule,
                        Thread f_RéfThread)
            {
            RéfLiaisonVcl=f_RéfLiaisonVcl;
            RéfCellule=f_RéfCellule;
            RéfThreadAnimation=f_RéfThread;
            RéfCellule.SetRéfLocalisation(RéfLiaisonVcl);
            }
      public void run()
            {
            this.setPriority(NORM_PRIORITY);
            sousReseauATM w_RéfSousRéseau=(sousReseauATM)
                  RéfLiaisonVcl.GetRéfSousRéseau();
            w_RéfSousRéseau.AddRéfThread(this);
            //
            while (RéfThreadAnimation.isAlive())
                  {
                  if (!w_RéfSousRéseau.isThreadActif(this))
                        {
      RéfLiaisonVcl.GetRéfSupportPhy().DelRéfLiaisonLog(RéfLiaisonVcl);
                        break;
                        }
                  w_RéfSousRéseau.AbandonCPU();
                  }
            //
            DelRéfTrame(RéfCellule);
            if (w_RéfSousRéseau.isThreadActif(this))
                  {
                  ((equipementATM)
            RéfLiaisonVcl.GetRéfEquipementCible()).DataGET(RéfCellule);
                  }
            }
      }
      }
/*=====================================================================*/

package Reseau.Atm;
/* Classe celluleATM                    */
import java.util.*;
import java.awt.*;
import Presentation.P_trame;
import Reseau.trame;
/*=============================================================*/
public final class celluleATM extends trame
      {
      // les champs d'un objet celluleATM
      liaisonVCL RéfLiaisonVcl;
      int PT;
      int CLP;
      /* les méthodes de la classe */
      //
      celluleATM(liaisonVCL f_RéfLiaisonVcl, int f_Pt, int f_Clp, String
            f_Donnée)
            {
            RéfLiaisonVcl=f_RéfLiaisonVcl;
            PT=f_Pt;
            CLP=f_Clp;
```

```
            Donnée=f_Donnée;
            RéfObjetGraphique=null;
            RéfLiaisonVcl.GetRéfLiaisonVcc().AddRéfTrame(this);
            }
      synchronized void SetRéfLiaisonVcl(liaisonVCL f_RéfLiaisonVcl)
            {
            RéfLiaisonVcl=f_RéfLiaisonVcl;
            }
      public synchronized liaisonVCL GetRéfLiaisonVcl()
            {
            return RéfLiaisonVcl;
            }
      public synchronized liaisonVCC GetRéfLiaisonVcc()
            {
            return RéfLiaisonVcl.GetRéfLiaisonVcc();
            }
      public synchronized int GetVPI()
            {
            return RéfLiaisonVcl.GetVPI();
            }
      public synchronized int GetNuméroVcl()
            {
            return RéfLiaisonVcl.GetNuméroLiaison();
            }
      synchronized void SetCLP(int f_CLP)
            {
            CLP=f_CLP;
            }
      public synchronized int GetCLP()
            {
            return CLP;
            }
      public synchronized int GetPT()
            {
            return PT;
            }
      public synchronized machineHote GetRéfMachineHôteCible()
            {
            return GetRéfLiaisonVcc().GetRéfMachineHôteCible();
            }
      public synchronized machineHote GetRéfMachineHôteSource()
            {
            return GetRéfLiaisonVcc().GetRéfMachineHôteSource();
            }
      }
/*=====================================================================*/

package Reseau.Atm;
import java.util.Vector;
import java.awt.*;
import Presentation.P_couche;
import Reseau.couche;
import Reseau.equipement;
/*=============================================================*/

public final class coucheAPL extends couche
      {
      /* les méthodes de la classe */
      public coucheAPL(String f_NomCouche, equipement f_RéfEquipement)
            {
            NomCouche=f_NomCouche;
```

```
            RéfEquipement=f_RéfEquipement;
            NiveauOSI=7;
            RéfObjetGraphique=null;
            ListeCouchesSup=new Vector();
            ListeCouchesInf=new Vector();
            }
      }
/*=====================================================================*/

package Reseau.Atm;
import java.util.Vector;
import java.awt.*;
import Presentation.P_couche;
import Reseau.couche;
import Reseau.equipement;
import Reseau.trame;
/*=====================================================================*/
public final class coucheAAL extends couche
      {
      /* les méthodes de la classe */
      public coucheAAL(String f_NomCouche, equipement f_RéfEquipement)
            {
            NomCouche=f_NomCouche;
            RéfEquipement=f_RéfEquipement;
            NiveauOSI=4;
            RéfObjetGraphique=null;
            ListeCouchesSup=new Vector();
            ListeCouchesInf=new Vector();
            }
      synchronized boolean DataIND(String f_Donnée)
      /* service fourni à la couche ATM */
            {
            return true;
            }
      public synchronized boolean DataREQ(liaisonVCC f_RéfLiaisonVcc,
             int f_Volume)
      /* service fourni à la couche APL */
            {
            traiteDataAAL ActionAAL=new traiteDataAAL(f_RéfLiaisonVcc,
                                                      f_Volume);
            ActionAAL.start();
            return true;
            }
private class traiteDataAAL extends Thread
      {
      liaisonVCC RéfLiaisonVcc;
      int NombreCellules;
/* constructeur */
      traiteDataAAL(liaisonVCC f_RéfLiaisonVcc, int f_Volume)
            {
            NombreCellules=f_Volume;
            RéfLiaisonVcc=f_RéfLiaisonVcc;
            }
      public void run()
            {
            this.setPriority(NORM_PRIORITY);
            sousReseauATM w_RéfSousRéseau=(sousReseauATM)
                        GetRéfSousRéseau();
            w_RéfSousRéseau.AddRéfThread(this);
            coucheATM w_RéfCouche = (coucheATM)
                  GetRéfEquipement().GetCoucheWithNom("ATM");
```

```
            //
            int i=0;
            //
            while (i<NombreCellules)
                  {
                  while (w_RéfSousRéseau.isThreadSuspended())
                        {
                        w_RéfSousRéseau.AbandonCPU();
                        }
                  //
                  if (w_RéfSousRéseau.isThreadActif(this))
                        {
                        w_RéfCouche.DataREQ(RéfLiaisonVcc, "hhhh");
                        }
                  else
                        {
                        break;
                        }
                  //
                  i++;
                  w_RéfSousRéseau.AbandonCPU(600);
                  }
            //
            while (RéfLiaisonVcc.GetListeTrames().size()!=
                  RéfLiaisonVcc.GetNbreCellulesRecues())
                  {
                  w_RéfSousRéseau.AbandonCPU();
                  }
            //
            w_RéfSousRéseau.DelRéfLiaisonVcc(RéfLiaisonVcc);
            }
      }
}
/*=====================================================================*/

package Reseau.Atm;
import java.util.Vector;
import java.awt.*;
import Presentation.P_couche;
import Presentation.P_sousReseauATM;
import Reseau.couche;
import Reseau.equipement;
import Reseau.trame;
/*=================================================================*/
/* Classe coucheATM                    */
public final class coucheATM extends couche
      {
      //traiteDataAtm ActionATM;
      /* les méthodes de la classe */
      coucheATM(String f_NomCouche, equipement f_RéfEquipement)
            {
            NomCouche=f_NomCouche;
            RéfEquipement=f_RéfEquipement;
            NiveauOSI=3;
            RéfObjetGraphique=null;
            ListeCouchesSup=new Vector();
            ListeCouchesInf=new Vector();
            }
      //
      synchronized boolean DataIND(celluleATM f_RéfCellule)
      /* service fourni à la couche physique */
```

```
        {
        switch (this.RéfEquipement.GetTypeEquipement())
              {
              //0=machineHôte, 1=commutateur, 2=routeur

        /*-------------------------------------------------*/
              case 0 :
                    try {
                          traiteDataAtm ActionATM=new
                                traiteDataAtm(this, f_RéfCellule);
                          ActionATM.start();
                          }
                    catch(OutOfMemoryError signal)
                          {
                          System.out.println("** ERREUR **
                                (coucheATM - DataIND) Out of memory :");
                          System.out.println("    "+signal);
                          return false;
                          }
                    break;
        /*-------------------------------------------------*/
              case 1 : //coucheATM d'un commutateur
                    /* consultation de la table de commutation */
                    /* Mise à jour de la RéfLiaisonVcl */
                    f_RéfCellule.SetRéfLiaisonVcl(
                          ((commutateurATM)
                    RéfEquipement).GetRéfLiaisonVclSortant(
                          f_RéfCellule.GetRéfLiaisonVcl()));
                    if (f_RéfCellule.GetRéfLiaisonVcl()==null)
                          {
                          return false;
                          }
                    try {
                          traiteDataAtm ActionATM=new
                                traiteDataAtm(this, f_RéfCellule);
                          ActionATM.start();
                          }
                    catch(OutOfMemoryError signal)
                          {
                          System.out.println("** ERREUR **
                                (coucheATM - DataIND) Out of memory :");
                          System.out.println("    "+signal);
                          return false;
                          }
                    break;
        /*-------------------------------------------------*/
              }
        return true;
        }
  synchronized boolean DataREQ(liaisonVCC f_RéfLiaisonVcc,
                               String f_Donnée)
  /* service fourni à la couche AAL */
        {
        try
              {
              traiteDataAtm ActionATM=new traiteDataAtm(this,
                                f_RéfLiaisonVcc, f_Donnée);
              ActionATM.start();
              }
        catch(OutOfMemoryError signal)
              {
```

```
                System.out.println("** ERREUR **
                      (coucheATM - DataREQ) Out of memory :");
                System.out.println("    "+signal);
                return false;
                }
            return true;
            }
private class traiteDataAtm extends Thread
      {
      liaisonVCC RéfLiaisonVcc;
      celluleATM RéfCellule;
      coucheATM RéfCouche;
      char TypeTraitement;  //R=REQ, I=IND
/* constructeur 1 => lors d'un DataIND */
      traiteDataAtm(coucheATM f_RéfCouche, celluleATM f_RéfCellule)
            {
            RéfCouche=f_RéfCouche;
            RéfCellule=f_RéfCellule;
            TypeTraitement='I'; //IND
            RéfCellule.SetRéfLocalisation(RéfCouche, "1");
            }
/* constructeur 2 => lors d'un DataREQ */
      traiteDataAtm(coucheATM f_RéfCouche, liaisonVCC f_RéfLiaisonVcc,
                  String f_Donnée)
            {
            RéfCouche=f_RéfCouche;
            /* fabrication d'une cellule */
            RéfLiaisonVcc=f_RéfLiaisonVcc;
            RéfCellule=new celluleATM((liaisonVCL)
                  RéfLiaisonVcc.GetRéfFirstLiaisonVcl(), 0,1,f_Donnée);
            TypeTraitement='R'; //REQ
            RéfCellule.SetRéfLocalisation(RéfCouche, "0");
            }
      public void run()
            {
            this.setPriority(NORM_PRIORITY);
            sousReseauATM w_RéfSousRéseau=(sousReseauATM)
                  RéfCouche.GetRéfSousRéseau();
            w_RéfSousRéseau.AddRéfThread(this);
            w_RéfSousRéseau.GoSleep();
            switch (TypeTraitement)
                  {
                  case 'R' :
                  /*-------------------------------------------------*
                              SERVICE    data  R E Q U E S T
                  *-------------------------------------------------*/
                        //
                        P_sousReseauATM pMonRéseau = (P_sousReseauATM)
                              w_RéfSousRéseau.GetRéfObjetGraphique();
                        pMonRéseau.TrameGraphique(RéfCellule);
                        //
                        while (w_RéfSousRéseau.isThreadSuspended())
                              {
                              w_RéfSousRéseau.AbandonCPU();
                              }
                        if (w_RéfSousRéseau.isThreadActif(this))
                              {
                              ((couchePHY)
                  RéfCouche.GetCoucheWithNom(
            RéfCouche.GetListeCouchesInf(),"PHY")).DataREQ(RéfCellule,
                                    RéfCellule.GetRéfLiaisonVcl());
```

```
                        }
                    break;

                case 'I' :
                /*-----------------------------------------*
                    SERVICE   data  I N D I C A T I O N
                rmq : un DataIND dans un commutateur change le VCI et
                    fait un DataREQ de la couche physique.
                *-----------------------------------------*/
                    while (w_RéfSousRéseau.isThreadSuspended())
                        {
                        w_RéfSousRéseau.AbandonCPU();
                        }
                    if (w_RéfSousRéseau.isThreadActif(this))
                        {
                        if (RéfCellule.GetRéfMachineHôteCible()==
                              RéfCouche.GetRéfEquipement())
                              {
                    RéfCellule.GetRéfLiaisonVcc().CelluleArrivée();
                              ((coucheAAL)
                    RéfCouche.GetCoucheWithNom(GetListeCouchesSup(),
                              "AAL")).DataIND(RéfCellule.GetDonnée());
                              }
                        else
                              {
                              ((couchePHY)
            RéfCouche.GetCoucheWithNom(RéfCouche.GetListeCouchesInf(),
            "PHY")).DataREQ(RéfCellule, RéfCellule.GetRéfLiaisonVcl());
                              }
                        }
                }
                }
        }
        }
/*=====================================================================*/

package Reseau.Atm;
/* Classe couchePHY                    */
import java.util.Vector;
import java.awt.*;
import Presentation.P_couche;
import Reseau.couche;
/*=============================================================*/
public final class couchePHY extends couche
        {
        // les champs d'un objet couchePHY
        /* les méthodes de la classe */
        //
        couchePHY(String f_NomCouche, equipementATM f_RéfEquipement)
                {
                NomCouche=f_NomCouche;
                RéfEquipement=f_RéfEquipement;
                NiveauOSI=1;
                RéfObjetGraphique=null;
                ListeCouchesSup=new Vector();
                ListeCouchesInf=null;
                }
        synchronized boolean DataIND(celluleATM f_RéfCellule)
                /* service fourni à son commutateur (le contenant) */
                {
                try {
```

```
                traiteDataPhy ActionPHY=new traiteDataPhy(this,
                                f_RéfCellule);
                ActionPHY.start();
                }
        catch(OutOfMemoryError signal)
                {
                System.out.println("** ERREUR **
                      (couchePHY - DataIND) Out of memory :");
                System.out.println("    "+signal);
                }
        return true;
        }
synchronized boolean DataREQ(celluleATM f_RéfCellule, liaisonVCL
                                f_RéfLiaisonVcl)
        /* service fourni à la couche ATM */
        {
        try {
                traiteDataPhy ActionPHY=new traiteDataPhy(this,
                                f_RéfCellule, f_RéfLiaisonVcl);
                ActionPHY.start();
                }
        catch(OutOfMemoryError signal)
                {
                System.out.println("** ERREUR **
                       (couchePHY - DataREQ) Out of memory :");
                System.out.println("    "+signal);
                }
        return true;
        }
private class traiteDataPhy extends Thread
        {
        celluleATM RéfCellule;
        couchePHY RéfCouche;
        liaisonVCL RéfLiaisonVcl;
        char TypeTraitement;
        /* constructeur lors d'une IND */
        traiteDataPhy(couchePHY f_RéfCouche, celluleATM f_RéfCellule)
                {
                RéfCouche=f_RéfCouche;
                RéfCellule=f_RéfCellule;
                RéfLiaisonVcl=null;
                TypeTraitement='I';
                RéfCellule.SetRéfLocalisation(RéfCouche, "1");
                }
        /* constructeur lors d'un REQ */
        traiteDataPhy(couchePHY f_RéfCouche, celluleATM f_RéfCellule,
                                liaisonVCL f_RéfLiaisonVcl)
                {
                RéfCouche=f_RéfCouche;
                RéfCellule=f_RéfCellule;
                RéfLiaisonVcl=f_RéfLiaisonVcl;
                TypeTraitement='R';
        RéfCellule.SetRéfLocalisation(RéfCouche, "0");
                }
        public void run()
                {
                this.setPriority(NORM_PRIORITY);
                sousReseauATM w_RéfSousRéseau=(sousReseauATM)
                            RéfCouche.GetRéfSousRéseau();
                w_RéfSousRéseau.AddRéfThread(this);
                w_RéfSousRéseau.GoSleep();
```

