Université de bourgogne
Multimédia

par

Paul Albert
Johan Jegard
Julien Virey

Création d'une application web permettant de générer une page web

Rapport rendu le 4 juin 2010

Enseignants :
Mme. Bourgeois-République
Mme. Dominique Faudo

Nous tenons tout d'abord à nous excuser de ne pas vous avoir demandé l'autorisation de mettre à trois pour faire ce projet. Ce n'était pas du tout correct de notre part. Nous espérons néanmoins que vous apprécierez le travail fournis.

Paul, Johan et Julien

Sommaire

Liste des figures

Introduction

Le net d'aujourd'hui est couramment utilisé pour effectuer des tâches tant professionnelles que personnelles. Mais lorsqu'un utilisateur veut apporter sa contribution en ajoutant du contenu, la tâche est tout de suite moins aisée que lorsqu'il doit effectuer une simple recherche. Divers outils de création de pages web hors ligne, à installer sur son propre ordinateur, sont disponibles mais il n'existe que trop peu de solutions simples d'accès ne nécessitant aucune installation.

Avec un accès toujours plus simple à internet, grâce à l'adsl, la 3g et bientôt la fibre, le nombre de machine interconnectées ne cesse de grandir. Et avec le *cloud computing*, qui consiste à rendre disponible les ressources pour nos machines en ligne, les applications ont de plus en plus tendance à être décentralisées et accessibles en lignes. C'est le cas pour des applications comme la suite bureautique de google, *google Document* [7], photoshop online [1] utilisant la machine virtuelle *Air* [2] ou encore l'excellent laboratoire musical *audiotool* [4]. Une application web pour réaliser des pages web est donc complètement dans l'air du temps.

Analyse

1

Sommaire

1 Analyse du projet

1 Rappel du sujet

Le but de ce projet est de créer une application web permettant de générer facilement une page web static en *XHTML* strict pouvant contenir toutes les balises autorisées par ce standard. Cette page doit être générée grâce au langage *perl*. Les éléments HTML doivent être ajoutés facilement, avec l'utilisation de la souris par exemple. Enfin, toute référence au javascript ou au CSS en ligne doit être supprimée.

2 Cahier des charges

a Public ciblé

Nous devons donc réaliser une application web permettant de créer une page web aisément. Les utilisateurs ciblés sont donc des débutants en programmation, voulant créer une page web facilement, en ayant accès facilement aux différents éléments HTML.

L'application n'est par contre pas destinée à des néophytes sans aucunes connaissances, car il est nécessaire de connaître le rôle des différentes balises HTML afin de concevoir sa page. Cependant, toute personne aidée d'une bonne documentation [13] peut se débrouiller.

b Application web

Pour l'application web, il faut vraiment que l'utilisation soit ergonomique et intuitive. C'est pourquoi nous avons décidé d'utiliser le *glisser/déposer*, qui est une action très visuelle parfaitement indiquée pour placer des éléments. Nous pensions au début prendre des éléments à la souris pour les déposer sur un *textarea*, mais nous avons très vite pensé utiliser une représentation du code en arbre. HTML étant un langage basé sur la notation XML, la structuration imbriquée des balises se prête parfaitement à cette représentation. Il suffit alors pour l'utilisateur de sélectionner un élément pour le glisser à l'endroit de son choix dans l'arbre HTML de sa page. Le code devra bien entendu être affiché et mis à jour en fonction des choix de l'utilisateur.

L'application devra également présenter en temps réel un aperçu de la page générée, pour visualiser l'avancement du travail accompli. Finalement, l'utilisateur doit être capable de récupérer son travail, sous forme de fichier HTML et CSS.

c Résumé

Notre application devra donc comporter :
- Une liste d'éléments HTML.
- Un arbre HTML présentant la page à construire.
- La gestion du glisser/déposer, pour insérer des éléments dans l'arbre HTML.
- Deux zone de code présentant le code HTML et le code CSS.
- Un formulaire permettant d'ajouter facilement des propriétés CSS aux éléments HTML.
- Une zone d'aperçu présentant en temps réel l'évolution de la page construite.
- Une bouton permettant d'enregistrer le travail accompli.

