LA RÉFÉRENCE TECHNIQUE DES PROFESSIONNELS DE L'INFORMATIQUE

it-expert

Nomadisme et système d'information

PAGE 6

Les nouveaux standards du Web pour un client riche

PAGE 21

BEA WebLogic Workshop & Platform

PAGE 32

Les mécanismes de synchronisation

PAGE 36

Tablet PC

PAGE 49

n°49
Bimestriel
Mai/Juin 2004
9€

Sommaire

6
NOMADISME
& système d'information

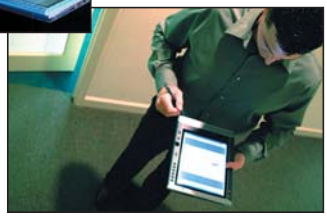
21 **Les nouveaux standards du Web pour un client riche**

Les mécanismes de synchronisation

36

Tablet PC

49



Les nouveaux standards du Web pour un client riche

L'apparition de nouveaux navigateurs relance la concurrence du marché sur ce secteur. Ceci nécessite de mettre en place des standards de développement qui s'appuyent sur les recommandations du W3C afin de favoriser la compatibilité de l'application produite vis-à-vis des navigateurs récents. Cette compatibilité doit permettre de pouvoir mettre en place des ergonomies « enrichies » (effets dynamiques) et déporter des traitements non métier sur le navigateur pour améliorer les performances et proposer des ergonomies plus attractives avec des fonctionnalités avancées.

Ainsi, une nouvelle terminologie de « client riche » commence à faire son apparition dans le vocable technique pour la distinguer des terminologies plus anciennes telles que « client lourd » ou « client léger ». Mais est-ce que le « client riche » sous-entend une révolution d'interface machine impactant les architectures applicatives ou n'est-il qu'un effet d'annonce pour rajeunir des principes technologiques anciens ?

Pour éclaircir le débat autour de ce nouveau principe, il est important de recenser les attentes du marché pour ensuite faire le tour des possibilités techniques offertes actuellement pour y répondre.

Nouvelle approche : client riche

Convergence du client/serveur et du Web

Le principal objectif du client riche est de concilier les avantages du client lourd, propre aux architectures propriétaires du client/serveur et ceux du client léger propre aux architectures Web multi-niveaux standardisées. Ainsi, il offre la possibilité de déployer une application à moindre coût et d'accéder aux traitements métiers et aux contenus informatifs distants via des standards tels que le HTTP(S) et le XML. Le client riche garantit une meilleure portabilité car il ne nécessite pas une compilation de l'application pour chacun des systèmes d'exploitation ciblés.

Ergonomie du client léger « enrichie »

Le client léger avait pour vocation de fournir une ergonomie basée sur une navigation hypertexte afin d'accéder à un contenu informatif ou à la validation de formulaire pour effectuer des transactions. Ce type d'ergonomie multiplie les requêtes et contraint le découpage des applications de gestion transactionnelle en pages unitaires. Nous sommes loin de pouvoir gérer un dossier métier sur une seule page, en sachant que chaque action de l'utilisateur nécessite un aller/retour avec le serveur sous forme de page.

A contrario, le client riche apporte un dynamisme aux pages, réalise des traitements événementiels liés à des actions client et sépare très nettement le contenu informatif présent dans une page de son rendu. Son ergonomie s'approche de celle du client lourd plus intuitive pour l'utilisateur. Ainsi, par l'intermédiaire de composants graphiques enrichis, il ouvre un éventail de possibilités moins limitatif pour une maîtrise d'ouvrage.
Les objections habituellement invoquées par la maîtrise d'œuvre du type « impossible vis-à-vis des temps de réponse ou de la technicité utilisée », « nécessité de découper ce formulaire en plusieurs pages », « cette ergonomie est trop complexe » sont ainsi levées.
Par rapport au client léger, le client riche améliore les temps de réponse et la capacité de montée en charge du serveur en effectuant lui-même une partie des traitements non-métiers (tri et filtrage de l'information, génération du rendu IHM, version imprimable...).

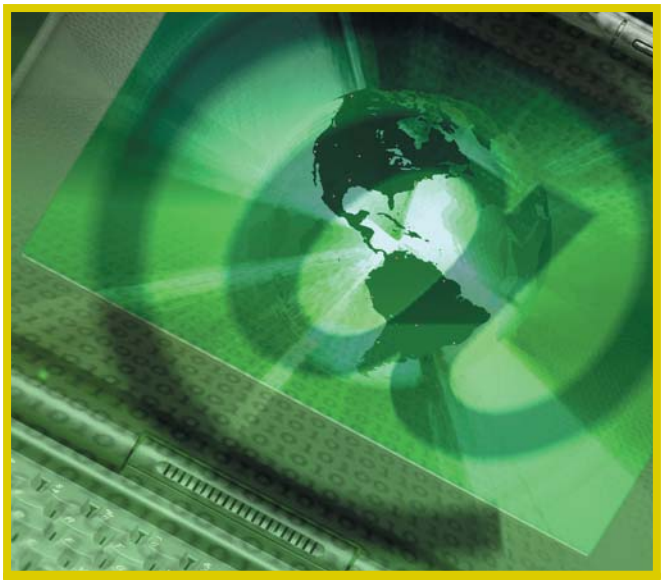
Respect des standards et de la charte graphique

La mise en place d'une application basée sur les principes du client riche nécessite que le code produit respecte les standards Internet. Ceci garantie la compatibilité avec les navigateurs récents du marché (voir encadré 1).

