Enseignement du langage C à l'aide d'un cédérom et d'un site Architecture logicielle
Teaching C with a CDROM and a web site – Software Architecture

Christian Queinnec
Université Paris 6 — Pierre et Marie Curie
LIP6, 4 place Jussieu, 75252 Paris Cedex
France – Email: `Christian.Queinnec@lip6.fr`

Résumé

Entre octobre 1999 et janvier 2000, dans le cadre de la licence d'informatique de l'université Paris 6, a été menée une expérience pédagogique consistant à enseigner le langage C à l'aide d'un cédérom. Cet article présente l'architecture du logiciel mis en place pour cette réalisation et principalement la notion de « parcours » qui permet d'agencer des cheminements complexes et réactifs au sein d'une base d'informations découpées en pages.

Mots-clés : Applications de formation sur le réseau et cédérom, Génie logiciel des systèmes de formation, Modélisation et production de contenus, langage C, polycopié électronique.

This year, from October 1999 till January 2000, while teaching the C programming language to a cohort of nearly 300 under-graduate students at the University of Paris 6, we used a CD-ROM and an associated Web site. This paper presents the software architecture of that realization. We introduce the concept of "tracks". A track specifies a precise path, expressed as a program using continuations, through a multitude of pre-existing web pages.

Keywords : Teaching through Web and CD-ROM, Software architecture for learning systems, Content production, C programming language, e-lecture support.

Le 20 mai 1998 a été approuvé, par l'université Paris 6 et avec l'appui de Mme Nicole Bernard, chargée de mission pour les technologies nouvelles, la mise en chantier d'un cédérom destiné aux étudiants de licence d'informatique pour l'apprentissage du langage C. Ce cédérom a été pressé à 1000 exemplaires en septembre 1999 et distribué gratuitement à tous les étudiants de second cycle d'informatique pour lesquels il a servi de « polycopié électronique ». Le dispositif est complété par un site (`http://videoc.lip6.fr/`) régulièrement enrichi et arpenté par jusqu'à une centaine de visiteurs par jour. Nous distinguerons par la suite le cédérom et le site qui, quoique initialement de même contenu, ont des caractéristiques assez différentes compte tenu de la nature de leur support. VideoC est le nom générique de l'ensemble[1].

Cette expérimentation se décline en deux articles, celui-là [CQS00] présente les aspects pédagogiques et leur observation ; celui-ci se focalise sur l'aspect technique de l'architecture informatique du cédérom et du site associé, les difficultés rencontrées, les solutions adoptées et les enseignements que l'on peut en tirer. Nous espérons que cette expérimentation, ici rapportée, permettra à d'autres d'éviter nos erreurs.

La originalité de cette réalisation est la technique d'implantation des « parcours » qui permettent d'agencer un cheminement au sein d'une base d'informations découpées en pages. La thèse est que les enseignants qui ont à prodiguer à de futurs informaticiens un savoir de plus en plus éprouvé, écriront de moins en moins de pages d'informations brutes mais exploiteront en revanche, et de plus en plus, des pages déjà écrites en les assortissant d'un commentaire illustrant le propos du parcours pour lequel ils ont décidé de passer précisément par ces pages. Cette volonté de ré-emploi devrait permettre la sélection naturelle des bonnes pages tout en laissant libre l'imagination de parcours variés et adaptés au public visé comme, par exemple, « C pour les nuls », « C en deux heures pour programmeurs confirmés », « Les différentes formes d'allocation en C », etc.

Cet article est essentiellement technique et s'adresse principalement aux auteurs de sites ou de cédéroms. L'architecture générale de VideoC sera présentée en première section. Au travers des choix concernant l'affichage, l'interprétation des liens, le processus de production des pages et le langage de programmation du logiciel d'accompagnement du cédérom, nous relaterons nos difficultés et nos futurs plans. La notion de parcours fera l'objet de la seconde section et détaillera le concept de continuation et le parcours vu comme un programme. Perspectives et conclusions achèveront ce tour d'horizon des grandes lignes de notre réalisation.

Tous les logiciels produits dans le cadre de cette réali-

[1] Nous utiliserons ce nom pour désigner à la fois le site et le cédérom.

sation sont libres et diffusés sous licence GPL[2].

1 Motivations et architecture

Le cours de C est fondé sur deux idées pédagogiques. C est un assembleur portable de haut niveau ne devant être utilisé que dans des domaines d'emploi précis. C est également une lingua franca pour l'écriture d'interfaces, dont il faut maîtriser et le modèle mémoire et le modèle d'exécution, pour faire communiquer des programmes écrits dans des langages différents. La seconde idée est que tout langage s'accompagne d'une culture régissant son emploi, encourageant certains styles de programmation, fondant son histoire, etc. Tous ces éléments culturels s'expriment par des documents variés (normes, guides de style, textes fondateurs, foire aux questions, programmes bien connus et exemplaires, etc.) Outre de proposer notre cours, nos exercices et les annales de nos examens sur C, nous souhaitions proposer les éléments les plus représentatifs de cette culture sur le cédérom à savoir d'autres cours en français ou en anglais, des sites entiers dédiés à C (90 Moctets), des logiciels libres et/ou gratuits (150 Moctets) reliés à C, pour Linux et Windows.

1.1 Le dispositif d'affichage

VideoC devait être exploitable de multiples manières : sur Linux et Windows, sur la machine personnelle de l'étudiant (connectée ou pas à Internet) et sur les serveurs des salles en libre service de l'université (reliés ou pas à Internet), à partir du cédérom ou du site. Cette multiplicité de configurations nous a fait choisir HTML comme vecteur principal d'affichage.

