MASTER EN SCIENCES INFORMATIQUES

Outil d'apprentissage de l'algorithmique via le Web

Collin, Nathalie

Award date:
2003

FUNDP
INSTITUT D'INFORMATIQUE

Outil d'apprentissage de l'algorithmique via le web

Nathalie Collin

Mémoire présenté en vue de l'obtention du grade de
Licenciée en Informatique.

Année académique 2002-2003

Résumé

L'algorithmique constitue une matière difficile à faire passer auprès des étudiants sous forme magistrale. Une approche interactive semble plus pertinente tant pour permettre l'illustration d'un exposé magistral que pour la phase d'appropriation individuelle de la matière par les étudiants. Ceci induit l'adaptation des nouvelles technologies à la pédagogie et suppose une restructuration et une réorganisation des contenus pédagogiques et des scénarios d'apprentissage, ainsi qu'un réaménagement en profondeur des programmes existants.

Ce mémoire s'inscrit dans ce contexte. Après avoir exposé la pertinence de la méthodologie par prototypage évolutif, il propose un outil de visualisation sous divers modes des phases d'exécution d'un algorithme à travers une interface web et en donne une implémentation en utilisant le langage de programmation orienté objets Java. Ensuite, pour inscrire son action au service d'une véritable politique de changement, il présente les perspectives d'utilisation de l'outil implémenté.

Mos-clefs : *algorithme, animation, applet, générateur de code, Java, mini-compilateur, nouvelles technologies de l'information et de la communication, nouvelles technologies éducatives, programmation orientée objets, prototype évolutif, simulation.*

Abstract

Algorithmic lectures constitute a difficult subject to teach. An interactive approach seems more relevant to illustrate a lecture as well as to help students to appropriate the subject on their own. This results in the adaptation of new technologies in educational methods and implies a restructuring and reorganization of educational contents and learning methods, as well as a deep reorganization of the current curriculums.

This dissertation falls within that context. After the presentation of the relevance of the methodology by evolutionary prototyping, it includes a visualization tool in different ways of the running stages of an algorithm by means of a web interface using the Java object oriented programming language. Finally, it sets out the use prospects of the implemented tool so that it can serve a real policy of change.

Keywords : *algorithm, animation, applet, code generator, Java, mini-compiler, new information and communication technologies, new educational technologies, object oriented programming, evolutionary prototyping , simulation*

Avant-Propos

Voici un travail qui clôture trois années passées à l'Institut d'Informatique des Facultés Universitaires Notre-Dame de la Paix de Namur.

J'ai été heureuse de m'investir dans ce mémoire, en raison des nombreuses satisfactions que sa réalisation m'a apportées.

C'est donc empreinte d'une certaine joie que je termine celui-ci, la joie d'avoir approfondi mes connaissances et d'avoir repoussé mes limites.

Ce que j'ai le plus apprécié tient, sans aucun doute, à l'interdisciplinarité latente présente tout au long de ce travail.

Je ne souhaitais pas conclure cet avant-propos sans procéder à une série de remerciements.

C'est tout naturellement à mon promoteur, le professeur Naji Habra, qui a supervisé ce mémoire, que j'adresse mes plus vifs remerciements. Ce qui n'était au départ qu'un amas informe d'objectifs est devenu, grâce à ses conseils avertis, un tout cohérent.

Je remercie le professeur Jean-Marie Jacquet et l'assistant François Schoubben pour les pistes pertinentes qu'ils m'ont proposées et que j'ai suivies.

Je remercie Isabelle, Joëlle et Christophe, membres de mon « comité de lecture », pour avoir accepté d'en faire partie et pour le regard critique qu'ils n'ont pas hésité à poser.

Je remercie Mesdames d'Udekem d'Acoz-Gevers et Lobet-Maris pour avoir donné un visage humain à ces années de licence. Elles m'ont, toutes deux, donné l'impression d'estimer avec grande justesse, la difficulté que représente, au niveau individuel, la combinaison d'une vie professionnelle, d'une vie familiale et d'une formation universitaire à horaire décalé.

Je remercie Madame Massart pour son efficacité, sa compétence et sa réserve inégalables.

Je remercie la famille Lfahem d'avoir fleuri la petite chambre que je leur ai louée pour peaufiner la rédaction de mon mémoire. Les petits bonbons étaient très bons. Ce sont ces petites attentions qui ont rendu la phase de rédaction moins pénible en cet été caniculaire.

Enfin, j'exprime ma gratitude et ma reconnaissance appuyée à celui qui m'accompagne depuis quelques années dans la vie. Il a été à mes côtés durant ces trois années qui se seront révélées être des années faites de doutes et de découragements mais aussi de satisfactions et de bonheurs ... grâce à lui. Si ce travail a pu être mené à son terme, c'est sans doute dû, pour une grande part, à sa présence à mes côtés. Il n'a de cesse de croire en moi et c'est en lui que je puise force et courage.

Je dédie ce mémoire à Alexis et à ses deux premières dents !
Puisse-t-il, lui aussi, exercer plus tard une profession qui le passionne ...

Table des matières

Liste des Figures

Liste des Tableaux

Chapitre 1

Introduction

L'algorithmique constitue une matière difficile à faire passer auprès des étudiants sous forme magistrale. Une approche interactive semble plus pertinente pour son assimilation.

Le point de départ de ce travail est le souhait d'un professeur d'algorithmique de disposer d'un outil permettant la visualisation sous divers modes des phases d'exécution d'un algorithme.

Il s'agit d'une part, de permettre l'illustration d'un exposé magistral et d'autre part, de fournir un outil manipulable par les étudiants lors de leur phase d'appropriation individuelle de la matière.

Les nouvelles technologies de l'information et de la communication offrent une grande variété d'outils pouvant contribuer à enrichir le matériel didactique d'un cours et jouissent d'une popularité sans cesse croissante dans l'enseignement universitaire. Le réseau Internet, par exemple, est de plus en plus utilisé comme support à l'apprentissage.

Nous profitons de cette mouvance pour concevoir un outil interactif d'apprentissage de l'algorithmique via le web et pour nous interroger sur la façon de l'intégrer efficacement dans le curriculum des étudiants.

Le résultat de ce travail est une application client-serveur opérationnelle, écrite en Java.

Ce mémoire est structuré en trois parties : une première partie, introductive, place le projet dans son contexte et présente la méthodologie de travail retenue ; une deuxième partie est dédiée aux résultats obtenus, par l'application de cette méthodologie ; et une dernière partie tire les conclusions sur le travail réalisé et ouvre de nouvelles perspectives.

Une annexe reprend ensuite une série de documents illustrant différentes parties de notre travail.

Première Partie

Le **chapitre 2** présente le contexte général des approches pédagogiques en usage actuellement à l'université et met en évidence l'importance de l'introduction des nouvelles technologies éducatives. Il énonce les objectifs visés par le présent travail. Une analyse de l'état de l'art conduit à la justification des choix à suivre.

Le **chapitre 3** argumente l'originalité, le bien-fondé et la pertinence de notre projet et en pose les jalons méthodologiques. Il fixe également les limites raisonnables nécessaires pour livrer un tout cohérent dans le temps imparti.

Deuxième partie

Le **chapitre 4** analyse en profondeur les besoins sur base d'un recensement des demandes du professeur et énonce les bénéfices escomptés.

Le **chapitre 5** spécifie les fonctionnalités de l'outil et propose une découpe en sous-fonctions. Ces sous-fonctions sont ensuite regroupées en différents modules. Sur base de ces découpes, des outils seront définis.

Le **chapitre 6** présente les composantes graphiques de l'interface hommes-machine de l'outil. Ceux-ci sont le résultat d'une analyse approfondie de la tâche principale.

Le **chapitre 7** propose une modélisation architecturale de l'outil. Il commence par une étude des normes existantes et expose les choix techniques qui ont été faits. Une architecture de générateur de code est ensuite proposée. Les objets définitifs de l'interface graphiques sont présentés. Et une solution pour lier le programme implémentant un algorithme et les représentations visuelles de l'exécution de celui-ci est proposée.

Le **chapitre 8** met en évidence les difficultés de codage qu'a engendré l'implémentation des différents modules de l'outil.

Le **chapitre 9** expose le déploiement du logiciel.

Le **chapitre 10** présente l'utilisation de l'outil, ainsi que ses limites.

Troisième partie

Le **chapitre 11** présente nos conclusions ainsi que les perspectives d'amélioration de notre mémoire.

Quatrième partie

Les documents fournis en **annexe** comprennent entre autres, le manuel de l'utilisateur apprenant, le manuel de l'utilisateur traducteur, la spécification de langage LSD^{03} et les copies d'écran de l'interface graphique de l'outil d'apprentissage de l'algorithmique.

Première Partie

Le projet

Chapitre 2

Contexte

2.1. Le contexte général

2.1.1. Les approches pédagogiques en usage à l'université

L'exposé magistral de type conférence caractérise le mieux l'enseignement universitaire tel que majoritairement dispensé actuellement. Cette approche pédagogique a pour effet de cantonner les étudiants dans une attitude d'écoute passive.

Les nouvelles technologies de l'information et de la communication (communément appelées les NTIC) ne semblent jouer encore qu'un rôle très limité dans la formation universitaire.

Les effets de l'enseignement magistral ont été analysés par [DIM98] lors d'un séminaire sur les méthodes d'enseignement. De cela retenons que l'exposé magistral convient parfaitement à l'enseignement devant de grands groupes d'étudiants et permet à l'enseignant de communiquer un maximum d'informations en un minimum de temps. Nous pouvons donc parler d'une formule d'enseignement très économique, ce qui constitue un avantage certain lorsqu'on connaît les maigres ressources financières des universités. Elle se veut en outre très rassurante pour l'enseignant puisqu'elle lui offre une maîtrise complète des événements mais également pour l'étudiant qui est parfaitement familiarisé avec ce type d'enseignement.

L'exposé magistral devant un grand nombre d'étudiants permet de présenter rapidement et efficacement les concepts à transmettre et idéalement de terminer par une synthèse de la leçon et ce, sans l'aide d'outils pédagogiques sophistiqués. Cette approche pédagogique rend les étudiants grandement tributaires de leur professeur.

L'exposé magistral comporte cependant quelques faiblesses pédagogiques importantes. En effet, même si cette formule d'enseignement permet d'atteindre simultanément un grand nombre d'étudiants, S. Hooper [HOO92] constate que plus le nombre d'étudiants augmente, plus l'effort individuel tend à diminuer, réduisant d'autant l'intérêt pour la matière présentée. En outre, cette formule présente l'inconvénient majeur de rendre quasiment impossible l'assimilation de la connaissance, telle que le permettent, par exemple, la résolution d'exercices ou les travaux pratiques.

L'acquisition des connaissances et le développement des compétences qui sont indispensables à l'obtention d'un diplôme universitaire, nécessitent des situations pédagogiques qui poussent l'étudiant à sortir de cet état de passivité.

Pour ces raisons, les nouvelles technologies de l'information et de la communication devraient apparaître de plus en plus dans les formations universitaires, à condition d'être utilisées avec discernement.

2.1.2. Pourquoi introduire de nouvelles[1] technologies en formation universitaire ?

Les enseignants tant au niveau universitaire qu'au niveau secondaire constatent que les étudiants d'aujourd'hui, et donc à fortiori de demain, éprouvent de plus en plus de difficultés à comprendre, à conceptualiser, à structurer et à intégrer une matière. Cela entraîne une perte de motivation voire d'intérêt vis-à-vis du processus d'apprentissage.

Les jeunes d'aujourd'hui sont-ils donc différents des jeunes d'hier ?

Sans doute vous souvenez-vous d'un spot publicitaire, vieux de quelques années, pour une marque d'appareil photo : Dans une salle d'accouchement, un bébé japonais naissait avec ... un appareil photo dans la main, photographiant ses parents ébahis ! En ce qui nous concerne, nous serions tentés de penser que les jeunes d'aujourd'hui sont nés avec une télécommande ou un clavier dans les mains.

Dès la naissance, ils sont exposés à un nombre croissant de stimuli variés. Ils sont nés à l'ère de la télévision numérique, du GSM et de l'Internet.

Ces évolutions technologiques ont toutes induit dans l'esprit de ces jeunes – que nous aimons appeler « de la génération zapping » –, une attitude commune face aux questions qui se posent à eux dans leur quotidien : « J'ai une réponse immédiate, instantanée à la question que je me pose. Je peux accéder à n'importe quelle information au moment qui m'importe et ce, sans effort ou presque ! Et si je ne l'obtiens pas assez vite, je zappe/passe à autre chose ! J'appuie sur un bouton et la télévision, l'ordinateur ou le GSM s'allume sans poser de questions. » Ils sont issus d'une génération « tout et tout de suite ».

Retenons à titre d'illustration :

- Les images en direct sur une chaîne américaine internationale des attentats du 11 septembre 2001 à New York.
- Les informations sur la prochaine séance de cinéma via le site web du cinéma le plus proche.
- Les informations sur le restaurant thaïlandais le plus proche en composant le numéro de téléphone des renseignements nationaux.
- La dernière mélodie du chanteur le plus en vogue chargée sur le GSM.

La médiation qu'apportaient hier encore les adultes aux jeunes et qui leur permettait de décoder ces stimuli et donc, d'en comprendre le sens, semble

[1] L'utilisation du mot « nouvelles » est un abus de langage. Il serait plus raisonnable de parler de technologies que de « nouvelles » technologies puisqu'on utilise l'ensemble des moyens actuellement disponibles permettant de véhiculer le langage écrit ou parlé, l'image fixe ou animée et le son.

aujourd'hui diminuer. Divers facteurs peuvent expliquer cet état de fait : les deux parents engagés dans une vie professionnelle souvent accaparante, passivité devant les nombreux programmes télévisés, éclatement des familles, ...

Les enseignants se plaignent aujourd'hui de « blocages » de la part de leurs étudiants. Ces derniers sont habitués à ingérer des informations passivement et éprouvent des difficultés à maîtriser les concepts abstraits et à articuler leur pensée en une sémantique cohérente. Ils ont besoin d'un tuteur qui les guide dans une mise en situation. La jeunesse d'aujourd'hui réclame davantage de relations entre les savoirs.

L'enseignant doit dès lors axer le processus d'apprentissage sur les interactions en développant une médiation appropriée. Puisque les étudiants maîtrisent l'art du « zapping », pourquoi ne pas utiliser les nouvelles technologies de l'information et de la communication pour les aider à démarrer dans leur cheminement logique et créatif ? Ce « zapping » au sens large peut s'avérer digne d'intérêt à partir du moment où l'étudiant se trouve dans la phase exploratoire d'une situation d'apprentissage. Par approximations successives, l'étudiant élabore alors une représentation mentale de la structure globale du contenu de la matière visée et identifie la stratégie optimale en fonction du type de tâche à réaliser.

Avec l'apparition de nouveaux médias, de nouvelles stratégies cognitives sont nées. Elles ont entraîné une évolution des méthodes d'appropriation. L'enseignement doit s'adapter à ces nouvelles stratégies.

L'outil d'apprentissage est considéré comme un tuteur offrant une formation personnalisée à l'étudiant. En effet, la matière ainsi abordée permet autant de retours en arrière que l'étudiant en éprouve la nécessité et ce, sans jamais rougir de honte lorsqu'il se trompe. Par opposition à l'approche traditionnelle et linéaire, un cours basé sur les nouvelles technologies de l'information et de la communication permet aux apprenants de maîtriser l'ensemble des objectifs sur une durée qui variera selon qu'ils sont novices ou plus expérimentés, et selon leur rapidité d'acquisition propre.

Les nouvelles technologies éducatives[2], tout en exerçant une influence considérable sur l'enseignement, ne remettent en cause, ni la vocation fondamentale de l'université, ni les finalités de la formation universitaire. L'ordinateur multimédia et l'accès aux réseaux modifient certes le système pédagogique et la relation professeur/étudiants mais éclairent le rôle irremplaçable de l'enseignant. Face à la surabondance d'informations, l'étudiant peut se retrouver noyé sous les données. Dans ce contexte, le rôle de l'enseignant est de lui donner des outils pour trier les informations et lui permettre de construire cet ensemble multidimensionnel de données dont il faut conserver les vraies informations pour construire des savoirs formant à leur tour des connaissances. C'est justement l'art du professeur que de l'aider à intégrer les différentes informations de cet ensemble.

[2] C'est à dire les nouvelles technologies de l'information et de la communication adaptées à la pédagogie.

Un triangle didactique [DEV92] subsiste qui lie :

- un savoir ou une compétence à acquérir
- un candidat à l'acquisition et
- un médiateur de chair et d'os.

Fig. 1 - Le triangle didactique selon Develay [DEV92]

Develay [DEV92] développe l'idée qu'un outil d'apprentissage ne remplacera jamais le professeur. Selon lui, l'apprentissage repose sur des interactions directes, personnalisées et régulières entre les enseignants et les étudiants. Il est illusoire de croire que les nouvelles technologies éducatives peuvent techniquement remplacer les interactions humaines comme sources de formation de l'humain. Cette vérité élémentaire est la base même de cette réalité anthropologique fondamentale qu'est la tâche éducative. De ce point de vue, les nouvelles technologies de l'information et de la communication n'ont qu'une fonction périphérique et instrumentale. Les rapports interactifs professeur/étudiants constituent le cœur de l'action pédagogique.

Ces nouvelles technologies éducatives offrent aujourd'hui à l'enseignant une chance sans précédent de répondre, avec le discernement nécessaire, à une demande de formation de plus en plus massive et de plus en plus diversifiée. L'enjeu est de taille : la capacité de l'apprenant à accéder à l'information et à la traiter est déterminante non seulement pour l'obtention de son diplôme universitaire mais également pour son intégration sur le marché de l'emploi[3] et son intégration sociale.

2.1.3. Avantages des nouvelles technologies éducatives

Il est intéressant de mettre en lumière les différentes caractéristiques de l'ordinateur et des outils connexes qui peuvent contribuer à son succès en tant qu'outil d'enseignement :

[3] Le lecteur intéressé lira l'article de presse en annexe (annexe 14.1) illustrant l'impact des nouvelles technologies sur la réussite économique et politique des gens.

1. *La disponibilité spatiale et temporelle*

L'étudiant a la possibilité de s'exercer de chez lui ou à l'université ou depuis tout autre endroit doté d'un accès à l'outil d'apprentissage, et ce, à un moment et pendant une durée qui lui conviennent.

2. *La flexibilité*

Les outils basés sur les nouvelles technologies éducatives permettent une plus grande flexibilité de la formation que les outils traditionnels. Ils permettent en effet de jouer sur différentes variables de l'environnement pédagogiques telles que le lieu, le temps, le parcours, le rythme, les situations et d'adapter ainsi l'apprentissage aux besoins, aux contraintes ou aux attentes des étudiants. On parle de libre accès aux ressources pédagogiques mises à la disposition des apprenants.

3. *L'individualisation*

Des points *1.* et *2.*, découle le fait que les rythmes de l'étudiant peuvent être respectés. Par « rythmes », nous entendons période et durée d'activité et rapidité d'acquisition.

4. *L'interactivité*

Les outils basés sur les nouvelles technologies éducatives demandent une participation active de la part des étudiants qui doivent collaborer et s'impliquer dans le processus d'apprentissage.

5. *L'intégration des médias*

Vidéo, images, animations et sons peuvent venir enrichir et illustrer le contenu écrit.

6. *La répétition*

Les outils basés sur les nouvelles technologies éducatives peuvent redonner de manière infinie la même information, autant de fois que l'étudiant le juge nécessaire. Et cela, sans que ce dernier n'en éprouve quelque gêne.

7. *La structuration des contenus d'enseignement*

Avant d'être intégrés dans l'outil, les contenus pédagogiques sont analysés, réorganisés et structurés en fonction des objectifs et des scénarios d'apprentissage. L'usage de l'informatique est structurant tant pour le pédagogue que pour l'apprenant.

2.1.4. Limites des nouvelles technologies éducatives

Malgré les avantages qu'elles offrent en pédagogie universitaire, les nouvelles technologies éducatives présentent tout de même des faiblesses importantes qu'il

est essentiel d'avoir à l'esprit si l'on veut échapper aux désillusions et aux déceptions qui pourraient entraîner un rejet sans la moindre nuance de ces technologies par les principaux intéressés, à savoir tant les enseignants que les étudiants.

Sans être forcément plus efficaces que l'enseignement traditionnel, un des intérêts principaux des nouvelles technologies éducatives réside dans le fait qu'elles peuvent être plus stimulantes vu qu'elles favorisent la curiosité. Un étudiant motivé investit plus d'efforts dans la tâche qui lui est demandée.

Soulignons que l'attrait de la nouveauté s'effaçant au fil des ans, une nouvelle approche pédagogique peut rapidement cesser d'être séduisante. Car si l'effet de nouveauté peut permettre à des outils d'apprentissage de qualité incertaine d'obtenir de bons résultats pédagogiques, baser le succès de l'intégration des nouvelles technologies éducatives à l'université sur un tel leurre est un piège à éviter.

Certains professeurs rechignent à préparer leurs cours à l'aide des nouvelles technologies de l'information et de la communication en raison du temps d'investissement et du coût que cela représente. Ceci met en évidence le besoin pour les universités de prévoir l'infrastructure et la formation nécessaires à l'intégration de ces nouvelles technologies par le personnel enseignant dans leurs formations : appareils performants, plates-formes rapidement obsolètes à remplacer, accessibilité aux réseaux à grande vitesse, ...

Notons que les nouvelles technologies de l'information et de la communication peuvent être sources d'économies financières importantes car une fois les nouveaux outils d'apprentissage mis au point, ceux-ci peuvent aisément être reproduits, diffusés, mis à jour et utilisés à un coût inférieur à celui de la mise à jour des outils traditionnels : réimpression de manuels, syllabi, ...

Soyons conscients que généraliser l'utilisation des nouvelles technologies éducatives dans le cadre des programmes existants n'est pas la garantie d'un succès. Depuis la simple illustration multimédia lors d'un exposé magistral jusqu'à l'outil d'auto formation supervisé, les formes d'intégration des nouvelles technologies de l'information et de la communication dans l'enseignement universitaire peuvent être multiples. Il serait toutefois dommage de se limiter à demander aux étudiants d'utiliser des outils d'apprentissage individualisés pour compléter l'exposé magistral du professeur. L'introduction de ces nouvelles technologies demande que les programmes fassent l'objet d'un réaménagement en profondeur afin que les étudiants prennent mieux en charge leur formation.

2.2. Présentation du mémoire et de ses objectifs

Ce mémoire vise l'élaboration d'un outil d'apprentissage de l'algorithmique accessible via le réseau Internet.

2.2.1. Le public cible

Le public cible comprend les étudiants qui ont dans leur cursus le cours d' « Introduction à l'Algorithmique ». Il s'agit typiquement des étudiants de 1ère candidature en sciences économiques et en informatique.

2.2.2. Les objectifs du client

Le client est le professeur en charge du cours d'Introduction à l'Algorithmique, ainsi que ses assistants.

Les objectifs du client sont purement pédagogiques.

L'algorithmique constitue un contenu difficile à faire passer auprès des étudiants sous forme magistrale. Perçus le plus souvent comme « allant de soi » lorsqu'ils sont présentés sous forme magistrale, les exemples introductifs du cours seront plus favorablement abordés par une mise en situation à travers un outil interactif.

L'outil doit faciliter la compréhension de l'algorithmique par le biais d'animation d'algorithmes. Il doit en outre provoquer et soutenir l'intérêt de l'étudiant.

L'outil doit montrer l'exécution d'algorithmes complets ainsi que l'état des données en mémoire pendant celle-ci et permettre de valider ces algorithmes par la méthode de construction par des invariants.

L'outil fera au maximum abstraction des aspects liés à un langage particulier pour proposer à l'élève un ensemble de primitives simples, idéalement graphiques pour permettre de réaliser des exemples amusants, avec les structures de contrôle classiques (itération, séquence, alternative, itération).

Le client désire utiliser l'outil d'apprentissage lors de ses cours magistraux, comme valeur ajoutée illustrant les concepts théoriques et abstraits de l'algorithmique. Il désire également que cet outil soit accessible par ses étudiants lors des séances de travaux dirigés ou travaux pratiques et à partir de quelque endroit que ce soit, afin d'offrir le maximum de flexibilité aux apprenants.

Le client estime qu'un outil offert dans plusieurs langues peut s'avérer être un plus, étant donné qu'il constate que de plus en plus d'étudiants n'ont pas pour langue maternelle le français.

L'introduction des nouvelles technologies en pédagogie peut poser des difficultés de plusieurs ordres aux apprenants. Le client insiste pour que la maîtrise de la technologie passe rapidement au second plan au profit de l'apprentissage.

Le client demande également que les données composant le contenu pédagogique de l'outil (algorithmes, traductions, aide, ...) soient stockées dans des fichiers éditables afin de simplifier au maximum l'accessibilité et la compréhension de l'outil aux utilisateurs du corps professoral.

2.2.3. Bénéfices escomptés

L'outil doit satisfaire le client, par l'apport qu'il constitue dans son exposé magistral traditionnel.

De plus, l'outil permet à l'enseignant d'élargir le cadre pédagogique en actualisant et dynamisant ses pratiques d'enseignement puisqu'il développe une formule flexible d'apprentissage :

- possibilité de formation asynchrone à distance
- accroissement de l'investissement individuel de l'apprenant
- possibilité de travail coopératif
- développement de l'autonomie de l'apprenant

Le client voit dans l'utilisation des nouvelles technologies une occasion de motiver les étudiants à la condition que celles-ci servent à structurer une véritable démarche d'apprentissage.

Le client peut en outre espérer créer des partenariats pédagogiques avec ses collègues en mettant en commun des modules de formation, en partageant et en réutilisant des modules didactiques.

Enfin, puisque l'outil doit permettre un meilleur apprentissage des étudiants, il doit – du moins l'espérons-nous - accroître le taux de réussite dans la matière visée.

2.2.4. Les objectifs institutionnels

Les objectifs de l'Institut d'Informatique voire de l'Université relèvent quant à eux d'une stratégie politique.

Ce nouvel outil multimédia permet une transformation des paradigmes d'enseignement par le centrage sur l'apprenant.

Il améliore l'accès à la formation pour tous, d'une part, parce qu'il est multilingue, d'autre part, parce qu'il est disponible via une page internet.

Il augmente également le rendement éducatif par le maintien – voire l'amélioration - de la qualité de la formation malgré la croissance du nombre d'étudiants.

Il favorise l'image d'une université cherchant à moderniser ses moyens d'enseignement pour répondre à la concurrence

2.2.5. Objectifs personnels

Les objectifs qui m'ont incitée à choisir ce sujet de mémoire sont les mêmes que ceux qui m'ont motivée, il y a trois ans, à m'inscrire à la licence en informatique à horaire décalé. Je désirais parfaire tant mon cursus constitué d'un graduat en informatique que mes compétences accumulées pendant ces sept dernières années passées dans le monde professionnel de l'informatique, par l'acquisition d'une démarche scientifique et méthodologique qui me faisait défaut.

Ce qui m'a également plu dans le sujet de ce mémoire est qu'il intègre plusieurs disciplines vues lors de notre cursus :

- Principes des langages : syntaxe et sémantique des langages de programmation,
- Principes des langages : paradigmes de programmation,
- Conception des systèmes d'information,
- Principes des organisations,
- Ingénierie du logiciel,
- Conception d'interface homme/machine,
- Gestion de projets.

Ces disciplines se croisent également dans les projets d'envergure professionnelle. Il m'importait donc de prouver que j'étais capable d'y faire face, même si, en général, dans le monde professionnel, chaque sous-tâche d'un projet est effectuée par un acteur ayant un profil spécialisé (un même acteur pouvant néanmoins réaliser plusieurs de ces sous-tâches.).

Ce sujet me permettait également d'approfondir ma connaissance des concepts orientés-objets et de la programmation en langage Java. Ces deux éléments pourraient s'avérer être décisifs dans la tournure de ma carrière professionnelle. De plus, l'utilisation d'outils auxquels je n'étais pas familiarisée, a rendu le travail très motivant.

Le dernier élément qui m'a séduite, mais non le moindre, est l'idée que ce mémoire allait contribuer à aider les étudiants débutants dans l'apprentissage de l'algorithmique. Cela m'a paru très motivant. Quoi de plus stimulant que de créer un outil qui est très attendu par un professeur et qui sera utilisé par des étudiants novices ?

2.3. Etat de l'art

La volonté d'utiliser le multimédia pour assister – voire remplacer – les enseignants n'est pas neuve. Il y a presque un siècle déjà, les films allaient révolutionner – du moins, le croyait-on – l'enseignement en reléguant les livres au second plan. Venaient ensuite la radio, puis la télévision.

Dès la fin des années soixante et durant les années septante, l'enseignement programmé par ordinateur faisait son apparition avec pour objectifs, l'amélioration des performances de l'apprentissage, la maîtrise des coûts exponentiels de tout le système éducatif et la volonté de limiter la crise de l'enseignement, crise provoquée par le rendement insuffisant de l'enseignement devant l'afflux d'étudiants couplé à l'accroissement exponentiel des connaissances.

Après 1980, c'est l'enseignement assisté par ordinateur qui doit révolutionner les méthodes d'enseignement.

Avec le recul, on constate qu'aucun de ces moyens n'a réellement remplacé l'enseignant mais qu'ils sont tous devenus des outils complétant les méthodes traditionnelles de l'enseignement.

En quelques années, les ordinateurs sont devenus si puissants qu'ils ont permis l'utilisation de programmes combinant texte, images et son. Depuis peu, le multimédia associé à Internet fait l'objet de recherches par de nombreuses universités ainsi que par le monde de l'industrie des logiciels dans le but de créer des outils de formation interactifs efficaces.

En matière d'apprentissage de l'algorithmique, une pléthore d'outils existe dont le développement s'étale sur les deux dernières décennies.

En 1981, une vidéo « **Sorting Out Sorting** » [EA1] fut créée par Ronald Baecker à l'Université de Toronto. Celle-ci animait un algorithme de tri. Elle eut pour conséquence de motiver les enseignant à adopter l'animation d'algorithmes dans leur cours pour faciliter le compréhension et l'apprentissage de ceux-ci par les étudiants.

Entre 1980 et 1990, deux systèmes majeurs ont vu le jour et ont eu une influence considérable sur tous ceux qui allaient suivre. Il s'agit de **BALSA-I** développé par Brown en 1984 et de **TANGO** développé en 1990.

BALSA-I [EA2] est un système interactif d'animation d'algorithme qui offre de multiples vues simultanées des structures de données d'un algorithme et peuvent afficher l'exécution simultanée de plusieurs algorithmes.

TANGO [EA3] présente le paradigme de chemin-transition pour la conception d'animation. Elle ébauche un nouveau cadre conceptuel pour le système d'animation d'algorithme, qui est adopté par beaucoup de systèmes postérieurs en tant qu'architecture.

De nouveaux systèmes basés sur BALSA-I et TANGO ont été développés.

BALSA-II, développé par Brown en 1998, est un système d'animation d'algorithmes, qui permet une initialisation graphique d'un algorithme, un contrôle de l'exécution d'un algorithme, une comparaison graphique de deux algorithmes, le calcul du coût d'un algorithme, une synchronisation d'algorithmes.

XTANGO, développé par Stasko en 1992, est une version X window de TANGO.

POLKA [EA4] et son interface **Samba** ont été conçus en 1997 dans le but de permettre l'animation concourante pour des programmes parallèles. C'est un système d'animation 2D orienté objet. Il a ensuite été étendu à un système à trois dimensions : POLKA 3D.

POLKA 3D fournit des vues 3D et des primitives 3D comme des cônes, des sphères, des cubes. Les utilisateurs n'ont pas besoin de maîtriser l'infographie 3D pour pouvoir utiliser l'interface samba de cet outil. POLKA 3D est un interprète interactif d'animation qui prend en entrée des commandes Ascii et effectue les actions correspondantes d'animation.

L'interface Samba de POLKA 3D existe également en langage java. Il s'agit de **JSamba** [EA5].

D'autres systèmes populaires d'animation d'algorithmes ont été développés.

Mocha [EA6] est un modèle distribué avec une architecture client-serveur qui découpe de manière optimale les composants logiciels d'un système typique d'animation d'algorithmes. Sa spécificité réside dans le fait que seul le code de l'interface est exporté sur la machine du client alors que l'algorithme s'exécute sur le serveur.

Zeus [EA7] est une bibliothèque d'animations d'algorithmes développée pour Modula-3 en 1991 par Brown. Elle offre des vues synchronisées multiples d'algorithmes. Ces vues sont éditables, ce qui permet aux utilisateurs de modifier la représentation de certaines données. Ce travail a été étendu notamment par Mentor et ZADA.

Mentor encapsule la bibliothèque d'animation Zeus pour Modula-3 dans une application unique disponible sous la forme de « package » pour les systèmes Linux.

ZADA [EA8] est une collection d'animations graphiques interactives d'algorithmes distribués et de protocoles de transmission. Zada étend le système d'animation **Zeus** pour faire face aux algorithmes parallèles. ZADA a été réalisé en Modula-3.

Leonardo [EA9] est un environnement intégré pour le développement et l'animation de programmes écrits en langage C. Il présente la particularité d'intégrer le développement et l'animation des programmes avec les fonctions de contrôle utilisateur de l'animation..

Le système **PAVANE** [EA10], construit par Roman et Cox en 1992 a pour vocation la construction d'animations d'algorithmes. Il comprend un compilateur dont la fonction est de traduire les programmes (initialement en langage Swarm, ensuite en langage C) et la définition des visualisations dans une forme exécutable. Il exécute ensuite les programmes compilés et les règles de visualisation en produisant des traces décrivant les animations et permet de lire les traces décrivant les animations et de construire les représentations graphiques finales.

CATAI [EA11] est un système d'animation d'algorithmes en langage C++. Il se fonde sur la technologie des objets distribués CORBA et permet à plusieurs des utilisateurs de partager la même animation grâce à un serveur Java.

La librairie **MacLib** [EA12] implante un sous-ensemble de l'interface *QuickDraw* du *Macintosh* complété par quelques routines permettant de construire une illustration animée du fonctionnement d'algorithmes

Keld Helsgaun [EA13] présente, sur la page web de son cours 'Algoritmik', une collection d'applets java illustrant de nombreux algorithmes.

Gawain [EA14] est un système d'animation d'algorithmes. Il peut être exécuté de deux façons, soit comme applet exécuté dans un navigateur web, soit comme application java (standalone mode).

ZStep'95 [EA15] est un outil de correction ('debugging') de programme écrit en Lisp, développé en 1995. Il est destiné à aider le développeur à comprendre la correspondance entre le code source statique de son programme et son exécution dynamique. Il permet de visualiser graphiquement les étapes des algorithmes contenus dans le programme Lisp.

AnimAl [EA16] est un logiciel éducatif créé en 1992. Il est destiné à un environnement Microsoft Windows. Il contient un grand nombre d'exemples sur les tableaux, les

algorithmes récursifs, les listes chaînées, les arbres. Il est multilingue. Il permet de marquer une ligne de l'algorithme, l'algorithme s'exécute alors jusqu'à ce point d'arrêt.

Conclusion critique

L'animation d'algorithmes recouvre la visualisation assistée par ordinateur des algorithmes et a pour objectif de faciliter la compréhension du fonctionnement de ceux-ci.

L'émergence d'Internet et l'évolution du langage Java ont permis de rendre les systèmes d'animation d'algorithmes indépendants de la plate-forme et donc plus accessibles. Quelques concepteurs ont également incorporé le multimédia dans leurs outils. L'utilisation des outils d'animation d'algorithmes n'est plus confinée aux auditoires et aux salles de classe mais se prolonge à l'enseignement à distance.

Nous avons conclu à une série de critiques permettant de distinguer l'outil que nous proposons des différents systèmes énoncés ci-dessus.

La première critique est que dans les systèmes cités ci-dessus, les représentations sont généralement liées à l'implémentation de l'algorithme présenté. Cette approche ne permet donc pas d'utiliser une animation pour un autre algorithme, ni d'animer une implémentation différente d'un même algorithme.

Notre seconde critique par rapport à certains de ces systèmes, porte sur le fait que les animations proposées ne décrivent pas le fonctionnement de l'algorithme, mais qu'elles visualisent uniquement les modifications qu'entraîne l'exécution de cet algorithme sur ses données. Cela implique que l'utilisateur de tels systèmes ne trouve pas la moindre balise lui permettant de comprendre l'algorithme sous-jacent.

Notre troisième critique porte sur le fait que la plupart de ces systèmes sont développés dans un langage de programmation qui ne leur permet pas d'être distribués au travers une page web. L'outil doit être téléchargé ou un CD-ROM acheté pour pouvoir se le procurer et doit être ensuite installé, encore faut-il posséder l'environnement auquel il destiné.

Ces trois critiques nous mènent à la conclusion que si les outils existants permettent une individualisation de l'apprentissage, ils ne permettent pas de mettre en place une pédagogie de découverte et de construction des connaissances ou ne permettent pas une utilisation à distance.

Nous pensons qu'en axant notre travail sur la prise en considération des différentes étapes engendrées par l'exécution de chacune des instructions des algorithmes étudiés, nous proposons une démarche intéressante pour l'enseignement de l'algorithmique et des techniques de programmation. Nous pensons également que rendre l'outil accessible via le web peut fortement encourager son utilisation auprès du public visé, à savoir les étudiants.

2.4. Un nouvel outil d'animation d'algorithmes ! Encore ? Notre contribution.

Pour obtenir un outil pédagogique et l'intégrer dans un cours, il y a habituellement deux façons de procéder.

- La première façon de procéder consiste à développer cet outil. Au début du développement d'un tel outil, il est d'usage de ne prendre en compte que les besoins les plus urgents. Il est en effet impossible de tout animer. Dans une première version, certaines données sont négligées: types abstraits, paramètres, etc.

  Ensuite, sur base des premières évaluations de l'outil, la modélisation peut être enrichie mais cela peut engendrer des répercussions non négligeables. A titre d'illustration, changer le degré d'une équation peut aboutir à une modification importante dans la formule de résolution.

  Si les évaluations s'avèrent satisfaisantes, un module d'optimisation pourra être ajouté. On peut aussi avoir besoin de passer d'une simulation statique (algorithmes prévus par le professeur) à une simulation dynamique (algorithmes créés en temps réel par les étudiants).

  Dans un autre ordre d'idées, on peut éprouver le besoin de lier des modélisations déjà existantes, ce qui informatiquement implique souvent d'inclure un ou plusieurs modules d'un programme dans un autre.

- La seconde façon de procéder consiste à acheter un produit logiciel de simulation sur le marché Mais cela n'est pas toujours la bonne solution, car ceux-ci ne sont généralement pas suffisamment ouverts à des développements spécifiques. En effet, la modélisation et les algorithmes ne sont pas toujours décrits avec la précision exigée par le professeur, et par conséquent le domaine de validité du logiciel n'est pas toujours connu avec assez de précision. De plus, le professeur est tenté d'ajouter les exemples qu'il crée et/ou adapte en fonction des concepts qu'il souhaite illustrer. Ces outils le permettent-ils ?

Le besoin d'évolutivité met en évidence la difficulté dans le choix d'une solution. Toutes deux semblent en effet inadéquates.

De plus, bien souvent, le code conçu « proprement » au départ, devient vite difficile à gérer. Un changement simple dans son principe, tel que l'introduction d'un nouveau type abstrait (exemple : une liste d'éléments de type simple) peut être du point de vue de l'implémentation très compliqué, et faire appel à une connaissance de la globalité du code. Dès lors, la tentation est forte de faire plusieurs versions spécialisées du logiciel. Mais cela ne résout pas le fond du problème : les modifications sont difficiles.

Ces nécessités de perfectionnement continuel tendent à démontrer l'utilité de concevoir un outil spécialement dédié au cours d' « Introduction à l'Algorithmique ». Visant le long terme, il nous semble opportun de procéder à une bonne analyse et à une conception solide avant de développer l'outil final.

Pour durer, il est essentiel que nous mettions en œuvre une approche évolutive. Le nombre de lignes de programme à adapter lorsqu'on souhaite faire évoluer l'outil, doit être réduit au strict minimum.

Comment est-ce possible ?

Les aspects susceptibles de varier de façon indépendante doivent être découplés. Pour cela, nous estimons que le plus important est de respecter le principe de dissimulation des informations : un objet ne doit pas connaître la structure interne d'un autre.

Nous concentrons ensuite notre effort sur l'obtention d'une première version valide.

Ensuite, un travail d'analyse des performances, allié à la connaissance des algorithmes, permettra de distinguer les parties à optimiser. On pourra alors prévoir le stockage des résultats intermédiaires, ainsi que d'autres optimisations, en perturbant le moins possible l'architecture du système.

2.5. Frein à nos ambitions – Limites de notre outil

En vue de mener à bonne fin la réalisation de notre outil et dans le souci d'éviter de trop nous éparpiller, il a fallu fixer des limites aux objectifs initiaux.

Nous rappelons que l'outil doit permettre aux étudiants de procéder à des simulations d'exécution d'algorithmes afin de les aider à appréhender les notions de base de l'algorithmique.

Nous avons choisi de traiter trois types de données simples : les entiers, les booléens, les chaînes de caractères, et un type de données complexe : les tableaux composés de ces trois types simples, à savoir, les tableaux d'entiers, les tableaux de booléens et les tableaux de chaînes de caractères.

L'outil d'apprentissage, tel que réalisé, montre l'exécution de quatre algorithmes. Ces quatre algorithmes ont été identifiés par notre client comme étant ceux introduisant des nouveaux concepts importants pour la compréhension de son cours. A ce stade-ci, il s'agit bien du professeur, concepteur du contenu de l'outil, qui crée les instructions de l'algorithme et non l'étudiant qui s'exerce à diverses manipulations pertinentes ou non dans celles-ci.

Il s'agit d'animer l'exécution d'algorithmes illustrant les concepts suivants :
- L'affectation.
- La manipulation d'un tableau.
- La notion de procédure
- L'accès en écriture et en lecture dans un fichier

Un deuxième outil permet au professeur, concepteur du contenu, d'introduire lui-même de nouveaux algorithmes similaires selon les standards que nous avons définis : langage algorithmique, types des variables, etc.

Un troisième outil permet d'ajouter une nouvelle langue d'interface à l'outil d'apprentissage afin de le rendre plus convivial et plus accessible.

Deux manuels d'utilisation de notre outil sont prévus, un pour l'étudiant et un pour le concepteur de contenu, ainsi qu'un guide de référence destiné aux développeurs susceptibles de perfectionner le projet dans le futur.

Chapitre 3

Méthodologie de développement

3.1. Nécessité d'une méthodologie de développement

Un aspect essentiel d'une méthodologie de développement d'un outil logiciel est de développer un logiciel de qualité. Le terme « qualité » signifie que l'on cherche à développer un logiciel qui correspond aux besoins d'un utilisateur de ce logiciel.

Différents critères [LES01] de qualité permettent de définir différents types de qualité. Un développement peut être fait pour satisfaire tout ou partie de l'ensemble de ces critères.

- Exactitude
  L'exactitude d'un logiciel consiste en son aptitude à fournir des résultats voulus dans les conditions normales d'utilisation. Il s'agit du plus important des critères de qualité. Il est l'essence même de l'informatique : l'informaticien souhaite développer des logiciels qui répondent aux besoins des utilisateurs.

- Robustesse
  La robustesse d'un logiciel réside dans le fait de bien réagir lorsque l'on s'écarte des conditions normales d'utilisation.

- Fiabilité
  Le logiciel ne doit pas causer de dommages physiques ou économiques en cas de défaillance.

- Extensibilité
  L'extensibilité exprime la facilité avec laquelle un programme pourra être adapté pour faire face à une évolution des besoins de l'utilisateur.

- Réutilisabilité
  La réutilisabilité vise la possibilité d'utiliser certaines parties du logiciel pour développer un autre logiciel répondant à d'autres besoins. Cette notion est souvent reliée à l'orienté objet où une classe générale sera facilement réutilisable.

- Efficience
  L'efficacité d'un logiciel consiste en son aptitude à bien utiliser les ressources matérielles (mémoire, CPU,...).

- Portabilité
  La portabilité est la facilité avec laquelle on peut exploiter un logiciel dans différentes implémentations matérielles et/ou logicielles.

- Facilité d'utilisation et maniabilité
  Le logiciel doit avoir une interface utilisateur appropriée et de la documentation

- Facilité de maintenance
  La facilité de maintenance consiste en l'aptitude du logiciel à être facilement modifié.

- Intégrité
  L'intégrité d'un logiciel est son aptitude à protéger ses différents composants contre des modifications ou des accès non autorisés.

Une méthodologie de développement permet de faciliter la satisfaction des critères de qualité. C'est donc en cela qu'elle s'avère nécessaire.

