
Denis Diderot
Université Paris VII
Travail d'Etudes et de Recherche
Surf Bayésien

ALEXANDRE BERTAILS

7 juillet 2004

Table des matières

1	Présentation	2
1.1	Première approche : le Google Bayésien	2
1.2	Commentaires et deuxième approche	3
1.3	Attentes et cahier des charges	4
2	Organisation générale	5
2.1	Architecture de Burfiks	5
2.2	Burfiks-Serveur	7
2.2.1	Communication par socket en Caml	7
2.2.2	Sérialisation de données	9
2.3	burfiks.xul	10
3	Mozilla, XUL, et du courage	11
3.1	Introduction	11
3.2	Présentation rapide de XUL	12
3.3	Manipuler le DOM	13
3.4	Enregistrement dans Chrome	16
3.5	Utilisation de sockets sous Mozilla	17
3.6	Encore plus fort	18
4	Conclusion	23

Chapitre 1

Présentation

1.1 Première approche : le Google Bayésien

[Tiré de <http://www.pps.jussieu.fr/~dicosmo/ACADEMIC/Stages/Bgoogle.html>]

Ces dernières années ont vu un essor spectaculaire des technologies liées au Web, et à l'accroissement exponentiel de l'information disponible en ligne. Nous nous intéressons ici en particulier au problème de la qualité des résultats fournis par la recherche d'information à l'aide de moteurs de recherche tels GOOGLE. L'expérience courante montre que, avec le temps, la qualité des résultats qui sont retournés en réponse à une requête donnée posée par un internaute se dégrade inexorablement : les bons résultats sont inévitablement noyés dans une marée de résultats qui ne nous intéressent pas. Ceci est une conséquence inévitable de l'ambiguïté du langage naturel, et de la difficulté, non seulement pour les non experts, de reformuler la requête initiale avec un choix de mots clefs suffisamment précis pour écarter de la masse des résultats ceux qui ne sont pas pertinents.

Ce problème n'est pas nouveau, et on a pu voir plusieurs solutions qui ont été proposées depuis la généralisation de l'usage du Web :

- classification humaine¹ : des centaines d'employés passent leur temps arpenter le Web et classifient les sites selon des catégories fixées à l'avance
- extraction de réseaux sémantiques² : chaque document se voit associer un réseau de concepts (*topics*) et l'utilisateur peut alors choisir d'affiner sa recherche en incluant ou excluant certains des *topics* qu'il a rencontrés

¹comme dans le cas de *Yahoo*

²comme dans le cas de *LiveTopics*

- exploitation des réseaux endogènes³ : on considère comme document plus significatif celui vers lequel pointe un maximum d'autres documents repérés par la requête
- réseaux d'utilisateurs⁴ : chaque utilisateur peut créer plusieurs *profils* pour ses recherches (ex : profil *cinéfil d'auteur*, profil *acheteur de voiture ancienne*, etc.), et on peut fédérer les utilisateurs en mettant en commun les requêtes correspondants à un même profil

Et pourtant, il y a une approche intéressante, et facile à mettre en oeuvre, qui n'a jusque là pas été mise en pratique, du moins à ma⁵ connaissance : *la recherche par filtrage Bayésien*. Le filtrage Bayésien a été popularisé par Paul Graham pour éliminer automatiquement un maximum de spam⁶ des messages électroniques que nous recevons quotidiennement. Il utilise un théorème élémentaire de théorie des probabilités, connu sous le nom de *formule de Bayes*, pour extraire des messages bons et mauvais déjà reçus une distribution de probabilités qui a des bonnes propriétés de prédiction sur les nouveaux messages qui arrivent.

Le but de ce TER est de tester l'approche Bayésien dans le cas des recherches Web, en développant une fine couche logicielle qui utilise GOOGLE comme moteur de recherche, mais filtre ensuite les pages retrouvées, en utilisant les indications de l'utilisateur pour les séparer en bonnes et mauvaises, de la même façon qu'un filtre anti-spam Bayésien. Cet approche peut être utilement composé avec l'approche des réseaux d'utilisateurs, et la notion de profil de recherche.

La construction du prototype est un travail pratique d'implantation, mais la mise à point du modèle statistique nécessite quelques notions de théorie des probabilités.

1.2 Commentaires et deuxième approche

Le filtrage des résultats d'une requête GOOGLE est vite apparu trop contraignant lors d'une recherche sur internet. Typiquement, le comportement de l'internaute lors d'une recherche est en fait un vrai surf à partir de pages qui ont pu être atteinte depuis GOOGLE.

La deuxième approche de ce TER a donc naturellement été d'implanter un *surf bayésien*. L'utilisateur étant noyé sous une masse de liens qui s'offrent à lui, il aimerait bien être guidé lors de sa recherche vers des pages ayant un

³comme dans le cas de *Google*

⁴comme dans le cas de *Galilei*

⁵M. Di Cosmo

⁶Courrier électronique non sollicité

contenu intéressant. L'idée est donc la suivante : offrir un moyen au surfeur d'obtenir en *temps réel* la pertinence de chacun des liens présents sur une page, et selon un profil donné.

