# Un cédérom pour Scheme
# Chacun son entraineur, un entraineur pour tous

Anne Brygoo* Anne.Brygoo@ufr-info-p6.jussieu.fr
Titou Durand* Titou.Durand@ufr-info-p6.jussieu.fr
Pascal Manoury** Pascal.Manoury@pps.jussieu.fr
Christian Queinnec*** Christian.Queinnec@lip6.fr
Michèle Soria*** Michele.Soria@lip6.fr
UPMC – UFR d'informatique – * DRI, ** PPS, *** LIP6

## Résumé

Cet article relate une expérimentation pédagogique dans laquelle nous avons conçu un cédérom pour accompagner un cours d'initiation à l'informatique. Ce cédérom procure un environnement de développement enrichi d'exercices et d'auto-évaluations. De façon autonome c'est-à-dire non connectés à Internet, les étudiants peuvent étudier leur cours, écrire des programmes et les soumettre pour obtenir une appréciation immédiate de leur travail. Les réponses sont accumulées puis transmises dans une base de données centrale où elles peuvent être analysées et assurer ainsi un suivi personnalisé.
**Mots-clés :** Pédagogie multimédia et télétutorat, Expérimentation et retour d'usage, Environnements d'apprentissage.

We describe a teaching experiment where an introductory course to Computer Science is accompanied by a computerized training engine. This whole engine relies on the existence of an interpreter of the taught programming language that allows us to offer quizzes as well as exercises with some automatic marking facility. Students may then perform their homework with an immediate feedback without being connected to the Internet. However students' answers are eventually gathered in a central database where they may be analyzed thus providing the means for "personal coaching".
**Keywords :** Educational methods in multimedia and tutoring, Experiments and feedbacks, Learning environment.

## Introduction

À l'Université Pierre et Marie Curie (Paris 6), le premier module du DEUG-MIAS[1] est consacré à l'étude du « processus d'évaluation ». Le langage support de cet enseignement est (un sous-ensembke de) Scheme. Le processus d'évaluation est le dispositif qui permet de faire exécuter par un ordinateur un texte – par exemple un programme – en vue d'obtenir un résultat. C'est un concept central de l'informatique. Il est à la base de la compréhension de ce que sont les évaluateurs (interprètes ou compilateurs), y compris ceux qui sont inclus dans les applicatifs complexes (tableurs par exemple) et, de façon générale de ce qu'est la programmation.

L'enseignement de la programmation peut être caractérisé comme étant l'enseignement d'un savoir-faire avec des savoirs associés (concepts et méthodes de travail), des connaissances sur les règles de l'art (par exemple, conventions d'écriture et commentaires nécessaires) et l'appropriation d'outils logiciels utilisables pour fabriquer des programmes (par exemple, éditeur de texte, interpréteur ou compilateur).

Traditionnellement, cet enseignement est divisé en séances de cours, de travaux dirigés et de travaux pratiques (ou travaux encadrés sur machine). Les cours permettent d'exposer les savoirs et, un peu, les méthodes. Les travaux dirigés aident les étudiants à s'approprier ce savoir, en vue de son utilisation lors de l'écriture de programmes. Pour ce faire, afin que les étudiants comprennent les objectifs de leur travail, une activité importante est la mise en perspective des exercices par rapport aux savoirs et aux savoir-faire enseignés. Enfin, le savoir-faire est évalué, lors des séances de travaux pratiques – par les enseignants, mais surtout par les étudiants eux-mêmes – lorsqu'ils se confrontent au verdict de la machine. Notons que c'est la partie de l'enseignement préférée des étudiants, par ses aspects ludiques et concrets.

Sous l'influence de l'un d'entre nous [Que00a, CQS00] qui avait conçu un cédérom pour l'enseignement du langage C en licence, nous avons rédigé les exercices en vue d'une médiatisation sur cédérom. Cette année, un enseignement utilisant spécifiquement ce cédérom a été proposé à 2 groupes de travaux-dirigés (sur les 24 groupes en tout). Les étudiants qui ont participé à cette expérience dite « semi-présentielle à distance » (SPAD) étaient moins présents sur le campus (en ce qui concerne l'informatique, 2 heures par semaine au lieu de 3h30), le temps ainsi gagné devant être consacré à un travail personnel plus important que celui des étudiants traditionnels. Ils étaient tous volontaires et nous leur avons garanti la possibilité d'intégrer à tout moment les groupes traditionnels. Cette promesse, et la nécessité de passer les mêmes examens, impliquent que leur progression soit celle des autres étudiants. De fait, nous avons observé deux transferts inverses.

Pédagogiquement, tout en utilisant un « environnement interactif d'apprentissage avec ordinateur », nous avons voulu que l'aspect présentiel dans l'enseignement SPAD soit ex-

[1] Mthématiques, Informatique et Applications aux Sciences

trêmement structurant, les enseignants devant guider et accompagner les étudiants : cela constitue – à notre connaissance – une expérience unique. Ainsi, ce n'est pas l'étudiant qui définit son parcours, c'est l'équipe pédagogique qui impose, et vérifie, le travail de la semaine. Les enseignants sont là aussi pour aider les étudiants en répondant à leurs questions, soit sous une forme électronique (forum, courriel), soit, *de visu*, lors des 2 heures hebdomadaires de rendez-vous pédagogiques. En revanche, l'aspect « à distance » permet aux étudiants de gérer leur temps en leur permettant de travailler chez eux, quand ils veulent, tout en étant reliés à leurs enseignants.

