Architecture d'un Serveur Multimédia pour les Sciences de l'Ingénieur

Nicolas Delestre, Béatrice Rumpler

Architecture d'un Serveur Multimédia pour les Sciences de l'Ingénieur

Nicolas Delestre(1), Béatrice Rumpler(2)

(1) PSI LIRINSA - INSA de Rouen
BP 08 Place Emile Blondel
76131 Mont Saint Aignan Cedex
Tel : (33) 2 35 52 84 70
Fax : (33) 2 35 52 84 98
E mail : Nicolas.Delestre@insa-rouen.fr

(2) LISI - INSA de LYON
20, avenue Albert Einstein
69621 Villeurbanne Cedex France
Tel : (33) 4 72 43 83 92
Fax : (33) 4 72 43 85 18
E-mail : Beatrice.Rumpler@if.insa-lyon.fr

Résumé

Cet article présente l'architecture d'un serveur internet multimédia pour les sciences de l'ingénieur nommé SEMUSDI. L'originalité de ce serveur découle de l'utilisation conjointe d'une base de données orientée objet, d'un ensemble de scripts CGI et de librairies Javascript. Cela permet d'obtenir un serveur homogène, rapide et véritablement interactif. Dans une première partie, nous allons rappeler les objectifs du projet. Puis nous présenterons les fonctions du serveur et l'architecture générale définie. Ensuite, nous détaillerons l'analyse et les solutions techniques proposées pour le système et nous terminerons par les perspectives d'évolution du projet.

Mots clés : Serveur Internet, Base de données orientée objet, Javascript, Java, CGI

Abstract :

In this paper, we present the architecture of a multimedia Internet server specialized in engineering science field, called SEMUSDI. The original feature of this server is the co-using of an object-oriented database and a set of CGI scripts and Javascript libraries. This association permits to propose an Internet server homogeneous, performant and really interactive. In the first part, we introduce the goals of our project. Then, we describe the server functions and the software architecture. In the last part, we explain the technical solutions chosen for the system, and we finish with the prospects.

Keywords : Internet server, Object-oriented database, Javascript, Java, CGI.

Introduction.

Il existe aujourd'hui un besoin important donc une demande croissante d'informations spécialisées dans les sciences de l'ingénieur aussi bien de la part des étudiants, que des enseignants, des ingénieurs ou des chercheurs, sur le réseau Internet. Or, celui-ci renvoie généralement un flot d'information trop important et mal ciblé donc peu exploitable ou sans intérêt. Ce constat nous a conduit à travailler sur la définition d'un SErveur MUltimédia pour les Sciences De l'Ingénieur (SEMUSDI), adapté d'une part à la demande d'utilisateurs isolés qui pourront être des étudiants ou des ingénieurs et d'autre part aux organismes d'enseignement et de formation.

Ainsi, nous avons conçu un serveur de documents multimédia capable de proposer des documents simples, élémentaires très spécialisés que l'on nommera « briques ». Le système que nous avons défini ne permet pas de proposer une formation complète structurée, gérant un cursus de formation dans un domaine scientifique, il met à la disposition des utilisateurs potentiels, formateurs ou étudiants, des documents courts techniques et très spécialisés.

La réalisation de ce projet est principalement assurée par des équipes de recherche des INSA de Rouen et de Lyon en collaboration avec d'autres partenaires : institutionnels comme le CRDP (Centre Régional de Documentations Pédagogiques), universitaires comme l'Université Laval (Québec) et industriels comme Renault.

Cet article présente le projet SEMUSDI [HREF1] et sa réalisation. Dans une première partie, nous rappellerons les objectifs du projet puis nous présenterons les fonctions du serveur et l'architecture générale définie. Ensuite, nous détaillerons l'analyse et les solutions techniques proposées pour le système, et nous terminerons par les perspectives d'évolution du projet.

Le serveur SEMUSDI.