Ainsi, une conception suivant les recommandations du W3C simplifie la maintenance d'un site du fait d'une bonne répartition des logiques de programmation d'une page en externalisant les fichiers source (*.css, *.html, *.xml, *.js). En s'affranchissant des technologies éphémères et en s'interdisant d'utiliser les spécificités de tel ou tel navigateur, cette pratique favorise une plus grande durée de vie des pages.
Cette standardisation lors de la phase de développement ne coûte pas plus cher et son respect vérifié au moyen des validateurs fournis par le W3C.

Validation W3C

Par ailleurs, la réalisation d'une charte graphique est un atout majeur pour le plan de communication d'une entreprise. Elle pérennise l'image de l'entreprise, facilite pour l'utilisateur la prise en main, la recherche d'information, l'accessibilité et l'interactivité des fonctionnalités du site... La mutualisation et le poids des visuels doivent aussi être pris en compte. Il est conseillé de s'entourer de professionnels (Webagency) pour formaliser cette conception autour d'un référentiel ergonomique et ainsi commander les visuels et les feuilles de style nécessaires. Les composants graphiques du client riche reposeront par la suite sur cette charte graphique pour garantir une cohérence ergonomique de l'ensemble des sites de l'entreprise.



Historique du client léger : émergence des standards du W3C

Lors de l'émergence des technologies Web, seul le standard HTML 3.0 était admis par les deux principaux éditeurs de navigateur de l'époque : Microsoft et Netscape.
Rapidement de nouvelles technologies ont surgi pour enrichir les possibilités de ce langage avec l'apparition des feuilles de style (modélisation décrivant le rendu des éléments constituants une page) et du Document Object Model (DOM : modélisation décrivant l'ensemble des éléments constituant une page pour pouvoir les influer).
Hélas, cette émergence technologique arrive dans un contexte de « guerre intense » entre ces deux éditeurs, qui essayent chacun d'imposer leur vision d'ajout de dynamismes sur une partie d'un document HTML. Ainsi lors de la sortie de la version 4 de leurs navigateurs, Microsoft introduit les balises « div » / « iframe » et les notions de DHTML tandis que Netscape enrichit son DOM Level 0 avec ses balises « layer » / « ilayer ». Cette différenciation obligeait le développement d'un code « cross-browser » pour intégrer ces deux stratégies autour d'un même code par détection de navigateur.
Heureusement à la fin de cette guerre, le consortium du W3C entre en scène pour chercher un consensus autour de standard Internet en réunissant l'ensemble des acteurs du marché. Ainsi, voyons-nous apparaître les normes XML, le XHTML (balisage XML structuré), le CSS (feuilles de style) et le DOM W3C. La stratégie du W3C est alors de fournir un formalisme pour le contenu informatif d'une page (XML), pour la définition des éléments présents dans une page (XHTML), pour le rendu de ces éléments (CSS) et pour le dynamisme de ces éléments et la manipulation du contenu informatif (DOM). D'ailleurs, cette logique a été récemment étendue avec l'adoption des standards SVG pour les graphiques vectoriels animés en deux dimensions et le SMIL pour les multimédias synchronisés concurrents directs des « players » propriétaires tels que Macromédia Flash ou Windows Média.

Encadré 1

Le mode déconnecté

La mobilité des utilisateurs devient un besoin de plus en plus récurrent dans les applications modernes de gestion. Ce nomadisme nécessite de pouvoir effectuer des saisies sans forcément être connecté à un serveur distant.
Dans un contexte connecté, le client riche prend en charge la couche présentation et le transport des données. En mode déconnecté, il est alors possible de pré-charger l'ensemble des données nécessaires et d'exécuter ensuite son interface en local vis-à-vis de ces seules données. Une fois l'utilisateur reconnecté, le client riche prend en charge la transmission des seules données modifiées, ajoutées ou supprimées. Reste à définir côté serveur la stratégie pour la synchronisation de ces modifications vis-à-vis de la source de données. Des alertes sous forme de compte rendu doivent être remontées à l'utilisateur final pour une prise de décision sur toute modification concurrente ou non conforme à une règle de gestion métier.

Choix technologiques pour le client riche : trois philosophies distinctes

L'offre du marché pour cette nouvelle approche client dépend essentiellement du choix du socle d'exécution du client riche. Présentement, ce choix technologique se limite à trois propositions complètement distinctes dont chacune de leur spécificité est débattue ici pour aider une maîtrise d'œuvre dans sa prise de décision.

