LEHIANY Raphaël Master 1 IMAGINA
CIMBÉ Pierre-Alexandre Master 1 IMAGINA
LESNYAK Viktor Master 1 DECOL

Rapport de TER

Le WEB tel qu'il devrait être

Table des matières

Introduction

Si le World Wide Web est désormais connu et utilisé partout dans le monde, “cette idée simple … reste largement inexploitée” selon son concepteur Tim Berners-Lee lorsqu’il parle du Web Sémantique, mouvement collaboratif mené par ses collègues du W3C et lui-même. Si son appel nommé “Raw Data Now” (des données brut maintenant) n’a pas suffit à faire émerger le Web Sémantique, les technologies évoluent et la transition vers un Web des données liées s’effectue petit à petit.

La liaison et l’ouverture de cette masse de données brut nous amène à nous questionner sur la manière d’afficher ces ressources pour l’utilisateur de tous les jours. La séparation du fond et de la forme est une philosophie très présente dans le milieu informatique pour des raisons pratiques, et pourtant, la plupart des sites webs proposant des ressources en contrôle également la mise en forme.

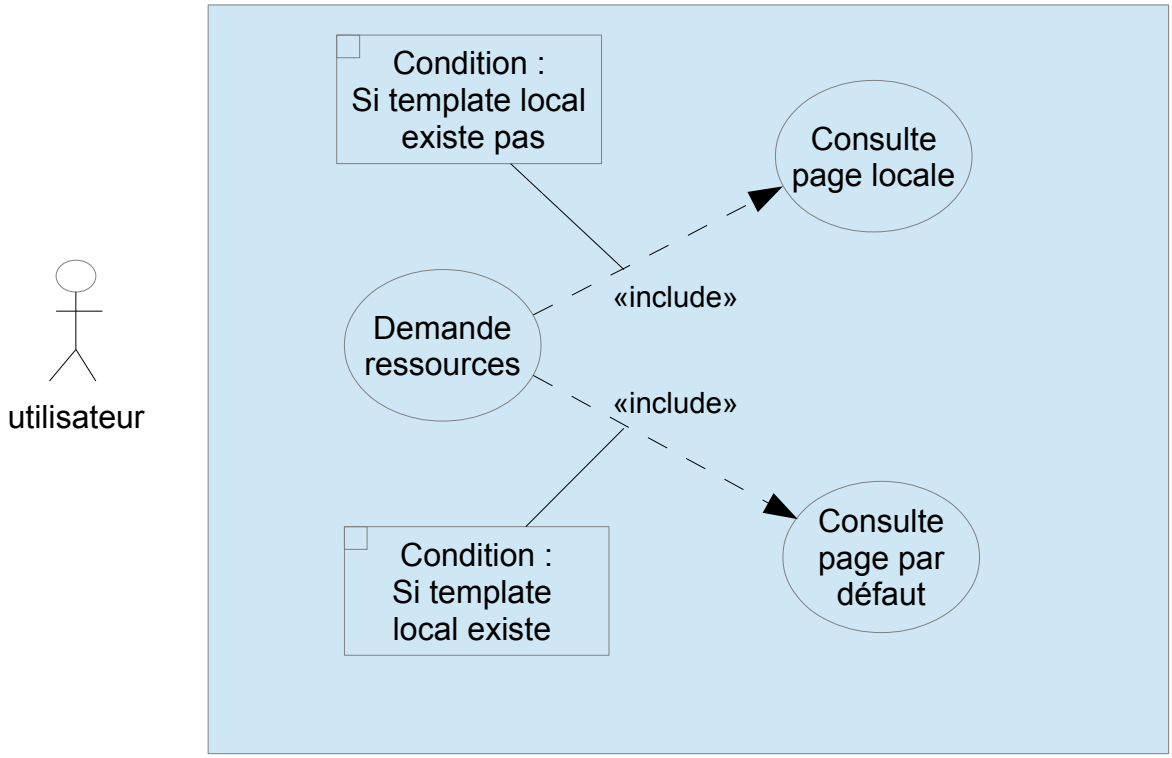
Nous souhaitons mettre en place un nouveau paradigme de navigation web, au moyen d’une application client-serveur qui permettra à l’utilisateur de modifier la mise en forme des données qu’il souhaite consulter, et étudier les limites d’un tel paradigme.

Nous feront premièrement le point sur les technologies existantes qui nous ont permis de répondre à cette problématique

En second lieu, nous étudierons l’état de l’art actuel, les solutions déjà apportées au domaine et les manques à pallier de ce dernier.

Ensuite, nous présenterons l’application comme elle a été conçu, ainsi que les choix que l’on a été amené à faire.

Enfin, avant de conclure, nous discuterons des perspectives d’avenir de cette application ainsi que des difficultés rencontrées au cours de sa réalisation.



Cas d'utilisation de l'application

Background Technique

RFC

Les RFC (Request For Comments) sont un ensemble de documents qui font référence auprès de la Communauté Internet et qui décrivent, spécifient, aident à l'implémentation, standardisent et débattent de la majorité des normes, standards, technologies et protocoles liés à Internet et aux réseaux en général. Peu de RFC sont des standards, mais tous les documents publiés par l'IETF sont des RFC.

HTTP

Nous allons rappeler certains points essentiels pour la compréhension du fonctionnement d'une application web.

Pour qu'une application web puisse communiquer avec plusieurs serveurs, un protocole doit être défini pour se comprendre. HTTP (HyperText Transfere Protocol) est un protocole de communication client-serveur.

Le fonctionement de base de ce protocole utilisé pour la comunication client-serveur est le suivant :

Le navigateur Web envoie au serveur des requêtes relatives à des pages Web. Le serveur répond aux demandes en envoyant les pages au navigateur Web. Le navigateur affiche alors les pages à l'utilisateur.