2 Mais comment va-t-on le faire ce projet ?

1 Application web

Une grande partie de l'application étant exécutée par le navigateur client, nous avons dû utiliser en majorité du javascript pour la construction des pages. Le but étant de construire "briques par briques" une page web digne de ce nom, nous avons donc choisi d'appréhender le sujet avec une approche dite "de construction bas niveau" : l'utilisateur dispose d'une variété d'objets élémentaires d'une page HTML puis les dispose et les imbrique comme bon lui semble, pour finir par leur donner leur apparence finale. Pour ce faire, il a été décidé d'utiliser une représentation du code en arbre, et de gérer le glisser/déposer ainsi que toutes les exceptions le concernant en fonction du type (inline, block, élément inline auto-refermé tel que *<img />*) de l'élément à déposer dans l'arbre, et de l'endroit où l'utilisateur veut le déposer. De plus, chaque élément possédant des attributs lui étant propres (que ce soit graphiques ou fonctionnels), il a fallu créer une interface de personnalisation de ces attributs.

2 Script perl

Perl [8] étant un langage exécuté côté serveur, le script cgi/perl permet de créer le fichier de la page finale et d'y placer son contenu, après que celle-ci ait été conçue par l'utilisateur à l'aide de l'interface HTML/javascript. Pour transférer la page à créer, du navigateur au script perl sur le serveur, on a mis en place une requête asynchrone de type ajax à l'aide de l'objet javascript XMLHttpRequest, qui nous permet d'envoyer la page à générer sans recharger la page courante dans le navigateur, et de récupérer dans une balise object le contenu de cette page comme aperçu.

3 Technologies utilisées

Pour réaliser l'application web, nous avons utilisé la librairie javascript *Mootools* [10] (licence MIT [12]) et son plugin de gestion d'arbres *Miftree* [9] (licence creative commons by-nc-sa [6]). L'application est codé en *HTML5* et *CSS3*, afin de profiter des dernières fonctionnalités du langage HTML. Nous avons également utilisé la librairie *EditArea* [5] (licence LGPL [15]) afin de colorer le code HTML et CSS dans les zones d'édition.

En raison de l'utilisation de technologies récentes comme le CSS3 et l'HTML5, l'application n'est que peu compatible avec *Internet Explorer*. Elle fonctionne sous internet explorer 8, mais pour profiter au mieux de notre projet, il est préférable d'utiliser un navigateur respectant les standards du web, tel que *Firefox* [11] ou *Chromium* [14].

Programmation

2

Sommaire

Nous détaillerons dans ce chapitre les différentes étapes de notre travail.

1 La base du site

1 HTML

Nous avons commencé par définir une structure de site classique, avec une entête munie d'un menu, un corps qui contiendra l'application, et un pied de page.

Après l'analyse théorique, nous avons délimité le rôle de chaque composant de notre application :

- La liste d'éléments HTML sera un arbre d'éléments.
- L'utilisateur pourra glisser/déposer ces éléments dans un second arbre, celui ci gérant l'indentation des balises.
- Le bloc de code HTML sera non éditable et affichera un aperçu en temps réel du code de la page créée avec l'arbre.
- Le bloc de code CSS sera éditable et permettra la modification *à la volée* du code CSS associé à la page.
- Le bloc de visualisation permettra un aperçu en temps réel de la page créée avec l'arbre.

Il a donc fallu positionner ces différents blocs dans le corps de la page principale afin qu'ils occupent un espace homogène, un *grand rectangle* sans espaces vides, et que les modules de l'arbre et de la sélection des éléments à placer soient côtes-à-côtes pour faciliter l'étape de glisser/déposer. De plus nous avons choisi d'implémenter une fonctionnalité de glisser/déposer sur les blocs afin de permettre à l'utilisateur de les placer à sa convenance. Et cela est fait en HTML5 pur, sans librairie supplémentaire.