Didactiquement, l'enseignement médiatisé comprend trois éléments : un cours, des auto-évaluations et des exercices. Le cours est un texte essentiellement linéaire agrémenté de quelques digressions du genre « autre vision » ou « pour en savoir plus » sous forme de liens hypertexte. Les auto-évaluations qui accompagnent le cours rendent possible l'appropriation du savoir en vue de son utilisation dans le savoir-faire. Les exercices proposés sont de même nature que ceux utilisés dans les séances classiques de travaux dirigés et de travaux pratiques et le système sait actuellement dire si la production de l'étudiant répond au cahier des charges et à certaines règles de l'art. Idéalement, le système devrait pouvoir aussi

- aider les étudiants dans l'emploi des outils logiciels (édition, mise au point, tests),
- guider les étudiants dans leur recherche d'une solution,
- obliger les étudiants à réfléchir aux objectifs de l'exercice.

L'activité de programmation est une activité dynamique et, si l'on veut évaluer le travail des étudiants, le résultat final (les sources des programmes) ne suffit pas. Aussi, un système d'apprentissage de la programmation doit mémoriser, et traiter, la trace de l'activité des apprenants. Ces traces sont également utiles pour évaluer l'enseignement lui-même en permettant de détecter des difficultés didactiques non prévues au départ, des explications peu claires ou des enchaînements d'exercices non pertinents.

Le présent article décrit, en section 1, l'architecture du cédérom. Dans la section 2, nous exposons l'utilisation du cédérom et les résultats de notre expérience. Enfin, en section 3, nous indiquons nos perspectives immédiates, en termes de logiciels, de contenus et d'organisation de l'enseignement.

# 1 Architecture du cédérom

Le cédérom est le « polycopié électronique » ([Que00a]) du cours. Il contient les notes cours en HTML et en PDF qui offrent des liens vers les séries d'auto-évaluations, des exercices et quelques autres documents secondaires. Il contient également les exécutables composant l'environnement de programmation retenu dans le cadre de ce cours, à savoir DrScheme de l'université de Rice [FCF+01].

Le contenu du cédérom[2] est disponible en ligne. Ce site, souvent réactualisé, est utilisé dans les salles de TP et dans les salles d'ordinateurs en libre service. Un instantané de ce site est diffusé, sur cédérom, aux étudiants au début du semestre.

[2] http://www.infop6.jussieu.fr/cederoms/VideoScm2001

Le cédérom a été conçu pour permettre aux étudiants de travailler sur leur machine personnelle de façon autonome c'est-à-dire non connectés de façon permanente à Internet. Nous pensons en effet, à l'inverse de la tendance prônée par plusieurs plates-formes pédagogiques, que, pour au moins la décennie à venir, les étudiants ne seront pas tous, tout le temps, connectés (en particulier dans les contrées défavorisées) et qu'il faut donc fournir des moyens de travail autonome.

Le travail autonome implique que les étudiants puissent répondre aux défis posés et avoir un retour immédiat évaluant les solutions qu'ils proposent sans connexion à Internet. Nous nous sommes donc intéressés à la génération automatique de ces réponses. Par ailleurs, comme nous souhaitons pouvoir suivre la progression des étudiants, les solutions qu'ils proposent sont archivées localement sur leur machine et transmises de façon asynchrone via Internet (ou même via disquette) aux serveurs de l'université lorsqu'ils se connectent (ou se déplacent à l'université).

La figure 1 présente l'architecture générale de notre cédérom. Les étudiants sont en face de deux outils.

- **Un navigateur** permet de lire le cours et de répondre aux auto-évaluations. Dans ce dernier cas, les questions (en HTML) sont engendrées (de façon invisible pour l'étudiant) par une tâche écrite en Scheme et exécutée dans DrScheme (on utilise pour ce faire, un serveur ouèbe inclus dans DrScheme). Les réponses que fournit l'étudiant à ces auto-évaluations sont analysées par des programmes Scheme qui synthétisent la réponse appropriée (toujours en HTML) et enchaînent ou non sur les questions suivantes.
Le navigateur est également l'outil de communication utilisé pour consulter les nouvelles fraîches du site ouèbe du cours ou pour interagir avec le forum associé à l'enseignement.

- **Un environnement de programmation**, ici DrScheme, procure un interpréteur interactif, un éditeur de programme, un metteur au point, bref tous les outils usuels d'un environnement de développement. À cet environnement nous avons ajouté des compléments proposant des exercices et évaluant les solutions soumises par les étudiants.

La collecte asynchrone des réponses aux auto-évaluations et aux exercices enrichit une base de traces au rythme du travail des étudiants. Côté enseignant, un simple navigateur permet de visualiser l'état d'avancement d'un groupe d'étudiants (qui a fait, et comment, quel exercice ou auto-évaluation), l'état d'avancement d'un étudiant (quelles appréciations a-t'il obtenues pour les exercices ou auto-évaluations auxquels il s'est essayé) ou le contenu de ses solutions successives (qu'a-t'il écrit ? qu'avait-il écrit juste auparavant ?).

Outre le suivi des étudiants, l'intérêt des traces enregistrées est de pouvoir livrer des statistiques de succès ou d'échec sur les différentes questions ou auto-évaluations.

Ces deux activités, auto-évaluations et exercices, sont décrites plus en détail ci-après.