D'autres choix étaient possibles. Nous avons écarté PDF dont les côtés interactifs sont encore peu répandus, nous souhaitions en effet réaliser des QCM, des animations et coupler des pages à des programmes extérieurs tels qu'Emacs ou le compilateur C. Nous avons aussi écarté tous les outils sophistiqués de conception de cédéroms pour au moins quatre raisons rédhibitoires : *(i)* ils ne s'adressent le plus souvent qu'au monde Windows, *(ii)* les formats internes sont opaques : aucune garantie de ré-emploi du contenu ne peut être faite à plus de quelques années, *(iii)* aucune manipulation non prévue par le logiciel ne peut être opérée sur les données ingérées, *(iv)* tout doit se faire (et se refaire) à la main (et non par programme) sans aucune réelle possibilité d'automatisation.

Les outils ou brouteurs (pour *browsers*) dominant communs à Linux ou Windows, pour afficher de l'HTML sont *Netscape Communicator* et *MicroSoft IExplorer*. Ils ont en commun HTML 4.0, JavaScript et les CSS (pour *Cascading Style Sheets*) et paraissaient donc le bon compromis pour l'affichage. Afin de ne pas devoir engendrer, pour chaque contenu, statique ou dynamique, quatre pages spécialisées pour les quatre brouteurs ciblés (*Communicator* pour Linux, *Communicator* pour Windows, *IExplorer 4* et *IExplorer 5*, tous subtilement différents), nous avons recherché des techniques HTML, JavaScript et CSS compatibles. L'élaboration d'un sous-ensemble des langages HTML 4.0+CSS et la réalisation d'une bibliothèque JavaScript nous a finalement permis de nous rendre indépendants du dispositif final d'affichage au bout d'un temps malheureusement déraisonnable.

Parmi les possibilités originales de VideoC, on trouve des sortes de bulle d'aide qui permettent, quand la souris les approche, de faire apparaître un texte. Ce dispositif est utilisé pour fournir des indices dans des exercices ou pour commenter des programmes de façon non intrusive. Ce point est particulièrement important pour l'enseignement de la programmation où l'on a à montrer et commenter des programmes (cf. figure 1). Certes la notion de commentaires existe dans les langages de programmation mais leur emploi obéit à des règles d'usage qui sont incompatibles avec un discours autour du programme. Nous avons donc créé cette notion de bulles d'aide procurant une sorte de sous-texte annotant un programme avec une perturbation minimale (actuellement, seule la couleur du texte permet d'indiquer la présence d'une annotation). De plus, ces annotations ne perturbent pas les techniques de copier/coller qui ne capturent que le programme et non les annotations.

Malgré le temps passé à nous rendre indépendants du logiciel d'affichage final, nous restons satisfaits de ce choix. En revanche, comme les besoins d'interactivité augmentent et que les brouteurs de demain différeront encore plus subtilement de ceux d'aujourd'hui, nous envisageons que la prochaine édition du cédérom puisse n'utiliser qu'un unique brouteur fourni avec le cédérom. Parmi les possibilités figurent les dernières versions de Mozilla ou HotJava, tous deux écrits en Java.

1.2 Le concentré d'Internet

Le cédérom contient une masse considérable (100 Mo) de documents en HTML, en Postscript, en PDF ou en texte seul. Les documents HTML et PDF comportent des liens relatifs (`href="d/e"`), semi-relatifs (`href="/d/e"`) ou absolus (`href="http://..."`) qu'il importe de respecter de manière à ce qu'ils soient effectifs tant sur le cédérom que sur le réseau, que la machine soit connectée à Internet ou pas. Le tableau 1 récapitule les deux types de liens relatifs (les seuls à poser des problèmes) et la résolution qui en est faite suivant l'url originalement demandée (et la nature du protocole *file* ou *http*) :

Ce tableau montre que les liens semi-relatifs ont toutes chances d'être mésinterprétés avec le protocole *file*, et

[2] `http://www.gnu.org/copyleft/gpl.html`

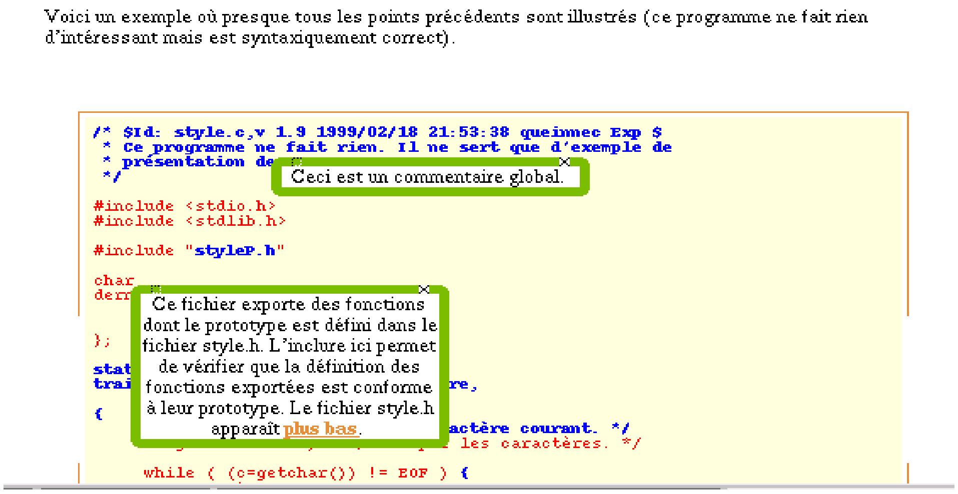

FIG. 1 – Une bulle d'aide sur du code (le curseur est sur « `styleP.h` »)

	lien relatif	lien semi-relatif
url originale	`href="d/e"`	`href="/d/e"`
`file:/a/b/c`	`file:/a/b/d/e`	`file:/d/e`
`http://localhost:19991/a/b/c`	`http://localhost:19991/a/b/d/e`	`http://localhost:19991/d/e`
`http://n:q/a/b/c`	`http://n:q/a/b/d/e`	`http://n:q/d/e`

TAB. 1 – Différentes sortes d'URL

qu'ils peuvent conduire à des conflits de noms en passant par le serveur de pages (`http:...19991/`).