Il a également fallu *pré-placer* les fenêtres cachées au lancement de la page, les fenêtres d'édition de contenu et de stylisation, qui apparaissent respectivement lorsqu'on dépose un élément de type *contenu* dans l'arbre et quand on édite le style d'un élément de l'arbre.

Pour ce qui est des blocs de code, nous les avons incorporés sous forme de *textarea* améliorés à l'aide du plugin javascript *EditArea*, leur permettant, entre autre, de colorer syntaxiquement le code HTML et CSS pour qu'il soit plus facilement compréhensible par l'utilisateur.

Pour finir la description structurelle de la page, nous avons placé un menu en bas de l'entête : le premier bouton permet de retourner sur l'application, cela recharge la page, les deux autres boutons mènent vers la page d'aide et la page *à propos*.

2 CSS

Nous avons décidé d'utiliser la troisième révision des feuilles de style en cascade. Cela nous a permit de styliser notre site tout en nous amusant avec les ombres et les coins arrondis.

Le CSS3 permet de gérer indépendamment les effets sous le moteur gecko de mozilla, ou sous webkit (chromium & safari). Par exemple, pour faire les coins arrondis et les ombres sur le corps de notre application, nous utilisons les paramètres :

```
#corps {
    -moz-border-radius: 8px;
    -webkit-border-radius: 8px;
    -moz-box-shadow: -2px 0px 100px black, 2px 0px 50px black;
    -webkit-box-shadow: -2px 0px 100px black, 2px 0px 50px black;
}
```

Deux fichiers CSS on été créés, `style.css` pour la charte globale de notre site, et `tree.css` pour configurer nos arbres. Tous les icônes des arbres, ainsi que les icônes du glisser/déposer sont définis dans ce fichier de style.

3 Charte graphique

La charte graphique de notre site à été soigneusement élaborée à l'aide d'*Inkscape* pour la schématisation et de *the GIMP* pour la réalisation finale.