Socle d'exécution basé sur une machine virtuelle

Cette philosophie repose sur une compilation en local du code source pour une exécution sur une machine virtuelle. Le client riche repose sur une bibliothèque de composants graphiques propriétaires à l'éditeur, limitant ainsi les possibilités associées à une charte graphique. L'inconvénient tient à ce que la technologie utilisée imprime fortement son empreinte sur le rendu de l'application (application windows .Net ou java...) au même titre que pour un client lourd (application Visual Basic ou C ou Delphi...). Le rendu de l'image d'entreprise s'avère alors souvent moins fidèle que celui autorisé par l'application stricte d'une charte graphique par une pure application web.
A partir de ce socle d'exécution, la gestion du transport de données peut être effectuée à partir de flux HTTP/XML ou s'incorporer dans une architecture orientée services sur des flux SOAP. Le mode déconnecté en fonction de l'éditeur est souvent pris en charge. La grande force de ce principe réside sur sa technologie de déploiement comme pour Microsoft et Sun permettant de déployer les services métier du serveur d'application. Leur client riche a ainsi une gestion asynchrone de ces services tout en assurant une cohérence d'ensemble propre aux avantages d'une architecture web centralisée.

Reste tout de même que ces technologies fondent l'IHM sur des solutions propriétaires et sont loin de les ouvrir vers des standards du W3C plus garant d'une interopérabilité et d'une plus grande longévité.

Socle d'exécution basé sur le navigateur Web

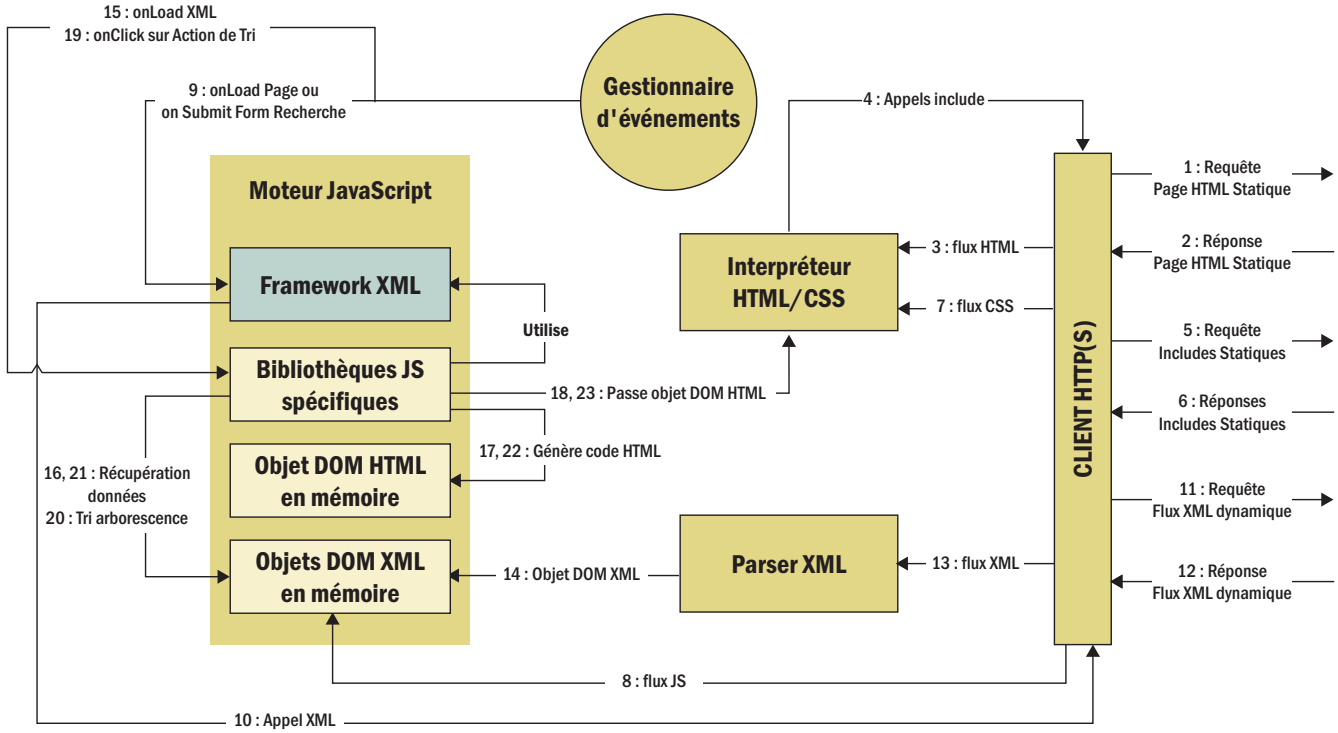
Contrairement à celle basée sur une machine virtuelle, cette philosophie repose seulement sur les standards Internet définis par le W3C en s'appuyant sur le navigateur client, son interpréteur JavaScript et son parser XML.

