Projet de semestre

DocReuse:

template based document editor

Jonathan Wafellman

SIN Master Student

06/06/2008

Table des matières

Jonathan Wafellman
SIN, Master Student

Introduction

DocReuse

Docreuse est un système orienté internet qui a pour but de structurer des données.

De nombreuses informations sont présentes dans des types de documents différents, stoqués dans différents endroits et détenues par différentes personnes. Ces informations sont présentes mais difficilement utilisables. En donnant une structure à l'information, nous la rendons plus accessibles et facilitons l'ajout ou la modification.

DocReuse simplifie cette tâche, assisté par de nombreuses techniques, l'utilisateur ne devra plus qu'insérer ses données aux endroits prévus à cet effet. Elles seront dés lors disponibles à tous et aisément utilisables.

De façon semi-automatique, l'utilisateur peut ordonner les connaissances. Dés lors il sera possible de réutiliser ces informations. L'utilisation de DocReuse se fait via un navigateur. Il est donc très facile d'utilisation pour le client et permet des modifications rapides du système. De plus tout changement dans le programme reste invisible à l'utilisateur.

Technologie

DocReuse utilise la technologie XML pour stocker les données structurées. Un nouveau langage XML, appelé XTIGER, a été développé dans l'optique de créer des patrons de documents. Grâce à ce système, un document peut être défini structurellement, il n'est alors de la responsabilité de l'utilisateur que de remplir ce document avec ses données.

Plus qu'un simple formulaire avec des champs à remplir, les patrons XTIGER permettent la représentation directe et l'insertion d'éléments complexes laissant une grande liberté à l'utilisateur. Il permet d'avoir une structure et une représentation de celle-ci.

Le projet

Le projet concerne la partie cliente de l'application et plus précisément l'interface. Avant le commencement du projet, une interface pour la création d'instance XTIGER était déjà implémentée. Cette interface simple était produite côté serveur et ne représentait que les champs remplissables, tenant peu compte de la structure des documents XTIGER et des données.

Le projet consiste à développer un système de représentation de patrons XTIGER qui fonctionnerait du côté client. Suivant les concepts IHM, la présentation doit permettre à l'utilisateur d'utiliser au mieux les documents XTIGER.

Le système fonctionne dans un navigateur, ce qui nous laisse de nombreux choix quant à la technologie à employer. Cependant, par ce fait, de nombreux problèmes viennent se greffer : délai

de réponse, incompatibilité des navigateurs, des technologies, etc..

Contenu du rapport

Dans la partie suivante j'explique les choix qui ont été fait lors de l'élaboration de ce projet. Je m'intéresserai ensuite aux problèmes rencontrés lors du développement de la librairie. Une dernière partie pour expliquer les différentes améliorations possibles. Vous trouverez en annexe le manuel de l'utilisateur et celui du développeur où sont expliqués l'utilisation et le développement plus en détail.

Choix théoriques

Du développement d'interface à la génération d'interface

Initialement le projet consistait à développer des interfaces utilisables pour la représentation de documents XTIGER, patrons et instances. Il est rapidement apparu qu'un tel développement fixe n'était pas adapté car une représentation ne pouvait ne pas correspondre aux besoins de l'utilisateur et la modification de ces interfaces avaient un coût élevé en temps de développement. Pour modifier une interface, cela correspondait à réécrire une grande partie du système, quel que soit la technologie employée. Ce n'était donc pas, ou peu, réutilisable.

Pour palier à ce problème, l'idée était de développer un système qui générerait une interface en fonction des besoins de l'utilisateur. Il devait pouvoir définir simplement les transformations souhaitées dans un langage simple.

La librairie a donc été développée dans cette optique. L'utilisateur peut développer les interfaces en utilisant simplement du HTML. Le système s'occupe par la suite, en fonction de ce qu'a définit l'utilisateur, d'appliquer les transformations et de générer la visualisation.

Le choix technologique

Plusieurs technologies semblaient adaptées pour développer le système : XSLT, javascript, XBL sont les trois principales.

Le système doit être paramétrable, récursif, il doit pouvoir s'appliquer sur un document entier comme sur un nœud, intégrer tous les outils nécessaires au développement d'interface. L'utilisateur doit pouvoir changer sa représentation d'une partie du document à une autre. Il faut en définitive que cette transformation s'effectue du côté client.

Le système doit pouvoir s'intégrer facilement à des programmes modifiant ou créant des documents XTIGERS. Les systèmes en cours de développement sont développés en javascript, utilisant notamment des requêtes AJAX.

Le choix s'est donc porté sur le javascript. Premièrement car il répondait aux contraintes sus-mentionnées et deuxièmement s'intègre d'autant plus facilement aux applications actuellement

développées.