Respecter les liens est donc un problème ardu car il n'est pas envisageable de modifier tous ces liens pour toutes ces configurations, c'est un travail gigantesque, délicat, quasiment impossible quand les liens ne sont pas statiques mais engendrés dynamiquement par du JavaScript et à refaire chaque fois que les documents évoluent. Il faut donc exploiter les liens tels qu'ils sont.

Les pages ressortissent à plusieurs catégories. Les pages des sites aspirés sont telles qu'elles furent conçues. Les pages du cours et des exercices sont le résultat d'un processus de fabrication, détaillé en section 1.3, et contiennent de nombreuses informations filtrées par le serveur et/ou le brouteur. Elles comportent des liens vers les sites aspirés et des liens calculés permettant de servir des pages tout en contrôlant le dispositif de filtrage (faire apparaître les solutions ou pas, les références amont ou aval à d'autres pages, plus ou moins de détails, les crédits aux auteurs, etc.)

La solution que nous avons prise pour le cédérom en septembre 1999 est présentée en figure 2. La page principale du cédérom (un fichier `index.html`, comme il se doit) permet de s'orienter parmi les différents cours et, si l'on choisit notre cours de C, de voir comment lancer le serveur particulier de pages qui opère sur le port 19991.

C'est ce serveur qui est, dans presque tous les cas, responsable des filtrages mentionnés plus haut. Chaque URL reçue par ce serveur est considérée comme un petit programme sur une seule ligne qu'il s'agit de calculer. Voici quelques exemples typiques de ce langage d'URL :

```
/related/find?type=cours&title=Bogues    (1)
/more/randomPage                         (2)
/TOC?prefix=t                            (3)
```

La première de ces URL cherche (`find`) une page de *cours* dont le titre est *Bogues* puis demande (`related`) les pages reliées en amont ou aval de cette page : les pages résultantes sont alors affichées. La recherche d'une page par son titre permet de ne pas dépendre du nom du fichier dont le processus de production, voir section 1.3, n'assure pas la persistance. La seconde URL demande une page au hasard (`randomPage`) mais exige que cette page soit affichée avec plus de détails (`more`). La troisième URL demande les pages de la table des matières (`TOC`) dont le titre commence par la lettre `t`. D'autres commandes permettent d'obtenir des informations sur l'état interne du serveur, etc.

Tous ces liens calculés apparaissent, préfixés d'un petit symbole □ indiquant leur nature. Calculés, ils peuvent être placés en favoris et fonctionner même si le contenu de VideoC évolue.