3.2. Méthodologie choisie

Dans le cheminement de la pensée qui va de la naissance d'un projet informatique jusqu'à son terme, plusieurs grandes étapes peuvent être distinguées. On parle du cycle de vie d'un logiciel. Il en existe plusieurs modèles.

Le modèle du cycle de vie d'un logiciel est une représentation idéalisée des principales activités durant le processus de développement. Il vise à établir un paradigme de développement, définit les fonctionnalités du logiciel, fournit un vocabulaire consistant et commun entre les intervenants, mais surtout il rend le processus de développement visible et donc contrôlable en fournissant des jalons de repère et aide les intervenants à mieux percevoir les relations entre les différentes étapes de ce processus.

3.2.1. Cycle de vie d'un logiciel

« Le cycle de vie d'un logiciel, c'est la période de temps qui débute au moment de la définition et du développement d'un produit logiciel et se termine lorsque le produit logiciel n'est plus disponible pour utilisation » (Définition de l'IEEE[4])

Le processus de développement d'un logiciel peut être très long : plusieurs mois, voire plusieurs années. Notre outil d'apprentissage étant typiquement un logiciel de nature évolutive, il faut réduire le temps de mise en marche, pouvoir produire des versions partielles du système et pouvoir ajouter graduellement des fonctionnalités et des qualités.

[4] Institute of Electrical and Electronics Engineers, Inc. Il s'agit d'une association sans but lucratif de professionnels désireux de promouvoir la recherche et le développement de l'électricité, l'électronique et l'informatique.

Nous penchons dès lors pour un modèle évolutif de développement au cours duquel des versions de logiciels de plus en plus complexes et détaillées seront développées.

Le modèle de cycle de vie que nous utilisons pour notre outil d'apprentissage est basé sur le développement par prototypage. Le prototypage est la clé de voûte du développement itératif.

3.2.2. Prototypage

L'objectif du prototypage est d'évaluer l'utilisabilité du système final à partir d'une version préliminaire : le prototype. Il permet ensuite d'identifier des problèmes, d'analyser concrètement leurs causes et de proposer des solutions qui sont mises en oeuvre dans le prototype suivant.

Dans un premier temps, nous situons notre travail parmi les différentes interprétations du terme "prototypage".

Les prototypes se différencient selon leur degré de réalisme.

a) Prototypes horizontaux ou verticaux

J. Nielsen [NIE93] distingue deux degrés de prototypage selon le niveau d'interaction offert par le prototype:

- Les prototypes construits suivant un axe horizontal présentent uniquement la partie visible du logiciel, c'est à dire les composants graphiques de l'application : fenêtres, boutons, menus, etc. Les commandes du logiciel ne fonctionnent pas. Il s'agit parfois d'une simple maquette sur papier. Ils permettent de réaliser un test de perception qui permet au développeur d'évaluer la compréhension de l'interface par le client et les utilisateurs. Cette première étape permet de vérifier le comportement local de l'interface. Elle sert aussi à identifier les points critiques où des problèmes d'utilisabilité sont susceptibles d'apparaître.

- Les prototypes construits suivant un axe vertical couvrent ensuite la mise en œuvre d'un ensemble cohérent de fonctionnalités de l'application afin que l'utilisateur puisse réaliser complètement un scénario typique d'utilisation du logiciel. Ils permettent de réaliser des tests d'utilisabilité qui permettent d'identifier les problèmes et d'analyser leurs causes. Sur base de ces analyses, des solutions sont ensuite élaborées et mises en œuvre dans une seconde version du prototype qui va faire l'objet d'une nouvelle série de tests, et ainsi de suite jusqu'à ce que les problèmes soient corrigés.

Généralement, les prototypes réalisés sont des deux types.

b) Prototypes jetables ou réutilisables

Les prototypes créés à des fins de vérification fonctionnelle ou technique (faisabilité) peuvent être jetables. Ces derniers sont conçus rapidement, sans soin particulier, pour être écartés dès qu'ils ont rempli leur rôle qui peut être, par exemple, d'aider à clarifier ou à formuler les besoins. L'attention est particulièrement portée sur les besoins les moins biens compris.

D'autres prototypes peuvent être réutilisables, comme lors d'un cycle de vie itératif qui étoffe, à chaque itération, le prototype initial jusqu'au produit fini. Les prototypes réutilisables sont des programmes de qualité qui sont échafaudés les uns à partir des autres et continuellement modifiés en fonction de l'avancement dans la connaissance du domaine. L'attention est ici davantage portée sur les besoins les mieux compris. Le prototype réutilisable est généralement appelé « prototype évolutif ».

3.2.3. Prototype évolutif

Notre perspective étant de partir d'une réalisation partielle complétée de façon incrémentale, notre méthodologie correspond à une approche de prototypage évolutif.

Malgré toute l'attention qu'on y porte, la première version d'un logiciel ne sera jamais satisfaisante. C'est pourquoi, le développement doit suivre un processus itératif où chaque prototype est évalué avec les utilisateurs et amélioré en fonction des problèmes rencontrés lors des tests. Le système complet se construit progressivement par itérations successives d'une forme simple vers une forme complexe, chaque cycle d'évolution fournissant un prototype de plus en plus élaboré et de plus en plus proche du système à produire. Chaque itération est un prototypage.

Grâce au prototypage évolutif, le produit prend la place centrale qu'il doit avoir dans le cadre d'un projet informatique. Cette approche est orientée sur le produit, plutôt que sur le processus de fabrication qui lui donnera naissance. Grâce au prototype, l'utilisateur peut recevoir un rôle de choix, puisqu'il peut être plus facilement intégré dans le processus de modélisation.

Le grand avantage de ce modèle est de toujours placer le produit au centre du projet. L'évaluation est essentielle. Elle ne se limite pas à autoriser la poursuite des travaux mais vise également l'amélioration du produit.

Très tôt dans ce type de mise en œuvre, des prototypes sont soumis aux commanditaires et aux utilisateurs. Ces prototypes sont l'objet central d'une communication qui peut s'instaurer entre les concepteurs et les utilisateurs.

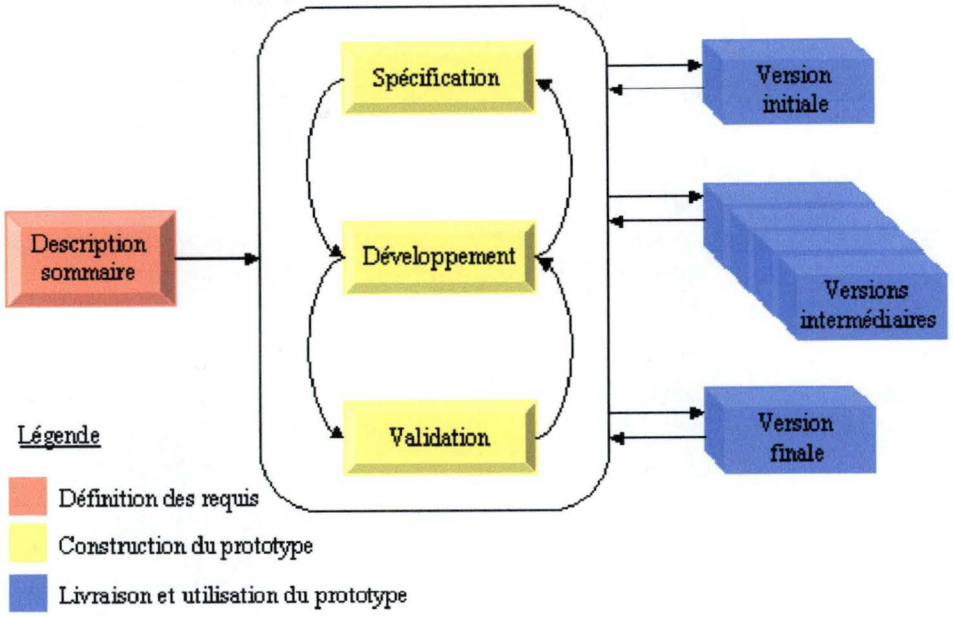


Fig. 2 - modèle évolutif de cycle de vie [HAB01]

Processus d 'ingénierie - définitions

Spécification : Définir les besoins et contraintes du système
Développement : Construire le système
Validation : Vérifier l'adéquation entre les spécifications et le développement

3.2.4. Validation d'un prototype

Le prototype est validé après trois vérifications concluantes :

- Vérification du domaine d'information
  - Les bonnes données sont-elles traitées ?
  - Les bons résultats sont-ils produits ?
- Vérification des fonctionnalités
- Vérification du comportement
  - IHM5
  - Messages
  - Erreurs

3.2.5. Les forces du prototypage évolutif

Le développement d'un logiciel par prototypage évolutif présente un grand nombre de points positifs :

- L'utilisateur se fait rapidement une idée de ce que sera l'outil ;

- Cette approche met à la disposition du développeur, du client et des utilisateurs un système d'apprentissage et d'aide à l'élaboration du logiciel à produire : le développeur apprend à mieux connaître les besoins de

[5] Interface Homme-Machine

l'utilisateur qui lui-même apprend ce qu'un système informatique peut lui apporter et cerne progressivement les effets de l'introduction de ce système dans son environnement de travail ;

- Le développement d'un prototype permet une forte réactivité aux besoins et donc une meilleure communication entre les utilisateurs et les développeurs. En découle une meilleure compréhension du problème entre le(s) client(s), le(s) développeur(s) et les utilisateurs ;

- Il favorise la détection rapide de problèmes imprévus et permet ainsi une correction immédiate du système en développement. D'après [LES01], plus une erreur est détectée tard dans le cycle de vie du logiciel, plus sa correction est coûteuse ;

- Le prototypage évolutif permet de consolider le design de l'interface hommes-machine ;

- Il permet d'optimiser la démarche de développement en travaillant sur des prototypes au niveau de la conception puisque chaque développement est moins complexe et les intégrations progressives ;

- Il permet des livraisons et mises en service après chaque incrément ;

- Le prototype évolue et incorpore progressivement les fonctionnalités voulues. Une fois celui-ci achevé, plus des trois quarts du développement est déjà réalisé.

3.2.6. Les faiblesses du prototypage évolutif

Le développement d'un logiciel par prototypage évolutif présente également un certain nombre de risques :

- Le risque de remise en cause du noyau (fonctionnalités de base) en cours du développement ;

- La tentation d'abréger le processus et de se contenter d'un prototype incomplet ;

- Une (trop) grande facilité de changement entraînant un risque d'instabilité des spécifications. Il faut dès lors maîtriser[6] le nombre de rétroactions dans le cycle de prototypage, c'est-à-dire le retour sur la réalisation du prototype après démonstration et décision. Tracer les évolutions est non seulement un moyen d'éviter de revenir sur des problèmes déjà traités, mais aussi une manière de mettre en évidence des points délicats en terme d'utilisabilité sur lesquels il conviendra d'être vigilant dans la suite du développement ;

- Un prototype peut être identifié au produit final ;

[6] Contractuellement, par exemple.

- Par rapport à une validation classique, la validation par prototypage induit une fragilité des décisions. Le travail de validation est compliqué par la superposition de plusieurs plans : celui des fonctionnalités, celui de l'ergonomie et celui des performances ;

- L'approche par prototypage évolutif pose également des problèmes en terme de calendrier et de budget souvent irréalistes ;

3.3. Adéquation de la méthodologie choisie avec notre projet

Dire que l'on opte pour un cycle de vie par prototype est un raccourci qui, hors des petits projets, n'est pas assez formalisé pour soutenir l'organisation nécessaire aux projets plus importants. On préfèrera des cycles de vie en spirale avec évaluation des risques, RAD[7] ou DSDM[8] qui tout en intégrant des phases de prototypage, proposent par ailleurs un formalisme adapté aux grands projets.

Sur les petits projets menés par des petites équipes, l'intérêt est de gagner en flexibilité, et de s'abstraire des contraintes d'une méthodologie plus large. Il s'agit de la démarche que nous adoptons pour notre outil d'apprentissage car l'approche par prototypage évolutif nous offre des avantages qui cadrent parfaitement les buts que nous nous sommes fixés, notamment, en termes d'adaptabilité, de modularité et d'adéquation avec le monde réel.

3.4. Récapitulatif de notre démarche

Une première difficulté est survenue lorsque nous avons tenté d'obtenir un énoncé complet et cohérent des besoins de notre client. La description de ces objectifs était assez sommaire. Pour pallier cette difficulté, nous avons entamé l'élaboration d'un premier prototype suivant l'axe horizontal dans le but de simuler le comportement du logiciel tel que perçu par l'utilisateur et de stabiliser ainsi l'interface hommes-machine. Cela permettait surtout d'avoir une réaction rapide de l'utilisateur pour valider les spécifications par rapport aux besoins réels et de procéder aux ajustements nécessaires.

Quand ce prototype[9] a été accepté, après trois itérations, il a été possible de figer le résultat de l'analyse, c'est-à-dire de rédiger une spécification du logiciel

Le prototypage horizontal ayant validé la faisabilité du projet, un développement plus concret a pu être réalisé et des prototypes verticaux ont été proposés. Nous avons pu définir clairement les fonctionnalités de notre outil, les implémenter et vérifier leur comportement. Nous avons procédé étape par étape, fonctionnalité par fonctionnalité avant d'obtenir un tout cohérent.

L'approche par prototype horizontal nous a permis de prouver la faisabilité des besoins de notre client et l'approche par prototype vertical nous a permis de définir et de vérifier les fonctionnalités du système (cf. Figure 3).

---

[7] Rapid Applications Development
[8] Dynamic Systems Development Method
[9] Voir annexe 1.

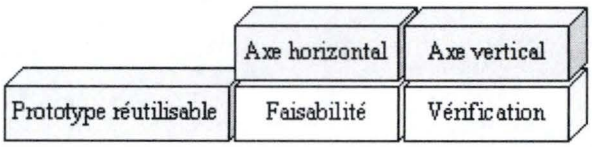


Fig. 3 - méthodologie choisie

Ces différentes phases sont échelonnées dans le temps. Une phase de développement se termine lorsque la validation de cette phase est faite. Une phase ne peut commencer que lorsque la précédente est terminée. A la fin de chaque phase, le client et le développeur sont d'accord. La décomposition en phases de développement permet donc le suivi du projet par le client.

Deuxième Partie

Le produit

Introduction

Cette deuxième partie présente le résultat de la démarche par prototypage évolutif que nous avons adoptée. Nous invitons dès lors le lecteur à considérer qu'il s'agit de la spécification du travail dans sa dernière mouture.

Chapitre 4

Analyse des besoins

Notre analyse des besoins recouvre un simple recensement de demandes auprès de l'enseignant et des assistants chargés du cours d'Introduction à l'Algorithmique. Leurs demandes ont été spontanées ou influencées par une politique d'établissement ouverte au multimédia.

Nous aurions pu mener une enquête circonstanciée auprès de l'ensemble des acteurs concernés, soit la direction, les directeurs de programmes d'enseignement, les enseignants et les étudiants. Mettre les étudiants au centre du dispositif de formation en les interrogeant sur leurs besoins, leurs attentes et leurs difficultés est une manière de faire correspondre l'outil aux besoins des étudiants et pas seulement aux besoins des enseignants ou de l'université. Notre client, expert en pédagogie connaissant parfaitement le profil de ses étudiants, a cependant estimé qu'il valait mieux commencer par développer un prototype d'outil d'apprentissage permettant de traiter quelques cas précis et d'ensuite soumettre ce prototype à ses étudiants afin d'en évaluer sa pertinence par rapport aux objectifs et aux bénéfices escomptés. Si cette évaluation conclut à la pertinence d'introduire cet outil dans le cours concerné et à la valeur ajoutée du multimédia pour la pédagogie, le développement de l'outil d'apprentissage pourra être complété en tenant compte des nuances relevées lors de l'évaluation. La clé du succès résidera – nous l'espérons -- en ce que l'outil est conçu pour les étudiants et non pour l'enseignant qui se fait plaisir.

4.1. Cadrage du champ d'investigation

L'outil pédagogique multimédia interactif que nous souhaitons développer dans le cadre de ce mémoire répond à un besoin détecté par un enseignant chargé du cours d'Introduction à l'Algorithmique, donné aux étudiants de première candidature en sciences économiques et de première candidature en informatique :

- Une difficulté à transmettre certaines connaissances avec les moyens traditionnels : public en difficulté, profils cognitifs différents des élèves.
- Une difficulté à transmettre un certain type de connaissance : nécessité de voir, de faire l'expérience.
- La nécessité de moderniser les outils, les moyens et les méthodes pédagogiques devenus obsolètes ou inadaptés au public, à la période actuelle.

Il constitue un moyen pour cet enseignant de diversifier ses méthodes d'enseignement dans le but principal d'accroître l'intérêt de ses étudiants pour son cours en leur permettant de mieux appréhender les concepts abstraits de la matière.

Sur un plan institutionnel, le développement de ce produit multimédia correspond à une stratégie de l'université cherchant à moderniser ses moyens d'enseignement pour répondre à la concurrence. Mais il correspond également à une prise de conscience du bien fondé et de la valeur ajoutée du multimédia dans l'enseignement.

Le public concerné présente une certaine hétérogénéité puisqu'il s'agit d'étudiants en première candidature provenant de formations secondaires diverses ou d'étudiants, universitaires ou non, qui se réorientent.

4.2. Constat

Vu les programmes du secondaire desquels l'algorithmique est généralement absente, nous considérons que les étudiants, au profil de départ pourtant principalement hétérogène, présentent une certaine homogénéité dès lors que l'on discerne des caractéristiques communes :

- âge,
- niveau d'études,
- conditions de vie,
- langue
- et a priori, aucune connaissance en algorithmique.

Comme nous l'avons déjà constaté (cf. 2.1.2.), ces étudiants éprouvent de plus en plus de difficultés à comprendre, à conceptualiser, à structurer et à intégrer une matière, ce qui entraîne une perte de motivation voire d'intérêt vis-à-vis du processus d'apprentissage.

4.3. Diagnostic

L'introduction des nouvelles technologies de l'information et de la communication nous semble utile dans le cadre du cours d'Introduction à l'Algorithmique en ce qu'elles permettent d'élaborer des activités d'apprentissage interactives par la simulation d'animation d'algorithmes, mais également par la mise à disposition de l'outil d'apprentissage sur le réseau Internet.

4.3.1. Pourquoi proposer un outil d'apprentissage sur le réseau Internet ?

- pour offrir des activités d'apprentissage interactives
- pour donner un encadrement pédagogique
- pour encourager des activités exploratoires par simulations d'algorithmes
- pour visualiser des concepts abstraits
- pour permettre l'apprentissage à distance
- pour créer un environnement d'apprentissage coopératif
- pour donner un complément aux notes de cours du professeur et à son cours magistral, pour continuer les discussions hors des murs des auditoires

4.3.2. Intervenants

Quatre acteurs interviennent principalement dans l'intégration des nouvelles technologies de l'information et de la communication dans les processus d'apprentissage à l'université :

- La direction	:	le directeur de l'Institut d'Informatique, voire le recteur des Facultés Universitaires
- Les experts en pédagogie	:	le Département Education et Technologie
- Le personnel enseignant	:	l'enseignant titulaire du cours d'Introduction à l'Algorithmique et ses assistants
- L'apprenant	:	l'étudiant qui assiste audit cours

Le rôle de la direction est d'ouvrir la voie à l'innovation pédagogique par l'utilisation des nouvelles technologies de l'information et de la communication et de la coordonner.

Le rôle des experts en pédagogie est de former les enseignants aux nouvelles technologies de l'information et de la communication et de les motiver pour qu'ils les intègrent à bon escient dans leur curriculum. Toutefois, les nouvelles technologies, si elles se limitent à faciliter l'accès aux informations, pourront certes jouer un rôle utile, mais ne suffiront pas à assurer un apprentissage de qualité. Progrès technologique et progrès pédagogique ne vont pas obligatoirement de pair. C'est pourquoi, les experts en pédagogie doivent aider, par leurs recherches, les enseignants à combler l'écart qui existe entre les méthodes traditionnelles de design pédagogique et les possibilités qu'apportent les nouvelles technologies sur le plan pédagogique pour apprendre, aider à apprendre ou concevoir de nouveaux moyens d'apprendre.

Le rôle du personnel enseignant est d'innover en transformant son enseignement par l'introduction de nouveaux paradigmes. La technologie ne doit cependant en aucun cas se substituer au professeur. Il appartient à ce dernier de guider l'étudiant dans son apprentissage en l'informant des possibilités mais aussi des limites de cette technologie.

Le rôle de l'étudiant est d'acquérir un ensemble de connaissances par un travail intellectuel et par l'expérience. Il doit ensuite être à même de démontrer ses compétences. Apprendre reste un acte personnel qui découle de l'engagement de l'apprenant. On ne peut apprendre à la place de l'étudiant. On ne peut que le motiver et le guider en lui montrant comment apprendre.

4.3.3. Enjeu anthropologique

Le véritable enjeu de l'intégration d'un outil d'apprentissage dans un cours traditionnel n'est pas technique mais bien anthropologique. Dans le cadre de cours d'apprentissage avec l'appui des nouvelles technologies de l'information et de la communication :

- les enseignants transforment leurs pratiques pédagogiques, et
- les étudiants, leurs pratiques d'apprentissage,

en conjuguant leur savoir-faire à la puissance de l'outil afin d'aboutir à une acquisition de compétences.

Pour innover, il faut désirer induire un changement. Pour mener à bien l'innovation que représente l'utilisation d'un outil d'apprentissage dans un cours, il nous faut :

- tenir compte des pratiques d'enseignement existantes, les analyser pour les transformer,
- faire accepter la transformation de ces pratiques,
- reconnaître et accepter que le tâtonnement inhérent à toute innovation entraîne des résistances et des craintes,
- aborder l'outil innovateur non pas comme un but en soi mais comme un moyen d'améliorer le processus d'apprentissage.

L'enseignant doit maintenant accorder à la toque du maître qui sait et qui transmet ses connaissances, la casaque du médiateur qui devra :

- sélectionner les objets de connaissance,
- les soumettre à l'action des étudiants par le biais de l'outil d'apprentissage,
- superviser le processus d'apprentissage et le réguler,
- aider les étudiants à poser de nouvelles hypothèses leur permettant ainsi de construire un sens à leur action,
- les inciter à des vérifications,
- leur susciter des contradictions,
- les déstabiliser dans leurs certitudes ou leurs préjugés,
- proposer des voies de solutions que les étudiants mettront eux-mêmes en œuvre.

4.3.4. Identification des apports pédagogiques les plus profitables pour les étudiants

Notre client a identifié dans son cours les contenus qui seraient davantage compris et assimilés par une illustration sur Internet. Il pense ainsi rendre ses étudiants plus actifs dans leur démarche d'apprentissage. Il est également conscient que les compétences que les étudiants peuvent acquérir en utilisant les nouvelles technologies de l'information et de la communication sont importantes pour leur avenir professionnel.

4.3.5. Ces apports sont-ils supérieurs à ceux des autres moyens didactiques ?

Notre client désire que l'outil d'apprentissage ne soit pas une réplique intégrale du document papier. En effet, la version papier reste efficace, compte tenu de son accessibilité et de la possibilité de l'annoter.

La question fondamentale est dès lors celle de la valeur ajoutée.

Il s'agit, en effet, de permettre à l'étudiant de visualiser les différentes actions engendrées par l'exécution d'algorithmes. Ces animations constituent un atout majeur par rapport aux possibilités restreintes d'un document statique.

4.3.6. Bénéfices escomptés pour les étudiants :

L'étudiant concerné par cet apprentissage s'approprie l'utilisation de l'outil, initialement lors de travaux dirigés dans le cadre de sa formation, ensuite via le réseau Internet à partir de quelque lieu que ce soit et à quelque moment qu'il trouve opportun dans son parcours d'apprenant.

L'étudiant est à même de :

- comprendre chaque étape de l'exécution d'un algorithme,
- visualiser les objets et l'état de la mémoire à chaque étape de cette exécution,
- s'approprier l'utilisation de l'outil,
- utiliser l'outil sans contrainte de temps ni d'espace,
- apprendre à son rythme (« Learning by doing »),
- s'approprier la démarche stratégique en résolution de problème.

4.3.7. Considérations techniques

Notre client estime que les étudiants ne doivent pas posséder des compétences informatiques autres que celles qui leur permettent d'installer un navigateur Internet récent sur leur ordinateur et d'accéder à une page web via Internet.

L'outil d'apprentissage doit être développé pour l'équipement le plus utilisé actuellement car les étudiants ne possèdent pas tous, l'équipement dernier cri et n'ont pas tous accès à la technologie la plus récente.

L'outil doit être supporté par différents systèmes. Actuellement, le pool informatique de l'université est équipé d'un système Solaris alors que la plupart des étudiants travaillent à domicile sur des systèmes Windows, voire des systèmes Linux.

4.4. Le projet

Il s'agit d'un projet pédagogique c'est-à-dire destiné à faciliter l'apprentissage.

Le projet concerne la mise au point d'un outil d'apprentissage de l'algorithmique devant être diffusé grâce aux nouvelles technologies de l'information et de la communication.

L'outil doit mettre en œuvre l'exécution d'algorithmes avec une visualisation du code de l'algorithme en langage algorithmique, de l'organigramme y associé, de l'état des données en mémoire, de l'état de la console pour illustrer les mécanismes de lecture/écriture.

L'outil doit pouvoir stopper l'exécution de l'algorithme à chaque instruction afin de bien en visualiser les conséquences. L'outil doit également permettre de revenir en arrière, instruction par instruction, dans l'exécution des algorithmes.

L'outil doit permettre la validation des algorithmes par la méthode des invariants.

L'outil doit permettre l'introduction de nouveaux algorithmes similaires.

4.5. Plan d'action

4.5.1. Mobilisation des acteurs de la formation

Le concepteur est responsable de la production des tâches et des activités offertes dans l'outil. Le concepteur est typiquement l'enseignant ou un des ses assistants experts dans son domaine qui connaît la matière enseignée et les objectifs pédagogiques. Le concepteur connaît la pédagogie et les processus d'apprentissage de la matière enseignée.

Ses fonctions consistent donc à :

- fournir ou à produire les contenus,
- participer à la conception générale du produit pédagogique multimédia avec l'équipe de développement,
- déterminer le niveau d'interactivité s'accordant avec les objectifs pédagogiques.

Le développeur du produit pédagogique multimédia doit convertir les contenus pour les faire passer du support papier au support numérisé, support permettant d'exploiter aux mieux les divers médias type image, animation, etc. Ce support est ici une page web sur Internet.

Le développeur est en principe l'interlocuteur du concepteur.

Le scénario de production adopté dans le cadre de ce projet est le plus courant dans les petits et moyens projets dans les milieux universitaires : le concepteur travaille directement avec le développeur. Ce dernier aide le concepteur pour tous les aspects techniques, de la création à l'inclusion de nouveaux contenus. Il appartient toutefois au concepteur d'approuver chaque étape.

Le tuteur assiste les étudiants dans l'apprentissage de l'outil d'apprentissage. Le tuteur est typiquement l'enseignant ou un des ses assistants. Il montre la portée et les limites de l'outil. Il explique aux étudiants les objectifs poursuivis et leur apporte assistance lorsque cela s'avère nécessaire.

La figure 4 met en lumière les interactions entre chacun des acteurs du système.

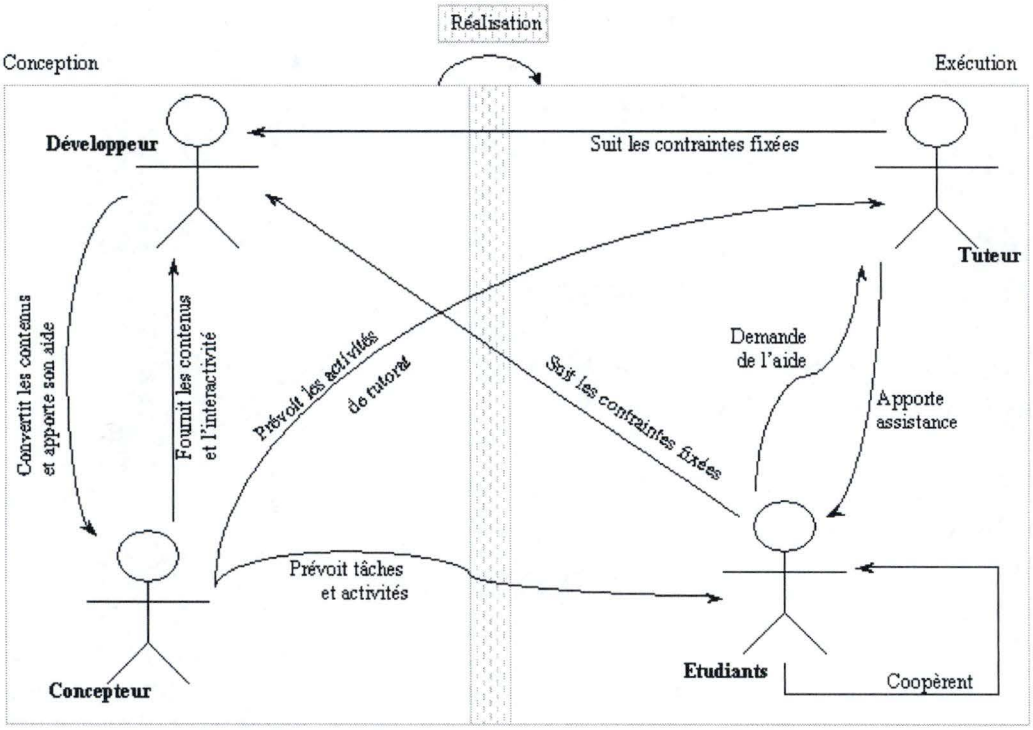


Fig. 4 - Coopération entre les acteurs du système

4.5.2. L'existant

Nous avons choisi d'utiliser le langage LSD^{03}, le Langage Simple et Didactique créé en 2003 par une équipe de professeurs et d'assistants de l'Institut d'Informatique des Facultés Universitaires Notre-Dame de la Paix de Namur [SCH03]. Le LSD^{03} ressemble à un langage pascal simplifié et s'inspire du langage algorithmique utilisé généralement dans les cours d'introduction à la programmation et à l'algorithmique, et d'autres cours connexes.

4.5.3. Méthode

Nous utilisons l'approche par prototypage évolutif pour les raisons que nous avons expliquées au point 3.4. de ce mémoire.

C'est ainsi que nous stabilisons d'abord le développement de l'interface utilisateur avec notre client, avant de nous attaquer aux fonctionnalités proprement dites.

4.5.4. Fonctionnalités du système

Nous devons spécifier un langage et définir une logique. Le langage est le langage LSD^{03} déjà partiellement défini. La logique est la logique des prédicats du premier ordre, communément utilisée pour écrire les invariants des algorithmes.

Nous devons ensuite définir deux traducteurs afin que ce langage et cette logique puissent être compris par le langage de développement de l'outil.

Nous devons ensuite définir les fonctionnalités mises en œuvre par l'exécution des algorithmes, à savoir :

- l'affichage des données relatives à l'algorithme choisi,
- l'affichage des algorithmes en LSD[03],
- l'affichage de l'invariant,
- l'affichage de l'organigramme,
- l'affichage de la console,
- l'affichage de l'état de la mémoire,
- l'internationalisation,
- l'exécution de l'algorithme,
- la trace des actions entreprises,
- l'accès à l'aide.

Ces points sont abordés au chapitre 5.

4.5.5. Estimation de la charge de travail

Selon la formule de Peter Fenrich [FEN97] permettant d'estimer les charges de travail et sachant que le développement de produits pédagogiques interactifs demande en général entre 100 et 300 heures par heure d'interactivité produite pour les étudiants, on obtient les estimations suivantes :

Charge de travail pour le concepteur du contenu pédagogique de l'outil :

25 % d'expertise du contenu + 20 % de conception pédagogique = 45 % de charge de travail pour le concepteur.

Donc pour produire une heure d'enseignement interactif, le concepteur, c'est-à-dire celui qui produit le contenu et la conception pédagogique de l'outil, prendra entre 45 et 135 heures.

Charge de travail pour le développeur de l'outil :

35 % de programmation + 10 % d'animation graphique + 10% autres = 55 % de charge de travail pour le développeur.

Donc pour produire une heure d'enseignement interactif, le développeur de l'outil passera entre 55 et 165 heures.

Il s'agit d'une estimation toute théorique citée à titre indicatif, étant donné que les deux rôles de concepteur et développeur s'entremêlent régulièrement dans le présent travail et que le contenu pédagogique de l'outil était connu par notre client dès le départ.

Chapitre 5

Spécification des fonctionnalités

Nous allons dans un premier temps procéder à l'identification et à la décomposition des fonctionnalités en sous-fonctions. Nous regrouperons ensuite ces fonctionnalités en modules. Ces modules seront ensuite intégrés dans des outils spécifiques. Nous examinerons ensuite les interactions entre les différents outils.

5.1. Identification des fonctionnalités

1. Procéder au choix d'une langue de l'outil
2. Procéder au choix d'un algorithme
3. Exécuter complètement l'algorithme
4. Exécuter la prochaine instruction de l'algorithme (« avancer »)
5. Défaire la dernière instruction exécutée de l'algorithme (« reculer »)
6. Visualiser l'historique des opérations réalisées
7. Imprimer l'historique des opérations réalisées
8. Visualiser les rubriques d'aide
9. Imprimer les rubriques d'aide
10. Visualiser les informations concernant l'outil

5.2. Décomposition des fonctionnalités en sous-fonctions

1. Procéder au choix d'une langue

   L'outil d'apprentissage de l'algorithmique est livré dans les langues nationales : français, néerlandais, allemand ainsi qu'en anglais.

   Le fait de choisir une langue a des répercussions tant sur les éléments graphiques de l'interface utilisateur de l'outil, que sur les messages de dialogues et sur les rubriques d'aide.

   D'autres langues pourront être ajoutées dans le futur, c'est pourquoi, une interface permettant d'ajouter des traductions est prévue.

2. Procéder au choix d'un algorithme

L'outil d'apprentissage doit permettre à l'utilisateur de choisir un algorithme parmi ceux préalablement introduits par le concepteur de contenu.

Pour introduire ces algorithmes dans l'outil d'apprentissage, le concepteur doit utiliser un langage algorithmique et le soumettre à un traducteur qui sera chargé de traduire l'algorithme dans le langage de développement de l'outil d'apprentissage.

Le choix de l'algorithme est une opération qui est rapportée dans l'historique.

3. Exécuter complètement l'algorithme
4. Exécuter la prochaine instruction de l'algorithme (« avancer »)
5. Défaire la dernière instruction exécutée de l'algorithme (« reculer »)

Ces trois fonctionnalités peuvent être décomposées en mêmes sous-fonctions, c'est pourquoi nous les avons regroupées.

L'exécution d'un algorithme ou d'une instruction de l'algorithme génère un ensemble d'illustrations graphiques :

- L'organigramme associé
- L'invariant associé
- La console qui illustre les interactions entrées/sorties de l'algorithme
- L'état de la mémoire qui montre les variables de l'algorithme avec les différentes valeurs prises au cours de son exécution

L'utilisateur a la possibilité d'exécuter l'algorithme complètement ou instruction par instruction. Il peut même revenir en arrière, instruction par instruction.

Les différentes étapes de l'exécution de l'algorithme sont rapportées dans l'historique des opérations.

6. Visualiser l'historique des opérations réalisées
7. Imprimer l'historique des opérations réalisées

Les différentes opérations liées à l'exécution d'un algorithme sont consignées dans un historique pour permettre aux utilisateurs apprenants de visualiser les différentes actions qu'ils ont réalisées. Cet historique peut être imprimé

8. Visualiser les rubriques d'aide
9. Imprimer les rubriques d'aide

L'utilisateur peut consulter et imprimer les rubriques d'aide fournies par l'outil d'apprentissage. Une aide contextuelle est également prévue pour faciliter la navigation dans l'outil.

10. Visualiser les informations concernant l'outil

La « fiche d'identité » de l'outil est accessible aux utilisateurs.

5.3. Conception de modules

5.3.1. Module « Création Langue »

Dans un pays qui compte trois langues officielles, il nous paraît naturel d'offrir l'outil d'apprentissage en ces différentes langues. De plus, il nous semble qu'un outil mis à disposition sur le réseau Internet se doit de proposer également une interface en anglais, langue dominante du réseau.

Nous considérons la traduction comme une phase importante puisque l'outil doit être développé de telle façon qu'il soit facilement traduisible.

Cela implique que les textes à afficher à l'écran soient séparés du programme. Afin de maintenir la cohérence des différentes traductions, un utilitaire d'édition doit être développé pour permettre la traduction de l'outil vers de nouvelles langues.

Cet outil maintient une base de données pour les modifications des différents fichiers de messages. Il permet à la personne en charge de la traduction de localiser rapidement les endroits où une mise à jour est nécessaire en montrant, dans une fenêtre le message d'origine dans la langue initiale, dans une autre fenêtre, le traducteur pourra introduire la traduction.

Les traductions ne seront généralement pas faites par le concepteur, mais plutôt par des personnes maîtrisant la langue d'origine et une des langues cibles.

A chaque développement d'un nouveau prototype, le principal problème est de s'assurer que toutes les versions sont à jour au niveau linguistique.

Le module « Création Langue» stocke chaque langue séparément afin que l'utilisateur de l'outil d'apprentissage puisse aisément et instantanément passer de l'une à l'autre et que l'ajout de nouvelles traductions puissent s'organiser efficacement.

L'outil de création de traduction permet d'ajouter et de supprimer la traduction d'une nouvelle langue. Il ne permet en aucun cas de supprimer une des quatre langues de base prévues.

5.3.2. Module « Affichage Langues »

L'outil d'apprentissage permet à l'utilisateur de choisir l'une ou l'autre langue : tous les éléments de l'interface graphique, ainsi que les messages de dialogue et d'aide s'affichent dans la langue choisie.

5.3.3. Module « Création Algorithme »

Un algorithme sert à transmettre un savoir-faire. Il décrit les étapes à suivre pour réaliser une tâche.

Le professeur Charles Duchâteau [DUC95] définit l'algorithmique en ces termes : *« L'algorithmique (et la programmation qui en est une incarnation), c'est l'art et la méthode de « déplier » complètement une tâche pour l'expliquer et la faire faire par 'un autre' ».*

Un algorithme est une forme bien écrite d'une résolution de problème. Il constitue une suite d'étapes plus ou moins élémentaires où sont précisés des traitements. Ces étapes s'inscrivent séquentiellement. Chacune est définie de façon précise et rigoureuse, en cohérence avec l'ensemble : ce sont les instructions de l'algorithme.

Contrairement à un programme, un algorithme ne prend pas du tout en compte les contraintes architecturales d'un ordinateur tels les registres, emplacements mémoires, etc. Il doit être facilement traduisible en n'importe quel langage informatique. Cela dit, c'est un langage qui respecte une syntaxe, aussi souple soit-elle.

Pour communiquer, les individus ont besoin d'un langage commun. Les informaticiens ont également besoin d'un langage plus ou moins codifié pour se comprendre. Le traitement d'algorithmes implique la nécessité de définir un langage générique qui permet le traitement de problèmes par la concaténation d'instructions élémentaires. Ce langage est appelé « langage algorithmique ».

5.3.3.1. Le langage algorithmique

Le langage algorithmique est un langage didactique à la base de tous les langages de programmation impératifs et est utilisé pour sa simplicité. Il doit permettre aux étudiants d'appréhender correctement les différents concepts abordés dans l'apprentissage de l'algorithmique.

Ce langage doit être :

- *spécialisé* pour écrire des algorithmes qui ne sont ni des poèmes, ni des recettes de cuisine
- *de haut niveau,* déchargé de détails techniques, ce n'est pas un langage de programmation
- *typé*

Le langage algorithmique que nous utilisons ressemble au langage pascal fortement simplifié. Un tel langage algorithmique a été défini par une équipe de l'Institut Informatique des Facultés Universitaires Notre-Dame de la Paix de Namur. Il s'agit du LSD[03].

Nous l'utilisons dans notre projet en ne reprenant toutefois que les éléments qui nous intéressent dans les limites que nous nous sommes fixées au point 2.5 du présent travail :

- Structure de base
- Notion de variables
- Opérateurs
- Instructions
- Choix conditionnel
- Boucles
- Procédures et fonctions

Nous verrons cependant qu'afin de satisfaire nos besoins, nous devons étendre le LSD[03] dans certains cas. Nous proposons par conséquent une variante[10] au LSD[03]. Les éléments ajoutés ou modifiés feront systématiquement l'objet d'une remarque précédée du symbole **.

Nous présentons ci-dessous, de manière intuitive, les différents éléments qui composent le LSD[03]. Pour une définition formelle du langage LSD[03] original, le lecteur se reportera utilement au document [SCH03] en annexe D.

Structure de base

```
Program {nom du programme} ;
{déclaration des variables}
...
Begin
    ...
    {liste des instructions}
    ...
End.
```

Notion de variables

- Définitions:

Une **variable** est un espace mémoire nommé, de taille fixée, prenant au cours du déroulement de l'algorithme, un nombre indéfini de valeurs différentes. Ce changement de valeur se fait par l'opération d'**affectation**. La variable diffère de la notion de **constante** qui, comme son nom l'indique, ne prend qu'une unique valeur au cours de l'exécution de l'algorithme.

- Attributs d'une variable :

Un identificateur : nom qui sert à repérer la variable
Un type : format des données qu'elle contient
Une portée : variable globale, locale ou entrée/sortie
Une valeur

- Différents types de variables :

Type entier	:	**Integer**	0,1,2,3, tab[1] ...
Type chaîne de caractères	:	**Char**	aa,ab,...,aaa,...,a3x,...
Type booléen	:	**Boolean**	true,false
Type tableau	:	**Array**	Tab

** En complément au LSD03, nous introduisons le type Char et nous étendons le type Array. Les tableaux peuvent, en effet, être des tableaux d'entiers, de chaînes de caractères et de booléens.

[10] Nous attirons l'attention du lecteur sur le fait que dorénavant, ce que nous appellerons LSD[03] est en fait une variante du LSD[03], et ce, par souci d'éviter de surcharger le texte. Nous espérons ne pas choquer les auteurs du LSD[03].

- Déclaration des variables :

La déclaration des variables permet de spécifier quelles seront les variables utilisées au cours de l'algorithme ainsi que le type de valeur qu'elles doivent respectivement prendre. Elle permet en outre d'assurer la cohérence du type de ces variables.

Exemples distincts de déclaration de variables :

```
var i : Integer ;
var lettre : Char ;
var isValid, trouve : Boolean ;
var tab : array[1,10] of Integer ;
```

** En complément au LSD03, nous introduisons la possibilité de déclarer deux variables de même type en une seule instruction.

- Valeur d'une variable

Dans la partie déclaration, les variables n'ont pas de valeur. Attribuer une valeur à une variable est une instruction. Initialiser une variable, c'est lui donner une première valeur.

Les expressions

- Les opérateurs et les expressions arithmétiques :

Opérateurs arithmétiques binaires

Sur les entiers : **+, -, *, /, mod**
(a mod b : reste de la division euclidienne de a par b)

Opérateur arithmétique unaire

Sur les entiers : -

Expression arithmétique

Une expression arithmétique est soit :
- Une constante numérique
- Une variable numérique
- Une combinaison des précédentes au moyen des opérateurs et des parenthèses

ex : (a+b)*c

- Les opérateurs et les expressions booléennes :

Opérateurs relationnels binaires
- Est égal : noté =
- Est différent : noté <>
- Est strictement plus grand : noté >

- Est strictement plus petit : noté **<**
- Est plus grand ou égal : noté **>=**
- Est plus petit ou égal : noté **<=**

Ces opérateurs permettront d'utiliser des prédicats appelés tests.

Opérateurs booléens binaires
- Et : noté **and**
- Ou : noté **or**

Opérateur booléen unaire
- Non : noté **not**

Expressions booléennes

Une expression booléenne est :
- une constante booléenne : (`true` ou `false`)
- une variable booléenne
- deux expressions arithmétiques combinées avec un opérateur relationnel
- des expressions booléennes combinées au moyen des opérateurs booléens

exemples :
a, b des entiers
c, d des booléens
- `a=b`
  expression booléenne dont la valeur est vraie si a=b et faux si a<>b
- `p` **`and`** `q`
- `p` **`and not`** `q`
- `(a=b)` **`or`** `q`

Instructions

- définitions :

Une **instruction** est une action élémentaire commandant à la machine un calcul, ou une communication avec un de ses périphériques (entrant ou sortant).

