MMM: un navigateur Web en Caml

François Rouaix
INRIA Rocquencourt
Projet Cristal

Novembre 1995

Caractéristiques

- écrit en Caml Light / Caml Special Ligh
- bibliothèques libunix, camltk, (libstr)
- HTML 2.0 (y compris formulaires)
- HTTP 1.0
- multi-fenêtres, asynchrone
- applets en Caml Special Light
- 10000 lignes de code

Plan

- Fonctionnalité et interfaces homme-machine (IHM)
- Interface entre Caml et Tcl/Tk
- Asynchronisme sans concurrence
- Applets

IHM et Fonctionnalité

- eléments graphiques (*widgets*) (boutons, menus, ...)
- interactions (*callbacks*) associés à un évenement
- boucle de gestion des évenements

Chaque interaction est une fonction.

Paramètres: données spécifiques à l'évènemer
Fermeture

Caml et Tcl/Tk

Tcl: langage interprété, à base de chaînes de caractères. Facile à étendre, facile à embarquer.
Tk: bibliothèque IHM pour Tcl. Écrite en C. Ne peut pas être facilement dissociée de Tcl.

- interface en C pour les appels de Caml vers Tcl et de Tcl vers Caml
- code Caml support (gestion des callbacks, nommage des widgets, ...)
- langage de description de widgets et commandes

Caml et Tcl/Tk: interface C

Appels vers Tcl/Tk au plus bas niveau possible: pas d'analyse lexicale ou syntaxique, pas d'évaluation des arguments.

```
type tkArgs =
    TkToken of string
  | TkTokenList of tkArgs list
  | TkQuote of tkArgs
val tkEval : tkArgs array -> string
```

Appels vers Caml: hash-table de fonctions

```
/* The Tcl command for evaluating callback in Caml *
int CamlCBCmd(clientdata, interp, argc, argv)
     ClientData clientdata;
     Tcl_Interp *interp;
     int argc;
     char *argv[];
{
  camltk_dispatch_callback(copy_string_list(argc,arg
  /* Assume no result */
  /* Never fails (Caml would have raised an exceptio
  interp->result = "";
  return TCL_OK;
}
```

Caml et Tcl/Tk: support

- table des callbacks:

```
let callback_naming_table =
   (Hashtblc.new 401 :
     (string, callback_buffer -> unit) Hashtblc.t)
let callback_memo_table =
   (Hashtblc.new 401 : (widget, string) Hashtblc.t)
```

en Tcl: `{camlcb f112 0.34 0.98}`

en Caml: `Hashtblc.find callback_naming_table id args`

où `id="f112"`, `args=...`

- mécanisme de nommage des widgets:

  widget en Tcl/Tk: procédure globale

  `.top.hgroup.vgroup.buttonOk`

  widget en Caml: type abstrait (fonctions de création)

Caml et Tcl/Tk: langage de description

```
subtype option(line) {
   ArrowStyle ["-arrow"; ArrowStyle]
   ArrowShape ["-arrowshape"; [Units; Units; Units]]
   CapStyle ["-capstyle"; CapStyle]
   FillColor
   ...
   }

widget button {
 # Standard options
 option ActiveBackground
 option ActiveForeground
 ...
 # Widget specific options
 option Command ["-command"; function ()]
 ...
 function () flash [widget(button); "flash"]
 function () invoke [widget(button); "invoke"]
 }

module pack {
 function () configure
   ["pack"; "configure"; widget list; option(pack) ]
 ...
 function (widget list) slaves ["pack"; "slaves"; wi
 }
```

Caml et Tcl/Tk: code produit

```
type options =
   ...
   | ArrowShape of units * units * units
   | ArrowStyle of arrowStyle
   ...

let create parent options =
  let w = new_widget_atom "button" parent in
   tkEval [|
     TkToken "button";
     TkToken (widget_name w);
     TkTokenList (List.map (function x ->
 cCAMLtoTKoptions w options_button_table x) options)
     |];
   w

let flash v1 =
tkEval [|cCAMLtoTKwidget widget_button_table v1;
        TkToken"flash"|];()
```


Comparaison avec Tcl/Tk

- typage statique (sauf “sous-typage” des c
  structeurs)
- nommage implicite des widgets:
  style fonctionnel
- callbacks: vraies fermetures
- fonctions get/set séparées
- exceptions au lieu de chaînes vides

Asynchronisme sans concurrence

- boucle d'évènements
- interaction = fonction "courte"
- évenements sur descripteur de fichiers

Exemple:
descripteur sur connexion réseau
pour un document HTML
interaction = lecture et affichage d'un lexème

Continuations explicites

Asynchronisme: chaque traitement/calcul do
être écrit comme un *callback*.