## 1.1 Auto-évaluations

Les auto-évaluations regroupées en séries sont accessibles depuis le document de cours. On peut les ranger en deux ca-

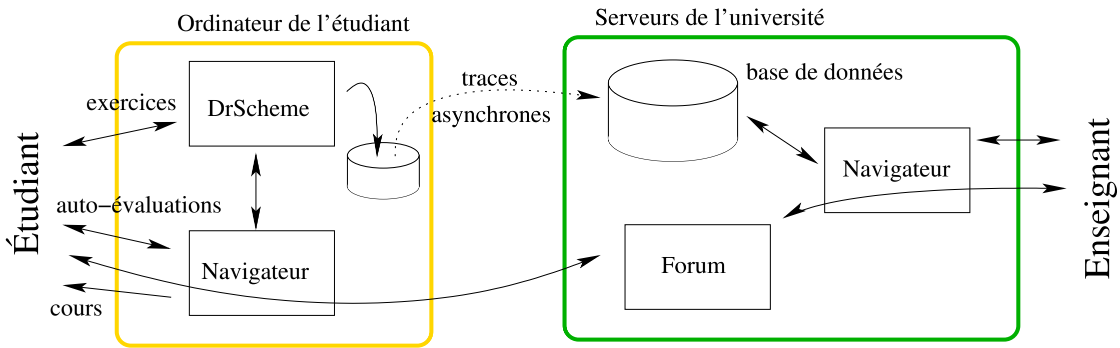



FIG. 1 – Déploiement du cédérom

tégories principales :
- les auto-évaluations dont les réponses ne sont pas analysées et pour lesquelles une réponse standard est fournie (pouvant comporter des liens vers des éléments du cours) ;
- les auto-évaluations dont les réponses sont analysées soit au moyen d'expressions rationnelles soit au moyen de l'évaluateur de Scheme.

Quelles qu'elles soient, toutes les réponses sont archivées.

Techniquement, chaque série d'auto-évaluations est un fichier (un programme) écrit en Scheme évalué par le serveur ouèbe inclus dans l'environnement de programmation. Le programme Scheme définit les questions à poser ainsi que le dialogue les enchaînant.

Les questions ont été standardisées afin que leur apparence et leur correction soient régulières. Parmi les questions corrigées on trouve :
- Quelle est la valeur de l'expression ... ?
- Écrivez un programme dont la valeur est ...
- Quel est le type de la fonction ... ?
- Définissez une fonction répondant à la spécification ..., (cf. figure 2).
- etc.

De manière interne, les questions sont représentées par des objets munis de quelques méthodes : l'une permet d'engendrer la page HTML de l'énoncé, une autre d'apprécier une réponse, une troisième et dernière d'engendrer un commentaire sur la solution. La standardisation des questions passe par la définition d'une bibliothèque de fonctions de génération de questions. Ainsi, pour engendrer la question « quelle est la valeur de `(list (+ 1 2))` ? » suffit-il d'écrire :

```
(evaluation-question
  "q-ab-list3"      ;un identifiant unique pour cette question
  '(list (+ 1 2)))
```

Toute la génération d'HTML, la procédure de vérification (qui lira une chaîne de caractères – depuis la requête HTTP – correctement parenthésée, la convertira en une S-expression, l'évaluera pour la comparer à la valeur attendue) ainsi que la génération de commentaire élogieux ou pas est entièrement assuré par la fonction `evaluation-question`. Le premier argument de cette fonction est un nom unique permettant d'identifier la question notamment dans les traces.

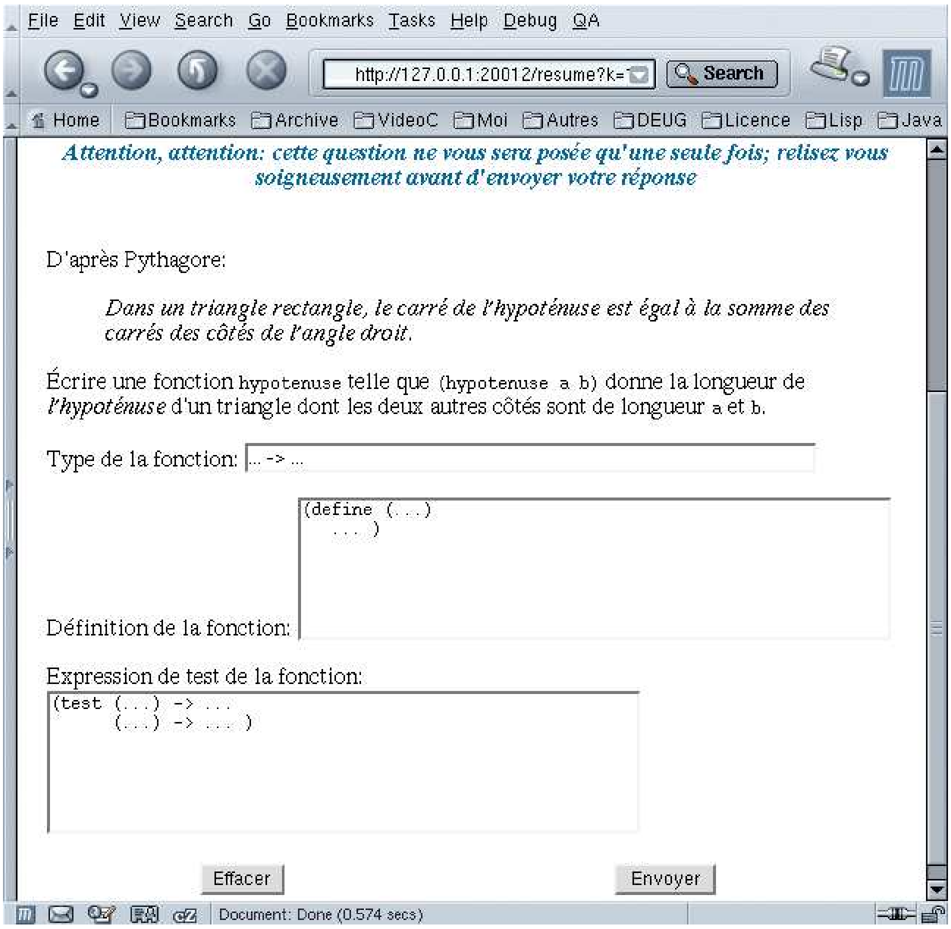
Une fois que les questions sont créées, elles sont assemblées en dialogues. Les dialogues sont créés par des combinateurs composant les questions élémentaires. Ces combinateurs permettent, entre autres, de :
- poser une question jusqu'à obtention d'une bonne réponse,
- poser une question jusqu'à obtention d'une bonne réponse mais au plus un certain nombre de fois,
- poser un groupe de questions simultanées au plus une fois,
- poser une question sans laisser voir la correction (c'est le mode « interrogation écrite » où, de plus, le dialogue interdit de fournir plus d'une réponse à toute question).