Les navigateurs récents du marché sont capables de gérer les flux XML grâce au parser qu'ils embarquent.
Ce dernier vérifie la conformité du format du document XML (sinon envoi d'une exception susceptible de fournir les informations nécessaires à une gestion d'erreur) et monte en mémoire le contenu du document XML dans un objet DOM. Cet objet DOM

Encadré 2 : Principales offres du marché pour un socle d'exécution basé sur une machine virtuelle

Editeur	Microsoft	Sun	Macromédia	Dreamfactory	Open Source
Produit	.Net	JRE	Flash MX	Dreamfactory Enterprise Edition 6.0	PHP-GTK
Machine virtuelle	CLR	JVM	Lecteur Flash	Plug-in Dreamfactory Runtime	Zend Engine
Technologie de déploiement	No Touch Deployment	Web Start	Navigateur	Navigateur	Manuellement
Framework de développement	Framework .Net (Windows Forms)	Java Foundation Classes (Java Swing)	Bibliothèques de composant Flash	Bibliothèques J2EE ou .Net	GTK+ Classes
Environnement de développement	Payant	Fluctuant suivant l'offre	Payant	Payant	Gratuit

Figures : Modèles de collaboration

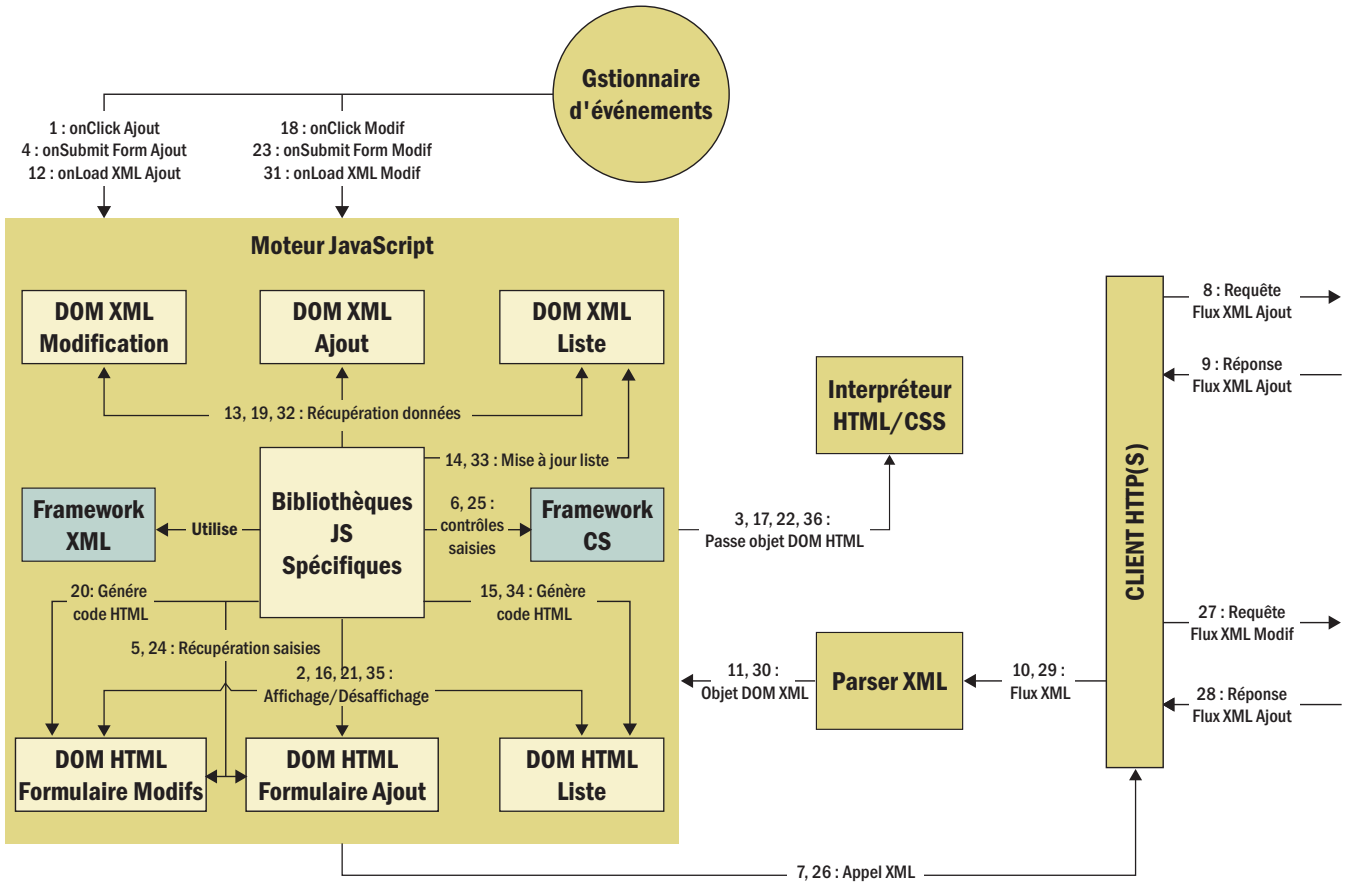


Modèle de collaboration sur le poste client lors de la consultation d'une liste de données

Description des différentes étapes :