Le serveur de pages joue aussi le rôle d'un serveur

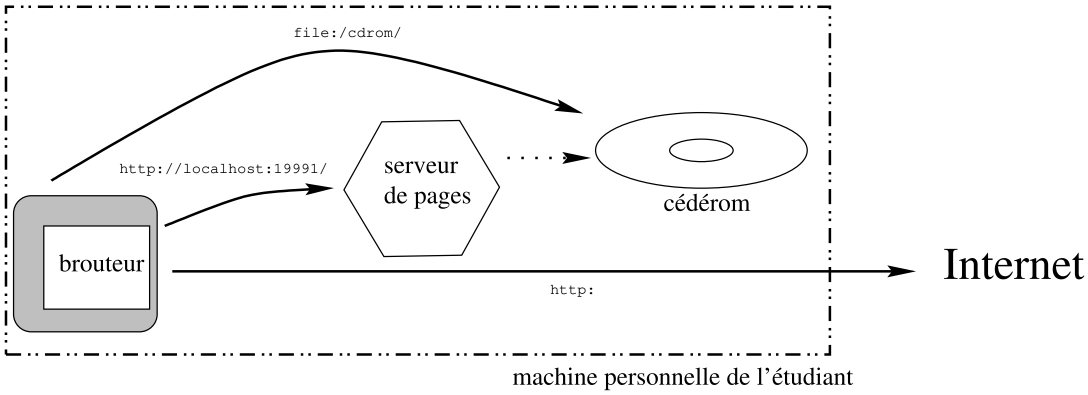

FIG. 2 – Le cédérom de septembre 99

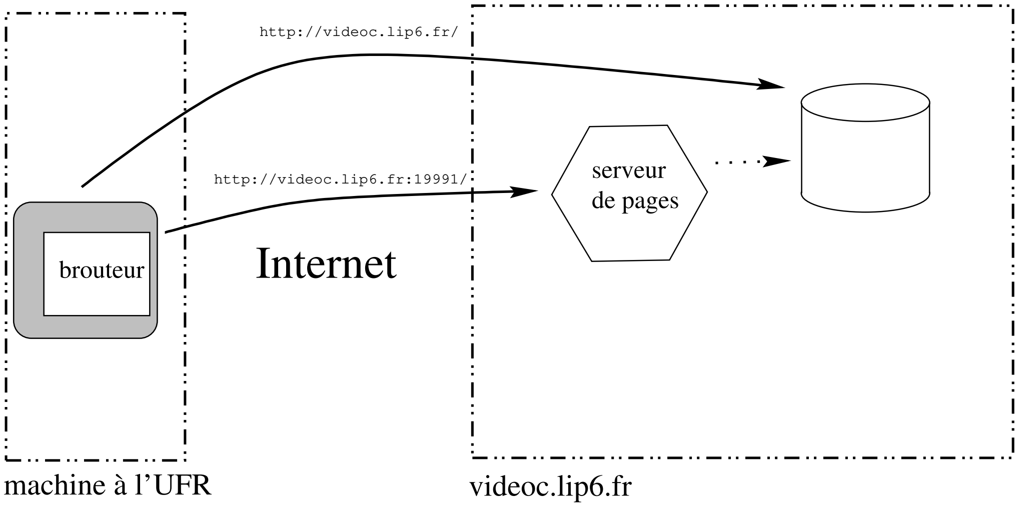

FIG. 3 – Le cédérom vu depuis Internet

HTTP standard pour les fichiers contenus dans VideoC. Lorsqu'une de nos pages contient une référence vers une page du réseau archivée sur le cédérom, deux liens apparaissent : l'un vers la page originale sur le réseau, l'autre, préfixé d'un petit symbole, vers la page copiée en VideoC.

Cette organisation confère au cédérom et au site un comportement identique mais, malheureusement, ne résoud pas le problème des liens semi-relatifs.

En fait, la réédition du cédérom pour septembre 2000 adoptera la nouvelle architecture de la figure 4. Cette architecture requiert une certaine coopération de la part de l'utilisateur puisqu'il faut lui demander de configurer son brouteur pour passer par un relais (ou *proxy*) que nous fournissons sur le cédérom (en Java). Le relais ainsi interposé voit tout ce qui arrive au brouteur ainsi que toutes les URL que requiert le brouteur. Ainsi une page quelconque est référencée par un unique nom qui est son URL originale. C'est le rôle du serveur que de détourner cette URL vers un fichier du cédérom lorsque la machine n'est pas connectée. Nous envisageons d'utiliser Muffin[3] comme relais (toujours écrit en Java).

Cette configuration est assez technique, a des modalités changeant avec la version du brouteur et pose des problèmes lorsque l'étudiant oublie de lancer son relais lorsqu'il souhaite utiliser son brouteur pour autre chose que la consultation du cédérom. Il faut parfois même chaîner plusieurs relais lorsque plusieurs applications en demandent chacune un voire effectuer un choix lorsque les applications ne semblent pas tolérer d'autre relais qu'elles. En revanche, cette technique permet d'assurer que le relais voit tout ce qui est émis ou reçu par le brouteur, que les liens semi-relatifs seront correctement traités et permettra également de mieux répondre au problème de l'implantation des parcours que ne le faisaient les précédentes architectures.

Un point important commun à toutes ces architectures que nous conserverons pour la nouvelle, est la notion de mode dégradé. La robustesse du logiciel du serveur/relais est primordiale. Comme la variabilité des configurations est très étendue, il est impératif, en cas d'indisponibilité du serveur/relais, de pouvoir directement consulter les pages du cédérom (avec le protocole *file :*). Nous avons pris le parti d'encapsuler toute l'information relative à une page dans la page même, sous forme HTML ou en commentaire HTML. Ainsi extraire les pages suggérées, les solutions, les crédits revient à filtrer la page suivant divers critères syntaxiques. Mais dans tous les cas, la page brute est encore affichable (même si trop riche) et permet d'assurer un service minimum égalitaire pour tous les étudiants (dotés de machines personnelles, soit 98% des étudiants de licence d'informatique). Nous nous félicitons encore de ce choix.

[3] `http://muffin.doit.org/`

1.3 Production des pages

Écrire les textes de notre enseignement directement en HTML était a priori exclu car ne permettant pas d'obtenir une version typographiée correcte. Nous sommes donc resté à LaTeX pour lequel nous avons conçu un style particulier nous permettant d'engendrer à partir des mêmes sources, des documents en Postscript, PDF ou HTML. Cette dernière traduction utilise HEVEA [Mar99].

Un cours sur des langages de programmation se doit de montrer des fragments de code du langage enseigné. La programmation littéraire [Knu92] consiste à intégrer ensemble code et commentaire du code. Quoique séduisante, nous n'avons pas retenu cette idée car la moindre correction d'une faute d'orthographe dans le commentaire oblige à recompiler tous les programmes, les retester, regénérer la documentation etc. De plus, il n'est pas très pratique d'éditer un texte avec deux modes syntaxiques différents. Enfin, le code est si dilué dans le texte qu'il est difficile de l'appréhender.

Nous avons réutilisé un outil que nous avions mis au point quelques années auparavant : LiSP2TeX. Cet outil permet d'insérer dans un fichier LaTeX des fragments de code C (ou Lisp comme son nom le suggère). Ainsi le texte est séparé du code et le code est écrit comme recommandé par les guides de style, il peut donc être diffusé tel quel comme exemple, édité ou mis au point avec les outils standard. Lorsque l'on regénère le cours, LiSP2TeX insère automatiquement les fragments de C concernés là où des directives l'instruisent de placer la définition d'une fonction, d'une macro ou d'un type. Un `Makefile` de regénération permet de reconstruire tous les fichiers LaTeX qu'HEVEA transforme alors en HTML découpé en pages individuelles. Divers scripts en Perl extraient les mots définis (pour le moteur de recherche) et la table des titres de pages (pour permettre des références calculées).

Finalement, les bulles d'aide sont spécifiées tout en amont, par l'outil `annote` qui les insère dans le source LaTeX ce qui permet d'engendrer les versions Postscript, PDF ou HTML. Les annotations apparaissent en JavaScript en aval afin de laisser le brouteur exploiter ces bulles sans interférence avec le serveur.