La décomposition entre questions et dialogues permet d'agencer (possiblement dynamiquement) d'anciennes questions en de nouveaux dialogues, de procurer des raccourcis ou parcours détaillés suivant la performance des étudiants. Les dialogues reposent sur l'usage de continuations [Que00b] qui permettent, outre la gestion d'un dialogue linéaire ausi bien que l'utilisation des fonctionnalités de retour en arrière et de clonage de fenêtre des navigateurs.

## 1.2 Exercices

Un exercice correspond à un énoncé suivi d'une série ordonnée de questions visant chacune à écrire et tester une ou plusieurs fonctions décrites dans l'énoncé. Pour pouvoir mener à bien cette tâche, l'étudiant interagit principalement avec l'environnement de programmation : il lit l'énoncé (dans le navigateur présent dans l'environnement de programmation), écrit ses fonctions, les évalue, les teste à la main dans la fenêtre d'interaction ou écrit des tests dans la fenêtre d'édition afin de les automatiser (cf. figure 3 fenêtre à gauche en bas). Lorsque son travail lui paraît achevé, il appuie sur le bouton « Tester » pour soumettre sa solution qui est alors évaluée. Un commentaire est émis en retour (cf. figure 3 à droite) qui apparaît dans le navigateur interne de l'environnement. Une note est produite dont le principal rôle est de permettre, au-delà d'un seuil déterminé par l'exercice, de révéler une ou plusieurs solutions.

Un point qui nous semble important est que la structure procurant les exercices est greffée comme une adjonction à un réel environnement de programmation, ce que ne peut égaler en confort d'utilisation un environnement à base d'HTML et