```
                if (TypeTraitement=='R')
                      {
                      while (w_RéfSousRéseau.isThreadSuspended())
                            {
                            w_RéfSousRéseau.AbandonCPU();
                            }
                      if (w_RéfSousRéseau.isThreadActif(this))
                            {
                            ((equipementATM)
                      RéfCouche.GetRéfEquipement()).DataSEND(RéfCellule,
                      RéfLiaisonVcl);
                            }
                      }
                else
                      {
                      while (w_RéfSousRéseau.isThreadSuspended())
                            {
                            w_RéfSousRéseau.AbandonCPU();
                            }
                      if (w_RéfSousRéseau.isThreadActif(this))
                            {
                            ((coucheATM)RéfCouche.GetCoucheWithNom(
                                  RéfCouche.GetListeCouchesSup(),
                            "ATM")).DataIND(RéfCellule);
                            }
                      }

                }
          }
    }
```

III. Module *Présentation*

```
package Presentation;

import java.awt.*;
/*============================================================*/
/* Classe AttributsFormes                    */

public class AttributsFormes

    {
    float AxeX;
    float AxeY;
    float AxeZ;

    Color Couleur;

    int RotationAxeX;
    int RotationAxeY;
    int RotationAxeZ;
    float Inclinaison;

/*============================================================*/

    public synchronized void SetAxeX(float f_AxeX)
    {
        AxeX=f_AxeX;
    }
```

```
	public synchronized void SetAxeY(float f_AxeY)
	{
		AxeY=f_AxeY;
	}

	public synchronized void SetAxeZ(float f_AxeZ)
	{
		AxeZ=f_AxeZ;
	}

	public synchronized void SetCouleur(Color f_Couleur)
	{
		Couleur=f_Couleur;
	}

	public synchronized void SetRotationAxeX(int f_RotationAxeX)
	{
		RotationAxeX=f_RotationAxeX;
	}

	public synchronized void SetRotationAxeY(int f_RotationAxeY)
	{
		RotationAxeY=f_RotationAxeY;
	}

	public synchronized void SetRotationAxeZ(int f_RotationAxeZ)
	{
		RotationAxeZ=f_RotationAxeZ;
	}

	public synchronized void SetInclinaison(float f_Inclinaison)
	{
		Inclinaison=f_Inclinaison;
	}

/*============================================================*/

	public synchronized float GetAxeX()
	{
		return AxeX;
	}

	public synchronized float GetAxeY()
	{
		return AxeY;
	}

	public synchronized float GetAxeZ()
	{
		return AxeZ;
	}

	public synchronized float GetRouge()
	{
		float w_rouge;

		w_rouge=(float)(Couleur.getRed())/(float)255;
		return w_rouge;
	}

	public synchronized float GetVert()
```

```
	{
		float w_vert;

		w_vert=(float)(Couleur.getGreen())/(float)255;
		return w_vert;
	}

	public synchronized float GetBleu()
	{
		float w_bleu;

		w_bleu=(float)(Couleur.getBlue())/(float)255;
		return w_bleu;
	}

	public synchronized Color  GetCouleur()
	{
		return Couleur;
	}

	public synchronized int GetRotationAxeX()
	{
		return RotationAxeX;
	}

	public synchronized int GetRotationAxeY()
	{
		return RotationAxeY;
	}

	public synchronized int GetRotationAxeZ()
	{
		return RotationAxeZ;
	}

	public synchronized float GetInclinaison()
	{
		return Inclinaison;
	}

}

/*============================================================*/
/*============================================================*/

package Presentation;

import Presentation.AttributsFormes;
import java.awt.*;

/*============================================================*/
/* Classe AttributsBoite                    */

public class AttributsBoite extends AttributsFormes
{
	float TailleX;
	float TailleY;
	float TailleZ;

	public AttributsBoite()
	{
```

```
	}

	public AttributsBoite(float f_AxeX,float f_AxeY, float f_AxeZ, Color
	f_Couleur, float f_TailleX,float f_TailleY,float f_TailleZ)
	{
		AxeX=f_AxeX;
		AxeY=f_AxeY;
		AxeZ=f_AxeZ;

		Couleur=f_Couleur;

		RotationAxeX=0;
		RotationAxeY=0;
		RotationAxeZ=1;
		Inclinaison=0;

		TailleX=f_TailleX;
		TailleY=f_TailleY;
		TailleZ=f_TailleZ;
	}

	//--------Fin du constructeur
	/*============================================================*/

	public synchronized void SetTailleX(float f_TailleX)
	{
		TailleX=f_TailleX;
	}

	public synchronized void SetTailleY(float f_TailleY)
	{
		TailleY=f_TailleY;
	}

	public synchronized void SetTailleZ(float f_TailleZ)
	{
		TailleZ=f_TailleZ;
	}

	public synchronized float GetTailleX()
	{
		return TailleX;
	}

	public synchronized float GetTailleY()
	{
		return TailleY;
	}

	public synchronized float GetTailleZ()
	{
		return TailleZ;
	}
}

/*============================================================*/
/*============================================================*/

 package Presentation;

import Presentation.AttributsFormes;
```

```
Group {
      children[
            NavigationInfo{
                  type "EXAMINE"
            }

            DEF Entree Viewpoint{
                  description "Vue Entrée"
            }

            DEF Haut Viewpoint{
                  position 0 20 -7.5
                  orientation 1 0 0 -1.57
                  description "Vue Aérienne"
            }

            Transform{
                  rotation 1 0 0 -0.5
                  children[
                        Transform{
                              rotation 0 0 1 .33
                              children[
                                    DEF Perspective Viewpoint{
                                          position 18 3 10
                                          orientation 0 1 0 .86325
                                          description "Vue en perspective"
                                    }
                              ]
                        }
                  ]
            }

      ]
}

DEF lampe PointLight{
      location 0 15 -7.5
      intensity .3
      color 1 1 1
      ambientIntensity .5
}
```

Glossaire

ATM	*Asynchronous Transfer Mode* ou Mode de Transfert Asynchrone, une technique de commutation où l'information est encapsulée dans des petits paquets (cellules) et qui permet d'atteindre de hauts débits. ATM est la technologie de commutation retenue pour le RNIS LB.
Cellule	un paquet de taille fixe comportant 48 octets de données et 5 octets d'en-tête.
Classe	un type abstrait spécifiant les attributs, les opérations applicables à ses instances et le mécanisme de création de ses instances. Les instances d'une classe sont appelées des objets.
Méthode	Une unité de comportement d'un objet définie dans sa classe d'appartenance. Elle est une action élémentaire applicable à un objet et possède une signature. Une signature est l'ensemble formé par le nom de la méthode, le type du résultat et la liste de paramètres.
Multimédia	un nom commun pour désigner les applications utilisant plusieurs médias simultanément. Ces médias peuvent être du texte, des images, la voix, la vidéo, etc.
Objet	Une entité identifiable à laquelle sont attachées des opérations (méthodes) et un ensemble d'attributs (état de l'objet).
Port	une interface physique d'une ligne dans un équipement.
Protocole	une séquence de règles à suivre dans un échange d'informations.
Protocole de routage	un mécanisme permettant, lors d'une demande de connexion, de déterminer une route reliant l'appelant et son correspondant.
PVC	*Permanent Virtual Connection* ou Circuit Virtuel Permanent, un: circuit virtuel établi d'une manière permanente entre deux équipements terminaux.
RNIS LB	*RNIS à Large Bande*, la génération à grande vitesse du réseau RNIS, prévu pour intégrer tous les services de communication dans un réseau unique. Il est basé sur la technologie ATM.
SVC	*Switched Virtual Connection* ou Circuit Virtuel Commuté, un circuit virtuel établi et libéré à l'initiative d'un des correspondants.
VCC	*Virtual Channel Connection*, aussi appelée *ATM connection*, qui fournit un chemin de bout en bout assurant le transfert unidirectionnel ordonné mais non garanti de cellules entre utilisateurs de la couche ATM.

VCL — *Virtual Channel Link*, un moyen de transfert unidirectionnel de cellules entre un point où un VCI (*Virtual Channel Identifier*) est assigné à une cellule et un point où cette valeur est supprimée ou modifiée. Ces points sont des commutateurs ou des équipements terminaux.

VPC — *Virtual Path Connexion*, un moyen de transfert ordonné mais non garanti de cellules entre deux points où les VCI (*Virtual Channel Identifier*) sont créés ou modifiés. Ces points sont des commutateurs ou des équipements terminaux.

VPL — *Virtual Path Link*, un moyen de transfert unidirectionnel de cellules entre un point où un VPI (*Virtual Path Identifier*) est assigné à une cellule et un point où cette valeur est supprimée ou modifiée. Ces points sont des équipements terminaux, des commutateurs ou des brasseurs. A l'intérieur d'un lien VPL, le VPI n'est pas modifié.

VRML — *Virtual Reality Modeling Language*, un langage de modélisation de réalité virtuelle en trois dimensions facilement intégrable dans des pages HTML.

BIBLIOGRAPHIE

Programmation

Livres

[BOUZE97]
M. Bouzeghoub, G. Gardarin et P. Valduriez
LES OBJETS
Eyrolles, Avril 1997

Langage Java

❑ Livres

[ANUFF96]
Ed Anuff
PROGRAMMER EN JAVA
Sybex, Septembre 1996

[BONJOU96]
M. Bonjour, G. Falquet, J. Guyot et A. Le Grand
JAVA : DE L'ESPRIT A LA METHODE
International Thomson Publishing France, Août 1996

❑ Sites web

☆ Le langage JAVA
http://java.sun.com
http://poseidon.etmn.cifom.ch/java/Java.htm
http://www.mis.enac.fr/~danel/java/java.html

☆ Writing Java Programs: Table of Contents
http://java.sun.com/docs/books/tutorial/java/TOC.html

☆ Online Java Documentation
http://crystal.clare.cam.ac.uk/java/java.applet.html

☆ Java Concurrent Programming
http://www.netlink.co.uk/users/kcs/java/javathreads
http://java.sun.com/docs/books/tutorial/java/threads/state.html

☆ The Java(TM) Boutique: A Collection of Java Applets
http://javaboutique.internet.com/index.html

☆ FAQ sur Java
http://sunsite.unc.edu/javafaq/javafaq.html

Langage de modélisation VRML

❑ Livres

[HARTMAN96]
Jed Hartman et Josie Wernecke (Silicon Graphics)
THE VRML 2.0 HANDBOOK BUILDING MOVING WORLDS ON THE WEB
Addisson Wesley Developpers Press, Août 1996

[LEA96]
Rodger Lea, Kouichi Matsuda et Ken Myashita

JAVA FOR 3D AND VRML WORLDS
New Riders, 1996

❑ Sites web

☆ consortium VRML
http://www.vrml.org

☆ Silicon Graphics incorporation
http://vrml.sgi.com

☆ Le site VRML qui contient de nombreux articles récents
http://www.vrmlsite.com

☆ VRML Repository
http://www.sdsc.edu.com

☆ Le FAQ de l'EAI
http://www.tomco.net/~raf/faqs/eaifaqs.html

☆ Hermetica
http://www.hermetica.com/technological/java/jvrml

☆ Un site plein de liens et d'historique
http://hiwaay.net/~cripsel/vrml/

☆ FAQ sur Java/JavaScript/VRML
http://www.aereal.com/faq/#faq

Télécommunications et Réseaux

❑ Livres

[TANEN97]
A. Tanenbaum
RESEAUX, 3ème édition
InterEditions, Juillet 1997

[MELIN97]
J.-L. Mélin
PRATIQUE DES RESEAUX ATM,
Eyrolles, Juin 1997

[VANBA95]
Philippe van Bastelaer, Véronique Nachtergaele
Cours de téléinformatique et réseaux, cours de deuxième Licence et Maîtrise en informatique
Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix, 1995

Divers

[UNIVER95]
EAO, Encyclopaedia Universalis France, 1995

[CHAVA96]
Pierre Chavanne,
Une extension du didacticiel TcpIp : adjonction du protocole ATM
Mémoire présenté en vue de l'obtention du grade de licencié et maître en informatique
Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix,1996

[JVDKT95]
Jean Vanderdonkt,
Interface Homme/Machine, cours de seconde licence et maîtrise en informatique
Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix, 1995

L'ABC des réseaux, http://www.culture.fr/culture/dglf/abc-dglf.htm

❑ ECL
http://www.ecl-lyon.fr
http://www.spi.cnrs-dir.fr
http://www.cnrs.fr/
http://www.ictt.fr/

❑ CNET
http://www.cnet.fr

❑ LICEF
http://www.teluq.uquebec.ca/Alice/developp/m_licef.htm

```
import java.awt.*;

/*============================================================*/
/* Classe AttributsBoite                    */

public class AttributsLiens extends AttributsFormes
{
      float Rayon;
      float Longueur;


      public AttributsLiens(float f_AxeX,float f_AxeY, float f_AxeZ,
  Color f_Couleur, int f_RotationAxeX,int f_RotationAxeY,int
f_RotationAxeZ, float f_Inclinaison,float f_Rayon,float f_Longueur)
      {
            AxeX=f_AxeX;
            AxeY=f_AxeY;
            AxeZ=f_AxeZ;

            Couleur=f_Couleur;

            RotationAxeX=f_RotationAxeX;
            RotationAxeY=f_RotationAxeY;
            RotationAxeZ=f_RotationAxeZ;
            Inclinaison=f_Inclinaison;

            Rayon=f_Rayon;
            Longueur=f_Longueur;

      }

/*============================================================*/

      public synchronized  void SetRayon(float f_Rayon)
      {
            Rayon=f_Rayon;
      }

      public synchronized  void SetLongueur(float f_Longueur)
      {
            Longueur=f_Longueur;
      }


      public synchronized  float GetRayon()
      {
            return Rayon;
      }

      public synchronized  float GetLongueur()
      {
            return Longueur;
      }


}
/*============================================================*/
/*============================================================*/

package Presentation;
```

```
import java.awt.*;
import Reseau.sousReseau;
import Reseau.equipement;

      //Importation des package nécessaires à l'interaction avec VRML

import vrml.external.field.*;
import vrml.external.Node;
import vrml.external.Browser;
import vrml.external.exception.*;

/*============================================================*/
/* Classe P_sousReseau                    */

public abstract class P_sousReseau
{
      sousReseau RéfSousRéseau;
      EventInMFNode Ajout;
      EventInMFNode Suppr;
      EventInMFNode AjoutCell;
      EventInMFNode SupprCell;
      Browser browser;

/*============================================================*/
/*============================================================*/

      public abstract void AdapterGraphique();

      public synchronized void DessineToi()
      {
            int i=0;
            while (i<RéfSousRéseau.GetListeEquipements().size())
            {
            ((equipement) RéfSousRéseau.GetListeEquipements().elementAt(i)
                        ).GetRéfObjetGraphique().DessineToi();
            i++;
            }
      }

      public synchronized void SetRéfSousRéseau(sousReseau f_RéfSousReseau)
      {
            RéfSousRéseau=f_RéfSousReseau;
      }

      public synchronized sousReseau GetRéfSousRéseau()
      {
            return RéfSousRéseau;
      }

      public synchronized EventInMFNode GetAjout()
      {
            return Ajout;
      }

      public synchronized EventInMFNode GetSuppr()
      {
            return Suppr;
      }

      public synchronized EventInMFNode GetAjoutCell()
      {
```

```
            return AjoutCell;
      }

      public synchronized EventInMFNode GetSupprCell()
      {
            return SupprCell;
      }

      public synchronized Browser GetBrowser()
      {
            return browser;
      }
}

/*============================================================*/
/*============================================================*/

package Presentation;

import java.awt.*;
import java.util.*;

import Reseau.Atm.sousReseauATM;
import Reseau.equipement;
import Reseau.couche;
import Reseau.Atm.couchePHY;
import Reseau.Atm.coucheATM;
import Reseau.Atm.coucheAAL;
import Reseau.Atm.coucheAPL;
import Reseau.Atm.celluleATM;

//Importation des package nécessaires à l'interaction avec VRML

import vrml.external.field.*;
import vrml.external.Node;
import vrml.external.Browser;
import vrml.external.exception.*;