1 Appel d'une page HTML statique dans la frame de contenu suite à une navigation de l'utilisateur si la page n'est pas déjà présente dans le cache du navigateur.
2 Réponse réceptionnée avec un code statut valide sinon page d'erreur.
3 Page HTML statique interprétée par le navigateur.
4 L'interpréteur détecte l'ensemble des éléments constitutifs au document HTML et effectue les requêtes nécessaires sur les éléments externes (*.css, *.js, *.gif et *.jpg).
5 Requêtes sur l'ensemble des éléments externes inclus dans le document HTML.
6 Réception des réponses associées.
7 Feuille de style spécifiée dans l'en-tête du document HTML envoyée à l'interpréteur dès réception pour qu'il puisse intégrer les styles lors de l'interprétation du corps du document HTML.
8 Bibliothèques JS spécifiques chargées en mémoire pour interprétation par le moteur JavaScript. Elles sont précisées dans l'en-tête du document HTML.
9 Interprétation terminée du document HTML déclenchant l'événement « onLoad » ou événement « onSubmit » déclenché lors de la validation d'un formulaire de recherche. Le gestionnaire d'événement appelle la fonction de chargement de document XML présente dans le framework.
10 Vérification avant appel du fichier XML si l'objet DOM XML désiré n'est pas déjà présent en mémoire (mémoire associée à la fenêtre principale). Sinon appel XML par le biais d'une URL en ajoutant les critères saisis valides si appel déclenché par l'intermédiaire d'un formulaire de recherche.
11 Requête au serveur pour récupérer le document XML désiré.
12 Réception du document XML désiré.
13 Passage du document XML au parseur XML pour validation du format. Si non valide, une gestion d'erreur est déclenchée et détectée par le framework pour affichage à l'utilisateur.
14 Si format valide, montée en mémoire de l'objet DOM XML associé à ce document XML.
15 Appel de la fonction de génération HTML présente dans la bibliothèque JS spécifique. Cet appel est effectué par le gestionnaire d'événement après montée en mémoire de l'objet DOM XML.
16 Récupération des données utiles par interrogation de l'arbre présent dans l'objet DOM XML.
17 Génération de l'objet DOM HTML (DIV ou zone dynamique présent dans le document HTML) en y intégrant les données récupérées. L'étape 16 et 17 s'effectue en parallèle sur une même boucle par soucis d'optimisation.
18 Demande d'interprétation de cet objet DOM HTML pour affichage à l'utilisateur.
19 Click souris par l'utilisateur sur l'en-tête d'une colonne d'un tableau déclenchant l'appel d'une fonction de tri présent dans la bibliothèque JS spécifique.
20 Tri par parcours de l'arbre présent dans l'objet DOM XML. Ce tri s'effectue par l'intermédiaire d'une fonction présente dans le framework. Cette fonction est optimisée par la mise en œuvre d'un parcours du type Quick Sort et permet de trier suivant un critère de date, de nombre ou de chaîne de caractères.

21-22-23 : Idem que les étapes 16-17-18.



Modèle de collaboration sur le poste client lors de la validation d'un formulaire

Description des différentes étapes :

Pré-requis : le DOM XML lié à une liste de données est déjà chargé en mémoire comme défini dans le modèle de collaboration lors de la consultation d'une liste de données.

1 Click utilisateur sur un bouton d'ajout d'un élément à la liste. Cet événement déclenche une fonction présente dans la bibliothèque JS spécifique par le gestionnaire d'événements.

2 Affichage du formulaire d'ajout et désaffichage de la consultation de la liste par l'intermédiaire d'API DOM.

3 Répercussion de cette gestion sur l'affichage de la page résultante à l'utilisateur.

4 Soumission du formulaire après saisie des nouvelles entrées par l'utilisateur. Le gestionnaire d'événements appelle la fonction d'ajout présente dans la bibliothèque JS spécifique.

5 Récupération des données saisies par l'utilisateur par l'intermédiaire d'API DOM.

6 Contrôles de format, de critère de présence obligatoire et de règles fonctionnelles sur les saisies en appelant les fonctions génériques présentes dans le framework de contrôles de surface et en passant en paramètres les données fonctionnelles et techniques spécifiques au cas d'utilisation.

7 Appel du document XML par l'appel d'une fonction présente dans le framework XML. Cet appel s'effectue en précisant dans l'url d'appel les paramètres de saisies de l'utilisateur.

8 Requête au serveur pour récupérer le document XML désiré.

9 Réception du document XML désiré contenant les données ajoutées en base (Pas forcément le même formalisme que celui saisi par l'utilisateur).

10 Passage du document XML au parseur XML pour validation et montée en mémoire de l'objet DOM XML associé.

11 Montée en mémoire de l'objet DOM XML associé au document XML réceptionné.

12 Appel par le gestionnaire d'événements de la fonction de retour d'un ajout présente dans la bibliothèque JS spécifique.

13 Récupération des données ajoutées en base ou d'une erreur de traitement entraînant l'affichage d'un message d'erreur présent dans l'objet DOM XML.