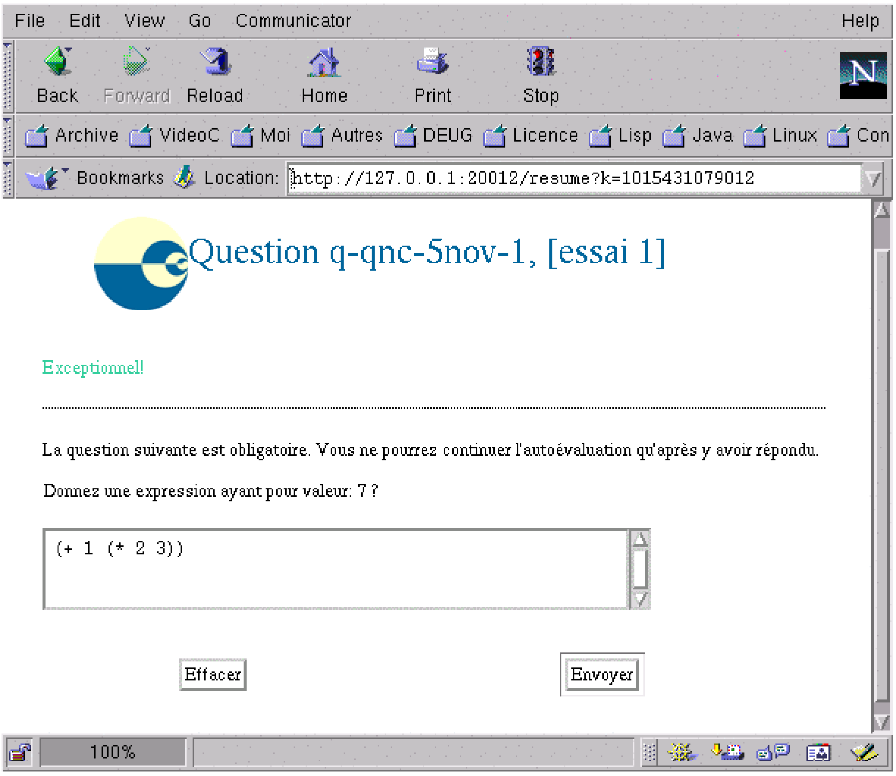



FIG. 2 – Deux exemples d'auto-évaluations

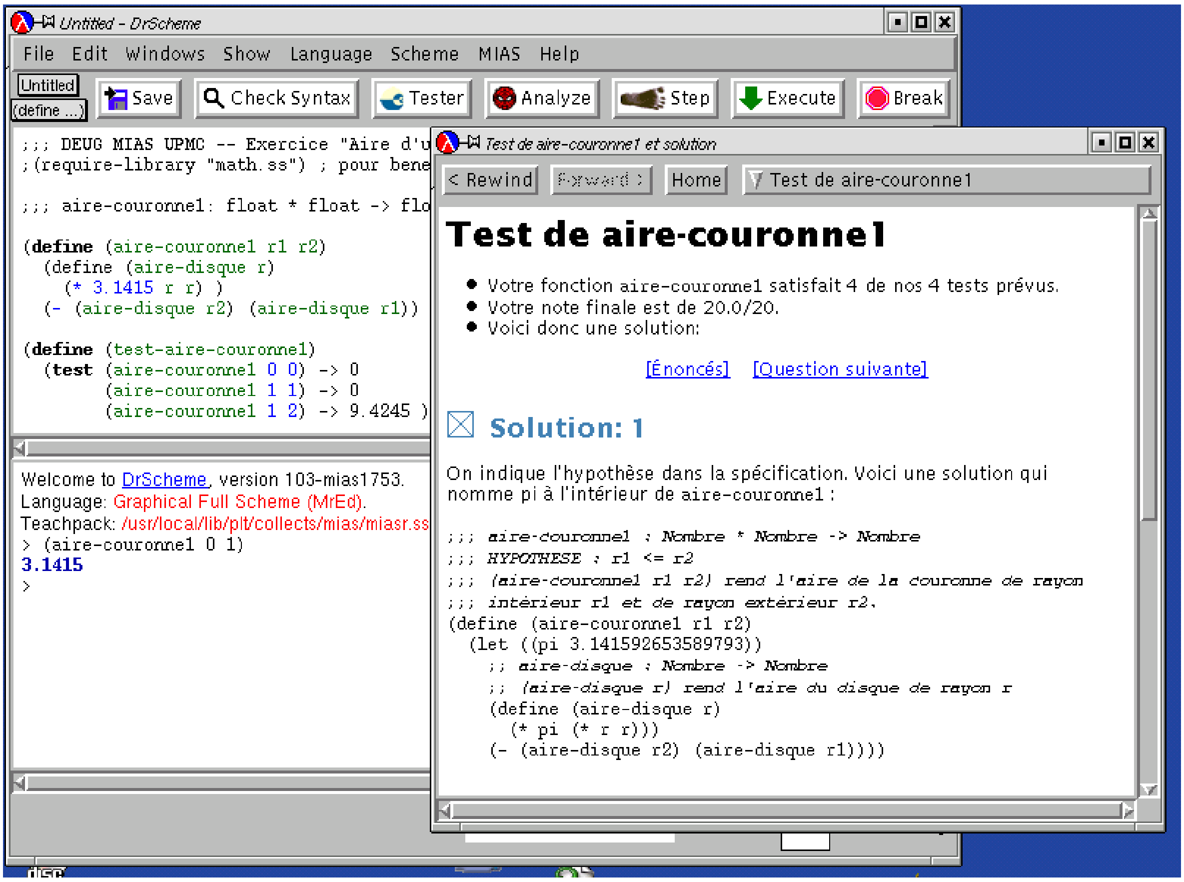



FIG. 3 – Un exemple d'exercice

de navigateur.

Que les exercices soient appréciés (notés) fonde l'autonomie du système. Lorsqu'une fonction $f$ est demandée, l'étudiant doit non seulement écrire une fonction Scheme nommée `f` mais aussi un prédicat de test associé nommé `test-f`. Ce prédicat ne renvoie Vrai que si la fonction `f` satisfait tous les tests unitaires prévus par l'étudiant. La note est calculée à partir des éléments suivants :

1. Les fonctions (de l'étudiant) `f` et `test-f` sont-elles correctement définies ?
2. La fonction `f` de l'étudiant satisfait-elle le prédicat `test-f` de l'étudiant ? On vérifie également que la fonction `f` est au moins invoquée une fois par `test-f`.
3. Pour chaque solution de l'enseignant, disons $f_T$, la fonction $f_T$ satisfait-elle la fonction `test-f` de l'étudiant ? Ceci assure que les tests ne sont pas biaisés en fonction de la réponse de l'étudiant.
4. La fonction `f` de l'étudiant satisfait-elle le prédicat `test-f`$_T$ de l'enseignant ?

Les fonctions des enseignants sont fournies sous forme compilée et elles ne peuvent être utilisées que par les programmes de notation. Les pages HTML contenant les solutions sont cryptées sur le cédérom et ne sont décryptées que par le serveur ouèbe inclus dans l'environnement de programmation.

**Remarques** La version du système actuellement en ligne contient 380 auto-évaluations et 58 exercices totalisant 245 fonctions différentes à écrire.

Un point important qui a facilité la réalisation de ce système est la simplicité du langage enseigné et surtout la simplicité de son évaluateur [Que94] que concrétise la fonction `eval` apte à prendre dynamiquement un programme, le compiler/interpréter et renvoyer son résultat quasiment instantanément. Cette simplicité permet l'instrumentation de cette fonction afin de : vérifier les caractéristiques du sous-ensemble linguistique utilisé par les étudiants ; compter les nombres d'appels aux constructeurs, sélecteurs, appels de fonction pour estimer la complexité de la solution ; analyser statiquement le code fourni pour le typer ou vérifier quelques autres caractéristiques, etc.

# 2 Utilisation du cédérom pour l'apprentissage

Ce cédérom a été conçu pour faciliter l'acquisition par l'étudiant de connaissances en Scheme et de savoir-faire de base en programmation. Dans cette section nous présentons d'abord la logique qui a présidé à la construction pédagogique du cédérom, puis nous expliquons l'utilisation qui en a réellement été faite par les étudiants et par les enseignants.

## 2.1 Intentions

Tout en étant à la « pointe du progrès technologique » ( :+), nous partons du principe que l'apprentissage comporte la lecture du cours, l'exécution de « gammes »(les auto-évaluations), et la résolution d'exercices complets.

L'armature au niveau du contenu du cédérom est donc le cours. Ce cours est organisé classiquement en une quarantaine de sections, d'environ trois pages chacune. Bien quáccessible *via* un navigateur hypertexte, if est fait pour être lu linéairement, de la première à la dernière section.