À côté des fichiers C écrits de façon normale, les textes du cours et des exercices ont une structure syntaxique proche de XML c'est-à-dire une structure d'arbre. LiSP2TeX saurait insérer du code au sein d'XML mais il ne semble pas intéressant d'utiliser XML comme langage source car trop verbeux et encore peu commode d'édition. En revanche, XML permettrait de pouvoir vérifier simplement le respect d'une grammaire stricte pour l'écriture d'exercices ou de notes de cours ainsi que des manipulations directes de contenu dans le brouteur final ce qui

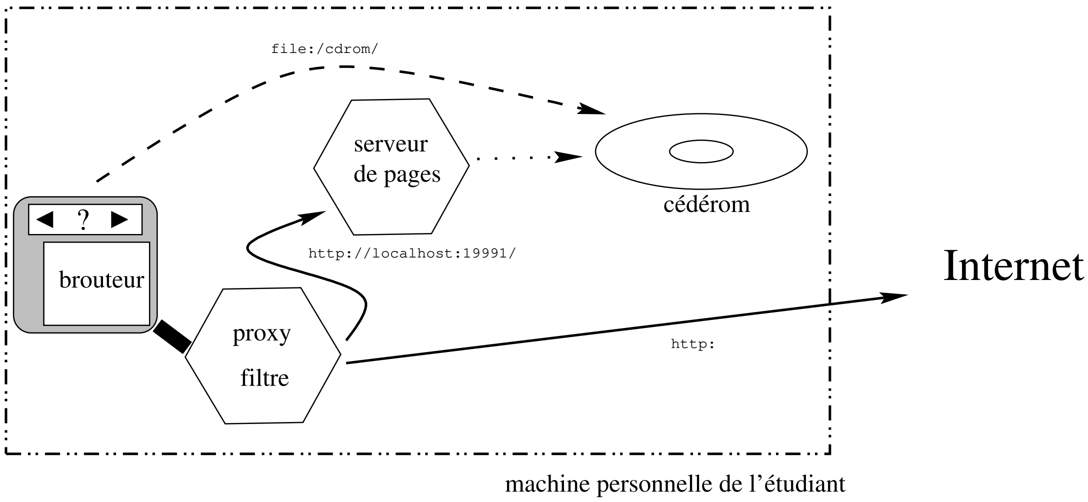

FIG. 4 – Le cédérom de septembre 2000

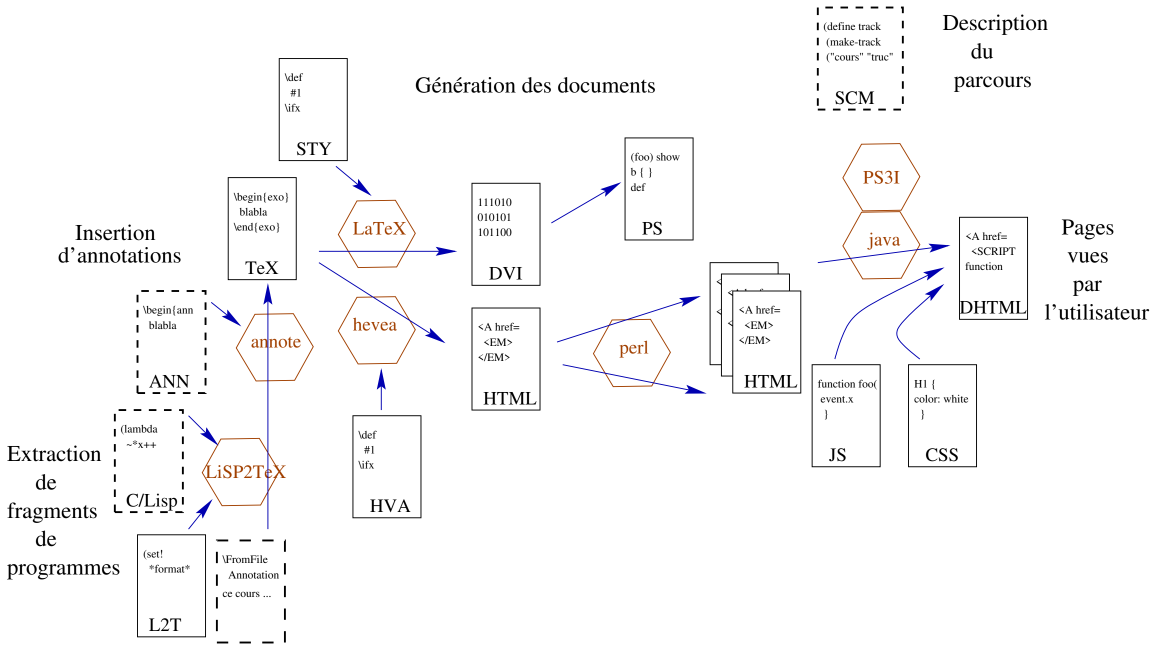

FIG. 5 – Vue d’ensemble de la production des textes et fichiers de configuration. Les sources sont encadrées en tiretés.

déchargerait le serveur. DocBook[4] semble être une autre possibilité intéressante. Nous n'avons pas encore fait ce choix.

1.4 À propos de Java

Tous les outils ayant à tourner depuis le cédérom sont en Java car il est possible d'en trouver des implantations redistribuables sur Linux et Windows. Ce n'est pas le seul langage possible : Ocaml, Perl ou Python sont dans le même cas. Les avantages de Java sont néanmoins multiples : typage, bibliothèque de communication réseau, metteur au point, nombreux programmes libres ou gratuitement utilisables sans oublier son petit côté à la mode.

Ces avantages sont contre-balancés par une certaine inefficacité due à sa grande jeunesse, une évolution constante de ses bibliothèques, des restrictions dans la diffusion et des difficultés de portage (principalement dans les interfaces graphiques) et d'installation.