14 Ajout d'un nouvel élément dans l'arbre de l'objet DOM XML de la liste des éléments par API DOM.

15 Génération du code HTML lié à cette liste.

16 Désaffichage du formulaire d'ajout et affichage de la liste des éléments par API DOM.

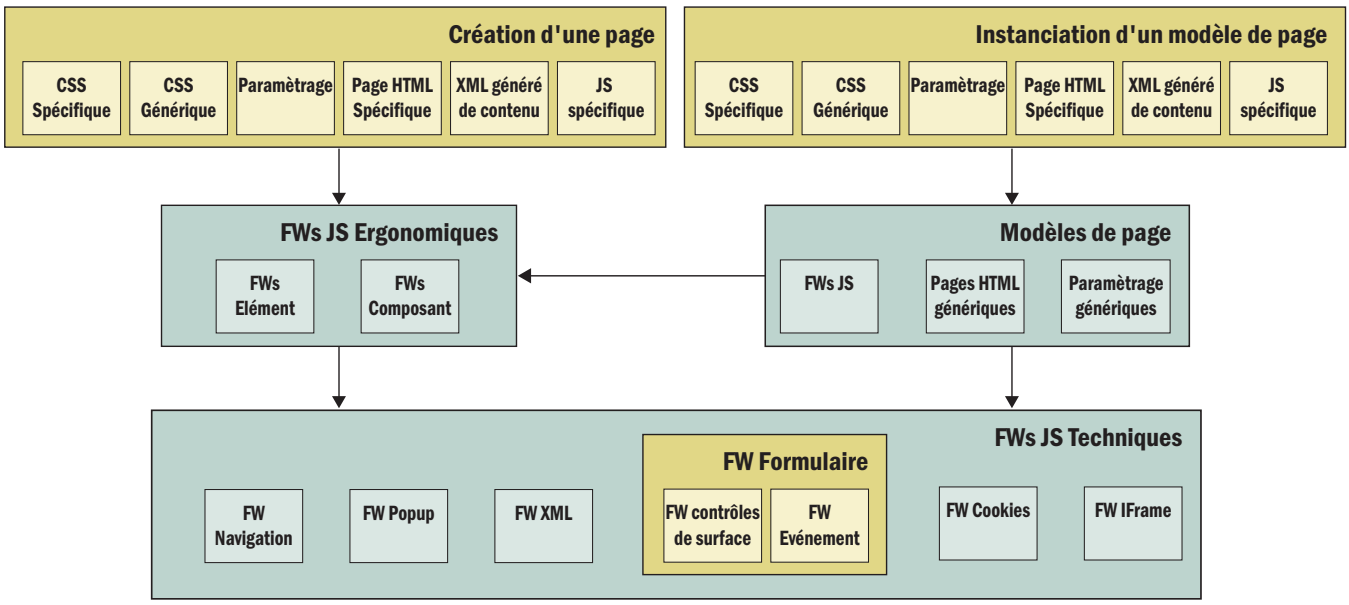
17 Demande d'interprétation des objets DOM HTML pour affichage à l'utilisateur.

18 La gestion d'une modification d'un élément de la liste s'effectue par l'intermédiaire des mêmes étapes que lors d'un ajout. Ici, le document XML renvoyé reflète les données de l'élément mis à jour en base et deux étapes supplémentaires sont effectuées.

19 Récupération des données de l'élément à modifier dans l'objet DOM XML de la liste des éléments.

20 Initialisation du formulaire de saisies de modification pour afficher les données à modifier. Cette initialisation s'effectue soit par génération HTML ou par initialisation d'un formulaire « caché » présent dans la page par l'intermédiaire d'API DOM.

Encadré 3 : Découpage du framework ergonomique JavaScript client riche



Principe du framework ergonomique JavaScript client riche

Cette figure présente une description structurelle du framework ergonomique à implémenter.
Ce découpage du framework ergonomique en plusieurs frameworks JavaScript a pour objectif de répondre à deux besoins de développement :

- Instancier une nouvelle page à l'aide d'un modèle de page. Cette instanciation consiste alors juste à effectuer un paramétrage spécifique au cas d'utilisation métier et à ajouter les codes (X)HTML (positionnement des éléments constitutifs dans un tableau…) et JavaScript (contrôles de surface…) spécifiques. Le code HTML spécifique est ajouté au code HTML fourni par le modèle de page.
- Développer une nouvelle page ne répondant pas à un modèle de page existant. Ce développement peut bénéficier des frameworks ergonomiques déjà présents pour faciliter sa mise en œuvre. Dans ce cas, le développement nécessaire reprend l'ensemble des contraintes décrites lors d'une instanciation mais avec la nécessité de développer l'ensemble du code HTML d'une part, les éléments et composants spécifiques à ce cas d'utilisation d'autre part. Avant d'effectuer ce développement, il sera intéressant d'étudier si ce développement peut être mutualisé soit pour ses éléments ou ses composants spécifiques, voir même, réaliser l'objet d'un modèle de page.