### 2.1.1 Auto-évaluations

À la fin de chaque section se greffent trois séries d'auto-évaluations ciblées sur les notions introduites dans la section. Les deux premières séries visent à donner à l'étudiant un outil pour contrôler immédiatement sa compréhension du cours. La troisième série permet de replacer les connaissances acquises dans la perspective (et les buts) du cours tout entier. Chaque notion introduite est systématiquement soumise à des questions : questions simples, questions équivalentes formulées de différentes façons, questions pièges.

Dans les premières séries d'auto-évaluations dites « Exercices d'assouplissement », les réponses de l'étudiant sont analysées et corrigées en utilisant l'interprète Scheme, mais volontairement il n'est pas proposé de solution. Un petit nombre de questions sont bloquantes, au sens où, tant que l'étudiant n'a pas donné une réponse jugée correcte, il est lui impossible de passer à la question suivante. Chaque réponse analysée donne lieu à un commentaire sur la nature de la réponse mais ces réponses ne sont pas notées : à cette étape, nous préférons concentrer nos efforts sur la transmission et l'acquisition de « noyaux » durs de connaissance en Scheme plutôt que sur l'appréciation de cette acquisition.

Dans les séries d'auto-évaluations dites « Questions de cours » l'étudiant est invité à formuler (par écrit) sa compréhension des notions introduites. Les réponses ne sont pas analysées en direct, mais des pointeurs vers les parties du cours relatives aux points discutés sont fournis en retour. Dans les séries d'auto-évaluations dites « Pour approfondir » les réponses ne sont pas non plus analysées.

Toutes les réponses, pour les trois types de séries d'auto-évaluations, sont enregistrées et laissent des traces qui pourront ensuite faire l'objet d'analyse. Chaque série d'auto-évaluations nécessite environ une demi-heure de travail : les questions sont ciblées sur une seule notion, et l'étudiant est guidé pour contrôler sa compréhension de cette notion, de façon élémentaire puis de plus en plus approfondie.

Un trait remarquable des auto-évaluations est leur aspect « ludique » qui semble influer sur l'apprentissage : les étudiants se mettent « doucement » au travail, guidés par le fil des questions, et ont l'impression d'être hors de l'environnement de programmation. Mais ils s'apercoivent rapidement qu'ils peuvent en parallèle tester et expérimenter ces questions en DrScheme.

### 2.1.2 Exercices

Comparés aux auto-évaluations, les exercices demandent souvent plus de réflexion et de temps de mise en œuvre : ils obligent l'étudiant à se mobiliser pour déterminer et rassembler plusieurs types de connaissances.

Au niveau de la programmation, l'étudiant peut travailler directement dans l'environnement de DrScheme en utilisant la fenêtre d'interaction pour tester et visualiser les résultats d'exécution. Mais il peut aussi utiliser l'environnement dédié du cédérom (dit « environnement MIAS »). Cet environnement implique certaines contraintes (respecter les noms, accompagner chaque fonction d'une fonction de test, ...) mais offre aussi différents avantages :

– enregistrement automatique de toutes les réponses proposées par l'étudiant ;
– appréciation des fonctions proposées sous la forme d'un commentaire et d'une quantité numérique entre 0 et 20 ;
– présentation d'explication et de solutions alternatives lorsque la note précédemment attribuée atteint un certain seuil.

Cette « notation automatique » n'est pas conçue pour évaluer l'étudiant mais plutôt pour permettre l'affichage de solutions documentées. En réponse à chaque question sont proposées en général plusieurs solutions. L'objectif est non seulement de montrer un modèle de résolution et de présentation pour que l'étudiant apprenne par l'exemple, mais aussi de lui faire prendre conscience de la multiplicité des solutions informatiques : différentes façons de résoudre le problème ou d'implanter une fonction.

Par ailleurs, la notation automatique peut aussi constituer une aide pour l'étudiant, qui peut ainsi visualiser sa progression vers une solution acceptable. En revanche, pour l'enseignant, les tests automatiques, s'ils dégrossissent le travail de correction, ne le dispensent pas d'une appréciation qualitative plus fine (lisibilité, efficacité, esthétique).

## 2.2 Observations

Nous avons recueilli les remarques et commentaires des étudiants d'une part au cours du semestre, via le forum, et d'autre part, en fin de semestre via le questionnaire d'évaluation de l'enseignement et la réunion de bilan avec les étudiants SPAD. Mais la principale source d'observation de cet enseignement reste les traces enregistrées en base. Ce sont les observations de ces dernières que nous voulons rapporter ici.

### 2.2.1 Rôle des traces

Les traces servent trois buts concomittants :

– suivi personnalisé de l'étudiant ;
– suivi du groupe d'étudiants ;
– réflexion *a posteriori* sur l'enseignement.

L'étude des traces d'un étudiant permet de voir une partie de son travail (son travail tracé) et les notions pour lesquelles il a eu (ou non) des difficultés. Nous pouvons alors l'aider, quasiment en temps réel, par courriel ou sur le forum. De plus lorsque les traces d'une auto-évaluation ou d'un exercice montrent que, pour un groupe, certaines notions ont été mal assimilées, nous pouvons intervenir lors du rendez-vous pédagogique suivant. Enfin, pour améliorer l'enseignement, un certain nombre de rétro-analyses statistiques s'avèrent précieuses.

### 2.2.2 Auto-évaluations

Lors de l'expérimentation SPAD du premier semestre 2001-2002, nous avons pu recueillir « presque » toutes les traces de 21 étudiants. L'analyse de ces traces a montré que pour chacune des 380 auto-évaluations, il y avait eu de 2 à 113 essais.

Une première analyse rapide montre bien sûr que les questions les plus traitées sont celles des « Exercices d'assouplissement » et les moins traitées celles des séries « Pour approfondir ».