1.3 Attentes et cahier des charges

Voici ce que l'utilisateur est en droit d'exiger de BURFIKS⁷ :

1. une parfaite intégration dans son navigateur Web. Le module ne doit pas gêner le surf, doit être paramétrable, et activable à volonté.
2. une interface claire : BURFIKS est censé être une aide dans le surf. Son utilisation doit être claire et facile.
3. Le principe même de BURFIKS nécessitera un préchargement des pages et de leur contenu. Cela ne doit en rien ralentir la connexion. Le module doit paraître invisible à l'utilisateur, presque *magique*.

Ces points expriment des contraintes sur l'application. MOZILLA⁸ nous est apparu comme étant un bon choix de navigateur cible. Son implémentation est libre, et il existe une communauté de programmeurs qui ne cesse de s'étendre. Surtout, l'utilisation de GECKO au moyen de XUL, JAVASCRIPT et les CSS permettent une grande souplesse d'intégration.

Malheureusement, par problème de temps, ce TER ne sera pas pleinement fonctionnel pour la date fixée. Il est cependant important de prendre le temps de décrire ce qui a été fait, en particulier les problèmes rencontrés, de décrire l'utilisation du XUL dans MOZILLA, et de fixer les axes à suivre pour la suite.

Enfin, M. Di Cosmo⁹ m'a clairement montré que l'idée est dans l'air du temps. La question principale reste donc : est-ce que le modèle envisagé, reposant sur les lois de probabilités de Bayes, est valable ?

⁷nom *temporaire*(?) donné au module

⁸<http://www.mozilla.org/>

⁹<http://www.dicosmo.org>

Chapitre 2

Organisation générale

Nous présentons ici l'organisation générale de l'application. Nous privilégierons dans un premier temps une architecture permettant de manipuler facilement le modèle considéré, afin de pouvoir facilement le faire évoluer. Nous verrons plus tard comment améliorer cette architecture en l'intégrant complètement dans MOZILLA.

2.1 Architecture de Burfiks

L'architecture de BURFIKS est résumée dans la figure 3.4. Quelques explications...

Tout d'abord, nous avons l'interface `burfiks.xul`. C'est elle qui définit l'interface utilisateur dans MOZILLA. Nous verrons de quelle manière elle est implémentée. Ensuite, il y a `Burfiks-Serveur`. C'est le point névralgique du système : il possède les ressources pour évaluer la pertinence des pages, et `burfiks.xul` peut l'interroger à ce sujet. Celui-ci obtient ces pages grâce au proxy POLIPO¹ qui se charge de renseigner le serveur en préchargeant des pages, relativement au surf de l'utilisateur depuis MOZILLA.

Nous détaillons maintenant les modules utilisés. Nous renvoyons à la page Web de POLIPO pour son utilisation. Nous réserverons une section à `Burfiks-Serveur`, et une autre à `burfiks.xul`. Ces sections présenteront les techniques à maîtriser et nécessaires à l'implémentation de BURFIKS. Nous insisterons sur l'approche suivie et sur les fichiers exemples utilisés. En particulier, nous nous attacherons à présenter TOUS les problèmes rencontrés, ainsi que les solutions (bien sûr). Ces sections pourront donc être considérées comme des didacticiels.