Il est nécessaire ici de définir ce qu'est un URL, un URI et un URN :

- Un URI(Uniform Resource Identifier), soit littéralement “identifiant uniforme de ressource”, est une chaîne de caractères identifiant une ressource sur un réseau (par exemple une ressource Web) physique ou abstraite, et dont la syntaxe respecte une norme d'Internet mise en place pour le World Wide Web, et aussi obligé de defenir la ressource de façon permanente, même si celle la a été suprimé ou deplacé. La norme était précédemment connue sous le terme UDI.

- Un URL(Uniforme Resource Locator) ne fait pas qu'identifier la ressource mais permet aussi d'effectuer des actions sur cette ressource, ou du moins il doit permettre d'accéder à la ressource. Par exemple http://www.google.com est un URL qui identifie une ressource, ici la page d'accueil du site Google, que l'on peut récupérer grâce au protocole HTTP depuis un réseau hôte www.google.com.

- Et un URN(Uniforme Resource Name) identifie une ressource. Par exemple un ISBN d'un livre peut etre vu comme un URI car il fait référence au livre, mais n'indique ni où ni comment on peut en récuperer un exemplaire.

Les URL et URN sont donc tout les deux des URI, comme définit dans la RFC.

Dans le protocole HTTP, une méthode est une commande spécifiant un type de requête, c'est-à-dire qu'elle demande au serveur d'effectuer une action. En général l'action concerne une ressource identifiée par l'URL qui suit le nom de la méthode.

Il existe plusieurs methodes :

- GET : C'est la méthode la plus courante pour demander une ressource. Une requête GET est sans effet sur la ressource.
- HEAD : Cette méthode ne demande que des informations (meta-données) sur la ressource, sans demander la ressource elle-même.
- POST : Cette méthode est utilisée pour transmettre des données au serveur en vue d'un traitement (le plus souvent depuis un formulaire HTML)
- OPTION : Cette méthode permet d'obtenir les options de communication d'une ressource ou du serveur en général.

- CONNECT : Cette méthode permet d'utiliser un proxy comme un tunnel de communication.
- TRACE : Cette méthode demande au serveur de retourner ce qu'il a reçu, dans le but de tester et effectuer un diagnostic sur la connexion.
- PUT :Cette méthode permet de remplacer ou d'ajouter une ressource sur le serveur. L'URI fourni est celui de la ressource en question.
- PATCH : Cette méthode permet, contrairement à PUT, de faire une modification partielle d'une ressource.
- DELETE : Cette méthode permet de supprimer une ressource du serveur.

Ces 3 dernières méthodes nécessitent généralement un accès privilégié.

Routage

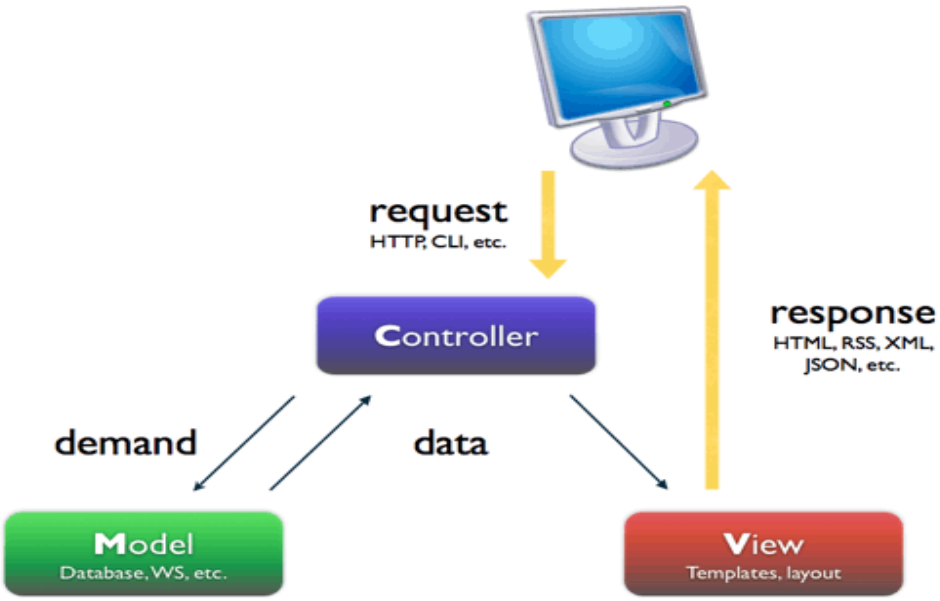
Une route est un lien vers une ressource physique ou non, définit par le serveur. A chaque route est associé un traitement que le serveur effectue en réponse à une requête HTTP, selon son type (GET, POST, HEAD, etc).

Django

Django est un framework de développement web en Python disponible en Open-Source.

Ce framework s'inspire de l'architecture MVC séparant l'architecture d'une application en 3 entités distinctes (Modele-Vue-Controleur) :

- le Modèle, qui gère la partie métier de l'application, a pour rôle d'effectuer des traitements sur les données de l'application (ajouts, mise à jour, suppression, ...).
  Il assure généralement la liaison avec la base de données ;
- la Vue, qui gère la partie graphique de l'application, est mise à jour lorsque le modèle change ;
- le Controleur , qui gère la partie évenementielle de l'application, assure le lien entre les interactions de l'utilisateur, la modification du modèle et la mise à jour de la vue qui en découlent.