Le javascript nous donne la possibilité de travailler avec les arbres XML et HTML et quelque soit la technologie employée, le javascript aurait été intégré.

Le choix de la plateforme

Lors du commencement du développement, il était nécessaire de choisir un navigateur pour tester le code. Le choix s'est directement porté sur Mozilla Firefox. Il a été fait naturellement pour trois raisons : la première est que les outils de développement mis à disposition sur ce système rendaient le développement rapide et efficace, la seconde est que Mozilla Firefox est disponible pour tous les systèmes d'exploitation, la dernière est que ce navigateur gère efficacement la technologie XML.

Implémentation

Description du système

La librairie javascript intègre toutes les fonctions de transformations du document XTIGER. Elle s'applique dans une page internet, en prenant en entrée une document XTIGER et un fichier XHTML pour générer une représentation HTML du document.

Dans un premier temps elle initialise un objet contenant toutes les structures et les types du document ainsi que les structures de transformation définies par l'utilisateur.

Ensuite elle permet de parcourir en parallèle deux arbres. Le premier arbre est le document XTIGER qui ne sera pas modifié, mais qui permet de récupérer les informations sur les nœuds. Le deuxième arbre est la représentation HTML qui lui sera modifié. A chaque transformation de noeuds XTIGER, le système appelle une fonction appelée callBack ayant en paramètre les nœuds courants des deux arbres, ce qui permet à l'utilisateur de définir des étapes supplémentaires à effectuer.

Vous trouverez plus de détails sur le mode de fonctionnement du système dans les manuels de la libraire.

XML et HTML, faux frères

Les nœuds XML intégrés dans une architecture HTML ne sont pas interprétés, parfois même si le tag est un tag HTML valide. Pour exemple un nœud "span" extrait d'un document XML ne sera pas forcément interprété comme un nœud HTML, ne parlons évidemment pas de nœuds XML tel que "xt:component" qui, dépendant du navigateur, peut aller jusqu'à voir ses "enfants" devenir des "frères". La solution simple à ce problème est de restreindre l'utilisateur à utiliser des documents XHTML à la place du HTML. Dés lors, tout nœud est correctement interprété.

Lors du développement, il était fréquent de travailler sur des nœuds sans savoir au préalable si ce sont des nœuds HTML ou XML. Ceci peut poser problème dés lors que nous utilisons des attributs

qui ne fonctionnent que pour l'un ou l'autre. La solution a été de tester l'URI, ce qui nous permet de définir à quel type de nœud nous sommes confrontés et ainsi de quelle façon les traiter.

Concepts XTIGER

Le système XTIGER mélange les données et leur présentation. Les éléments d'interface, les tags HTML, et les données sont vues par le système comme des types que les éléments à insérer peuvent prendre. Il ne fait pas de distinction entre une balise HTML et une chaîne de caractère ou un nombre.

Un premier problème surgit : la définition des types du langage cibles. Il faut commencer par lister les types possibles. Pour donner à l'utilisateur la possibilité de restreindre ou agrandir la liste de ces types, je les ai définis dans le fichier XHTML contenant les structures.

Ces types posent un second problème, la structure correspondante à chacun de ces types doivent eux aussi être définis. Il faut gérer les transformations pour chacun. En ce qui nous concerne il faut définir une structure pour chacun des tags HTML. Dans la librairies, les structures sont définies génériquement par un nœud "span" contenant le nom du type. L'utilisateur peut définir lui-même manuellement d'autre transformation pour chacun des éléments en suivant les indications fournies dans le "user manual".

Délais conséquents

Le code initial qui avait été effectué était plus axés sur le bon fonctionnement de la librairie qu'à son optimisation. Il fallait donc sept secondes au navigateur pour générer la représentation. Il n'était pas tolérable qu'un utilisateur doive attendre autant de temps pour utiliser le système. Surtout dans un navigateur. J'ai donc chercher à optimiser le code. Pour ce faire, j'ai employé une simple technique de timer et ai ainsi pu définir quelles parties du programme prenait le plus de temps. Lorsque les opérations les plus lentes sont repérées, il a fallut comprendre pourquoi et comment gagner du temps. Voici quelques exemples de problèmes que j'ai rencontré.

Dans un premier temps, il est apparu que certaines opérations étaient inutiles dans le cas de traitement sur des nœuds HTML, bien obligatoire sur des nœuds XML. Il a suffit pour résoudre ce problème de tester le type du nœud auquel nous avions affaire et d'appliquer les opérations nécessaires.

Dans un second temps, le système perdait beaucoup de temps lors de certaines récursions. En approfondissant la recherche il s'est avéré que bon nombre de nœuds étaient visités lors du parcours de l'arbre alors que ce n'était pas nécessaire. Le système cherche les nœuds XTIGER et continuaient de descendre dans l'arbre alors qu'aucun descendant n'en était un.