¹voir <http://www.pps.jussieu.fr/~jch/software/polipo/>, par Juliusz Chroboczek

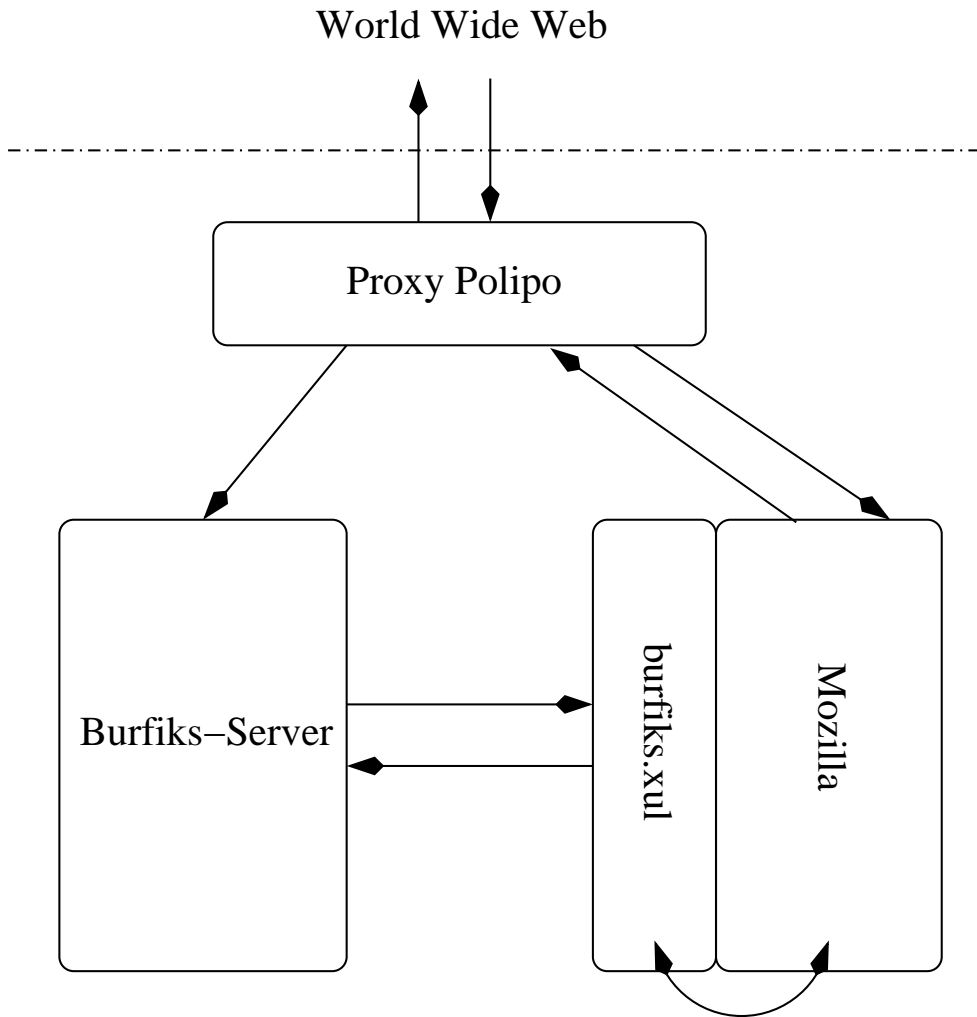


FIG. 2.1 – Architecture de BURFIKS

2.2 Burfiks-Serveur

Rappelons l'intérêt du serveur. Il devra contenir la description des profils gérés par l'utilisateur, une base de données des mots rencontrés dans les pages Web, ainsi que leurs fréquences dans des contenus *black-listés* ou *white-listés*². C'est l'utilisateur qui renseignera le serveur sur ces pages.

2.2.1 Communication par socket en Caml

Le serveur est écrit en OCaml. Ce choix fut pris pour de multiples raisons : le code s'exécute rapidement, le runtime est léger, le code est portable, et on a quand même la possibilité de faire de la sérialisation de données. Problème : maîtriser la communication par socket en OCaml.

Voici pour exemple le code d'un serveur multi-threadé, qui se borne à renvoyer ce qu'il lit sur la socket obtenue. Il attend en paramètre un numéro de port.

```
open Sys;;
open Unix;;

(* si le signal EINTR est capté, f est relance *)
let rec restart_on_EINTR f x =
  try f x with Unix_error (EINTR, _, _) -> restart_on_EINTR f x;;

(* essaie f x, et fait finally de y quoi qu'il arrive *)
let try_finalize f x finally y =
  let res = try f x with exn -> finally y; raise exn in
  finally y;
  res;;

(* cree la socket *)
let install_tcp_server_socket addr =
  let s = socket PF_INET SOCK_STREAM 0 in
  try
    bind s addr;
    listen s 10;
    s
  with z -> close s; raise z;;
```

²nous utiliserons l'anglicisme *whitelist* (resp. *blacklist*) ainsi que ses dérivés pour désigner des pages notées positivement (resp. négativement)

```

(* le serveur : attend l'adresse et la fonction associe de traitement *)
let tcp_server treat_connection addr =
  ignore (signal sigpipe Signal_ignore);
  let server_sock = install_tcp_server_socket addr in
  while true do
    let client = restart_on_EINTR accept server_sock in
      treat_connection server_sock client
  done;;

(* fournit le service *)
let double_fork_treatment server service (client_descr, _ as client) =
  let treat () =
    match fork () with
    | 0 ->
      if fork() <> 0 then exit 0;
      close server; service client; exit 0
    | k ->
      ignore (restart_on_EINTR (waitpid [] k) in
        try_finalize treat () close client_descr;;

(* le serveur lui-même *)
let server () =
  if Array.length Sys.argv < 2 then begin
    prerr_endline "Usage: _client_<port>_<command>_[arg1 _..._ argn]";
    exit 2;
  end;
  let port = int_of_string Sys.argv.(1) in
  let args = Array.sub Sys.argv 2 (Array.length Sys.argv - 2) in
  let addr = ADDR_INET (inet_addr_of_string "0.0.0.0", port) in
  let treat sock (client_sock, client_addr as client) =
    (* log information *)
    begin match client_addr with
      ADDR_INET(caller, _) ->
        prerr_endline ("Connection_from_"
          ^ string_of_inet_addr caller);
    | ADDR_UNIX _ ->
        prerr_endline "Connection_from_the_Unix_domain_(???)";
    end;
  (* connection treatment *)
  let service (s, _) =
    let buffer_size = 4096 in
    let buffer = String.create buffer_size in
    let rec response () =

```

```

    match read s buffer 0 buffer_size with
    | 0 -> ()
    | n -> ignore (write s (String.uppercase buffer) 0 n);
                response () in
    response () in
    double_fork_treatment sock service client in
tcp_server treat addr;;

```

```
handle_unix_error server ();;
```

Et voici la ligne de compilation :

```
ocamlc -o serveur unix.cma serveur.ml
```

Signalons un problème de débutant que j’ai eu sur le serveur, au niveau de l’utilisation du `bind` : je liais la socket à une mauvaise adresse d’écoute. Pour pallier à ce problème, il a suffit de donner l’adresse “0.0.0.0” pour que toutes les adresses soient écoutées.