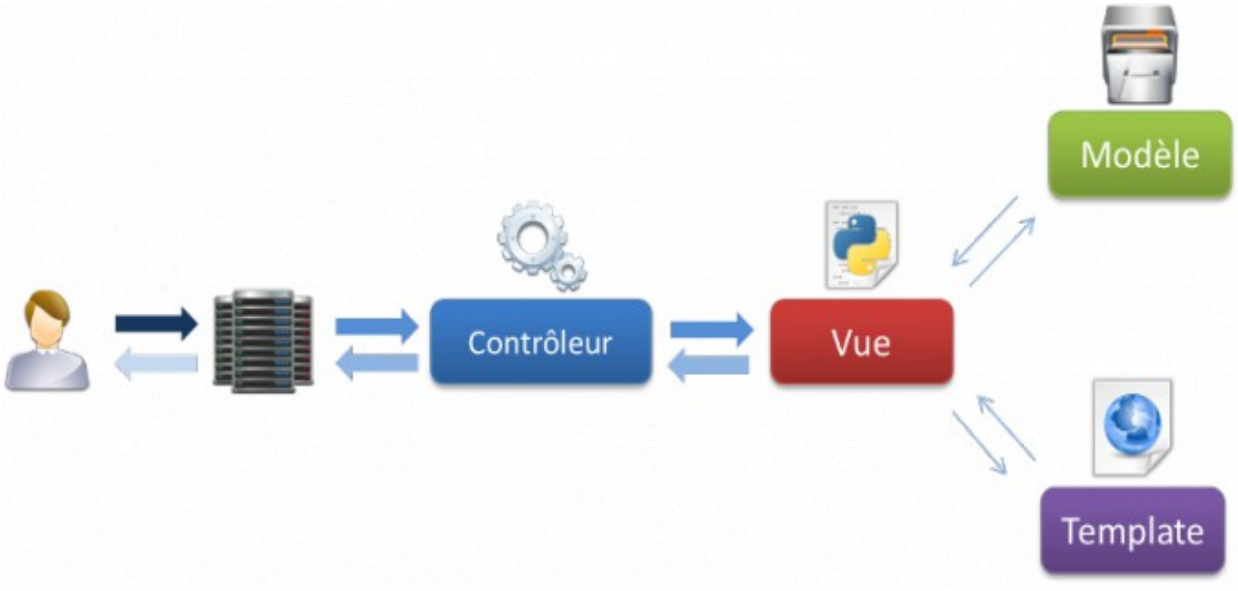


Fonctionement d'une architecture MVC

Django utilise une architecture similaire au MVC, nommée MVT(Modele-Vue-Template).
Dans l'architecture MVT, le contrôleur est supprimé car géré directement par le framework, tandis que vient s'ajouter l'entité Template.

Le Template est un fichier texte représentant la mise en page d'une ressource particulière. Il est généralement plus simple à utiliser que les langages natifs tels qu'HTML.
Dans le cas de Django, le template contient du code HTML et des blocs interprétables par le moteur de template.

Lors de l'accès à une ressource, le moteur de template parse cette ressource pour la mettre en page et génère le fichier HTML correspondant qui est retourné à l'expéditeur de la requête.
On peut ainsi générer une page au contenu dynamique à partir de la description de sa mise en page.



Fonctionement d'une architecture MVT

Lorsqu'il est sollicité pour afficher une page web, un serveur réalisé avec Django effectue les étapes suivantes :

- le serveur appelle la vue correspondant à la requête reçue selon sa table de routage ;
- elle récupère les données nécessaires ainsi que le template, les parse et génère le fichier HTML qui en résulte ;
- ce dernier est retourné en réponse à la requête initiale.

Django propose une division de chaque projets en modules, séparant ainsi l'implémentation des différents composants et facilitant leur réutilisation.

Play ! Framework

C'est un framework de développement web en Java qui n'utilise pas l'"API Servlet, le package standard de développement web pour J2EE, mais un autre framework HTTP asynchrone RESTful . Il respecte l'architecture MVC. Dans le cas de Play**,** les entités Modèle et Contrôleur s'implémentent en Java et l'entité Vue en Scala. Ainsi, ce framework permet de développer les serveurs web sur des JVM(Java Virtual Machine).

XUL

XUL se prononce "Zoul" et signifie "XML-based User interface Language". C'est un langage basé sur XML pour décrire une interface graphique, utilisé dans Firefox et les autres logiciels Mozilla.

Il possède ainsi toute une série de balises correspondant à des boutons, des menus, des arborescences (treeviews), zones d'éditions, la fenêtre principale, la barre d'adresse etc. Cela permet de définir le comportement du navigateur à notre guise, grâce à l'accès aux événements de ses composants. XUL décrit donc un contenu avec un comportement, puis une présentation pour ce contenu. Le XUL est donc un fichier XML interprété par un moteur de rendu : XULRunner, directement inclus dans Mozilla et Firefox. Le langage d'interaction utilisé derrière XUL est le Javascript.

La structure arborescente standard des fichiers d'une extension XUL est sous la forme suivante :

- <MonExtension>
  - chrome.manifest
  - install.rdf
  - content
    - browserOverlay.xul
    - broswerOverlay.js
  - skin
    - browserOverlay.css
  - local
    - en-US
      - browserOverlay.dtd
      - browserOverlay.properties

Le chrome est l'ensemble des éléments de l'interface utilisateur d'une application qui sont situés en dehors de la zone de contenu d'une fenêtre. Les barres d'outils, les barres de menus, les barres de progression, et les titres de fenêtres sont tous des exemples d'éléments qui font habituellement partie du chrome.
Le fichier *chrome.manifest* indique à Firefox où chercher les fichiers chrome.

Le fichier *install.rdf* est un fichier contenant les informations nécessaires au gestionnaire d'extensions pour installer le module complémentaire. Il contient des informations identifiant le module, sur l'auteur, l'endroit où trouver plus d'informations, la compatibilité avec les différentes versions des applications, comment il doit être mis à jour, etc.

Les fichiers XUL définissent généralement l'une des deux choses suivantes : les fenêtres ou les superpositions (overlay). Une superposition étend une fenêtre existante, en ajoutant de nouveaux éléments ou en remplaçant certains des éléments qu'elle contient.

Dans le fichier *browser.xul* on va définir les éléments qui composent l'interface venant se rajouter (superposition) à l'interface de base de Firefox *browser.xul*.

C'est dans le fichier *browserOverlay.js* que l'on va écrire tout le fonctionnement de l'extension.

On peut capturer tous les événements provenant du navigateur (interaction avec l'interface, réception d'une page HTML, etc) pour agir en conséquence.
Le fichier *browserOverlay.css* est une feuille de style.