C'est dans cette optique que ce découpage suit le principe de « poupées russes » pour découpler et surcharger les frameworks répondant aux problématiques suivantes :

- Techniques :
  - « Framework Navigation » mutualisant l'ensemble du code nécessaire à la navigation en prenant en compte les problématiques de présence de frames et/ou d'iFrames. Ce framework gère aussi l'intégration ou non de la navigation dans l'historique du navigateur.
  - « Framework Popup » pour la gestion des fenêtres surgissantes avec spécification du rendu de la fenêtre, de la gestion de la demande d'impression de son contenu, du caractère d'unicité de cette fenêtre…
  - « Framework XML » pour la mutualisation de la gestion des flux XML vis-à-vis des appels, des messages d'attente, de la gestion des erreurs et du TimeOut, de l'appel de la fonction liée à la réception du flux XML, de la sauvegarde des flux XML sur disque… Ce framework fournit aussi une boîte à outils d'interrogation et de manipulation avancée de ces flux XML (Tri, test de valeur optionnelle ou de sous-arbre vide, test de la typologie du nœud XML avant interrogation ou manipulation…).
  - « Framework Formulaire » pour gérer les contrôles de surface sur la saisie du client, de valider le formulaire par la touche ENTER lors de la saisie et d'empêcher les validations non désirées tant que la première validation n'a pas été réalisée.
  - « Framework Cookies » pour la création, interrogation, manipulation et suppression des cookies sessions.
  - « Framework IFrame » pour la gestion d'IFrame dans une page en mutualisant l'interrogation et la mise à jour de son contenu suivant le type d'IFrame désiré (taille fixe avec scroll-bar ou taille en fonction du contenu).
- Ergonomiques : génération et gestion d'éléments et composants liés à la charte graphique et aux besoins des applications ciblées par l'entreprise. Ce framework permet de pouvoir répondre aux deux besoins de développement et faciliter la maintenance future par son effort de mutualisation.
- Modèles de page : surcharge du framework ergonomique pour gérer un ensemble d'éléments ou de composants dans le cadre d'un modèle de page.
- Métier : ne faisant pas partie du framework ergonomique mais d'un cas d'utilisation donné sous la forme de fonctions ou de bibliothèques ou de frameworks JS spécifiques.

L'implémentation de ce framework doit être conforme vis-à-vis d'un standard de développement, décrit dans une documentation détaillée référençant l'ensemble des APIs fournies et illustré dans un guide de développement pour offrir une sorte de « manuel utilisateur » aux développeurs.

permet d'accéder directement à l'arborescence des éléments décrits dans le document XML via des APIs DOM sous formes de directives JavaScript conformes au standard W3C.

Ces APIs interrogent et manipulent une arborescence XML au même titre que celle constituée par les éléments d'un document (X)HTML. Lors de la détection d'événement en JavaScript, il est ainsi possible de déclencher des traitements locaux tels que la génération (X)HTML associée au contenu informatif présent dans le flux XML, des dynamismes d'affichage/désaffichage, de glisser/déplacer, de validation de formulaire, de tri, de filtrage…

Contrairement à une ergonomie dite « classique » consistant à baser le client sur une navigation par page, une ergonomie complexe s'avère avantageuse dans certains cas d'utilisation propices comme la gestion de plusieurs actions ou l'appropriation du rendu de la page par l'utilisateur… Des modèles de collaboration illustrant ces principes d'ergonomie complexe sont proposés (voir figures).

Ainsi, une seule page (X)HTML statique est nécessaire pour gérer un métier ou un cas d'utilisation car l'ensemble des résultats liés aux actions client est affiché ou masqué par le biais de zones dynamiques (tag). Les données résultantes de ces actions sont appelées et montées en mémoire au format XML et converties au format (X)HTML sur le client.
Par cet intermédiaire, seules les données sont générées par le serveur permettant pour la gestion d'un métier de prendre en compte la persistance des données sur le client dans le cycle de vie des actions de l'utilisateur.

Autre point positif, l'accès en mémoire sous forme d'arbre XML aux données rapatriées offre la possibilité d'interagir avec celles-ci sur le client comme lors du tri d'une liste, de l'ajout d'un élément après confirmation du serveur dans l'arbre XML pour réaffichage de la liste des éléments…

Ce type d'ergonomie a donc pour avantages :

- plus grande capacité de montée en charge du fait d'une sollicitation moindre des couches multi-canal et services. Cette diminution est renforcée avec la possibilité de garder en mémoire le contenu informatif (persistance de l'objet DOM) tout le long du cycle de vie des actions de l'utilisateur pour un cas d'utilisation métier.
- souplesse ergonomique plus importante par l'utilisation de zones dynamiques.
- flux d'échange de taille plus faible.
- utilisation optimum du cache client par l'utilisation d'objets statiques plus importante (.html et .js).

A contrario, ce type d'ergonomie a pour inconvénient de restreindre la compatibilité des navigateurs car elle nécessite la conformité des normes W3C récentes. D'un autre côté, elle restreint seulement les navigateurs qui ne sont plus supportés par leurs éditeurs (les versions 4 de Netscape et de MSIE d'une part et MSIE sur Mac d'autre part).
Mais, l'inconvénient principal de ce type d'ergonomie tient à la complexité de programmation liée à la prise en compte de flux XML pour générer côté client le code HTML associé.

Il en découle souvent la nécessité de réaliser un framework ergonomique JavaScript client riche (voir encadré 3) pour masquer la complexité de mise en œuvre et ainsi diminuer les coûts de maintenance et de développement.