2.2.2 Sérialisation de données

La sérialisation de données en OCAML repose sur le module `Marshal`. Voici juste un exemple fourni par M. Di Cosmo :

```

[bertails@betehess rapport]> ledit ocaml
Objective Caml version 3.07+2

# let t = Hashtbl.create 16;;
val t : ('a, 'b) Hashtbl.t = <abstr>
# Hashtbl.add t 3 "toto";;
- : unit = ()
# Hashtbl.add t 10 "tata";;
- : unit = ()
# t;;
- : (int, string) Hashtbl.t = <abstr>
# let s = Marshal.to_string t [];;
val s : string =
  "\132\149\000\000\000%\000\000\000\006\000\000\000 [Tronqué]
# let t' : (int, string) Hashtbl.t = Marshal.from_string s 0;;
val t' : (int, string) Hashtbl.t = <abstr>
# Hashtbl.find t' 3;;
- : string = "toto"

```

```
# Hashtbl.find t' 5;;  
Exception: Not_found.
```

On remarquera l'importance de spécifier le type des données lors de la désérialisation.

Et voici des commandes qui nous seront utiles :

```
# Marshal.to_channel;;  
- : out_channel -> 'a -> Marshal.extern_flags list -> unit = <fun>  
# Marshal.from_channel;;  
- : in_channel -> 'a = <fun>
```

2.3 burfiks.xul

`burfiks.xul` est l'interface utilisateur du système. Elle s'intègre dans le navigateur MOZILLA, dans la *SideBar*. Nous réserverons un chapitre à la programmation de BURFIKS, en détaillant le parcours suivi, ainsi que les techniques utilisées et les problèmes rencontrés.

Chapitre 3

Mozilla, XUL, et du courage

Pourquoi du *courage* ? Parce que cette partie, la plus importante du projet, m'est complètement nouvelle, et s'accompagne d'une documentation pas vraiment adaptée ici à nos besoins.

Il existe pourtant de nombreux tutoriaux sur le Web, les plus intéressants étant :

- <http://www.xulfr.org/>, en français
- <http://www.xulplanet.com/>, en anglais, et plus complet
- <http://books.mozdev.org/chapters/>, en anglais, issu du livre *Creating Applications with Mozilla* publié chez *O'Reilly*

Cependant, ces liens sont surtout indiqués pour la programmation en XUL, le langage de manipulation de GECKO, le moteur de rendu graphique de MOZILLA¹.

Ce chapitre présentera mon parcours d'apprentissage de XUL et JAVASCRIPT, et pourra se voir comme un petit tutoriel sur la programmation XUL et les mystères² de MOZILLA.

3.1 Introduction

MOZILLA n'est pas un simple navigateur. C'est un ensemble d'outils multi-plateformes qui s'appuient sur des standards (exemple : CSS) et sur un langage de manipulation d'interfaces : XUL. Comme langage applicatif, il utilise JAVASCRIPT. Cela nous posera d'ailleurs quelques problèmes par la suite, car JAVASCRIPT est un langage prévu pour la réalisation de sites Web et, par conséquent, a des *contraintes fortes de sécurité* : il n'est pas possible,

¹MOZILLA est écrit en XUL et JAVASCRIPT, au-dessus de GECKO

²et il y en a encore!!!

par exemple, d'accéder à des fichiers sur disque dur ou d'ouvrir des sockets de communication.

3.2 Présentation rapide de XUL

XUL est un sous-langage du XML. Il permet de décrire très facilement des interfaces graphiques. Le rendu est très riche et plutôt rapide³. Lorsqu'il s'intègre dans une page Web⁴, il est possible de manipuler le document comme avec du XML, grâce aux spécifications du DOM⁵. C'est là que JAVASCRIPT prend le relai en permettant de rendre l'interface interactive, en manipulant la représentation-objet DOM du document. Je décrirai plus tard comment faire cela.

XUL se présente donc comme du XML, mais il faut déclarer l'*espace de nom* (*namespace* en anglais) pour pouvoir l'utiliser. C'est le rôle de l'attribut `xmlns=...` dans la balise `xmlns`, dont la présence est obligatoire. L'URL n'est en fait pas téléchargée, car MOZILLA la connaît en interne. On peut aussi préciser le type d'encodage, ainsi que le style de page (dans une feuille de style fichier CSS, mais ici on se contentera du style fourni par défaut).