Nous constatons qu'il existe dans nos écoles d'ingénieurs ou universités un volume important d'informations souvent peu exploitées parce qu'elles ne sont pas connues ou difficilement accessibles.

D'autre part, la notion de document brut, élémentaire mais bien ciblé constitue un support de base qui intéresse fortement les enseignants, les étudiants, les producteurs de systèmes d'enseignement assisté par ordinateur tels que le système METADYNE (Delestre, Gréboval et Pécuchet 1997), mais aussi les ingénieurs dans le cadre de la formation continue.

Ces constats nous ont naturellement conduit à définir un système capable de regrouper ces informations en les organisant pour les rendre facilement accessibles ; de plus nous avons profité de l'apport des nouvelles technologies pour définir un serveur de documents multimédia accessible par le réseau Internet. La notion de documents multimédia est en effet essentielle dans un contexte de formation, elle permet de travailler sur des images, d'effectuer des manipulations sur des objets réels ou virtuels, de gérer l'interactivité.

Les fonctions du serveur.

Le serveur SEMUSDI est conçu pour *collecter*, *gérer*, *stocker*, *rechercher*, *consulter* et *diffuser* des documents « bruts » de type hypermédia à des utilisateurs répartis sur plusieurs sites géographiques distants. Les documents pourront être sous différentes formes :

- texte et hypertexte (HTML),
- images fixes ou animées,
- son,
- logiciel d'animation et de simulation,
- exercices d'application courts,
- exemples de cas concrets.

L'ensemble de ces documents bruts sera géré au sein d'une base de données, accessible via le WEB. La notion de document « brut » signifie que le document est court et très ciblé sur un point technique, qu'il peut s'intégrer dans une structure organisée comme un cours, mais il ne constitue pas à lui seul un support suffisant pour une formation. Le document brut pourra par exemple être utilisé par un enseignant pour illustrer un cours, par un étudiant pour retrouver des démonstrations et formules physiques précises.

Ces documents « bruts » que l'on nommera aussi « briques élémentaires » seront principalement caractérisés par les éléments suivants :

- un titre,
- un résumé court,
- des mots clés,
- un thème,
- le nom de l'auteur,
- la date.

L'ensemble des fonctions du serveur SEMUSDI proposé aux utilisateurs est représenté dans la figure 1. Des fonctions d'administration du système sont égale-