Une instruction peut être :

- Une **affectation** et/ou opération arithmétique.
  L'affectation est l'action élémentaire principale puisque c'est par son intermédiaire que l'on peut modifier la valeur d'une variable. L'affectation a pour syntaxe :

```
variable := valeur ;
```

- Un **affichage**.
  L'affichage est l'action élémentaire permettant à l'utilisateur de fournir un ou plusieurs résultats issus de son algorithme. Ainsi l'affichage peut être une simple phrase mais aussi peut permettre la visualisation du contenu (typé) d'une variable. L'affichage dans le langage algorithmique se fait par l'intermédiaire de la commande :

```
write ''simple phrase'';
```

  ** Contrairement au LSD[03], nous introduisons la possibilité d'afficher non pas sur la sortie standard mais dans la console de notre outil.
  ** En complément au LSD[03], nous élargissons le **write** à l'affichage de chaînes de caractères et de booléens.

- Une **lecture** au clavier ou dans un fichier.
  La lecture au clavier est l'action élémentaire permettant de spécifier par une intervention humaine la valeur d'une variable. Cette action symbolise donc la communication avec un périphérique d'entrée tel que le clavier. Bien évidemment, la valeur saisie par l'utilisateur de l'algorithme se doit d'être du même type que la variable recevant la valeur. La saisie se fait par l'intermédiaire de la commande :

```
read variable ;
```

  ** Contrairement au LSD[03], nous introduisons la possibilité d'effectuer une lecture non pas depuis l'entrée standard mais au départ de la console de notre outil.
  ** En complément au LSD[03], nous élargissons le **read** à la lecture de chaînes de caractères et de booléens.

Les choix conditionnels

Le choix conditionnel en algorithmique est une instruction de branchement permettant de décider, dans un contexte donné, quelle sera la séquence d'instructions à appliquer. Elle permet ainsi à l'algorithme de prendre des décisions concernant son exécution. Sa syntaxe est :

```
If <condition>
    Then        InstructionA
                ...
    Else        InstructionB
                ...
Fi
```

La section **Else** ... est facultative. La partie <condition> est essentielle puisque c'est elle qui décide de l'exécution des instructions conditionnées. Elle est de type booléen. Cela signifie que:

- si le prédicat `<condition>` vaut `true`, seul le bloc instructionA sera exécuté.
- si le prédicat `<condition>` vaut `false`, seul le bloc instructionB (s'il existe) sera exécuté.

L'introduction de '**FI**' permet d'éviter toute ambiguïté dans le cas de choix conditionnels imbriqués.

Les boucles

Si le langage algorithmique se limitait aux structures précédentes, nous ne pourrions pas faire grand chose de plus qu'avec une calculatrice. On pourra notamment remarquer que lorsque l'on a un grand nombre d'opérations similaires à faire, le programme se déroulant de façon linéaire, il est indispensable d'écrire ces opérations autant de fois que nécessaire. On introduit donc d'autres structures de contrôle : les boucles.

- Boucles « Tant Que »

Dans le cas, par exemple, où l'on cherche le premier entier i>1 tel que la i^4+4 soit premier, on ne peut (à moins de réfléchir un peu) pas savoir la taille de l'ensemble d'indices à observer. Peut-être même qu'un tel nombre n'existe pas... Une chose est certaine, si ce nombre existe, une boucle permettra de le trouver.

```
While <condition> do
    Instructions
    ...
od
```

Nous rappelons que nous nous conformons aux limites que nous avons fixées au point 2.5. Par conséquent, nous n'avons pas besoin d'autres boucles dans le cadre de notre travail. Il nous semble que la boucle « Répéter ... jusqu'à ... », non prévue dans LSD[03], constitue pourtant un élément judicieux pour l'apprentissage des concepts de l'algorithmique. Son implémentation pourrait faire l'objet d'un prototype ultérieur et être intégrée dans l'outil de création d'algorithme et dans l'outil d'apprentissage.

Les procédures

Lorsqu'un ensemble d'instructions réalise un certain algorithme et que cet ensemble est utilisé à différents endroits, une bonne pratique de programmation consiste à définir une procédure.

La structure d'une procédure ressemble à s'y méprendre à celle d'un programme si ce n'est qu'elle peut prendre un certain nombre de paramètres en entrée.

```
Procedure nomProcedure(var var1: type1,…)
{Déclaration des variables locales}
…
Begin
    …
    {Liste des instructions}
    …
End ;
```

Nous avons été tentés d'ajouter au langage LSD[03], la notion de fonction qui diffère de la procédure en ce qu'elle doit renvoyer une valeur. Nous avons cependant jugé cette extension inutile car un résultat peut être transmis en ajoutant un argument à la procédure.

```
Procedure NomFonction(var var1: type1,…, var
varRetour : typeRetour)
{Déclaration des variables locales}
…
Begin
    …
    {Liste des instructions}
    …
    varRetour := { valeur retournée }
    …
End ;
```

Le LSD[03] traite uniquement le cas du passage des arguments par variable. Un élément important dans l'algorithmique est le passage des arguments par variable et par valeur. Il nous semble que le prototype pourrait utilement être élargi, ultérieurement, à ces notions.

5.3.3.2. Conception d'un traducteur de LSD[03]

Pour être compris par l'outil d'apprentissage, l'algorithme écrit en LSD[03] doit être traduit dans un langage de programmation compris par l'outil. Il faut donc construire un traducteur transformant le LSD[03] dans le langage de développement de l'outil même.

5.3.3.3. Finalisation

La dernière étape de la réalisation du traducteur d'algorithme en langage LSD[03] consiste à fournir à l'utilisateur les éléments nécessaires à son utilisation, à savoir une méthode de traduction et une aide en ligne.

5.3.4. Module « Affichage Algorithme »

L'algorithme est affiché comme une succession d'instructions écrites en LSD[03]. Lors de l'exécution de l'algorithme, l'instruction courante est mise en évidence.

5.3.5. Module « Création Invariant »

5.3.5.1. La logique des prédicats du premier ordre

La logique des prédicats du premier ordre permet d'introduire les notions nécessaires à l'écriture des invariants.

Rappelons-en quelques éléments essentiels dans le tableau 1.

Symbole	Signification
Variable	
Nom de variable (variable dite libre)	Nom d'une variable provenant du programme de l'algorithme en LSD[03]. Cette variable a un type et une valeur : Il peut s'agit d'un entier, d'une chaîne de caractères, d'un booléen ou d'un tableau d'entiers, d'un tableau de chaîne de caractères, d'un tableau de booléens. Ceci comprend également un élément d'un tel tableau (exemple : un_tableau[1] pourrait être l'entier contenu dans le tableau 'un_tableau' à l'indice 1)
Nom de variable (variable dite liée)	Une variable liée est une variable qui ne provient pas du programme de l'algorithme mais est créé dans l'invariant pour établir la propriété de relation entre les variables libres (exemple : la variable indice qui permet de parcourir tous les éléments d'un tableau provenant de l'algorithme).
Opérateurs relationnels binaires	
>	Plus grand
<	Plus petit
=	Égal
>=	Plus grand ou égal
<=	Plus petit ou égal
!=	Différent
Constantes booléennes	
vrai	Vrai
faux	Faux
Opérateurs binaires de logique mathématique	
Λ	Et
V	Ou
$\Rightarrow$	Implication
Opérateur unaire de logique mathématique	
$\neg$	Non

Tab. 1 - Symboles de la logique des prédicats du premier ordre

5.3.5.2. Conception d'un traducteur pour la logique du premier ordre

La conception d'un traducteur pour la logique de premier ordre suit le même raisonnement que celle pour le langage LSD[03].

5.3.5.3. Finalisation

Nous veillons à également fournir à l'utilisateur les éléments nécessaires à l'utilisation du traducteur pour la logique de premier ordre, à savoir une méthode de traduction et une aide en ligne.

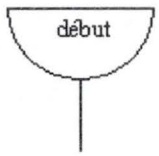
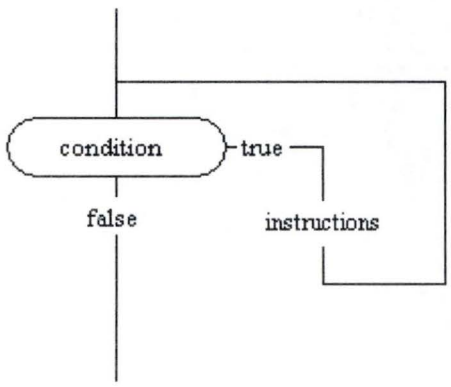
5.3.6. Module « Affichage Invariant »

L'invariant est affiché sous forme de logique des prédicats du premier ordre. Lors de l'exécution de l'algorithme, un test est exécuté pour contrôler si l'invariant est vérifié. Si c'est le cas, l'invariant est mis en évidence.

5.3.7. Module « Affichage organigramme »

L'organigramme doit être capable de représenter toutes les instructions de l'algorithme. Il nous faut donc parcourir la définition du langage algorithmique LSD[03] pour définir les éléments de l'organigramme.

La figure 5 présente un tableau de correspondances.

Début de programme	:	
Instruction simple (affectation, lire, écrire)	:	instruction
Boucle :	:	

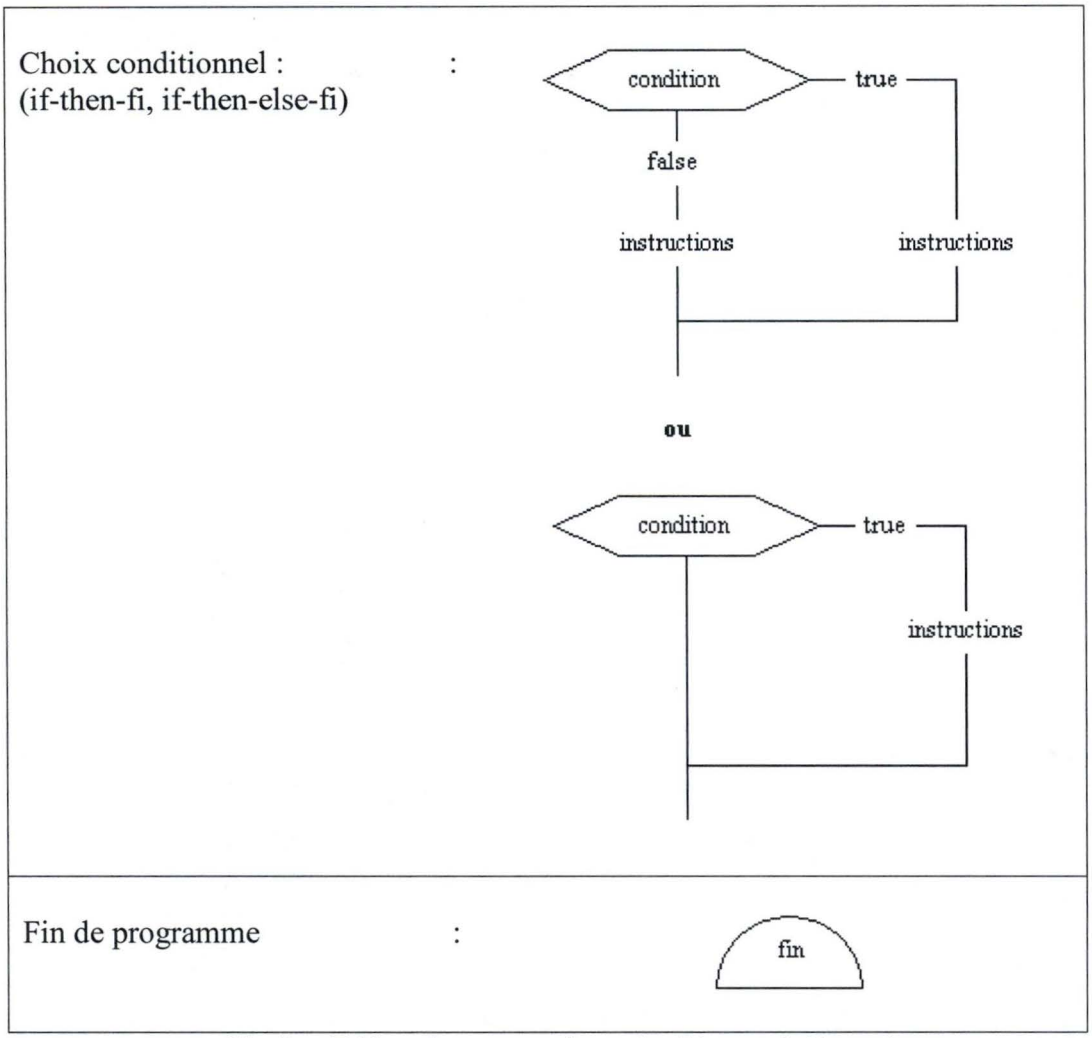


Fig. 5 - Tableau de correspondance des éléments de l'organigramme

La principale difficulté dans la conception de ces éléments d'organigramme réside dans le fait que certains d'entre eux peuvent être imbriqués.

L'élément dans l'organigramme associé à l'instruction courante de l'algorithme en exécution est mis en évidence.

5.3.8. Module « Affichage Console »

La console représente les interactions entrées/sorties de l'algorithme. Ce sont les instructions « read » et « write » du LSD[03] qui génèrent ces illustrations.

L'instruction « read » attend, de la part de l'utilisateur, la saisie au clavier d'une certaine valeur et son affichage dans la console pour pouvoir continuer l'exécution de l'algorithme.

L'instruction « write » affiche un résultat dans la console.

5.3.9. Module « Affichage Etat de la mémoire »

L'état de la mémoire illustre les données nécessaires à l'exécution de l'algorithme et qui sont stockées en mémoire, ainsi que les valeurs qu'elles prennent à chaque étape de cette exécution.

5.3.10. Module « Affichage Historique »

L'historique reprend séquentiellement les opérations qui sont effectuées comme le choix d'un algorithme par l'utilisateur et l'exécution de cet algorithme.

L'historique peut être visualisé et imprimé.

5.3.11. Module « Affichage Aide »

L'aide apportée à l'utilisateur est de deux types. Il s'agit soit de la consultation de rubriques d'aide, soit de l'aide contextuelle à travers tout l'outil.

5.4. Conception d'outils

Les outils que nous développons sont donc au nombre de trois.

5.4.1. L'outil d'apprentissage à l'algorithmique

L'outil d'apprentissage proprement dit, objectif principal de notre mémoire, qui reprend les modules :

- « Affichage Algorithme »,
- « Affichage Invariant »,
- « Affichage Organigramme »,
- « Affichage Console »,
- « Affichage Etat de la mémoire »,
- « Affichage Langue »,
- « Affichage Aide »,
- « Affichage Historique ».

5.4.2. L'outil de création d'algorithmes

L'outil de création d'algorithmes, destiné au concepteur de contenu, comprend les modules :

- « Création Algorithme »
- « Création Invariant ».

5.4.3. L'outil de création de traduction

L'outil de création de traduction comprend le module :

- « Création Langue ».

5.5. Interactions entre les outils

Les outils de création de traduction et de création d'algorithmes interagissent avec l'outil d'apprentissage à l'algorithmique.

En effet, l'outil de création de traduction met à jour le module « Affichage Langue ». Et l'outil de création d'algorithmes, les modules « Affichage Invariant », « Affichage Algorithme », « Affichage Organigramme », « Affichage Console » et « Affichage Etat de la mémoire ».

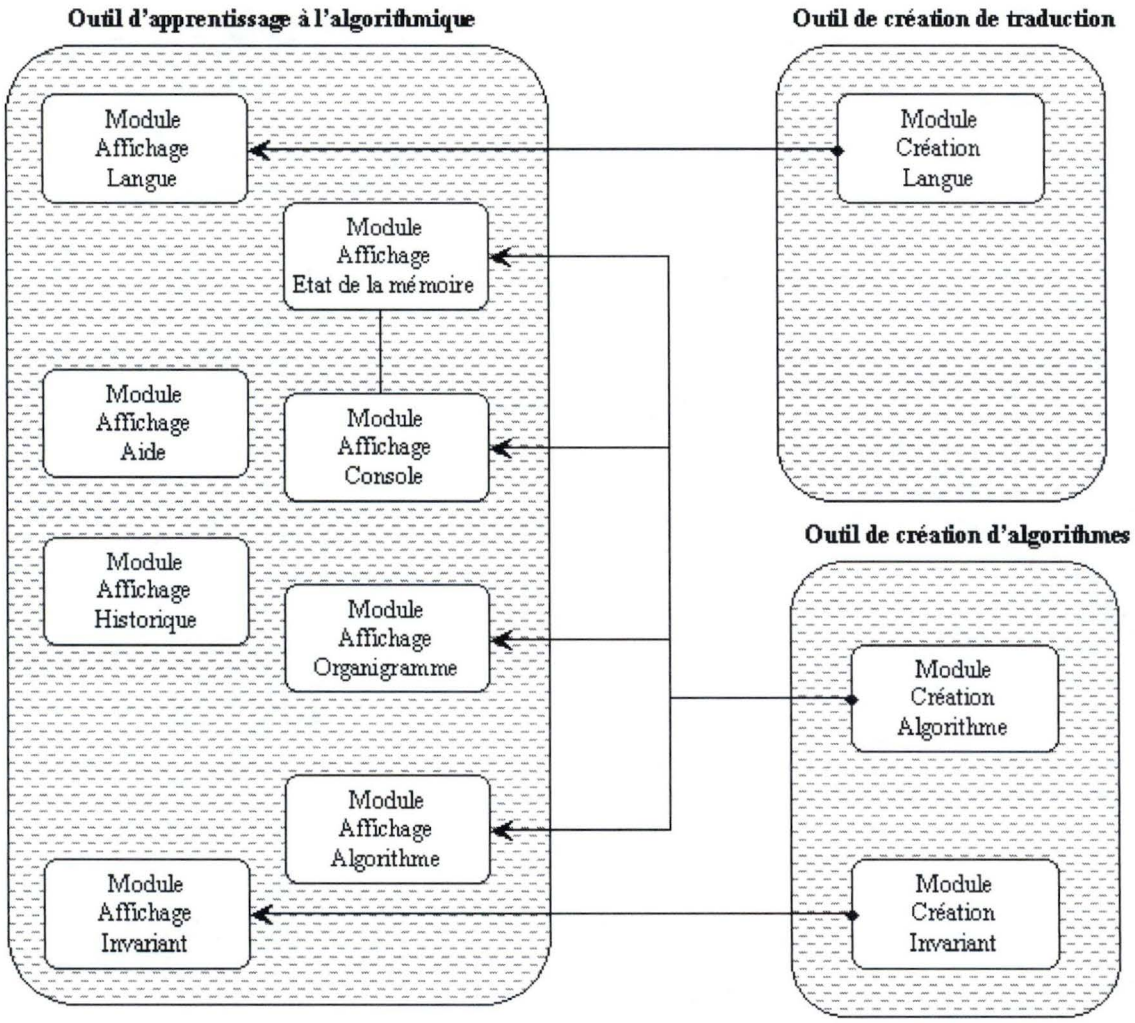


Fig. 6 - Interactions entre les outils

Chapitre 6

Spécification de l'interface

6.1. De l'importance d'une interface utilisateur de qualité

L'analyse des besoins et la spécification des fonctionnalités achevées ainsi que la pertinence du projet confirmée, l'étape suivante, la plus importante, est la conception du design traduisant le souci pédagogique.

L'une des principales activités de la conception d'applications informatiques est la conception de l'interface utilisateur, c'est-à-dire la création des entrées et des sorties impliquées lorsque l'utilisateur interagit avec le système pour exécuter une tâche.

Ce chapitre porte sur le dialogue entre l'utilisateur et l'outil informatique, un phénomène mieux connu sous le nom d'interaction hommes-machine ou IHM.

L'interface est une composante de tout premier ordre dans les processus d'apprentissage.

Pour chaque entrée, nous devons étudier l'interaction entre l'utilisateur et l'ordinateur, puis concevoir une interface capable de traiter cette entrée. Nous devons de même concevoir l'interaction pour chaque sortie produite à la demande d'un utilisateur.

Ce chapitre débute par une discussion de l'interface utilisateur en présentant les fondements de la conception orientée utilisateur, le développement du domaine de l'interaction hommes-machine et plusieurs métaphores utilisées pour décrire une interface utilisateur. Il propose également diverses astuces pour assurer la convivialité d'un système.

Nous avons décidé de séparer la conception des interfaces des trois[11] outils à développer, car elles font chacune appel à des expertises et des technologies différentes. Comme pour toutes les autres composantes du système, une coordination méticuleuse de la conception est toutefois indispensable.

Beaucoup d'utilisateurs croient que l'interface utilisateur est développée et ajoutée à un système vers la fin du processus de développement. L'interface utilisateur d'un système interactif est cependant bien davantage que cela : elle représente tout ce qui entre en contact avec l'utilisateur final lorsqu'il se sert du système. Pour l'utilisateur, l'interface

[11] L'outil d'apprentissage à l'algorithmique, l'outil de création d'algorithmes et l'outil de création de traduction (cf. point 5.4.)

utilisateur est le système. L'interface utilisateur joue un rôle clé dans le prototypage évolutif. En effet, elle apparaît très tôt dans le processus d'ingénierie avec pour effet de favoriser les échanges avec le client et est un vecteur à part entière dans le développement.

Vu l'importance de l'interface utilisateur pour les utilisateurs du système, nous concentrons notre attention sur une technique d'analyse et de conception qui place l'interface utilisateur au centre du processus de développement. Cette technique met l'accent sur trois grands principes :

- Se concentrer dès le début sur les utilisateurs, c'est-à-dire le professeur donnant le cours d'Introduction à l'Algorithmique et ses étudiants.
- Analyser l'environnement pour garantir au maximum la convivialité du système.
- Utiliser un développement itératif, ce qui conforte notre approche par prototypage évolutif, développée au point 3.4.

6.2. Méthode utilisée pour la conception de l'IHM

Nous avons retenu de notre cursus universitaire[12] que l'intuition et le bon sens permettaient de développer des interfaces-utilisateur attrayantes mais que cela ne suffisait pas.

Outre les interviews avec notre client, nous avons attaché une importance particulière à l'observation[13] des étudiants sur leur lieu de cours afin de comprendre l'activité effectivement réalisée. Ces deux approches ont consolidé notre analyse de la tâche principale.

En effet, connaître la tâche permet de structurer l'interface hommes-machine selon le point de vue de l'utilisateur.

Cette démarche nécessite tout d'abord de déduire de la tâche, les diverses contraintes qui devront déterminer l'interface. Pour ce faire, nous devrons tout d'abord procéder à une analyse du contexte (poste de travail et stéréotype des utilisateurs) afin de dégager un premier niveau de contraintes. Ensuite, nous effectuerons une analyse de la tâche au sens strict, c'est-à-dire que nous nous pencherons sur la tâche elle-même afin de la structurer. A partir de cette analyse de la tâche, nous pourrons alors préciser les spécifications ergonomiques et les spécifications fonctionnelles. Les résultats de cette analyse de la tâche nous fourniront un deuxième niveau de contrainte : ces contraintes seront déterminées par les critères d'utilité et d'utilisabilité de l'interface.

---

[12] En particulier, le cours de « Conception d'Interfaces Hommes-Machine » du Professeur François Bodart [BOD01]

[13] Etant donné mes activités professionnelles qui ont bien évidemment lieu pendant que les étudiants du jour ont cours, je me suis basée sur mes propres expériences d'étudiante.

6.3. Interface de l'outil d'apprentissage à l'algorithmique

6.3.1. Exigences de l'interface

Si l'interface de l'ordinateur était passive comme celle de la télévision, la conception d'un tel outil d'apprentissage perdrait de son sens. En effet, tout l'intérêt de l'ordinateur tient à ses capacités interactives. Un acte interactif consiste essentiellement à constater un objet dans l'interface, par exemple une liste de choix sur un écran, et à susciter une action sur cet objet désiré par une sélection souvent accomplie par la souris ou par un clavier.

L'interface de l'outil est accessible via une page web.

L'interface doit montrer clairement les objectifs à atteindre.

L'interface de l'outil doit permettre aux utilisateurs de choisir une langue, de choisir un algorithme à exécuter, d'exécuter cet algorithme instruction par instruction, d'avancer ou de reculer d'une instruction, d'accéder à l'historique des opérations qu'ils ont effectuées, d'accéder à l'aide dont il en besoin, ainsi qu'aux informations propres à l'outil.

Pour obtenir une bonne satisfaction subjective de la part des utilisateurs, il faut obtenir des résultats rapidement à travers une interface conviviale et attrayante.

La facilité d'apprentissage et la simplicité d'utilisation sont deux concepts souvent conflictuels, car une interface facile à apprendre n'est pas nécessairement facile à utiliser.

La présentation électronique a modifié la façon de lire des utilisateurs, par rapport à la version papier. En effet, sa lecture n'est plus linéaire et systématique. Il faut donc adapter le contenu au médium c'est-à-dire rendre l'information plus accessible, la synthétiser et surtout planifier l'ordre de présentation.

L'utilisateur peut tester le modèle théorique proposé en se 'promenant' dans l'algorithme puisqu'il peut l'exécuter, instruction par instruction, et revenir en arrière, instruction par instruction.

Il peut aussi, à tout moment, interrompre l'exécution d'un algorithme, passer à l'exécution d'un autre algorithme, c'est-à-dire rendre son apprentissage plus personnel et plus adapté.

Tout au long de ses manœuvres, l'utilisateur est guidé par le programme. Ce sont toutes ces possibilités qui donnent à l'outil d'apprentissage un aspect plus attractif et plus convivial.

6.3.2. Analyse du contexte

6.3.2.1. Analyse du poste de travail

L'environnement physique correspond à un ordinateur « classique » ayant un accès (intranet, internet, voire extranet) à l'outil d'apprentissage à algorithmique et muni d'un navigateur internet récent.

Nous supposons tout de même que l'utilisateur dispose d'un écran de bonne dimension, soit supérieure ou égale à dix-sept pouces.

6.3.2.2. Stéréotype des utilisateurs

Nous distinguons deux stéréotypes d'utilisateurs concernés par le logiciel : D'une part, le professeur chargé du cours d'Introduction à l'Algorithmique et ses assistants et d'autre part, les étudiants inscrits à ce cours.

Du point de vue du professeur et de ses assistants :

Nous pouvons dire de ceux-ci qu'ils possèdent une certaine habitude et habilité à l'utilisation de systèmes informatiques. Nous estimons donc que leur expérience du système est moyenne.

Le professeur et ses assistants maîtrisent complètement l'exécution des algorithmes. Nous qualifions dès lors leur expérience de la tâche d'élevée.

La motivation à employer l'outil d'apprentissage sera vraisemblablement élevée pour le professeur et ses assistants, demandeurs d'un tel outil.

Quant à l'expérience d'un moyen d'interaction complexe, elle peut être qualifiée de modérée. En effet, le moyen d'interaction d'un ordinateur et d'un navigateur internet n'est plus considéré comme complexe. Malgré tout, il n'est pas immédiat : sans quoi, tout le monde pourrait allumer un ordinateur, se connecter à Internet et surfer sur des pages web. Cela s'apprend.

Du point de vue des étudiants :

Nous pouvons dire de ceux-ci qu'ils possèdent vraisemblablement une certaine habitude et habilité à l'utilisation de systèmes informatiques. Nous estimons donc que leur expérience du système est moyenne.

Les étudiants, principaux destinataires de l'outil, en sont, en principe, à leurs débuts en algorithmique. Cependant, ils possèdent au sortir de l'enseignement secondaire – du moins, l'espère-t-on – des notions de mathématiques et de logique. Ils peuvent donc très vite lire les

instructions d'un algorithme et en percevoir le sens, même s'ils n'en perçoivent pas toutes les nuances ou la portée. Nous qualifions dès lors leur expérience de la tâche de modérée.

La motivation a priori des étudiants à employer l'outil d'apprentissage est difficile à évaluer : nous devons nous en tenir à une vue théorique puisque l'outil ne leur a pas encore été soumis. Néanmoins, cet outil devant les aider à mieux appréhender les concepts de l'Algorithmique, nous supposons que leur motivation peut être qualifiée de moyenne à élevée.

Quant à l'expérience d'un moyen d'interaction complexe, nous estimons que, à l'instar du stéréotype du professeur, elle peut être qualifiée de modérée.

Conclusion
L'outil d'apprentissage étant destiné aux étudiants, nous ne prenons en compte que le stéréotype des étudiants pour notre analyse.

Nous rassemblons les paramètres descriptifs développés ci-dessus dans un tableau synthétique.

Paramètre	Valeur
Expérience de la tâche	Modérée
Expérience du système	Moyenne
Motivation	Moyenne à élevée
Expérience d'un moyen d'interaction complexe	Modérée

Tab. 2 - Paramètres descriptifs du stéréotype des étudiants

6.3.2.3. Contexte de réalisation de la tâche

L'outil ayant pour objectif l'apprentissage de l'algorithmique par l'animation d'exécutions d'algorithmes, nous estimons qu'il peut être utilisé soit dans l'auditoire par le professeur chargé de cours, soit dans un endroit calme (depuis la maison, depuis une bibliothèque) par les étudiants.

Nous considérerons que la tâche est mono-traitement, l'utilisateur ne faisant rien d'autre lorsqu'il utilise le logiciel

6.3.2.4. Conclusions

Cet examen du contexte de la tâche nous conduit à identifier certaines contraintes qui influencent l'analyse de la tâche et donc le développement de l'IHM.

Nous pouvons, en effet, déjà exprimer les contraintes suivantes :

- Il importe de réduire l'effort mental de l'utilisateur face à l'IHM puisqu'elle lui servira uniquement pour l'apprentissage des quelques concepts de base du cours d'Introduction à l'Algorithmique. Il faut que l'IHM reste d'une utilisation très intuitive. Autrement dit, les distances sémantiques et articulatoires, à l'œuvre dans les golfes d'exécution et d'évaluation devront être aussi minimes que possible. Il nous faudra donc identifier la représentation mentale et cognitive que l'utilisateur se forge des concepts de base d'un cours d'Introduction à l'Algorithmique, afin de la reproduire au sein de l'IHM.

- Afin de gérer la taille limitée d'un écran standard d'ordinateur, il faut une interface d'outil qui affiche toute l'information relative à l'exécution d'un algorithme et ce, afin de reproduire la vision qu'en a l'étudiant lorsque le professeur l'illustre au tableau.

- L'utilisation d'artifices graphiques et d'objets interactifs doit être redondante vis-à-vis de la sémantique des illustrations, afin de produire un univers d'information cohérent, où l'utilisateur pourra sans encombre visualiser les informations dont il a besoin.

- Tant les messages que les objets interactifs doivent être suffisamment suggestifs pour que l'utilisateur s'y retrouve sans compétence spécifique dans cet outil particulier.

6.3.3. Analyse de la tâche

L'analyse de la tâche recouvre deux aspects distincts :

- d'une part, la définition de l'activité de la tâche,
- d'autre part la structuration de la connaissance relative à son exécution.

6.3.3.1. Cadrage du champs d'investigation

Afin de limiter l'analyse de la tâche, nous avons uniquement traité les deux tâches les plus importantes de l'outil d'apprentissage à l'algorithmique, à savoir « Exécuter un algorithme » et « Exécuter une instruction de l'algorithme ».

La même analyse pourrait être réalisée pour les tâches connexes, « Choix de la langue », « Consultation/Impression de l'aide », « Consultation/impression de l'historique des opérations », « Consultation des informations relatives à l'outil ». Mais comme nous le verrons, cette analyse de la tâche est coûteuse en temps et en effort.

Etant donné que l'outil d'apprentissage de l'algorithmique que nous développons n'est pas une application « critique », comme notamment les logiciels embarqués qui assurent le fonctionnement d'un avion, par exemple, nous estimons que l'investissement dans une analyse complète de la tâche n'est pas opportune.

6.3.3.2. Définition de l'activité de la tâche

Choisir un algorithme : il s'agit de choisir un algorithme parmi une liste actuellement de quatre.

Exécuter un algorithme : il s'agit d'exécuter intégralement un algorithme c'est-à-dire de parcourir l'intégralité de ses instructions jusqu'à sa terminaison[14].

Exécuter une instruction d'un algorithme : il s'agit d'exécuter une (et une seule) instruction bien précise de l'algorithme.

Animer l'organigramme : il s'agit d'afficher l'organigramme correspondant aux instructions de l'algorithme. Dans le cas de l'exécution d'une instruction de l'algorithme, l'élément de l'organigramme qui représente cette instruction en cours d'exécution est mis en évidence.

Animer l'état des données en mémoire : il s'agit d'afficher les données dont l'algorithme à besoin lors de son exécution et de mettre leur valeur à jour au cours de l'exécution.

Animer la console des entrées/sorties : il s'agit d'afficher les entrées/sorties générées par l'exécution de l'algorithme.

Animer l'invariant : il s'agit d'afficher l'invariant lié à l'algorithme en cours d'exécution. A chaque instruction, la validité de l'invariant est vérifiée. Si l'invariant est vérifié, celui-ci est mis en évidence.

Animer l'instruction : il s'agit de mettre en évidence l'instruction qui est en cours d'exécution.

6.3.3.3. Structuration de la tâche

Le graphe de structuration de la tâche (Figure 7) représente graphiquement la décomposition de la tâche en buts et sous-buts. Les flèches en trait plein matérialisent la relation de structuration, tandis que les flèches en trait pointillé dénotent la relation de séquencement.

L'utilisation de couleur a pour seul but de tenter un tant soit peu de clarifier le graphe. Toutes les flèches issues d'une même tâche/sous-tâche sont de même couleur sauf dans le cas de la sous-tâche 'Choisir un algorithme' pour laquelle les flèches de couleur rose sont les relations de séquencement sous la hiérarchie de la tâche 'Exécuter une instruction d'un algorithme' et les flèches de couleur mauve, les relations de séquencement sous la hiérarchie de la tâche 'Exécuter un algorithme'.

[14] Nous posons l'hypothèse que l'algorithme termine.

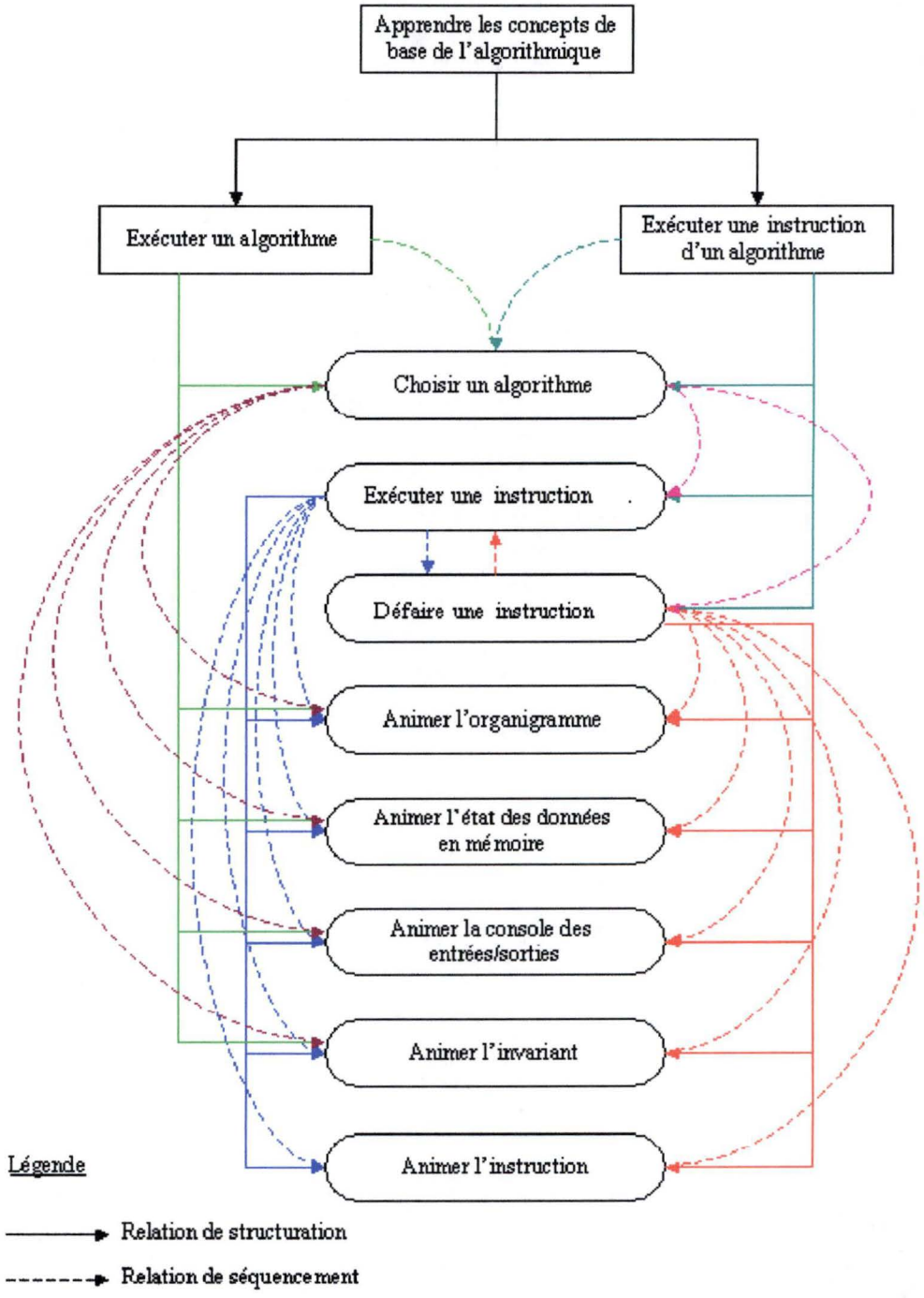


Fig. 7 - Graphe de structuration de la tâche

6.3.3.4. Décomposition en procédures et actions

La décomposition pour la première tâche 'Exécuter un algorithme' peut être la suivante :

```
      (* choix parmi quatre algorithmes *)
algorithme_choisi := choisir_algo(algorithme) ;
type_animation := choisir_anim(animation) ;
```

```
SI type_animation = total
      ALORS
         animer_organigramme(→algorithme_choisi) ;
         animer_mémoire(→algorithme_choisi) ;
         animer_console(→algorithme_choisi) ;
         animer_invariant(→algorithme_choisi) ;
FINSI
```

La décomposition pour la seconde tâche 'Exécuter une instruction d'un algorithme' peut être la suivante :

```
      (* choix parmi quatre algorithmes *)
algorithme_choisi := choisir_algo(algorithme) ;
instruction_courante := 1 ; (* première instruction *)
compter(→algorithme_choisi, ←nb_instructions) ;
type_animation := choisir_anim(animation) ;
      (* on avance d'une instruction *)
TANT QUE type_animation = avance
         ET vérifier_instruction(→instrution_courante,
                                 →nb_instructions,
                                 ←instruction_vérifiée)
      DEBUT
            animer_organigramme(→algorithme_choisi,
                                →instruction_courante) ;
            animer_mémoire(→algorithme_choisi,
                           →instruction_courante) ;
            animer_console(→algorithme_choisi,
                           →pinstruction_courante) ;
            animer_invariant(→algorithme_choisi,
                             →instruction_courante) ;
            animer_instruction(→algorithme_choisi,
                               →instruction_courante) ;
            instruction_courante := instruction_courante
                                    + 1 ;
      FIN
      (* on recule d'une instruction *)
TANT QUE type_animation = recule
         ET vérifier_instruction(→instrution_courante,
                                 ←instruction_vérifiée)
      DEBUT
            instruction_courante := instruction_courante
                                    - 1 ;
            animer_organigramme(→algorithme_choisi,
                                →instruction_courante) ;
            animer_mémoire(→algorithme_choisi,
                           →instruction_courante) ;
            animer_console(→algorithme_choisi,
                           →instruction_courante) ;
            animer_invariant(→algorithme_choisi,
                             →instruction_courante) ;
            animer_instruction(→algorithme_choisi,
                               →instruction_courante) ;
      FIN
```

Ces décompositions sont très simplifiées puisqu'elles ne comportent aucune gestion d'erreurs (exemple : afficher un message d'erreur lorsque l'utilisateur veut avancer d'une instruction alors qu'il est arrivé au bout de l'algorithme).

6.3.4. Expression du produit de l'analyse de la tâche

6.3.4.1. Identification des fonctions sémantiques

Les fonctions sémantiques (au sens des services de l'application) sont directement issues des actions identifiées dans les procédures :

Les arguments sont précédés de :
→ si l'argument est en entrée,
← si l'argument est en sortie

```
compter(→algorithme_choisi, ←nb_instructions) ;
animer_organigramme(→algorithme_choisi) ;
animer_organigramme(→algorithme_choisi,
                    →instruction_courante) ;
animer_mémoire(→algorithme_choisi) ;
animer_mémoire(→algorithme_choisi,
                →instruction_courante) ;
animer_console(→algorithme_choisi) ;
animer_console(→algorithme_choisi,
                →instruction_courante) ;
animer_invariant(→algorithme_choisi) ;
animer_invariant(→algorithme_choisi,
                  →instruction_courante) ;
animer_instruction(→algorithme_choisi,
                    →instruction_courante) ;
vérifier_instruction(→instrution_courante,
                      →nb_instructions,
                      ←instruction_vérifiée) ;
vérifier_instruction(→instrution_courante,
                      ←instruction_vérifiée) ;
```

6.3.4.2. Graphe d'enchaînement des fonctions sémantiques

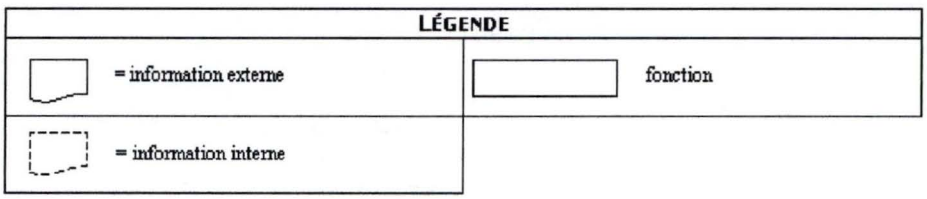


Fig. 8 - Légende pour la figure du graphe d'enchaînement

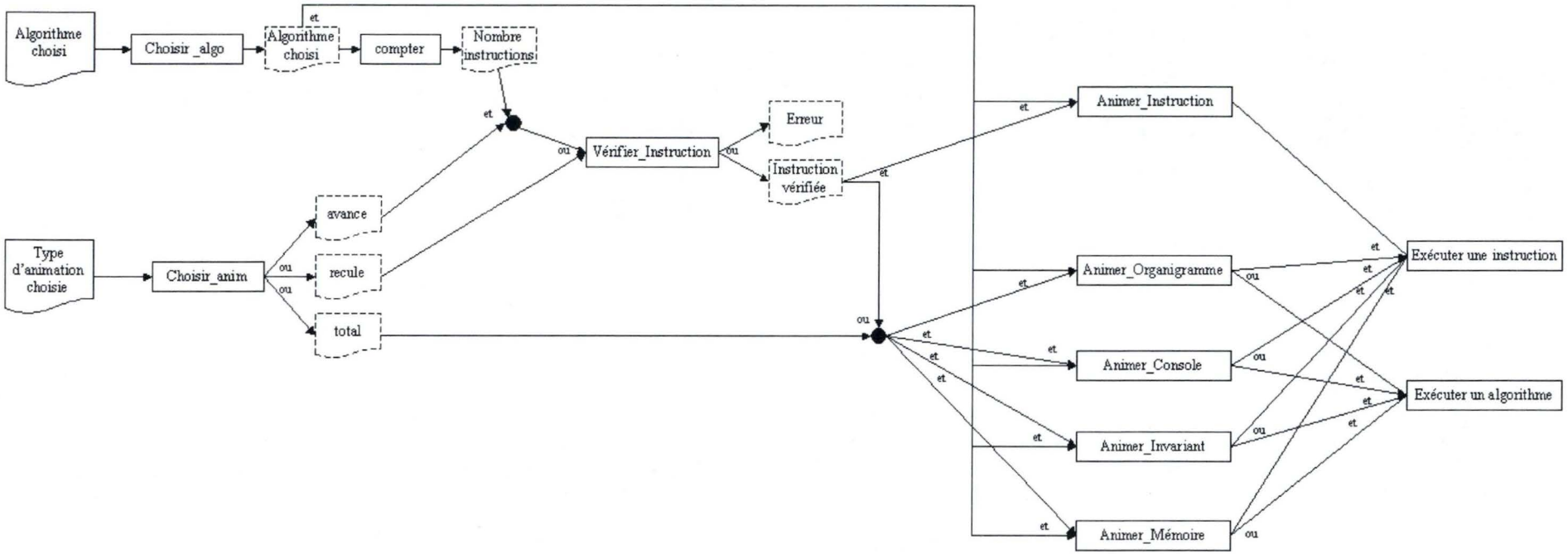


Fig. 9 - Graphe d'enchaînement

6.3.5. Choix des attributs de dialogue

Les quatre attributs caractérisant le dialogue de l'interface sont :

1. *Le contrôle du dialogue* :
   Le contrôle du dialogue est globalement externe pour la tâche, c'est-à-dire sur l'initiative de l'utilisateur. Il est cependant partiellement interne pour les listes de choix chargées de lancer les procédures des sous-tâches : adapter la langue de l'interface, afficher les données relatives à un algorithme.