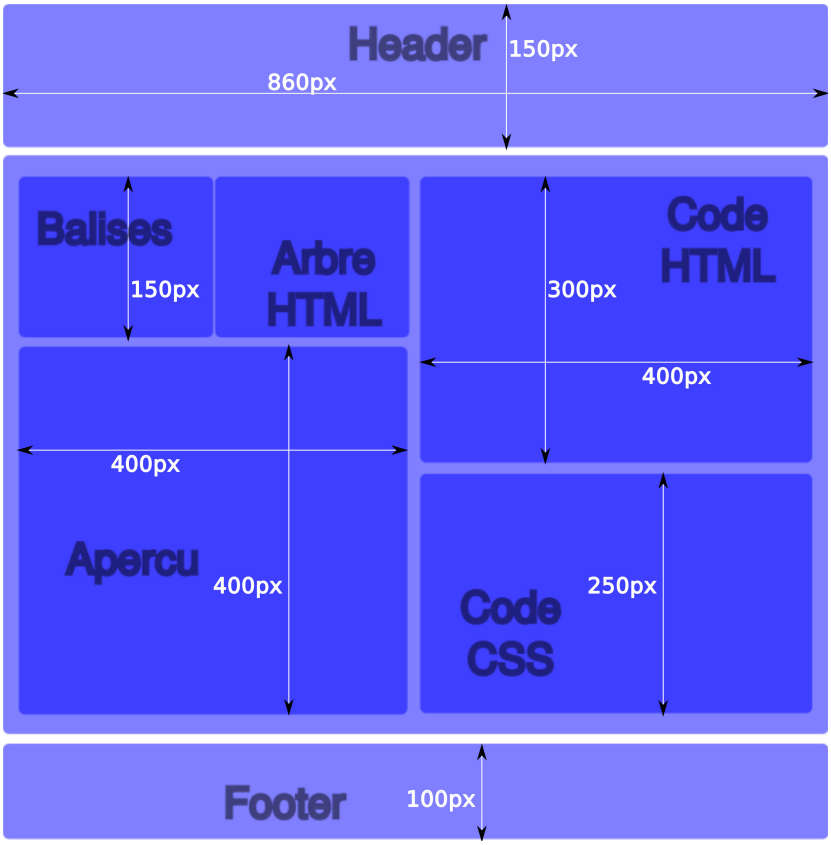

FIGURE 2.1 – *Charte graphique de notre site web.*

2 Arbre HTML et glisser/déposer

1 Ajouter un élément

a Ajouter ou déplacer un élément par Glisser-Deposer

L'arbre d'éléments et l'arbre HTML sont deux arbres générés à l'aide du plugin MifTree qui utilise, lui, les fonctions de glisser/déposer de mootools. Lors de l'action *déposer* d'un élément du premier arbre dans le second, un événement **beforeDrop** est invoqué et permet de tester si l'élément déposé peut l'être ici et d'agir en conséquence en modifiant ou non l'arbre HTML. Le langage HTML étant constitué de balises ouvrantes, fermantes, auto-refermantes et de contenus, il a fallu trouver une astuce pour pouvoir aussi insérer du contenu dans l'arbre. Celle que nous avons trouvé consiste à créer un élément fictif auto-refermant, appelé dans le programme **__contenu__**, qui sera remplacé dans le code HTML final par son contenu.

Pour ajouter tous les événements aux composants de la page, nous avons également utilisé une fonctionnalité de mootools, le méta-événement **domready** appelé lorsque que le dom est entièrement chargé, mais que la page n'est pas encore graphiquement chargé. De ce fait un **addevent** appelé dans ce méta-événement permet d'ajouter efficacement un événement sur un objet. On est alors sûr qu'il sera dans le dom, mais on aura pas besoin d'attendre l'affichage complet de la page comme avec **body.onLoad() ;**.

b Autorisation des balises

Lors de l'événement **beforeDrop**, on vérifie donc si un élément à le droit d'être inséré à cet endroit dans l'arbre, par exemple il est interdit de mettre un **div** dans le **head**, pour ce faire, chaque type d'élément dispose d'un tableau d'autres éléments dans lesquels il a le droit d'être déposé et, juste avant le **drop**, on compare si le futur élément fils est bien dans la liste des éléments parents autorisés, si tel n'est pas le cas, on inhibe l'action de création de l'élément dans l'arbre HTML.

c Indentation

Le code généré doit être facilement lisible, c'est à dire avec une balise par ligne, et indenté, comme dans un éditeur de développement intégré. Lors du parcours de l'arbre à transformer en code, à l'ajout d'une balise dans l'arbre, on regarde si une balise est incluse dans l'élément père, si c'est une balise de fermeture, style </ ... >, ou si la balise la précédent en était une et on agit en conséquence dans la génération du code.

2 Supprimer un élément

Pour supprimer un élément, l'utilisateur n'a qu'à sélectionner ce dernier et à presser la touche **suppr** de son clavier (interceptions de l'événement **keyPress**). La fonction associée à cet événement supprime alors tous le nœud (la balise elle-même et ses fils) ainsi que la balise suivante s'il s'agit d'une balise refermable (la balise sera alors de la forme </... >).