/*============================================================*/
/* Classe P_sousReseauATM                    */

public class P_sousReseauATM extends P_sousReseau
{

      public P_sousReseauATM (sousReseauATM f_RéfSousRéseau)
      {
            RéfSousRéseau=f_RéfSousRéseau;
            RéfSousRéseau.SetRéfObjetGraphique(this);
      }

      public P_sousReseauATM (sousReseauATM f_RéfSousRéseau,Browser
      f_browser, EventInMFNode f_Ajout, EventInMFNode f_Suppr,
      EventInMFNode f_AjoutCell, EventInMFNode f_SupprCell)
      {
            RéfSousRéseau=f_RéfSousRéseau;
            RéfSousRéseau.SetRéfObjetGraphique(this);
            browser=f_browser;
            Ajout=f_Ajout;
            Suppr=f_Suppr;
            AjoutCell=f_AjoutCell;
            SupprCell=f_SupprCell;
```

```
}

public synchronized void TrameGraphique(celluleATM f_Trame)
{           //création des représentation graphique des cellules
            //attribution de couleurs aux cellules

      P_trame w_ObjetGraphiqueTrame=new P_trame(f_Trame);

      if (f_Trame.GetRéfLiaisonVcc().GetListeTrames().size()==1)
      {
            w_ObjetGraphiqueTrame.GetRéfAttributs().SetCouleur(
                  new Color((float)(1.0), (float) Math.random(),
                                          (float) (0.39215686)));
            //choix des couleurs limité aux couleurs chaudes (rouge,
            orange, jaune)
      }

      else
      {
            P_trame w_Objet0;

            w_Objet0=
            ((celluleATM)
            f_Trame.GetRéfLiaisonVcc().GetListeTrames().elementAt(0)
                                          ).GetRéfObjetGraphique();

            w_ObjetGraphiqueTrame.GetRéfAttributs().SetCouleur(
                  w_Objet0.GetRéfAttributs().GetCouleur());
      }
}

public synchronized void AdapterGraphique()
{
      for(int i=0;i<RéfSousRéseau.GetListeEquipements().size();i++)
      {
            equipement w_EquiTemp=
            (equipement)
                  RéfSousRéseau.GetListeEquipements().elementAt(i);

            switch (w_EquiTemp.GetTypeEquipement())
            {
                  case 0: //création des couches de la machine hôte
                        CouchesHote(w_EquiTemp);
                        break;

                  case 1: //création des couches du commutateur
                        CouchesCommu(w_EquiTemp);
                        break;
            }
      }
}

public synchronized void DessineToi()
{
      int i=0;
      while (i<RéfSousRéseau.GetListeEquipements().size())
      {
            ((equipement)
            RéfSousRéseau.GetListeEquipements().elementAt(i))
                        .GetRéfObjetGraphique().DessineToi();
```

```
                i++;
        }
}

private void CouchesHote(equipement f_RéfEquipement)
{
        for(int j=0;j<f_RéfEquipement.GetListeCouches().size();j++)
        {
                couche
                w_CoucheTemp=(couche)f_RéfEquipement.GetListeCouches().
                elementAt(j);

                if(w_CoucheTemp.GetNomCouche()=="PHY")
                {
                        float X=f_RéfEquipement.GetRéfObjetGraphique().
                                                        GetPositionX();
                        float Y=f_RéfEquipement.GetRéfObjetGraphique().
                              GetPositionY();
                        float Z=f_RéfEquipement.GetRéfObjetGraphique().
                                                        GetPositionZ();
                        AttributsBoite w_Attributs1=
                              new AttributsBoite(X,Y,Z,
                                   new Color(136,159,238),
                                    1,(float)(0.5),1);
                              P_couche w_pCouche=new
                              P_couche(w_CoucheTemp,w_Attributs1);
                }
                else
                {
                        if(w_CoucheTemp.GetNomCouche()=="ATM")
                        {
                              float X=f_RéfEquipement.GetRéfObjetGraphique()
                                                        .GetPositionX();
                              float Y=f_RéfEquipement.GetRéfObjetGraphique()
                                          .GetPositionY()
                                          +(float)(0.5);
                              float Z=f_RéfEquipement.GetRéfObjetGraphique()
                                                    .GetPositionZ();
                              AttributsBoite w_Attributs2=new
                                     AttributsBoite(X,Y,Z,
                                           new Color(164,223,165),
                                           1,(float)(.5),1);
                              P_couche w_pCouche1=new
                                       P_couche(w_CoucheTemp,w_Attributs2);
                        }
                        else
                        {
                              if(w_CoucheTemp.GetNomCouche()=="AAL")
                              {
                                    float X=f_RéfEquipement.
                                      GetRéfObjetGraphique().GetPositionX();
                                    float Y=f_RéfEquipement.
                                      GetRéfObjetGraphique().GetPositionY()
                                                    +(float)(1);
                                    float Z=f_RéfEquipement.
                                      GetRéfObjetGraphique().GetPositionZ();

                                   AttributsBoite w_Attributs3=new
                                        AttributsBoite(X,Y,Z,
                                         new Color(149,106,196),
                                         1,(float)(.5),1);
```

```
                                    P_couche w_pCouche2=new
                                     P_couche(w_CoucheTemp,w_Attributs3);
                              }
                              else
                              {
                                    if(w_CoucheTemp.GetNomCouche()=="APL")
                                    {
                                          float X=f_RéfEquipement
                                                .GetRéfObjetGraphique(
                                                ).GetPositionX();
                                          float Y=f_RéfEquipement
                                                .GetRéfObjetGraphique(
                                                ).GetPositionY()
                                                      +(float)(2);
                                          float Z=f_RéfEquipement
                                                .GetRéfObjetGraphique(
                                                ).GetPositionZ();

                                          AttributsBoite w_Attributs4=new
                                             AttributsBoite(X,Y,Z,
                                             New Color(0.75f,0.75f,0.75f),
                                             1,(float)(1.5),1);

                                          P_couche w_pCouche3=new
                                          P_couche(w_CoucheTemp,w_Attributs4
                                          );
                                    }
                              }
                        }
                  }
            }
      }

      private void CouchesCommu(equipement f_RéfEquipement)
      {
            for(int j=0;j<f_RéfEquipement.GetListeCouches().size();j++)
            {
                  couche w_CoucheTemp=
                    (couche)f_RéfEquipement.GetListeCouches().elementAt(j);
                  AttributsBoite w_Attributs=new AttributsBoite();

                  if(w_CoucheTemp.GetNomCouche()=="PHY")
                  {
                        float X=f_RéfEquipement.GetRéfObjetGraphique().
                                                      GetPositionX();
                        float Y=f_RéfEquipement.GetRéfObjetGraphique().
                                                      GetPositionY();
                        float Z=f_RéfEquipement.GetRéfObjetGraphique().
                                                      GetPositionZ();
                        AttributsBoite w_Attributs5=new
                                    AttributsBoite(X,Y,Z,
                                       new Color(136,159,238),
                                       1,(float)(0.5),1);

                        P_couche w_pCouche4=new
                             P_couche(w_CoucheTemp,w_Attributs5);
                  }
                  else
                  {
                        if(w_CoucheTemp.GetNomCouche()=="ATM")
```

```
                        {
                              float X=f_RéfEquipement.GetRéfObjetGraphique()
                                                      .GetPositionX();
                              float Y=f_RéfEquipement.GetRéfObjetGraphique()
                                                      .GetPositionY()
                                                      +(float)(0.5);
                              float Z=f_RéfEquipement.GetRéfObjetGraphique()
                                                      .GetPositionZ();
                              AttributsBoite w_Attributs6=new
                                          AttributsBoite(X,Y,Z,
                                            new Color(164,223,165),
                                            1,(float)(.5),1);
                              P_couche w_pCouche5=new
                                   P_couche(w_CoucheTemp,w_Attributs6);
                        }
                  }
            }
      }
}

/*============================================================*/
/*============================================================*/
package Presentation;

import java.awt.*;
import Reseau.couche;

//Importation des package nécessaires à l'interaction avec VRML

import vrml.external.field.*;
import vrml.external.Node;
import vrml.external.Browser;
import vrml.external.exception.*;

/*============================================================*/
/* Classe P_couche                    */
      public class P_couche
      {
            couche RéfCouche;
      Node[] NoeudVRML;       //variable liée à un objet qui sera //inséré
      ou supprimé de la
                                    //scène VRML
      EventInMFNode Ajout;    //permet l'insertion d'un nouvel objet dans le
                              //monde VRML
      EventInMFNode Suppr;    //permet la suppression d'un élément existant
      Browser browser;
      AttributsBoite RéfAttributs;
      Color AncienneCouleur;


/*============================================================*/
/*============================================================*/

□
      public P_couche (couche f_RéfCouche, AttributsBoite f_RéfAttributs)
□
      {

            RéfCouche=f_RéfCouche;
            RéfCouche.SetRéfObjetGraphique(this);
            browser=RéfCouche.GetRéfEquipement().GetRéfSousRéseau(
```

```
                        ).GetRéfObjetGraphique().browser;
        Ajout=RéfCouche.GetRéfEquipement().GetRéfSousRéseau(
                        ).GetRéfObjetGraphique().Ajout;
        Suppr=RéfCouche.GetRéfEquipement().GetRéfSousRéseau(
                        ).GetRéfObjetGraphique().Suppr;
        RéfAttributs=f_RéfAttributs;
        AncienneCouleur=RéfAttributs.GetCouleur();
    }

    public synchronized void DessineToi()
    {
        try
        {
            NoeudVRML=browser.createVrmlFromString("Transform{ \n"+
                "     translation "+RéfAttributs.GetAxeX()+
                                  " "+RéfAttributs.GetAxeY()+
                                  " "+RéfAttributs.GetAxeZ()+" \n"+
                "     rotation "+RéfAttributs.GetRotationAxeX()+
                                " "+RéfAttributs.GetRotationAxeY()+
                                " "+RéfAttributs.GetRotationAxeZ()+
                              " "+RéfAttributs.GetInclinaison()+" \n"+
                "     children[ \n"+
                "          Shape { \n"+
                "               appearance Appearance{ \n"+
                "                    material Material{ \n"+
                "                         diffuseColor
                                    "+RéfAttributs.GetRouge()+
                                  □
                                     " "+RéfAttributs.GetVert()+
                                   " "+RéfAttributs.GetBleu()+" \n"+
                "                    } \n"+
                "               } \n"+
                "               geometry Box { \n"+
                "                         size"
                                     +RéfAttributs.GetTailleX()+
                                  □
                                     " "    +RéfAttributs.GetTailleY()+
□
                                     " "    +RéfAttributs.GetTailleZ()+
                                     " \n"+
                                □
                "               } \n"+
□
                "          } \n"+
□
                "     ] \n"+
□
                "} \n");
□
        }
□
        catch (InvalidVrmlException e){}
□
        Ajout.setValue(NoeudVRML); //ajout du nœud dans la scène VRML
□

□

□
            //cette deuxième partie de la méthode ajoute dans le
            //monde VRML le sigle se
```

```
                //trouvant sur le haut des commutateurs
            if((RéfCouche.GetNomCouche()=="ATM") &&
               (RéfCouche.GetRéfEquipement().GetTypeEquipement()==1))
            {
                Ajout.setValue(AjoutDessinCommu());

            }

            if(RéfCouche.GetNomCouche()=="APL")
            {
                Ajout.setValue(AjoutDessinHote());
            }
        }

/*=============================================================*/

        public synchronized void ChangeCouleurCouche(couche f_RéfCouche,
                                                     Color f_Couleur)
                                                   □
        {
□
            P_couche w_pCouche=f_RéfCouche.GetRéfObjetGraphique();
            w_pCouche.RéfAttributs.SetCouleur(f_Couleur);
            Suppr.setValue(NoeudVRML);
            w_pCouche.DessineToi();
        }

//-------------------------------------

        private Node[] AjoutDessinCommu()
        {
            String w_NomDessin=QuelDessin(RéfCouche); //appel à une méthode
                                                      //qui détermine
                                                            //le dessin  qui
                                                      //surplombe le commu.
                                                      □

□
            Node[] w_DessinSupérieur=null;
□
            float w_hauteur=RéfAttributs.GetAxeY()+(float)(0.01);
□

□
            try
□
            {
□

        w_DessinSupérieur=browser.createVrmlFromString("Transform{ \n"+
□
                          "     translation "+RéfAttributs.GetAxeX()+
                                    " "+w_hauteur+
                                    " "+RéfAttributs.GetAxeZ()+" \n"+
                          "          children[ \n"+
                          "                Shape { \n"+
                          "                appearance Appearance{ \n"+
                          "                      material Material{ \n"+
                          "                        diffuseColor .07 .49 .13
                                                        \n"+
                                                   □
```

```
                    "                       shininess 0.0 \n"+
                    "                       transparency 0 \n"+
                    "                     } \n"+
                    "                     texture ImageTexture{ \n"+
                    "                       url "+w_NomDessin+"
                                                  \n"+
                    "                       repeatS FALSE \n"+
                    "                       repeatT FALSE \n"+
                    "                     } \n"+
                    "                 } \n"+
                    "                 geometry IndexedFaceSet{ \n"+
                    "                     coord Coordinate{ \n"+
                    "                         point[ -.5  .25  .5,
                                                  #0 \n"+
                    "                                .5  .25  .5,
                                                  #1 \n"+
                    "                                .5  .25 -.5,
                                                  #2 \n"+
                    "                                 -.5  .25 -.5
                                                  #3 \n"+
                    "                             ] \n"+
                    "                     } \n"+
                    "                     coordIndex [0,1,2,3,-1] \n"+
                    "                      texCoord TextureCoordinate{
                                                  \n"+
                    "                         point[0 0, #0 \n"+
                    "                               1 0, #1 \n"+
                    "                               1 1, #2 \n"+
                    "                               0 1  #3 \n"+
                    "                         ] \n"+
                    "                     } \n"+
                    "                     texCoordIndex[0,1,2,3,-1]
                                                  \n"+
                    "                 }     \n"+
                    "        } \n"+
                    "     ] \n"+
                    "} \n");
                }
```

```
                catch (InvalidVrmlException e){}
□
□
                return w_DessinSupérieur;
□
        }
□
□
//-------------------------------------------------

        private Node[] AjoutDessinHote()
        {
                String w_NomDessin=QuelDessin(RéfCouche);
                        //appel à une méthode qui détermine
                □
                        //le dessin  qui surplombe le commu.
□
□
                Node[] w_DessinSupérieur=null;
□
                float w_hauteur=RéfAttributs.GetAxeY()+(float)(0.01);

                try
                {

        w_DessinSupérieur=browser.createVrmlFromString("Transform{ \n"+
                                "     translation "+RéfAttributs.GetAxeX()+
                                                " "+w_hauteur+
                                              " "+RéfAttributs.GetAxeZ()+"
                                                        \n"+
                                                □
                                "     children[ \n"+
□
                                "           Shape { \n"+
□
                                "                 appearance Appearance{ \n"+
                                "                   material Material{
                                                        \n"+
                                                        □
                                "                    diffuseColor .07 .49 .13
                                                        \n"+
                                                      □
                                "                    shininess 0.0 \n"+
□
                                "                    transparency 0 \n"+
□
                                "                      } \n"+
□
                                "                   texture ImageTexture{
                                                        \n"+
                                                      □
                                "                    url "+w_NomDessin+" \n"+
                                "                    repeatS FALSE \n"+
                                "                    repeatT FALSE \n"+
                                "                      } \n"+
                                "                 } \n"+
                                "                 geometry IndexedFaceSet{
```

```
                                                        \n"+
                                                    □
                                "                  coord Coordinate{ \n"+
□
                                "                   point[ -.5  .75  .5,
                                                                #0 \n"+

                                "                          .5  .75  .5,
                                                                #1 \n"+
                                                           □
                                "                          .5  .75 -.5,
                                                                #2 \n"+
                                                           □
                                "                         -.5  .75 -.5
                                                                #3 \n"+
                                                           □
                                "                        ] \n"+
□
                                "                   } \n"+
□
                                "                   coordIndex [0,1,2,3,-1]
                                                           \n"+
                                                    □
                                "                   texCoord
                                                      TextureCoordinate{
                                                                \n"+