2. *Le mode de dialogue* :
   Il y a quatre sous-tâches : choisir la langue de l'interface, exécuter un algorithme, consulter une rubrique d'aide, consulter l'information relative à l'outil. Le mode de dialogue est asynchrone au sein de chaque sous-tâche : l'ordre d'exécution des actions est non-prédéterminé, non-séquentiel (par exemple, un algorithme peut être choisi avant le choix de la langue).
   L'ordre d'exécution des actions à l'intérieur d'une des sous-tâches est toutefois synchrone. Il faut, en effet, d'abord choisir un algorithme avant de procéder à son exécution.

3. *Le mode de déclenchement des fonctions*
   Le mode de déclenchement des fonctions est manuel et explicite car les fonctions doivent être déclenchées sur l'initiative de l'utilisateur à l'aide des actions prévues à cet effet. Ces actions se matérialiseront ultérieurement par des boutons de commande et des listes de choix (par exemple, un bouton 'Avancer d'une instruction' ou 'Imprimer').

4. *La métaphore* :
   Elle est basée sur la métaphore du mini-monde[15] :
   - le tableau noir de la classe peut être simulé électroniquement,
   - la métaphore est celle de la confrontation entre l'apprenant et le professeur écrivant au tableau.

   Nous revenons sur le rôle de la métaphore au point 6.3.8.

En résumé, le contrôle du dialogue est principalement externe; le mode de dialogue, asynchrone; le mode de déclenchement explicite; la métaphore est celle du mini-monde.

Eu égard aux paramètres identifiés à la section 6.3.2, on constate en parcourant le tableau de correspondance cité en annexe, intitulé « Sélection du style d'interaction ou de dialogue en fonction des paramètres descriptifs d'un stéréotype d'utilisateur » que deux styles d'interaction peuvent être retenus (ce sont ceux qui minimisent les écarts par rapport aux valeurs de référence dans les tableaux) : le langage naturel et le multi-fenêtrage.

[15] Métaphore présentée dans le cadre du cours de « Conception des Interfaces Hommes-Machine ».

6.3.6. Définition de la présentation

6.3.6.1. Identification des unités de présentation (UP)

Du graphe d'enchaînement présenté au point 6.3.4.2 et des sous-tâches que nous n'avons pas analysées en profondeur telles que mentionnées au point 6.3.3.1, nous dégageons quatre unités de présentations qui correspondent chacune à une sous-tâche.

UP1 : Choix d'une langue
UP2 : Exécution d'un algorithme
UP3 : Consultation de l'historique des opérations
UP4 : Consultation de l'aide et des informations relatives à l'outil

L'unité de présentation UP1 (Choix d'une langue) est la seule à interagir sur l'ensemble des autres.

L'unité de présentation UP2 (Exécution d'un algorithme) agit sur l'unité de présentation UP3.

6.3.6.2. Identification des fenêtres

Partant de la découpe en unités de présentation ci-dessus et du graphe d'enchaînement (voir point 6.3.4.2), on peut matérialiser les fenêtres et les panneaux ('panels') en regroupant certaines sous-tâches.

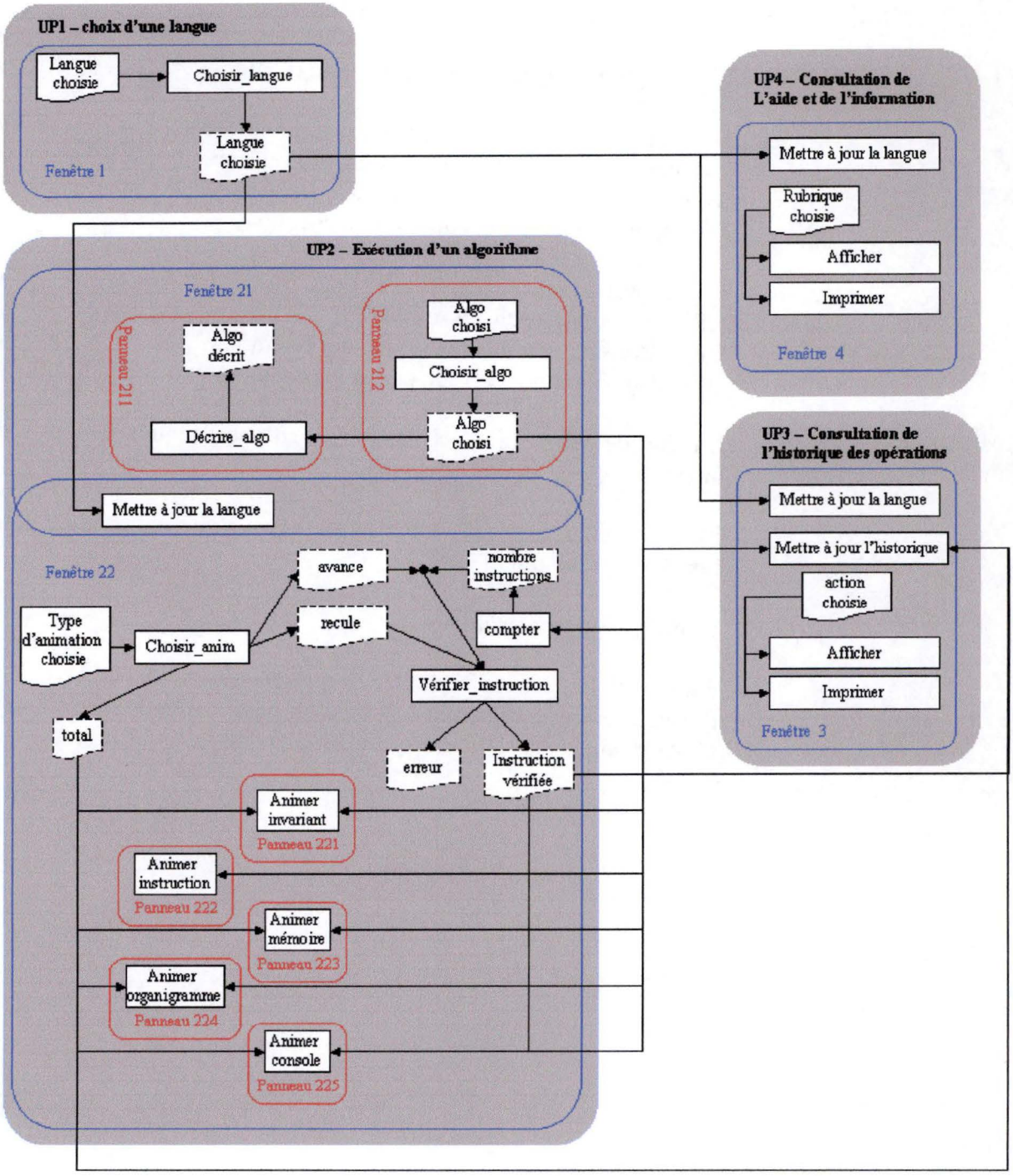


Fig. 10 - Unités de présentation de l'IHM

6.3.6.3. Sélection des Objets Interactifs Abstraits (OIA)

Les fenêtres qui viennent d'être définies constituent des fenêtres logiques. Du style d'interaction retenu pour ces fenêtres, nous pouvons en déduire que chaque fenêtre logique se concrétisera par une zone de dialogue.

Etant donné que notre outil doit apparaître dans un navigateur internet, nous optons pour les panneaux à onglets qui nous semblent tout à fait appropriés à l'illustration du multi-fenêtrage.

Le langage naturel quant à lui s'applique à la consultation de l'aide, de l'historique et des informations relatives à l'outil.

Les OIA associés aux informations à saisir dans les fenêtres sont :

Pour la fenêtre 1 → Fenêtre
Choix d'une langue : Liste de choix

Pour la fenêtre 2 → Fenêtre
Fenêtre 21 → Onglet
Panneau 211 :
Choix d'un algorithme : Liste de choix
Panneau 212 :
Description de l'algorithme : Champ de texte multilinéaire non éditable
Fenêtre 22 → Onglet
Type d'animation choisie : Bouton 'Avance', Bouton 'Recule', Bouton 'Exécute' et Bouton 'Réinitialise'
Panneau 221 :
Animation de l'invariant : Champ de texte multilinéaire non éditable
Panneau 222 :
Animation de l'instruction : Champ de texte multilinéaire non éditable
Panneau 223 :
Animation de l'état de la mémoire : Champ graphique multilinéaire non éditable
Panneau 224 :
Animation de l'organigramme : Champ graphique non éditable
Panneau 225 :
Animation de la console : Champ de texte multilinéaire éditable

Pour la fenêtre 3 → Onglet
Visualisation de l'historique : Champ de texte multilinéaire non éditable
Impression de l'historique : Bouton 'Imprimer'
Effacement de la vue l'historique : Bouton 'Effacer'

Pour la fenêtre 4 → Onglet
Choix d'une rubrique d'aide : Liste de choix
Visualisation de l'aide ou de l'information : Champ de texte multilinéaire non éditable
Impression de l'aide : Bouton 'Imprimer'
Afficher l'information : Bouton 'Afficher'

6.3.7. Critères d'utilité, d'utilisabilité

Les critères suivants d'évaluation de l'utilité et de l'utilisabilité sont ceux proposés par Ben Shneiderman [SHN92].

Critères	Commentaires	Niveau
Temps d'apprentissage	L'étudiant ne doit pas perdre de temps lors de la recherche d'une information particulière. L'effort intellectuel doit être concentré sur la compréhension du contenu, l'apprentissage de l'outil ne doit pas perturber la concentration de l'étudiant au risque de le décourager.	Faible
Rémanence d'apprentissage	L'objectif principal de la tâche à réaliser au moyen de cette interface implique de relativement courtes périodes d'usage intensif. Ces périodes d'utilisation intensives devraient être concentrées sur les périodes de cours présentiel, d'appropriation individuelle et de blocus. Cela a pour conséquence de ne pas exiger une période de rémanence importante.	Faible
Rapidité d'exécution	La rapidité d'exécution n'est pas fondamentale. Cependant, le coût de la communication internet pourrait rebuter certains utilisateurs. A cet effet, la concision du texte et le choix des illustrations sont primordiaux. Le poids de l'outil sera minimum afin de ne pas ralentir excessivement leur chargement. En outre, ce critère n'est pas sans impact sur la satisfaction subjective des utilisateurs. (cf. ci-dessous)	Modéré
Taux d'erreur de manipulation	Les erreurs de manipulation sont tolérables vu que l'importance de la tâche n'est pas critique. Nous veillerons tout de même à ce que ce taux reste raisonnable.	Faible
Taux d'erreur d'intention	Les erreurs d'intention risquent d'être faibles étant donné le faible nombre de fonctionnalités et le caractère très intuitif de l'interface.	Faible
Satisfaction subjective	Nous considérons que c'est là le critère le plus important pour ce genre de produit. L'outil se veut avant tout être une nouvelle méthodologie pédagogique. A cette fin, le besoin de changer la mentalité et les habitudes de l'apprenant nécessite de le stimuler fortement à l'usage de cette méthodologie. Il faut à tout prix éviter de susciter le désintérêt : un étudiant déçu ne sera pas enclin à poursuivre son exploration ou à y revenir ultérieurement..	Élevé
Couverture de la tâche	La couverture de la tâche doit être suffisante car les fonctionnalités qui ne seraient pas traitées représenteraient une raison pour l'étudiant de ne pas utiliser le produit. C'est en effet le degré de couverture de l'outil qui aidera à convaincre l'étudiant de l'intérêt voire de l'avantage qu'il peut en retirer	Élevé

Tab. 3 - Tableau d'évaluation de l'utilité et de l'utilisabilité

6.3.8. Le pouvoir des signes

Le mot-clé dans une interface-utilisateur est « métaphore ».

Nous rappelons que le principe de l'apprentissage à l'aide de métaphores est de se baser sur des connaissances antérieures dans un domaine connu pour acquérir des connaissances dans un nouveau domaine.

Le rôle d'une métaphore consiste donc à reproduire un environnement familier à l'apprenant afin que l'interface utilisateur devienne la plus intuitive possible. Nous avons choisi de développer la « Métaphore du Tableau Noir[16] » ainsi que la « Métaphore du lever du doigt ».

C'est en écoutant notre client que nous nous sommes approprié sa compréhension du système. Nous avons estimé que le « tableau noir » est un des éléments les plus importants du système actuel et que dans tous les cas, notre outil devait au minimum permettre ce que le « tableau noir » permet à l'enseignant pour arriver à son objectif. Cette métaphore informatique permet de proposer aux étudiants un univers plus ou moins analogue au monde réel puisque le professeur dessine actuellement au tableau les différentes étapes que génère l'exécution des algorithmes.

Nous avons également imaginé un petit personnage qui « lève le doigt », un petit guide interactif, qui peut apporter réponses et indications à l'utilisateur en difficulté, tout comme le professeur qui répondrait à un étudiant qui lèverait le doigt en classe pour lui poser une question. Quoi de plus indiqué qu'un lever du doigt pour interrompre le processus d'exécution normal de l'outil afin de préciser certains concepts, indiquer des éléments clés de l'interface ou s'assurer de la compréhension de l'apprenant ? Celui-ci est surtout utile lorsque l'outil d'apprentissage est utilisé individuellement à distance par les apprenants.

Nous avons jugé ces deux métaphores particulièrement adaptées et utiles à l'apprentissage des étudiants :

- la première, la « Métaphore du Tableau Noir », en ce qu'elle permet aux étudiants de retrouver la même approche des algorithmes que celle donnée magistralement en classe par le professeur ;
- et la seconde, la « Métaphore du Lever du Doigt », en ce qu'elle apporte assistance à l'apprenant au fur et à mesure de son cheminement dans l'outil.

Le « Tableau Noir » est concrétisé par la juxtaposition à l'écran de toutes les données concernées par l'exécution d'un algorithme choisi, tout comme le professeur les aurait juxtaposées sur son tableau lors d'un cours présentiel.

Le « Lever du Doigt » est concrétisé par une petite icône représentant un lever de doigt.

[16] Bien que nous avons constaté qu'à l'université, la majorité des tableaux sont blancs.

6.3.9. Conception ergonomique de l'interface

Pour la conception de l'interface de l'outil d'apprentissage à l'algorithmique, nous avons respecté au maximum les règles ergonomiques proposées par Jean Vanderdonckt [VAN93, VAN99].

A titre d'illustration, voici quelques règles que nous avons trouvé appropriées pour concevoir notre outil le plus ergonomique possible :

[VAN99 - 1.22]	Assurer une certaine flexibilité linguistique
[VAN99 - 2.1]	L'accès à l'information doit être rapide
[VAN99 - 3.1]	L'information la plus importante doit être placée en premier lieu
[VAN99 - 3.5]	L'information doit être complète sur le même écran
[VAN99 - 3.14]	Les informations liées sémantiquement doivent être présentées conjointement
[VAN99 - 5.5]	Veiller à la taille des cadres et fenêtres
[VAN99 - 6.1]	Utiliser les graphiques de manière appropriée
[VAN99 - 7.2]	Préférer les fonds d'écran clairs, de basse intensité
[VAN99 - 7.3]	Éviter les fonds d'écran avec motifs
[VAN99 - 7.14]	Utiliser les polices sans sérif pour la lecture en ligne
[VAN93 - 2.3]	Contrôle du (multi-)fenêtrage
[VAN93 - 6.1.5.1]	Localisation des objets de contrôle et de leurs libellés
[VAN93 - 6.1.5.2]	Localisation des boutons de commande
[VAN93 - 6.1.5.7]	Localisation des champs d'édition
[VAN93 - 6.1.5.9]	Localisation des listes de sélection
[VAN93 - 6.1.6.2]	Localisation des boîtes de dialogue
[VAN93 - 6.2.3]	Dimensionnement des objets de défilement

6.4. Interface de l'outil de création d'algorithmes

L'outil de création d'algorithmes est destiné au concepteur de contenu, qui a des compétences certaines en informatique.

Nous n'avons dés lors pas besoin d'une interface très sophistiquée pour lui permettre d'ajouter des algorithmes et des invariants dans l'outil.

L'outil sera donc accessible par une simple commande en ligne à exécuter manuellement.

Cette commande sera un appel à un fichier de commande en mode batch.

L'importance de cette interface est essentiellement de produire une aide appropriée et des messages d'erreur explicites pour permettre au concepteur de contenu de corriger son algorithme et son invariant le cas échéant.

6.5. Interface de l'outil de traduction

L'outil de traduction ne nécessite pas non plus d'interface très sophistiquée. Mais elle n'est pas à priori destinée à des utilisateurs informaticiens.

Elle se composera dès lors d'une interface graphique mettant en parallèle le vocabulaire de l'outil en langue source (langue française en l'occurrence) et la traduction à réaliser en langue cible, ainsi qu'un champ permettant de nommer cette nouvelle langue.

Une langue pourra également être supprimée pour autant qu'elle ne soit pas une des langues de base de l'outil (français, néerlandais, allemand, anglais).

Chapitre 7

Analyse technique : architecture et modélisation

7.1. Description générale de l'architecture de l'outil

7.1.1. Etude préliminaire : Normes et architectures existantes

L'industrie du logiciel s'intéresse depuis plusieurs années au domaine des technologies éducatives et est demandeuse d'une définition de standards qui faciliteraient la compatibilité de développement d'applications éducatives.

Il existe divers groupes de travail dont l'action est d'élaborer des spécifications et des protocoles qui sont proposés à l'industrie du logiciel éducatif.

Citons, à titre d'exemples, les consortiums non européens, IEEE Learning Technology Standards Committee (LTSC)[17], IMS Global Learning Consortium[18], AICC Aviation Industry CBT Committee[19], Advanced Distributed Learning[20]. En Europe, les groupes ARIADNE[21] et PROMETEUS[22] sont les principaux acteurs.

Il ressort de ces travaux une certaine constance :

- La préférence va aux architectures de type client-serveur et ce, dans la mesure où le déploiement des outils éducatifs se fait généralement via Internet. Toutefois, afin de dépasser les limites du langage HTML trop orienté vers la présentation de contenu, le langage XML semble être suggéré comme support de transfert de contenu entre le serveur web et les clients, postes de travail des étudiants.
- Une meilleure réutilisabilité des contenus éducatifs est recherchée. L'idée est d'extraire de ces contenus deux types d'éléments :
  - Les aspects liés à l'interface hommes-machine
  - Les éléments de navigation qui définissent l'orientation proposée aux étudiants d'un contenu éducatif à l'autre.

[17] http://ltsc.ieee.org/
[18] http://www.imsproject.org/
[19] http://www.aicc.org/
[20] http://www.adlnet.org/index.cfm?fuseaction=scormabt
[21] http://www.ariadne.ac.uk/
[22] http://www.prometeus.org/

7.1.2. Choix techniques

7.1.2.1. Architecture client-serveur

De nombreuses applications fonctionnent selon un environnement client/serveur, cela signifie que des machines clientes (des machines faisant partie du réseau) contactent un serveur, une machine généralement très puissante en terme de capacités d'entrée-sortie, qui leur fournit des services. Ces services sont des programmes fournissant des données.

- Avantages de l'architecture client-serveur pour notre outil

L'architecture client-serveur offre des ressources centralisées. Puisque le serveur est au centre du réseau, il peut offrir le même outil à un grand nombre d'utilisateurs.

L'architecture client-serveur donne à notre outil une facilité d'évolution puisque les adaptations se font au niveau du serveur et pas des clients. Un nouveau prototype d'outil peut être mis en production de façon tout à fait transparente pour les utilisateurs.

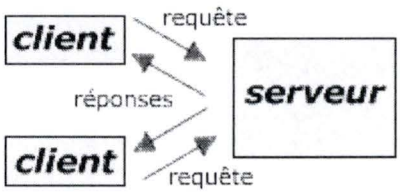
Les données d'un serveur sont généralement intégrées dans un processus de sauvegardes[23] régulières.

- Inconvénient de l'architecture client-serveur pour notre outil

Le serveur est le seul maillon faible du réseau client-serveur, étant donné que tout le réseau est construit autour de lui ! Heureusement, le serveur a généralement une grande tolérance aux pannes (notamment grâce au système RAID).

- Fonctionnement d'un système client/serveur

Un système client/serveur fonctionne selon le schéma suivant :



7.1.2.2. Programmation objet

La taille des applications informatiques ne cesse de croître. Toujours plus complexes et évolutives, elles font l'objet de développements réalisés par des équipes souvent différentes au fil du temps. Ce mode de conception, réalisation et de maintenance est qualifié de modulaire.

[23] Backup

La programmation orientée objets, introduite dès les années septante, est un des moyens imaginés par les informaticiens pour y répondre. Contrairement à la programmation traditionnelle (qui consiste à définir des données d'une part et les fonctions à leur appliquer d'autre part), données et fonctions sont ici associées dans une classe, susceptible d'être déclinée en objets lorsque des données spécifiques lui sont fournies. En outre, les classes peuvent être étendues avec de nouvelles données ou spécialisées par de nouvelles fonctions. Cette dernière notion, particulièrement délicate à mettre en œuvre, est essentielle pour la modularité.
Cette méthode de programmation renforce la cohérence de l'application et simplifie, de façon appréciable, sa maintenance et surtout son évolution.

La programmation objet semble donc tout à fait adaptée à notre conception par prototypage évolutif.

7.1.2.3. Le langage de développement

Nous avons choisi de développer notre outil en langage Java[24]. Un certain nombre de critères ont été envisagés pour réaliser ce choix :

- Le caractère Open Source du langage

Depuis peu, le statut de Java s'est clarifié et ce langage est maintenant clairement Open Source. Cette liberté lui assure une grande pérennité puisque qu'il est possible de disposer des sources et d'effectuer des modifications pour corriger des bugs ou bien améliorer les performances du langage. Par exemple, un groupe de développeurs peut développer un compilateur plus performant et cela de manière indépendante du projet principal.

- L'envergure du projet

C'est un des critères les plus décisifs, car la principale difficulté d'un projet de développement en gestion résulte dans le volume du code à gérer, lors du développement et surtout lors du perfectionnement des futurs prototypes de notre outil.

Java est un langage qui incite à produire du code lisible et aisé à maintenir. Sur ce point, le caractère objet du langage Java le rend effectivement adapté à un développement susceptible d'évoluer et de se perfectionner. Il permet, en outre, une bonne division du travail entre les développeurs et entre les versions de prototypes grâce au principe de l' encapsulation.

[24] http://java.sun.com/

- Le caractère portable

Une application Java présente l'avantage d'être portable sur toutes les plates-formes informatiques majeures : MS-Windows, UNIX OSF/Motif, Mac OS7/8, OS/2, Linux, Solaris, etc.

- La facilité d'utilisation

Java est facile à interfacer au contenu web et offre plusieurs fonctionnalités qui simplifie le développement d'animation. Java est un langage orienté objet. Sa syntaxe ressemble à celle du C++, tout en apportant d'appréciables améliorations, notamment en ce qui concerne la gestion de la mémoire. Elle dispose en outre d'une librairie d'objets abondante et d'une vaste documentation.

Par ces caractéristiques, nous constatons que Java se prête fort bien à une exploitation sur le réseau Internet. L'une des nombreuses possibilités est de créer des applets, petits programmes intégrés dans une page web. L'applet sera exécutée en vase clos, sans danger pour l'ordinateur hôte.

Sun assure la pérennité du langage Java en ne cessant de le développer et de lui adjoindre d'autres technologies

- Unicode

Java gère correctement les caractères Unicode, ce qui nous intéresse dans le cadre de la logique des prédicats du premier ordre.

- Qualité d'interaction

Lorsqu'on envisage le développement d'un outil pédagogique interactif multimédia, on s'attend à retrouver la même qualité d'interaction que celle utilisée sur Internet.

- Multi-tâches

Avec la classe Thread, Java offre au programmeur la possibilité de programmer en multi-tâches. Ce qui est un gros avantage pour développer notre outil qui comporte différentes animations à exécuter parallèlement : animation du langage algorithmique, animation des données en mémoire, animation de l'organigramme, animation des entrées/sorties, animation de l'invariant.

7.1.2.4. Applet

Notre outil est livré sous forme d'une applet, c'est-à-dire directement dans une page web.

Cette applet sera téléchargée sur le poste de l'utilisateur et sera exécutée au sein d'une page html, grâce à la Java Virtual Machine

(JVM) qui décode le programme Java et le traduit dans le langage de la machine hôte.

La JVM est présente dans la dernière mouture du navigateur Netscape (version 7.02) mais pas dans celle d'Internet Explorer (version 6). Nous citons ces navigateurs vu qu'ils sont les plus utilisés actuellement.

Il est d'usage aux Facultés de Namur d'encourager les étudiants à utiliser le navigateur « Mozilla ». Le « Java Plug-In » conçu pour Netscape convient parfaitement.

Pour pouvoir visualiser correctement l'applet, il est possible de télécharger le dernier « Java Plug-In » à l'adresse suivante :

http://java.sun.com/j2se/1.4.1/download.html en choissant 'Download J2SE (TM) v 1.4.1 – JRE' et de suivre les instructions pour son installation selon le système d'exploitation :

Windows : http://java.sun.com/j2se/1.4.1/jre/install-windows.html,
Linux : http://java.sun.com/j2se/1.4.1/jre/install-linux.html
Macintosh : http://www.apple.com/java/
Solaris : http://www.java.com/en/download/help/solaris_install.jsp

- Sécurité des applets

Afin d'assurer la sécurité de l'utilisateur, les possibilités des applets sont limitées :

- Une applet ne peut pas écrire de fichiers sur la machine du client
- Une applet ne peut communiquer qu'avec la machine serveur
- Une applet ne peut pas lancer de programmes sur le client
- Une applet ne peut pas charger de programmes écrits en langage natif sur le client

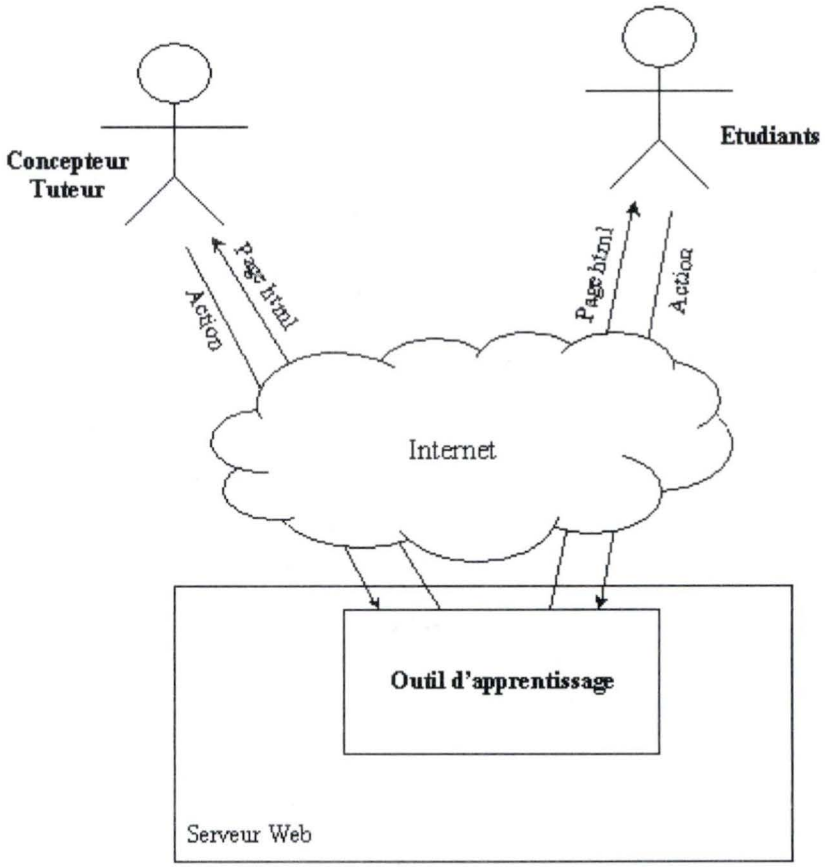


Fig. 11 - Architecture

7.2. Construction de traducteurs

7.2.1. Fonctionnement d'un traducteur

L'objectif d'un traducteur consiste en la transformation d'un langage source en un langage cible.

Nous devons construire deux traducteurs :

- un traducteur de LSD03 en langage Java
- un traducteur de logique du 1er ordre en langage Java

La traduction obtenue en java pourra alors être compilée afin d'être comprise par l'outil d'apprentissage à l'algorithmique.

L'élaboration d'un traducteur suit la même démarche de conception qu'un compilateur. En effet, la compilation vise également la traduction d'un langage source vers un langage cible. Nous verrons cependant qu'un traducteur est plus simple à réaliser qu'un compilateur.

7.2.1.1. Les phases de la compilation

La compilation se décompose grosso modo en 6 phases : 3 phases d'analyses et 3 phases pour la génération du code. La figure suivante, issue de [AHO00], montre le schéma de fonctionnement d'un compilateur.

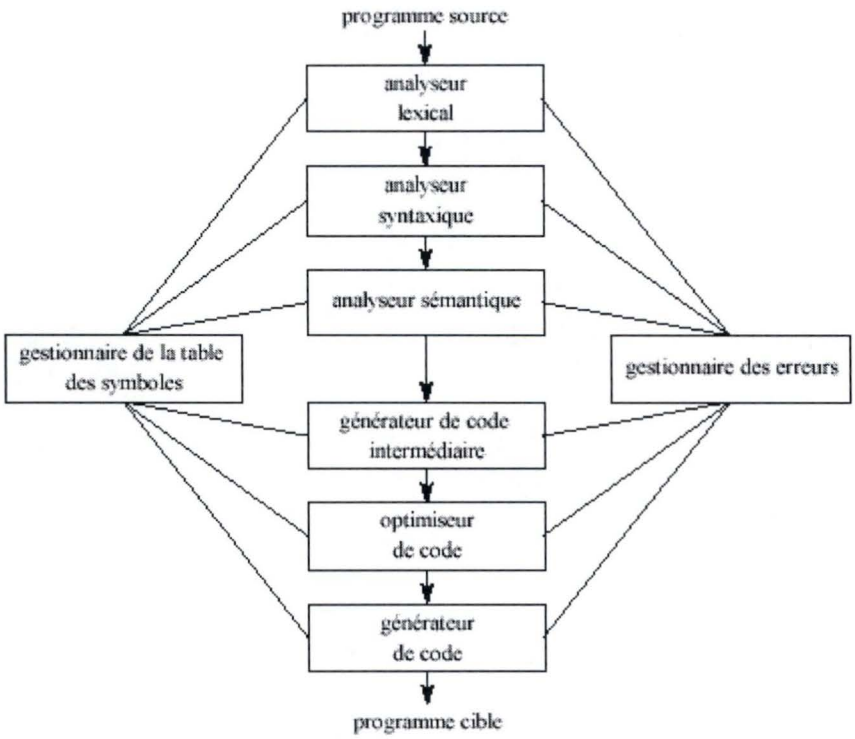


Fig. 12 - Fonctionnement d'un compilateur selon [AHO00]

Analyse lexicale

L'analyse lexicale définit les différents éléments (les lexèmes ou tokens) du langage à interpréter (opérateurs, ponctuation, mots-clés, etc.) et produit un programme écrit dans le langage de destination, typiquement le même langage que celui utilisé pour développer l'outil d'apprentissage. Les expressions régulières permettent de retrouver les variables, valeurs, commentaires, etc. L'analyse lexicale stocke ces lexèmes dans une table des symboles.

Analyse syntaxique

L'analyse syntaxique reconnaît les expressions, instructions, déclarations et corps du programme en entrée. La liste des règles traduisant les contraintes imposées par le langage à interpréter est appelée grammaire. L'analyseur syntaxique prend en paramètre cette grammaire et vérifie si une séquence de lexèmes (tokens) satisfait aux règles de la grammaire et indique s'il y a une erreur de syntaxe. L'analyse syntaxique produit un arbre syntaxique abstrait.

La gestion de la table des symboles

La table des symboles regroupe tous les symboles définis dans le programme. Elle conserve également les propriétés importantes de ces symboles. Par exemple, pour une variable, on retiendra son type, sa portée, son emplacement mémoire. La table des symboles est utilisée tout au long de la compilation, depuis son remplissage par l'analyseur lexical, jusqu'à la génération de code. De par son utilisation importante, les algorithmes de consultation et de mise à jour de la table doivent être optimisés le plus possible.

Analyse sémantique

L'analyseur syntaxique décrit précédemment sera utilisé pour fournir des structures syntaxiques utilisées pour l'analyse sémantique c'est-à-dire la table des symboles ainsi que l'arbre syntaxique abstrait. L'analyse sémantique consiste à associer leur sens aux différentes phrases du texte source de l'algorithme.

L'analyse sémantique permet notamment de déterminer :
- si une variable a bien été déclarée avant d'être utilisée,
- si une variable a été déclarée plus d'une fois,
- si un appel à une fonction possède le bon nombre d'arguments, ...

La gestion des erreurs

La gestion des erreurs est une partie importante d'un compilateur. Une bonne gestion des erreurs permet de détecter un maximum d'erreurs en même temps, ce qui permet de toutes les corriger avant de réessayer de compiler.

La production du code intermédiaire

Il s'agit ici de produire un programme équivalent au code source dans un langage plus proche du langage machine. En général, il utilisera un adressage symbolique et un set d'instructions réduit et standard. La mémoire nécessaire pour stocker les variables du programme sera également calculée. Le but de cette étape est d'effectuer un certain nombre de choix (mémoire à prévoir, type d'instructions à utiliser, ...) dans un langage indépendant du langage cible. Ceci permet une portabilité plus grande des quatre premières phases du compilateur.

L'optimisation

L'optimisation est probablement la plus grosse partie du compilateur, et la plus compliquée. L'optimisation peut se faire soit sur la taille du code, soit sur la mémoire utilisée par le programme, soit encore sur le temps d'exécution. Il faut remarquer que ces critères sont souvent mutuellement exclusifs.

Génération de code

Cette phase produit le code final équivalent au programme de départ. Elle dépend fortement du langage de destination.

7.2.1.2. Les phases de conception d'un traducteur

Nos traducteurs sont plus simples car ils ne disposent pas de production de code intermédiaire, ni de phase d'optimisation. En effet, le but de nos traducteurs est de fournir un fichier java à compiler. C'est donc en les soumettant au compilateur java que les tâches d'allocation de mémoire et d'optimisation s'exécuteront. Notre outil n'étant pas un outil critique en terme de rapidité d'exécution, l'optimisation de nos traducteurs n'est pas une priorité.

Dès lors, la conception de notre traducteur passe par les quatre phases suivantes :

- Analyse lexicale
- analyse syntaxique
- analyse sémantique
- génération de code

Cela nous pousse à la conclusion que la dénomination de « générateur de code java » serait plus appropriée que celle de « traducteur ».

7.2.2. Choix des outils

Plusieurs outils de création de compilateur existent. La principale exigence qui doit nous guider dans le choix de ces outils est de générer du code java.

Nous avons retenu de nos recherches les outils JLex et CUP. Ceux-ci sont très proches des outils « Flex » et « Bison » que nous avons utilisés dans le cadre des travaux pratiques illustrant le cours de Syntaxe et Sémantique suivi en 1ère licence [SCH01]. Nous avons estimé que le choix de ces outils nous permettait d'optimiser le temps pour la partie implémentation de notre travail.

JLex et CUP diffèrent de Flex et Bison en ce qu'ils génèrent du code java et non du code C.

JLex[25] est un générateur d'analyseurs lexicaux écrit en java pour le langage Java. JLex utilise un fichier en entrée contenant la spécification d'un analyseur lexical, et il génère un code source en langage Java qui doit être compilé pour obtenir l'analyseur lexical.

CUP[26] est un outil qui permet de générer des analyseurs syntaxiques de type LALR. Il écrit en Java. Il utilise un fichier de spécification contenant du code Java et une grammaire permettant de générer l'analyseur syntaxique

[25] http://www.cs.princeton.edu/~appel/modern/java/JLex/
[26] http://www.cs.princeton.edu/~appel/modern/java/CUP/

correspondant en langage Java. CUP a besoin d'unités lexicales « tokens » qui sont des suites de caractères en provenance d'un scanner, qui peut être un analyseur lexical généré par JLex.

7.2.3. Fonctionnement des générateurs de code java

Les deux générateurs de code java pour le LSD^{03} et la logique des prédicats suivent le même principe de conception et de raisonnement. Seules diffèrent les spécifications des lexèmes et de la grammaire. Nous nous contentons dès lors d'expliquer le fonctionnement de générateur de code java conçu pour le langage LSD^{03}.

Pour réaliser ce générateur de code, nous nous sommes inspiré du travail réalisé par Croft [CRO99].

7.2.3.1. Analyse Lexicale

Nous avons défini les lexèmes du langage LSD^{03}. Le lecteur peut consulter le fichier de spécifications en annexe G.

7.2.3.2. Analyse syntaxique

Nous avons ensuite spécifié la grammaire du langage LSD03 en étant particulièrement vigilants à la précédence des opérateurs. L'analyse syntaxique stocke les différentes données dans un arbre syntaxique abstrait. Le lecteur peut consulter le fichier de spécifications en annexe H.

7.2.3.3. Analyse sémantique

Pour mener à bien l'analyse sémantique, nous avons utilisé le « Composite Pattern » [COO98]. Les spécialistes définissent un pattern comme une solution à un problème récurrent de conception.

Le « Composite Pattern » est généralement utilisé pour construire et parcourir des représentations de données avec des arbres ('trees'). Selon [COO98], *« un « composite » est une collection d'objets, dont chacun peut être un « composite » ou un objet primitif. Dans la nomenclature des arbres, certains objets peuvent être des nœuds avec des branches additionnelles et d'autres, des feuilles. »*

L'analyse sémantique consiste en le parcours de l'arbre syntaxique abstrait créé lors de l'analyse syntaxique afin de détecter d'éventuelles erreurs comme un appel à une procédure avec un nombre incorrect d'arguments, par exemple.

Grâce au « Composite Pattern », la sémantique peut être vérifiée dans l'arbre entier en propageant l'appel à une méthode de contrôle de sémantique à chacun des nœuds fils.

Chaque objet doit pouvoir accéder à son ancêtre dans l'arbre pour pouvoir propager le contrôle de sémantique.

Ainsi, pour contrôler si une variable a bien été déclarée avant son emploi, la méthode de contrôle de sémantique doit traverser la lignée pour voir si une telle déclaration a bien lieu au commencement.

L'utilisation du « Composite Pattern » permet d'alléger le code source et permet d'ajouter des types de nœuds sans modifier les classes qui utilisent la méthode de contrôle de sémantique.

7.2.3.4. Génération de code

Deux modèles de conception sont utilisés pour générer le code java à partir de la représentation intermédiaire modélisée par l'arbre syntaxique abstrait. Il s'agit du « Composite Pattern » et du « Visitor Pattern » [COO98].

Selon [COO98], « *le « Visitor Pattern» renverse le rôle du modèle orienté objet et crée une classe externe pour traiter les données dans d'autres classes. C'est utile s'il y a un petit nombre d'instances ou un nombre restreint de classes et que l'on désire effectuer une certaine opération qui les fait toutes participer ou la plupart d'entre elles.* »

Chaque nœud de l'arbre implémente une méthode de génération. Lorsque cette méthode est appelée, le nœud génère du code pour lui-même et ses descendants. Dans la méthode de génération du nœud parent, des appels sont faits à la méthode de génération de ses nœuds enfants selon le modèle de conception « Composite Pattern » dont nous avons parlé au point précédent.

Afin de permettre la traduction de l'arbre syntaxique abstrait vers le langage java, sans devoir recompiler les classes de l'arbre, nous exploitons le modèle de conception « Visitor Pattern ».

La méthode de génération comporte un argument qui implémente une interface « Visitor ». Cette interface définit les méthodes qui sont capables de générer du code pour chacun des types de nœuds de l'arbre.

Cet argument passé à chacune des méthodes est une instance du type de nœud indiqué qui contient les données et les méthodes d'accès exigées pour produire du code.

Ainsi conformément au modèle de conception « Visitor Pattern », chaque fois que la méthode de génération de code d'un nœud est appelée, elle appelle à son tour la méthode appropriée de génération de code à l'interface « Visitor » avec lui-même en argument.

L'utilisation du « Visitor Pattern » permet donc d'éviter des modifications dans les classes de nœuds de l'arbre pour la génération de code java ou éventuellement pour la génération d'un autre langage dans le futur ou pour une autre plate-forme.

7.2.3.5. Synthèse

Le schéma de la page suivante illustre le générateur de code Java pour LSD^{03}. Le gestionnaire de la table des symboles et le gestionnaire des erreurs ont été volontairement omis pour ne pas alourdir le schéma. Ces deux structures se retrouvent clairement illustrées dans la figure 12 – Fonctionnement d'un compilateur selon [AHO00].

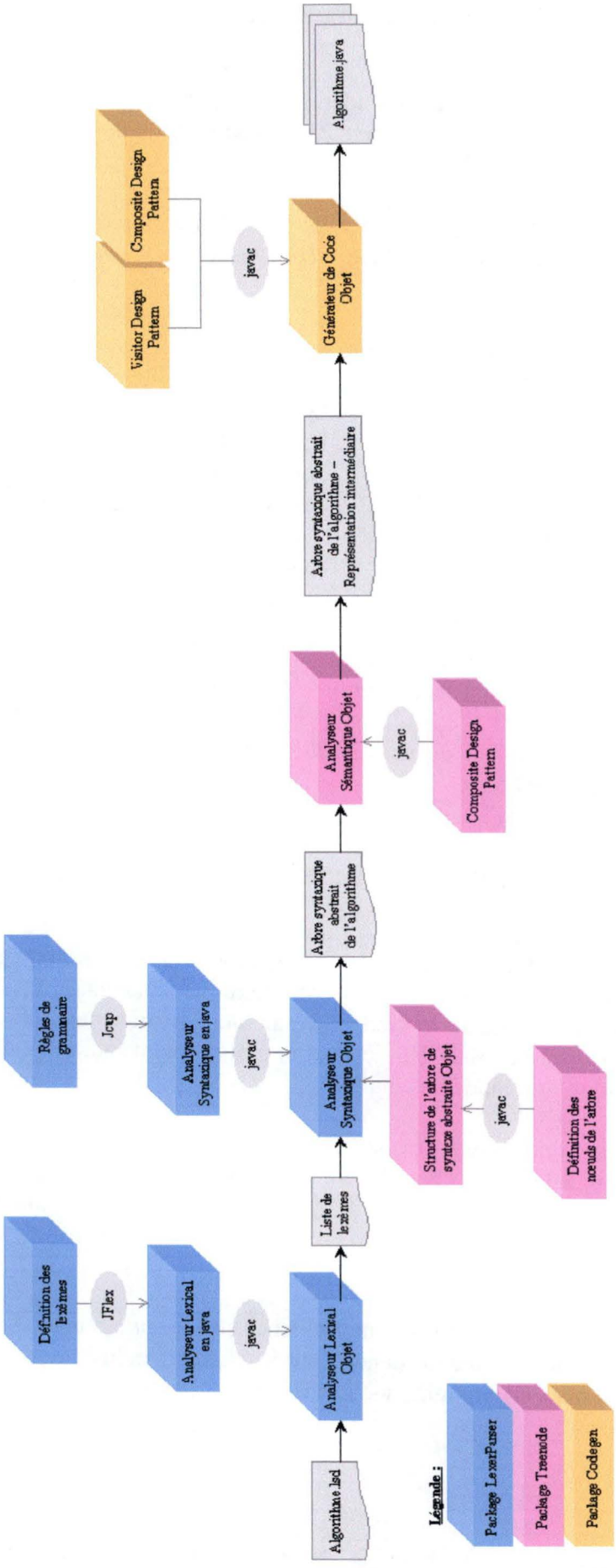


Fig. 13 - Générateur de code java pour le LSD[03]

7.3. Construction de l'interface hommes-machine

Au point 6.3.6.3, nous avons identifié les Objets Interactifs Abstraits (OIA) de l'interface utilisateur. Il convient maintenant de transformer ces OIA en Objets Interactif Concrets (OIC).

A chaque OIA sélectionné dans l'étape précédente, nous faisons correspondre les OIC propres à l'environnement physique retenu. Il s'agit, dans notre cas, du package Swing de Java.

Composants Swing utilisés par notre interface :

Fenêtre	:	JApplet
Onglet	:	JTabbedPane
Liste de choix	:	JComboBox
Champ de texte multilinéaire	:	JTextArea
Bouton	:	JButton
Panneau	:	JPanel
Champ graphique	:	JCanvas

A ces OIC, s'ajoutent d'autres éléments comme notamment des Jlabels (libellé) ou des JscrollPane (panneaux à ascenceurs).