3 Code CSS

a Formulaire CSS

Quand on double-clique sur un élément de l'arbre HTML, on peut alors éditer ses propriétés CSS : une fenêtre apparaît par dessus la fenêtre de code et une liste non exhaustive de propriétés CSS apparaît, ce qui permet d'accélérer la saisie de ceux-ci dans le *textearea* à côté. De plus deux champs permettent de choisir un id ou une classe pour la balise en édition, champs que l'on valide par l'appui sur la touche entrée (grâce au surchargement de la méthode **keyPress**) et qui ajoutent automatiquement le sélecteur CSS correspondant. Le code saisi s'ajoute alors au contenu déjà présent dans la variable contenant tout le CSS ainsi qu'au *textearea* du code CSS et la méthode de mise à jours de l'aperçu est alors appelée.

b Édition à la volée

On peut également éditer à la main le code CSS présent dans le *textearea*, à chaque appui sur une touche du clavier dans le *editarea*, grâce à la définition d'une fonction de callback à la création du *editarea*, on remet à jour la variable du CSS. Lors de l'appui sur le bouton *générer* sous le code CSS, on ré-envoie le code au script perl sur le serveur pour qu'il régénère le fichier CSS et donc l'aperçu.

3 Aperçu de la page

1 Ajax

L'aperçu de la conception de la page se fait en temps réel pour que l'utilisateur puisse visualiser son travail au fur et à mesure qu'il ajoute des balises dans l'arbre HTML. L'ajax est géré dans le fichier `Callserver.js`. Voici le contructeur ajax :

```
1 function CallServer(){
2     this.xhr;
3     this.server_response;
4 
5     this.createXMLHTTPRequest = createXMLHTTPRequest;
6     this.sendData = sendData;
7     this.displayAnswer = displayAnswer;
8     this.launch = launch;
9 }
```

Les données sont envoyées au `cgi-perl` via la fonction :

```
function sendData(data){
    var xhr = this.xhr;
    xhr.open("POST", "/cgi-bin/formulaire", true);
    xhr.setRequestHeader("Content-type", "application/x-www-form-
       urlencoded");

    xhr.send(data);
    //
    if (xhr.readyState == 4)
    {
        this.server_response = xhr.responseText;
    }
}
```

2 Perl

Le script `Perl` s'occupe de récupérer le code HTML et le code CSS afin de créer les fichiers sur le serveur. Le script commence donc par vérifier qu'il a bien l'autorisation d'écrire dans le répertoire de `/resultat` :

```
my $CSS = 'resultat/style.CSS';
my $HTML='resultat/index.HTML';

open(FILECSS,">$CSS") || die ("Erreur d'ouverture en ecriture ");
open(FILEHTML,">$HTML") || die ("Erreur d'ouverture en ecriture ");
}
```

On récupère ensuite l'entrée standard avant d'effectuer les traitements nécessaires pour peupler les fichiers index.html et style.css.

```
read(STDIN, $in, $ENV{CONTENT_LENGTH});

@champs = split(/&/,$in);
print "Content-type: text/HTML\n\n";
print '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//
   EN">', "\n";
print "<HTML><head><title>$CSS</title></head><body>\n";
# traitement de chaque élément $e de la liste @champs
foreach $e (@champs) {
    # dissocie chaque élément, de la forme nom::valeur,
    # en une paire de variable (nom,valeur)
    ($nom, $valeur) = split(/::/,$e);
    # transforme tous les caractères saisis en minuscules
    $valeur = tr/A-Z/a-z/;
    # impression qui va bien
```

```
    if($nom eq "CSS"){
        print FILECSS "$valeur";
    }
    if($nom eq "HTML"){
        print $valeur;
        print FILEHTML "$valeur";
    }
    # crée à partir du tableau @champs,
    # une liste associative %champs
    $champs{$nom}=$valeur;
}
print FILECSS "\n".'}';
close (FILEHTML);
close (FILECSS);
print "</body></HTML>\n";
```