L'autre atout majeur de cette philosophie du client riche est d'associer à la gestion technique de l'enrichissement du client et de l'ergonomie complexe, des APIs de génération d'éléments constitutifs d'une page en accord avec la charte graphique.
Ainsi, l'utilisation du framework ergonomique JavaScript client riche garantit la cohérence systématique avec la charte graphique. La maintenance du site vis-à-vis de la charte graphique s'avère simplifiée car elle n'impacte le plus souvent que le framework et non plus l'ensemble des pages du site.
Dans cette philosophie, le mode déconnecté est gérable en téléchargeant le site ou une partie du site comme lors de l'utilisation d'un aspirateur de site. Les différences se situent dans la gestion des flux XML placés sur le disque local et sauvegardés après manipulation sur ce même emplacement disque. L'application renvoie lors de la connexion suivante les seules

modifications apportées par l'utilisateur par comparaison de fichier XML. Des services métier de synchronisation avec la source de données sont alors à prévoir pour gérer la récupération de ces données en central. Cette implémentation côté serveur détermine seulement la stratégie de synchronisation désirée car elle fait appel aux mêmes services métier que lors du mode connecté.

Socle d'exécution basé sur des formulaires XML interactifs

Une autre possibilité est susceptible d'être intéressante en téléchargeant ou en recevant par mail des documents contenant des formulaires interactifs. Ainsi, Infopath de Microsoft (disponible depuis la version 2003 de sa suite Office), Adobe PDF (disponible à partir de la version 6.0 de son reader) et XForms du W3C permettent à l'utilisateur de stocker sa saisie sous format XML et de la renvoyer via des WebServices. Des langages de script intégrés au document réalisent des calculs et effectuent les contrôles de surface au même titre que le code JavaScript interagissant avec un formulaire présent dans une page Web.

Les principaux avantages de cette philosophie sont de faciliter la prise en charge du mode déconnecté et l'automatisation de workflow XML. Les inconvénients résident dans une simplicité des cas d'utilisation (un seul formulaire présent dans le document) et dans la non émergence de réel standard malgré celui avancé par le W3C. XForms n'a pas encore la maturité suffisante ni le poids des principaux acteurs de suite bureautique (OpenOffice, Microsoft et PDF sont chacun partis dans une direction plus ou moins propriétaire).

Il est en revanche possible de cumuler les avantages du socle d'exécution basé sur des formulaires XML interactifs et ceux du navigateur Web en offrant un même service métier on-line via un navigateur Web ou off-line via un formulaire XML interactif. Lors d'une saisie sur un formulaire off-line, il est alors possible de sauvegarder la saisie dans un fichier XML pouvant servir d'auto alimentation au service on-line.

Conclusion

Les critères de choix pour le client riche doivent prendre en compte d'une part un existant applicatif basé sur des clients légers ou lourds et d'autre part les nouveaux besoins ergonomiques et fonctionnels en terme de cohérence et d'enrichissement.

Ainsi, une solution basée sur une offre d'éditeur peut facilement s'intégrer dans une architecture existante déjà basée sur cette même offre particulièrement pour les applications de back-office. L'adoption d'une solution proche des standards Internet limite aussi les impacts d'architecture globale en intégrant par exemple des briques de l'open source dans une architecture existante. Cette solution est particulièrement bien adaptée pour les applications en front-office. Enfin, une solution basée seulement sur des formulaires XML interactifs peut tout à fait suffire à une application simple limitée en nombre d'utilisateurs simultanés. Elle est d'ailleurs idéale pour gérer certaines fonctionnalités applicatives en mode déconnecté.

Quoi qu'il en soit, le client riche ne doit pas être perçu comme une révolution remettant en cause l'ensemble des décisions stratégiques mais plutôt comme une maturité du marché autour des standards plus décisifs dans les choix stratégiques. ■

Pour en savoir PLUS

Actualités sur les standards
http://www.w3.org/
http://openweb.eu.org/
http://xmlfr.org/

Socle d'exécution basé sur une machine virtuelle
http://www.dotnetguru.org/articles/WinFormsAndSwing/SwingForms.htm
http://www.windowsforms.net/
http://java.sun.com/products/jfc/index.jsp
http://www.dreamfactory.com/products/technology.html
http://www.macromedia.com/resources/business/rich_internet_apps/
http://gtk.php.net/

Socle d'exécution basé sur un navigateur web
http://savannah.gnu.org/
http://www.quirksmode.org/dom/importxml.html
http://www.webreference.com/programming/javascript/domwrapper/index.html

Socle d'exécution basé sur des formulaires XML interactifs
http://www.microsoft.com/france/office/infopath/prodinfo/default.asp
http://www.adobe.com/enterprise/xml.html
http://www.w3.org/MarkUp/Forms/

Steve PÉGUET
Chef de projet,
Expert architectures distribuées

Aubay est une société européenne de service et de conseil spécialisée dans les nouvelles technologies de l'information. Son offre est particulièrement reconnue dans le domaine des architectures complexes, du décisionnel, de la sécurité, des réseaux à très haut débit et de la Gede. Avec ses 650 collaborateurs et un CA de 50 Meuros en 2003, la société est implantée en Belgique, au Luxembourg, en Espagne, en Italie et en France.

Pour en savoir plus : www.aubay.com