Pour le positionnement de ces Objets Interactifs Concrets, nous invitons le lecteur à consulter l'annexe C.

7.4. Problématique des systèmes d'animation d'algorithmes

Les systèmes d'animation d'algorithmes ont pour finalité de permettre la spécification, la visualisation et l'animation de représentations graphiques d'algorithmes. De ce fait, ces systèmes ne prennent pas en considération le langage de programmation utilisé, même si ce choix affecte le système résultant. La principale difficulté réside donc dans la méthode utilisée pour lier le programme implémentant un algorithme choisi et la représentation graphique de celui-ci.

La solution que nous proposons est de définir pour chaque type d'instruction (déclaration, affectation, boucle, etc.) une méthode permettant au système de connaître l'animation qu'il doit exécuter. L'ajout de la méthode adéquate se réalise lors de la génération de code java.

A titre d'illustration :
Toutes les instructions auront une incidence sur l'organigramme.
Une affectation aura une incidence[27] sur les données en mémoire.
Par contre, une écriture aura une incidence sur la console.

[27] Même si celle-ci ne sera pas toujours visible. Par exemple : procéder consécutivement à deux affectations identiques. L'utilisateur aura l'impression que les données en mémoire sont inchangées. Pourtant, il s'agit bien d'une mise à jour.

Avantages de ce procédé de couplage « algorithme - représentation graphique »:

Ce procédé :

- ne demande pas la réécriture de l'algorithme,
- permet la réutilisation des représentations graphiques pour d'autres algorithmes,
- offre une trace et permet donc la constitution d'un historique de l'exécution de l'algorithme,
- permet une décomposition visuelle des opérations.

Conclusions

Notre volonté d'indépendance par rapport à un langage de programmation et notre attachement à construire des représentations graphiques montrant les différentes opérations effectuées par un algorithme sont atteints.

Chapitre 8

Production : codage

Une fois établies la portée de la matière et son approche pédagogique, nous avons entrepris la réalisation d'un prototype afin de valider l'interface graphique. Un soin tout particulier est apporté pour l'élaboration du code. Tout au long de la vie du logiciel, le code va évoluer.

Une deuxième phase de mise au point pourra s'avérer nécessaire après un premier essai d'utilisation auprès d'élèves. Des améliorations pourront venir enrichir le logiciel ou alors, tout simplement, il faudra adapter le code à l'évolution des ressources informatiques.

Pour pouvoir permettre relectures et modifications avec le plus de facilité et de sécurité, le code doit être clair avec une certaine unité. Un bon découpage en modules, le développement d'outils de programmation correctement documentés et d'utilisation simple, l'usage systématique de noms de variables symboliques, des commentaires dans le code, permettent ce type de programmation. Nous avons adopté ces consignes.

8.1. Problèmes techniques inhérents à l'architecture choisie

La guerre que se livrent les navigateurs Netscape et Internet Explorer pour s'imposer comme ressource de navigation sur Internet apparaît très clairement lorsqu'on souhaite développer une application compatible pour les deux solutions. Chacun reconnaît et interprète Java à sa manière et on doit toujours s'attendre à des surprises en visionnant l'applet avec l'un ou l'autre.

La dernière version du Plug-In Java utilisée conjointement avec la dernière version de Netscape ou d'Internet Explorer n'a pas posé de problèmes majeurs (cf. point 7.1.2.4 pour les versions et les adresses de téléchargement de ces logiciels).

Le rafraîchissement graphique pose parfois problème mais nous n'avons pas encore pu identifier s'il s'agissait d'un problème lié au Plug-In, aux navigateurs ou à certaines ressources systèmes.

8.2. La construction de l'interface utilisateur

La construction de l'interface utilisateur n'a pas posé de problèmes majeurs. La principale difficulté a résidé dans l'apprivoisement des gestionnaires de positionnement des composants graphiques de Java Swing: les « Layout Manager ».

8.3. L'implémentation de l'outil de création d'algorithmes

L'implémentation du « Module de création d'algorithmes » a coûté énormément de temps et a nécessité la maîtrise de modèles[28] de conception en Java. L'apprentissage de ces modèles de conception a nécessité un effort important mais une fois maîtrisés, ils ont contribué à l'écriture d'un code clair.

Nous avons rencontré une difficulté importante dans la traduction de code LSD^{03} en langage Java au niveau de l'implémentation des procédures. Nous avons du tenir compte du fait que Java permet :

- le passage de paramètres par valeur pour tous les types de base c'est-à-dire tout objet créé par une simple déclaration
- et le passage de paramètres par adresse (ou référence) pour tous les autres objets dynamiques créés avec le constructeur new.

L'implémentation du « Module de création d'invariants » est similaire à celle du « Module de création d'algorithme », ce pourquoi nous ne développons pas plus avant.

8.4. L'implémentation de l'outil de traduction

L'implémentation du « Module Création de Langue » n'a pas posé de difficulté majeure. Il s'agissait de faire appel aux méthodes adéquates d'Entrée/Sorties de Java et de s'assurer que l'utilisateur a un accès en écriture sur les fichiers concernés par le stockage des traductions. Il fallait en outre s'assurer que l'utilisateur ne puisse pas effacer les traductions de base de l'outil (français, néerlandais, allemand et anglais).

8.5. L'implémentation de l'outil d'apprentissage à l'algorithmique

8.5.1. Module d'affichage de la langue

La classe java « ResourceBundle » nous a permis de gérer efficacement le changement dynamique de la langue de l'interface hommes-machine. Cela n'a pas posé de problème particulier.

8.5.2. Module d'affichage de l'algorithme

L'affichage d'un algorithme ainsi que son animation, à savoir la mise en évidence de l'instruction courante par l'utilisation d'une couleur n'a pas été toute simple. Cela a nécessité un approfondissement des classes « DefaultTableCellRenderer » et « DefaultTableModel ».

8.5.3. Module d'affichage de l'état de la mémoire

Le module d'affichage des données en mémoire pendant l'exécution de l'algorithme a nécessité la récupération des données issues de la constitution d'un historique[29] créé lors de la création de celui-ci via l'outil de création d'algorithmes. Ce module n'a pas posé de problèmes majeurs.

---

[28] Design Pattern.
[29] Cf. point 7.4.

8.5.4. Module d'affichage de la console

L'implémentation du « Module d'affichage de la console » n'a pas posé de difficulté particulière. Il s'agissait de faire appel aux méthodes adéquates d'Entrée/Sorties de Java.

8.5.5. Module d'affichage de l'organigramme

Pour la construction de l'organigramme, nous avons du garder à l'esprit que des boucles ou des choix conditionnels pouvaient se trouver imbriqués. Nous avons également rencontré de grosses difficultés lors du rafraîchissement de l'organigramme.

8.5.6. Module d'affichage de l'Historique

L'implémentation du « Module d'affichage de l'historique » n'a pas posé de difficulté particulière. Il s'agissait de faire appel aux méthodes adéquates d'Entrée/Sorties de Java.

8.5.7. Module d'affichage de l'aide

L'implémentation du « Module d'affichage de l'aide » n'a pas posé de difficulté particulière. Il s'agissait de faire appel aux méthodes adéquates d'Entrée/Sorties de Java.

8.5.8. Module d'affichage de l'information relative à l'outil

L'implémentation du « Module d'affichage de l'information » n'a pas posé de difficulté particulière. Il s'agissait de faire appel aux méthodes adéquates d'entrées de Java.

8.6. Gestion des bases de données

Afin de répondre à la demande explicite de notre client[30], nous stockons toutes les données liées au contenu pédagogique (algorithme, invariant, descriptif d'algorithme, traduction, historique des opérations, ...) dans des fichiers plats afin de lui en faciliter l'accès.

8.7. Conclusion

L'implémentation de notre outil est la partie du mémoire qui nous a coûté le plus en terme de temps et d'efforts. Ce constat provient principalement du fait que nous ne maîtrisions pas le langage Java au commencement de notre travail.

[30] Cf. point 2.2.2.

Chapitre 9

Le déploiement du logiciel

Le déploiement est la phase finale du développement d'une applet. C'est une étape indispensable pour la mise en place de l'outil.

9.1. Les pré-requis

Poste serveur :

- Windows XP, Windows 2000, Windows NT, Windows 9X
- Pentium II processeur 500 Mhz minimum
- Moniteur SVGA
- 128 Mb RAM minimum
- JRE ou SDK version 1.4.1_01
- Serveur web : Apache, IIS, Netscape Enterprise Server, ...

Poste client

- Windows XP, Windows 2000, Windows NT, Windows 9X, Solaris, Linux.
- Netscape 7 ou supérieur
- Internet Explorer 6 ou supérieur
- Mozilla 1.3.1 ou supérieur

9.2. Préparation au déploiement

Dans un souci de restreindre le temps de chargement, nous livrons notre outil sous forme d'un fichier JAR (Java Archive). Il s'agit d'une compression dans un fichier signé téléchargeable en une seule transaction réseau de toutes les ressources d'une applet.

Nous livrons également une page html contenant l'appel à l'applet.

9.3. Installation des fichiers sur le serveur web

Les fichiers décrits sous la rubrique précédente doivent être copiés sur le serveur Web. Ainsi pour lancer l'applet, l'utilisateur (poste client) se connecte au serveur web via son navigateur et charge la page HTML correspondante.

9.4. Configuration du Plug-In Java (version 1.4.1)

Ce Plug-In Java est un utilitaire qui exécute l'applet en utilisant le « Java Runtime Environment (JRE) 1.4.1 » à la place de la machine virtuelle par défaut du navigateur.

9.5. Conclusion

Une applet entraîne un coût de déploiement et de mise à jour minimal

Chapitre 10

Utilisation du produit

10.1. Intégration et évaluation du prototype

L'outil que nous avons développé n'a pas encore fait l'objet d'une évaluation par les utilisateurs principaux, à savoir les étudiants. Nous avons cependant estimé qu'une bonne démarche d'intégration de l'outil comportait quatre phases essentielles.

Durant la phase de préparation, l'enseignant devrait :

- communiquer clairement les objectifs d'apprentissage poursuivis,
- planifier l'emploi du temps,
- voir à la mise en place des moyens logistiques requis.

Durant le déroulement de la simulation, l'enseignant devrait veiller à maintenir certains équilibres :

- diriger l'activité en faisant preuve de souplesse,
- doser les aspects récréatifs et éducatifs de l'activité,
- donner un certain tempo sans toutefois stresser les participants,
- canaliser l'esprit de compétition et encourager la collaboration,
- laisser la place aux participants tout en étant partout à la fois,
- voir à ce que la satisfaction des besoins du groupe laisse de la place à la satisfaction des besoins individuels.

Durant la phase de révision :

- les participants doivent pouvoir exprimer ce qu'ils ont rencontré comme difficultés,
- le déroulement de la simulation devrait être rappelé, de façon aussi objective que possible,
- les liens entre la simulation et la réalité devraient être explicités en fonction des objectifs d'apprentissage,
- l'enseignant doit encourager les étudiants à formuler les théories, modèles ou principes que la manipulation de l'outil a permis d'illustrer et élaborer davantage sur la réalité simulée.

Pendant la phase d'évaluation, l'enseignant devrait demander aux étudiants d'évaluer l'outil.

10.2. Cas d'utilisation

Plusieurs cas d'utilisation de l'outil peuvent être envisagés. Les encadrements que nous avons identifiés sont au nombre de quatre. Ils mettent en œuvre différentes technologies.

- Utilisation par le professeur dans un auditoire :
  - *utiliser le logiciel pour donner le cours* (projection sur écran) : interaction synchrone entre le poste de travail du professeur et ceux des apprenants,
  - *proposer des exercices :* possibilité d'accès par le professeur aux postes des apprenants.

- Utilisation par le professeur à distance :
  - diffusion non interactive
  - interaction asynchrone avec les apprenants (réponses différées)
  - interaction synchrone avec les apprenants
    - ⇒ possibilité pour les apprenants d'adresser des questions pendant l'exposé au professeur (Par exemple, une demande d'intervention. Si celle-ci est accordée, expression de la demande avec possibilité de réponse au demandeur uniquement ou multicast),
    - ⇒ possibilité de visioconférence avec partage de documents de part et d'autre.

- Utilisation par l'apprenant : utilisation off-line pour :
  - apprendre: anticiper une matière ou revoir un contenu non compris en classe,
  - s'auto évaluer sur une matière donnée,
  - dialoguer (coopérer) avec le professeur et avec les autres apprenants.

- Utilisation par l'apprenant : utilisation on-line pour :
  - apprendre: suivre l'exposé ou une démonstration de l'enseignant,
  - dialoguer (coopérer) avec le professeur et avec les autres apprenants.

10.3. Limites du produit

Une des exigences formulées par le client était de rendre les étudiants plus autonomes. Il fallait toutefois veiller à ne pas les rendre dépendants d'un dispositif simplement différent. Les algorithmes tels qu'abordés dans l'outil ne vont peut-être pas offrir une réelle autonomie aux apprenants mais bien leur enseigner des attitudes et des savoirs-faire nouveaux.

Le produit n'a pas été conçu dans une perspective d'auto formation mais d'illustration et de support.

Pour développer un outil plus intégré, il faudrait regrouper des compétences multiples :

1. Un spécialiste du contenu c'est-à-dire le professeur et éventuellement ses assistants afin :
    - d'identifier les objectifs,
    - de concevoir le contenu pédagogique, les activités d'apprentissage et d'évaluation,
    - de définir l'encadrement.

2. Un spécialiste en applications pédagogiques des nouvelles technologies de l'information et de la communication qui aide :
    - à l'identification des objectifs,
    - à la structuration du contenu,
    - au design pédagogique et graphique de l'outil éducatif,
    - à la conception des activités d'apprentissage, des outils interactifs et des évaluations,
    - à la gestion du projet,
    - à l'évaluation et l'implémentation de l'outil éducatif.

3. Une équipe multimédia qui réalise du matériel de qualité adapté pour les sites web : Mise en page, animation.

4. Un informaticien qui programme les écrans, l'interactivité et la gestion de l'outil.

Troisième Partie

Conclusion et perspectives

Chapitre 11

Conclusions et perspectives

11.1. Conclusion

Le point de départ de ce travail était le souhait d'un professeur d'algorithmique de disposer d'un outil permettant la visualisation sous divers modes des phases d'exécution d'un algorithme. Une double perspective d'utilisation était envisagée. Il s'agissait d'une part de permettre l'illustration d'un exposé magistral et d'autre part de fournir un outil manipulable par les étudiants lors de leur phase d'appropriation individuelle de la matière.

Le présent mémoire a exposé les éléments de réflexions qui ont guidé cette tâche et les produits qui en ont résulté. Nous avons commencé par détailler le contexte et les objectifs du produit. Puis nous avons exposé la méthodologie par prototypage et exposé en quoi elle est apparue la plus pertinente pour la réalisation de l'outil.

Les éléments d'analyse et de spécification ont été exposés dans une version résultant de la synthèse des multiples itérations du processus de développement par prototypage.

L'architecture et les principaux éléments du codage ont été présentés.

Enfin, revenant explicitement aux objectifs d'exploitation pédagogique du produit qui n'ont cessé de guider son développement, nous avons resitué les perspectives d'utilisation de l'outil implémenté qui nous apparaissent les plus pertinentes.

De plus, des manuels de l'utilisateur ont été rédigés et sont livrés en annexe du présent document.

La réalisation de ce travail a été une formidable occasion de réaliser une synthèse et un élargissement de nombreux éléments de la formation de ces deux années de licence. La réflexion sur l'algorithmique et les éléments fondamentaux qui la composent ne sont que la pointe de l'iceberg de l'ensemble des connaissances acquises au cours de ce parcours universitaire qui ont été exploitées au cours de cette réalisation.

D'une façon évidente, elle a été l'occasion d'une réflexion sur les méthodologies de développement de logiciels et de gestion de projet. Les outils d'analyse des systèmes d'informations se sont révélés d'une grande utilité ainsi que la démarche de conception de l'architecture et de l'interface hommes-machine. La rédaction d'un générateur de code et l'utilisation du langage de programmation Java ont été l'occasion d'exploiter les

éléments de syntaxe et sémantique avec de nouveaux outils (JFlex et CUP) et d'explorer plus avant le paradigme de programmation orienté objet.

Enfin le souci de rendre l'application disponible via une interface web a été une occasion de mettre en œuvre une application de type client-serveur.

11.2. Perspectives

Le produit dans son état actuel a été réalisé dans les limites de temps imparties à la rédaction d'un mémoire. Il propose un ensemble de quatre algorithmes exploitables. Il gagnera évidemment à être enrichi de nouveaux algorithmes.

Un outil connexe à l'application a été conçu de façon à permettre au professeur d'introduire aisément de nouveaux algorithmes. Un souci essentiel dans les choix qui ont été posés au niveau de l'architecture du logiciel a été de favoriser l'évolutivité du produit. La modularité du code permet d'envisager de nombreuses adaptations et évolutions du produit.

Par exemple, une interface plus intuitive pour l'outil de création d'algorithmes pourrait être ajoutée pour permettre à l'étudiant de soumettre lui-même un nouvel algorithme.

Afin de suivre les avancées des étudiants, il pourrait être envisagé de disposer d'outils permettant une trace dynamique et permanente de toutes les actions effectuées par les étudiants en ligne. Ces traces constitueraient un outil de pilotage du cours et permettraient l'amélioration continue de l'outil et de son contenu.

Des activités d'auto évaluation pourraient être envisagées.

Afin de rendre l'outil davantage compatible avec les travaux de normalisation menés par les consortiums internationaux centrés sur l'E-learning, les contenus pourraient être stockés dans le format XML.

De façon plus large, on peut envisager d'élargir le produit à la possibilité d'ajout à distance de nouveaux algorithmes, nouvelles langues, … Ceci nécessitera notamment une réflexion sur les aspects de sécurité,…

Les possibilités sont vastes qui pourraient – pourquoi pas – donner lieux à de nouveaux travaux…

Glossaire

Algorithme	Spécification d'un schéma de calcul sous forme d'une suite finie d'opérations élémentaires, appelées instructions, obéissant à un enchaînement déterminé.
Convivialité	La convivialité désigne le degré de facilité avec lequel on peut comprendre et utiliser un système.
Distance articulatoire	La distance articulatoire est faible si l'on peut déduire facilement de la forme d'une expression sa signification.
Distance sémantique	La distance sémantique est faible si l'on peut facilement exprimer dans le langage de l'interface ce que l'on veut réaliser et comprendre le résultat.
Ergonomie	Étude de l'interaction entre l'homme et les machines en général.
Golfe d'évaluation	Ecart entre les variables d'état du système et l'évaluation de l'objectif, selon Norman illustré dans [BOD1].
Golfe d'exécution	Ecart entre les objectifs de la personne et les variables d'action du système physique, selon Norman illustré dans [BOD01].
Instruction	Opération élémentaire d'un algorithme.
Invariant	La méthode la plus courante quand il y a des boucles est la récurrence, qui en algorithmique s'appelle méthode de l'invariant. On cherche une propriété vraie à l'entrée de la boucle et encore vraie à la fin.
Langage algorithmique	Compromis entre le langage naturel et un langage de programmation explicitant clairement les opérations élémentaires d'un algorithme.
LSD^{03}	Abréviation de « Langage Simple et Didactique ». Ce langage est décrit dans [SCH03]. Nous l'utilisons comme langage algorithmique.
Moyen d'interaction	Un moyen ou dispositif d'interaction (MDI) est un outil à l'aide duquel l'utilisateur va pouvoir manipuler les Objets interactifs concrets.
Objet interactif abstrait OIA	Un objet interactif abstrait (OIA) représente l'abstraction d'un ensemble d'OIC de même type, et ce, indépendamment des environnements physiques qui l'accueillent.
Objet interactif concret OIC	Un objet interactif concret (OIC) est tout objet visible par l'utilisateur au sein d'un environnement physique donné.
Style d'interaction	Un style d'interaction ou style de dialogue (SDI) définit la manière d'organiser une suite d'aquisitions-restitutions d'information.

Bibliographie

Internet est un outil révolutionnaire car il ouvre une fenêtre sur une profusion d'informations sur tous les sujets possibles et imaginables. Pour réaliser ce mémoire, j'ai amassé un grand nombre d'informations souvent disparates sur une multitude de sites web qu'il serait trop long et vain d'énumérer. Ces informations m'ont aidée à me forger une idée de la direction que devait prendre mon mémoire et son argumentation. J'ai cependant toujours veillé à analyser ces données d'un œil critique et à prendre la distance qui s'impose lorsque je me trouvais face à un contenu issu d'un site web à caractère commercial.

[AHO00] Alfred Aho, Ravi Sethi, Jeffrey Ullman, « Compilateurs. Principes, techniques et outils. Cours et exercices », Dunod, 2000.

[BOD01] François Bodart, cours de « Conception d'Interfaces Hommes-Machine », Facultés Universitaires Notre-Dame de la Paix de Namur, Institut d'Informatique, 2001-2002.

[COO98] James W. Cooper, « The Design Patterns – Java Companion », Addison-Wesley, 1998

[CRO99] David Wallace Croft, « A nested-procedure programming language to Java translator using object-oriented design patterns », http://www.croftsoft.com/portfolio/mini/, 1999

[DIM98] La Didacthèque internationale en management public, Séminaire sur les méthodes d'enseignement, http://www.enap.uquebec.ca/seminaires-nov98/methode/expose_magistral.htm, novembre 1998

[DEV92] Michel Develay, « De l'apprentissage à l'enseignement », p.75, Paris, Editions E.S.F. Edition, 1992.

[DUC95] Charles Duchâteau, « Images pour programmer. Apprendre les concepts de base », De Boeck Université, 1ère édition, 2ème tirage, 1995.

[FEN97] Peter Fenrich, Practical Guidelines for Creating Instructional Multimedia Application », Dryden Press/Harcourt Brace College Publishers, 1997.

[HAB01] Naji Habra, Cours d' « Ingénierie du logiciel », Facultés Universitaires Notre Dame de la Paix de Namur, 2001-2002.

[HOO92] S. Hooper, « Cooperative Learning and Computer Based Instruction », In ETR&D, Volume 40: 21-38, 1992.

[LES01] Roland Lesuisse, Cours de « Gestion de projets informatiques », Facultés Universitaires Notre Dame de la Paix de Namur, 2001-2002.

[NIE93] J. Nielsen, « Usability Engineering », Academic Press, 1993.

[SCH00] Pierre-Yves Schobbens, Cours de « Principes des langages : syntaxe et sémantique des langages de programmation », Facultés Universitaires Notre Dame de la Paix de Namur, 2000-2001.

[SCH03] Pierre-Yves Schobbens, Yves Bontemps, François Schoubben, Isabelle Linden, Sébastien Verbois, « LSD^{O_3} : le retour de la vengeance », Facultés Universitaires Notre-Dame de la Paix de Namur, Institut d'Informatique, 23 janvier 2003.

[SHN92] Ben Shneiderman , « Designing the User Interface : Strategies for Effective Human-Computer Interaction », Addison-Wesley, 1992

[VAN93] Jean Vanderdonckt, « Corpus minimal de règles ergonomiques », 1993

[VAN99] Jean Vanderdonckt, Extrait du syllabus de « Conception ergonomique de pages Web », Institut d'administration et de gestion de l'Université Catholique de Louvain,1999-2000

Référence des outils cités dans l'état de l'art (point 2.3)

[EA1] http://www.dgp.utoronto.ca/people/RMB/rmb.html
[EA2] http://www.cs.brown.edu/memex/HT_87_Keynote_Address.html#BALSA
[EA3] http://www.cs.brown.edu/publications/techreports/reports/CS-89-30.html
[EA4] http://www.cc.gatech.edu/gvu/softviz/parviz/polka.html
[EA5] http://www.cc.gatech.due/gvu/softviz/parviz/samba.html
[EA6] http://loki.cs.brown.edu:8080/pages/
[EA7] http://research.compaq.com/SRC/zeus/home.html
[EA8] http://ls4-www.cs.uni-dortmund.de/RVS/zada.html
[EA9] http://www.dis.uniroma1.it/~demetres/Leonardo/Features.html
[EA10] http://swarm.cs.wustl.edu/pavane.html
[EA11] http://catai.dia.unisa.it/Description.htm
[EA12] http://www.enseignement.polytechnique.fr/profs/informatique/Philippe.Chassignet/MACLIB/index.html
[EA13] http://www.dat.ruc.dk/~keld/algoritmik_e99/Applets/
[EA14] http://www.cs.princeton.edu/~ah/alg_anim/gawain-4.0/
[EA15] http://lieber.www.media.mit.edu/people/lieber/Lieberary/ZStep/ZStep.html
[EA16] http://www.iut-orsay.fr/~fournier/animal.html

Quatrième Partie

Annexes

Annexe A

Article de Presse – synthèse

Le Soir rapporte dans son édition du 29 juin 2003 les réflexions qui ressortent de trois jours de visite intensive du World Education Market (WEM) qui s'est tenu à Lisbonne du 19 au 22 mai : *« Le monde se divise en deux groupes de gens : ceux qui font partie de la société de la connaissance et ceux qui en sont exclus. Le pouvoir économique et politique appartient aux premiers et leur défi est de le garder. Ceux qui en sont exclus risquent de devenir les nouveaux pauvres des temps futurs : les analphabètes du numérique ! »* Et Viviane Reding, la commissaire européenne responsable de l'Éducation et de la Culture, de rappeler le credo européen : *« Il faut briser la séparation traditionnelle entre l'éducation d'un côté et la formation professionnelle de l'autre. Les nouvelles technologies peuvent aider à cette réforme en favorisant les partenariats entre les écoles, universités et d'autres sources de connaissances comme les musées et les industries locales. »*

L'apprentissage en ligne peut combler l'écart traditionnel entre l'apprentissage et la réalité du travail. Les futurs employés qui auront déjà été habitués à cette nouvelle méthode de formation lors de leur cursus universitaire surmonteront d'autant mieux les difficultés inhérentes à tout nouvel outil d'apprentissage en ligne proposé par leur employeur. La formation en ligne rencontre de plus en plus de succès auprès des entreprises car elle permet de se former rapidement sans aucune contrainte de temps, de déplacement et de disponibilité.

Annexe B

Ecrans du prototype horizontal stabilisé

Outil d'apprentissage d'algorithmique

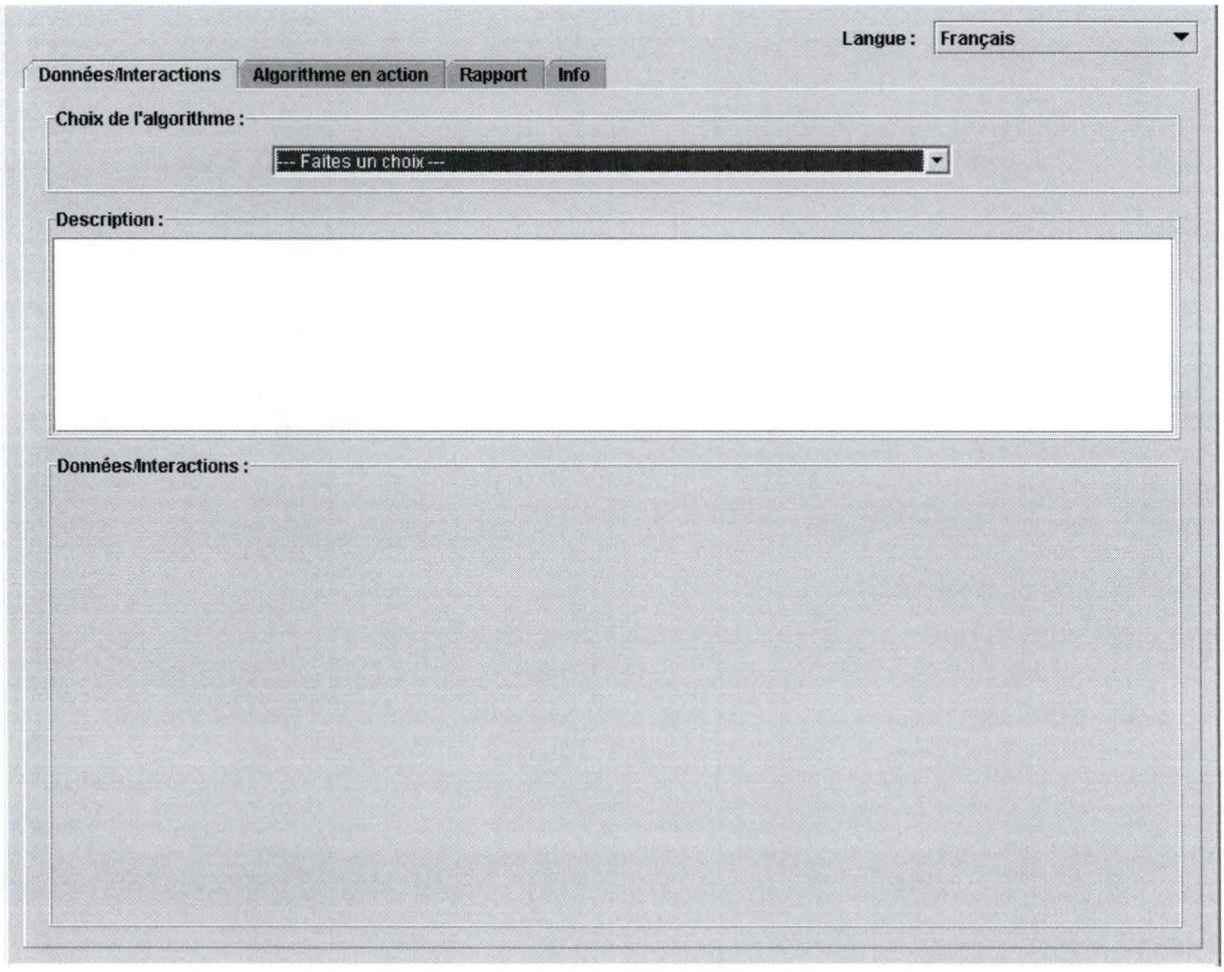


Fig. 14 - Onglet 1 de l'IHM de l'outil d'apprentissage

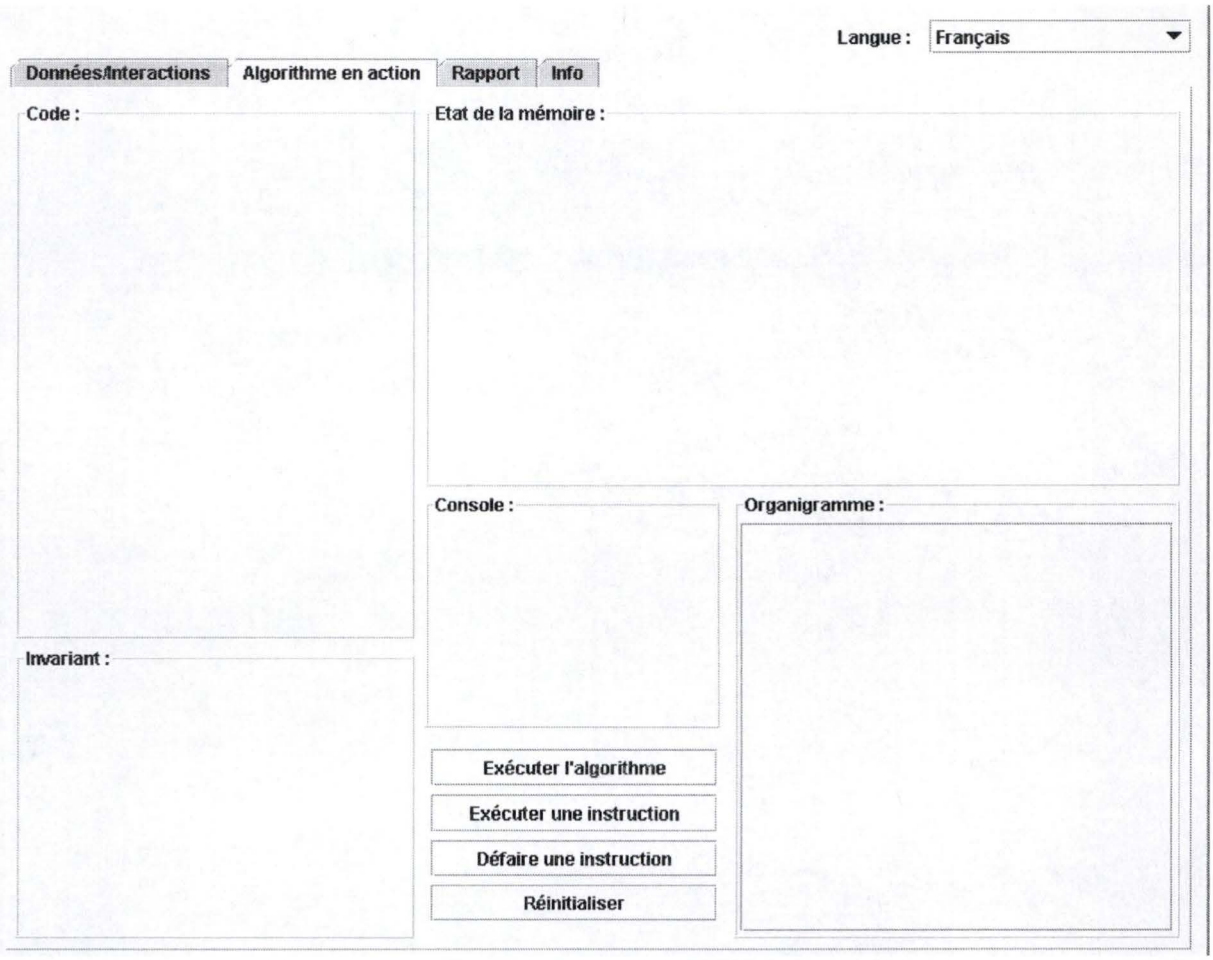


Fig. 15 - Onglet 2 de l'IHM de l'outil d'apprentissage

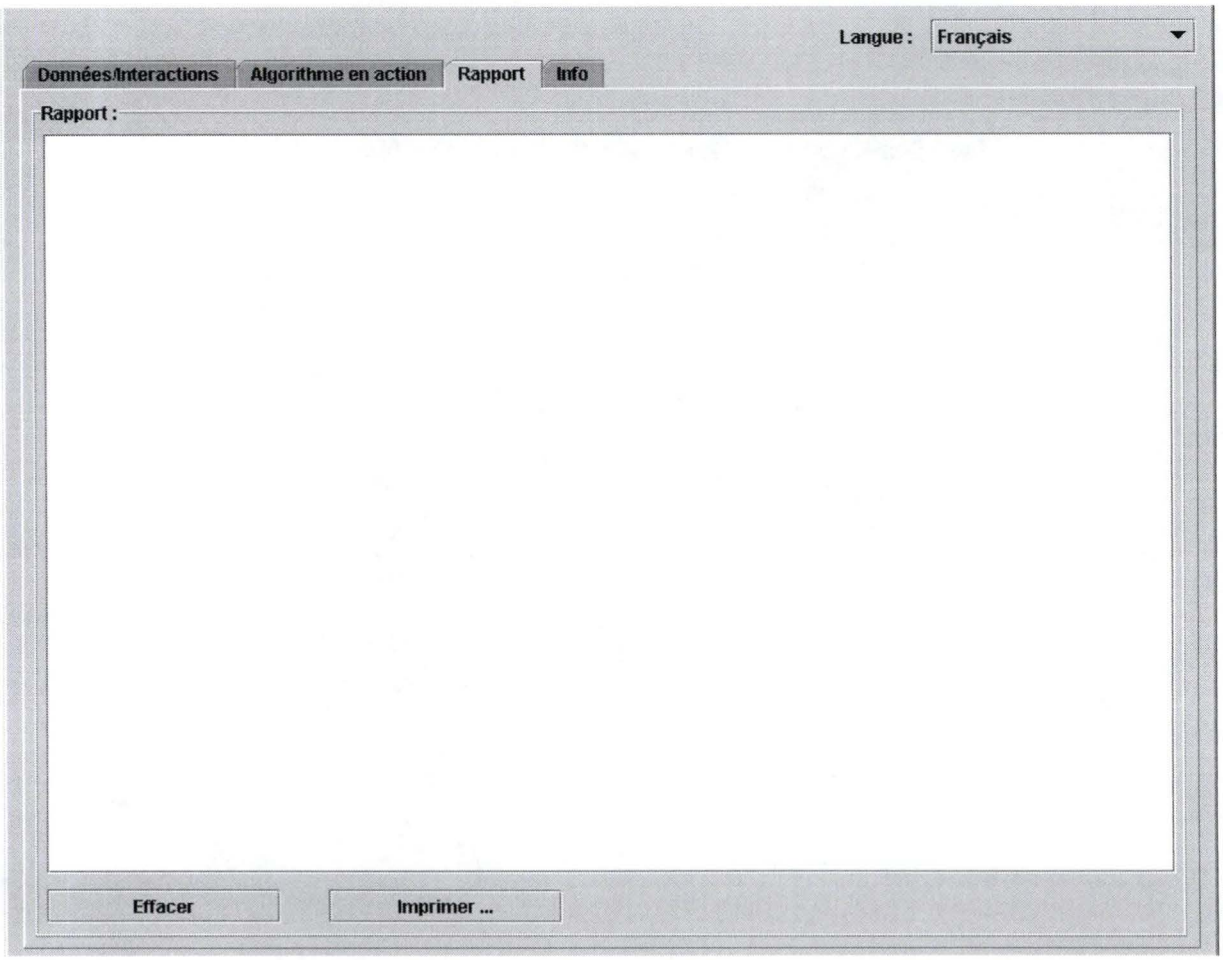


Fig. 16 - Onglet 3 de l'IHM de l'outil d'apprentissage

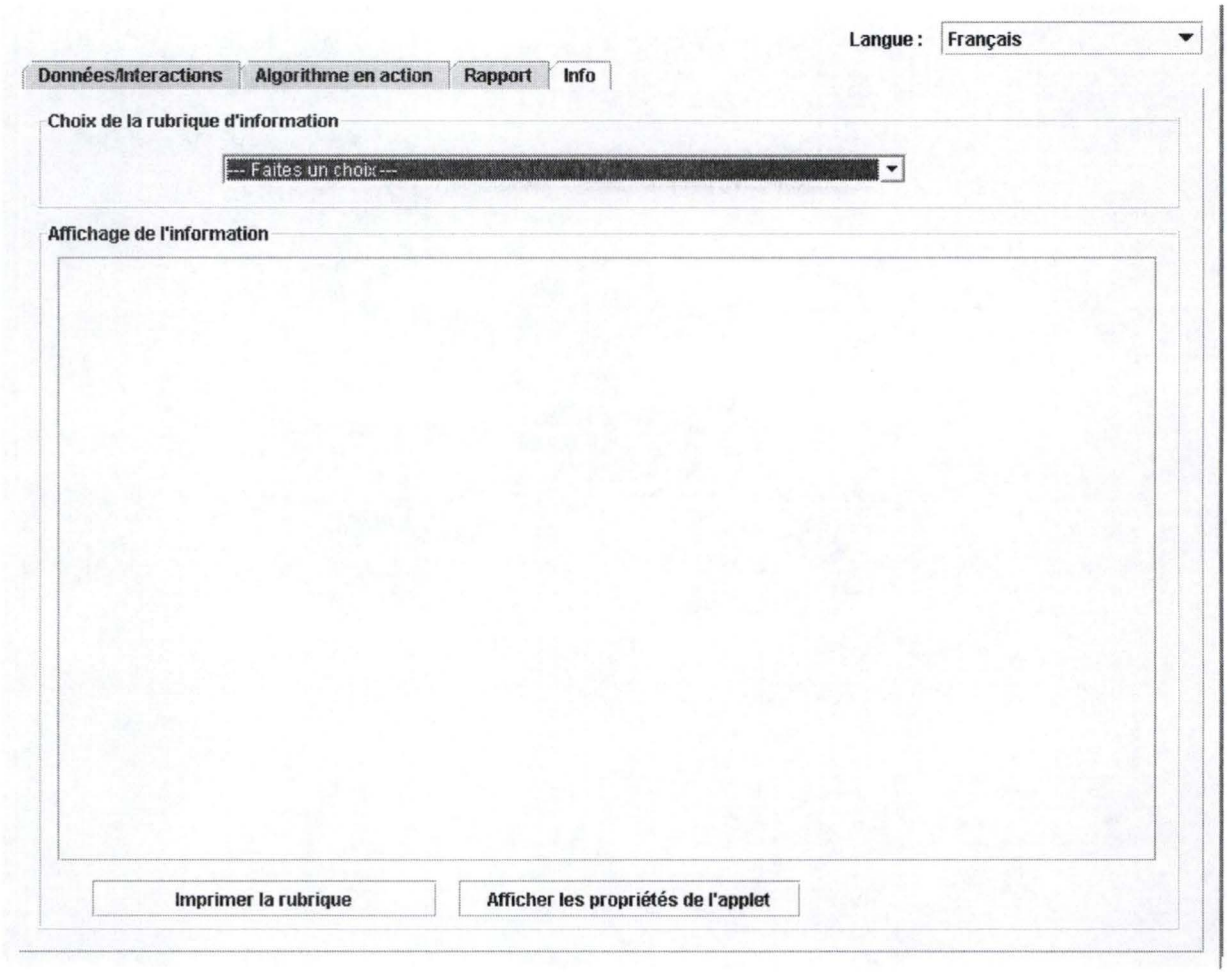


Fig. 17 - Onglet 4 de l'IHM de l'outil d'apprentissage

Outil de traduction

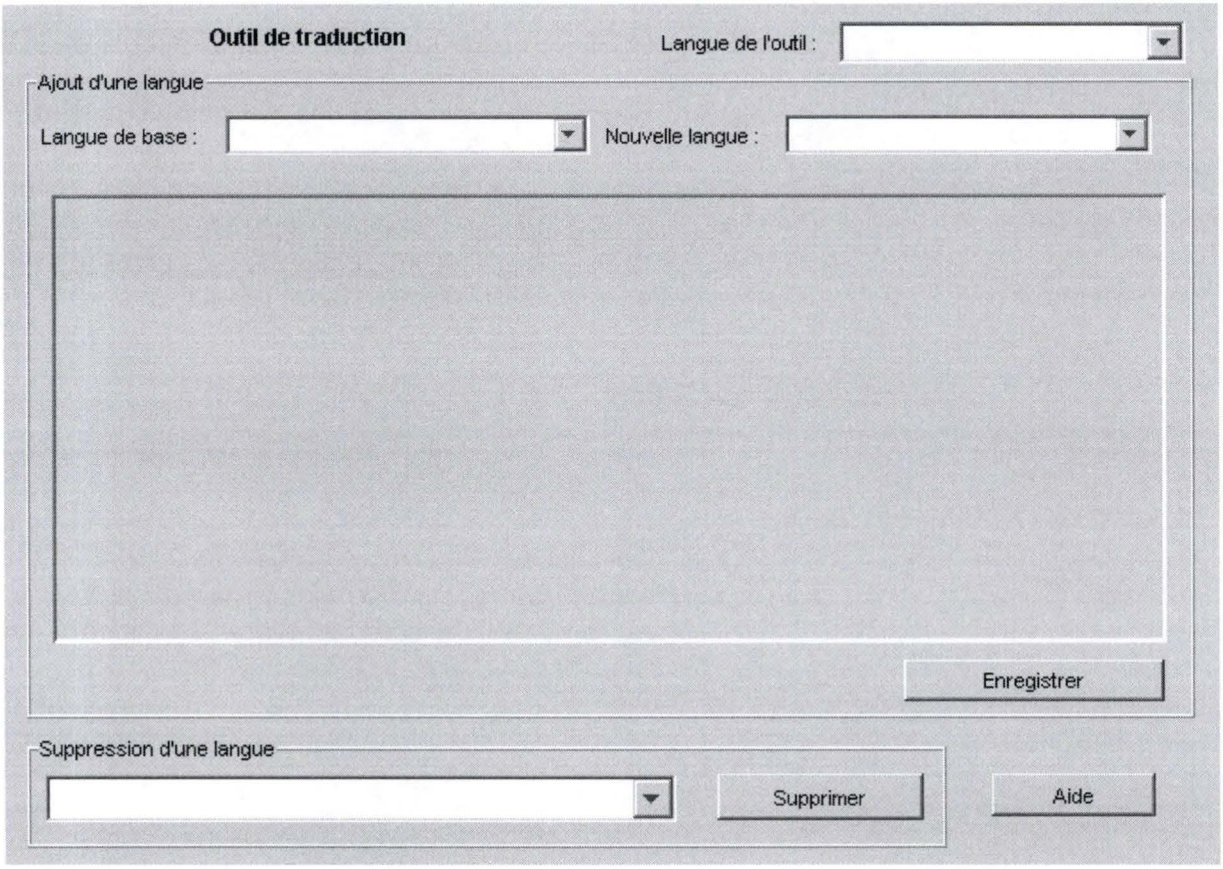


Fig. 18 - IHM de l'outil de traduction

Annexe C

Tableau de sélection du style d'interaction

Tableau de sélection du style d'interaction en fonction des paramètres descriptifs d'un stéréotype d'utilisateur.