                                "                          point[0 0, #0
                                                                \n"+
                                                           □
                                "                                1 0, #1
                                                                \n"+
                                                           □
                                "                                1 1, #2
                                                                \n"+
                                                           □
                                "                                0 1  #3
                                                                \n"+
                                                           □
                                "                          ] \n"+
□
                                "                      } \n"+
□
                                "                   texCoordIndex[0,1,2,3,-1]
                                                                \n"+
                                                           □
                                "                }     \n"+
□
                                "       } \n"+

                                "     ] \n"+
                                "} \n");
                }
                catch (InvalidVrmlException e){}

                return w_DessinSupérieur;
        }

//---------------------------------------------

        private String QuelDessin(couche f_RéfCouche)
```

```
{
        //attention cette méthode est liée à l'implémentation
        //topologique du réseseau
        //elle a dons un lien avec les applets principales.

        float w_place;
        String w_NomDessin=" ";

        w_place=RéfCouche.GetRéfEquipement().GetRéfObjetGraphique(
                                                    ).GetPositionX();
                                                □

□
        if(w_place == (float)(-2.5))
□
        {
□
              w_NomDessin=new String(" \" commu1.png\" ");
        }
        if(w_place == (float)(0.5))
        {
              w_NomDessin=new String(" \" commu2.png\" ");
        }
        if(w_place == (float)(2.5))
        {
              w_NomDessin=new String(" \" commu3.png\" ");
        }
        if(w_place == (float)(-6.5))
        {
              if(RéfCouche.GetRéfEquipement().GetRéfObjetGraphique(
                                    ).GetPositionZ() ==
                              □
                                    (float)(-14.5))
□
              {
□
                    w_NomDessin=new String(" \" HoteA.png\" ");
□
              }
□
              else
□
              {
□
                    w_NomDessin=new String(" \" HoteC.png\" ");
□
              }
□
        }
□
        if(w_place == (float)(6.5))
        {
              if(RéfCouche.GetRéfEquipement().GetRéfObjetGraphique(
                                    ).GetPositionZ() ==
                              □
                                    (float)(-14.5))
□
              {
□
                    w_NomDessin=new String(" \" HoteB.png\" ");
□
```

```
                    }
□
                    else
□
                    {
□
                          w_NomDessin=new String(" \" HoteD.png\" ");
                    }
              }

              return w_NomDessin;
        }
/*============================================================*/

□
        public synchronized void SetRéfCouche(couche f_RéfCouche)
□
        {
□
              RéfCouche=f_RéfCouche;
□
        }
□

□
        public synchronized void SetRéfAttributs(AttributsBoite
                                          f_RéfAttributs)
        {
              RéfAttributs=f_RéfAttributs;
        }

        public synchronized couche GetRéfCouche()
        {
              return RéfCouche;
        }

        public synchronized AttributsBoite GetRéfAttributs()
        {
              return RéfAttributs;
        }

        public synchronized Color GetAncienneCouleur()
        {
              return AncienneCouleur;
        }

}

/*============================================================*/
package Presentation;
□

□
import java.awt.*;
□
import Reseau.equipement;
□
import Reseau.couche;
□

/*============================================================*/
```

```
/* Classe P_equipement                    */

□
public abstract class P_equipement
□
{
□
      equipement RéfEquipement;
□
      float PositionEquipement[]=new float[3];
□
      public synchronized void SetRéfEquipement(equipement f_RéfEquipement)
      {
            RéfEquipement=f_RéfEquipement;
      }

      public synchronized void SetPositionX(float f_PositionX)
      {
            PositionEquipement[0]=f_PositionX;
      }

      public synchronized void SetPositionY(float f_PositionY)
      {
            PositionEquipement[1]=f_PositionY;
      }

      public synchronized void SetPositionZ(float f_PositionZ)
      {
            PositionEquipement[2]=f_PositionZ;
      }

      public synchronized void SetPositionEquipement(float[]
                                                f_PositionEquipement)
                                          □
      {
□
            PositionEquipement=f_PositionEquipement;
□
      }
□


      public synchronized equipement GetRéfEquipement()
      {
            return RéfEquipement;
      }

      public synchronized float GetPositionX()
      {
            return PositionEquipement[0];
      }
      public synchronized float GetPositionY()
      {
            return PositionEquipement[1];
      }
      public synchronized float GetPositionZ()
      {
            return PositionEquipement[2];
      }
      public synchronized float[] GetPositionEquipement()
      {
```

```
            return PositionEquipement;
      }

      void DessineToi()
      {
      }
}

/*================================================================*/
/*================================================================*/
package Presentation;


import java.awt.*;

import Reseau.equipement;

import Reseau.Atm.machineHote;

import Reseau.couche;

/*================================================================*/
/* Classe P_machineHote                    */
public class P_machineHote extends P_equipement

{


      public P_machineHote(machineHote f_RéfEquipement,
                           float[] f_RéfPositionEquipement)

      {

            RéfEquipement=(machineHote) f_RéfEquipement;
            RéfEquipement.SetRéfObjetGraphique(this);
            PositionEquipement=f_RéfPositionEquipement;
      }

      public synchronized void DessineToi()
      {
            int i=RéfEquipement.GetListeCouches().size()-1;
            while (i>-1)
            {
                  ((couche) RéfEquipement.GetListeCouches().elementAt(i)
                        ).GetRéfObjetGraphique().DessineToi();
                  i--;
            }
      }

}

/*================================================================*/
/*================================================================*/

package Presentation;


```

```
□
import java.awt.*;
import java .util.*;
import Reseau.equipement;
import Reseau.Atm.commutateurATM;
import Reseau.couche;
import Reseau.liaisonPHY;
import Reseau.liaisonLOG;
import Reseau.trame;

/*============================================================*/
/* Classe P_commutateurATM                    */

□
public class P_commutateurATM extends P_equipement
□
{
□

□
      public P_commutateurATM(commutateurATM f_RéfEquipement,
□
                                        float[] f_RéfPositionEquipement)
□
      {
□
            RéfEquipement=(commutateurATM) f_RéfEquipement;
            RéfEquipement.SetRéfObjetGraphique(this);
            PositionEquipement=f_RéfPositionEquipement;
      }

      public synchronized void DessineToi()
      {
            int i=RéfEquipement.GetListeCouches().size()-1;
            while (i>-1)
            {
                  ((couche) RéfEquipement.GetListeCouches().elementAt(i)
                              ).GetRéfObjetGraphique().DessineToi();
                  i--;
            }

            for(int j=0;j<RéfEquipement.GetListeLiaisonsPhy().size();j++)
            {
                  liaisonPHY w_LiaisonPhyTemp=
                        (liaisonPHY)RéfEquipement.GetListeLiaisonsPhy(
                                                      ).elementAt(j);

                  w_LiaisonPhyTemp.GetRéfObjetGraphique().DessineToi();

            }
      }
}

/*============================================================*/
/*============================================================*/

□
package Presentation;
□
```

```
import java.awt.*;
import java.util.*;
import java.io.*;
import java.lang.Math;
import java.awt.*;

import Reseau.liaisonPHY;
import Reseau.liaisonLOG;
import Reseau.equipement;
import Reseau.trame;
import Reseau.sousReseau;

import Presentation.P_trame;

//Importation des package nécessaires à l'interaction avec VRML

import vrml.external.field.*;

import vrml.external.Node;

import vrml.external.Browser;

import vrml.external.exception.*;


/*============================================================*/
/* Classe P_liaisonPHY                    */

public class P_liaisonPHY

{


      liaisonPHY RéfLiaisonPhy;

      Node[] NoeudVRML;

      EventInMFNode Ajout;
                  //permet d'ajouter un objet dans la scène VRML

      EventInMFNode AjoutCell;
                  //permet d'ajouter une cellule dans la scène VRML
      EventInMFNode SupprCell;
                  //permet de supprimer une celllule dans la scène VRML

      Browser browser;
                  //permet de lier l'objet au monde VRML

      AttributsLiens RéfAttributs;

      Vector ListeAnimables;


/*============================================================*/
```

```
□
      public P_liaisonPHY (liaisonPHY f_RéfLiaisonPhy)
□
      {
□
            RéfLiaisonPhy=f_RéfLiaisonPhy;
□
            RéfLiaisonPhy.SetRéfObjetGraphique(this);
□
            browser=RéfLiaisonPhy.GetRéfSousRéseau(
                                    ).GetRéfObjetGraphique().GetBrowser();
                                □
            Ajout=RéfLiaisonPhy.GetRéfSousRéseau(
                                    ).GetRéfObjetGraphique().GetAjout();
                                □
            AjoutCell=RéfLiaisonPhy.GetRéfSousRéseau(
                                    ).GetRéfObjetGraphique().GetAjoutCell();
                                □
            SupprCell=RéfLiaisonPhy.GetRéfSousRéseau(
                                    ).GetRéfObjetGraphique().GetSupprCell();
                                □
            RéfAttributs=new AttributsLiens(
                                          -900,0,0,Color.white,0,0,1,0,1,1);
                                □
            ListeAnimables=new Vector();
□
      }
□
□
      public P_liaisonPHY (liaisonPHY f_RéfLiaisonPhy,
                              AttributsLiens f_RéfAttributs)
                        □
      {
□
            RéfLiaisonPhy=f_RéfLiaisonPhy;
□
            RéfLiaisonPhy.SetRéfObjetGraphique(this);
            browser=RéfLiaisonPhy.GetRéfSousRéseau(
                                    ).GetRéfObjetGraphique().GetBrowser();
                                □
            Ajout=RéfLiaisonPhy.GetRéfSousRéseau(
                                    ).GetRéfObjetGraphique().GetAjout();
                                □
            AjoutCell=RéfLiaisonPhy.GetRéfSousRéseau(
                                    ).GetRéfObjetGraphique().GetAjoutCell();
                                □
            SupprCell=RéfLiaisonPhy.GetRéfSousRéseau(
                                    ).GetRéfObjetGraphique().GetSupprCell();
                                □
            RéfAttributs=f_RéfAttributs;
□
            ListeAnimables=new Vector();
□
      }
□
□
      public synchronized void DessineToi()
□
```

```
        {
□
              if (RéfAttributs.GetAxeX()!=-900)
□
              {
□
                    try
                    {
□
                      NoeudVRML=browser.createVrmlFromString("Transform{ \n"+
                                "translation "+RéfAttributs.GetAxeX()+
                                            " "+RéfAttributs.GetAxeY()+
                                            " "+RéfAttributs.GetAxeZ()+" \n"+
                                            "rotation "
                                             +RéfAttributs.GetRotationAxeX()+

                                           " "+RéfAttributs.GetRotationAxeY()+
                                           " "+RéfAttributs.GetRotationAxeZ()+
                                           " "+RéfAttributs.GetInclinaison()+"
                                             \n"+
                                      □
                                "  children[ \n"+
□                                 "     Transform{ \n"+
□                                 "           rotation 0 0 1 1.57 \n"+
□                                 "           children[ \n"+
□                                 "                 Shape{ \n"+
                                  "                       appearance Appearance{
                                                            \n"+
                                                      □
                                  "                         material Material{
                                                            \n"+
                                                            □
                                  "                           diffuseColor "
                                                  +RéfAttributs.GetRouge()+
                                                  □
                                               " "+RéfAttributs.GetVert()+
□
                                               " "+RéfAttributs.GetBleu()+"
                                                            \n"+
                                                            □
                                  "                       } \n"+

                                  "                 } \n"+
                                  "                 geometry Cylinder{ \n"+
                                  "                       radius
                                                  "+RéfAttributs.GetRayon()+"
                                                            \n"+
                                                      □
                                  "                       height
                                                  "+RéfAttributs.GetLongueur()
                                                            +" \n"+
                                                      □
                                  "                 } \n"+
□
                                  "                 } \n"+
□
```

```
                                "           ] \n"+
□
                                "     } \n"+
□
                                " ] \n"+
□
                                "} \n");
□
                }

                catch(InvalidVrmlException e) {}
                Ajout.setValue(NoeudVRML);
        }
}


public synchronized Thread AnimationTrame(
                                equipement F_RéfEquipementSource,
                                P_trame f_pTrame)
        □

          //attention tous les changements dans les paramètres
          //de cette méthode devront
          //être répercuté dans la classe LiaisonLOG.java du package
          //Reseau\Atm
{
        animationDataUnit w_ProcessusPourXZ;

        float w_ExtrémitéX1=
              RéfLiaisonPhy.GetRéfSupport1().GetRéfEquipementSource(
        □
                              ).GetRéfObjetGraphique().GetPositionX();

        float w_ExtrémitéX2=
              RéfLiaisonPhy.GetRéfSupport1().GetRéfEquipementCible(
        □
                              ).GetRéfObjetGraphique().GetPositionX();
□
□
        float w_ExtrémitéZ1=
              RéfLiaisonPhy.GetRéfSupport1().GetRéfEquipementSource(
        □
                              ).GetRéfObjetGraphique().GetPositionZ();

        float w_ExtrémitéZ2=
              RéfLiaisonPhy.GetRéfSupport1().GetRéfEquipementCible(
        □
                               ).GetRéfObjetGraphique().GetPositionZ();
□
□
        if (f_RéfEquipementSource ==
□
                RéfLiaisonPhy.GetRéfSupport2().GetRéfEquipementCible())
        {
              w_ProcessusPourXZ=new animationDataUnit(w_ExtrémitéX1,
                                                      w_ExtrémitéX2,
                                                      w_ExtrémitéZ1,
                                                      w_ExtrémitéZ2,
                                                      f_pTrame);
```

```
        }
        else
        {
              w_ProcessusPourXZ=new animationDataUnit(w_ExtrémitéX2,
                                                      w_ExtrémitéX1,
                                                      w_ExtrémitéZ2,
                                                      w_ExtrémitéZ1,
                                                      f_pTrame);
        }

        w_ProcessusPourXZ.start();
        return w_ProcessusPourXZ;
  }
  class animationDataUnit extends Thread
  {
        P_trame RéfpTrame;
        float Xdébut;     /* trajectoire en X (en largeur) */
        float Xfin;
        float Zdébut;     /* trajectoire en Z (en profondeur)*/
        float Zfin;
        float LongueurVecteur;
        float InverseLongueur;
        float DeltaX;
        float DeltaZ;

        /* constructeur */
        animationDataUnit(float f_Xdébut, float f_Xfin,
                          float f_Zdébut, float f_Zfin,
                          P_trame f_RéfpTrame)
        {
              RéfpTrame=f_RéfpTrame;
              Xdébut=f_Xdébut;
              Xfin=f_Xfin;
              Zdébut=f_Zdébut;
              Zfin=f_Zfin;
              LongueurVecteur=(float)(Math.sqrt(
                    Math.pow((double)(Xfin-Xdébut),(double)2)
                    +Math.pow((double)(Zfin-Zdébut),(double)2)));

              InverseLongueur=(float)(1/LongueurVecteur);
              DeltaX=(float)((Xfin-Xdébut)*InverseLongueur);
              DeltaZ=(float)((Zfin-Zdébut)*InverseLongueur);

        }

        /* ce que doit faire le processus */
        public synchronized void run()
        {
              float i= Xdébut;
              float k= Zdébut;

              this.setPriority(NORM_PRIORITY);
              sousReseau w_RéfSousRéseau=
                          RéfLiaisonPhy.GetRéfSousRéseau();
                    □
              w_RéfSousRéseau.AddRéfThread(this);

              if (Xdébut <= Xfin)
                          //déplacement des unités de données de la
                          //gauche vers la droite
                          □
```

```
                {
                        while (i<Xfin)
                        {
                              while(w_RéfSousRéseau.isThreadSuspended())
                              {
                                    try{Thread.sleep(100);}
                                    catch(InterruptedException e){return;}
                              }
                              if(w_RéfSousRéseau.isThreadActif(this))
                              {
                                    RéfpTrame.SetRéfAttributs(
                                          new AttributsBoite(
                                                i,RéfAttributs.GetAxeY()+
                                                0.125f,k,new Color(
                                                  RéfpTrame.GetRéfAttributs(
                                                      ).GetRouge(),
                                                  RéfpTrame.GetRéfAttributs(
                                                      ).GetVert(),
                              RéfpTrame.GetRéfAttributs(
                                                      ).GetBleu()),
                                    (float)(.25),(float)(.25),(float)(.5)));
                              try
                              {
                                          RéfpTrame.NoeudActuel=
                                    DessineTrame(RéfpTrame);
                              }
                              catch(InvalidVrmlException e) {}
      AjoutCell.setValue(RéfpTrame.NoeudActuel);