Les fichier DTD et properties sont la façon la plus efficace d'afficher du texte, il s'utilise pour la génération de texte dynamique.

Les composantes les plus importantes :

- RDF (Resource Description Framework) :
  Permet de créer des fichiers XML qui reprennent la structure des composants qui les utilisent.
  On peut alors, dans XUL, créer une liste structurée à partir d'un fichier en décrivant un seul élément.

- XBL (eXtensible Binding Language) :
  Son but est de modifier les balises XML de XUL, ou les remplacer par d'autres. Donc de changer l'interface d'une application.
  Le langage définit les éléments XML pour les widgets, évènements, propriétés, méthodes.

- Overlay :
  Le but est de fractionner un GUI en plus petits composants réutilisables.
  C'est aussi le mécanisme d'ajout de composants à une application XUL.
  Overlay est un ensemble de fichiers XUL utilisé pour décrire le contenu de l'interface et l'adapter aux préférences de l'utilisateur.

- XPCom (Cross Platform Component Object Model) :
  A pour but l'utilisation de code natif par une application XUL, et améliorer les performances. La partie JavaScript de l'interface XUL peut ainsi être réécrite en C et compilée.
  Cela fournit un système de logiciel modulaire. Un composant XPCom peut être écrit en tout langage et utilisé par un programme écrit en tout langage.

- XPConnect :
  A pour but l'utilisation de librairies binaires avec XUL.
  C'est une interface permettant à JavaScript d'utiliser les fichiers XPCom. Les objets JavaScript peuvent utiliser les objets binaires définie à travers XPCom et inversement.

Ces technologies nous permettent de faire de nombreux type d'extensions très puissantes et très flexibles.

Les points forts de XUL :

- Langages conçus spécialement pour créer des interfaces utilisateurs portables. Une interface peut être implémentée et modifiée rapidement.
- A tous les avantages des autres langages XML. Par exemple, XHTML ou d'autres langages XML peuvent y être insérés.
- Les textes affichés avec XUL sont aisément localisables, ce qui signifie qu'ils peuvent être traduits dans d'autres langues.
- Il est entièrement libre et gratuit, de même que tous les outils qui l'accompagne.
- Respectueux des standards W3C, il produit des interfaces graphiques très complètes.
- Axé sur du Javascript, ses possibilités sont nombreuses.
- Gestion du format RDF pour gérer des gabarits par exemple et ce sans efforts.

Les points faibles :

- Le principal inconvénient du XUL est l'absence de message en cas d'erreur. L'interface ne s'affiche pas, sans plus.
- Les prérequis sont nombreux.
- L'emploi de RDF est assez complexe.
- L'emploi de XPCom, XPConnect l'est encore plus.
- En voie d'extinction pour laisser la place à HTML 5.

Javascript

Il est principalement utilisé sur le Web côté client :c'est le navigateur qui exécute le code. Le JavaScript est utile pour tout ce qui concerne les interactions du client sur la page Web. Il permet ainsi d'améliorer la présentation et l'interactivité des pages Web.

Mais il peut aussi être utilisé côté serveur. Il existe plusieurs déclinaisons du langage qui permettent de l'utiliser dans de nombreux domaines.

Javascript propose des fonctionnalités pour travailler facilement avec le DOM, ce qui facilite l'accès aux balises d'une extension XUL. En effet, il est le langage de script utilisé pour développer des extension XUL. Donc pour récupérer des événements sur ces objets, Javascript est incontournable. Coder différentes extensions en Javascript permet des les faire interagir entre elles, et donc d'aller au-delà de leurs champ d'application. C'est pour cela que la plupart des extensions Firefox utilisent Javascript comme langage de script.

Yacc et Lex

Lex est un générateur d'analyseur lexical codé en C, standard dans la plupart des systèmes Unix. Il a pour but de transformer une série de symboles en terminaux, en vérifiant si ceux-ci appartiennent bien au vocabulaire défini par le développeur. Lex n'effectuant qu'une analyse lexicale, il est généralement complété par une analyse syntaxique fournie par Yacc (Yet Another Compiler Compiler) afin de pouvoir interpréter/compiler un langage dans sa globalité.
La syntaxe d'un fichier lex est relativement simple :

```
Section Définition
%%
Section Règles
%%
Section Code
```

La section Définition contient des imports, des macros et du code C qui seront intégrés au début du code généré par Lex.
La section Règles contient des expressions régulières associées à des procédures. A chaque fois qu'une série de symbole lu par le lexer correspond à une des expressions régulières saisies dans cette section, la procédure qui lui est associée est appelée. C'est notamment dans cette procédure que l'on retourne le jeton correspondant à l'expression lue.

La section Code contient la fonction `main()` qui appelle l'analyseur et du code C intégrés dans le code généré par Lex.

Exemple :

```
%{
 int charcount=0,linecount=0;
%}
%%
.   {charcount++;}
\n   {linecount++; charcount++;}
%%
int main(){
 yylex();
 printf("Il y a %d caractères et %d lignes\n",
 charcount,linecount);
 return 0;
}
```

Cet analyseur lexical compte le nombre de caractères et lignes dans le fichier texte qui lui est passé en entrée. Chaque caractère lu par le lexer appelle la procédure associée à la première règle, chaque changement de ligne appelle la seconde.

Yacc est un générateur d'analyseur syntaxique codé en C, un standard dans beaucoup de systèmes Unix à l'instar de lex. Il génère un programme qui prend en entrée une donnée textuelle et la parse. Il est généralement couplé à Lex, car il est beaucoup plus simple de parser un ensemble de symboles terminaux, ce que fournit ce dernier.

A eux deux, Lex et Yacc permettent d'interpréter/compiler n'importe quel langage basé sur un automate à état fini, Lex s'assurant de la validité des lexèmes utilisés, et Yacc de la syntaxe.

La syntaxe d'un fichier yacc est similaire à celle d'un fichier lex :

```
Section Déclarations
%%
Section Règles de production
%%
Section Code
```

Dans la section Déclarations se rajoutent la déclaration des symboles terminaux et non terminaux utilisés.