Voici un exemple simple de fichier XUL, que l'on peut engistrer dans un fichier `.xul`, et lancer depuis MOZILLA (sous certaines conditions, mais ça marche pour les fichiers locaux n'utilisant pas JAVASCRIPT) :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<window
  id="findfile-window"
  title="Recherche de fichiers"
  orient="horizontal"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<vbox flex="1">

  <description>
    Entrez votre critère de recherche ci dessous et appuyer sur le
    bouton Rechercher.
```

³aucun ralentissement notable, et il paraît que la version 1.7 de MOZILLA améliore encore cela

⁴il peut servir à écrire des applications autonomes, qui ne s'appuient pas sur MOZILLA

⁵*Document Object Model*

```

</description>

<spacer style="height: 10px"/>
<groupbox orient="horizontal">
  <caption label="Critères de recherche"/>
  <menulist id="searchtype">
    <menupopup>
      <menuitem label="Nom"/>
      <menuitem label="Taille"/>
      <menuitem label="Date de modification"/>
    </menupopup>
  </menulist>
  <spacer style="width: 10px;"/>
  <menulist id="searchmode">
    <menupopup>
      <menuitem label="Est"/>
      <menuitem label="N'est pas"/>
    </menupopup>
  </menulist>
  <spacer style="width: 10px;"/>
  <textbox id="find-text" flex="1" style="min-width: 15em;"/>
</groupbox>

<spacer style="height: 10px"/>

<hbox>
  <spacer flex="1"/>
  <button id="find-button" label="Rechercher" default="true"/>
  <button id="cancel-button" label="Annuler"/>
</hbox>

</vbox>

</window>

```

Le résultat est celui de la figure 3.1.

3.3 Manipuler le DOM

Il nous serait maintenant utile de pouvoir modifier l'interface de façon dynamique. C'est là qu'intervient le JAVASCRIPT, qui permet de modifier

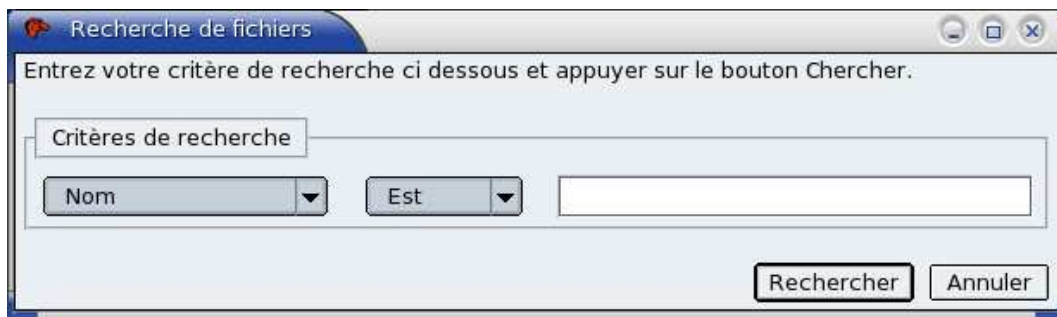


FIG. 3.1 – Exemple d’interface XUL

le DOM. Quelques explications : le DOM est une API standard qui permet d’obtenir une représentation d’un document XML (et donc XUL ou HTML), de naviguer dans cette représentation, et de la modifier. Pour résumer : modifier l’objet DOM d’une page, cela a pour effet de modifier l’apparence.

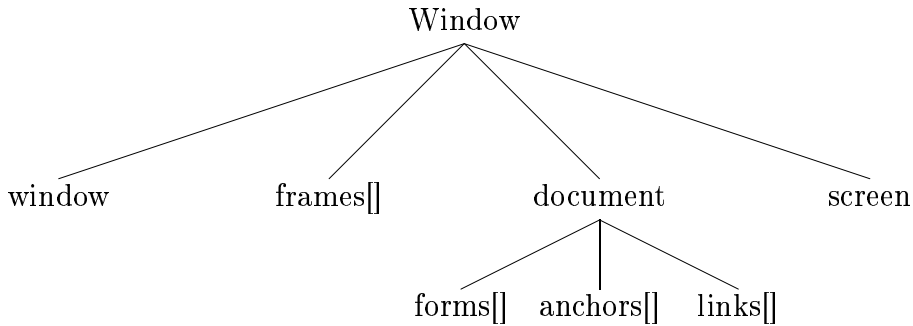
Le JAVASCRIPT offre la possibilité de modifier le DOM. Signalons que le code JAVASCRIPT peut se trouver dans un fichier séparé, mais nous le déclarerons dans une balise `script`. Si aucun attribut n’est spécifié, MOZILLA considère qu’il a affaire à du JAVASCRIPT. Le JAVASCRIPT est un langage objet, dont la syntaxe ressemble beaucoup à celle de Java.

Les objets DOM se présentent sous la structure d’arbres⁶. On parle alors de noeuds, sachant qu’il existe différents types de noeuds. Nous n’utiliserons que ceux de type `Element` et `Text`. Pour les noeuds de types `Element`, signalons les champs suivants :

- `Node firstChild` qui correspond au premier fils, `null` s’il n’y en a pas
- `Node nextSibling` qui correspond au frère suivant, `null` s’il n’y en a pas

Dans le DOM, l’objet principal est celui qui représente le document : `document`. Les différentes fenêtres sont accessibles au travers du fils-attribut `window`, dont voici une partie de la hiérarchie, au niveau 0 du DOM :

⁶avec des pointeurs sur le premier fils, le père, les frères



De manière générale, voici quels mécanismes utiliser pour la manipulation du DOM :

1. la méthode `getElementById(String id)`, qui renvoie le premier noeud identifié par la chaîne `id`. Pour nommer un noeud, il a suffi de rajouter l'attribut `id` dans la balise considérée
2. la méthode `createElement(String id)`, qui renvoie un nouveau noeud, qui correspond donc à l'insertion d'une balise `<id />`
3. la méthode `setAttribute(String attr, String value)`, qui insère l'attribut `attr` en lui associant la valeur `value`
4. la méthode `appendChild(Node node)` qui ajoute un noeud fils à un noeud existant (pas forcément encore rattaché au document).

Voici maintenant un exemple très simple, qui ajoute un nouveau bouton à chaque fois que l'on clique sur le premier :