      ListeAnimables.addElement(RéfpTrame.NoeudActuel);
                              if (Zdébut < Zfin)
                              {
                                    k=k+(Math.abs(DeltaZ)*0.20f);
                              }
                              else
                              {
                                    k=k-(Math.abs(DeltaZ)*0.20f);
                              }
                              i=i+(Math.abs(DeltaX)*0.20f);
```

```
                                    if(RéfpTrame.Animation!=1)
                                    {

        if(ListeAnimables.contains(RéfpTrame.NoeudPrécédent))
                                          {

        SupprCell.setValue(RéfpTrame.NoeudPrécédent);
                                          }
                                    }
                                    RéfpTrame.MiseAJour();
                                      //permet de prévenir qu'au tour
                                      //suivant
                                      □
                                      //l'ancien noeud sera celui que l'on
                                      □
                                      //vient de créer
                                      □

□                                   RéfpTrame.MiseAJourAnimation();
□
□                               }
□                               else
□                               {
□                                     if(RéfpTrame.NoeudPrécédent!=null)
□                                     {
                                            SupprCell.setValue(
                                                  RéfpTrame.NoeudPrécédent);
                                            □
                                      }
□                                     if(RéfpTrame.NoeudActuel!=null)
□                                     {
□                                           SupprCell.setValue(
                                                  RéfpTrame.NoeudActuel);
                                                  □
                                      }
□                                     break;
□                               }
□
□                               w_RéfSousRéseau.AbandonCPU();
□                         }
□                   }
□                   else
```

```
                        //déplacement des unités de données de la droite
                        //vers la gauche
                        □
                  {
□
□
                        while (i>Xfin)
□
                        {
□
                              while(w_RéfSousRéseau.isThreadSuspended())
□
                              {
□
                                    try{Thread.sleep(100);}
□
                                    catch(InterruptedException e){return;}
□
                              }
                              if(w_RéfSousRéseau.isThreadActif(this))
                              {
                                    RéfpTrame.SetRéfAttributs(
                                          new AttributsBoite(
                                          i,RéfAttributs.GetAxeY()+0.125f,k,
                                    □
                                          new Color(
                                           RéfpTrame.GetRéfAttributs(
                                                ).GetRouge(),
                                          □
                                           RéfpTrame.GetRéfAttributs(
                                                ).GetVert(),
                                           RéfpTrame.GetRéfAttributs(
                                                ).GetBleu()),
                                          □
                                    (float)(.25),(float)(.25),(float)(.5)));
□
                                    try
□
                                    {
□
                                          RéfpTrame.NoeudActuel=
                                                DessineTrame(RéfpTrame);
                                          □
                                    }
□
                                    catch(InvalidVrmlException e) {}
□
                                    AjoutCell.setValue(
                                          RéfpTrame.NoeudActuel);

                                    ListeAnimables.addElement(
                                                RéfpTrame.NoeudActuel);
                                          □
                                    if (Zdébut < Zfin)
□
                                    {
□
                                          k=k+(Math.abs(DeltaZ)*0.1f);
□
```

```
                        }
                        else
                        {
                              k=k-(Math.abs(DeltaZ)*0.10f);
                        }
                        i=i-(Math.abs(DeltaX)*0.10f);
                        if(RéfpTrame.Animation!=1)
                        {
                              if(ListeAnimables.contains(
                                    RéfpTrame.NoeudPrécédent))
                              {
                                    SupprCell.setValue(
                                      RéfpTrame.NoeudPrécédent);
                              }
                        }
                        RéfpTrame.MiseAJour();
                     //permet de prévanir qu'au tour suivant
                     //l'ancien noeud sera celui que l'on
                     //vient de créer
                        RéfpTrame.MiseAJourAnimation();
                  }
                  else
                  {
                        if(RéfpTrame.NoeudPrécédent!=null)
                        {
                              SupprCell.setValue(
                                    RéfpTrame.NoeudPrécédent);
                        }
                        if(RéfpTrame.NoeudActuel!=null)
                        {
                              SupprCell.setValue(
                                    RéfpTrame.NoeudActuel);
                        }
                        break;
                  }
                  w_RéfSousRéseau.AbandonCPU();
               }
            }
         }
      }
```

```
□
      private Node[] DessineTrame(P_trame f_RéfpTrame)
□
      {
            Node[] w_Noeud;

            w_Noeud=browser.createVrmlFromString(
                        "Transform{ \n"+
                        "     translation
                              "+f_RéfpTrame.GetRéfAttributs().GetAxeX()+"  "
                              "+f_RéfpTrame.GetRéfAttributs().GetAxeY()+"  "
                              "+f_RéfpTrame.GetRéfAttributs().GetAxeZ()+
                              □
                                                                  " \n"+
                        "     rotation "+RéfAttributs.GetRotationAxeX()+
                                    " "+RéfAttributs.GetRotationAxeY()+
                                    " "+RéfAttributs.GetRotationAxeZ()+
                                    " "+RéfAttributs.GetInclinaison()+"
                                                                  \n"+
                                                            □
                        "     children[ \n"+
□
                        "           Shape{ \n"+
□
                        "                 appearance Appearance{ \n"+
                        "                       material Material{ \n"+
                        "                         diffuseColor"+
                              " "+f_RéfpTrame.GetRéfAttributs(
                                                ).GetRouge()+
                                                □
                              " "+f_RéfpTrame.GetRéfAttributs(
                                                ).GetVert()+
                              " "+f_RéfpTrame.GetRéfAttributs(
                                                ).GetBleu()+
                                          □
                                                      " \n"+
                        "                       } \n"+
                        "                 } \n"+
                        "                 geometry Box{ \n"+
                        "                       size"+
                              " "+f_RéfpTrame.GetRéfAttributs(
                                                ).GetTailleX()+
                                          □
                              " "+f_RéfpTrame.GetRéfAttributs(
                                                ).GetTailleY()+
                              " "+f_RéfpTrame.GetRéfAttributs(
                                                ).GetTailleZ()+
                                          □
                                                      " \n"+
                        "                 } \n"+
                        "           } \n"+
                        "     ] \n"+
                        "} \n");

            return w_Noeud;
      }
}
/*============================================================*/
/*============================================================*/
```

```
package Presentation;


import java.awt.*;
import Reseau.trame;
import Reseau.Atm.celluleATM;

//importation package de collaboration java et VRML

import vrml.external.field.*;

import vrml.external.Node;

import vrml.external.Browser;

import vrml.external.exception.*;


/*============================================================*/
/* Classe P_sousReseauATM                    */


public class P_trame

// cette classe ne contient pas de méthode DessineToi car c'est la
// représentation

// des liens physique (P_liaisonPHY) qui se charge de représenter les
// unités de

// données qui voyagent sur chaque liaison.


{

      trame RéfTrame;

      Node[] NoeudActuel;

      Node[] NoeudPrécédent;

      public int Animation;

      AttributsBoite RéfAttributs;


      /*============================================================*/
      //premier constructeur

      public P_trame (trame f_RéfTrame)

      {
```

```
            RéfTrame=f_RéfTrame;
□
            RéfTrame.SetRéfObjetGraphique(this);
□
            Animation=1;
□
            RéfAttributs=new AttributsBoite(-900,0,0,
                                        new Color (0.5f,0f,0.5f),0,0,0);
      }

      //deuxièmes constructeur
      public P_trame (trame f_RéfTrame,AttributsBoite f_RéfAttributs)
      {
            RéfTrame=f_RéfTrame;
            RéfTrame.SetRéfObjetGraphique(this);
            RéfAttributs=f_RéfAttributs;
            Animation=1;
      }


      public synchronized void MiseAJour()
      {
      //Cette méthode sert pour l'affichage de cellules dans la scene VRML

            NoeudPrécédent=NoeudActuel;
      }

      public void MiseAJourAnimation()
      {
            Animation++;
      }


      public synchronized void SetRéfAttributs(
                                        AttributsBoite f_RéfAttributs)
      {
            RéfAttributs=f_RéfAttributs;
      }

      public synchronized AttributsBoite GetRéfAttributs()
      {
            return RéfAttributs;
      }

}

/*============================================================*/
/*============================================================*/
```

IV l'applet Java

```
/*
□
Cette applet est la composition des trois simulations créées dans les cadre
du projet P2-Verso.  Il permet à l'apprenant de revoir ces trois
simulations comme il le désire.*/
□

□
import java.util.*;
□
```

```
import java.applet.*;
import java.awt.*;

import Reseau.equipement;
import Reseau.liaisonPHY;
import Reseau.couche;

import Reseau.Atm.machineHote;
import Reseau.Atm.sousReseauATM;
import Reseau.Atm.liaisonVCC;
import Reseau.Atm.liaisonVCL;
import Reseau.Atm.commutateurATM;
import Reseau.Atm.coucheAAL;
import Reseau.Atm.coucheAPL;
import Reseau.Atm.celluleATM;

import Presentation.P_equipement;
import Presentation.P_machineHote;
import Presentation.P_sousReseauATM;
import Presentation.P_liaisonPHY;
import Presentation.P_commutateurATM;
import Presentation.P_couche;
import Presentation.AttributsLiens;
import Presentation.AttributsBoite;

//importation des packages nécessaires à la collaboration avec VRML

import vrml.external.field.*;
□
import vrml.external.Node;
import vrml.external.Browser;
import vrml.external.exception.*;
import netscape.javascript.JSException;

public class SimuLibre extends java.applet.Applet implements Runnable
{
            //variables propres au réseau
      sousReseauATM MonRéseau;
      P_sousReseauATM pMonRéseau;

      machineHote HoteUN, HoteDEUX, HoteTROIS, HoteQUATRE;
      P_machineHote     pHoteUN, pHoteDEUX, pHoteTROIS, pHoteQUATRE;

      commutateurATM SwitchUN, SwitchDEUX, SwitchTROIS;
      P_commutateurATM pSwitchUN, pSwitchDEUX, pSwitchTROIS;

      liaisonPHY Liaison01, Liaison02, Liaison03, Liaison04,
                  Liaison05,Liaison06,Liaison07;

      P_liaisonPHY pLiaison01,pLiaison02,pLiaison03,pLiaison04,
                    pLiaison05,pLiaison06,pLiaison07;

      //déclaration des variables utiles pour la gestion de l'applet java

      Thread actif;
□
      Image imgTampon;
□
      Graphics gTampon;
□
```

```
        float position1[]=new float[3]; //position des 7 equipements
        float position2[]=new float[3];
        float position3[]=new float[3];
        float position4[]=new float[3];

        float position5[]=new float[3];
        float position6[]=new float[3];
        float position7[]=new float[3];

        boolean erreur=false;
        //indique si l'initialisation se passe correctement

        boolean Pause;  //indique si on a appuié sur le bouton Pause
        int TypeSimu;
        //indique quelle simulation est sélectionnée dans les boutons radio

        int Passage=1;//indique si le simu vient d'être lancé.




              //déclarations liées au run de la commutation
        Color w_CouleurCelluleUn;
        Color w_CouleurCelluleDeux;

              //ces variables sont utilisés pour prendre l'information à
              //afficher dans l'applet dans les tables de routage des
              //commutateurs.
        liaisonVCL w_Vcl1;
        liaisonVCL w_Vcl2;
        liaisonVCL w_Vcl20;
        liaisonVCL w_Vcl3;
        liaisonVCL w_Vcl4;
        liaisonVCL w_Vcl5;
        liaisonVCL w_Vcl50;

        //déclarations liées au run de l'étude des couches
        Color CouleurMiseEnEvidence=new Color(255,128,0);

        String Description=new String("test");
        String NomCouche="APL";

        //déclarations liées au run de l'étude des données
        Color w_CouleurAAL;

        //déclaration des variables utiles pour l'interface de l'applet java
        Panel PanneauChoix=new Panel();
        Panel PanneauCommande=new Panel();
```

```
Panel PanneauMagneto=new Panel();
Font PoliceDansPanneau=new Font("Dialog",Font.PLAIN,12);
CheckboxGroup var = new CheckboxGroup();
      Checkbox box1 = new Checkbox("Commutation",var,true);
      Checkbox box2 = new Checkbox("Unités de données",var,false);
      Checkbox box3 = new Checkbox("Couche",var,false);
      Checkbox box4 = new Checkbox("Multiplexage",var,false);

Button BoutonPlay =new Button("Démarrage");
Button BoutonPause =new Button("Pause");
Button BoutonStop=new Button ("Stop");

int VitesseDéfaut = 4000;
      //Les couches dorment 4 secondes

int VitesseMax = 500;
     //La vitesse la pus rapide ->les couches dorment 0.5 sec
int VitesseMin = 10000;
     //La vitesse la pus lente ->les couches dorment 10 sec

int VitesseIncrément = 1000;
     //Valeur de pas 1 sec


public Scrollbar vitesse = new Scrollbar(Scrollbar.VERTICAL,
                                         VitesseDéfaut,
                                         VitesseIncrément,
                                         VitesseMax,
                                         VitesseMin);


Label Ligne1=new Label();

Label Ligne2=new Label();




//variable pour l'interaction avec Vrml

EventInMFNode addChildren;

EventInMFNode removeChildren;

EventInMFNode addCell;
EventInMFNode removeCell;
Browser browser=null;
Node MONDE;
Node DYNAMIQUE;

      //variables pour la gestion des points de vue
Node Vue1;
EventInSFBool LienVue1;
Node Vue2;
EventInSFBool LienVue2;
```

```
public void init()
{
	//applet java
	setBackground(Color.white);

	setLayout(new BorderLayout(10,10));
	setFont(PoliceDansPanneau);


	add("West",PanneauChoix);
	add("East",PanneauCommande);


	PanneauChoix.setLayout(new GridLayout(4,1));
	PanneauChoix.setFont(PoliceDansPanneau);

	PanneauCommande.setLayout(new BorderLayout(5,5));
	PanneauCommande.setFont(PoliceDansPanneau);

	PanneauChoix.add(box1);
	PanneauChoix.add(box2);
	PanneauChoix.add(box3);
	PanneauChoix.add(box4);

	TypeSimu=1;

	PanneauCommande.add("Center",PanneauMagneto);

	PanneauMagneto.setLayout(new GridLayout(3,1));
	PanneauMagneto.add(BoutonPlay);
	PanneauMagneto.add(BoutonPause);
	PanneauMagneto.add(BoutonStop);

	Panel PanneauBidon=new Panel();

	PanneauCommande.add("West",vitesse);
	PanneauCommande.add("East",PanneauBidon);

	add(Ligne1,Label.CENTER);
	add(Ligne2,Label.CENTER);



	//création de la zone d'affichage de l'applet
	imgTampon=createImage(size().width,size().height);
	gTampon=imgTampon.getGraphics();

	browser=null;

	int essai=0;

	do
	{
		showStatus("Chargement de VRML en cours...");
		try
		{
			//création du lien entre VRML et l'applet
			browser=Browser.getBrowser(this);
		}
```

```
                catch(Throwable t)
                {
                        System.out.println(
                                "Impossible de trouver le navigateur VRML");
                        □
                        System.out.println("Throwable :" + t);
□
                        System.out.println("Stack Trace : ");
□
                        t.printStackTrace(System.out);
□
                }
□
                if (browser==null)
□
                {
                        try {Thread.sleep(2000);}
                        catch (InterruptedException e){}
                }
                essai++;
        } while (browser==null && essai < 10);