Tableau issu du cours de « Conception d'Interfaces Hommes-Machine ».

Style / Paramètre	Expérience de la tâche	Expérience du système d'information	Motivation	Expérience d'un MDI complexe
Langage de commande	riche	riche	élevée	modéré (1)
Langage naturel (2)	élémentaire à riche (3)	élémentaire à moyenne	faible à élevée	modérée (si présence d'un clavier) (1) faible (si interacteur vocale)
Langage d'interrogation	riche	moyenne	modérée	modérée
Questions / Réponses	élémentaire	élémentaire à moyenne	faible	modérée
Touches de fonction	moyenne à riche	moyenne à élevée	modérée	modérée
Sélection de menu	élémentaire	élémentaire	faible à modérée	élémentaire
Remplissage de formulaire	élémentaire à moyenne	élémentaire à moyenne	faible à modérée	élémentaire à modérée
Multi-fenêtrage	moyenne à riche	moyenne	faible à élevée	élémentaire à modérée
Manipulation directe	élémentaire	élémentaire à moyenne	faible à élevée	élémentaire à riche (4)
Interaction iconique	élémentaire à moyenne	moyenne	faible à modérée	élémentaire

(1) L'expérience requise pour l'utilisation du clavier est considérée comme *modérée* étant donné l'existence de MDI bien plus complexes (par exemple, les gants et lunettes utilisés pour l'immersion d'un individu dans un monde virtuel).

(2) Le langage naturel s'applique dans des cas extrêmes allant de la consultation élémentaire de bases de données jusqu'à l'utilisation de systèmes experts de diagnostic.

(3) *Elémentaire* : lors, par exemple, de la gestion de questions téléphoniques dans un centre d'appel.
Riche : lors, par exemple, de l'utilisation d'un système expert.

(4) *Riche* : lors, par exemple, de l'utilisation d'accessoires de navigation dans un monde virtuel (gants, lunettes,...) ou de l'utilisation d'outils spécifiques aux opérations chirurgicales assistées par ordinateur.

Annexe D

Le langage LSD[03]

LSD^{03}: le retour de la vengeance

Pierre-Yves Schobbens Yves Bontemps François Schoubben
Isabelle Linden Sébastien Verbois

23 janvier 2003

Table des matières

1 Introduction

1.1 La bête

Ce document présente le langage LSD^{03}, i.e. "Langage Simple et Didactique". Le 03 signifie basiquement 2003, l'année de mise en service du langage mais peut également signifier "troisième tentative". Ce langage est l'héritier de LSD80 et LSD_02.

Il est conçu pour être simple. Il n'est pas exceptionnel et ne révolutionnera pas l'informatique de demain. Mais il devrait vous permettre de comprendre correctement les différents concepts abordés au cours de Syntaxe et Sémantique. Ceci justifie l'aspect didactique du langage.

LSD^{03}ressemble, d'une manière très simplifiée, à Pascal. Il vous permettra d'utiliser des booléens et des entiers comme types scalaires et des tableaux comme type composé. Il offre toutes les structures de contrôle présente dans un bon langage impératif. Cela signifie, en particulier, que l'instruction goto a reçu la place qu'elle mérite: la poubelle. Le langage permet de définir des procédures récursives et d'utiliser des passages de paramètres par valeur et par variable. Comme dans Pascal, toute variable doit être déclarée avant son utilisation. La déclaration de la variable indique son type, qui ne peut être transformé lors de l'exécution.

1.2 Syntaxe, sémantique, domaine, à quoi ça sert?

1.2.1 Syntaxe et grammaire

Comme expliqué au cours, la *syntaxe* du langage décrit l'ensemble des suites de symboles terminaux qui sont légales. Il est très important de connaître cette syntaxe, lorsque l'on utilise des langages de programmation. C'est ce qui vous permet d'écrire des programmes qui vont pouvoir être lus et traités par le compilateur.

La syntaxe va donc décrire un ensemble Program. Par exemple, la phrase

```
program vide ;
var
begin skip end.
```

appartient à l'ensemble Program.

Ces langages sont des ensembles très grands, typiquement infinis. Il faut donc être très (très!) patient si l'on veut énumérer leur contenu. Afin de décrire d'une manière finie (et donc, "imprimable") le contenu de ces langages, nous utilisons des *grammaires non-contextuelles*. Pour rappel, une grammaire non-contextuelle est composée de quatre éléments:

1. V_T, un ensemble (fini) de symboles *terminaux*;
2. V_N, un ensemble (fini) de symboles *non-terminaux*;
3. $S \in V_N$, un non-terminal de départ;
4. $P \subseteq V_N \times (V_T \cup V_N)^*$, un ensemble de *productions*. Une production détermine par quelle suite de non-terminaux et de terminaux on a le droit de remplacer un non-terminal. Une production $(A,\alpha_1 \cdot \ldots \cdot \alpha_n)$ s'écrira plus lisiblement $A \rightarrow \alpha_1 \ldots \alpha_n$.

Par exemple, la grammaire des nombres en LSD^{03}serait:

1. $V_T = \{$ 0 , 1 , ... , 9 , − $\}$, les caractères de 1 à 9 et le signe "-";
2. $V_N = \{NbSig, Nb, Chiffre\}$,
3. $S = NbSig$

4. les productions sont:

$$\begin{array}{rcl} \textit{NbSig} & \rightarrow & \textit{Nb} \\ \textit{NbSig} & \rightarrow & - \textit{Nb} \\ \textit{Nb} & \rightarrow & \textit{Chiffre} \\ \textit{Nb} & \rightarrow & \textit{Nb Chiffre} \\ \textit{Chiffre} & \rightarrow & \texttt{0} \\ \textit{Chiffre} & \rightarrow & \texttt{1} \\ & \vdots & \\ \textit{Chiffre} & \rightarrow & \texttt{9} \end{array}$$

Afin de déterminer quel langage est défini par une grammaire non-contextuelle, nous avons besoin du concept de *production*. D'abord, on dit qu'une grammaire *produit directement* un mot de symboles terminaux et/ou non-terminaux m', à partir d'un autre mot m, ssi on peut trouver:

1. un non-terminal A dans le mot m;
2. une production $A \rightarrow \phi$ dans la grammaire;

tels que m' est identique à m sauf qu'une des occurrences de A a été remplacée par ϕ.

Par exemple, la grammaire ci-dessus produit directement $-$ *NbChiffreNb* à partir du mot $-$ *NbNb*. Si la grammaire G produit directement m' à partir de m, on écrit: $m \Rightarrow_G m'$.

Une *dérivation* de m' à partir de m est une séquence finie de transformations comme ci-dessus. Pour dire que m' *se dérive de* m dans une grammaire G, il faut donc trouver une séquence de mots $m_1, \ldots, m_n$ telle que:

1. $m = m_1$
2. $\forall i : 1 < i \leq n : m_{i-1} \Rightarrow_G m_i$
3. $m_n = m'$

Cette séquence de mots est appelée une *dérivation*. Si il existe une dérivation de m' à partir de m selon G, nous écrirons $m \Rightarrow_G^* m'$ [1].

Etant donné une grammaire G, son langage $\mathcal{L}_G$, qui est un ensemble de suites de symboles terminaux, $\mathcal{L}_G \subseteq V_T^*$, est défini comme l'ensemble des phrases de terminaux qui sont dérivables du symbole initial de G.

$$\mathcal{L}_G = \{m \in V_T^* | S \Rightarrow_G^* m\}$$

L'occurrence d'un non-terminal qui est remplacé est choisie *au hasard*. Si l'on s'oblige à toujours transformer la *première* occurrence d'un non-terminal dans m, il s'agit d'une dérivation gauche et, si l'on doit toujours d'abord traiter le *dernier* non-terminal de m, il s'agit d'une dérivation droite.

Par exemple, le mot $-$ 1 3 appartient au langage des nombres défini ci-dessus [2]. Par contre, les mots $-$ $-$ 2 et $-$ *Chiffre* n'y appartiennent pas. Le premier, parce-qu'il n'y a pas de dérivation de *NbSig* qui permette d'obtenir ce mot, et le second, parce-qu'il contient un non-terminal.

En résumé, il faut retenir que la syntaxe d'un langage de programmation est un ensemble infini de suites d'éléments terminaux qui est décrit de manière finie par une grammaire non-contextuelle.

Program est l'ensemble des programmes LSD^{0_3} syntaxiquement valides. La grammaire décrivant Program est donnée tout au long de ce document.

1.2.2 Sémantique

Les éléments appartenant à Program n'ont pas de signification. Le but de la *définition sémantique* est de leur fournir cette signification. Pour cela, on procède en deux étapes. D'abord, on fournit un *domaine sémantique*, qui est l'ensemble des significations possibles. Ensuite, on donne une fonction qui fait correspondre, à chaque élément du langage, un élément du domaine sémantique.

1. Exercice: démontrez que la relation $\Rightarrow^*$ est **transitive** ($m_1 \Rightarrow_G^* m_2$ et $m_2 \Rightarrow_G^* m_3$ implique que $m_1 \Rightarrow_G^* m_3$, pour tout m_1,m_2,$m_3 \in (V_T \cup V_N)^*$), **réflexive** ($m \Rightarrow_G^* m$, pour tout $m \in (V_T \cup V_N)^*$) mais qu'elle n'est ***pas*** **anti-symmétrique** ($m_1 \Rightarrow_G^* m_2$ et $m_2 \Rightarrow_G^* m_1$ implique que $m_1 = m_2$, pour tout m_1,$m_2 \in (V_T \cup V_N)^*$). La dernière partie doit se démontrer par un *contre-exemple*.

2. Exercice: donnez plusieurs dérivations (générales, droites, gauches) permettant de montrer cette appartenance

TAB. 1 – *Une très longue table de correspondance*

Nb	$\mathbb{Z}$
0	0
1	1
− 1	−1
⋮	⋮
1578	1578
⋮	⋮

Un exemple devrait éclairer cette obscure explication. Prenons le langage des nombres LSD^{0_3}, défini dans la section ci-dessus. L'ensemble des nombres syntaxiquement valides est appelé Nb. Il nous faut d'abord trouver un domaine sémantique, càd un ensemble de valeurs que *représentent* les éléments de Nb. Nous choisissons $\mathbb{Z}$, l'ensemble des nombres entiers, positifs, négatifs ou nuls. Ensuite, nous devons expliquer comment interpréter les éléments syntaxiques. Nous devons donc faire correspondre, à chaque élément de Nb, un élément de $\mathbb{Z}$. Vu que Nb et $\mathbb{Z}$ sont infinis, nous ne pouvons pas, évidemment, écrire une longue table de correspondance, comme la table 1.

Précédemment, nous avons expliqué que l'on pouvait décrire les langages, qui sont des ensembles infinis, de manière finie, en utilisant des grammaires. Partant du même principe, nous allons définir, de manière finie, les fonctions qui, à chaque élément du domaine syntaxique, font correspondre un élément du domaine sémantique. Pour obtenir cette définition "finie", nous utiliserons typiquement une induction sur la structure des éléments syntaxiques.

La manière de procéder sera donc la suivante. Pour chacune des formes possibles des termes du domaine syntaxique, nous dirons comment l'élément du domaine sémantique peut être calculé. Le grand principe qui soutient la *sémantique dénotationnelle* est la *compositionnalité*: "la sémantique d'un élément ne peut être définie qu'en fonction de la sémantique de ses composants directs".

Par exemple, si l'on se réfère à Nb, l'ensemble des nombres LSD^{0_3}, nous avons décidé que le domaine sémantique était $\mathbb{Z}$. La fonction qui fera correspondre à chaque élément de Nb un élément de $\mathbb{Z}$ s'appelle: $\mathcal{N}$. Elle possède la signature suivante: $\mathcal{N} : \mathsf{Nb} \to \mathbb{Z}$. La flèche $\to$ signifie que $\mathcal{N}$ est une fonction totale: elle est définie pour tous les éléments de Nb.

Nous définissons $\mathcal{N}$ par les équations ci-dessous. Notez l'utilisation des doubles crochets $[\![$ et $]\!]$. Ils ne servent qu'à encadrer les éléments qui appartiennent au domaine syntaxique, de façon à mettre en évidence la distinction entre éléments "syntaxiques" et "sémantiques".

$$\begin{array}{rcl} \mathcal{N}[\![- n]\!] & = & 0 - \mathcal{N}[\![n]\!] \\ \mathcal{N}[\![n \cdot c]\!] & = & (10 * \mathcal{N}[\![n]\!]) + \mathcal{N}[\![c]\!], \text{ avec } n \in \mathit{Chiffre}^+ \\ \mathcal{N}[\![\mathtt{0}]\!] & = & 0 \\ \mathcal{N}[\![\mathtt{1}]\!] & = & 1 \\ & \vdots & \\ \mathcal{N}[\![\mathtt{9}]\!] & = & 9 \end{array} \tag{1}$$

Si il s'agit d'une suite de chiffres n, précédés d'un signe $-$, alors, l'entier correspondant est l'opposé de l'entier représenté par n. Si n est un simple chiffre, alors, on donne, par une liste *finie*, l'entier qui lui correspond. Par exemple, 3 correspondra à 3. Si n peut être divisé en deux parties: n' et c telles que n' est une suite de chiffres (non vide) et c est un chiffre, alors, il faut "décaler vers la gauche" l'entier représenté par n' et y ajouter la valeur de c dans la partie des unités.[3]

3. Relisez, dans la section 1.2.1, la description des grammaires non-contextuelles et essayez d'identifier quels sont les éléments syntaxiques, quels sont les domaines sémantiques et comment la correspondance entre les deux est faite.

1.2.3 Oui, mais ... à quoi ça sert?

L'utilité de la description syntaxique du langage est indiscutable. Elle permet au programmeur ou au constructeur d'un compilateur, de déterminer, sans discussion possible, quelles constructions sont légales dans le langage et lesquelle ne le sont pas.

L'utilité de la définition sémantique peut sembler moins frappante. Pourtant, il suffit de réfléchir aux nombreux choix que vous pouvez poser si vous devez interpréter un appel de procédure. Comment sont passés les paramètres? Que signifie "passage par adresse" ou "passage par résultat"? Dans quel ordre dois-je évaluer les composantes d'une expression? Que se passe-t-il lorsque l'on déclare un tableau possédant un index vide? Que fait l'assignation de tableau à tableau? Avoir une sémantique formelle permet de répondre à ces questions, car elle n'induit qu'une seule interprétation possible des programmes écrits dans un langage.

Donc, dans un monde idéal, les langages seraient tous dotés d'une sémantique formelle, que les programmeurs consulteraient avant de construire les compilateurs. De cette manière, un programme d'un langage $\mathcal{L}$ donnerait les mêmes résultats, qu'il soit compilé et exécuté sur une plateforme X ou Y.

Votre travail, lors de ce TP, sera de construire un *compilateur*. Un compilateur est un programme qui, prenant en entrée un programme P_s, écrit dans le langage *source* Source, produit en sortie un programme P_o dans le langage *objet*, Obj. Cette traduction doit *préserver la sémantique de* P_s. Formellement, étant donné un domaine sémantique $\mathbb{D}$ et deux fonctions sémantiques: $\mathcal{S}_s$: Source $\rightarrow \mathbb{D}$ et $\mathcal{S}_o$: Obj $\rightarrow \mathbb{D}$, si P_o est le programme compilé à partir de P_s, alors,

$$\mathcal{S}_s[\![P_s]\!] = \mathcal{S}_o[\![P_o]\!]$$

Le compilateur que vous devez construire pourra traduire n'importe quel programme LSD^{03} syntaxiquement valide et respectant les contraintes sur les identificateurs en un programme P-Code équivalent. De plus, votre compilateur devra refuser de traduire des textes qui ne sont pas des programmes LSD^{03} syntaxiquement valides ou qui violent les contraintes sur les identificateurs.

Les contraintes sur les identificateurs ne sont pas considérées exactement comme des contraintes syntaxiques, car elles ne peuvent être exprimées au moyen d'une grammaire non-contextuelle.

1.3 Conventions de notation

Au point de vue syntaxique, les terminaux seront notés en police `courrier gras` , comme dans e_1 + e_2. Il ne faudra donc pas confondre + et +. Le premier représente le terminal "+" qui est un caractère tandis que le deuxième est le signe représentant l'opération d'addition sur les entiers, avec sa sémantique usuelle.

Les ensembles syntaxiques sont écrits en caractère sans sérif, comme Nb. Les fonctions sémantiques en caractère "calligraphié", comme $\mathcal{I}$, et les domaines sémantiques avec des majuscules à "double barre", comme dans $\mathbb{Z}$.

Lorsque nous écrivons des grammaires, nous utilisons une forme abrégée des grammaires non-contextuelles, où {,},(,),$^+$,*,| sont des méta-symboles (des "caractères réservés" de la grammaire non-contextuelle).

$\phi_1|\phi_2$ représente le choix entre ϕ_1 et ϕ_2,

$\{\phi_1\}$ représente le fait que ϕ_1 est optionnel,

ϕ_1^+ correspond à n répétitions de ϕ_1 (avec $n \geq 1$),

ϕ_1^* représente n répétitions de ϕ_1 (avec $n \geq 0$),

1.4 Structure du document

Dans ce document, la syntaxe et la sémantique du langage vont vous être présentées. Plutôt que de diviser la présentation en donnant d'abord toute la syntaxe et ensuite toute la sémantique, nous travaillons par "petits bouts", en remontant des éléments les plus simples du langage (les expressions) vers l'élément le plus important (le programme).

2 Domaine sémantique

Un programme LSD^{0_3}est un transformateur de suites d'entiers. Il reçoit en entrée une suite finie d'entiers et produit en sortie une suite d'entiers. Il se peut également qu'il produise une erreur.

Afin de transformer son entrée en la sortie désirée, le programme manipule deux structures de données: une mémoire et un environnement. La mémoire assigne des valeurs à des adresses. Certaines adresses ne sont pas utilisées. La mémoire sera donc modélisée par une fonction *partielle*. Une fonction totale de A dans B est représentée par $A \rightarrow B$ tandis qu'une fonction partielle est écrite $A \hookrightarrow B$. Afin de simplifier la présentation et d'abstraire les choix d'implémentation, nous supposerons que la mémoire est infinie. Une valeur peut être soit une valeur scalaire, soit un tableau.

L'environnement permet d'obtenir l'adresse correspondant à chaque identificateur de variable apparaissant dans le programme. Il permet aussi de retrouver la "procédure" correspondant à chaque identificateur de procédures. Il est important de se rendre compte qu'un identificateur peut être attaché à diffférentes adresses ou procédures, en fonction du contexte dans lequel il est évalué. Un environnement sera donc une suite de fonctions: $\mathsf{Id} \hookrightarrow \mathbb{A} \cup \mathbb{P}$. Soit un environnement $e = f_1 \cdot f_2 \cdot \ldots \cdot f_n$. e peut être vu comme une pile, où les contextes les plus récents sont "au-dessus" (position 1) et les contextes les plus vieux "en-dessous" (position n). Pour trouver l'élément auquel un identificateur id correspond dans l'environnement e, on regarde d'abord s'il est défini dans le contexte sur le dessus de la pile. Si id n'est pas défini dans le contexte courant (celui au-dessus de la pile), on regarde si c'est défini dans un contexte antérieur.

$$\begin{array}{lcll} e(id) & = & f_1(id) & \text{si} \quad f_1(id) \neq \bot \vee n = 1 \\ e(id) & = & (f_2 \cdot \ldots \cdot f_n)(id) & \text{si} \quad f_1(id) = \bot \end{array}$$

Par exemple, en supposant que l'on ait un environnement e défini comme ci-dessous et que l'ensemble des identificateurs soit $\{x,y\}$

$$e = [x \mapsto 123, y \mapsto \bot] \cdot [x \mapsto g, y \mapsto 256] \cdot [x \mapsto \bot, y \mapsto true]$$

alors $e(x) = 123$ et $e(y) = 256$.

Formellement, nous utiliserons donc les ensembles suivants:

$\mathbb{Z}$ représente l'ensemble des entiers,

$\mathbb{B}$ représente l'ensemble des booléens, $\mathbb{B} = \{$ vrai , faux $\}$,

$\mathbb{V}$ est l'ensemble des valeurs scalaires ($\mathbb{V} = \mathbb{B} \cup \mathbb{Z}$),

$\mathbb{A}$ est l'ensemble des adresses,

$\mathbb{I}$ **et** $\mathbb{O}$ sont les ensembles d'inputs et d'outputs, par définition, $\mathbb{I} = \mathbb{O} = \mathbb{Z}^*$,

$\mathbb{P}$ est l'ensemble qui contient toute l'information nécessaire pour exécuter une procédure:

$$\mathsf{Bloc} \times \left((\mathsf{ExprG} \cup \mathsf{ExprD})^* \rightarrow (\mathbb{S}\mathrm{tate} \hookrightarrow \mathbb{S}\mathrm{tate}) \right)$$

Cela comprend deux éléments:

- le *corps* de la procédure. Cet élément syntaxique est appelé *bloc* dans le langage LSD^{0_3}. Celui-ci contient les définitions de variables et de procédures locales, ainsi que les instructions qui composent la procédure.
- une fonction qui *passe les paramètres* de la procédure. Le but de cette fonction est, de manière très opérationnelle, de "créer" l'espace mémoire pour les paramètres, évaluer leur valeur et les rendre accessibles à l'intérieur de la procédure, tout en respectant leur type de passage formel.

$\mathbb{T}$ est l'ensemble des tableaux. Il s'agit de triplets de la forme $(\mathbb{Z} \hookrightarrow \mathbb{A}) \times \mathbb{Z} \times \mathbb{Z}$. Le premier élément est la fonction qui associe à chaque index une valeur, les deuxième et troisième éléments sont, respectivement, les bornes minimales et maximales du tableau.

$\mathbb{E}$ est l'ensemble des environnements, càd des "piles" de la forme $(\mathsf{Id} \hookrightarrow \mathbb{P} \cup \mathbb{A})^*$,

$\mathbb{S}$ est l'ensemble des mémoires, càd des fonctions de signature $\mathbb{A} \hookrightarrow \mathbb{T} \cup \mathbb{V}$.

State est l'ensemble des *états* du programme. Un état doit contenir toute l'information qui permet de calculer l'effet d'une instruction. Nous définissons State comme $\mathbb{S} \times \mathbb{E} \times \mathbb{I} \times \mathbb{O}$.

3 Identificateurs

3.1 Syntaxe

Un identificateur est un élément appartenant à Mot \ Keywords où la catégorie syntaxique Mot est définie par la grammaire ci-dessus et la catégorie syntaxique Keywords contient tous les mots-cles du langage LSD^{0_3}. Nous ne définissons pas ces mots-clefs en extension, mais le lecteur les découvrira en parcourant les différentes règles.

$$
\begin{array}{rcl}
\textit{Mot} & \rightarrow & \textit{Lettre}(\textit{Chiffre}|\textit{Lettre})^* \\
\textit{Lettre} & \rightarrow & \texttt{a} \mid \texttt{b} \mid \texttt{c} \mid \texttt{d} \mid \texttt{e} \mid \texttt{f} \mid \texttt{g} \mid \texttt{h} \mid \texttt{i} \mid \texttt{j} \mid \texttt{k} \mid \texttt{l} \mid \texttt{m} \\
 & \mid & \texttt{n} \mid \texttt{o} \mid \texttt{p} \mid \texttt{q} \mid \texttt{r} \mid \texttt{s} \mid \texttt{t} \mid \texttt{u} \mid \texttt{v} \mid \texttt{w} \mid \texttt{x} \mid \texttt{y} \mid \texttt{z}
\end{array}
$$

3.2 Sémantique

Le domaine sémantique des identificateurs est un petit peu particulier. Il s'agit de l'ensemble (syntaxique) des identificateurs. Un identificateur sera interprété par ...lui-même! La fonction sémantique sera simplement id, la fonction identité ($\forall x : id(x) = x$).

4 Expressions droites

Les expressions droites sont des expressions dont l'évaluation retourne une valeur scalaire. Elles portent ce nom car elles apparaissent dans la partie *droite* des instructions d'affectation.

L'évaluation d'une expression peut échouer, parce-que l'on se réfère à un élément d'un tableau qui n'existe pas, par exemple. La sémantique d'une expression droite sera donc donnée par une fonction *partielle*. Cette fonction sera donc définie si l'évaluation réussi.

4.1 Expressions entières

4.1.1 Syntaxe

Les expressions entières, qui appartiennent à la catégorie syntaxique ExprE sont définies par la grammaire suivante, où *ExprE* est le non-terminal de départ:

$$
\begin{array}{rcl}
\textit{ExprE} & \rightarrow & \textit{ExprE} + \textit{ExprE} \\
 & \mid & \textit{ExprE} - \textit{ExprE} \\
 & \mid & \textit{ExprE} * \textit{ExprE} \\
 & \mid & (\ \textit{ExprE}\ ) \\
 & \mid & \textit{NbSig} \\
 & \mid & \textit{VarRef}
\end{array}
$$

Une référence à une variable constituera une expression entière à la condition qu'elle soit de type entier.

Afin de mettre un programme LSD^{0_3}sous une forme non-ambigüe, il faut connaître les règles de précédence des différents opérateurs. Nous choisissons la règle suivante:

$$\{ +, - \} < \{ * \} < (\ .\ )$$

où $a > b$ signifie a doit être évalué avant b. De plus, nous choisissons de rendre l'opération de soustraction associative à gauche. Par exemple, l'expression

$$\texttt{x} + 3 * \texttt{y} - 5 - \texttt{z}$$

doit être interprétée comme

$$\texttt{x} + \Big(((3 * \texttt{y}) - 5) - \texttt{z}\Big)$$

4.1.2 Sémantique

La sémantique de cette catégorie syntaxique est donnée par la fonction

$$\mathcal{I} : \mathsf{ExprE} \to (\mathbb{S}\mathrm{tate} \hookrightarrow \mathbb{Z})$$

que nous définissons comme suit. Cette fonction utilise la fonction $\mathcal{V}$ qui calcule la valeur d'une référence à une variable ou à une cellule d'un tableau, et qui sera définie par les équations (14) et (15)

$$\begin{aligned}
\mathcal{I}[\![e_1 + e_2]\!]s &= \mathcal{I}[\![e_1]\!]s + \mathcal{I}[\![e_2]\!]s && (2)\\
\mathcal{I}[\![e_1 - e_2]\!]s &= \mathcal{I}[\![e_1]\!]s - \mathcal{I}[\![e_2]\!]s && (3)\\
\mathcal{I}[\![e_1 * e_2]\!]s &= \mathcal{I}[\![e_1]\!]s * \mathcal{I}[\![e_2]\!]s && (4)\\
\mathcal{I}[\![n]\!]s &= \mathcal{N}[\![n]\!]s, \qquad \text{où } n \in \mathsf{Nb} && (5)\\
\mathcal{I}[\![x]\!]s &= \mathcal{V}[\![x]\!]s, \qquad \text{où } x \in \mathsf{Var} && (6)
\end{aligned}$$

Dans la définition ci-dessus, nous avons également supposé que les opérations mathématiques propageaient la valeur d'erreur $\perp$. Par exemple, $\forall x : \perp + x = \perp$.

4.2 Expressions booléennes

4.2.1 Syntaxe

Les expressions booléennes, appartenant à ExprB, possèdent la syntaxe (abstraite) suivante:

$$\begin{array}{rcl}
\mathit{ExprB} & \to & \texttt{true} \\
& | & \texttt{false} \\
& | & \mathit{ExprB}\ \texttt{and}\ \mathit{ExprB} \\
& | & \mathit{ExprB}\ \texttt{or}\ \mathit{ExprB} \\
& | & \texttt{not}\ \mathit{ExprB} \\
& | & \mathit{ExprE} < \mathit{ExprE} \\
& | & \mathit{ExprE} = \mathit{ExprE} \\
& | & \mathit{ExprE} > \mathit{ExprE} \\
& | & (\ \mathit{ExprB}\ ) \\
& | & \mathit{VarRef}
\end{array}$$

Une référence à une variable ne peut constituer une expression booléenne que si son type est booléen.

La règle de priorité pour évaluer les expressions booléennes est

$$\{\ \texttt{not}\ \} < \{\ \texttt{or}\ ,\ \texttt{and}\ \} < \{\ >\ ,\ <\ ,\ =\ \} <\ (\ .\ )\ .$$

4.2.2 Sémantique

La sémantique d'une expression booléenne est calculée par la fonction

$$\mathcal{B} : \mathsf{ExprB} \to (\mathbb{S}\mathrm{tate} \hookrightarrow \mathbb{B})$$

de la façon suivante, pour tout $s \in \mathbb{S}\mathrm{tate}$,

$$\mathcal{B}[\![\texttt{ false }]\!]s = \text{faux} \tag{7}$$

$$\mathcal{B}[\![\texttt{ true }]\!]s = \text{vrai} \tag{8}$$

$$\mathcal{B}[\![e_1 \texttt{ and } e_2]\!]s = \begin{cases} \text{vrai} & \text{si } \mathcal{B}[\![e_1]\!]s = \text{ vrai et } \mathcal{B}[\![e_2]\!]s = \text{ vrai} \\ \text{faux} & \text{si } \mathcal{B}[\![e_1]\!]s = \text{ faux ou } \begin{cases} \mathcal{B}[\![e_1]\!]s = \text{ vrai} \\ \text{et} \\ \mathcal{B}[\![e_2]\!]s = \text{ faux} \end{cases} \end{cases} \tag{9}$$

$$\mathcal{B}[\![e_1 \texttt{ or } e_2]\!]s = \begin{cases} \text{faux} & \text{si } \mathcal{B}[\![e_1]\!]s = \text{ faux et } \mathcal{B}[\![e_2]\!]s = \text{ faux} \\ \text{vrai} & \text{si } \mathcal{B}[\![e_1]\!]s = \text{ vrai ou } \begin{cases} \mathcal{B}[\![e_1]\!]s = \text{ faux} \\ \text{et} \\ \mathcal{B}[\![e_2]\!]s = \text{ vrai} \end{cases} \end{cases} \tag{10}$$

$$\mathcal{B}[\![\texttt{ not } e]\!]s = \begin{cases} \text{vrai} & \text{si } \mathcal{B}[\![e]\!]s = \text{ faux} \\ \text{faux} & \text{si } \mathcal{B}[\![e]\!]s = \text{ vrai} \end{cases} \tag{11}$$

$$\mathcal{B}[\![e_1 < e_2]\!]s = \begin{cases} \text{vrai} & \text{si } \mathcal{I}[\![e_1]\!]s < \mathcal{I}[\![e_2]\!]s \\ \text{faux} & \text{si } \mathcal{I}[\![e_1]\!]s \geq \mathcal{I}[\![e_2]\!]s \end{cases} \tag{12}$$

$$\mathcal{B}[\![e_1 > e_2]\!]s = \begin{cases} \text{vrai} & \text{si } \mathcal{I}[\![e_1]\!]s > \mathcal{I}[\![e_2]\!]s \\ \text{faux} & \text{si } \mathcal{I}[\![e_1]\!]s \leq \mathcal{I}[\![e_2]\!]s \end{cases} \tag{13}$$

$$\mathcal{B}[\![\texttt{ (} e \texttt{) }]\!]s = \mathcal{B}[\![e]\!]s \tag{14}$$

$$\mathcal{B}[\![x]\!]s = \mathcal{V}[\![x]\!]s, \quad \text{où } x \in \mathsf{Var} \tag{15}$$

Notez que cette fonction sémantique impose *l'ordre* dans lequel les expressions booléennes doivent être évaluées. Intuitivement, elles doivent être évaluées de la gauche vers la droite et l'évaluation doit se terminer dès que possible. Par exemple, dans l'hypothèse où `t` est un tableau dont les bornes sont $0 \ldots 9$, `( true or t[10] > 5 )` s'évaluera à vrai , même si l'évaluation de `t[10]` mène à une erreur.

4.3 Références

Une référence à une variable est donnée par la syntaxe (abstraite) suivante. L'ensemble des références à des variables est Var.

4.3.1 Syntaxe

$$\begin{array}{rcl} \textit{VarRef} & \rightarrow & \textit{Id} \\ & | & \textit{Id} \texttt{ [} \textit{ExprE} \texttt{]} \end{array}$$

4.3.2 Sémantique

Afin d'évaluer la valeur d'une variable, nous avons besoin de l'environnement courant et de l'état de la mémoire. La fonction suivante, qui calcule la sémantique d'une référence,

$$\mathcal{V} : \mathsf{Var} \rightarrow (\mathbb{S}\text{tate} \hookrightarrow \mathbb{V} \cup \mathbb{T})$$

est définie de la façon suivante, inductivement, sur la structure des références. Pour une référence sans indirection, c'est-à-dire que l'on ne référence pas un élément d'un tableau, on renvoit la valeur stockée en mémoire à l'adresse renseignée dans l'environnement.

Nous supposons que la variable x apparaissant dans une référence à une variable a été déclarée auparavant. Cela signifie, en particulier, que x a reçu un certain type et que $\varepsilon(x) \neq \bot$.

Notez que si l'identifiant x est un tableau, alors cette fonction retournera le triplet (f, b_{min}, b_{max}) qui définit le tableau.

$$\mathcal{V}[\![x]\!](\sigma,\varepsilon,i,o) \quad = \quad \sigma(\varepsilon(x)) \tag{16}$$

Pour la référence à un élément d'un tableau, on vérifie que l'index recherché se trouve bien dans les bornes du tableau, et que l'évaluation de l'index ne se termine pas par une erreur.

$$\begin{array}{rcll} \mathcal{V}[\![x\ [\ e\ ]\ ]\!](\sigma,\varepsilon,i,o) & = & \sigma(t(n)) & \text{si } b_1 \leq n \leq b_2 \\ & = & \bot & \text{sinon} \end{array} \tag{17}$$

où $n = \mathcal{I}[\![e]\!](\sigma,\varepsilon,i,o)$ et $(t,b_1,b_2) = \sigma(\varepsilon(x))$.

A nouveau, nous demandons que le tableau ait été déclaré avant d'être utilisé ($\varepsilon(x) \neq \bot$). Si le tableau x a été déclaré comme "tableau de type t", alors $x\ [\ e\ ]$ est de type t.

4.4 Expressions droites

4.4.1 Syntaxe

Finalement, une expression droite quelconque est simplement une expression entière, une expression boolénne ou une référence à une variable,

$$\begin{array}{rcl} \mathit{ExprD} & \rightarrow & \mathit{ExprE} \\ & | & \mathit{ExprB} \end{array}$$

4.4.2 Sémantique

La fonction $\mathcal{E}$, avec la fonctionnalité ExprD $\rightarrow$ (State $\hookrightarrow \mathbb{V} \cup \mathbb{T}$) est définie comme:

$$\mathcal{E}[\![e]\!]s \quad = \quad \mathcal{I}[\![e]\!]s \qquad \text{si } e \in \mathsf{ExprE} \tag{18}$$

$$\mathcal{E}[\![e]\!]s \quad = \quad \mathcal{B}[\![e]\!]s \qquad \text{si } e \in \mathsf{ExprB} \tag{19}$$

5 Expressions gauches

5.1 Syntaxe

Une expression gauche sert à référencer une variable ou un élément d'un tableau. Elle est définie par la grammaire:

$$\mathit{ExprG} \quad \rightarrow \quad \mathit{Id}|\mathit{Id}\ [\ \mathit{ExprE}\ ]$$

5.2 Sémantique

Sa sémantique est donnée par la fonction

$$\mathcal{A} : \mathsf{ExprG} \rightarrow (\mathsf{State} \hookrightarrow \mathbb{A})$$

L'adresse d'une variable simple ou un tableau est obtenue en regardant dans l'environnement.

$$\mathcal{A}[\![x]\!](\sigma,\varepsilon,i,o) \quad = \quad \varepsilon(x) \tag{20}$$

Pour obtenir l'adresse d'un élément d'un tableau, on évalue l'expression d'index e. Si cette évaluation se termine sans erreur et que l'index obtenu se trouve dans les bornes du tableau, on renvoie l'adresse stockée à cet index dans le tableau.

$$\begin{array}{rcll} \mathcal{A}[\![x\ [\ e\ ]\ ]\!](\sigma,\varepsilon,i,o) & = & t(n) & \text{si } b_1 \leq n \leq b_2 \\ & = & \bot & \text{sinon} \end{array} \tag{21}$$

où $n = \mathcal{I}[\![e]\!](\sigma,\varepsilon,i,o)$ et $(t,b_1,b_2) = \sigma(\varepsilon(x))$.

6 Instructions

6.1 Syntaxe

Les instructions sont données par la grammaire suivante:

$$
\begin{array}{rcl}
\mathit{Instr} & \rightarrow & \texttt{if } \mathit{ExprB} \texttt{ then } \mathit{Instr} \texttt{ else } \mathit{Instr} \texttt{ fi} \\
& | & \mathit{Instr} \texttt{ ; } \mathit{Instr} \\
& | & \mathit{ExprG} \texttt{ := } \mathit{ExprD} \\
& | & \texttt{read } \mathit{ExprG} \\
& | & \texttt{write } \mathit{ExprE} \\
& | & \texttt{skip} \\
& | & \texttt{while } \mathit{ExprB} \texttt{ do } \mathit{Instr} \texttt{ od} \\
& | & \mathit{Id} \texttt{ (} \mathit{ParEffList} \texttt{)} \\
\mathit{ParEffList} & \rightarrow & \\
& | & (\mathit{ExprG}|\mathit{ExprD})(\texttt{ , } (\mathit{ExprG}|\mathit{ExprD}))^*
\end{array}
$$

Dans l'instruction d'affectation, l'expression gauche ne peut être un tableau. De plus, l'expression droite doit avoir le même type que l'expression gauche.

Dans l'instruction de lecture, nous demandons que l'expression gauche soit de type "entier".

En ce qui concerne l'appel de procédure, la procédure appelée doit être définie dans l'environnement où l'appel a lieu. Il est demandé que les types des paramètres effectifs soient *identiques* aux types des paramètres formels. Leur nombre doit également coïncider. Enfin, si le i-ème paramètre formel est déclaré comme passé par adresse, le i-ème paramètre effectif doit être une ExprG.

Si un paramètre formel requière un tableau d'entiers défini sur les indices [1,5], on ne peut pas lui passer un tableau "plus large" d'entiers, par exemple, d'index [0,5], ni un tableau de même taille mais d'index différent, comme [12,16].

Ces exigences peuvent toutes être vérifiées *avant* l'exécution du programme, car les bornes du tableau sont définies par des constantes. De plus, la visibilité des procédures (i.e. savoir quelles procédures sont définies à quelle ligne du programme) est également analysable statiquement.

6.2 Sémantique

La sémantique des instructions est donnée par la fonction $\mathcal{S}$, déclarée comme

$$\mathcal{S} : \mathsf{Instr} \rightarrow (\mathbb{S}\mathrm{tate} \hookrightarrow \mathbb{S}\mathrm{tate}).$$

La définition de cette fonction inductivement, sur la structure des instructions, comme d'habitude.

6.2.1 Choix conditionnel

Le choix conditionnel `if` e `then` st_1 `else` st_2 `fi` signifie que si, e est évalué à vrai dans l'état actuel s, alors, st_1 doit s'exécuter. Si, au contraire, e est évalué à faux , alors, st_2 doit s'exécuter. Il est possible que l'évaluation de e ne soit pas définie. Dans ce cas, l'ensemble de l'instruction n'est pas non plus définie.

$$\mathcal{S}[\![\texttt{if } e \texttt{ then } st_1 \texttt{ else } st_2]\!]s = \begin{cases} \bot & \text{si } \mathcal{B}[\![e]\!]s = \bot \\ \mathcal{S}[\![st_1]\!]s & \text{si } \mathcal{B}[\![e]\!]s = \text{ vrai} \\ \mathcal{S}[\![st_2]\!]s & \text{si } \mathcal{B}[\![e]\!]s = \text{ faux} \end{cases} \tag{22}$$

6.2.2 Séquence

La séquence st_1 ; st_2 s'exécute de la manière suivante, dans l'état de départ s. D'abord, st_1 s'exécute, ce qui donne, si st_1 est définie dans l'état s, un nouvel état s'. Dans l'état s', st_2 est alors exécutée, ce qui résulte, si elle est définie, en un état s''. Le résultat de l'exécution de st_1 ; st_2 dans l'état s est donc l'état s'', ou $\bot$ si l'une des deux instructions échoue.

Formellement, la séquence d'instructions correspond à la composition fonctionnelle.

$$\mathcal{S}[\![st_1 \ ; \ st_2]\!](\sigma,\varepsilon,i,o) = (\mathcal{S}[\![st_2]\!] \circ \mathcal{S}[\![st_1]\!])\,(\sigma,\varepsilon,i,o) \tag{23}$$

Ceci est bien ce que nous voulons dire par la composition séquentielle. En effet, par définition, nous savons que,

$$\forall x,y : dom(f_1) : (f_2 \circ f_1)(x) = y \iff \exists z : codom(f_1) \cap dom(f_2) : f_1(x) = z \wedge f_2(z) = y.$$

Donc, il est possible de transformer un état du système s en un état s' en exécutant st_1 ; st_2 ssi nous pouvons trouver un état intermédiaire s'' tel que

1. $\mathcal{S}[\![st_1]\!]$ est définie pour s et $\mathcal{S}[\![st_1]\!]s = s''$;
2. $\mathcal{S}[\![st_2]\!]$ est définie pour s'' et $\mathcal{S}[\![st_2]\!]s'' = s'$;

Par conséquent, le résultat de la composition séquentielle sera indéfini si l'exécution de la première instruction dans l'état de départ est indéfinie ou si l'exécution de la deuxième instruction dans l'état intermédiaire est indéfinie.

6.2.3 Affectation

L'affectation d'une expression droite à une expression gauche procède en deux étapes. D'abord, l'expression droite et l'expression gauche sont évaluées dans l'état courant. Ensuite, si ces évaluations donnent un résultat, on remplace la valeur stockée en mémoire à l'adresse calculée, par la valeur de l'expression droite. Toutes les autres variables gardent leur valeur d'origine.

Remarquez que, comme l'évaluation des expressions (droite ou gauche) ne modifie pas l'état du système (i.e. n'a pas d'effet de bord), on peut les évaluer dans n'importe quel ordre.

Pour rappel, l'expression gauche apparaissant dans une affectation ne peut être de type "tableau"[4] et l'expression droite doit être du même type que l'expression gauche.

$$\mathcal{S}[\![x := e]\!](\sigma,\varepsilon,i,o) = \begin{cases} \bot & \text{si} \quad \mathcal{E}[\![e]\!](\sigma,\varepsilon,i,o) = \bot \\ & \quad\ \text{ou } \mathcal{A}[\![x]\!](\sigma,\varepsilon,i,o) = \bot \\ (\sigma\,[\mathcal{A}[\![x]\!](\sigma,\varepsilon,i,o)/\mathcal{E}[\![e]\!](\sigma,\varepsilon,i,o)]\,,\varepsilon,i,o) & \text{sinon} \end{cases} \tag{24}$$

Dans l'équation ci-dessus, nous avons utilisé la notation $f[x/y]$ qui définit une nouvelle fonction g à partir de la fonction f de la façon suivante:

$$g(z) = \begin{cases} f(z) & \text{si } z \neq x \\ y & \text{si } z = x \end{cases}$$

A partir de cette définition, il est évident que seule la cellule mémoire dont l'adresse est spécifiée dans la partie gauche de l'expression voit son contenu modifié par cette instruction.

6.2.4 Lecture sur le flux d'entrée

La lecture sur le flux d'entrée remplace la valeur stockée dans une cellule mémoire, dont l'adresse est passée en paramètre à l'instruction, par la première valeur apparaissant sur le flux d'entrée.

Deux situations peuvent amener à un échec de l'exécution de cette instruction:

1. il est impossible d'évaluer l'adresse passée en paramètre;
2. le flux d'entrée est vide.

$$\mathcal{S}[\![\ \texttt{read}\ x]\!](\sigma,\varepsilon,i,o) = \begin{cases} \bot & \text{si} \quad i = \epsilon \\ & \quad\ \text{ou } \mathcal{A}[\![x]\!](\sigma,\varepsilon,i,o) = \bot \\ (\sigma[\mathcal{A}[\![x]\!](\sigma,\varepsilon,i,o)/n],\varepsilon,i',o) & \text{si} \quad i = n \cdot i', \text{ pour } n \in \mathbb{Z} \wedge i' \in \mathbb{I} \\ & \quad\ \text{et } \mathcal{A}[\![x]\!](\sigma,\varepsilon,i,o) \neq \bot \end{cases} \tag{25}$$

Dans l'équation ci-dessus, ϵ représente le mot vide et $\cdot$ est l'opérateur de concaténation de mots.