– L'inefficacité n'est pas très gênante en usage personnel de croisière mais ralentit le lancement du serveur de pages. Il était par exemple déconseillé d'utiliser et JIT et processus légers dans la version pour Linux qui a été placée sur le cédérom ce qui contribuait à rendre encore plus lente cette version. Cette lenteur est plus que gênante lorsque 75 étudiants exploitent en parallèle le serveur de l'UFR.
– La plupart des API que nous avons utilisées telles que sérialisation, servlets, JSP ont fortement évolué tout au long du projet. Leur première implantation est en général inutilisable pour tout emploi sérieux mais vise à capter le marché et faire naître une communauté d'utilisateurs motivés par l'amélioration de l'API. Les palliatifs que l'on doit trouver pour employer ces logiciels sont à revoir dès que les solutions paraissent ce qui nécessite aussi de maintenir une veille active sur ces produits. La sérialisation de Java 1.1 n'est que rudimentaire et n'a atteint un niveau décent qu'avec Java 1.2. Servlets et JSP ont évolué deux à trois fois pendant la durée du projet.
– Les JSP sont des pages HTML contenant des fragments de Java, elles sont traduites en Java puis compilées en code-octet pour exécution. Cette dernière étape nécessite d'incorporer le compilateur Java vers code-octet dans le serveur du cédérom alors que ce même compilateur n'est pas librement re-diffusable. Nous avons donc dû pré-compiler ces pages ce qui rend alors leur mise à jour délicate.
– Le système d'exploitation Windows n'est pas multi-utilisateur et ne semble pas envisager que plusieurs versions d'un même produit puissent coexister sur une même machine. Ainsi pour utiliser Java sur Windows, doit-on installer la version du cédérom et ainsi écraser les éventuelles versions précédentes ce qui invalide (voire bousille gravement) les environnements de programmation (Visual J++, JBuilder etc.) et les utilitaires qui s'attendaient à trouver Java ailleurs etc. Inversement, ne pas installer la version du cédérom ne permettait pas de tourner le serveur de pages.

Malgré ces problèmes, coûteux en temps de développement, nous restons persuadés de l'intérêt du choix de Java car la qualité des implantations progresse et la définition des programmes utilitaires se stabilise.

2 Parcours

L'originalité principale de ce cédérom (et du site associé) est la notion de parcours ou plutôt la technique d'implantation que nous avons choisie. Un parcours est un cheminement que doit suivre un étudiant et qui le conduit à visiter un certain nombre de pages. Plusieurs contraintes sont à satisfaire :

– les pages doivent pouvoir être quelconques et donc ne pas avoir été spécialement préparées pour un (ou des) parcours particulier(s) ;
– un parcours peut s'étendre sur plusieurs mois ce qui oblige à rendre persistante la position de l'utilisateur dans ce parcours ;
– cette position doit pouvoir être transportable d'une machine à l'autre afin qu'un parcours débuté à l'université puisse être poursuivi chez soi ou inversement ;
– le parcours peut ne pas être linéaire et dépendre de résultats produits par des QCM ou de choix de sous-parcours.

La solution passe par la reconnaissance que le parcours est en fait un programme et qu'une position est une continuation (cf. section 2.1) ! Le programme doit être écrit dans un langage universel donc doté des structures de contrôle et de données qui en font un véritable langage de programmation. Il est ainsi aisé de spécifier qu'un étudiant — ne franchira une page (un QCM) qu'après un certain taux de réussite ou — sera soumis à de nouveaux détours tant qu'un certain résultat ne sera pas acquis ou — ne pourra sortir d'un ensemble de pages que lorsque plus de 75% d'entre elles auront été consultées, etc. Adopter un langage de programmation permet de se libérer du peu de puissance que procurent les langages graphiques pour auteurs.

2.1 Continuation

Cette section présente, par l'exemple, le concept de continuation. Nous utiliserons Scheme comme langage de programmation.

[4] http://www.docbook.org/

Pour évaluer une expression (dans un programme), il faut, au minimum, préciser l'environnement lexical associé c'est-à-dire des associations entre variables et valeurs. Il faut également savoir ce que l'évaluateur doit faire de la valeur de l'expression une fois qu'elle sera obtenue : la valeur peut être stockée dans une variable, servir d'argument à une fonction, additionnée à une constante, etc. La *continuation* représente l'entité à qui cette valeur est fournie. Cette entité représente la suite du calcul qui reste à effectuer pour peu qu'on lui donne la valeur qu'il attend. On peut également assimiler une continuation à une copie de la pile d'évaluation.

Pour fixer les idées, lors de l'évaluation de l'expression `(* (+ 1 1) (+ 3 4))`, la multiplication commence par requérir l'évaluation de son premier paramètre : `(+ 1 1)`. Lors de l'évaluation de `(+ 1 1)`, la continuation peut être représentée comme l'expression `(* [] (+ 3 4))` où `[]` indique là où injecter la valeur 2 obtenue par addition de 1 et 1. Lors du calcul du second paramètre de la multiplication c'est-à-dire l'expression `(+ 3 4)`, la continuation peut-être représentée comme `(* 2 [])` où l'on voit notamment que le premier argument de la multiplication a déjà été calculé.

Voici un calcul plus complexe où l'on souhaite calculer la factorielle de 5 avec la définition suivante de la factorielle :

```
(define (factorielle n)
  (if (= n 1)
      1
      (* n (factorielle (- n 1))) ) )
(factorielle 5)
```

La continuation, au moment où l'on calcule la factorielle de 1 est : `(* 5 (* 4 (* 3 (* 2 []))))`. Ceci montre que tout point de calcul a une continuation quelque soit la complexité de ce calcul.