        //

        if(browser != null)
        {
                System.out.println("Navigateur = "+ browser);
                try
                {
                        MONDE=browser.getNode("MONDE");
                        DYNAMIQUE=browser.getNode("DYNAMIQUE");
                        Vue1=browser.getNode("Entree");
                        Vue2=browser.getNode("Haut");

addChildren=(EventInMFNode)MONDE.getEventIn("addChildren");
removeChildren=(EventInMFNode)MONDE.getEventIn("removeChildren");
addCell=(EventInMFNode)DYNAMIQUE.getEventIn("addChildren");
removeCell=(EventInMFNode)DYNAMIQUE.getEventIn("removeChildren");
LienVue1=(EventInSFBool)Vue1.getEventIn("set_bind");
LienVue2=(EventInSFBool)Vue2.getEventIn("set_bind");
}
                catch (InvalidNodeException e)
                {
                        erreur=true;
                }
                catch(InvalidEventInException e)
                {
                        erreur=true;
                }

                //lancement des initialisations
                InitRéseauEtHotes();
                InitSwitches();
                InitLiaisonsPhy();
                InitCouches();
                InitAdresses();
                MonRéseau.AbandonCPU();

                pMonRéseau.DessineToi();
        }
        else
```

```
            {
                  erreur=true;
            }

            if(erreur==false)
            {
                  showStatus("Initialisation Ok");
            }
            else
            {
                  gTampon.setColor(Color.red);
                  gTampon.drawString(
                  "Echec lors de l'initialisation du monde VRML",150,90);
                  □
                  repaint();
□
            }
□
      }
□
□
            /* la méthode start() appartient à mon applet */
□
      public void start()
□
      {
□
            if ((actif==null) && (erreur==false))
□
                  {
□
                  actif=new Thread(this);
□
                  actif.start();
                  }
      }


      /* la méthode stop() appartient à mon applet */
      public void stop()
      {
            if (actif!=null)
            {
                  MonRéseau.StopAllThreads();
                  actif.stop();
                  actif=null;
            }
      }

      public void run()
      {
            if(Passage==1)
            {
                  Passage++;
                  Pause=true;
                  while(Pause)
                  {
                        MonRéseau.AbandonCPU(50);
                  }
            }
```

```
            switch(TypeSimu)
            {
                  case 1:
                        RunCommutation();
                        break;

                  case 2:
                        RunDonnees();
                        break;

                  case 3:
                        RunCouches();
                        break;

                  default:
                        box1.setState(true);

                        RunCommutation();
                        break;

            }

      }


      public boolean handleEvent(Event evt)
      {
            if(evt.target == vitesse)
            {
                  switch(TypeSimu)
                  {
                        case 3:
                              MonRéseau.SetSleepTime(vitesse.getValue());
                              break;

                        default:
                              break;
                  }
                  return true;

            }
            else return super.handleEvent(evt);
                  // appel de méthode handleEvent() par défaut !!!!!
      □
      }
□

□

□

      public boolean action(Event evt,Object arg)
□
      {
□
            if(evt.target instanceof Checkbox)
            {
                  gTampon.clearRect(0,0,size().width,size().height);
                  repaint(); //on nettoie la surface d'affichage.
                  showStatus("Changement de simulation");
                  MonRéseau.StopAllThreads();
```

```
                    if(evt.target==box1)
                    {
                          TypeSimu=1;
                    }
                    else
                    {
                          if(evt.target==box2)
                          {
                                TypeSimu=2;
                          }
                          else
                          {
                                if(evt.target==box3)
                                {
                                      TypeSimu=3;
                                }
                                else
                                {

      gTampon.clearRect(0,0,size().width,size().height);
                                      gTampon.setColor(Color.red);
                                      gTampon.drawString("Cette simulation
n'est pas implémentée",150,90);
                                }
                          }
                    }
              }
              else
              {
                    if(evt.target== BoutonPlay & (actif==null | Pause))
                    {
                          Pause=false;
                          if(actif==null)
                          {
                                actif=new Thread(this);
                                actif.start();
                          }
                          else
                          {
                                MonRéseau.ResumeAllThreads();
                          }
                    }
                    else
                    if(evt.target==BoutonPause)
                    {
                          Pause=!Pause;
                          if(Pause & actif != null)
                          {
                                showStatus("Programme mis en PAUSE");
                                MonRéseau.SuspendAllThreads();
                          }
                          if(!Pause & actif!=null)
                          {
                                showStatus("Programme en cours d'éxécution");
                                MonRéseau.ResumeAllThreads();
                          }
                    }
                    else
                    {
                          if(evt.target==BoutonStop & actif!=null)
                          {
```

```
        gTampon.clearRect(0,0,size().width,size().height);
                                repaint();
                                //on nettoie la surface d'affichage.
                        □
                                showStatus("Fin du programme");
□
                                MonRéseau.StopAllThreads();
□
                                if(TypeSimu==3)
□
                                {
□
                                      RestaurerTout();
□
                                }
□
                                Pause=false;
                        }
                  }
            }
            return true;
      }

      public void paint(Graphics f_Graphique)
      {
            f_Graphique.drawImage(imgTampon,0,0,this);
      }

      public void update(Graphics g)
      {
            paint(g);
      }

/*-------------------------------------------------------*/
/*------- Déclaration des nouvelles méthodes   --------*/
/*-------------------------------------------------------*/


/*-------------------------------------------------------*/
/*- Ce qui sera exécuté lors d'une simu "Commutation" -*/
/*-------------------------------------------------------*/

      public void RunCommutation()
      {
            if(!erreur)
            {
                  LienVue2.setValue(true);

                  showStatus("Programme en cours d'exécution");
                  coucheAAL CoucheAAL;

                  MonRéseau.EnableAllThreads();
                  MonRéseau.SetSleepTime(70);

                  //on laisse les cellules dormir quelques milli
                  //secondes dans la couche pour afficher le
            □
                  //changement dans la table
            □
```

```
                Pause=false;

                liaisonVCC LiaisonVCC1=new liaisonVCC(HoteUN,HoteDEUX);

                CoucheAAL = (coucheAAL) HoteUN.GetCoucheWithNom("AAL");
                CoucheAAL.DataREQ(LiaisonVCC1, 1);
                MonRéseau.AbandonCPU();


                liaisonVCC LiaisonVCC2=new
                           liaisonVCC(HoteTROIS,HoteQUATRE);
                CoucheAAL = (coucheAAL)
                           HoteTROIS.GetCoucheWithNom("AAL");
                CoucheAAL.DataREQ(LiaisonVCC2, 1);
                MonRéseau.AbandonCPU();

                while(MonRéseau.GetListeLiaisonsVcc().isEmpty())
                {
                     MonRéseau.AbandonCPU(100);
                }
                MonRéseau.AbandonCPU();

                celluleATM CelluleUn=

                (celluleATM)((LiaisonVCC1.GetListeTrames(
                                                  )).firstElement());
                //la boucle permet de s'assurer que CelluleUn != null


                celluleATM CelluleDeux=

                 (celluleATM)((LiaisonVCC2.GetListeTrames(
                                                  )).firstElement());
                //la boucle permet de s'assurer que CelluleDeux != null


                Pause=false;


                //affectattion des variables pour permettre l'affichage
                //des tables de commutation


                w_Vcl1=LiaisonVCC1.GetRéfFirstLiaisonVcl();

                w_Vcl2=SwitchUN.GetRéfLiaisonVclSortant(w_Vcl1);

                w_Vcl20=SwitchDEUX.GetRéfLiaisonVclSortant(w_Vcl2);
                     //
                w_Vcl3=LiaisonVCC2.GetRéfFirstLiaisonVcl();
                w_Vcl4=SwitchUN.GetRéfLiaisonVclSortant(w_Vcl3);
                     //
```

```
w_Vcl5=SwitchDEUX.GetRéfLiaisonVclSortant(w_Vcl4);
w_Vcl50=SwitchTROIS.GetRéfLiaisonVclSortant(w_Vcl5);
      //
while (!MonRéseau.GetListeLiaisonsVcc().isEmpty())
{
      gTampon.clearRect(0,0,size().width,size().height);

      gTampon.setColor(Color.lightGray);

      DessinRoute();

      DessinTableUn();

      DessinTableDeux();

      DessinTableTrois();

      if(CelluleUn!=null)
      {
            w_CouleurCelluleUn=
                  CelluleUn.GetRéfObjetGraphique(
                        ).GetRéfAttributs().GetCouleur();
                  □

□
            if((String)(

            CelluleUn.GetRéfLocalisation().elementAt(0))=="2")
            □
                  {
□
                        MonRéseau.AbandonCPU(100);
                        repaint();
                  }
                  else
                  {
                        couche w_CoucheTemp=
                              (couche)(
                                CelluleUn.GetRéfLocalisation(
                                          ).elementAt(1));
                                    □

□
                        if(w_CoucheTemp.GetRéfEquipement(
                                    ).GetTypeEquipement()==1)
                                    □
                        {
□
                              commutateurATM w_RéfCommu=

□
                                 (commutateurATM)(
                                 w_CoucheTemp.GetRéfEquipement());
                              □

□
                        if(w_RéfCommu.GetRéfObjetGraphique(
                        ).GetPositionX()== position5[0])
                  □
                        {
□
```

```
                        gTampon.setColor(w_CouleurCelluleUn);
                    □
                    ChangeLigneUnTableUn();
□
                    repaint();
□
                              }
□
                                    else
                                    {
      gTampon.setColor(w_CouleurCelluleUn);
                                          ChangeLigneUnTableDeux();
                                          repaint();
                                    }
                              }
                        }
                    }

                    if(CelluleDeux!=null)
                    {
                        w_CouleurCelluleDeux=
CelluleDeux.GetRéfObjetGraphique().GetRéfAttributs().GetCouleur();

      if((String)(CelluleDeux.GetRéfLocalisation().elementAt(0))=="2")
                        {
                              repaint();
                        }
                        else
                        {
                              couche w_CoucheTempo=
      (couche)(CelluleDeux.GetRéfLocalisation().elementAt(1));

      if(w_CoucheTempo.GetRéfEquipement().GetTypeEquipement()==1)
                              {
                                    commutateurATM w_RéfCommu=
      (commutateurATM)(w_CoucheTempo.GetRéfEquipement());

      if(w_RéfCommu.GetRéfObjetGraphique().GetPositionX()==
                                  position5[0])
                                    {
      gTampon.setColor(w_CouleurCelluleDeux);
                                          ChangeLigneDeuxTableUn();
                                          repaint();
                                    }
                                    else
                                    {
      if(w_RéfCommu.GetRéfObjetGraphique().GetPositionX()==
                                    position6[0])
                                          {
```

```
gTampon.setColor(w_CouleurCelluleDeux);

ChangeLigneDeuxTableDeux();
                                                    repaint();
                                                }
                                                else
                                                {

gTampon.setColor(w_CouleurCelluleDeux);
                                                    ChangeLigneTableTrois();
                                                    repaint();
                                                }
                                            }
                                        }
                                    }

                                    MonRéseau.AbandonCPU(200);
                                    if (MonRéseau.NombreThreads()==0)
                                    {
                                        MonRéseau.DelAllRéfLiaisonsVcc();
                                        break;
                                    }

                                }
                                gTampon.clearRect(0,0,size().width,size().height);
                                repaint();
                                showStatus ("Fin du programme");
                                actif=null;
                        }
                        else
                        {
                                gTampon.clearRect(0,0,size().width,size().height);
                                gTampon.drawString("Un problème est survenu lors de
l'initialisation de VRML.",150,70);
                                repaint();
                                showStatus ("Fin du programme");
                                actif=null;
                        }

        }

/*------------------------------------------------------------ */
/*- Ce qui sera exécuté lors d'une simu "Données" --------- */
/*------------------------------------------------------------*/


        public void RunDonnees()
        {
                if(erreur==false)
                {
                        LienVue2.setValue(true);

                        showStatus("Programme en cours d'exécution");

                        MonRéseau.EnableAllThreads();
                        MonRéseau.SetSleepTime(70);
                        Pause=false;
```

```
                liaisonVCC LiaisonVCC=new liaisonVCC(HoteTROIS,HoteDEUX);

                coucheAAL CoucheAAL = (coucheAAL)
HoteTROIS.GetCoucheWithNom("AAL");
                CoucheAAL.DataREQ(LiaisonVCC, 1);
                MonRéseau.AbandonCPU();

                while (MonRéseau.GetListeLiaisonsVcc().isEmpty())
                {
                      MonRéseau.AbandonCPU();
                }

                w_Vcl1=LiaisonVCC.GetRéfFirstLiaisonVcl();
                w_Vcl2=SwitchUN.GetRéfLiaisonVclSortant(w_Vcl1);
                w_Vcl20=SwitchDEUX.GetRéfLiaisonVclSortant(w_Vcl2);


     w_CouleurAAL=HoteUN.GetCoucheWithNom("AAL").GetRéfObjetGraphique(

          ).GetRéfAttributs().GetCouleur();


                while (LiaisonVCC.GetListeTrames().size()==0)
                {
                      MonRéseau.AbandonCPU();
                }
                MonRéseau.AbandonCPU();

                celluleATM
cell=(celluleATM)((LiaisonVCC.GetListeTrames()).firstElement());
                //la boucle permet d'être sur que cellule est différent
de null

                while (!MonRéseau.GetListeLiaisonsVcc().isEmpty())
                {
                      System.out.println("entree boucle");

                      gTampon.clearRect(0,0,size().width,size().height);

                      DessinerRoute();

                      DessinerCellule(cell);

                      DessinerInformation();

                      repaint();

                      MonRéseau.AbandonCPU();
                      if(MonRéseau.NombreThreads()==0)
                      {
                            MonRéseau.DelAllRéfLiaisonsVcc();
                            break;
                      }
                }
                gTampon.clearRect(0,0,size().width,size().height);
                repaint();
                showStatus("Fin du programme");
                actif=null;
          }
          else
          {
```

```
                    gTampon.clearRect(0,0,size().width,size().height);
                    gTampon.drawString("Un problème est survenu lors de
l'initialisation de VRML.",150,70);
                    repaint();
                    showStatus("Fin du programme");
                    actif=null;
               }
          }

/*-------------------------------------------------------------*/
/*- Ce qui sera exécuté lors d'une simu "Couches" ----------*/
/*-------------------------------------------------------------*/


          public void RunCouches()
          {
               if(!erreur)
               {
                    LienVue1.setValue(true);

                    boolean ancien;
                    String w_Localisation;
                    couche w_CoucheVisitée,w_AncienneCouche;

                    MonRéseau.SetSleepTime(vitesse.getValue());

                    MonRéseau.EnableAllThreads();
                    Pause=false;

                    liaisonVCC LiaisonVCC=new liaisonVCC(HoteUN,HoteDEUX);

                    coucheAPL CoucheAPL= (coucheAPL)
HoteUN.GetCoucheWithNom("APL");

          CoucheAPL.GetRéfObjetGraphique().ChangeCouleurCouche(CoucheAPL,

                     CouleurMiseEnEvidence);
                          //dessin dans l'applet en correspondance avec ce
que fait la couche
                    Description=" a des données à envoyer";
                    NomCouche="application";
                    gTampon.setColor(Color.black);
                    gTampon.clearRect(0,0,size().width,size().height);
                    gTampon.drawString(" "+Description+" ",280,30);
                    gTampon.drawString("La couche "+NomCouche+" :",160,30);
                    repaint();

                    MonRéseau.GoSleep();

                    CoucheAPL.GetRéfObjetGraphique(
                              ).ChangeCouleurCouche(CoucheAPL,

CoucheAPL.GetRéfObjetGraphique(

                   ).GetAncienneCouleur());


                    coucheAAL CoucheAAL = (coucheAAL)
HoteUN.GetCoucheWithNom("AAL");

          CoucheAAL.GetRéfObjetGraphique().ChangeCouleurCouche(CoucheAAL,
```

```
                    CouleurMiseEnEvidence);
                    //dessin dans l'applet en correspondance avec ce que fait
la couche
                    NomCouche="AAL";
                    Description=" segmente et adapte les flux d'informations
"+
                                        " en fonction de la classe de service
(AAL1, AAL2, ...)";
                    gTampon.setColor(Color.black);
                    gTampon.clearRect(0,0,size().width,size().height);
                    gTampon.drawString(" "+Description+" ",250,30);
                    gTampon.drawString("La couche "+NomCouche+" :",160,30);

      gTampon.setColor(CoucheAAL.GetRéfObjetGraphique().GetAncienneCouleur(
));
                    gTampon.fillRect(400,90,400,20);
                    gTampon.setColor(Color.black);
                    gTampon.drawString(" Charge Utile (48 octets) ",500,105);
                    repaint();