4. Exercice: Quel serait l'effet d'une affectation de tableau à tableau, en utilisant la définition présentée ici?

6.2.5 Ecriture sur le flux de sortie

L'écriture sur le flux de sortie ajoute une valeur en queue de la file de sortie. Cette instruction peut échouer si l'évaluation de l'expression entière à écrire sur le flux de sortie n'est pas définie dans l'état courant.

$$\mathcal{S}[\![\ \texttt{write}\ e]\!](\sigma,\varepsilon,i,o) = \begin{cases} \bot & \text{si} \quad \mathcal{I}[\![e]\!](\sigma,\varepsilon,i,o) = \bot \\ (\sigma,\varepsilon,i,o \cdot \mathcal{I}[\![e]\!](\sigma,\varepsilon,i,o)) & \text{sinon} \end{cases} \tag{26}$$

6.2.6 Appel à une procédure

Lorsqu'un appel à une procédure est effectué, nous récupérons le corps de la procédure (*bloc*) et sa fonction de passage des paramètres (*param*) dans l'environnement.

L'appel à une procédure est ensuite très simple:

1. nous ajoutons un nouveau contexte vide, au-dessus de la pile: $e_\perp$ $(\forall x : \mathsf{Id} : e_\perp(x) = \perp)$.
2. nous passons les paramètres, en utilisant la fonction *param*.
3. le *bloc* d'instruction est exécuté. En particulier, cette opération déclare les variables et sous-procédures locales, avant d'exécuter les instructions du corps de la procédure.
4. dans l'état résultant, nous "jetons" le contexte local e', qui se trouve au-dessus de la pile.

$$\mathcal{S}[\![p\ (\ arg\ )\ ]\!](\sigma,\varepsilon,i,o) = \begin{cases} \bot & \text{, si } \mathcal{B}loc[\![bloc]\!](par\ arg\ (\sigma,e_\perp \cdot \varepsilon,i,o)) = \bot \\ (\sigma',\varepsilon',i',o') & \text{, si } \mathcal{B}loc[\![bloc]\!](par\ arg\ (\sigma,e_\perp \cdot \varepsilon,i,o)) = (\sigma',e' \cdot \varepsilon,i',o') \end{cases} \tag{27}$$

où

$$\varepsilon(p) = (bloc,par)$$

6.2.7 Itération

L'opération d'itération est un petit peu plus difficile à définir. Nous allons procéder en deux temps. D'abord, nous donnerons la sémantique intuitive mais non-dénotationnelle et, ensuite, la sémantique dénotationnelle de la boucle.

Intuitivement, la sémantique du while est la suivante: si la condition de séjour est vraie, alors, on exécute une fois le corps de la boucle et puis, récursivement, on réévalue la totalité de la boucle. Si la condition de séjour est fausse, on ne fait rien.

En utilisant le *goto* tant détesté, on aurait donc le schéma suivant:

```
loop :  if e then
          st ;
          goto loop
        else
          skip
        fi
```

Autrement dit, on a

$$\texttt{while}\ e\ \texttt{do}\ st\ \texttt{od} \equiv \texttt{if}\ e\ \texttt{then}\ st\ ;\ \texttt{while}\ e\ \texttt{do}\ st\ \texttt{od}\ \texttt{else}\ \texttt{skip}\ \texttt{fi}$$

et la sémantique de la boucle se donnerait, directement, comme

$$\mathcal{S}[\![\ \texttt{while}\ e\ \texttt{do}\ st\ \texttt{od}\ ]\!]s = \begin{cases} \bot & \text{si } \mathcal{B}[\![e]\!]s = \bot \\ s & \text{si } \mathcal{B}[\![e]\!]s = \text{ faux} \\ \mathcal{S}[\![\ \texttt{while}\ e\ \texttt{do}\ st\ \texttt{od}\ ]\!](\mathcal{S}[\![st]\!]s) & \text{si } \mathcal{B}[\![e]\!] = \text{ vrai} \end{cases}$$

Le problème de cette définition est qu'elle n'est pas compositionnelle, ce qui est un principe sacro-saint de la sémantique dénotationnelle. Pour rappel, la sémantique d'un élément $\mathtt{T}\ a$ est dite

compositionnelle si sa définition ne dépend *que de* a. Ici, clairement, la définition d'une construction d'itération dépend directement de cette construction. Donc, cette définition ne convient pas. Dans la suite, nous proposons une définition compositionnelle, basée sur un calcul de point fixe, qui est équivalente à celle-ci. **Il n'est pas essentiel, pour la compréhension du langage, de comprendre le reste de cette section.**

$$\mathcal{S}[\![\texttt{while}\ e\ \texttt{do}\ st\ \texttt{od}]\!]s = (FIX\ F)s \tag{28}$$

avec

$$F\ g = \lambda s.\begin{cases} \text{si} & \mathcal{B}[\![e]\!]s = \bot & \text{alors} & \bot \\ \text{si} & \mathcal{B}[\![e]\!]s = \text{ vrai} & \text{alors} & (g \circ \mathcal{S}[\![st]\!])\ s \\ \text{si} & \mathcal{B}[\![e]\!]s = \text{ faux} & \text{alors} & s \end{cases}$$

L'équation 27 utilise la fonction F qui est une *transformatrice de fonction.* Sa signature est

$$(\mathbb{S}\text{tate} \hookrightarrow \mathbb{S}\text{tate}) \rightarrow (\mathbb{S}\text{tate} \hookrightarrow \mathbb{S}\text{tate}).$$

L'opérateur FIX calcule le plus petit point fixe de la fonction F. Pour rappel, un point fixe d'une fonction $f : A \rightarrow A$ est un élément a de A tel que $f(a) = a$.

Afin de calculer le *plus petit point fixe* de la fonction f, nous devons choisir un ordre sur A. Pour ce faire, nous prenons une relation $\sqsubseteq$: $A \times A$ qui est un ordre partiel sur A, càd que cette relation binaire possède les trois propriétés suivantes:

1. $\forall a : A : a \sqsubseteq a$ (réflexivité)
2. $\forall a,b,c : A : a \sqsubseteq b \wedge b \sqsubseteq c \implies a \sqsubseteq c$ (transitivité)
3. $\forall a,b : A : a \sqsubseteq b \wedge b \sqsubseteq a \implies a = b$ (antisymmétrie)

On peut montrer (exercice) que si A possède un élément minimal $\bot$, d'après la relation $\sqsubseteq$,($\forall a : A : \bot \sqsubseteq a$), alors $\bot$ est unique.

Dès lors, le *plus petit point fixe* de f est l'élément (unique) $a \in A$ tel que:

1. $f(a) = a$
2. $\forall b : A : f(a) \sqsubseteq f(b)$

Comme ordre sur $\mathbb{S}\text{tate} \hookrightarrow \mathbb{S}\text{tate}$, nous choisissons le *partage d'information,*

$$\forall f,g : \mathbb{S}\text{tate} \hookrightarrow \mathbb{S}\text{tate} : \Big(f \sqsubseteq g \iff \big(\forall s,s' : \mathbb{S}\text{tate} : f(s) = s' \implies g(s) = s'\big)\Big).$$

L'élément minimal de cet ordre est la fonction $f_\bot$ qui est telle que

$$\forall s : \mathbb{S}\text{tate} : f_\bot(s) = \bot.$$

7 Déclarations

Les déclarations de variables et de procédures mettent à jour le contexte courant (càd au-dessus de la pile de l'environnement courant), en assignant de nouvelles cellules mémoires aux identifiants déclarés.

7.1 Déclaration de variables

7.1.1 Syntaxe

$$\begin{array}{rcl} \textit{DeclVar} & \rightarrow & \textit{Id}\ \texttt{:}\ \textit{Type}\ \texttt{;} \\ \textit{Type} & \rightarrow & \textit{TypeScalaire} \mid \texttt{array [}\ \textit{Nb}\ \texttt{,}\ \textit{Nb}\ \texttt{]}\ \texttt{of}\ \textit{TypeScalaire} \\ \textit{TypeScalaire} & \rightarrow & \texttt{boolean} \mid \texttt{integer} \end{array}$$

Dans le programme, les déclarations de variables seront regroupées dans des blocs. A l'intérieur d'un tel groupe, une variable ne peut être déclarée qu'une seule fois.

Le type d'une variable x déclarée x : `boolean` est *booléen*, le type d'une variable déclarée x : `integer` est *entier* et le type de la variable déclarée x : `array` [b_1 , b_2] `of` t est *tableau de type* t. Cette information doit être utilisée pour vérifier la cohérence des types, avant la phase de génération de code. Pour rappel, un programme LSD^{0_3} bien formé nécessite que

1. toutes les variables utilisées dans une expression ait été déclarée préalablement à son utilisation;
2. tous les types des expressions soient cohérents.

7.1.2 Sémantique

La sémantique d'une déclaration de variable est donnée par la fonction

$$\mathcal{D}_v : \mathsf{DeclVar} \to (\mathbb{S}\mathrm{tate} \to \mathbb{S}\mathrm{tate}).$$

En supposant que $t \in \mathsf{TypeScal}$, on assigne à x une adresse a qui n'est pas encore utilisée dans l'environnement courant.[5]

$$\mathcal{D}_v[\![x : t\ ;\ ]\!](\sigma,\varepsilon,i,o) = (\sigma,\varepsilon[x/a],i,o), \qquad \text{pour un } a \in \mathbb{A} \text{ tel que } \forall id : \mathsf{Id} : \varepsilon(id) \neq a \tag{29}$$

Dans l'équation 28, nous avons utilisé l'opération $\varepsilon[x/a]$, où ε est un environnement, càd une *pile de contextes* (de fonctions). Donc, si $\varepsilon = e_1 \cdot e_2 \cdot \ldots \cdot e_n$, la mise à jour de ε est la mise à jour du contexte au-dessus de la pile: $e_1 \cup \{x \mapsto a\} \cdot \ldots \cdot e_n$.

Dans le cas d'un tableau, il faut trouver un ensemble d'adresses qui ne sont pas encore utilisées dans l'environnement courant. Le nombre d'adresses est la taille du tableau plus une, qui servira à stocker la définition du tableau en mémoire. On crée alors une fonction f qui, à chaque entier où l'index du tableau est défini, fait correspondre une adresse différente, et on stocke cette fonction en mémoire, avec les bornes du tableau, à une autre adresse a_t. Finalement, on ajoute au contexte courant la paire (x,a_t), en supposant que x est l'identifiant du tableau.

$$\mathcal{D}_v[\![x : \texttt{array}\ [\ b_1\ ,\ b_2\ ]\ \texttt{of}\ t\ ;\ ]\!](\sigma,\varepsilon,i,o) = (\sigma[a_t/(f,\mathcal{N}[\![b_1]\!],\mathcal{N}[\![b_2]\!])],\varepsilon[a_t/e],i,o) \tag{30}$$

où $a_t \in Addr$ et $f : \mathbb{Z} \hookrightarrow Addr$ satisfont les contraintes suivantes:

1. a_t, l'adresse du tableau, n'est utilisée par aucune autre variable dans l'environnement: $\nexists id : \mathsf{Id} : \varepsilon(id) = a_t$
2. l'adresse du tableau n'est pas une des cellules du tableau: $a_t \notin codom(f)$
3. l'ensemble des cellules du tableau n'est utilisé par aucune autre variable dans l'environnement: $\forall a : A : \nexists id : \mathsf{Id} : \varepsilon\ id = a$
4. les points où f est définie correspondent à l'intervalle de définition du tableau, $dom(f) = \{i | \mathcal{N}[\![b_1]\!] \leq i \leq \mathcal{N}[\![b_2]\!]\}$
5. f est surjective, càd que pour toute adresse du codomaine de f, il existe un et un seul indice dans le domaine de f qui référence cette adresse: $\forall a,b : dom(f) : f(a) = f(b) \implies a = b$

Remarquez que lorsque l'on définit un tableau sur un index vide (par exemple, x : `array` [3 , 1]), l'effet est

1. la création d'une fonction f ne référençant *aucune* cellule mémoire ($codom(f) = \emptyset$), ce qui implique que toutes les contraintes sur f sont trivialement valides,
2. la réservation d'une zone mémoire pour stocker f, ainsi que les bornes du tableau

Néanmoins, toute tentative d'accès à un des éléments de ce tableau provoquera une erreur. Si aucune tentative d'accès, tant en écriture qu'en lecture, n'a lieu, alors, ce tableau ne causera pas d'erreur. Tenez compte de cet élément en implémentant votre compilateur.

5. Remarquez que, *techniquement*, $\mathcal{D}_v$ n'est pas une fonction mais une relation, car il peut y avoir *plusieurs* a satisfaisant la condition. Néanmoins, tous les états résultants sont identiques, à un renommage des adresses près (i.e. ils sont isomorphes).

7.1.3 Liste de déclarations de variables

Lorsque nous devons traiter une liste de déclarations de variables, $dvl \in \mathsf{DeclVar}^*$, nous utilisons une itération sur les listes, pour traiter les déclarations les unes après les autres. L'itération est codée de manière récursive comme suit. Nous suivons le schéma de la fonction bien connue map en ML. Cas de base, lorsque la liste est vide, on ne fait rien,

$$\mathcal{D}_{vl}[\![\epsilon]\!]s = s \tag{31}$$

et le cas inductif, on enlève le premier argument, on y applique $\mathcal{D}_v$ et puis, récursivement, on applique $\mathcal{D}_{vl}$ au résultat:

$$\mathcal{D}_{vl}[\![dv \cdot dvl]\!]s = \mathcal{D}_{vl}[\![dvl]\!](\mathcal{D}_v[\![dv]\!]s) \tag{32}$$

7.2 Déclarations de procédures

7.2.1 Syntaxe

La déclaration de procédures obéit à la syntaxe suivante:

$$\begin{array}{rcl} \mathit{DeclProc} & \rightarrow & \texttt{procedure}\ \mathit{id}\ (\ \mathit{DeclArgList}\ );\\ & & \mathit{Bloc}\ ; \\ \mathit{DeclArgList} & \rightarrow & \\ & | & (\mathit{DeclArg}\ ,\ )^*\mathit{DeclArg} \\ \mathit{DeclArg} & \rightarrow & \{\ \texttt{var}\ \}\mathit{Id}\ :\ \mathit{Type} \end{array}$$

Il existe certaines contraintes sur les procédures. La première exigence porte sur l'unicité des noms de variables, paramètres et sous-procédures. Supposons que nous ayons une procédure de la forme suivante:

```
procedure p ( pf );
var dv
dp
begin
    st
end;
```

Les identifiants utilisés dans *pf*, *dv* et *dp* sont uniques. Donc, aucun nom de paramètre utilisé dans *pf* ne peut être le nom d'une variable de *dv* ou d'une procédure de *dp*. De la même façon, une variable de *dv* ne peut porter le même nom qu'un paramètre formel. Finalement, aucun de ces identificateurs ne peut être le nom de la procédure (p) qui est définie.

La deuxième exigence se réfère au "type" des variables passées par valeur. Un tableau ne peut être passé par valeur. On ne peut le passer que par adresse.

La deuxième exigence se réfère à l'*ordre* de déclaration des procédures. Les appels "en avant" (par des déclarations `forward` , en Pascal) sont interdits en LSD^{0_3}. Toute procédure doit donc avoir été déclarée avant d'être appelée. Les seuls appels autorisés dans *st* sont donc:

1. les appels à une sous-procédure, déclarée dans *dp*. A nouveau, seules les procédures du plus haut niveau sont prises en compte. Les autres ne sont pas visibles à partir de p.
2. les appels récursifs à p.
3. les appels à une procédure définie "plus haut" que p.

Par exemple, dans le programme de la figure 1, les quatre premiers appels, dans le corps de la procédure `p121` sont autorisés et le dernier est interdit. Dans le corps de la procédure `p11`, l'appel à `p12` est interdit, car, bien que déclarés au même niveau, `p12` est déclarée *après* `p11`.

7.2.2 Sémantique

La déclaration d'une procédure doit ajouter, dans l'état courant, le bloc qui y est associé. Nous devons également déterminer *comment le passage des paramètres effectifs s'effectuera.*

```
program P;
var
  procedure p1();
  var
     procedure p11();
     var
        procedure p111();
        var
        begin
        end;
     begin
         p12(); {Interdit}
     end;

     procedure p12();
     var
        procedure p121();
        var
        begin
           p12();
           p121();
           p11();
           p1();
           p111();   {Interdit}
        end;
     begin
     end;
  begin
  end;
begin
end.
```

FIG. 1 – *Programme avec appels de procédure*

Supposons que nous disposions d'une fonction

$$\mathcal{P}ar : \mathsf{DeclArgList} \to ((\mathsf{ExprD} \cup \mathsf{ExprG})^* \to (\mathbb{S}\mathrm{tate} \hookrightarrow \mathbb{S}\mathrm{tate}))$$

qui, lorsqu'on lui passe une liste de paramètres formels, retourne une fonction déterminant comment le passage des paramètres effectifs s'effectuera. Nous montrerons plus loin comment définir cette fonction.

Pour l'instant, il est évident que la sémantique d'une déclaration de procédure, qui est calculée par la fonction

$$\mathcal{D}_p : \mathsf{DeclProc} \to (\mathbb{S}\mathrm{tate} \to \mathbb{S}\mathrm{tate})$$

est (où *dal* signifie "déclaration d'arguments (liste)")

$$\mathcal{D}_p[\![\texttt{procedure}\ p\ (\ dal\ );\ b]\!](\sigma,\varepsilon,i,o) = (\sigma,\varepsilon[p/(b,\mathcal{P}ar[\![dal]\!])],i,o). \tag{33}$$

Attention, remarquez qu'aucun argument effectif et aucun état n'est passé à la fonction $\mathcal{P}ar$. Ceci est logique, puisque cette information ne sera connue que lors de *l'appel* de la procédure.

Il reste à définir comment les paramètres sont passés.

Lorsqu'il n'y a pas de paramètres, on ne fait rien:

$$\mathcal{P}ar[\![\epsilon]\!]\epsilon\ s = s \tag{34}$$

Lorsqu'un paramètre est passé par adresse, on modifie le contexte courant pour qu'il référence, sous l'identifiant du paramètre formel, l'adresse du paramètre effectif, ***dans l'environnement où l'appel a été effectué*** (et non dans le contexte modifié par la déclaration, bien sûr).

$$\mathcal{P}ar[\![al \cdot \{\ ,\ \} \cdot\ \texttt{var}\ x : t]\!](pel \cdot pe)s = (\sigma',\varepsilon'[x/\mathcal{A}[\![pe]\!]s],i',o') \tag{35}$$

où $pe \in \mathbb{A}$ et $\mathcal{D}_{arg}[\![al]\!]pel\ s = (\sigma',\varepsilon',i',o')$.

Lorsqu'un paramètre est passé par valeur, on modifie le contexte courant, pour qu'il référence une nouvelle cellule mémoire, sous l'identifiant du paramètre formel. Cette cellule est initialisée avec la valeur du paramètre effectif, évalué dans l'environnement où l'appel à lieu.

Désormais, nous pouvons comprendre entièrement la signification d'un appel de procédure.

1. Un nouveau contexte est ajouté sur la pile. L'environnement devient donc $c \cdot e$.
2. Les paramètres effectifs sont évalués dans l'environnement e. Pour chaque paramètre formel pf,
   (a) s'il est passé par valeur, l'adresse du paramètre effectif est calculée (p.ex. a), et la paire $(pf \mapsto a)$ est ajoutée à c: $c := c \cup \{pf \mapsto a\}$.
   (b) s'il est passé par adresse, une nouvelle adresse a est choisie, telle que la mémoire s est indéfinie en a ($s(a) = \bot$). Dans le contexte c, on ajoute la paire $(pf \mapsto a)$ et, à l'adresse a, on place la valeur du paramètre effectif (p.ex. v): $s := s \cup \{a \mapsto v\}$.
3. Le bloc de la procédure est alors exécuté. Ce bloc peut, à nouveau, déclarer des procédures et des variables dans le contexte c.
4. Le contexte c est simplement jeté de la pile des environnements. Donc, l'environnement résultant est e, càd l'environnement dans lequel la procédure avait été appelée. Remarquez que la mémoire a été modifiée par la procédure. Par conséquent, les variables passées *par adresse* voient bien leur contenu modifié par l'appel à la procédure.

$$\mathcal{P}ar[\![al \cdot \{\ ,\ \} \cdot x : t]\!](pel \cdot pe)s = (\sigma'[a/\mathcal{E}[\![pe]\!]s],\varepsilon'[x/a],i',o') \tag{36}$$

où $pe \in \mathbb{V}$, $\mathcal{D}_{arg}[\![al]\!]pels = (\sigma',\varepsilon',i',o')$ et l'adresse a est telle que $\nexists id : \mathsf{Id} : \varepsilon'(id) = a$.

De manière similaire à $\mathcal{D}_{vl}$, nous pouvons expliquer comment une ***liste*** de procédures est définie. Cela se limite à une simple itération sur une liste, où nous appliquons la fonction $\mathcal{D}_p$ à chaque élément (cfr. eq. (30) et (31)).

$$\mathcal{D}_{pl}[\![\epsilon]\!]s = s \tag{37}$$

$$\mathcal{D}_{pl}[\![dp \cdot dpl]\!]s = \mathcal{D}_{pl}[\![dpl]\!](\mathcal{D}_p[\![dp]\!]s) \tag{38}$$

8 Bloc

8.1 Syntaxe

$$
\begin{array}{rcl}
\mathit{Bloc} & ::= & \texttt{var}\ \mathit{DeclVar}^* \\
 & & \mathit{DeclProc}^* \\
 & & \texttt{begin} \\
 & & \quad \mathit{Instr} \\
 & & \texttt{end}
\end{array}
$$

8.2 Sémantique

Lorsque l'on exécute un bloc, les éléments suivants se suivent:

1. Un contexte vide est ajouté au-dessus de la pile d'environnements.
2. On déclare les variables locales au bloc dans ce nouveau contexte.
3. Les procédures locales sont déclarées et ajoutées au contexte courant.
4. L'instruction du bloc est exécutée dans l'environnement résultant.

Ces différentes étapes sont formalisées par la fonction

$$\mathcal{B}loc : \mathsf{Bloc} \to (\mathbb{S}\mathrm{tate} \hookrightarrow \mathbb{S}\mathrm{tate}).$$

$$
\mathcal{B}loc \left[\!\!\left[\begin{array}{l} \texttt{var}\ dvl \\ dpl \\ \texttt{begin} \\ \quad st \\ \texttt{end} \end{array} \right]\!\!\right] (\sigma,\varepsilon,i,o) = \mathcal{S}[\![st]\!]\Big(\mathcal{D}_{pl}[\![dpl]\!]\Big(\mathcal{D}_{vl}[\![dvl]\!](\sigma,e_\perp \cdot \varepsilon,i,o)\Big)\Big) \tag{39}
$$

9 Programme

9.1 Syntaxe

Un programme obéit à la syntaxe suivante:

$$
\begin{array}{rcl}
\mathit{Program} & \to & \texttt{program}\ \mathit{Id}\ \texttt{;} \\
 & & \mathit{Bloc}\ \texttt{.}
\end{array}
$$

A nouveau, nous insistons sur *l'unicité des identifiants.* Cela est aussi valable pour le *nom du programme*, qui ne peut, par conséquent, être identique à un des identifiants de variable ou de procédure déclarés dans le programme.

9.2 Sémantique

Sa sémantique est donnée par la fonction

$$\mathcal{P} : \mathbb{I} \hookrightarrow \mathbb{O}$$

qui est définie de la façon suivante, pour un flux d'entrée donné i et un programme $\texttt{program}\ p\ \texttt{;}\ b\ \texttt{.}$.

L'état de départ doit être tel que

1. l'environnement ne contient qu'un seul contexte, qui est vide: $e_\perp$.
2. le flux de sortie doit être vide (ϵ), ce qui signifie que rien n'a encore été écrit dans le fichier de sortie.

Remarquez que nous ne posons pas d'hypothèse sur l'état de la mémoire. Dans l'équation (39), nous utilisons donc une variable σ_0 qui peut représenter *n'importe quelle* mémoire. Même si toutes les cellules de la mémoire contiennent une valeur, cela n'a pas d'importance. En effet, afin de savoir si une cellule est allouée ou non, nous ne regardons pas dans la mémoire mais dans l'environnement.[6] Ceci correspond assez à un mécanisme de *garbage collection*: lorsqu'une cellule mémoire n'est plus référencée par aucun identificateur, on considère qu'elle est libérée et peut à nouveau être utilisée lors d'une prochaine allocation.

Dans cet état initial, nous exécutons le bloc b et nous jetons l'environnement, le fichier d'entrée et la mémoire, et nous contentons de retourner le flux de sortie, à condition que celui-ci existe.

$$\mathcal{P}[\![\ \texttt{program}\ p\ ;\ b\ .\ ]\!]i = \begin{cases} o' & \text{si} \quad \mathcal{B}loc[\![b]\!](\sigma_0,e_\perp,i,\epsilon) = (\sigma',\varepsilon',i',o') \\ \perp & \text{si} \quad \mathcal{B}loc[\![b]\!](\sigma_0,e_\perp,i,\epsilon) = \perp \end{cases} \tag{40}$$

Cette façon de faire peut-être considérée comme travaillant sur un "buffer". Les écritures sur le flux de sortie ne sont "officialisées" que lorsque le programme s'est correctement terminé. Donc, si le programme diverge ("se plante") sur sa dernière instruction et que des éléments ont déjà été écrits sur le flux de sortie auparavant, le résultat global de l'exécution du programme est $\perp$ et non pas un ensemble partiel de résultats.

10 Commentaires

10.1 Syntaxe

Des commentaires peuvent être introduits dans du code LSD^{03}. Un commentaire est une suite de caractères et de chiffres, placés entre accolades:

$$\textit{Comment} \quad \rightarrow \quad \{\ (\Sigma \setminus \{\ \{\ ,\ \}\ \})^*\ \}$$

où Σ représente l'ensemble des caractères disponibles (l'alphabet). Il contient les lettres, les chiffres, les symboles mathématiques, mais également les caractères de séparation tels que les tabulations, les retours à la ligne et les espaces.

10.2 Sémantique

Un commentaire est simplement ignoré par l'analyseur lexical. Sa sémantique est donc complètement nulle, du point de vue de l'exécution du programme.

11 Exemple de programme LSD^{03}

```
program palindrome ;
{
   Note :
      \in représente "appartient à", Nat est l'ensemble des nombres naturels,
      \forall représente "pour tout" et out/in dénotent les deux flux (sortie/entrée).

  Pré :
    le flux d'entrée "in" commence par un entier positif ou nul, x
    qui est inférieur ou égal à 100.
    Après cet entier, il contient au moins x entiers.
    Donc, in = x . s1 . ... sx . e, avec e \in Nat* et x,s1,..,sx \in N
  Post :
    le flux de sortie contient 1 ssi s1 . ... . sx est un palindrome, càd
```

6. A nouveau, remarquez que cela signifie que $\mathcal{P}$ est une *relation* et non une fonction. Le choix d'une mémoire particulière n'influe pas (ne doit pas influer) sur le résultat du programme.

```
        (a) x <= 1
    ou (b) s1 = sx et s2 ... s(x-1) est un palindrome
         càd \forall i : 1 <= i <=x :  si = s(l-i+1)
}

var
   t : array [1,100] of integer;
   l : integer;
   b : boolean;
{
  Pré :
    le flux d'entrée commence par un entier positif ou nul, x
    qui est inférieur ou égal à 100.
    Ensuite, il contient au moins x entiers.
    Donc, in = x . s1 . ... sx . e, avec e \in Nat* et x,s1,..,sx \in Nat
  Post : l = x et \forall j : 1 <= j <= l : t[j] = sj et in = e
}
procedure filltab(var t : array[1,100] of integer, var l : integer);
var i : integer;
begin
  read l; {l = x}
  if l > 0 then
    i:= 1;
    {Inv: forall j : 1 <= j < i: t[j] = sj }
    while i < l or i = l do
      read t[i];
      i := i +1
    od
  else
    skip
  fi
end;


{
  Pré : forall j : 1 <= j <= l :  t[j] est défini et l >= 0 et l = l0
  Post : t inchangé; ispal=true ssi t[1]...t[l] est un palindrome
}
procedure ispalindrome(var t : array[1,100] of integer, l : integer, var ispal : boolean);
var i : integer;
begin
  ispal := true;
  i := 1;
  {Inv:
      forall j : 1 <= j < i : t[j] = t[l0 - j + 1]
      l = l0 - i + 1
  }
  while (i < l or i = l) and ispal do
    ispal := t[i] = t[l];
    i := i + 1;
    l := l - 1
  od
end;
```

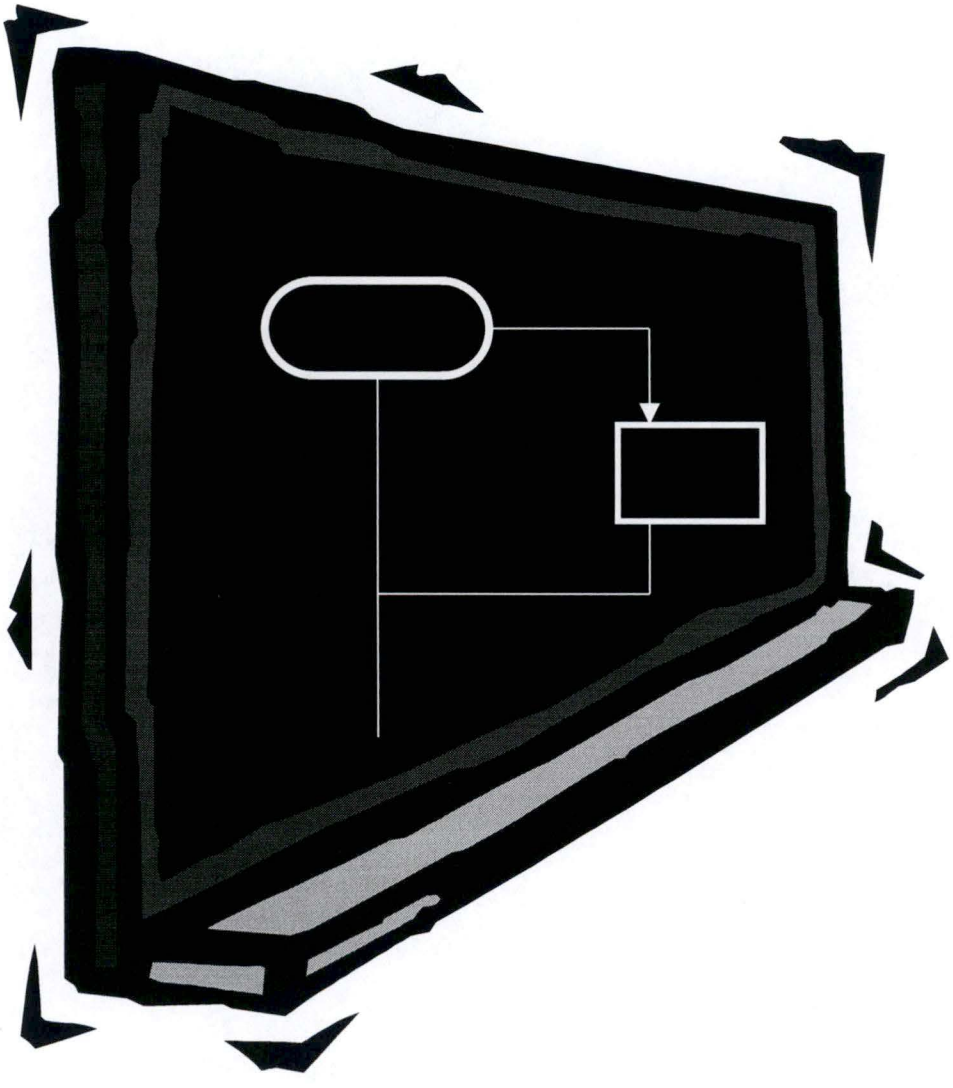
```
begin
  filltab(t,l);
  ispalindrome(t,l,b);
  if b then
    write 1
  else
    write 0
  fi
end.
```

Annexe E

Manuel de l'utilisateur apprenant

J'APPRENDS L'ALGORITHMIQUE

MANUEL UTILISATEUR À DESTINATION DE L'APPRENANT



- Version Provisoire -

- **Auteurs**

Nathalie Collin, nathalie@collin.com
Sous la Direction du Professeur Naji Habra http://www.info.fundp.ac.be/~nha

TABLE DES MATIÈRES

1. PRÉSENTATION

1.1. Avis au lecteur

- Destinataire

La présente documentation s'adresse à tout utilisateur de l'outil pédagogique « J'apprends l'algorithmique ».

- Suggestion de la démarche de lecture de la documentation

La présente documentation propose un scénario d'exécution. Nous proposons dès lors au lecteur de lire la documentation dans un ordre séquentiel. Dès la maîtrise du scénario d'exécution, les utilisateurs pourront d'eux-mêmes parcourir l'outil d'apprentissage dans l'ordre qui leur sied.

L'utilisateur trouvera une copie des écrans complets de l'outil à la fin du manuel. Par souci de lisibilité, nous n'avons inséré que des images réduites aux éléments qui s'avéraient concernés par la rubrique dans laquelle ils figurent.

- Type de logiciel éducatif

« J'apprends l'algorithmique » appartient à la catégorie des simulateurs méthodologiques et diagnostics c'est-à-dire qu'il s'agit d'un logiciel qui permet aux utilisateurs de comprendre des renseignements au sujet de méthodes de travail ou de modes de fonctionnement.

1.2. Public cible

- Ordre d'enseignement

Etudes supérieures universitaires ou non.

- Classe

Typiquement, les premières candidatures.

- Discipline et programme d'études

Introduction à l'algorithmique, Introduction à la Programmation

1.3. Objectifs pédagogiques

L'outil vise à permettre aux apprenants de visualiser sous divers modes les phases d'exécution d'un algorithme.

L'outil, d'une part, permet l'illustration d'un exposé magistral et d'autre part, fournit un outil manipulable par les étudiants lors de leur phase d'appropriation individuelle de la matière.

Il permet d'animer l'exécution d'algorithmes illustrant les concepts suivants :

- L'affectation.
- La manipulation d'un tableau.
- La notion de procédure
- L'accès en écriture et en lecture dans un fichier

2. ACCÈS À L'OUTIL « J'APPRENDS L'ALGORITHMIQUE »

L'utilisateur ouvre son navigateur internet et tape l'adresse de l'outil « J'apprends l'algorithmique » :
http://www.info.fundp.ac.be/~nha/.../.../japprendslalgorithmique.html
[adresse non encore définie]

Les auteurs de l'outil conseillent les pré-requis suivants :
Un des systèmes d'exploitation ci-dessous

- Windows XP, Windows 2000, Windows NT, Windows 9X, Solaris, Linux, Appel Macintosh.

Un des navigateurs web ci-dessous :

- Netscape 7 ou supérieur
- Internet Explorer 6 ou supérieur
- Mozilla 1.3.1 ou supérieur

Vérifiez que votre navigateur est configuré correctement pour l'exécution des programmes Java.

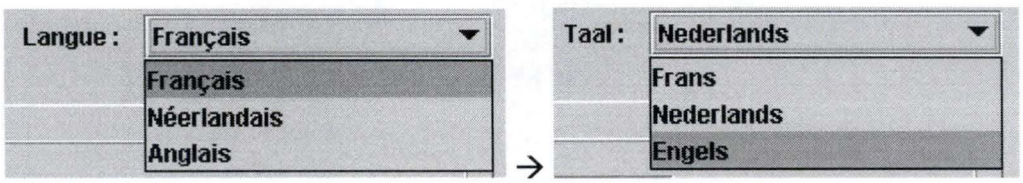
*** Si vous utilisez Netscape :**
Dans le menu : *« Edit » / « Preferences »*, choisissez l'option *« Advanced »*.
Vérifiez ensuite que les options *« Enable Java »* et *« Enable Java Script »* sont activées.

*** Si vous utilisez Internet Explorer :**
Dans le menu *« Tools »-« Internet Options »* et puis *« Advanced »*.
Assurez-vous que l'élément *« Microsoft VM »* existe
Assurez vous aussi que *« Java console enabled »* est activé et que *« JIT compiler for virtual machine enabled »* N'EST PAS activé (ce n'est pas nécessaire, mais cela pourrait être une source d'erreur)

Vous pouvez télécharger le « Java plug-in » à l'adresse suivante :
http://java.sun.com/j2se/1.4.1/download.html
choisissez : Download J2SE (TM) v 1.4.1 – JRE
et suivez les instructions pour son installation selon votre système d'exploitation :
Windows : http://java.sun.com/j2se/1.4.1/jre/install-windows.html,
Linux : http://java.sun.com/j2se/1.4.1/jre/install-linux.html
Macintosh : http://www.apple.com/java/
Solaris : http://www.java.com/en/download/help/solaris_install.jsp

3. CHOIX DE LA LANGUE

L'utilisateur peut d'emblée choisir la langue qui lui convient. L'outil devient alors un outil dans la langue choisie.



L'utilisateur peut changer la langue de l'interface à n'importe quel moment durant l'utilisation de l'outil.

4. PRÉSENTATION GÉNÉRALE DE L'OUTIL

L'outil s'articule sur 4 onglets.

Le premier onglet « Données/Interactions » permet de choisir l'algorithme que l'on désire voir s'exécuter et en donne une brève description en termes de données, de pré-conditions et de post-conditions.

Le deuxième onglet « Algorithme en action » permet de voir toutes les données relatives à l'exécution de l'algorithme choisi : les instructions de l'algorithme en langage algorithmique, l'invariant, l'organigramme, la console des entrées/sorties et l'état des données en mémoire.

Le troisième onglet « Rapport » permet à l'utilisateur de visualiser et d'imprimer les différentes actions qu'il a effectuées dans l'outil.

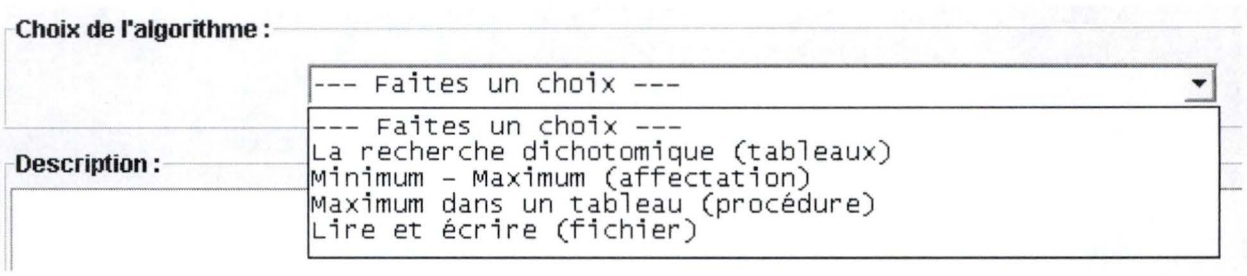
Le quatrième onglet « Info » propose une aide à l'utilisateur sous forme de rubriques et l'informe sur les propriétés de l'outil « J'apprends l'Algorithmique ».

5. SCÉNARIO D'EXÉCUTION D'UN ALGORITHME

ONGLET 1

5.1. Choix de l'algorithme

Dans le premier onglet, l'utilisateur choisit l'algorithme qu'il désire voir s'animer.

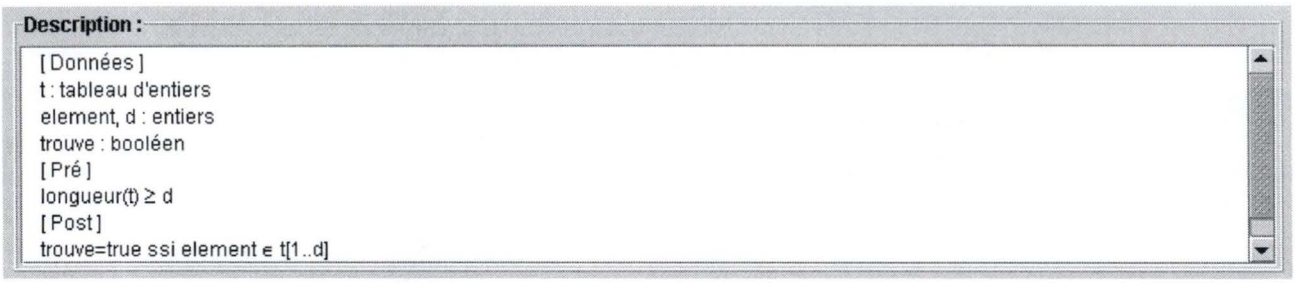


Afin de guider l'utilisateur, l'intitulé de l'algorithme choisi s'affiche en tête de l'outil de sorte que lorsque l'utilisateur change d'onglet, il garde toujours à l'esprit quel algorithme est en cours d'exécution.

Lorsque le choix de l'algorithme est effectué, sa description s'affiche dans la partie « Description ».

5.2. Description de l'algorithme

La description de l'algorithme se présente sous la forme de données, pré-conditions et post-conditions.



5.3. Données/Interactions de l'algorithme [non encore disponible]

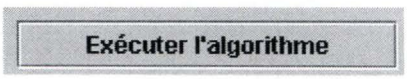
Néant. Non implémenté.

ONGLET 2

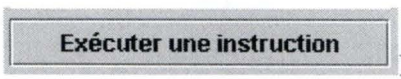
5.4. Exécution de l'algorithme

Lorsque l'algorithme a été choisi, l'utilisateur a la possibilité de l'exécuter. Il peut soit l'exécuter complètement, soit en exécuter que l'instruction courante. Il peut également défaire l'exécution de l'instruction précédente.

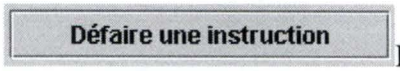
L'utilisateur a quatre boutons à sa disposition :



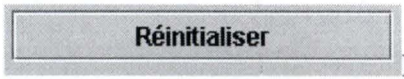
Exécute l'algorithme complètement



Exécute l'instruction courante de l'algorithme



Défait l'instruction précédemment exécutée



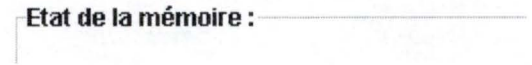
Réinitialise l'algorithme, c'est-à-dire se place sur la première instruction.

Ces quatre actions ont pour effet d'afficher et d'animer les rubriques suivantes :

Le code de l'algorithme est la suite d'instructions en langage algorithme qui le composent.

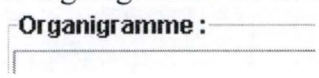
La console symbolise les interactions entrée/sortie de l'algorithme.

L'état de la mémoire affiche les données en mémoire et les valeurs qu'elles prennent en cours d'exécution de l'algorithme.



L'invariant lié à l'algorithme.

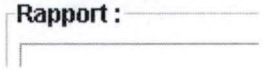
L'organigramme lié à l'algorithme.



ONGLET 3

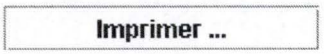
5.5. Historique des opérations

Toutes les données relatives au choix et à l'exécution d'un algorithmes sont notées séquentiellement dans un rapport



Le contenu du rapport peut être effacé en cliquant sur le bouton :

Le rapport peut être imprimé en cliquant sur le bouton :

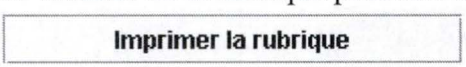


ONGLET 4

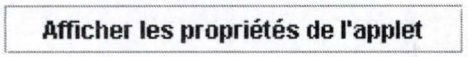
5.6. Rubriques d'aide

L'utilisateur peut choisir une rubrique d'aide.