```

<?xml version="1.0"?>

<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<window id="example-window" title="Example 7.2.1"
        xmlns:html="http://www.w3.org/1999/xhtml"
        xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
<script>
function addButton()
{
    var aBox = document.getElementById("aBox");

    var button = document.createElement("button");
    button.setAttribute("label", "A Button");
    aBox.appendChild(button);
    window.sizeToContent();
  
```

```

}
</script>

<box id="aBox">
  <button label="Add" onclick="addButton();" />
</box>

</window>

```

Cela donne quelque chose qui ressemble à la figure 3.2.



FIG. 3.2 – L'exemple addButton

Pour finir ce chapitre, signalons un problème que je n'ai pas su identifier pendant un bon moment : lorsqu'on déclare des scripts à l'intérieur même de documents XUL, il est interdit d'utiliser les caractères `<` et `&`, car ils sont intrinsèques à la grammaire du XML !!! Pour pouvoir les utiliser, la méthode la plus simple est d'utiliser à leur place `<` et `&`. C'est fou le temps que j'ai passé sur ce problème, tout ça parce que la seule erreur qui ressortait était *une simple erreur de parsing*, sans explications supplémentaires. Grrrr!!!

3.4 Enregistrement dans Chrome

Nous avons déjà dit que JAVASCRIPT, pour des raisons de sécurité, ne pouvait pas faire certaines opérations. Ce n'est pas tout à fait vrai : il faut pour cela avoir les privilèges adéquats. Une solution est de signer électroniquement les documents XUL. Mais la plus simple reste l'enregistrement dans le répertoire **Chrome**⁷. Une application est réellement enregistrée et possède des droits étendus si elle possède une entrée dans le fichier `chrome/installed-chrome.txt`.

Une entrée valide pour le répertoire `chrome/burfiiks/content` dans *Chrome* serait :

⁷CHROME est normalement un sous-répertoire du répertoire d'installation principal de MOZILLA

`content,install,url,resource:/chrome/burfiiks/content/`

On peut aussi déclarer dans *Chrome* les emplacements des skins CSS, des fichiers de description RDF, etc.

Le répertoire cible doit contenir la description de l'application, dans le fichier `contents.rdf`. A adapter selon les besoins :

```
<?xml version="1.0"?>
<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:chrome="http://www.mozilla.org/rdf/chrome#">

  <!-- list all the packages being supplied by this jar -->
  <RDF:Seq about="urn:mozilla:package:root">
    <RDF:li resource="urn:mozilla:package:burfiiks"/>
  </RDF:Seq>

  <!-- package information -->
  <RDF:Description about="urn:mozilla:package:burfiiks"
    chrome:displayName="burfiiks"
    chrome:author="Alexandre BERTAILS"
    chrome:name="burfiiks">
  </RDF:Description>

</RDF:RDF>
```

3.5 Utilisation de sockets sous Mozilla

Maintenant que nous savons enregistrer des applications dans *Chrome* et leur donner des droits étendus, nous pouvons passer à l'étude d'une certaine bibliothèque de fonctions JAVASCRIPT : JSLIB⁸. Elle fournit un ensemble de fonctions utiles au développement d'applications MOZILLA, et en particulier un système de *sockets*, avec le fichier `socket.js`.

Un exemple d'utilisation des sockets à l'aide de cette bibliothèque est disponible dans le `.xpi` livré avec la bibliothèque, dans `jslib/samples/socket.xul`, et est normalement disponible avec le document que vous lisez.

L'application est simple, et propose juste quelques boutons, pour ouvrir et fermer une socket sur un port donné, envoyer et recevoir des messages. Pour un exemple de session (voir la figure 3.3 pour la partie XUL), on peut utiliser l'outil `netcat` :

⁸disponible à <http://jslib.mozdev.org/>