La notion de continuation a été inventée pour exprimer la sémantique des débranchements dans les langages de programmation. Elle permet d'exprimer toutes les structures de contrôle séquentielles telles que *goto*, échappement ou coroutine. Les continuations sont des entités occultes dans tous les langages de programmation sauf en Scheme [KCR98] où on peut les capturer et les utiliser comme des valeurs fonctionnelles : puisqu'elles attendent une valeur pour dérouler un calcul, ce sont naturellement des fonctions unaires. On pourra consulter [Que94] pour plus de détails sémantiques ou implantatoires.

2.2 Exemple de parcours

À tout instant, l'état de l'utilisateur au sein de son parcours doit pouvoir être matérialisé et stocké. Lorsqu'un utilisateur se voit servir une page (par la fonction `show`, cf. l'exemple plus bas), la continuation de cette page, c'est-à-dire la suite du parcours de cet utilisateur, est stockée afin qu'il puisse la retrouver lorsqu'il souhaitera poursuivre son parcours. Un parcours est donc exprimé par un programme Scheme ; une position instantanée dans ce parcours par une continuation.

Les continuations sont, de plus, le bon concept pour déterminer ce que peut être la signification de l'emploi du bouton *back* dans un brouteur [Que00]. Ce bouton permet de revenir à d'anciennes pages et de soumettre de nouvelles réponses à d'anciennes questions. De nombreux serveurs ont des problèmes avec ce genre de comportement.

Donnons un exemple de parcours illustrant quelques-unes des possibilités.

```
(define (yes-or-no? url)
  ;; url -> bool
  (let ((request (show url)))
    (equal? (getParameter request
                          "answer" )
            "yes" ) ) )

(let ((pagesVisitees 0))
  (set! pagesVisitees (+ pagesVisitees 1))
  (if (yes-or-no? "url1")
      (for-each
       (lambda (url)
         (set! pagesVisitees
               (+ pagesVisitees 1) )
         (show url) )
       (list "url2" "url3" "url4") ) )
  (show
   (html
    (head (title "Bravo"))
    (body
      (p "Bravo! Vous avez exploré "
         pagesVisitees " pages.") ) ) ) ) )
```

Dans ce programme, est définie une fonction `yes-or-no?`, affichant une page contenant un bouton (ou menu) permettant d'effectuer un choix booléen. La seconde expression affiche une page url_1 qui propose ce choix booléen et qui permet ou non d'enchaîner (`for-each`) sur une série de trois autres pages (url_2, url_3 et url_4) avant de déboucher sur une page dynamiquement engendrée. Un compteur local, `pagesVisitees`, permet de dénombrer les pages servies, sa valeur sert à engendrer le texte de la page finale.

La continuation de la page url_1 est équivalente à l'expression :

```
(let ((pagesVisitees 1))
  (if (let ((request [] ))
        (equal? (getParameter request
                              "answer" )
                "yes" ) )
      (for-each
       (lambda (url)
```

```
          (set! pagesVisitees
                (+ pagesVisitees 1) )
          (show url) )
        (list "url2" "url3" "url4") ) )
  (show (make-Page
          "Bravo! Vous avez exploré "
          pagesVisitees
          " pages." )) )
```

Le symbole `[]` marque l'endroit où la page url_1 reviendra avec une nouvelle requête qui sera liée à la variable `request` dont on extraiera le paramètre nommé `answer` contenant le choix de l'utilisateur (`yes` ou `no`). Cette continuation marque bien que le parcours en est à la visualisation de la page url_1 dans la fonction `yes-or-no?` elle-même invoquée dans la condition d'une alternative, elle-même enserrée dans un bloc dont la variable locale `pagesVisitees` vaut 1 à cet instant précis.

2.3 Mise en œuvre

Techniquement et aujourd'hui, le serveur de pages du cédérom joue également le rôle d'un relais gérant les parcours. Un interprète Scheme (nommé PS3I pour *Persistent Server-Side Scheme Interpreter*) est couplé au serveur pour évaluer les parcours. L'interprète implante complètement et sans restriction les continuations ; l'interprète peut être utilisé dans un contexte multi-utilisateurs et multi-tâches pour ne pas mélanger les différents parcours. Les pages passant par le gérant des parcours sont instrumentées afin d'incorporer le bandeau standard des parcours. Ce bandeau standard permet d'agir sur le parcours c'est-à-dire de le poursuivre, de le reprendre (lorsque l'on a vagabondé loin du parcours) ou de revenir sur des pages précédemment parcourues. Toutes ces commandes sont implantées par manipulation de continuations déclenchées par des commandes du petit langage d'URL exposé plus haut.

Si l'instrumentation des pages HTML est possible, celle de pages en PDF ou Postscript ne l'est guère. Aussi pour la prochaine moûture du cédérom, nous maintiendrons l'information du bandeau standard dans une petite fenêtre à part et gérée par le brouteur (cf. détail ◄ ? ► en figure 4). Il sera alors possible de gérer son parcours indépendamment de la nature des pages servies. Cette individualisation du bandeau permettra, en outre, de ne pas perturber l'ordonnance des pages servies.

VideoC procure plusieurs parcours dont un parcours linéaire total de notre cours ainsi que des parcours de deux heures pour chaque séance de travaux pratiques.

2.4 Avantages

Utiliser des continuations nous permet de répondre favorablement aux contraintes énoncées plus haut en section 2. D'autres avantages se dévoilent aussi comme :

- la possibilité de pouvoir construire de nouveaux parcours que les étudiants peuvent télécharger et qui les feront arpenter les pages du cédérom qu'ils ont auparavant obtenu. Ainsi peut-on concevoir de nouvelles utilisations d'un corpus de pages existant et déjà distribué.
- la possibilité de proposer à la fois aux étudiants de vagabonder parmi de nombreux documents liés à C tout en leur permettant de se repositionner sur leur parcours en un seul clic.
- la possibilité de programmer en style direct sans se soucier de continuations dont l'emploi est confiné à la fonction `show` et au comportement du bandeau standard permettant de contrôler le parcours. Cela est particulièrement important lorsque l'on désire implanter une représentation de l'étudiant (ce qu'il a parcouru, les QCM qu'il a effectué et ses résultats, etc.)

