

Le navigateur Web mmm: un développement en Caml Light

François Rouaix
Projet Cristal, INRIA Rocquencourt

Résumé: mmm est un navigateur pour le World Wide Web (WWW) écrit en Caml Light 0.7, utilisant les bibliothèques `unix`, `str`, et `camltk`. Les performances de mmm sont voisines de celles des autres navigateurs Web disponibles, gratuits ou commerciaux. Cette expérience montre qu'il est possible d'utiliser Caml Light et ses bibliothèques pour réaliser une application complète, compétitive avec un développement similaire en C.

Introduction

Le Web est un ensemble de documents de natures variées (HTML, images, sons, postscript, etc...), situés sur différents hôtes, et accessibles par différents protocoles (`http`, `ftp`, `wais`, `gopher`, ...). Un document est référencé par une adresse, qu'on appelle URL (Uniform Resource Locator[1]), et contient éventuellement des liens vers d'autres documents (hypertexte). L'objectif d'un navigateur est de fournir une interface utilisateur conviviale autorisant la transmission et l'affichage d'un maximum de documents, tout en cachant les détails techniques à l'utilisateur.

1 Architecture

Le navigateur mmm se compose essentiellement des parties suivantes:

protocoles: l'adresse d'un document sur le Web (URL) spécifie le protocole qui permet de transférer ce document. mmm implante seulement le protocole `http`[4], laissant à un *proxy* [7] le soin de gérer les autres (sachant que `http` est au moins nécessaire pour communiquer avec le proxy).

cache: lors d'une session, l'utilisateur est souvent amené à demander plusieurs fois le même document, avec des adresses parfois syntaxiquement légèrement différentes quoique sémantiquement identiques. De même, un document inclus parfois plusieurs fois la même image, ou plusieurs documents d'un même site font référence aux mêmes images (charte graphique). Le rôle du cache est d'éviter d'avoir à recharger ces documents ou images en cas d'accès multiples.

navigateur: le navigateur a la charge de gérer un historique des documents affichés et l'interface générale.

afficheurs: les documents présents sur le Web sont de toute nature. Lors d'une connexion, le serveur indique au client, sous la forme d'un type MIME [5], le type du document demandé (`text/html`, `image/gif`, `application/postscript`, ...). Il est de la responsabilité du client d'afficher le document en fonction de son type. mmm implémente seulement les afficheurs pour les types `text/html` (documents au format HTML [3]), et `text/plain` (documents ASCII). L'affichage des autres documents est délégué à un programme externe de traitement de messages au format MIME, comme `metamail`.

L'afficheur HTML est lui-même composé de plusieurs parties:

analyseur lexical HTML: les lexèmes sont ceux de SGML, c'est-à-dire ouverture de `tag` avec attributs, fermeture de tag, et texte.

machine d’affichage: quelques primitives simples (afficher du texte, ouvrir un paragraphe, changer les attributs d’affichage, changer l’indentation) suffisent à afficher l’essentiel de HTML 2.0. Des primitives spécialisées plus complètes permettent d’implanter les images et les formulaires (boutons, zones de saisies).

machine d’interprétation HTML: c’est un automate qui travaille sur un flot de lexèmes. Il a la charge de transformer le source HTML en ordres pour la machine d’affichage. C’est donc essentiellement à cette étape que se décide le style de présentation du document HTML.

Cette architecture est évolutive, puisqu’il est facile d’ajouter de nouveaux protocoles, ou d’améliorer l’afficheur en suivant les évolutions de HTML, ou de changer le style de présentation, la sémantique de l’historique, ou la gestion du cache. De plus, l’architecture logicielle permet d’opérer ces modifications de façon relativement indépendante.

2 Fonctionnalité, asynchronisme et continuations

Pour une bonne convivialité de l’interface utilisateur, un navigateur doit gérer concurremment la transmission et l’affichage des documents, afin de ne pas être trop dépendant du comportement des réseaux (bloquages, faible taux de transfert).