Certaines fonctions utilisaient la récursion pour trouver des nœuds spécifiques, par exemple les nœuds ayant le nom de class égal à 'loop' lorsqu'on effectue une transformation. Ces opérations récursives utilisaient beaucoup plus de variables et d'appels de méthodes ce qui ralentissait le système. Supprimer ces récursions en utilisant un algorithme de pile a permis de diminuer le temps pris lors de ces opérations nécessaires.

Le clonage de nœuds et les expressions régulières sur des chaînes de caractères trop longues étaient les nouveaux poids en terme de temps. N'étant pas possible de se séparer complètement de ces opérations, il a fallut les éviter au maximum et dans le cas d'expression régulières, préférer des chaînes plus courtes.

Dépendant de la structure du document, il n'est pas possible d'éviter un temps long lors de la génération. Plus le patron XTIGER ou la structure de transformation sont complexe, plus la génération durera.

Ces tests de temps ont été effectués en utilisant le document XTIGER analysis_didactic.xtd . C'est un fichier XTIGER ayant beaucoup d'éléments et de référence à des components. Ceci a permis de faciliter la recherche des opérations lentes et donc de repérer ce qu'il fallait modifier en priorité.

Améliorations futures

Librairie indépendante du navigateur

Actuellement la librairie ne fonctionne qu'avec le navigateur Mozilla Firefox. Le choix de ce programme a été fait naturellement pour deux raisons : les outils de développement mis à dispositions sur ce système rendaient le développement rapide et efficace, la seconde est que Mozilla Firefox est disponible pour tous les systèmes d'exploitation.

Cependant nous remarquons que plus de 50% des utilisateurs utilisent encore Internet Explorer et que seulement 25% en tout utilisent Mozilla Firefox. Il est donc important de passer le système sur internet explorer et sur les autres navigateur, principalement safari qui détient aussi une grande part de marché.

Besoins de l'utilisateur

La librairie n'ayant à ce jour par encore été intégrée, il est difficile de prévoir les réels besoins des utilisateurs. De nombreux outils et méthodes devront être faits manuellement par eux. Les modifications de nœuds, de types, l'ajout et la suppression sont des exemples.

Lors de la transformation l'utilisateur pourrait souhaiter ajouter des informations au nœud tel que des identifiants ou des fonctions javascript de façon générique.

Intégration d'autres technologies

Le javascript est un langage interprété qui est assez lourd et peut demander beaucoup de ressources, surtout pour des documents et transformations conséquents comme dit plus haut. Il serait intéressant de proposer une alternative à la transformation récursive telle qu'elle est présente actuellement. L'une des possibilités seraient de générer, de la même façon que la librairie génère des représentations XHTML, des fichiers de transformation XSLT. Ainsi on pourrait appliquer les

transformations directement sur le document, n'utilisant alors le javascript que pour les opérations de modifications..

Optimisation

Actuellement la librairie est suffisamment rapide pour des documents et des structures de transformations testées. Dans le cas ou ils seraient plus importantes, il serait utile de trouver les parties de code qui peuvent être optimisées.

Conclusion

La librairie permet de créer rapidement de nouvelles interfaces ou de modifier celles déjà existantes. Les grands avantages de ce système est qu'il permet de tester sans aucune difficulté de nouvelles interfaces IHM.

L'utilisation de fichier externe contenant les différentes informations de transformation permet de modifier la manière de représenter les objets modifiés par exemple ou de donner à l'utilisateur le choix entre plusieurs interfaces disponibles.

Ce fichier externe permet aussi, étant donné que c'est du XHTML, d'avoir une visualisation de la structure directement en l'ouvrant dans un navigateur.

Remerciements

Je tiens à remercier les personnes du Center for Global Computing pour l'aide qu'ils m'ont apportée lors de l'élaboration de ce projet. Plus particulièrement Stéphane Sire pour ses nombreuses idées ainsi que son suivi, Micaël Paquier et Loïc Merz pour leurs conseils avisés.

Je remercie Cyril Junod, pour m'avoir servi d'API vivante et pour les différentes idées qui m'ont permises de résoudre de nombreux problèmes.

Je remercie en outre toutes les personnes qui m'ont aidé et soutenu.

Bibliographie

[1] McKerlie D. & MacLean A. EXPERIENCE WITH QOC DESIGN RATIONAL

[2] McKerlie D. & MacLean A. (1993) QOC IN ACTION – USING DESIGN RATIONAL TO SUPPORT DESIGN

[3] WORLD WIDE WEB CONSORTIUM - http://www.w3.org

[4] Zakas N. C. (2005) PROFESSIONAL JAVASCRIPT FOR WEB DEVELOPERS

[5] W3 SCHOOLS - http://www.w3schools.com/