                    MonRéseau.GoSleep();

                    w_AncienneCouche=CoucheAAL;//permet de savoir quelle
couche précède
                                                        //celle que l'on visite

                    CoucheAAL.DataREQ(LiaisonVCC, 1);

                    while (LiaisonVCC.GetListeTrames().size()==0)
                    {
                          MonRéseau.AbandonCPU();
                    }
                    MonRéseau.AbandonCPU();

                    celluleATM
cell=(celluleATM)((LiaisonVCC.GetListeTrames()).firstElement());
                          //la boucle while nous assure que l'affectation
dans
                          //la cellule ne sera pas nulle


                    while (LiaisonVCC.GetNbreCellulesTransmises()>

        LiaisonVCC.GetNbreCellulesRecues())
                    {
                          gTampon.clearRect(0,0,size().width,size().height);


      w_Localisation=(String)(cell.GetRéfLocalisation().elementAt(0));

                          if((w_Localisation=="2"))
                          {
                                DescriptionLien(cell);

                                if(w_AncienneCouche!=null)
                                {
                                      RestaurerCouleur(w_AncienneCouche);
                                      w_AncienneCouche=null;
                                      try{Thread.sleep(50);}
                                      catch(InterruptedException e){}
```

```
                                    }
                                }
                                else
                                {

        w_CoucheVisitée=(couche)(cell.GetRéfLocalisation().elementAt(1));


        if(w_CoucheVisitée.GetRéfObjetGraphique().GetAncienneCouleur()==

        w_CoucheVisitée.GetRéfObjetGraphique(

              ).GetRéfAttributs().GetCouleur())
                                    {
                                          DescriptionCouche(w_CoucheVisitée,cell);

                                          if(w_AncienneCouche!=null)
                                          {

        RestaurerCouleur(w_AncienneCouche);
                                          }
                                          w_AncienneCouche=w_CoucheVisitée;
                                          w_CoucheVisitée.GetRéfObjetGraphique(

).ChangeCouleurCouche(w_CoucheVisitée,CouleurMiseEnEvidence);
                                    }
                                }
                                MonRéseau.AbandonCPU();
                                if (MonRéseau.NombreThreads()==0)
                                {
                                      MonRéseau.DelAllRéfLiaisonsVcc();
                                      break;
                                }

                          }
                          if(MonRéseau.NombreThreads()!=0)
                          {
                                RestaurerCouleur(w_AncienneCouche);

                                CoucheAAL = (coucheAAL)
HoteDEUX.GetCoucheWithNom("AAL");

        CoucheAAL.GetRéfObjetGraphique().ChangeCouleurCouche(CoucheAAL,

                           CouleurMiseEnEvidence);
                                //dessin dans l'applet en correspondance avec ce
que fait la couche
                                NomCouche=" AAL";
                                Description=" met en oeuvre le processus de
réassamblage des données";
                                gTampon.clearRect(0,0,size().width,size().height);
                                gTampon.setColor(Color.black);
                                gTampon.drawString(" "+Description+" ",250,30);
                                gTampon.drawString("La couche "+NomCouche+"
:",160,30);

        gTampon.setColor(CoucheAAL.GetRéfObjetGraphique().GetAncienneCouleur(
));
                                gTampon.fillRect(400,90,400,20);
                                gTampon.setColor(Color.black);
```

```
                        gTampon.drawString(" Charge Utile (48 octets)
",500,105);
                        repaint();

                        MonRéseau.GoSleep();

                        CoucheAAL.GetRéfObjetGraphique(
                            ).ChangeCouleurCouche(CoucheAAL,

      CoucheAAL.GetRéfObjetGraphique(

).GetAncienneCouleur());
                              //permet de remmettre la couche dans son
ancienne couleur.

                        CoucheAPL= (coucheAPL)
HoteDEUX.GetCoucheWithNom("APL");

      CoucheAPL.GetRéfObjetGraphique().ChangeCouleurCouche(CoucheAPL,

CouleurMiseEnEvidence);

                        //dessin dans l'applet en correspondance avecce que
fait la couche
                        Description=" reçoit les données transmises";
                        NomCouche="application";
                        gTampon.clearRect(0,0,size().width,size().height);
                        gTampon.setColor(Color.black);
                        gTampon.drawString(" "+Description+" ",280,30);
                        gTampon.drawString("La couche "+NomCouche+"
:",160,30);
                        repaint();

                        MonRéseau.GoSleep();


      CoucheAPL.GetRéfObjetGraphique().ChangeCouleurCouche(CoucheAPL,

            CoucheAPL.GetRéfObjetGraphique(

                  ).GetAncienneCouleur());

                        gTampon.clearRect(0,0,size().width,size().height);
                        repaint(); //on nettoie la surface d'affichage.
                        actif=null;
                  }

                  gTampon.clearRect(0,0,size().width,size().height);
                  repaint(); //on nettoie la surface d'affichage.
                  actif=null;
            }
            else
            {
                  gTampon.clearRect(0,0,size().width,size().height);
                  gTampon.drawString("Un problème est survenu lors de
l'initialisation de VRML.",150,70);
                  repaint();
                  actif=null;
            }
      }
```

```
/*-------------------------------------------------------------*/
/*-- Méthodes Liées à la simu "Commutation" ----------------*/
/*-------------------------------------------------------------*/

      public void DessinRoute()
      {
            gTampon.setColor(Color.blue);
            gTampon.drawString("A->",120,62);
            gTampon.drawString("C->",120,110);
            gTampon.drawString("->",360,62);
            gTampon.drawString("->",360,110);
            gTampon.drawString("->B",610,62);
            gTampon.drawString("->",610,110);
            gTampon.drawString("->D",850,110);

      }
/*-------------------------------------------------------------*/

      public void DessinTableUn()
      {
            gTampon.setColor(Color.blue);
            gTampon.drawString("Commutateur 1",140,13);

            gTampon.setColor(Color.lightGray);
            gTampon.fill3DRect(140,20,200,110,true);

            gTampon.setColor(Color.black);
            gTampon.drawLine(140,38,340,38);
            gTampon.drawLine(140,40,340,40);
            gTampon.drawLine(140,85,340,85);
            gTampon.drawLine(240,20,240,130);
            gTampon.drawString("Entrée",175,33);
            gTampon.drawString("Sortie",275,33);
                  //première ligne de la table
            gTampon.drawString("VPI "+w_Vcl1.GetVPI()+" ", 150, 55);
            gTampon.drawString("VCI "+w_Vcl1.GetNuméroLiaison()+" ",
150,70);
            gTampon.drawString("VPI "+w_Vcl2.GetVPI()+" ", 250, 55);
            gTampon.drawString("VCI "+w_Vcl2.GetNuméroLiaison()+" ",
250,70);
                  //deuxième ligne de la table
            gTampon.drawString("VPI "+w_Vcl3.GetVPI()+" ", 150, 100);
            gTampon.drawString("VCI "+w_Vcl3.GetNuméroLiaison()+" ",
150,120);
            gTampon.drawString("VPI "+w_Vcl4.GetVPI()+" ", 250,100);
            gTampon.drawString("VCI "+w_Vcl4.GetNuméroLiaison()+" ",
250,120);
      }

/*-------------------------------------------------------------*/

      public void DessinTableDeux()
      {
            gTampon.setColor(Color.blue);
            gTampon.drawString("Commutateur 2",390,13);


            gTampon.setColor(Color.lightGray);
            gTampon.fill3DRect(390,20,200,110,true);
            gTampon.setColor(Color.black);
            gTampon.drawLine(390,38,590,38);
```

```
            gTampon.drawLine(390,40,590,40);
            gTampon.drawLine(390,85,590,85);
            gTampon.drawLine(490,20,490,130);
            gTampon.drawString("Entrée",425,33);
            gTampon.drawString("Sortie",525,33);
                  //première ligne de la table
            gTampon.drawString("VPI "+w_Vcl2.GetVPI()+" ",400,55);
            gTampon.drawString("VCI "+w_Vcl2.GetNuméroLiaison()+" ",
400,70);
            gTampon.drawString("VPI "+w_Vcl20.GetVPI()+" ",500,55);
            gTampon.drawString("VCI "+w_Vcl20.GetNuméroLiaison()+"
",500,70);
                  //deuxième ligne de la table
            gTampon.drawString("VPI "+w_Vcl4.GetVPI()+" ",400,100);
            gTampon.drawString("VCI "+w_Vcl4.GetNuméroLiaison()+" ",
400,120);
            gTampon.drawString("VPI "+w_Vcl5.GetVPI()+" ",500,100);
            gTampon.drawString("VCI "+w_Vcl5.GetNuméroLiaison()+" ",
      500,120);
      }

/*-------------------------------------------------------------*/

      public void DessinTableTrois()
      {
            gTampon.setColor(Color.blue);
            gTampon.drawString("Commutateur 3",640,13);


            gTampon.setColor(Color.lightGray);
            gTampon.fill3DRect(640,20,200,110,true);
            gTampon.setColor(Color.black);
            gTampon.drawLine(640 ,38,840,38);
            gTampon.drawLine(640,40,840,40);
            gTampon.drawLine(640,85,840,85);
            gTampon.drawLine(740,20,740,130);
            gTampon.drawString("Entrée",675,33);
            gTampon.drawString("Sortie",775,33);
                  //sSeule ligne de la table
            gTampon.drawString("VPI "+w_Vcl5.GetVPI()+" ",650,100);
            gTampon.drawString("VCI "+w_Vcl5.GetNuméroLiaison()+" ",
650,120);
            gTampon.drawString("VPI "+w_Vcl50.GetVPI()+" ",750,100);
            gTampon.drawString("VCI "+w_Vcl50.GetNuméroLiaison()+"
",750,120);
      }

/*-------------------------------------------------------------*/

      public void ChangeLigneUnTableUn()
      {
            gTampon.drawString("VPI "+w_Vcl1.GetVPI()+" ", 150, 55);
            gTampon.drawString("VCI "+w_Vcl1.GetNuméroLiaison()+" ",
150,70);
            gTampon.drawString("VPI "+w_Vcl2.GetVPI()+" ", 250, 55);
            gTampon.drawString("VCI "+w_Vcl2.GetNuméroLiaison()+" ",
250,70);
      }

/*-------------------------------------------------------------*/
```

```
	public void ChangeLigneDeuxTableUn()
	{
		gTampon.drawString("VPI "+w_Vcl3.GetVPI()+" ", 150, 100);
		gTampon.drawString("VCI "+w_Vcl3.GetNuméroLiaison()+" ",
150,120);
		gTampon.drawString("VPI "+w_Vcl4.GetVPI()+" ", 250,100);
		gTampon.drawString("VCI "+w_Vcl4.GetNuméroLiaison()+" ",
250,120);
	}

/*-------------------------------------------------------------*/

	public void ChangeLigneUnTableDeux()
	{
		gTampon.drawString("VPI "+w_Vcl2.GetVPI()+" ",400,55);
		gTampon.drawString("VCI "+w_Vcl2.GetNuméroLiaison()+" ",
400,70);
		gTampon.drawString("VPI "+w_Vcl20.GetVPI()+" ",500,55);
		gTampon.drawString("VCI "+w_Vcl20.GetNuméroLiaison()+"
",500,70);
	}

/*-------------------------------------------------------------*/

	public void ChangeLigneDeuxTableDeux()
	{
		gTampon.drawString("VPI "+w_Vcl4.GetVPI()+" ",400,100);
		gTampon.drawString("VCI "+w_Vcl4.GetNuméroLiaison()+" ",
400,120);
		gTampon.drawString("VPI "+w_Vcl5.GetVPI()+" ",500,100);
		gTampon.drawString("VCI "+w_Vcl5.GetNuméroLiaison()+" ",
	500,120);
	}

/*-------------------------------------------------------------*/

	public void ChangeLigneTableTrois()
	{
		gTampon.drawString("VPI "+w_Vcl5.GetVPI()+" ",650,100);
		gTampon.drawString("VCI "+w_Vcl5.GetNuméroLiaison()+" ",
650,120);
		gTampon.drawString("VPI "+w_Vcl50.GetVPI()+" ",750,100);
		gTampon.drawString("VCI "+w_Vcl50.GetNuméroLiaison()+"
",750,120);
	}

/*-------------------------------------------------------------*/
/*-- Méthodes Liées à la simu "Données" ---------------------*/
/*-------------------------------------------------------------*/


	public void DessinerRoute()
	{
		gTampon.drawString("VCC de C -> B = ",160,20);
		gTampon.drawString(" VPI: "+w_Vcl1.GetVPI()+" , ",260,20);
		gTampon.drawString("VCI: "+w_Vcl1.GetNuméroLiaison()+" ->
",310,20);
		gTampon.drawString(" VPI: "+w_Vcl2.GetVPI()+" , ",370,20);
		gTampon.drawString("VCI: "+w_Vcl2.GetNuméroLiaison()+" ->
",420,20);
		gTampon.drawString(" VPI: "+w_Vcl20.GetVPI()+" , ",480,20);
```

```
            gTampon.drawString("VCI: "+w_Vcl20.GetNuméroLiaison()+"
",530,20);

      }

/*-------------------------------------------------------*/

      public void DessinerCellule(celluleATM cell)
      {
            gTampon.setColor(w_CouleurAAL);
            gTampon.drawRect(160,30,40,20);
            gTampon.drawString("GFC",165,45);
            gTampon.drawRect(200,30,40,20);
            gTampon.drawString("VPI",205,45);
            gTampon.drawRect(240,30,40,20);
            gTampon.drawString("VCI",245,45);
            gTampon.drawRect(280,30,40,20);
            gTampon.drawString("PT",285,45);
            gTampon.drawRect(320,30,40,20);
            gTampon.drawString("CLP",325,45);
            gTampon.drawRect(360,30,40,20);
            gTampon.drawString("HEC",365,45);
            gTampon.fillRect(400,30,400,20);

            gTampon.setColor(Color.black);
            gTampon.drawString(" Charge Utile [PAYLOAD](48 Octets)
",500,45);

            gTampon.setColor(Color.red);
            gTampon.drawString(" "+cell.GetVPI()+" ",223,45);
            gTampon.drawString(" "+cell.GetNuméroVcl()+" ",263,45);

      }

/*-------------------------------------------------------*/

      public void DessinerInformation()
      {
            gTampon.setColor(Color.black);
            gTampon.drawString("GFC = Generic Flow Control",160,80);
            gTampon.drawString("VPI = Virtual Path Identifier",160,100);
            gTampon.drawString("VCI = Virtual Channel Identifier",160,120);
            gTampon.drawString("PT  = Payload Type",400,80);
            gTampon.drawString("CLP = Cell Loss Priority",400,100);
            gTampon.drawString("HEC = Header Error  Control",400,120);
      }

/*-------------------------------------------------------*/
/*-- Méthodes Liées à la simu "Couches" ----------------*/
/*-------------------------------------------------------*/

      public void DescriptionLien(celluleATM f_RéfCellule)
      {
            Color
w_CouleurPHY=HoteUN.GetCoucheWithNom("PHY").GetRéfObjetGraphique(

            ).GetAncienneCouleur();

            Color
w_CouleurATM=HoteUN.GetCoucheWithNom("ATM").GetRéfObjetGraphique(
```

```
            ).GetAncienneCouleur();

            Color
w_CouleurAAL=HoteUN.GetCoucheWithNom("AAL").GetRéfObjetGraphique(

            ).GetAncienneCouleur();