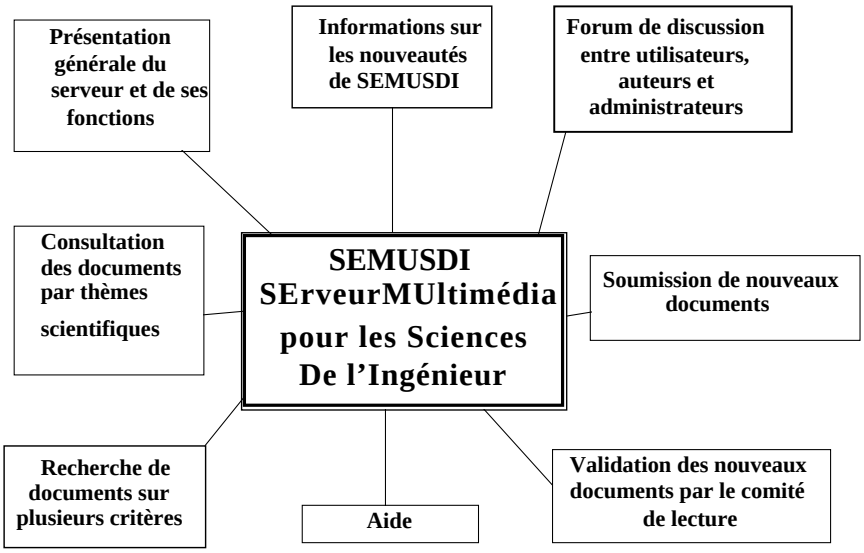

Figure 1 : Principales fonctions de SEMUSDI

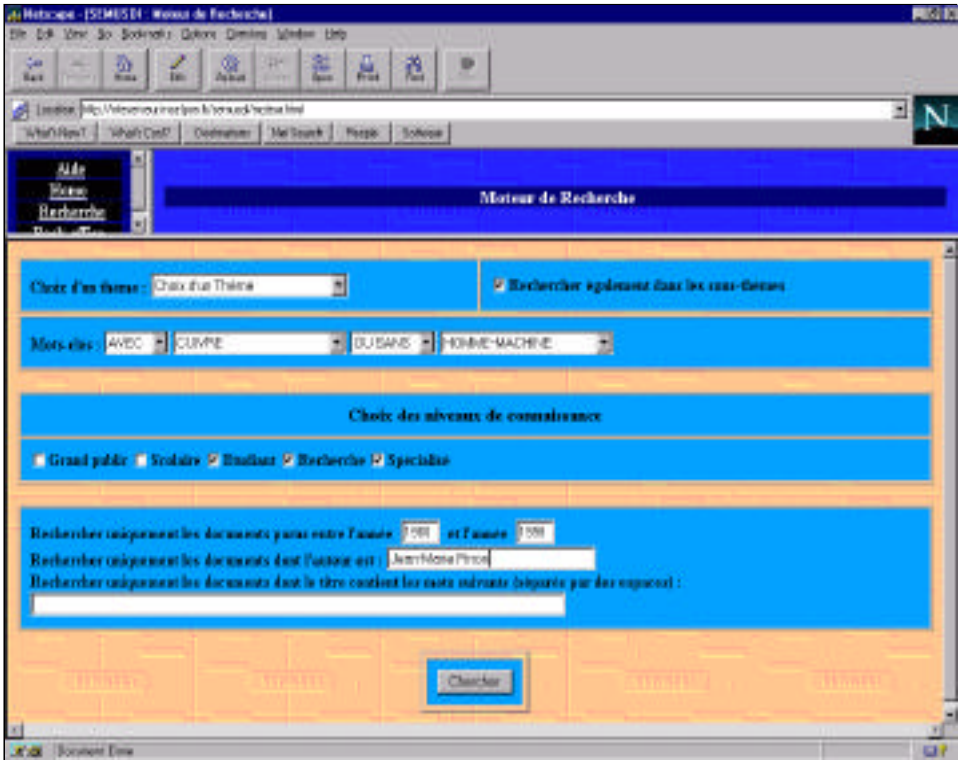

Figure 2 : Recherche multicritère.

ment possibles mais elles ne sont accessibles que par un nombre très restreint de personnes autorisées.

Ce synoptique correspond à la première page présentée par le serveur sur le WEB [HREF1].

Selon la complexité de la fonction sélectionnée, on sera automatiquement dirigé vers d'autres écrans plus ou moins imbriqués.

Présentation de quelques fonctions :

- La fonction de « recherche de documents sur plusieurs critères » obligera l'utilisateur à remplir une grille, où il précisera des critères comme : les mots clés ou (et) auteur, titre, date, thème. On propose également des menus déroulants qui permettent de sélectionner des mots clés ou thèmes parmi une liste. L'utilisateur indiquera également son niveau de connaissance par exemple scolaire, ingénieur, chercheur, etc. (Figure 2).
- La fonction « soumission de nouveaux documents » permet à des auteurs de soumettre de nouvelles « briques » au comité de lecture, dont le rôle est de valider ou non leur insertion dans la base. Cette fonction nécessite la saisie de l'identité et des coordonnées de l'auteur, du titre de la « brique » soumise, du thème, des mots clés, du résumé et de la date, entre autres. Pour accéder à ce type de fonction dite fonction de « Back office », il est nécessaire de posséder un mot de passe.
- La fonction « consultation par thème » permet à l'utilisateur de découvrir les thèmes du serveur et propose ainsi une consultation plus large que la consultation multicritère. Elle permet de faire évoluer l'utilisateur dans sa démarche de recherche.