La section Règles de production contient toutes les règles de production de la grammaire souhaitée et les procédures qui leurs sont assignées.

Exemple :

```
%token <Integer> INT
%start formule
%left PLUS
%%
formule : expr {printf("valeur=%d\n",$1);}
;
expr : expr PLUS expr {$$ = $1 + $3;}
| INT {$$=$1;}
;
%%
int yyerror(const char *msg) {
printf("ERREUR: %s\n", msg);
return 0;
}

int main(void) {
yyin = stdin;
yyparse();
}
```

Cet analyseur syntaxique lit et résout les additions d'entiers. Le token INT et l'opérateur PLUS sont récupérés d'une analyse lexicale par Lex. Le symbole non terminal "formule" est désigné comme racine de l'automate à état fini définissant la grammaire. Lorsqu'il est reconnu, la valeur de l'expression entière est affichée. Cette valeur est celle du symbole non terminal expr, lui même pouvant être décomposé en plusieurs additions successives. Dans la procédure assignée à chaque règle, $$ est la valeur associée à l'expression actuelle. Par récurrence, le langage résout donc l'addition et en affiche le résultat.
En cas d'erreur de syntaxe, la fonction yyerror est appelée.

GNU Bison et GNU Flex sont deux des programmes les plus utilisés implémentant respectivement yacc et lex.

Jison

Jison (http://zaach.github.io/jison/) est un générateur de parseurs open-source développé par Zachary Carter.

Il prend en entrée la description d'un langage au format JSON ou respectant une syntaxe similaire à celle de Lex et Yacc et génère un parseur en Javascript du langage défini.

Jison peut être utilisé pour générer un script js, mais il peut également être appelé depuis un script afin de générer localement une grammaire. De plus, il est possible de générer un parseur Jison depuis le site web de son développeur via un formulaire prenant en entrée la définition du langage souhaité.

Exemple :

```
%lex
%%

\s+                   /* skip whitespace */
[0-9]\b      return 'INT'
"+"                   return '+'
<<EOF>>               return 'EOF'
.                     return 'INVALID'
/lex

%left '+'

%start formule
%%
formule
    : e EOF
        {return $1;}
    ;
e
    : e '+' e
        {$$ = $1+$3;}
    | INT
        {$$ = Integer(yytext);}
    ;
```

Ce parseur résout les additions d'entiers. La syntaxe utilisée est proche de celle de Lex et Yacc mais beaucoup plus épurée.

Mustache

Mustache est un langage de template open-source qualifié de “logic-less”, c’est à dire qu’il permet uniquement certaines fonctionnalités basiques telles que les boucles et les conditions.

Pour effectuer des traitements plus complexes, ces derniers doivent être fait au préalable dans le langage utilisé implémentant mustache, puis les données doivent être envoyées au moteur de templating. On nomme cela le “data preprocessing”.

Le fonctionnement de mustache est simple : le moteur de templating de mustache prend en entrée une donnée textuelle respectant la syntaxe d’un template mustache ainsi qu’un ensemble de données structurées pouvant être représentées par une classe, contenant ainsi les fonctions nécessaires au data preprocessing, et retourne un document HTML.

Mustache utilise une syntaxe relativement intuitive : une variable {{var}} entourée par des doubles accolades (qui ont donné son nom à Mustache) est remplacée par sa valeur dans les données que l’on cherche à mettre en forme. L’utilisation d’un # devant le nom de la variable {{#var}} permet de créer une section. Si var est un booléen, alors cette section est une condition sur la valeur de ce booléen. Sinon, c’est une boucle “foreach”.

Prenons l’exemple du fichier au format JSON suivant :

```
"livres" : [
       {
           "titre" : "Le Web Sémantique",
           "auteurs" : [
          {“nom” : "Fabien Gandon"},
          {“nom” : “Catherine Faron-Zucker”},
          {“nom” : “Olivier Corby”}
     ]
       },
       {
           "titre" : "Weaving the Web",
           "auteurs" : [
          {“nom” : "Tim Berners-Lee"},
          {“nom” : “Mark Fishetti”}
     ]}]
```

Pour mettre en forme nos données, nous utilisons le template Mustache suivant :

```
 {{#livres}}
         <h2>{{titre}}</h2>
         <h3>Auteur(s):</h3>
         <ul>
         {{#auteurs}}
             <li>{{nom}}</li>
         {{/auteurs}}
         </ul>
{{/livres}}
```

Le moteur de templating de Mustache prend en entrée la donnée et le template, et rend un document HTML qui affichera un résultat similaire à ce dernier :

Le Web Sémantique
Auteur(s)

- Fabien Gandon
- Catherine Faron-Zucker
- Olivier Corby

Weaving the Web
Auteur(s)

- Tim Berners-Lee
- Mark Fishetti

État de l'art

Le Web des données est une initiative du W3C dont le but est de lier les données circulant sur la toile en un seul et unique grand réseau d'informations. Le Web des données est une démarche importante pour l'émergence du Web sémantique.

La ressource principale du Web des données est le format de fichier RDF qui regroupe un ensemble de triplets (sujet, predicat, objet) liant sémantiquement le sujet et l'objet selon le prédicat. Couplé à l'utilisation des URI comme identifiant unique de toute donnée, les bases de triplets RDF créent des liens entre les données habituellement isolées par paquet dans les serveurs. Les triplets de ces bases peuvent être exploitées afin d'en déduire de nouveaux liens sans l'intervention de l'homme. L'initiative a été suivie par de nombreux développeurs et de nombreuses sources de données en ont émergé :

- des sources de données générales, comme DBpedia qui produit depuis 2007 un grand nombre de triplets extraits des données de Wikipedia ;

- des sources de données gouvernementales, comme data.gouv.fr qui facilite l'accès et l'utilisation des informations publiques en France;
- des sources de données géographiques, comme GeoNames qui fournit près de 8 millions de données géographiques dans le monde au travers de plus de 143 millions de triplets RDF;
- et autres vocabulaires liés à des domaines précis (social, cryptographie, science du vivant, etc).