Sur les séries qui font apparaître une suite de questions équivalentes formulées de façons différentes, on peut souvent noter un pic dans le nombre d'essais à la première question, montrant ainsi que les étudiants se sont heurtés à une difficulté, se sont acharnés à la résoudre pour ensuite continuer le train-train des questions qui se suivent et se ressemblent.

D'autres pics dans le nombre d'essais effectués signalent soit une mauvaise formulation (la question donnant lieu à 113 essais comportait une erreur !), soit une difficulté conceptuelle qui nécessiterait le déploiement de questions intermédiaires ou de questions équivalentes formulées de façons différentes.

Lors de la réunion de bilan, il est apparu que les étudiants n'apprécient pas les questions bloquantes et préfèreraient voir la solution apparaître au bout d'un certain nombre d'essais infructueux.

En fait, toutes les réponses des étudiants, et en particulier les réponses *textuelles* pour expliquer leur compréhension, constituent un corpus d'une grande richesse, non encore exploité.

### 2.2.3 Exercices

Le suivi à la trace de leurs premiers pas en programmation nous a permis de distiller au fur et à mesure aux étudiants des conseils et des savoir-faire basiques qui ont accéléré leur progression. Une page récapitulative de conseils, en ligne, passe en revue des exemples de bonnes ou mauvaises présentations et de tournures de programmation à éviter ou à acquérir.

L'expérience a montré que les étudiants ne profitent pas pleinement des solutions données pour les exercices. Ils ne lisent en général pas l'explication proposée, ... sauf lorsqu'ils s'aperçoivent qu'il y en a plusieurs, ce qui n'est pas sans les intriguer.

Leur réflexe principal, lorsqu'ils voient s'afficher une solution, est de supposer que leur fonction est correcte – ce qui est faux puisque l'affichage est déclenché par l'atteinte d'un certain seuil pour la note. Et bien sûr cela peut impliquer des déboires dans les questions suivantes.

# 3 Perspectives

Bien qu'ayant été officiellement fabriqué pour la seule population des étudiants de l'expérience SPAD, le cédérom a profité d'une ambition plus large qui était de rendre également service aux étudiants traditionnels. Nous détaillons dans cette section les améliorations que nous comptons y apporter, tant pour le public initialement visé, que pour l'ensemble des étudiants de DEUG MIAS qui y trouveront ainsi un entraineur prompt, réactif et disponible.

### 3.1 Le cédérom

L'architecture « pédagogique » globale du cédérom (cours, auto-évaluations, exercices) n'est pas appelée à connaître de grands bouleversements. En revanche, sa logistique doit être simplifiée afin d'alléger la procédure d'installation et de fiabiliser la collecte des traces.

En particulier, les logiciels adventices pour la collecte locale et transmission des traces, actuellement écrits en Java, pour des raisons de portabilité, vont être repris en Scheme afin de réduire le nombre de logiciels à installer.

Pour guider l'installation et l'utilisation du cédérom, nous envisageons de réaliser un vidéogramme approprié et de mettre en place durant les premières semaines de la rentrée une permanence fournissant une aide en ligne par courriel ou téléphone.

### 3.2 Auto-évaluations

Les questions d'auto-évaluations sont actuellement organisées en séries constituant un bloc : si une série est abandonnée, elle doit être reprise à partir du début. Les étudiants se sont plaints, à juste titre, de ne pouvoir reprendre une série là où ils l'avaient abandonnée. La prochaine version du cédérom devra corriger ce défaut d'ergonomie.

Les étudiants ont également formulé deux autres remarques sur les auto-évaluations : que des séries d'auto-évaluations thématiques soient disponibles pour leurs révisions ; que, dans tous les cas, les solutions soient fournies.

La demande de génération de séries thématiques d'auto-évaluations est intéressante car elle nous permet de répondre à un autre besoin, émanant des enseignants, et visant à permettre la production de séries particulières d'auto-évaluations à partir d'auto-évaluations « sur étagère ». Cette mutation permettrait l'utilisation du système des séries d'auto-évaluations non seulement comme outil d'aide à l'apprentissage, mais aussi comme outil de contrôle de connaissances. Cela demande d'organiser les auto-évaluations en une base de données, de les annoter et indexer afin qu'elles puissent être retrouvées par thème et par difficulté.Un simple formulaire permettra d'enrichir la base de questions partagées ou de créer une nouvelle série dans un but particulier (révisions, épreuve de contrôle sur machine, etc.)

La seconde demande, qui concerne la publication des solutions, est encore l'objet de discussion entre enseignants. En effet, les auto-évaluations sont des questions élémentaires dont la réponse est souvent une simple citation du cours ou une variation sur un exemple. Répondre favorablement à la demande des étudiants de publier ces solutions ne favorise-t-il pas leur esprit de consommation, repoussant ainsi le moment où ils devront « vraiment s'y mettre » ? Ne doit-on pas plutôt inciter l'étudiant à (se) poser des questions, si vraiment il ne comprend pas, et ce, d'autant plus que les auto-évaluations sont des applications immédiates du cours ?

### 3.3 Exercices

L'usage a très vite montré que, concernant les exercices, plusieurs points sont à améliorer : la « notation » et sa présentation ainsi que la variété du type d'exercices proposés.

La notation est le résultat chiffré de l'appréciation qui permet de déclencher ou non l'affichage des solutions. Le processus est assez frustre pour l'instant et peut être perfectionné notamment par l'explicitation des tests qui ne passent pas. Cette « verbalisation » s'obtiendra par paraphrase de l'évaluation erronée. Ainsi, sans révéler les tests prévus par les enseignants (on ne fournira pas la réponse attendue), on pourra commenter le fait que le code de l'étudiant ne les satisfait pas.