Caml Light ne dispose pas de primitives pour gérer la concurrence, mais il est néanmoins possible de la simuler partiellement grâce à la fonctionnalité et à l’appel système Unix `select`. En effet, la boucle de gestion des événements de CamlTk, basée sur `select`, permet d’enregistrer un descripteur de fichier, de telle sorte qu’une fonction est invoquée lorsque des données sont disponibles à la lecture sur ce descripteur. Ainsi, si l’exécution de ces fonctions est “courte”, on obtiendra la concurrence en enregistrant plusieurs descripteurs de fichiers avec des comportements éventuellement différents. Toutes les interactions avec l’utilisateur fonctionnent selon ce principe (cliquer sur un bouton invoque une fonction).

On parvient donc à transmettre et afficher plusieurs documents HTML simultanément, car l’action invoquée lors de la disponibilité sur chaque connexion est brève : elle consiste à lire un lexème, et à appliquer l’automate d’affichage à ce lexème. Il y a bien sûr autant d’instances de l’automate d’affichage que de documents affichés.

Par ailleurs, un mécanisme de continuations qui repose sur le passage de fonctions en arguments est indispensable pour obtenir le maximum d’asynchronisme. En effet, en dehors de l’affichage proprement dit, la gestion des requêtes à un serveur se doit d’être asynchrone pour ne pas bloquer l’interaction avec l’utilisateur. Voici le scénario typique:

- le navigateur détecte un clic de souris sur une ancre affichée dans le document courant. Il faut donc aller chercher le document et le visualiser (de manière asynchrone), et si c’est un document avec un afficheur interne, l’ajouter dans l’historique et dans le cache.
- la requête est lancée au serveur. Nous ne savons toujours pas de quel type de document il s’agit, mais le serveur va nous le dire dans les en-têtes de sa réponse. La continuation de la lecture des en-têtes est donc l’appel de l’afficheur adéquat (ou, d’ailleurs, d’une fonction de sauvegarde sur fichier).
- l’afficheur décide si le document doit être ajouté au cache. La continuation de l’afficheur (si c’est un afficheur interne) est l’ajout dans l’historique.

Ce style de continuations par programmation fonctionnelle apparaît à de nombreux endroits dans le navigateur. Citons par exemple:

- suivi des documents déplacés,
- décodage au vol de documents (décompression),
- authentification par mot de passe,
- soumission de formulaires,
- affichage des images à l’intérieur des documents HTML.

3 Traits favorables du langage d'implémentation

Gestion mémoire: dans ce type d'application, interactive et tournant plusieurs heures ou plusieurs journées de suite, la qualité de la gestion mémoire est primordiale. Il faut d'une part que le ramasse-miette n'interrompe pas brutalement et pour une longue période l'exécution du programme, et d'autre part éviter les fuites de mémoire qui conduiraient à une occupation mémoire strictement croissante.

Impérativité: les traits impératifs du langage (références et structures de données mutables) sont essentielles dans la gestion de plusieurs composants de l'application (cache, automates, formateur de texte, ...).

Bibliothèques: mmm utilise les bibliothèques suivantes, distribuées avec Caml Light 0.7:

- bibliothèque standard (hashtbl, set, lexing)
- bibliothèque unix
- interface avec Tk 4.0 (camltk)

Outils: l'environnement de programmation Caml Light a aussi beaucoup aidé lors de ce développement, notamment le **débogueur**, mais aussi le **profileur** pour détecter les points critiques pour les performances.

4 Difficultés

Les principales difficultés techniques rencontrées sont liées au traitement des cas exceptionnels en présence de concurrence. Par exemple, l'utilisateur est susceptible de détruire une fenêtre à tout moment. Il ne faut donc pas que les fonctions en suspens (continuations) supposent l'existence de celle-ci au moment où elles seront effectivement évaluées. Le débogage est d'ailleurs assez difficile, puisque l'avance pas à pas dans le programme ne permet en général que de détailler le fonctionnement d'une fonction, le reste du calcul étant en suspens dans les continuations.