Face à une telle masse de données brut, on est amené à se demander de quelle manière les utilisateurs quotidiens pourraient les exploiter, étant donné la tâche difficile qu'est la lecture de données non structurées.

Une faiblesse actuelle de la navigation web concerne la séparation du fond et de la forme des ressources partagées. Si l'informatique utilise beaucoup ce concept en raison de son intérêt pratique, les serveurs possèdent malgré tout le monopole des ressources et de leur mise en forme, les navigateurs sont réduits à afficher des pages web imposées par leur fournisseur et l'utilisateur ne contrôle pas la mise en page de la ressource à laquelle il souhaite accéder.

La séparation du contenu d'une page web et de sa feuille de style via le couple de langage HTML / CSS est un premier pas dans la séparation fond/forme de la ressource. Elle permet notamment de changer l'apparence d'une page en changeant simplement de feuille de style. Pourtant cette séparation n'est pas complète puisque HTML garde le contrôle du découpage du document.

L'ajout des langages PHP et javascript permettant le développement de sites dynamiques a creusé l'écart entre la forme et le fond en permettant le changement dynamique du contenu d'une page unique.

Le templating découle de ces technologies, permettant de décrire une mise en page de document de façon intuitive utilisée ensuite pour afficher les données brutes. Cette pratique a également donné naissance au templating client, consistant à léguer au client Javascript la mise en forme de la donnée, afin d'alléger les transferts de données. Malgré cela, le choix de la mise en page des données restent au serveur.

Certaines initiatives ont été tentées concernant la séparation des données existantes et de leur forme. Weboob est une application qui regroupe plusieurs modules capables d'interagir avec différents sites web sans ouvrir de navigateur. L'utilisateur peut choisir ses modules afin d'obtenir uniquement certaines informations allant de vidéos aux horaires des trains.

La démarche est intéressante, car la donnée est filtrée de sa mise en page habituelle. Pourtant, Weboob est conditionné par les modules développés pour chaque site et n'accorde pas une liberté totale sur la mise en forme des ressources.

En conclusion, de nombreuses technologies existent autour de la séparation du fond et de la forme, mais la possibilité pour le client de contrôler la mise en page de la ressource qu'il récupère n'est toujours pas traitée. On s'inspirera des différentes technologies précédemment citées pour apporter une réponse à ce problème et étudier ses limites.

L'application Myweb

L'étude de cette problématique nous a amené à développer un plugin Firefox pour changer le comportement habituel du navigateur. L'application permet à l'utilisateur de demander à un serveur de la donnée brut. Pour mettre en forme cette donnée, on a besoin d'utiliser un template. Un template permet de définir quel partie de la ressource on veut, et de quel manière l'afficher. Ce fichier nous permet de générer un fichier HTML contenant de la donnée JSON. Chaque ressource a besoin d'un template pour être interprétée. Le serveur qui propose la donnée pourra fournir un template par défaut qui lui est associé pour simuler le fonctionnement actuel de la navigation web. Mais, de manière général, c'est le template local qui sera utilisé pour interpréter la donnée.

Quand l'utilisateur demande une ressource, le serveur envoi uniquement la donnée brut (en JSON). Dans le cas où le navigateur ne possède pas de template associé à la ressource demandé, il le demande au serveur. Sinon il utilise le template stocké en local, provenant soit d'un template par défaut téléchargé au préalable, soit d'un template édité par l'utilisateur. Le plugin se charge d'appeler le parseur pour générer un fichier HTML depuis le fichier JSON ainsi que son template. Une fois le fichier HTML produit, le plugin demande au navigateur d'afficher le résultat. L'utilisation de l'application doit être transparente vis à vis de la navigation web classique.

Nous avons donc divisé la réalisation de cet application en trois parties :

- Un serveur web
- Un langage de template
- Une extension firefox

Le serveur web dois être capable de fournir de la donnée brut de type JSON,

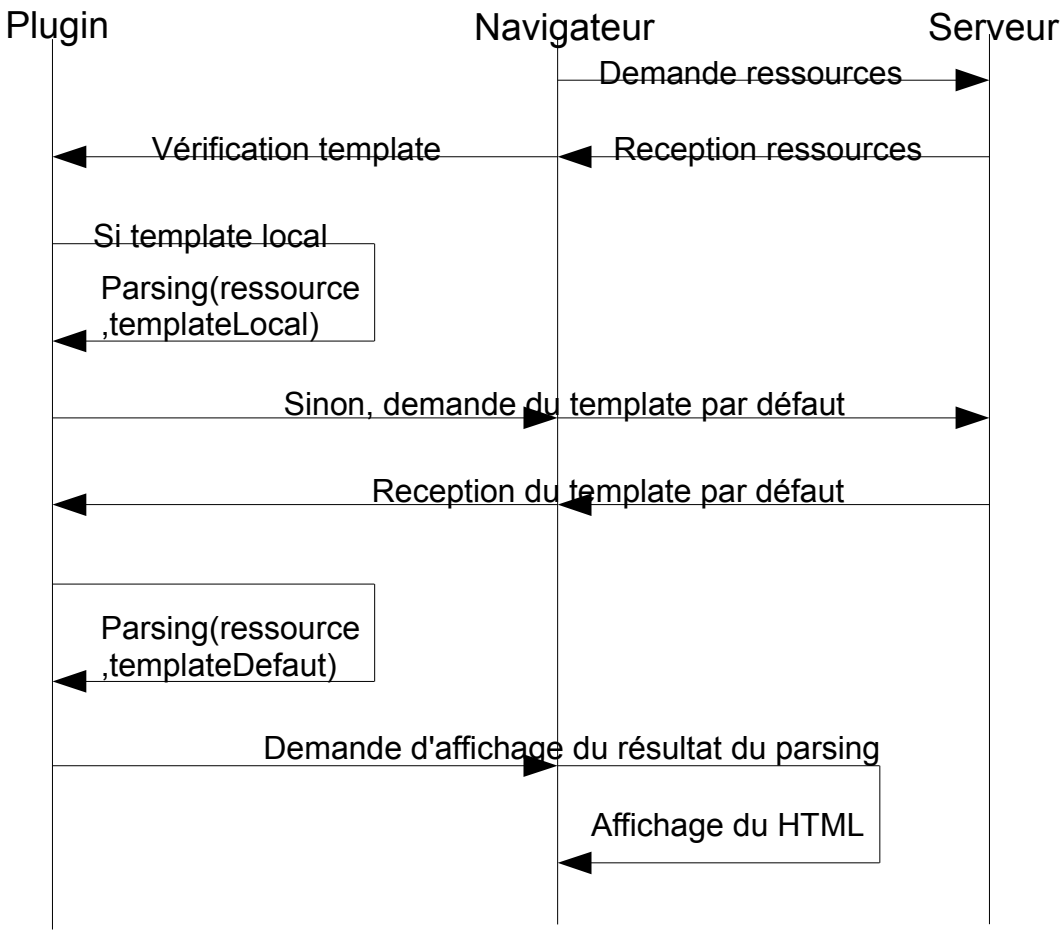
ainsi qu'un template associé à celle-ci.