3 Récupérer les fichiers

On récupère les fichiers à l'aide d'une page php qui change son entête en téléchargement forcé et qui joint un fichier. La page est alors appelée plusieurs fois pour récupérer les fichiers créés avec un timer entre chaque appel pour permettre au navigateur de gérer la requête.

```
<?php
    $file=$_GET['file'];
    if (($file != "") && (file_exists("./" . basename($file)))) {
        $size = filesize("./" . basename($file));
        header("Content-Type: application/force-download; name=\""
            . basename($file) . "\"");
        header("Content-Transfer-Encoding: binary");
        header("Content-Length: $size");
        header("Content-Disposition: attachment; filename=\"" .
            basename($file) . "\"");
        header("Expires: 0");
        header("Cache-Control: no-cache, must-revalidate");
        header("Pragma: no-cache");
        readfile("./" . basename($file));
        exit();
    }
?>
```

Conclusion

Une longue phase de conception théorique impliquant des choix, aussi bien ergonomiques (représentation en arbre, glisser-déposer...) que techniques (utilisation de mootools, du plugin miftree...), ayant pris une part assez importante de notre temps de travail, la mise en oeuvre des solutions envisagées n'en a été que plus aisée et nous avons pu nous concentrer sur la maîtrise des langages et techniques nécessaires au développement de notre applications web ainsi que sur l'apprentissage de l'utilisation des outils et bibliothèques de développement choisis.

Ce document a été préparé à l'aide des éditeurs de texte gedit et geany et du logiciel de composition typographique LaTeX 2ε, le tout dans un environnement de travail libre.

Annexes

A

Sommaire

1 Apache, plume et perl

Pour tester ce projet de manière décentralisée, nous avons mis en place un serveur hébergé à la maison avec apache. L'installation d'apache ne pose pas forcement de problème. Nous avons installé Archlinux[3] sur notre serveur, il suffit alors d'utiliser le gestionnaire de paquet afin d'installer apache, perl et le module perl pour apache :

```
yaourt -S apache perl mod_perl
```

La partie la plus distrayante est la configuration de la bête ! Il faut pour cela éditer le fichier `/etc/httpd/conf/httpd.conf`. On commence par ajouter le module perl :

```
LoadModule perl_module  modules/mod_perl.so
```

On définit le dossier qui va contenir le site, et on configure le dossier parent avec le minimum de fonctionnalités

```
DocumentRoot "/srv/http/htdocs"

<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
```

Il faut ensuite configurer le dossier *htdocs*, avec l'option permettant l'exécution de script.

```
<Directory "/srv/http/htdocs">
    Options Indexes FollowSymLinks ExecCGI Includes
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>
```

Il faut également définir un alias pour le dossier cgi-bin, en précisant à apache qu'il doit utiliser perl pour exécuter les scripts qui seront stockés dans ce dossier si le module perl est activé.

```
ScriptAlias /cgi-bin/ "/srv/http/cgi-bin/"

<Directory "/srv/http/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>


```

```
<IfModule mod_perl.c>
        AddHandler perl-script .pl
        PerlHandler ModPerl::PerlRunPrefork
        PerlOptions +ParseHeaders
        PerlSendHeader On
</IfModule>
```

Nous avons donc maintenant un serveur fonctionnel, pouvant exécuter des scripts perl sans problème. Il faut pour cela stocker les scripts dans `/cgi-bin` et le site dans `/htdocs`.

2 Subversion

Pour développer un projet de cette envergure à plusieurs, nous avons décidé de mettre en place un serveur subversion. Subversion (en abrégé svn) est un système de gestion de versions, distribué sous licence Apache et BSD. Après avoir installé subversion via le gestionnaire de paquets, nous avons utilisé notre serveur apache fraîchement configuré pour diffuser subversion. Subversion est donc utilisé comme un module d'apache !