            Description= " la cellule est acheminée sur le support
physique";
            NomCouche = "Lien Physique";

            gTampon.setColor(w_CouleurATM);
            gTampon.drawRect(160,90,40,20);
            gTampon.drawString("GFC",165,105);
            gTampon.drawRect(200,90,40,20);
            gTampon.drawString("VPI "+f_RéfCellule.GetVPI()+" ",205,105);
            gTampon.drawRect(240,90,40,20);
            gTampon.drawString("VCI "+f_RéfCellule.GetNuméroVcl()+"
",245,105);
            gTampon.drawRect(280,90,40,20);
            gTampon.drawString("PT",285,105);
            gTampon.drawRect(320,90,40,20);
            gTampon.drawString("CLP",325,105);

            gTampon.setColor(w_CouleurPHY);

            gTampon.clearRect(360,90,40,20);
            gTampon.drawRect(360,90,40,20);
            gTampon.drawString("HEC",365,105);

            gTampon.setColor(w_CouleurAAL);
            gTampon.fillRect(400,90,400,20);
            gTampon.setColor(Color.black);
            gTampon.drawString(" Charge Utile (48 octets) ",500,105);

            gTampon.setColor(Color.black);
            gTampon.drawString(" "+Description+" ",250,30);
            gTampon.drawString(" "+NomCouche+" : ",160,30);
            repaint();

      }

      public void DescriptionCouche(couche f_RéfCouche, celluleATM
f_RéfCellule)
      {
            Color w_CouleurATM=Color.magenta;
            Color w_CouleurPHY=Color.magenta;
            Color w_CouleurAAL=Color.magenta;

            String Description2;

            couche w_Couche;


      w_CouleurAAL=HoteUN.GetCoucheWithNom("AAL").GetRéfObjetGraphique(

            ).GetAncienneCouleur();

            gTampon.setColor(w_CouleurAAL);
            gTampon.fillRect(400,90,400,20);
```

```
            gTampon.setColor(Color.black);
            gTampon.drawString(" Charge Utile (48 octets) ",500,105);


            NomCouche=f_RéfCouche.GetNomCouche();

      if((String)(f_RéfCellule.GetRéfLocalisation().elementAt(0))=="0")
            //la cellule descend dans les couches
            {
                  if(f_RéfCouche.GetNomCouche()=="ATM")
                  {
                        w_CouleurATM=f_RéfCouche.GetRéfObjetGraphique(

      ).GetAncienneCouleur();

                        Description=" construit l'entête de la cellule et"+
                                          " associe l'identificateur
logique.";

                        gTampon.setColor(w_CouleurATM);
                        gTampon.drawRect(160,90,40,20);
                        gTampon.drawString("GFC",165,105);
                        gTampon.drawRect(200,90,40,20);
                        gTampon.drawString("VPI "+f_RéfCellule.GetVPI()+"
",205,105);
                        gTampon.drawRect(240,90,40,20);
                        gTampon.drawString("VCI
"+f_RéfCellule.GetNuméroVcl()+" ",245,105);
                        gTampon.drawRect(280,90,40,20);
                        gTampon.drawString("PT",285,105);
                        gTampon.drawRect(320,90,40,20);
                        gTampon.drawString("CLP",325,105);
                        gTampon.fillRect(360,90,40,20);

                  }
                  else
                  {
                        w_CouleurPHY=f_RéfCouche.GetRéfObjetGraphique(

      ).GetRéfAttributs().GetCouleur();

                        w_Couche=

      f_RéfCouche.GetCoucheWithNom(f_RéfCouche.GetListeCouchesSup(),"ATM");


      w_CouleurATM=w_Couche.GetRéfObjetGraphique().GetAncienneCouleur();

                        Description=" adapte les informations "+
                                          " au support physique et calcule
le champ de "+
                                          " contrôle de l'entête (HEC)";


                        gTampon.setColor(w_CouleurATM);
                        gTampon.drawRect(160,90,40,20);
                        gTampon.drawString("GFC",165,105);
                        gTampon.drawRect(200,90,40,20);
                        gTampon.drawString("VPI "+f_RéfCellule.GetVPI()+"
",205,105);
                        gTampon.drawRect(240,90,40,20);
```

```
                        gTampon.drawString("VCI
"+f_RéfCellule.GetNuméroVcl()+" ",245,105);
                        gTampon.drawRect(280,90,40,20);
                        gTampon.drawString("PT",285,105);
                        gTampon.drawRect(320,90,40,20);
                        gTampon.drawString("CLP",325,105);

                        gTampon.setColor(w_CouleurPHY);

                        gTampon.clearRect(360,90,40,20);
                        gTampon.drawRect(360,90,40,20);
                        gTampon.drawString("HEC",365,105);
                  }
            }
            else
            //la cellule monte dans les couches
            {
                  if(f_RéfCouche.GetNomCouche()=="ATM")
                  {
                        w_CouleurATM=f_RéfCouche.GetRéfObjetGraphique(

      ).GetAncienneCouleur();


      if(f_RéfCouche.GetRéfEquipement().GetTypeEquipement() == 1)
                        {
                              //la cellule est dans une machine hote
                              Description=" commute la cellule en fonction
des informations contenues dans la";
                              Description2=" table de routage du
commutateur";
                        }
                        else
                        {
                              //la celllule est dans un commutateur
                              Description=" supprime l'entête et remet à la
couche AAL correspondante les ";
                              Description2=" données de la celllule";
                        }

                        gTampon.drawString(" "+Description2+" ",250,45);
                        gTampon.setColor(w_CouleurATM);
                        gTampon.drawRect(160,90,40,20);
                        gTampon.drawString("GFC",165,105);
                        gTampon.drawRect(200,90,40,20);
                        gTampon.drawString("VPI "+f_RéfCellule.GetVPI()+"
",205,105);
                        gTampon.drawRect(240,90,40,20);
                        gTampon.drawString("VCI
"+f_RéfCellule.GetNuméroVcl()+" ",245,105);
                        gTampon.drawRect(280,90,40,20);
                        gTampon.drawString("PT",285,105);
                        gTampon.drawRect(320,90,40,20);
                        gTampon.drawString("CLP",325,105);
                        gTampon.fillRect(360,90,40,20);

                  }
                  else
                  {
                        w_CouleurPHY=f_RéfCouche.GetRéfObjetGraphique(
```

```
      ).GetAncienneCouleur();

                        couche w_couche=

      f_RéfCouche.GetCoucheWithNom(f_RéfCouche.GetListeCouchesSup(),"ATM");


      w_CouleurATM=w_couche.GetRéfObjetGraphique().GetAncienneCouleur();

                        Description=" transforme les signaux binaires en
cellule et vérifie le champ HEC";

                        gTampon.setColor(w_CouleurATM);
                        gTampon.drawRect(160,90,40,20);
                        gTampon.drawString("GFC",165,105);
                        gTampon.drawRect(200,90,40,20);
                        gTampon.drawString("VPI "+f_RéfCellule.GetVPI()+"
",205,105);
                        gTampon.drawRect(240,90,40,20);
                        gTampon.drawString("VCI
"+f_RéfCellule.GetNuméroVcl()+" ",245,105);
                        gTampon.drawRect(280,90,40,20);
                        gTampon.drawString("PT",285,105);
                        gTampon.drawRect(320,90,40,20);
                        gTampon.drawString("CLP",325,105);

                        gTampon.setColor(w_CouleurPHY);

                        gTampon.fillRect(360,90,40,20);
                  }
            }
            gTampon.setColor(Color.black);
            gTampon.drawString(" "+Description+" ",250,30);
            gTampon.drawString("Couche "+NomCouche+" : ",160,30);
            repaint();
      }

/*-------------------------------------------------------------*/

      public void RestaurerCouleur(couche f_RéfCouche)
      {
            f_RéfCouche.GetRéfObjetGraphique(
                        ).ChangeCouleurCouche(f_RéfCouche,

f_RéfCouche.GetRéfObjetGraphique(

).GetAncienneCouleur());
      }

/*-------------------------------------------------------------*/

      public void RestaurerTout()
      {
            equipement w_RéfEquipement;
            couche w_RéfCouche;
            int i=0;
            int j=0;
            Vector w_RéfListeEquipement=MonRéseau.GetListeEquipements();
            Vector w_RéfListeCouche;
```

```
            while(i<w_RéfListeEquipement.size())
            {

      w_RéfEquipement=(equipement)(w_RéfListeEquipement.elementAt(i));
                  w_RéfListeCouche=w_RéfEquipement.GetListeCouches();

                  while(j<w_RéfListeCouche.size())
                  {


      w_RéfCouche=(couche)(w_RéfListeCouche.elementAt(j));


      if(w_RéfCouche.GetRéfObjetGraphique().GetRéfAttributs().GetCouleur()
!=

      w_RéfCouche.GetRéfObjetGraphique().GetAncienneCouleur())
                        {
                              RestaurerCouleur(w_RéfCouche);
                        }
                        j++;
                  }
                  i++;
                  j=0;
            }
      }

/*-------------------------------------------------------------*/
/*-- Méthodes Liées à l'initialisation de l'applet ---------*/
/*-------------------------------------------------------------*/

      public void InitRéseauEtHotes()
            {
            MonRéseau=new sousReseauATM("200.200.");
            pMonRéseau=new P_sousReseauATM(MonRéseau,
                                                browser,
                                                addChildren,
removeChildren,
                                                addCell,
removeCell);

            HoteUN=new machineHote("100", MonRéseau);
            HoteDEUX=new machineHote("200", MonRéseau);
            HoteTROIS=new machineHote("300", MonRéseau);
            HoteQUATRE=new machineHote("400", MonRéseau);


            position1[0]=(float)(-6.5);
            position1[1]=(float)(-2);
            position1[2]=(float)(-14.5);
            pHoteUN=new P_machineHote(HoteUN, position1);

            position2[0]=(float)(6.5);
            position2[1]=(float)(-2);
            position2[2]=(float)(-14.5);
            pHoteDEUX=new P_machineHote(HoteDEUX, position2);

            position3[0]=(float)(-6.5);
            position3[1]=(float)(-2);
            position3[2]=(float)(-1.5);
            pHoteTROIS=new P_machineHote(HoteTROIS, position3);
```

```
            position4[0]=(float)(6.5);
            position4[1]=(float)(-2);
            position4[2]=(float)(-1.5);
            pHoteQUATRE=new P_machineHote(HoteQUATRE, position4);
      }

/*-------------------------------------------------------------*/

      public void InitSwitches()
      {
            SwitchUN=new commutateurATM(MonRéseau);
            SwitchDEUX=new commutateurATM(MonRéseau);
            SwitchTROIS=new commutateurATM(MonRéseau);

            position5[0]=(float)(-2.5);
            position5[1]=(float)(-2);
            position5[2]=(float)(-8.5);
            pSwitchUN=new P_commutateurATM(SwitchUN,position5);

            position6[0]=(float)(0.5);
            position6[1]=(float)(-2);
            position6[2]=(float)(-10.5);
            pSwitchDEUX=new P_commutateurATM(SwitchDEUX, position6);

            position7[0]=(float)(2.5);
            position7[1]=(float)(-2);
            position7[2]=(float)(-6.5);
            pSwitchTROIS=new P_commutateurATM(SwitchTROIS, position7);
      }

/*-------------------------------------------------------------*/

      public void InitLiaisonsPhy()
      {
            Liaison01=new liaisonPHY(1, HoteUN, SwitchUN);
            Liaison02=new liaisonPHY(2, HoteTROIS,SwitchUN);
            Liaison03=new liaisonPHY(3, SwitchUN,SwitchTROIS);
            Liaison05=new liaisonPHY(5, SwitchDEUX, SwitchTROIS);
            Liaison04=new liaisonPHY(4, SwitchDEUX,SwitchUN);
            Liaison06=new liaisonPHY(6, SwitchDEUX, HoteDEUX);
            Liaison07=new liaisonPHY(7, SwitchTROIS, HoteQUATRE);

            AttributsLiens AttriLien1=
                  new AttributsLiens(-4.5f,-2.25f,-
11.5f,Color.black,0,1,0,-0.98f,0.05f,7.2f);

            pLiaison01=new P_liaisonPHY(Liaison01, AttriLien1);

            AttributsLiens AttriLien2=
                  new AttributsLiens(-4.5f,-2.25f,-
5f,Color.black,0,1,0,1.04f,0.05f,8.1f);

            pLiaison02=new P_liaisonPHY(Liaison02, AttriLien2);

            AttributsLiens AttriLien3=
                  new AttributsLiens(0,-2.25f,-7.5f,Color.black,0,1,0,-
0.37f,0.05f,5.4f);

            pLiaison03=new P_liaisonPHY(Liaison03, AttriLien3);
```

```
            AttributsLiens AttriLien4=
                  new AttributsLiens(-1f,-2.25f,-
9.5f,Color.black,0,1,0,0.57f,0.05f,3.6f);

            pLiaison04=new P_liaisonPHY(Liaison04, AttriLien4);

            AttributsLiens AttriLien5=
                  new AttributsLiens(1.5f,-2.25f,-8.5f,Color.black,0,1,0,-
1.09f,0.05f,4.4f);

            pLiaison05=new P_liaisonPHY(Liaison05, AttriLien5);

            AttributsLiens AttriLien6=
                  new AttributsLiens(3.5f,-2.25f,-
12.5f,Color.black,0,1,0,0.57f,0.05f,7.2f);

            pLiaison06=new P_liaisonPHY(Liaison06, AttriLien6);

            AttributsLiens AttriLien7=
                  new AttributsLiens(4.5f,-2.25f,-4f,Color.black,0,1,0,-
0.89f,0.05f,6.5f);

            pLiaison07=new P_liaisonPHY(Liaison07, AttriLien7);
      }

/*-------------------------------------------------------------*/

      public void InitCouches()
      {
            MonRéseau.AdapterCouches();
            pMonRéseau.AdapterGraphique();
      }

/*-------------------------------------------------------------*/

      public void InitAdresses()
      {
            SwitchUN.AddRéfAdresseAtm(HoteUN, Liaison01);
            SwitchUN.AddRéfAdresseAtm(HoteDEUX, Liaison04);
            SwitchUN.AddRéfAdresseAtm(HoteTROIS, Liaison02);
            SwitchUN.AddRéfAdresseAtm(HoteQUATRE, Liaison04);

//
            SwitchDEUX.AddRéfAdresseAtm(HoteUN, Liaison04);
            SwitchDEUX.AddRéfAdresseAtm(HoteDEUX, Liaison06);
            SwitchDEUX.AddRéfAdresseAtm(HoteTROIS, Liaison04);
            SwitchDEUX.AddRéfAdresseAtm(HoteQUATRE, Liaison05);

//
            SwitchTROIS.AddRéfAdresseAtm(HoteUN, Liaison03);
            SwitchTROIS.AddRéfAdresseAtm(HoteDEUX, Liaison05);
            SwitchTROIS.AddRéfAdresseAtm(HoteTROIS, Liaison03);
            SwitchTROIS.AddRéfAdresseAtm(HoteQUATRE, Liaison07);

      }
}

/*=============================================================*/
/*=============================================================*/
```

V. Page HTML

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Developer Studio">
<META HTTP-EQUIV="Content-Type" content="text/html; charset=iso-8859-1">
<TITLE>Document Title</TITLE>
</HEAD>
<BODY>
      <embed src="genere1.wrl" border=0 width=100% height=75%>
      <Applet code="SimuLibre.class" mayscript width=100%
height=25%></Applet>

</BODY>
</HTML>
```

VI. Monde VRML

```
#VRML V2.0 utf8

#---------------------------------------------------------------
#--------------- Définition du Background ---------------------
#---------------------------------------------------------------

Background{
      groundColor .5 .5 .5
      skyColor .5 .5 .5
}

#---------------------------------------------------------------
#--- Définition du sol sur lequel se trouve le réseau --------
#---------------------------------------------------------------

Transform{
      translation -12 -2.25 -17
      children[
            Shape{
                  appearance Appearance{
                        material Material{
                              diffuseColor .95 .81 .99
                        }
                  }
                  geometry ElevationGrid{
                        xDimension 25
                        xSpacing 1
                        zDimension 20
                        zSpacing 1
                        height
      [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

```
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                        ]
                }
        }
    ]
}

#-------------------------------------------------------------
#---- Définition des noeuds complétés par Java ---------------
#-------------------------------------------------------------


DEF MONDE Group{
    children[
    ]
}

DEF DYNAMIQUE Group{
    children[
    ]
}


#-------------------------------------------------------------
#-- Définition des autres noeuds VRML (sans java) -----------
#-------------------------------------------------------------

DEF TEMPS TimeSensor{
    loop TRUE
    cycleInterval .1
}
```