  Voici par exemple, une fonction montrant comment on peut proposer une liste d'exercices (`exos`) sachant que l'on attend de l'étudiant qu'il en résolve un en moins de cinq minutes.

```
(define (test exos)
  (if (pair? exos)
      (let* ((debut-date (get-date))
             (request (show (car exos)))
             (fin-date (get-date)) )
        (if (resolu? request)
            (if (> (- fin-date debut-date)
                   300 )
                ;; trop lent : proposer un autre exercice
                (test (cdr exos))
                'impeccable )
            ;; faux : proposer le même exercice
            (test exos) ) )
      ;; plus aucun exercice à proposer
      'epuise ) )
```

3 Perspectives

La réalisation de ce cédérom a demandé environ cinq hommes*mois ce qui est un investissement assez lourd. Comme le montre l'article compagnon [CQS00], le cédérom et le site ont été très bien accueillis par les étudiants. Une nouvelle édition est donc en cours de préparation pour la rentrée de septembre 2000.

Comme mentionné plus haut et à plusieurs reprises, la nouvelle architecture vise à être plus facilement installable sur Windows, plus robuste et rapide et encore plus riche en contenu. Parmi les améliorations, nous pensons à :

- améliorer la production de contenu afin de convertir plus d'enseignants à nos techniques. Nous intégrerons notamment des projets d'étudiants de DESS

Génie des Logiciels Applicatifs et notamment :
  - un langage de description de QCM (écrit par Z. Mekni et E. Cobian) permettant d'engendrer des QCM qui peuvent tourner sur le brouteur en mode d'apprentissage ou sur le serveur en mode examen tout en maintenant la production de documents en Postscript, PDF et, surtout, HTML.
  - un nouveau moteur de recherches plus efficace et plus riche puisque proposant des expressions régulières (écrit par Y. Sihalathavong et D. Wang)
- permettre l'annotation de pages quelconques du réseau. Cette annotation doit se faire à la volée par réécriture de ces pages de façon non intrusive.
- mettre définitivement au point une manière de mettre à jour le contenu du cédérom et son logiciel à partir du réseau ou de disquettes.
- expérimenter la nouvelle architecture à base de relais afin de résoudre complètement et de façon unique les problèmes d'interprétation des liens.
- coupler correctement le serveur de page avec les outils d'édition, de compilation et de mise au point de la machine personnelle de l'étudiant.

Enfin et dans une direction un peu différente de ce que nous tentions de faire à savoir : n'avoir qu'une unique version des pages pour VideoC, nous nous demandons, pour améliorer la robustesse du mode dégradé, s'il ne serait pas envisageable de « compiler » les URL des textes archivés sur le cédérom c'est-à-dire de les résoudre à la création du cédérom plutôt que de les laisser calculer à l'exploration du cédérom ce qui requiert un serveur opérationnel.

4 Conclusions

Cet article a présenté les grandes lignes d'une architecture logicielle pour la réalisation d'un cédérom, les buts initiaux, les difficultés principales, les choix retenus, les améliorations envisagées. Nous espérons que ces informations seront utiles aux futurs concepteurs de sites ou de cédéroms qui souhaitent maîtriser les outils mis à leur disposition pour ce faire.

VideoC est la première incursion de l'auteur dans le domaine de l'enseignement assisté par ordinateur et ne couvre que les seuls aspects techniques de production d'un site et d'un cédérom avec les moyens modernes que sont les brouteurs programmables ou les serveurs dédiés. Les enseignements sur le plan pédagogique sont détaillés en [CQS00] [5].

L'originalité majeure de ce cédérom est la notion de parcours et l'introduction concommittante de continuations. L'abondance d'informations dans le cédérom, la forte incitation au vagabondage hors des parcours imposés, rendent nécessaire de pouvoir, en un clic, replacer l'étudiant sur son parcours. C'est ce que nous avons réalisé en exprimant des parcours sous forme de programmes dans un langage permettant de capturer puis de manipuler des continuations : Scheme. La puissance linguistique obtenue est considérable et permet de programmer des parcours procurant un commentaire autour de pages préexistantes favorisant ainsi une réutilisation de matériel pédagogique.

Quoique en cours de réécriture, nous rappelons que tous les logiciels écrits pour VideoC sont diffusés sous GPL et sont accessibles à partir de `http://videoc.lip6.fr/`.

Nous remercions tous nos étudiants et tous les membres de notre équipe pédagogique pour leur patience tout au long de cette expérience.

[5] `http://videoc.lip6.fr/queinnec/Papers/tice2000a.ps.gz`

Références

[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.

[KCR98] Richard Kelsey, William Clinger, and Jonathan Rees, editors. Revised^5 report on the algorithmic language Scheme. *Higher-Order and Symbolic Computation*, 11(1) :7–105, 1998. Also appears in ACM SIGPLAN Notices 33(9), September 1998.

[Knu92] Donald E Knuth. *Literate Programming*, volume Center for the Study of Language and Information. CSLI Publications, 1992.

[Mar99] Luc Maranget. Hevea, un traducteur de latex vers html en caml. In *Actes des 10èmes Journées Francophones des Langages Applicatifs*. INRIA, 1999. http ://pauillac.inria.fr/~maranget/hevea/.

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

[Que00] Christian Queinnec. The influence of browsers on evaluators or, continuations to program web servers. Montreal (Canada), September 2000.