On commence par éditer le fichier de configuration d'apache, `/etc/httpd/conf/httpd.conf`, pour y ajouter les modules nécessaires :

```
LoadModule dav_module           modules/mod_dav.so
LoadModule dav_fs_module        modules/mod_dav_fs.so
LoadModule dav_svn_module       modules/mod_dav_svn.so
LoadModule authz_svn_module     modules/mod_authz_svn.so
```

Nous utiliserons un accès ssl pour se connecter au serveur subversion, il faut donc créer un certificat x509 afin d'utiliser apache en https :

```
openssl req -new -x509 -keyout server.key -out server.crt -days 365
    -nodes
```

Nous avons maintenant un certificat x509 valide pour un an ! On ajoute finalement une location au serveur apache pour pouvoir accéder au svn depuis `https://ip_serveur/svn/` :

```
<Location /svn>
   DAV svn
   SVNParentPath /home/svn/repositories
   AuthzSVNAccessFile /home/svn/.svn-policy-file
   AuthName "SVN Repositories"
   AuthType Basic
   AuthUserFile /home/svn/.svn-auth-file
   Satisfy Any
   Require valid-user
</Location>
```

La configuration indique l'emplacement du serveur svn, les fichiers de configuration, `.svn-policy-file` pour les autorisations d'accès au dépôts et `.svn-auth-file` pour stocker les comptes d'utilisateurs. On indique également que les dépôts requièrent un utilisateur valide.

Le fichier `.svn-policy-file` se configure de la sorte pour autoriser les utilisateurs **Johan** et **Julien** à accéder au dépôt **multimedia** en lecture et écriture.

```
1 [multimedia:/]
2 Johan = rw
3 Julien = rw
```

Il faut bien entendu peupler le fichier `.svn-auth-file` auparavant :

```
1 htpasswd -cs /home/svn/.svn-auth-file Julien
2 htpasswd -s /home/svn/.svn-auth-file Johan
```

L'option `-c` permet de créer le fichier, elle n'est plus utile après. Il ne reste plus qu'à créer le dépôt multimédia :

```
1 svnadmin create /home/svn/repositories/multimedia
```

Nous avons ensuite utilisé le plugin subversion d'eclipse pour faire nos commit et mettre notre travail à jour.

3 Notre application web

Notre site est accessible à l'adresse `http://barbuk.org`.

FIGURE A.1 – *Visuel de notre application web.*

Bibliographie

[1] Adobe, 2010. https://www.photoshop.com/. Photoshop Online. (Cité page 1.)

[2] Adobe, 2010. http://www.adobe.com/products/air/. Adobe Air. (Cité page 1.)

[3] archlinux, 2010. http://www.archlinux.org/. A simple, lightweight distribution : Archlinux. (Cité page 14.)

[4] audiotool, 2010. http://www.audiotool.com/. Laboratoire musical audiotool. (Cité page 1.)

[5] c.dolivet, 2010. http://www.cdolivet.com/index.php?page=editArea. Librairie javascript editarea. (Cité page 4.)

[6] creativecommons.org, 2010. http://creativecommons.org/licenses/by-nc-sa/3.0/. Licence Creative Commons. (Cité page 4.)

[7] Google, 2010. http://docs.google.com. Google documents. (Cité page 1.)

[8] http ://www.perl.org/, 2010. http://www.perl.org/. Perl. (Cité page 4.)

[9] mifjs, 2010. http://mifjs.net/tree/. Plugin Mootools Miftree. (Cité page 4.)

[10] mootools, 2010. http://mootools.net/. Librairie javascript compact Mootools. (Cité page 4.)

[11] Mozilla, 2010. www.getfirefox.com. Navigateur Firefox de la fondation Mozilla. (Cité page 4.)

[12] opensource.org, 2010. http://www.opensource.org/licenses/mit-license.php. Licence opensource MIT. (Cité page 4.)

[13] selfhtml.org, 2010. http://fr.selfhtml.org/. Laboratoire musical audiotool. (Cité page 3.)

[14] Chromium Team, 2010. http://www.chromium.org/. Navigateur open-source Chromium. (Cité page 4.)

[15] vadeker, 2010. http://www.vadeker.net/humanite/copyleft/lgpl.html. Traduction française de la licence LGPL. (Cité page 4.)

Index