À côté du comportement fonctionnel, nous envisageons de vérifier quelques autres qualités dans le code de l'étudiant. Là encore, ces qualités peuvent être calculées du fait de la malléabilité de l'interprète de Scheme. Nous pensons ajouter :

- la vérification des traits langagiers utilisés par l'étudiant (présence d'un `let`, absence de variables locales, emploi de la fonction `map`, etc.) ;

- le typage des solutions proposées par les étudiants (en utilisant les résultats du typage souple [WC97]) ;

- l'observation expérimentale de la complexité de la solution proposée par l'étudiant par rapport aux complexités des solutions des enseignants ;

- la mesure du taux de couverture des tests réalisés par les étudiants (cet aspect vise à « tester les tests » !).

Plusieurs étudiants ont souhaité pouvoir, lors des révisions, revoir les corrections qu'ils avaient pu obtenir : il faut donc rendre persistant le « droit de voir une solution » (qu'il soit acquis à l'université ou chez soi). Certains souhaiteraient également retrouver les solutions qu'ils avaient proposées et non nécessairement sauvegardées. Ces remarques vont dans le sens de celles formulées à propos des auto-évaluations. Nous y apporterons des réponses analogues. En particulier, et toujours pour améliorer les conditions de révision, nous pensons annoter et indexer les exercices afin de pouvoir proposer des parcours thématiques associant exercices et auto-évaluations.

Enfin, nous projetons d'utiliser un nouveau type d'exercice proposé sous forme de « programmes à trous » que l'étudiant devra compléter ou modifier. Le but d'un tel type d'exercice est de développer chez les étudiants l'apprentissage de la lecture et de la compréhension de programmes qui est une activité aussi formatrice que généralement négligée. Dans le même ordre d'idée, afin d'inciter les étudiants à lire avec plus dáttention les solutions que nous proposons, nous pensons introduire des questions dépendant des solutions précédemment proposées pour, par exemple, les modifier, les comparer, mesurer le temps d'exécution, etc. qui sont, actuellement et commentaires mais que nous imaginons pouvoir mieux les pouvoir les analyser pour en tenir exercices.

En complément du logiciel d'apprentissage et dans la continuation du projet de vidéogramme expliquant l'installation et l'utilisation du cédérom, il nous est paru important de réaliser plusieurs vidéogrammes montrant (et verbalisant) comment s'utilise l'environnement de programmation, comment l'on analyse l'énoncé d'un exercice, comment l'on écrit des programmes, comment on les teste, comment on met au point, pourquoi plusieurs solutions, etc. Cet aspect « cinématographique », tout à fait nouveau pour nous, nous est apparu comme un complément à la fois naturel et nécessaire de l'utilisation des technologies de l'information et de la communication dans l'enseignement.

# 4 Conclusions

L'expérimentation que nous avons rapportée dans cet article ouvre un large champs de pespectives. Nous voudrions, pour conclure souligner seulement quelques points.

La nouveauté et le caractère expérimental de l'utilisation du cédérom pour Scheme dans le cadre du groupe SPAD a forcé un rapport plus étroit avec nos étudiants. Nous avons dûs ainsi être encore plus à l'écoute de leur processus d'apprentissage, ce qui nous a permis d'affiner non seulement le contenu de notre enseignement, mais également sa présentation (électronique, pour améliorer le contenu du cédérom, ou classique, lors des rendez-vous pédagogiques hebdomadaires). De leur côté, les étudiants SPAD, qui ne bénéficiaient pas d'un encadrement traditionnel, ont trouvé dans l'utilisation des outils mis à leur disposition un surcroît de motivation. Au final, l'existence et l'utilisation du cédérom ce sont avérées des composantes indispensables d'un enseignement de type « semi présentiel ».

La mise à l'épreuve des outils fournis par le cédérom et les leçons que nous en avons tirées permettent d'envisager leur utilisation dans le cadre des enseignements plus classiques où leur caractère attractif et leur adéquation avec les modes contemporains d'accés aux connaissances pourront jouer pleinement .

Le site du DEUG MIAS est accessible en :
http://www.infop6.jussieu.fr/deug/2001/mias/mias-a/public/
Le cédérom est, quant à lui, en :
http://www.infop6.jussieu.fr/cederoms/VideoScm2001/

# Références


[BDM+02] Anne Brygoo, Titou Durand, Pascal Manoury, Christian Queinnec, and Michèle Soria. Experiment around a training engine. In *IFIP WCC 2002 – World Computer Congress*, Montreal (Canada), August 2002. IFIP.

[CQS00] Claire Cazes, Christian Queinnec, and Chantal Steinberg. Enseignement du langage C à l'aide d'un cédérom et d'un site – Mise en œuvre et observation. Troyes (France), Atelier TICE, October 2000.

[FCF+01] R. Findler, J. Clements, C. Flanagan, M. Flatt, S. Krishnamurthi, P. Steckler, and M. Felleisen. Drscheme : A programming environment for scheme. *Journal of Functional Programming*, 2001.

[Que94] Christian Queinnec. *Les langages Lisp*. InterÉditions, Paris (France), 1994.

[Que00a] Christian Queinnec. Enseignement du langage C à l'aide d'un cédérom et d'un site – Architecture logicielle. In *Colloque international – Technologie de l'Information et de la Communication dans les Enseignements d'ingénieurs et dans l'industrie – TICE 2000*, pages 93–102, Troyes (France), October 2000. CNED.

[Que00b] Christian Queinnec. The influence of browsers on evaluators or, continuations to program web servers. In *ICFP '2000 – International Conference on Functional Programming*, pages 23–33, Montreal (Canada), September 2000.

[WC97] Andrew K. Wright and Robert Cartwright. A practical soft type system for scheme. *ACM Transactions on Programming Languages and Systems*, 19(1) :87–152, January 1997.