```
[bertails@betehehess rapport]> nc -v -v -l localhost -p 1234
listening on [any] 1234 ...
connect to [127.0.0.1] from localhost [127.0.0.1] 33652
salut netcat !!!
sent 0, rcvd 17
[bertails@betehehess rapport]>
```

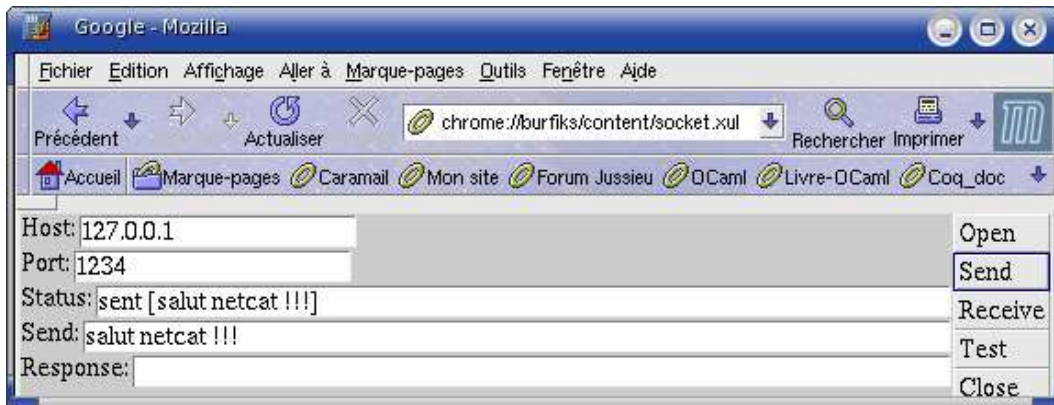


FIG. 3.3 – Utilisation de sockets

3.6 Encore plus fort

Bon, maintenant que nous sommes super-forts en XUL-JAVASCRIPT-DOM, on va commencer à se rapprocher de BURFIKS. On se propose d'enregistrer une application dans la *SideBar*. Celle-ci devra afficher tous les liens hypertextes de la page principale. La page se mettra à jour d'elle-même, et les liens qu'elle propose seront cliquables, et s'ouvriront dans la page principale.

Pour commencer, nous voulons engistrer notre application dans la SideBar. Celle-ci est fait juste une mini-page. Pour cela, nous pourrions utiliser les outils mis à disposition par MOZILLA, avec la commande magique JAVASCRIPT⁹.

```
window.sidebar.addPanel("Caml",
    "http://caml.inria.fr/mozcaml/content/main.xul", "");
```

Cependant, il y a un problème : MOZILLA n'accepte d'enregistrer que des liens commençant par *http ://*, pour des raisons de sécurité. C'est dommage,

⁹c'est dans le script d'installation de Mozcaml : <http://caml.inria.fr/mozcaml/>

car dans le cas d'une URL enregistrée dans *Chrome*, on a déjà accordé notre confiance dans le lien. Qu'à cela ne tienne, nous pouvons ajouter manuellement l'entrée voulue. Pour cela, il faut trouver le fichier qui devrait avoir la référence `~/mozilla/default/vjd6lj52.slt/panel.rdf`, et y ajouter la description suivante :

```
<RDF:Description about="urn:sidebar:3rdparty-panel:chrome://burfiks/content/burfiks
    NC:title="Burfiks"
    NC:content="chrome://burfiks/content/burfiks.xul"
    NC:persist="false" />
```

Une fois cela fait, il suffit juste d'activer le bon onglet (l'opération nécessite un redémarrage).

Ensuite, il nous faut connaître comment rendre un élément cliquable en XUL. Il faut juste utiliser le bon *namespace* ! ici, c'est bien sûr celui de HTML. La bonne commande JAVASCRIPT est la suivante, sachant que le bon namespace est `http://www.w3.org/1999/xhtml` :

```
createElementNS(String NameSpaceURL, String element)
```

Maintenant, il faut savoir comment rendre un élément de la *SideBar* cliquable, et faire en sorte que la page s'ouvre dans la page principale. Pour cela, il faut faire quelque chose du genre¹⁰ :

```
var ahtml = document.createElementNS("http://www.w3.org/1999/xhtml", "a");
ahtml.setAttribute("href", url);
ahtml.setAttribute("target", "_content");
```

Depuis la *SideBar*, l'élément `window` de la page principale est désigné par `window._content`. On a ensuite accès à son champ-tableau `links[]`, qui contient toutes les URLs de la page. On ne gardera que celles relevant du protocole *http*. Dans la version actuelle, la mise à jour n'est pas détectée proprement. Ici, cela correspond à une fonction qui s'exécute toutes les secondes. Cela donne le listing suivant :

```
<?xml version="1.0"?>

<!DOCTYPE window>

<window id="burfiks-devel"
```

¹⁰où `url` est la chaîne de caractères désignant l'URL. C'est la dernière ligne qui indique l'ouverture dans la page principale