Le contenu de la rubrique peut être imprimée en cliquant sur le bouton :



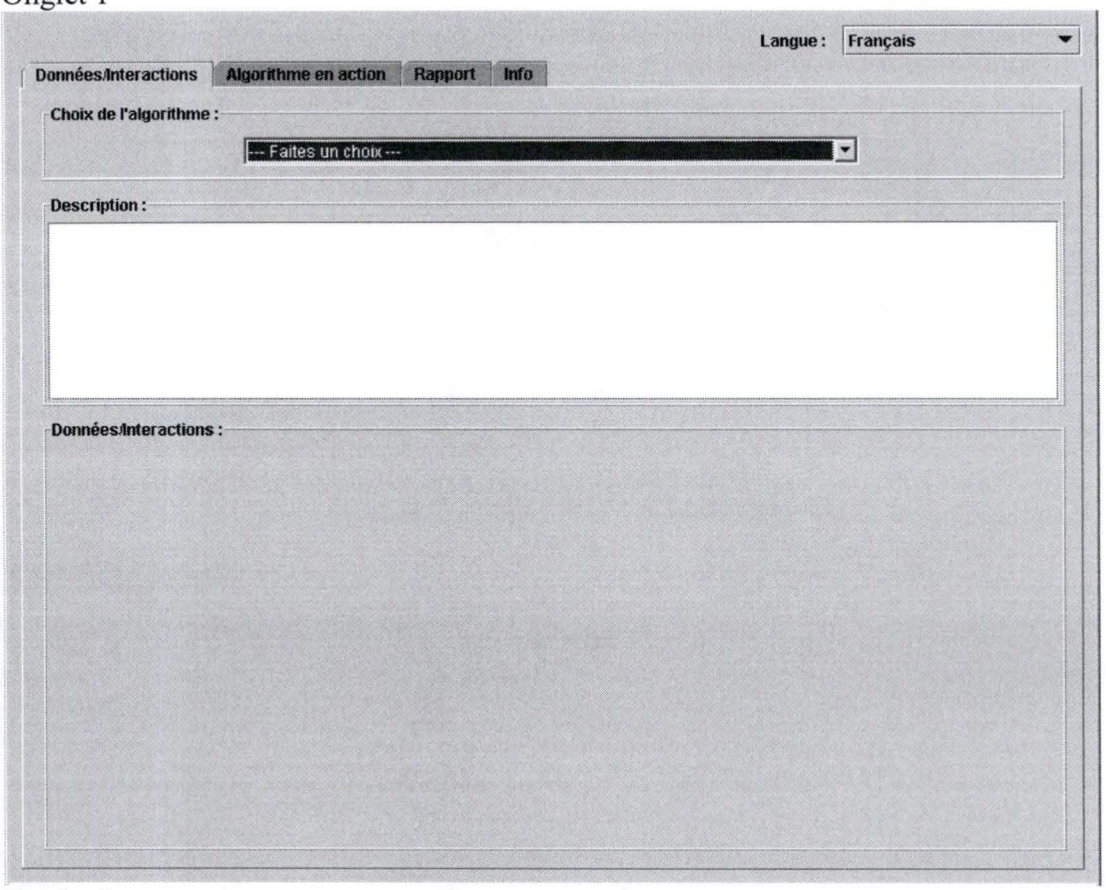
Les propriétés relatives à l'outil peuvent être affichées en cliquant sur le bouton :



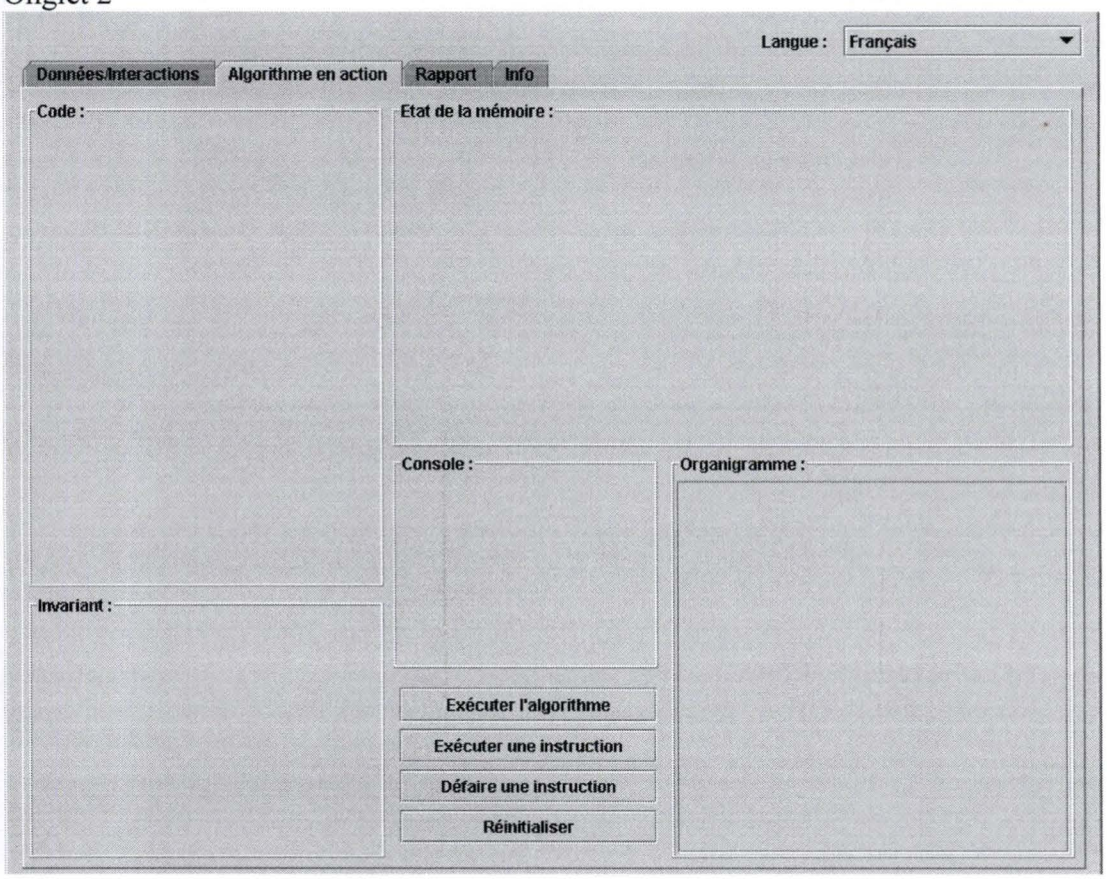
Elles peuvent être imprimées via le bouton 'Imprimer la rubrique'.

6. ECRANS

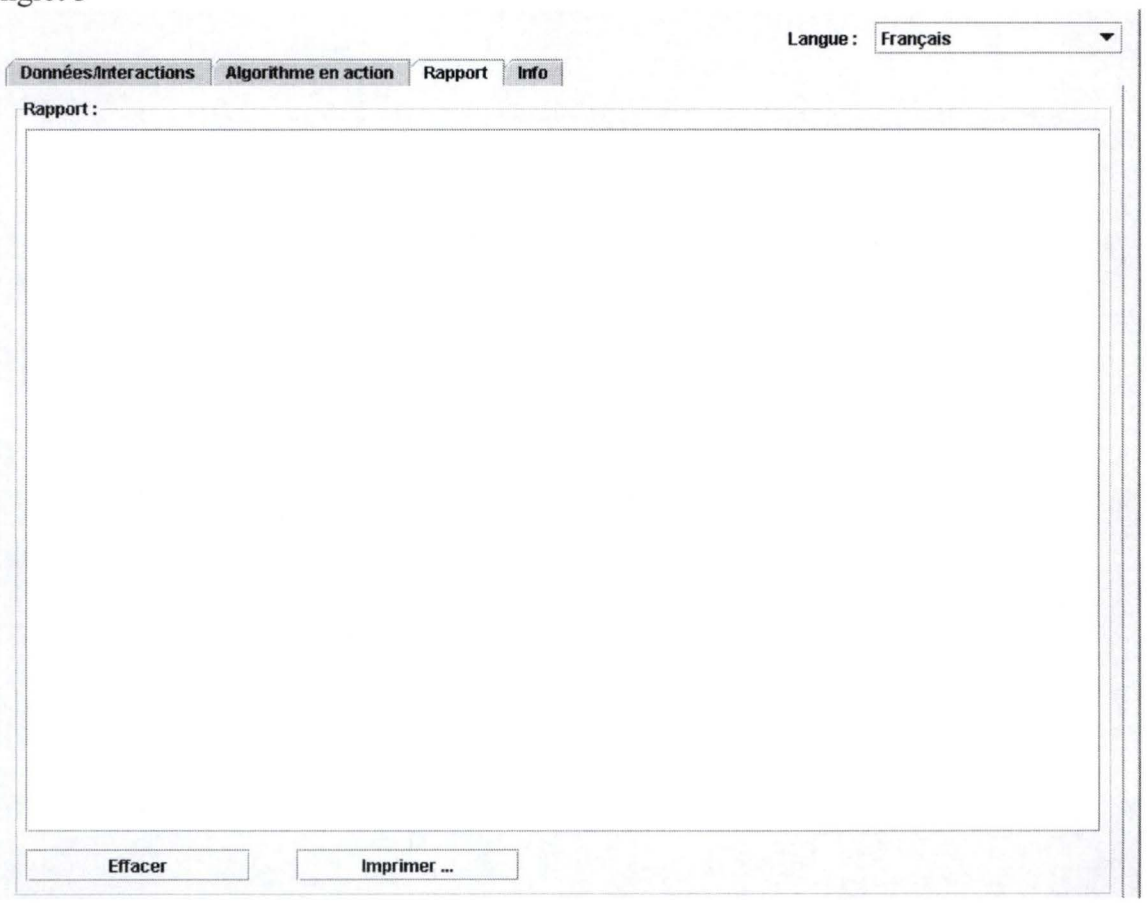
Onglet 1



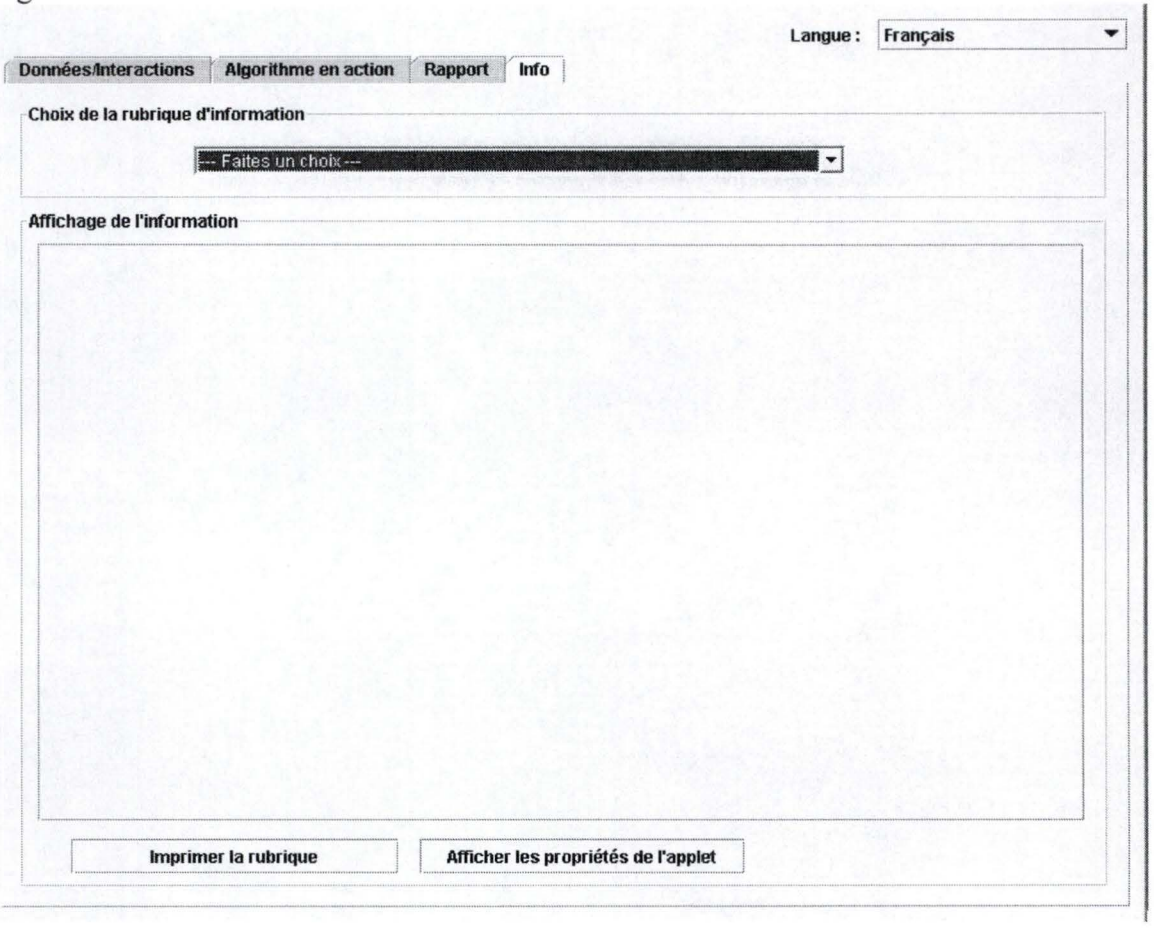
Onglet 2



onglet 3



onglet 4

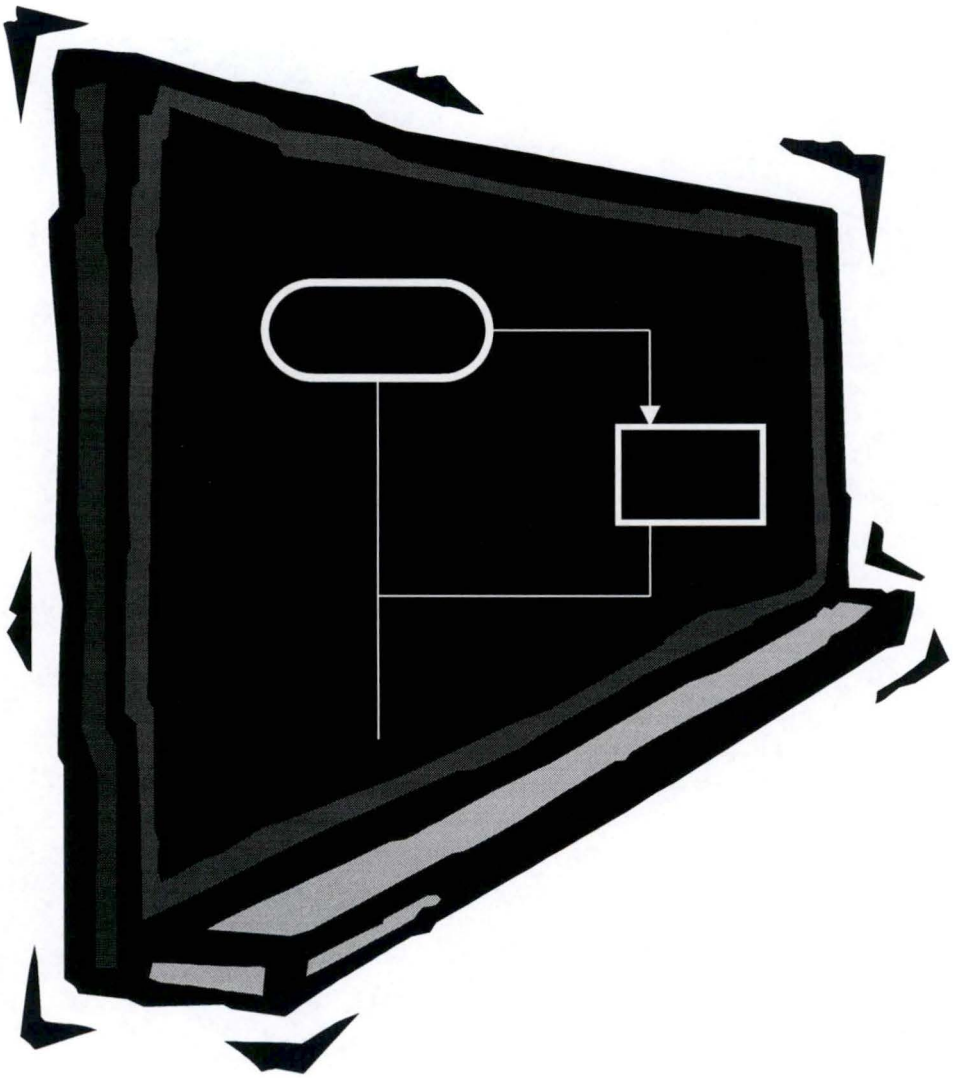


Annexe F

Manuel de l'utilisateur traducteur

J'APPRENDS L'ALGORITHMIQUE

MANUEL UTILISATEUR À DESTINATION DU TRADUCTEUR



- Version Provisoire -

- **Auteurs**

Nathalie Collin, nathalie@collin.com
Sous la Direction du Professeur Naji Habra http://www.info.fundp.ac.be/~nha

ENVIRONNEMENT MINIMAL REQUIS

TABLE DES MATIÈRES

1. PRÉSENTATION

1.1. Avis au lecteur

- Destinataire

La présente documentation s'adresse à tout traducteur de l'outil pédagogique « J'apprends l'algorithmique ».

- Type de logiciel éducatif

« J'apprends l'algorithmique » appartient à la catégorie des simulateurs méthodologiques et diagnostics c'est-à-dire qu'il s'agit d'un logiciel qui permet aux utilisateurs de comprendre des renseignements au sujet de méthodes de travail ou de modes de fonctionnement.

1.2. Public cible

- Ordre d'enseignement

Etudes supérieures universitaires ou non.

- Classe

Typiquement, les premières candidatures.

- Discipline et programme d'études

Introduction à l'algorithmique, Introduction à la Programmation

1.3. Objectifs pédagogiques

L'outil vise à permettre aux apprenants de visualiser sous divers modes les phases d'exécution d'un algorithme.

L'outil, d'une part, permet l'illustration d'un exposé magistral et d'autre part, fournit un outil manipulable par les étudiants lors de leur phase d'appropriation individuelle de la matière.

Il permet d'animer l'exécution d'algorithmes illustrant les concepts suivants :

- L'affectation.
- La manipulation d'un tableau.
- La notion de procédure
- L'accès en écriture et en lecture dans un fichier

2. ACCÈS À L'OUTIL « J'APPRENDS L'ALGORITHMIQUE »

L'utilisateur ouvre son navigateur internet et tape l'adresse de l'outil de traduction du logiciel « J'apprends l'algorithmique » :
http://www.info.fundp.ac.be/~nha/.../.../Traduction.html
[adresse non encore définie]

Les auteurs de l'outil conseillent les pré-requis suivants :
Un des systèmes d'exploitation ci-dessous

- Windows XP, Windows 2000, Windows NT, Windows 9X, Solaris, Linux, Appel Macintosh.

Un des navigateurs web ci-dessous :

- Netscape 7 ou supérieur
- Internet Explorer 6 ou supérieur
- Mozilla 1.3.1 ou supérieur

Vérifiez que votre navigateur est configuré correctement pour l'exécution des programmes Java.

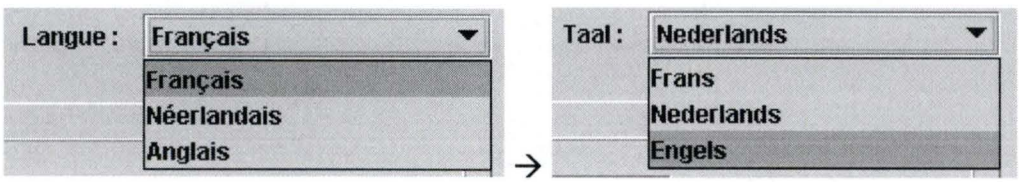
*** Si vous utilisez Netscape :**
Dans le menu : *« Edit » / « Preferences »*, choisissez l'option *« Advanced »*.
Vérifiez ensuite que les options *« Enable Java »* et *« Enable Java Script »* sont activées.

*** Si vous utilisez Internet Explorer :**
Dans le menu *« Tools »-« Internet Options »* et puis *« Advanced »*.
Assurez-vous que l'élément *« Microsoft VM »* existe
Assurez vous aussi que *« Java console enabled »* est activé et que *« JIT compiler for virtual machine enabled »* N'EST PAS activé (ce n'est pas nécessaire, mais cela pourrait être une source d'erreur)

Vous pouvez télécharger le « Java plug-in » à l'adresse suivante :
http://java.sun.com/j2se/1.4.1/download.html
choisissez : Download J2SE (TM) v 1.4.1 – JRE
et suivez les instructions pour son installation selon votre système d'exploitation :
Windows : http://java.sun.com/j2se/1.4.1/jre/install-windows.html,
Linux : http://java.sun.com/j2se/1.4.1/jre/install-linux.html
Macintosh : http://www.apple.com/java/
Solaris : http://www.java.com/en/download/help/solaris_install.jsp

3. CHOIX DE LA LANGUE

L'utilisateur peut d'emblée choisir la langue qui lui convient. L'outil devient alors un outil dans la langue choisie.



L'utilisateur peut changer la langue de l'interface à n'importe quel moment durant l'utilisation de l'outil.

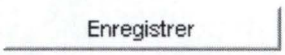
4. UTILISATION DE L'OUTIL

4.1. Ajout d'une langue

L'utilisateur choisi la langue de départ dans la liste des langues existantes de l'outil

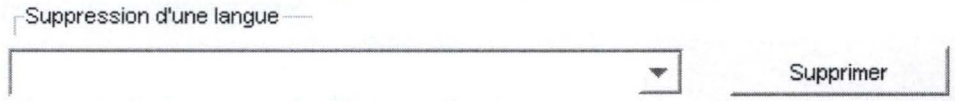
Il choisit ensuite la langue vers laquelle il désire traduire l'outil

Il exécute ensuite la traduction dans le tableau correspondant et clique ensuite sur le bouton :



4.2. Suppression d'une langue

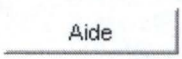
L'utilisateur choisit la langue de l'outil qu'il veut supprimer dans la liste.



Cette langue ne peut en aucun cas être une langue de base (allemand, anglais, français ou néerlandais). Il clique ensuite sur le bouton 'supprimer'.

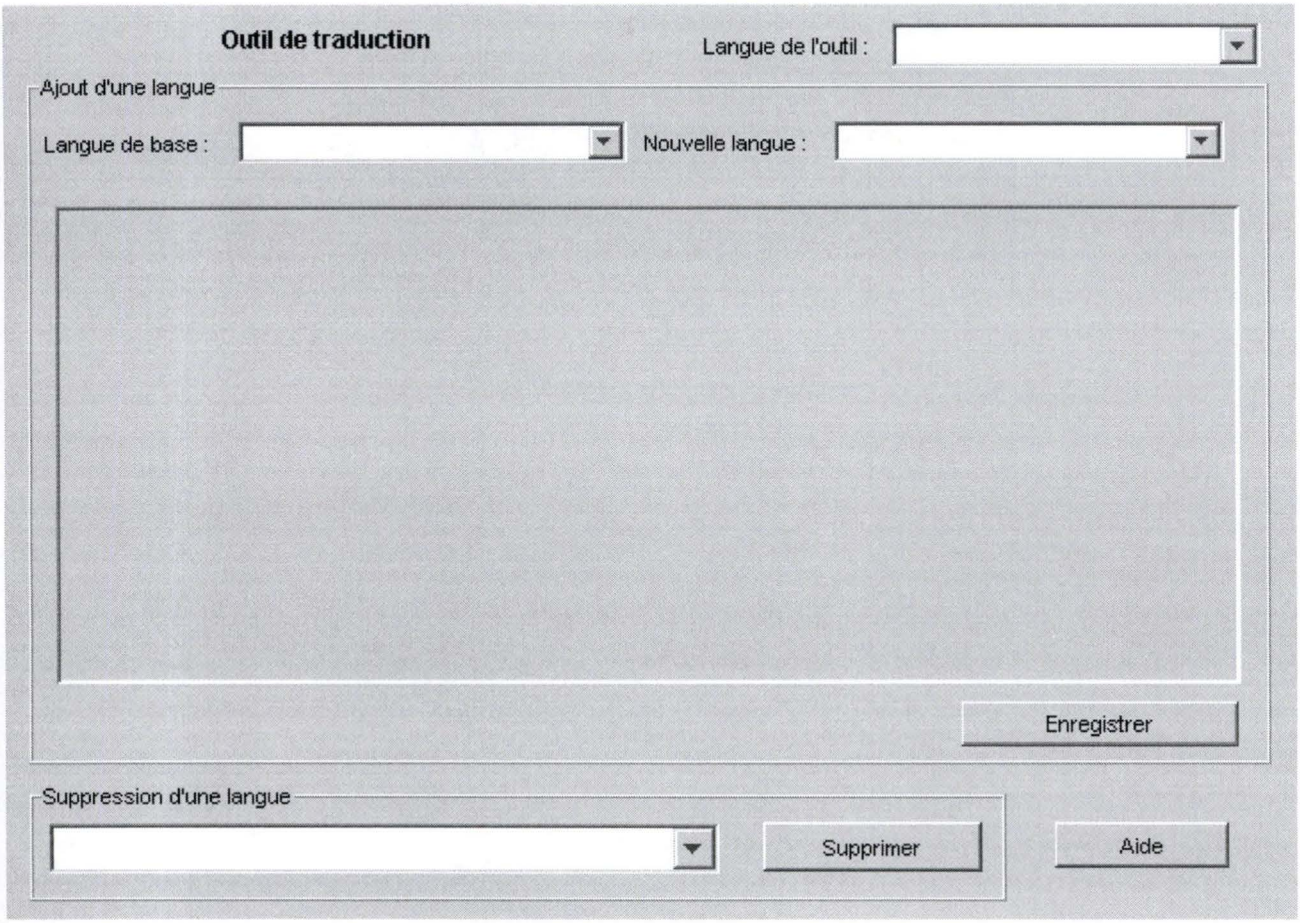
4.3. Aide

En cliquant sur le bouton :



l'utilisateur obtient de l'aide.

5. ECRAN


Outil de traduction
Langue de l'outil :
Ajout d'une langue
Langue de base :
Nouvelle langue :
Enregistrer
Suppression d'une langue
Supprimer
Aide

Annexe G

Spécification lexicale de LSD^{03}

```
1  package Lsd03Translator.lexerparser;
2
3  ////////////////////////////////////////////////
4  // Fichier de spécification pour JFLex        //
5  // ------------------------------------------ //
6  // Nom du fichier       : lsd03.lex           //
7  // Auteur               : nathalie@collin.com //
8  // Date de création     :  7 juin 2003        //
9  // Date de modification : 15 juin 2003        //
10 ////////////////////////////////////////////////
11
12 import java.io.* ;
13 import java.util.* ;
14 import java.text.* ;
15 import java_cup.runtime.* ;
16
17
18 /* ********************** */
19 /* Section Code Utilisateur */
20 /* ********************** */
21
22 %%
23
24 /* ************************************ */
25 /* Section des options et des déclarations */
26 /* ************************************ */
27
28 %{
29    String currentString ;
30
31    private int numberOfLines = 1;
32
33    // just addded
34    private java_cup.runtime.Symbol token(int token_class,Object
      token_value) {
35       return new java_cup.runtime.Symbol(token_class, yychar, yychar +
         yylength(), token_value);
36    }
37
38    private java_cup.runtime.Symbol token(int token_class) {
39       return new java_cup.runtime.Symbol(token_class, yychar, yychar +
         yylength(), null);
40    }
41
42 %}
43
44 %public
45
46 %class       Lsd03Lexer
47 %implements  Lsd03Symbols, CUPTokenLexerInterface
48 %function    nextToken
49 %type        java_cup.runtime.Symbol
50
51 %line
52 %char
53 %cup
54
55 %eofval{
56   return token(Symbol.EOF);
57 %eofval}
58
59
60 %state COMMENT
61 %state STRING
62
63 digit      = [0-9]
64 number     = 0|[1-9]{digit}*
```



```
65 letter     = [A-Za-z]
66 newLine    = \r|\n|\r\n
67 whiteSpace = [ \t\f\b\ ]
68
69 %%
70
71 /* ***************************** */
72 /* Section des règles lexicales */
73 /* ***************************** */
74
75 <YYINITIAL> {
76
77    /* blancs : on ne fait rien */
78    {whiteSpace}   { }
79
80    /* on compte les lignes */
81    {newLine}      { ++numberOfLines; }
82
83    /* ponctuation */
84    ";"            { return token(Symbol.SEMICOLON); }
85    ","            { return token(Symbol.COMMA    ); }
86    "."            { return token(Symbol.DOT      ); }
87
88    /* commentaires */
89    "{"            { yybegin(COMMENT); }
90
91    /* chaîne de caractères */
92    \"             { currentString = new String();
93                     yybegin(STRING);
94                   }
95
96    /* opérations relationnelles */
97    "<"            { return token(Symbol.LT ); }
98    "<="           { return token(Symbol.LE ); }
99    ">"            { return token(Symbol.GT ); }
100   ">="           { return token(Symbol.GE ); }
101   "="            { return token(Symbol.EQ ); }
102   "<>"           { return token(Symbol.NE ); }
103
104   [aA][nN][dD]   { return token(Symbol.AND); }
105   [oO][rR]       { return token(Symbol.OR ); }
106   "!"            { return token(Symbol.NOT); }
107
108   /* parenthèses */
109   "("            { return token(Symbol.LPAREN); }
110   ")"            { return token(Symbol.RPAREN); }
111
112   /* opérateurs arithmétiques */
113   "+"            { return token(Symbol.PLUS  ); }
114   "-"            { return token(Symbol.MINUS ); }
115   "*"            { return token(Symbol.TIMES ); }
116   "/"            { return token(Symbol.DIVIDE); }
117
118   /* affectation */
119   ":="           { return token(Symbol.ASSIGN); }
120
121   /* program */
122   [pP][rR][oO][gG][rR][aA][mM]     { return token(Symbol.PROGRAM
      ); }
123
124   /* déclaration */
125   [vV][aA][rR]                     { return token(Symbol.VAR
      ); }
126   [cC][oO][nN][sS][tT]             { return token(Symbol.CONST
      ); }
127   ":"                              { return token(Symbol.COLON
      ); }
```

```
128
129     /* types */
130     [iI][nN][tT][eE][gG][eE][rR]      { return token(Symbol.INTEGER
        ); }
131     [bB][oO][oO][lL][eE][aA][nN]      { return token(Symbol.BOOLEAN
        ); }
132     [cC][hH][aA][rR]                  { return token(Symbol.CHAR
        ); }
133     [aA][rR][rR][aA][yY]              { return token(Symbol.ARRAY
        ); }
134     "["                               { return token(Symbol.LEFTCROCH
        ); }
135     "]"                               { return token(Symbol.RIGHTCROCH
        ); }
136     ".."                              { return token(Symbol.FROMTO
        ); }
137     [oO][fF]                          { return token(Symbol.OF
        ); }
138     //[nN][eE][gG][aA][tT][iI][vV][eE]   { return
        token(Symbol.NEGATIVE); }
139     //[pP][oO][sS][iI][tT][iI][vV][eE]   { return
        token(Symbol.POSITIVE); }
140
141     /* valeurs */
142     [nN][uU][lL][lL]                  { return token(Symbol.NULL
        ); }
143     [tT][rR][uU][eE]                  { return token(Symbol.TRUE
        ); }
144     [fF][aA][lL][sS][eE]              { return token(Symbol.FALSE
        ); }
145
146     /* begin end */
147     [bB][eE][gG][iI][nN]              { return token(Symbol.BEGIN
        ); }
148     [eE][nN][dD]                      { return token(Symbol.END
        ); }
149
150     /* while do od */
151     [wW][hH][iI][lL][eE]              { return token(Symbol.WHILE
        ); }
152     [dD][oO]                          { return token(Symbol.DO
        ); }
153     [oO][dD]                          { return token(Symbol.OD
        ); }
154
155     /* if then else fi */
156     [iI][fF]                          { return token(Symbol.IF
        ); }
157     [tT][hH][eE][nN]                  { return token(Symbol.THEN
        ); }
158     [eE][lL][sS][eE]                  { return token(Symbol.ELSE
        ); }
159     [fF][iI]                          { return token(Symbol.FI
        ); }
160
161     /* for */
162     [tT][oO]                          { return token(Symbol.TO
        ); }
163
164     /* I/O */
165     [rR][eE][aA][dD]                  { return token(Symbol.READ
        ); }
166     [wW][rR][iI][tT][eE]              { return token(Symbol.WRITE
        ); }
167
168
169     /* skip -> ne rien faire */
```



```
170     [sS][kK][iI][pP]                  { }
171
172     /* procédures */
173     [p][r][o][c][e][d][u][r][e]       { return token(Symbol.PROCEDURE);
        }
174
175
176     {number}                          { return token(Symbol.INTVALUE,
        new Integer(yytext())); }
177     {letter}({letter}|{digit})*       { return token(Symbol.ID, yytext
        ()); }
178
179     .                                 { throw new Error(
        "Illegal character <"+yytext()+">"); }
180 }
181
182 <COMMENT> {
183     /* on sort des commentaires */
184     "}"                               { yybegin(YYINITIAL); }
185     /* on compte les lignes */
186     {newLine}                         { ++numberOfLines; }
187 }
188
189 <STRING> {
190
191     /* on compte les lignes */
192     {newLine}                                 { ++numberOfLines; }
193
194     /* il faut concaténer tout ce que l'on rencontre */
195     ({letter}|{whiteSpace}|{newLine})+        { currentString =
        currentString.concat(yytext()); }
196
197     /* on sort de chaîne des caractères */
198     \"                                        { System.out.println(
        currentString);
199                                                 yybegin(YYINITIAL);
200                                                 return token(Symbol.
                                                    STRINGVALUE,
                                                    currentString);
201                                               }
202 }
203
204 /* No token was found for the input so through an error */
205 [^]    { throw new Error("Illegal character <" + yytext() + ">"); }
206
207
208
209
```

Annexe H

Spécification syntaxique de LSD^{03}

```
1  package Lsd03Translator.lexerparser ;
2
3  import java_cup.runtime.* ;
4  import Lsd03Translator.treenode.* ;
5
6  ////////////////////////////////////////////
7  // Specification file for JavaCUP         //
8  // -------------------------------------- //
9  // Filename          : lsd03.cup          //
10 // Author            : nathalie@collin.com //
11 // Creation Date     : 7 juin 2003        //
12 // Modification Date : 30 juillet 2003    //
13 ////////////////////////////////////////////
14
15 /* ********* */
16 /* User Code */
17 /* ********* */
18
19 /* ---------------Declarations Section-------------*/
20 action code {:
21 :};
22
23 parser code {:
24
25    protected CUPTokenLexerInterface ctli ;
26
27    public Lsd03Parser(CUPTokenLexerInterface ctli) {
28        this.ctli = ctli;
29    }
30
31 :};
32
33 init with {:
34 :};
35
36 scan with {:
37    return ctli.nextToken();
38 :};
39
40 /* --------Section de déclaration des Terminaux et des
   Non-Terminaux-------- */
41
42 terminal          PROGRAM,
43                   BEGIN,
44                   END,
45
46                   WHILE,
47                   DO,
48                   OD,
49
50                   IF,
51                   THEN,
52                   ELSE,
53                   FI,
54
55                   TO,
56
57                   INTEGER,
58                   BOOLEAN,
59                   CHAR,
60                   ARRAY,
61                   FROMTO,
62                   OF,
63                   LEFTCROCH,
64                   RIGHTCROCH,
65
66                   READ,
```



```
67                   WRITE,
68
69                   PROCEDURE,
70
71                   ASSIGN,
72
73                   PLUS,
74                   MINUS,
75                   TIMES,
76                   DIVIDE,
77                   MOD,
78                   UMINUS,
79
80                   NULL,
81                   TRUE,
82                   FALSE,
83
84                   LT,
85                   LE,
86                   GT,
87                   GE,
88                   EQ,
89                   NE,
90
91                   AND,
92                   OR,
93                   NOT,
94
95                   SEMICOLON,
96                   LPAREN,
97                   RPAREN,
98                   COMMA,
99                   DOT,
100                  COLON,
101
102                  CONST,
103                  VAR;
104
105 terminal Integer  INTVALUE;
106 terminal String   STRINGVALUE,
107                  ID;
108
109 non terminal AssignmentStatementNode      assignmentStatement;
110 non terminal BlockNode                    block;
111 non terminal ComparisonNode               comparison;
112 non terminal DeclarationNode              declaration;
113 non terminal DeclarationSNode             declarationS;
114 non terminal ElementNode                  element;
115 non terminal ExpressionNode               expression;
116 non terminal ExpressionSNode              expressionS;
117 non terminal ForStatementNode             forStatement;
118 non terminal IdentifierNode               identifier;
119 non terminal IdentifierSNode              identifierS;
120 non terminal IfStatementNode              ifStatement;
121 non terminal OperatorNode                 weakOperator;
122 non terminal OperatorNode                 strongOperator;
123 non terminal ParameterSNode               parameterS;
124 non terminal ProgramNode                  program;
125 non terminal ReadStatementNode            readStatement;
126 non terminal RelationNode                 relation;
127 non terminal StatementNode                statement;
128 non terminal StatementSNode               statementS;
129 non terminal TermNode                     term;
130 non terminal WhileStatementNode           whileStatement;
131 non terminal WriteStatementNode          writeStatement;
132
133
```

```
/* ----------Section de précédence et d'associativité des Terminaux--------- */

/*
  La précédence des non-terinaux peut être déclarée ici.  Si on définit
  la précédence ici, on ne doit plus s'en inquiéter dans la grammaire.

  Règle :
  La ligne la plus basse aura toujours une précédence plus forte que la ligne
  avant elle.
*/

/* le 'else' va avec le 'then' le plus proche*/
precedence nonassoc  EQ, NE, GT, LT, GE, LE;
precedence left      PLUS, MINUS;
precedence left      TIMES, DIVIDE, MOD;
precedence left      UMINUS, LPAREN;


/* ------------------------------Section GRAMMAIRE------------------------ */

/**
 * Convention :
 *  - les mots en majuscules sont des terminaux
 *  - les mots en minuscules sont des non-terminaux
 */

start with program;

program              ::=   block:b
                           {:
                              try {
                                 RESULT = new ProgramNode(b) ;
                              } catch(SemanticErrorException see) {
                                 System.err.println(see.getMessage())
                                 ;
                                 RESULT = null ;
                                 throw see ;
                              }
                           :}
                           ;

block                ::=   PROGRAM identifier:id SEMICOLON declarationS:ds BEGIN statementS:ss END DOT
                           {:
                              // que faire de id ?
                              RESULT = new BlockNode(ds,ss) ;
                           :}
                           ;

declarationS         ::=   declaration:d
                           {:
                              RESULT = new DeclarationSNode(d) ;
                           :}
                           |
                           declarationS:ds declaration:d
                           {:
                              RESULT = new DeclarationSNode(ds,d) ;
                           :}
                           ;

statementS           ::=   statement:s
                           {:
                              RESULT = new StatementSNode(s) ;
```



```
                           :}
                           |
                           statementS:ss statement:s
                           {:
                              RESULT = new StatementSNode(ss,s) ;
                           :}
                           ;

declaration          ::=   INTEGER identifierS:ids SEMICOLON
                           {:
                              RESULT = new IntegerDeclarationNode(ids)
                              ;
                           :}
                           |
                           BOOLEAN identifierS:ids SEMICOLON
                           {:
                              RESULT = new BooleanDeclarationNode(ids)
                              ;
                           :}
                           |
                           CHAR identifierS:ids SEMICOLON
                           {:
                              RESULT = new StringDeclarationNode(ids)
                              ;
                           :}
                           |
                           ARRAY
                           {:
                              // to do
                           :}
                           |
                           PROCEDURE identifier:id block:b SEMICOLON
                           {:
                              RESULT = new
                              ProcedureDeclarationNode(id,null,b);
                           :}
                           |
                           PROCEDURE identifier:id LPAREN parameterS:p
                           RPAREN block:b SEMICOLON
                           {:
                              RESULT = new
                              ProcedureDeclarationNode(id,p,b);
                           :}
                           ;

statement            ::=   readStatement:s
                           {:
                              RESULT = s;
                           :}
                           |
                           writeStatement:s
                           {:
                              RESULT = s;
                           :}
                           |
                           assignmentStatement:s
                           {:
                              RESULT = s;
                           :}
                           |
                           ifStatement:s
                           {:
                              RESULT = s;
                           :}
                           |
                           forStatement:s
                           {:
```

```
256                                 RESULT = s;
257                              :}
258                              |
259                              whileStatement:s
260                              {:
261                                 RESULT = s;
262                              :}
263                              ;
264
265  readStatement          ::=  READ LPAREN identifierS:ids RPAREN
     SEMICOLON
266                              {:
267                                 RESULT = new ReadStatementNode(ids);
268                              :}
269                              ;
270
271  writeStatement         ::=  WRITE LPAREN expressionS:es RPAREN
     SEMICOLON
272                              {:
273                                 RESULT = new WriteStatementNode(es);
274                              :}
275                              ;
276
277  assignmentStatement    ::=  identifier:id ASSIGN expression:e SEMICOLON
278                              {:
279                                 RESULT = new AssignmentStatementNode(id,
                                    e);
280                              :}
281                              ;
282
283  ifStatement            ::=  IF comparison:c THEN statementS:then_ss FI
284                              {:
285                                 RESULT = new IfStatementNode(c,then_ss,
                                    null);
286                              :}
287                              |
288                              IF comparison:c THEN statementS:then_ss
                                 ELSE statementS:else_ss FI
289                              {:
290                                 RESULT = new IfStatementNode(c,then_ss,
                                    else_ss);
291                              :}
292                              ;
293
294  forStatement           ::=  TO expression:e DO statementS:ss OD
295                              {:
296                                 RESULT = new ForStatementNode(e,ss);
297                              :}
298                              ;
299
300  whileStatement         ::=  WHILE comparison:c DO statementS:ss OD
301                              {:
302                                 RESULT = new WhileStatementNode(c,ss);
303                              :}
304                              ;
305
306  parameterS          ::=   identifierS:ids
307                              {:
308                                 try {
309                                    RESULT = new ParameterSNode(ids);
310                                 } catch (SemanticErrorException see) {
311                                    System.err.println(see.getMessage());
312                                    RESULT = null;
313                                    throw see;
314                                 }
315                              :}
316                              ;
```



```
317
318  identifierS            ::=   identifier:id
319                              {:
320                                 RESULT = new IdentifierSNode(id);
321                              :}
322                              |
323                              identifierS:ids COMMA identifier:id
324                              {:
325                                 RESULT = new IdentifierSNode(ids,id);
326                              :}
327                              ;
328  expressionS         ::=   expression:e
329                              {:
330                                 RESULT = new ExpressionSNode(null,e);
331                              :}
332                              |
333                              expressionS:es COMMA expression:e
334                              {:
335                                 RESULT = new ExpressionSNode(es,e);
336                              :}
337                              ;
338
339  comparison             ::=  expression:eL relation:r expression:eR
340                              {:
341                                 RESULT = new ComparisonNode(eL,r,eR);
342                              :}
343                              ;
344
345  expression             ::=  term:t
346                              {:
347                                 RESULT = new ExpressionNode(t);
348                              :}
349                              |
350                              expression:e weakOperator:wo term:t
351                              {:
352                                 RESULT = new ExpressionNode(e,wo,t);
353                              :}
354                              ;
355
356  term                   ::=  element:e
357                              {:
358                                 RESULT = new TermNode(e);
359                              :}
360                              |
361                              term:t strongOperator:so element:e
362                              {:
363                                 RESULT = new TermNode(t,so,e );
364                              :}
365                              ;
366
367  element                ::=  INTVALUE:iv
368                              {:
369                                 RESULT = new IntegerElementNode(iv.
                                    intValue());
370                              :}
371                              |
372                              identifier:id
373                              {:
374                                 RESULT = id;
375                              :}
376                              |
377                              LPAREN expression:e RPAREN
378                              {:
379                                 RESULT = new ExpressionElementNode(e);
380                              :}
381                              ;
382
```

```
383 identifier              ::=   ID:id
384                               {:
385                                  RESULT = new IdentifierNode(id);
386                               :}
387                               ;
388
389 relation                ::=   EQ
390                               {:
391                                  RESULT = new RelationNode(Symbol.EQ);
392                               :}
393                               |
394                               LE
395                               {:
396                                  RESULT = new RelationNode(Symbol.LE);
397                               :}
398                               |
399                               LT
400                               {:
401                                  RESULT = new RelationNode(Symbol.LT);
402                               :}
403                               |
404                               GT
405                               {:
406                                  RESULT = new RelationNode(Symbol.GT);
407                               :}
408                               |
409                               GE
410                               {:
411                                  RESULT = new RelationNode(Symbol.GE);
412                               :}
413                               |
414                               NE
415                               {:
416                                  RESULT = new RelationNode(Symbol.NE);
417                               :}
418                               ;
419
420 weakOperator            ::=   PLUS
421                               {:
422                                  RESULT = new OperatorNode(Symbol.PLUS);
423                               :}
424                               |
425                               MINUS
426                               {:
427                                  RESULT = new OperatorNode(Symbol.MINUS);
428                               :}
429                               ;
430
431 strongOperator          ::=   TIMES
432                               {:
433                                  RESULT = new OperatorNode(Symbol.TIMES);
434                               :}
435                               |
436                               DIVIDE
437                               {:
438                                  RESULT = new
                                   OperatorNode(Symbol.DIVIDE);
439                               :}
440                               ;
441
442
443
```