Méta-connaissances pour générer des programmes sur le Web

Basile STARYNKEVITCH

http://starynkevitch.net/Basile/

basile@starynkevitch.net

(à titre **privé**, sans rapport avec mon employeur CEA-LIST)

Séminaire à la mémoire de Jean-Louis Laurière - 22 mars 2006

1. introduction

Ce que cet exposé tente d'être :

- une problématique de la construction pratique de sites Web
- une petite taxonomie des connaissances utiles à la création de sites Web dynamiques
- nécessité d' une approche par "intelligence artificielle"
- pourquoi des méta-connaissances sont nécessaires

Ce que **cet exposé n'est pas** :

- un panorama complet (des technologies Web)
- un exposé sur le Web Sémantique

  (ontologie, meta-données descriptives, ...)

- la description d'un système qui tourne [faute de temps....]

2. problématique

Un site Web dynamique, c'est :

- **plusieurs utilisateurs** simultanés : gérer des données propres à chaque utilisateur (sessions), et des données communes
- souvent une **base de données relationnelle** (MySQL, PostGreSQL)
- un graphe (plus ou moins arborescent) de pages
- des navigations en parallèle arbitraires : l'état d'une page est une **continuation** (C.Queinnec)
- des **formalismes variés et peu compatibles** (XHTML, CSS, Javascript, SVG, images, ...)
- des règles **stylistiques** ou **ergonomiques**
- **le poids historique des standards** (complexité des détails)

Rappels succints sur HTTP

(Hyper Text Transfer Protocol) protocole essentiellement textuel initié par le navigateur. Chaque **requête** du navigateur attend une **réponse** du serveur Web.

```
GET /chemin/acc%E8s?q=req  HTTP/1.1
Host: www.exemple.com
User-Agent: HTTPTool/1.0
                              ⟶
```

requête ⟶

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
<h1>Happy New Millennium!</h1>
```

réponse

Les requêtes sont principalement
- `GET` pour obtenir une resource idempotente (peu modifiable)
- `HEAD` pour tester l'existence d'une URL
- `POST` pour soumettre une information dans un formulaire.

En pratique, une même page nécessite des dizaines de requêtes, souvent `GET` (pour les images, les scripts, les styles, ...).

Les requêtes sont **complexes** dans leurs détails (en-têtes, notamment cookies)

Rappels succints sur URL, XHTML, CSS, Javascript, ...

exemple d'URL adresse réticulaire :

`http://Jojo:lApIn@www.exemple.com:8888/chemin/acc%E8s?q=req#signet`

XHTML : langage `www.w3.org` à balise XML, représentation d'un quasi-arbre ; nœuds décorés d'attributs, dont la valeur est une chaîne.

Formes dans XHTML (et XForms) : décrivent un formulaire ; les champs saisis sont envoyés (par POST) ; certains champs sont cachés.

CSS cascading style sheet : resource décrivant la présentation

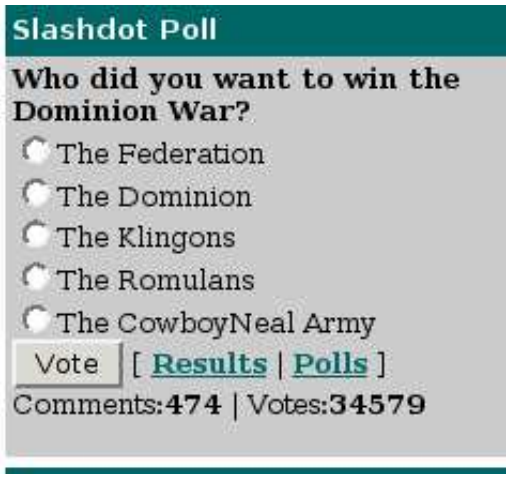
Javascript : langage interprété (à prototypes). Le navigateur interprète le Javascript d'une page, qui peut être modifiée par du code Javascript ! DOM Document Object Model, AJAX

PHP : langage interprété (coté serveur) générant du HTML (ou autre contenu)

CGI : Common Gateway Interface - appeler un programme (coté serveur) à chaque requête.

Exemple réel `http://slashdot.org/`