1. envoyer une requête
2. lire les en-têtes de la réponse
3. analyser les en-têtes, et lancer des traitements sup
   (redirection, autorisations, décodage)
4. sauver le document dans le cache
5. afficher le document

Chaque calcul est une fonction passée comm
"continuation" à la précédente.

Les Applets

applet: mini-application. Programme compilé, résidant sur un serveur, téléchargé par le navigateur, linké dynamiquement et exécuté dans le navigateur.

Problèmes: portabilité, liaison dynamique, efficacité, sécurité, puissance.

Solutions: bytecode, systèmes de types et de modules, authentification par cryptographie.

Compilation

Pourquoi:

- non diffusion du source
- efficacité
- distribution de “grosses” applications

Requis:

- portabilité (bytecode)
- compilation séparée
- liaison dynamique

Problèmes de sécurité

- crash du navigateur (ou de la machine, selon l'OS)
- modification du navigateur
- modification ou destruction d'informations sur la machine client
- envoi d'information confidentielle de la machine client vers l'extérieur
- exécution de programmes présentant les mêmes risques

Typage fort

Erreur de type/coercion non controlée = tro
de sécurité
en particulier *données = code*

Contrôle des accès mémoire (pointeurs) : forr
de protection intra-processus

Portée lexicale : contrôle de l'environnemer
d'exécution

Modules

Typage fort + compilation séparée :
le linker doit vérifier la conformité
export / import.

Système de module : implémentations (valeurs),
interfaces (types).

Unité de compilation: interfaces requises:

```
File slide.cmo
  Unit name: Slide
  Digest of interface implemented: ab26829916237eab286bbf12e
  Interfaces imported:
        00294355ddec387eb2b3ee9f081f5bc4        Safetk
        be68828628e238435aa3f7c4cf9e6f72        Safemmm
        3736054e3637ded258fa07b87c7c4ef1        Safestd
  Uses unsafe features: no
```


Environnements sûrs

- restrictions de module par contrainte d'in
  terface

  ```
  module Foo = (Foo : sig
     type t = Foo.t = <définition de type>
     val f : <type>
     ...
     end)
  ```

  pas de duplication de code
  partage des types

- fonctions enpaquetées
  Exemple: ouverture de fichiers
  interrogation de l'utilisateur.

Compromis entre sécurité et intégration de
applets dans le navigateur.

Authentification

Intégrité du bytecode et des informations pour le linker ?

- signature PGP
  authentification de l'auteur $\neq$ confiance dans l'auteur

- compilateurs fiables
  développement et test en local
  envoi du source au compilateur fiable
  réception du bytecode signé par le compilateur
  vérification égalité bytcode
  mise à disposition

Il suffit de quelques compilateurs "fiables".

Java/HotJava

C++ moins: pointeurs (structures, unions
fonctions, ...
Reste: types de base, tableaux, classes, objet

Typage fort mais pas complètement statique

Reconstruction de types, vérifications sur bytecode

- informations de types dans le bytecode (in
  structions spéciales)

- transmission des interfaces des classes

Code Mobile/Agents

Obliq
Telescript

Applets source

Grail (Python + Tcl/Tk), SurfIt (Tcl/Tk), ...
?

Conclusion

- facilité de programmation d'une IHM ave
  un langage fonctionnel

- applets: application directe des système
  de types et de modules

`http://pauillac.inria.fr/~rouaix/mmm/`

```
open Safestd
open Safemmm
open Safetk
(* These are sub-modules or the previous ones *)
open Tk
open Viewers
open Document
open Www

let rec load_image ctx w = function
    [] -> ()
  | url::rest ->
      let referer = match ctx.viewer_base with
         None -> failwith "I need a referer"
       | Some b -> b in
     (* Retrieve the image through the image scheduler *)
     Img.get
       (* The refering document *)
       {document_url = referer; document_stamp = 0}
       (* The link pointing to the image *)
       {h_uri = url; h_context = ctx.viewer_base;
        h_method = None; h_target = None}
       (* Continuation of image loading, stops if window was destroyed *
       (fun o -> if Winfo.exists w then begin
                   Label.configure w [o];
                   add_timer 3000 (fun () -> load_image ctx w rest);
                   ()
                 end);
     Img.flush()
```



```
(*
The applet can be invoked with
<EMBED SRC="../applets/slide.cmo">
<PARAM NAME="function" VALUE=f>
<PARAM VALUE="url of some image">
...other urls...
</EMBED>
*)
let f ctx _ args =
  match ctx.viewer_frame with
    Some w ->
     let l = Label.create w [Text "Slide Show"] in
      pack [l][];
      load_image ctx l args
  | None ->
      failwith "I need a context"

(* Register the applet function. *)
let _ = Applets.register "f" f
```