La partie template doit fournir un parseur prenant en entrée la donnée ainsi que son template et générer un document HTML.

L'extension doit être capable de :

- Stocker des templates en local ;
- Vérifier leurs existence ;
- Se procurer un template par défaut le cas échéant ;
- Appeler le parseur pour générer le fichier HTML finale.

Le diagramme suivant montre les interactions entre le serveur et le navigateur, ainsi que le rôle du plugin dans l'application.



Serveur

Pour le développement du serveur nous avions le choix entre utiliser un framework web comme ceux précédemment cité, ou d'utiliser des bibliothèques dédié au développement de serveur web d'un langage. En se penchant sur l'idée d'un serveur web écrit en python, nous nous somme rendu compte de la facilité à mettre en place un serveur minimaliste, ainsi que de son efficacité :

```
$ python -m SimpleHTTPServer
```

Il suffit d'une ligne, et le serveur est capable de répond à une requête de type GET en envoyant la ressource du fichier stocké à sa racine.

Cependant nous devons mettre un place un système de routage pour notre serveur. C'est la bibliothèque `BaseHTTPServer` qui nous donne accès à la classe `MyHandler`. Cette classe nous permet de définir les comportements des méthodes des requêtes HTTP reçues par sur le serveur. On a donc les outils nécessaire pour développer notre serveur web en python, et ceci de manière comparable à des framework, en terme de temps de prise en main, et temps de réalisation. Développer le serveur en python nous permet d'étudier le fonctionnement d'un serveur web d'assez bas niveau, ainsi que de pouvoir définir un comportement assez précis. Nous avons donc opté pour la seconde solution.

La Methode permettant de créer l'instance d'un serveur est la suivante :

```
server = HTTPServer((' ', 80), MyHandler)
```

Ensuite grâce à la méthode `serve_forever()`, on ouvre une socket qui attend des requêtes, sur le port 80. Le méthode `server.socket.close()` ferme la socket.

Nous devons maintenant définir le corps de la méthode do_GET() de la classe `MyHandler`. Pour définir des routes, nous avons utilisé la méthode endswith(string) sur l'URL :

```
if self.path.endswith(".json"):
```

Cette définition nous permet d'utiliser le même code pour n'importe quelle ressource demandée à condition que l'URL finisse par « .json ». Il faudra quand même récupérer le nom de la ressource pour envoyer le bon fichier, cela se fait grâce à quelques `split()` sur l'URL.

Pour pouvoir retrouver le fichier demandé par l'utilisateur, on a ordonné les

fichier en mettant toutes les données à la racine, et un dossier « template ». Ce dossier contient les templates qui correspondent au ressources. La correspondance se reconnaît car les deux fichiers ont le même nom, excepté leur extension. Un « .json » l'autre « .template ».

Si il s'agit d'une requête pour un fichier de type JSON, on envoi le contenu du fichier :

```
self.send_response(200)  //tout s'est bien passé

self.send_header('Content-type','text/html') //message à
interprété par le navigateur tel un HTML

self.end_headers()

self.wfile.write(fichier) //on demande d'afficher le contenu
du fichier
```

Si L'URL contient le paramètre « &template=false » c'est que le client demande un template et non une ressource. Il faut donc la repérer au préalable. Au lieu d'utiliser la méthode `endswith(string)` sur l'URL comme dans le cas précédant, on va utiliser la méthode `find(string)` :

```
if self.path.find('&template=false'):
```

Et on envoi ensuite le contenu du fichier template de la même façon que le JSON.

Template

L'objectif principal de l'application est de permettre à l'utilisateur de mettre en page les données qu'il reçoit du serveur de la manière qu'il souhaite. Dans ce but, nous souhaitons mettre en place un moyen simple de modéliser la mise en page souhaitée, l'utilisateur pouvant alors exprimer intuitivement ce qu'il souhaite sans formation avancée à l'outil.

En plus d'avoir l'avantage de séparer la forme et le fond, le templating rempli ce critère. Il est généralement beaucoup plus simple à utiliser que les langages natifs comme HTML pour un utilisateur occasionnel.

Souhaitant définir notre propre langage de template, il a d'abord fallu choisir entre l'utilisation d'un langage balisé, basé sur XML, et celle d'un langage textuel, comme le template de Django.

Le langage balisé possède l'inconvénient majeur d'être très difficile à lire à l'oeil nu, raison pour laquelle les développeurs de Django ont choisi un langage de template textuel. Nous avons donc opté pour la même solution.