5 Conclusions

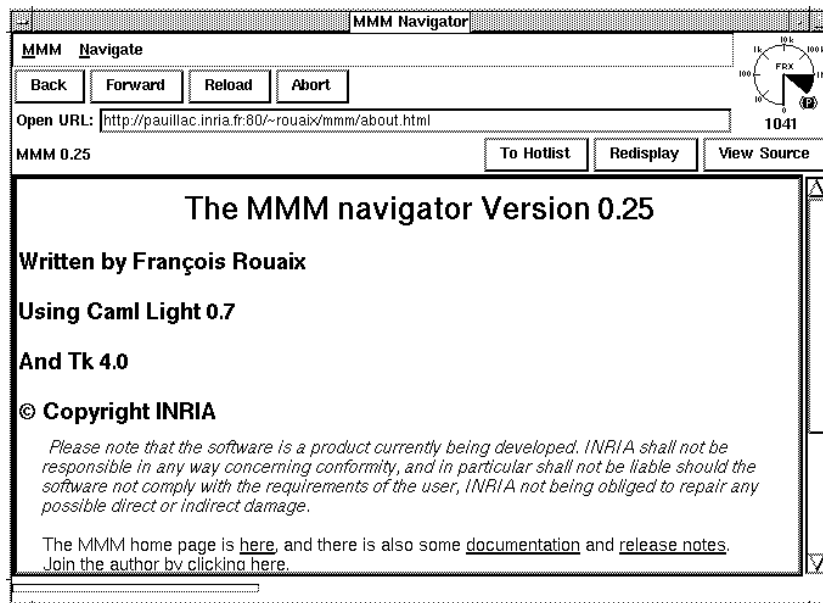
Le source du navigateur représente moins de 7000 lignes "brutes" de Caml Light, réparties en environ 40 modules. Même si certains tâches du navigateur sont déléguées à des agents extérieurs, mmm fournit en gros les mêmes possibilités de navigation, avec les mêmes performances, que d'autres navigateurs disponibles, commerciaux ou gratuits. mmm est distribué gratuitement sur Internet:

`ftp://ftp.inria.fr/INRIA/Projects/cristal/Francois.Rouaix/mmm`
`http://pauillac.inria.fr/~rouaix/mmm/`

References

- [1] T. Berners-Lee. Uniform Resource Locators (URL). RFC 1738, December 1994.
- [2] T. Berners-Lee. Universal Resource Identifiers in WWW. RFC 1630, June 1994.
- [3] T. Berners-Lee and D. Connolly. Hypertext Markup Language – 2.0. HTML Working group, IETF Internet Draft, August 1995.
- [4] T. Berners-Lee, R. T. Fielding, and H. Frystyk Nielsen. Hypertext Transfer Protocol – HTTP/1.0. HTTP Working group, IETF Internet Draft, March 1995.
- [5] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions). RFC 1341, June 1992.
- [6] Xavier Leroy. Programmation du système Unix en Caml Light. Rapport technique 147, INRIA, 1992.

Figure 1: Le navigateur mmm



- [7] Ari Luotonen and Kevin Altis. World Wide Web Proxies. In *Proceedings of the First International Conference on the World Wide Web*. Elsevier Science BV, May 1994.
- [8] John K. Ousterhout. *Tcl and the Tk Toolkit*. Number ISBN 0-201-63337-X. Addison-Wesley, 1994.
- [9] Pierre Weis and Xavier Leroy. *Le langage Caml*. Number ISBN 2-7296-0493-6. InterEditions, 1993.
- [10] Brent B. Welch. *Practical Programming in Tcl and Tk*. Number ISBN 0-13-182007-9. Prentice Hall, 1995.