```
<div class="content" id="poll-content">
 <form action="//slashdot.org/pollBooth.pl" method="post">
  <fieldset>
  <legend>Poll</legend>
  <input type="hidden" name="qid" value="1331">
  <input type="hidden" name="section" value="mainpage">
  <b>Who did you want to win the Dominion War?</b>
  <br><input type="radio" name="aid" value="1">The Federation
  <br><input type="radio" name="aid" value="2">The Dominion
  <br><input type="submit" value="Vote">
  [ <a href="//slashdot.org/pollBooth.pl?qid=1331&amp;aid=-1"> <b>Results</b></a> |
  <a href="//slashdot.org/pollBooth.pl"><b>Polls</b></a> ] <br>
  Comments:<b>474</b> | Votes:<b>34579</b>
  </fieldset>
 </form>
</div>
```



problèmes de sécurité et de performance

cf *Securing web application code by static analysis and runtime protection*

(Y-W.Huang, F.Yu, C.Hang, et al.) *WWW 2004*

- "cross site scripting" injection de valeur avec URL menaçante
- SQL injection valeur menaçante provoquant une requête SQL néfaste
- injection de script nom de fichier `foo ; rm -rf /` inséré dans commande `cp`
- violation d'intégrité
- débordement de tampon
- "phishing" (faire semblant d'être un site bancaire)

Mais aussi,

- coût (en code et en temps) de la **persistence des données** d'une requête à l'autre
- nombreux décodages successifs (SGBD, serveur, ...)
- coût de l'interprétation

3. connaissances utiles pour faire un site Web

le poids de l'histoire

Le Web d'aujourd'hui s'est façonné par *strates successives*.

La compatibilité avec l'existant est économiquement obligatoire. le Web serait très différent s'il était inventé aujourd'hui

Complexité des standards XHTML, Javascript, AJAX, ... et **spécificité** des navigateurs connaissances des bogues d'IE !

Validité du XHTML. Peu d'outils garantissent la validité du XHTML généré. typage statique : *Ocamlduce* (A.Frisch), *Xtatic* (B.Pierce)

Meta-validation des modifications XHTML par du Javascript aucun outil ne valide les transformartions du Javascript...

Séparation Présentation / Traitement (templates PHP, Wdialog)

règles d'ergonomie

explicitées (p.ex : *Ergonomie du Logiciel et Design Web* J-F.Nogier) informellement

1. informer dès la page d'accueil
2. minimiser la profondeur du site (3-4 niveaux maximum)
3. placer le chemin de progression en haut de page
4. fournir les repères de navigation depuis la page d'accueil
5. minimiser le poids des images
6. dans un formulaire proposer par défaut la valeur la plus courante
7. plus l'élément est important, plus il doit se voir
8. etc....

Formalisation dans un système à base de connaissances : **Conseils** (*Malice*, J.Pitrat) plutôt que règles

actuellement - pas de vérification / validation de l'ergonomie (mais bibliothèques PHP cohérentes...)

représentations multiples des données et des connaissances

Plusieurs représentations similaires pour la **même** information :

- representation tabulaire dans le SGBD
- représentation graphe (structure de données internes) dans le programme serveur (PHP ou CGI)
- représentation textuelle(XHTML, Javascript, CSS) pour le navigateur

Importance de la **persistence des données**

- une navigation = plusieurs pages = plein de requêtes, donc
- va et vient SGBD $\Leftrightarrow$ programme serveur $\Leftrightarrow$ navigateur client
- persistence en principe systématique, pas si simple que ça en pratique
- adaptation d'impendance entre les représentations
- persister les continuations
- données pérennes $\neq$ données sessions (plusieurs degrés de oersistence)
- gérer/prévoir les mises à jour et évolutions du système

Voir *Global abstraction-safe marshalling with hash types* (James J. Leifer, Gilles Peskine, et al ICFP 2003)

4. système à méta-connaissances pour le Web ?

NB : un vœu pieux actuellement - pas de réalisations

taxonomie des applications (e-commerce ≠ forum ≠ site d'aide) ?

méta-programmation indispensable : il faut générer des codes variés dans des langages *différents* :
– schéma SQL base de données parfois grosse, souvent déportée
– traitement de données et persistence (SGBD ↔ programme)
– scripts Javascript (validation de la saisie dans le navigateur)
– générateur XHTML
– meta-générateur de Javascript modifiant le document XHTML
méta-programmation **adaptative** : adapter dynamiquement le système (en re-générant son code) aux visites effectuées sur le site (apprentissage).

pas de language de programmation unique à tout faire !

[méta-] représentation des données

il faudrait définir un formalisme général décrivant les données et fournir les connaissances pour **générer** :

1. la bonne représentation SQL et en changer
2. les représentations données en mémoire
3. la sérialisation, le transfert SQL
4. les servitudes internes (ramasse-miettes dans un CGI - traitement court).
5. les changements de représentations
6. les générateurs de XHTML, Javascript

Ou même décrire les relations (contraintes) entre les représentations....

Formaliser les conseils d'ergonomie et d'organisation des sites

1. définir des formalismes pour exprimer les règles d'ergonomie
2. faire une taxonomie des connaissances ergonomiques
3. gérer les conflits ou priorités entre règles
4. aider à l'organisation d'un site tout entier
5. gérer la cohérence des styles CSS et de leurs utilisations

Ces connaissances sont *vagues* (de haut ou très haut niveau) et nécessitent des méta-connaissances pour être utilisées ou transcrites

Exprimer déclarativement le traitement applicatif

Pour certaines applications Web, l'essentiel du traitement est relatif aux représentations :

1. commerce en ligne
2. forums
3. etc...

D'autres applications requièrent un traitement spécifique difficile ou l'interface à un logiciel existant

1. moteur de recherches
2. gestion - comptabilité
3. base de données techniques, géographiques, ...
4. etc...

Le Web comme outil collaboratif

Ré-utiliser les idées de J-L.Laurière pour le Web

manipuler formellement les contraintes, relations, descriptions exprimant les formalismes et les correspondances entre eux.

gérer et mettre à jour ensemble les différentes meta-représentations

générer du code spécifique correspondant à un sous-problème spécialisé.

s'adapter à la **complexité du contrôle** (navigation → continuations en parallèle)

maîtriser la complexité du Web par les approches méta- de J.Pitrat et J-L.Laurière.

réflexivité - un tel système devrait être une application Web collaborative

importance du logiciel libre

la suite LAMP(Linux, Apache, MySQL, PHP) est le succès du logiciel libre (majorité des sites web).

Pour être véritablement utilisé, un système à méta-connaissances pour le Web **doit être un logiciel libre** (comme l'est LAMP)

Populariser l'approche méta dans de[s] logiciel[s] libre[s]

5. Questions ?

Merci !

`$Id: metacoweb.tex 105 2006-03-21 20:35:00Z basile $`