Django utilise un interpréteur maison pour parser les fichiers templates. Le développement d'analyseurs syntaxiques et lexicaux s'éloignant de notre problématique, nous avons souhaité nous passer de cette tâche en utilisant un générateur d'analyseurs à la manière des programmes Lex et Yacc.

Pour cela, nous avons découvert le programme open-source Jison, reprenant la syntaxe des deux précédents pour générer les analyseurs souhaités en JavaScript, langage que l'on utilise dans le plug-in.

Cependant, devant la difficulté à créer un langage efficace et fonctionnel dans les temps impartis, nous avons jugé que la définition du langage n'était pas prioritaire vis-à-vis de la problématique étudiée. Nous avons donc finalement choisi d'utiliser Mustache, un langage de template open-source, afin d'obtenir un résultat suffisant pour que l'application finale fonctionne.

Client

Nous avions le choix entre développer notre propre navigateur, ou d'ajouter un plugin à un navigateur déjà existant pour modifier son comportement. Développer notre propre navigateur nous aurait contraint à coder des fonctionnalités déjà fournies par les navigateurs existant, nous avons donc rapidement écarté cette solution, celle-ci ne présentant pas d'intérêt vis à vis de notre objectif. Ainsi nous avons étudié le langage le plus utilisé (XUL) pour la conception d'extension de navigateurs, et bien compris le rôle et le fonctionnement de ces derniers.

Nous avons choisi XUL comme langage d'extension car il offre une souplesse et une puissance au système d'extension que n'a pas encore atteint HTML 5. En effet, certaines parties de la technologie XUL sont absentes en HTML 5, comme les overlays.

La première tâche du plugin est de détecter une URL d'un serveur Myweb. On a donc convenu d'une pattern particulière d'adresse URL pour être reconnu Myweb : Elle doit contenir « /myweb/ » et doit finir soit par « .json » soit par « .template».

On se sert de l'évenèment onPageLoad de l'interface nsIWebProgressListener pour détecter le chargement d'une nouvelle page :

```
    gBrowser.addEventListener("DOMContentLoaded",
this.onPageLoad, false);
```

La variable globale gBrowser donne l'accès au navigateur. Le navigateur est un élément `tabbrowser` situé dans le `browser.xul`.

Quand le navigateur reçoit une ressource d'un serveur Myweb, l'application parcours le dossier ayant pour nom l'URL de la ressource et cherche le template dont la forme est la suivante : nomURI + ".template". Où « nomURI » est le nom de la ressource privé de son extension. Ranger les templates de cette façon palis le cas où deux templates des ressources différentes ai le même nom.

```
    let template = getLocalDirectory();
    template.append("/" + URL + "/" + nomURI + ".template");
    if (template.exists() == true)
```

Si un tel fichier est trouvé, le plugin appelle le parseur pour générer une page HTML à partir de la donnée JSON et du template local.

```
    var result = Mustache.render(template, data);
```

Le code du parseur de Mustache a été ajouté dans les ressources du module.

Si il n'y a pas de template en local ayant le même nom que la ressource, on redirige l'URL pour demander le template au serveur :

```
    window._content.document.location = url + "&template=false";
```

On récupère le template dans une variable :

```
    var templatedef = document.body.innerHTML;
```

Et on peut appeler le parser sur le template par défaut :

```
    var result = Mustache.render(templatedef, data);
```

Il ne reste plus qu'à afficher le résultat dans window :

```
    window.document.body.innerHTML = result;
```

Discussion

Pouvoir éditer son propre template depuis le navigateur faciliterait la personnalisation de template. En effet, un des points les plus important de notre sujet étant de stocker l'affichage coté client, il est nécessaire d'en faciliter son édition. Le fichier serait automatiquement sauvegardé dans le dossier Myweb.

Cette amélioration nous mène vers une autre amélioration qui serait de pouvoir stocker plusieurs templates locaux associés à une même ressource, et d'avoir le choix quant à laquelle utiliser.

Une des nécessité majeure est d'implémenter ce découpage data/template pour une multitude de types de donnée brut comme par exemple : XML, RDF, SQL, etc...

On pourrait mettre en place un filtre de donnée coté serveur, pour permettre au client de demander au serveur uniquement la partie de la donnée qu'il désire. Cela réduirait et optimiserait le flux de donnés transférées entre le client et le serveur.

Le plugin pourrait s'étendre à une importante interface de gestion de contenu pour pouvoir organiser l'affichage de la donnée dans des zones défini de la fenêtre principale du navigateur. Pouvoir faire une compilation de plusieurs donnée de serveurs Myweb différent, dans une même page. Par exemple, une zone dédiée pour afficher les nouvelles, une autre pour afficher les photos de Google image correspondant au document principale que l'utilisateur consulte, etc, ceci en introduisant le web sémantique, pour par exemple pouvoir établir un lien entre les sujets traités du document que l'utilisateur consulte, avec des videos Youtube (métadonnées).

Conclusion

Nous avons montré grâce à cette application une autre façon de naviguer sur le web.Une séparation complète du fond (donnée) et de la forme (template) donne la possibilité à l'utilisateur de choisir exclusivement l'information qu'il veut. Cette approche permet aussi de créer des pages internets simples, mais surtout de personnaliser son contenu et son affichage coté client.

Une des limites du paradigme étudié est la puissance du langage de template, vis à vis de ce qui existe déjà en matière de mise en forme des données. En effet, il peut être difficile d'obtenir le même résultat que les sites web actuel avec un langage restreint.

Le fait qu'il n'existe pas de normalisation universelle des données rend difficile la réalisation de templates réutilisables d'une ressource à l'autre. Un utilisateur souhaitant personnaliser l'affichage de plusieurs ressources différentes devra le faire au cas par cas, même si les ressources ont une sémantique similaire.

Cette séparation fond/forme de la ressource apporte une alternative à la navigation web actuelle. Bien qu'elle soit limitée par certains standards du web actuel, les avancés dans le domaine des donnés liées apporteront certainement des solutions qui permettront d'améliorer cette approche de la navigation et potentiellement d'en faire un standard du web 3.0.