```

title="burfiks"
style="background-color: #cccccc; width: 500px;"
xmlns:html="http://www.w3.org/1999/xhtml"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
onload="initialize();"
>
// importation de jslib, qui doit être installé par root
<script type="application/x-javascript" src="chrome://jslib/content/jslib.js" />
// partie javascript
<script type="application/x-javascript">
  try {
    // enablePrivilege is required if not running chrome'd
    // (other tweaks might apply, check out public.mozdev.jslib)
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
    include(jslib_socket);
  }
  catch( e ) { alert( e ); }

  var __ref;

  function initialize()
  {
    setInterval("addAllLinks()", 2000);
  }

  function addHTMLElement( url, string )
  {
    var aBox = document.getElementById("array");
    var lab = document.createElement("label");
    lab.setAttribute("value", string);
    var ahtml =
      document.createElementNS("http://www.w3.org/1999/xhtml", "a");
    ahtml.setAttribute("href", url);
    ahtml.setAttribute("target", "_content");

    ahtml.appendChild(lab);
    aBox.appendChild(ahtml);
  }

  function removeChildren( node )
  {

```

```

        while(node.hasChildNodes())
            node.removeChild(node.firstChild);
    }

    function addAllLinks()
    {
        var win = window._content;
        var doc = win.document;
        var ref = win.location.href;

        if(ref != __ref || first_time ||
            !document.getElementById("array").hasChildNodes()){
            first_time = false;
            __ref = ref + "";

            removeChildren(document.getElementById("array"));
            var size = doc.links.length;
            for(var i = 0; i < size; i++) {
                if(doc.links[i].protocol == "http:")
                    addHTMLElement(doc.links[i].href, doc.links[i].text);
            }
        }
    }
}
</script>

<vbox id="array" />
</window>

```



FIG. 3.4 – BURFIKS en développement

Chapitre 4

Conclusion

Ce TER a été pour moi l'occasion d'acquérir un bon bagage technique dans le domaine des outils du Web (avec JAVASCRIPT, XML, CSS et le DOM), ainsi que dans le développement d'application avec le framework MOZILLA.

Cependant, à cause de mes lacunes techniques sur le sujet et faute d'une documentation pas encore assez adaptée, j'ai passé l'essentiel de mon temps disponible à la maîtrise de tous les éléments qui entrent en compte. Je pense aujourd'hui y être à peu près parvenu¹.

Mais le projet doit avancer. Je peux maintenant fixer des objectifs plus précis, car je sais à peu près ce qu'il est possible de faire. Ce sera le travail à fournir durant cet été, avec quelques amis intéressés par le projet. Voici comment je pense qu'il faudra se répartir le travail : un groupe pourrait proposer une version de BURFIKS complète et finalisée (avec en particulier la gestion des profils) pour la fin Juillet, tandis que l'autre groupe pourrait se focaliser sur un côté technique important : XPCOM².

En particulier, il faudra étudier certains projets du libre³ de la communauté MOZILLA, pour voir comment certaines caractéristiques sont implémentées.

Pour finir, voici une liste (non exhaustive) d'éléments ou caractéristiques auxquels il nous faudra considérer :

- la maîtrise d'XPCOM
- la maîtrise des fichiers d'installation *.xpi*
- interfacer le C et OCAML

¹en particulier, je sais maintenant où aller chercher en cas de besoin

²c'est une interface qui permet de manipuler des objets, qu'on aura écrits en C par exemple

³l'un des plus intéressants reste DOMINSPECTOR

- ajouter des barres de couleurs derrière le texte des liens pour indiquer rapidement la pertinence des liens
- à la manière de DOMINSPECTOR, détourer les liens de la page principale lorsqu'ils sont sous le curseur de la souris (nécessite la maîtrise des événements en Javascript)
- trier les entrées selon leur pertinence
- trouver comment DOMINPECTOR prend conscience d'un changement de page
- essayer de s'abstraire de POLIPO, en faisant le téléchargement des pages depuis MOZILLA pour les évaluer
- s'abstraire de la bibliothèque JSLIB, et utiliser les fonctions avancées HTTP et implémenter un mini-serveur HTTP pour la communication. On peut imaginer y récupérer des DTDs prêtes à l'emploi
- ...

Quelques remerciements

Je tiens à remercier (l'ordre n'est pas significatif) :

- M. Roberto Di Cosmo, mon responsable de TER, pour m'avoir donné un sujet intéressant, ainsi que pour son aide technique et sa disponibilité
- M. Juliusz Chroboczek, pour son aide technique, sa grande disponibilité, ainsi que pour tous les chocolats qu'il m'a offerts au café d'en face
- ma chère et tendre Eve, qui m'a admirablement supporté et conforté alors que je me tirais les cheveux devant le petit lézard rouge

Quelques liens, en vrac

- <http://www.mozilla.org/>
- <http://www.dicosmo.org/>
- <http://www.pps.jussieu.fr/~jch/software/polipo/>
- <http://www.xulfr.org/>
- <http://www.xulplanet.com/>
- <http://books.mozdev.org/chapters/>
- <http://jslib.mozdev.org/>
- <http://caml.inria.fr/>
- <http://www.paulgraham.com/spam.html>
- <http://www.paulgraham.com/naivebayes.html>
- <http://www.mathpages.com/home/kmath267.htm>