Chaque fenêtre respecte les mêmes règles de présentation pour le confort et la facilité d'utilisation, un exemple est présenté en figure 2.

Architecture générale du système.

Nous venons de voir que le principal atout du serveur SEMUSDI est de fournir des outils de recherche, de consultation, de proposition et d'obtention de brique élémentaire à différents types d'utilisateur. Naturellement, ces opérations sont aisément effectuées à l'aide d'une base de données. Toutefois les caractéristiques des données multimédias empêchent l'utilisation de SGBD (Systèmes de Gestion de Base de Données) classiques, c'est-à-dire les SGBDR (Relationnels) ; l'emploi d'un SGBDOO (Orienté Objet) est donc ici tout à fait justifié (Rumbaugh 1994).

Dans ce chapitre, nous allons tout d'abord présenter les avantages que procure l'utilisation d'une base de données orientée objet puis nous citerons les classes définies dans le serveur SEMUSDI. Nous verrons ensuite les avantages et les inconvénients résultant de l'utilisation

de scripts CGI (Common Gateway Interface) pour gérer une base de données via Internet (Bergel 1996). Puis nous analyserons les apports du langage Javascript (Duan et Atlantic 1996). Enfin nous examinerons une nouvelle architecture du système qui est en ce moment à l'étude.

Utilisation du SGBDOO MATISSE.

Comme nous l'avons vu dans la première partie de cet article, SEMUSDI doit permettre de sauvegarder des données multimédia. Or, le stockage de ce type de données pose certains problèmes, principalement dus à l'hétérogénéité des formats et à la taille des fichiers : on trouve sur le marché des dizaines de formats pour stocker des images, des vidéos, des simulations, etc., de plus, la taille de chaque média peut varier de quelques kilo-octets (par exemple pour du texte ou des images) à plusieurs centaines, voire plusieurs méga-octets Chaque type de média peut exiger des traitements particuliers, par exemple, dans notre cas, chaque média doit pouvoir donner son type MIME, afin que les navigateurs puissent l'interpréter convenablement. Dès lors, ces particularités nous obligent à utiliser un SGBDOO [HREF7]. En effet, seuls ces systèmes permettent de stocker des données hétéroclites, de grandes tailles tout en leur assignant des traitements spécifiques (Lange 1993. Après l'étude de plusieurs SGBDOO (Kueng 1994) (Hadjiefthymiades et Martakos 1996), notre choix s'est porté sur le SGBDOO peu connu qu'est MATISSE [HREF2]. Ce SGBDOO possède en effet plusieurs avantages, tels que son architecture client-serveur, son caractère multi-plateforme, les nombreuses API (Application Programing Interface) disponibles ou encore ses capacités de stockage.

De plus, ce SGBDOO est totalement objet, c'est-à-dire que tous les éléments usuels d'un SGBD sont définis sous forme de classes (relations, index, attributs, méthodes, etc.) et de métaclasses (Figure 3), permettant ainsi d'ajouter ou de redéfinir facilement des attributs ou des méthodes aux classes (Muller 1997).

Par exemple, nous avons redéfini une métaclasse, que nous avons appelée *MtDocument* (sous classe de la classe *MtClass*, qui est la métaclasse par défaut), annexant à ses instances, c'est-à-dire les classes qui vont permettre de stocker des média, de nouveaux attributs. Le premier, nommé *AtPresentationClasse*, permet de décrire clairement le type de données que peuvent stocker les instances de la classe (par exemple « Fichier HTML ») : cela permet, via une simple requête, de connaître l'ensemble des types de documents acceptés par SEMUSDI. Le deuxième, nommé *AtTypeMime*, permet de stocker au sein de la classe le type MIME de chacune de ses instances (par exemple « image/jpeg » pour les objets dont le contenu est une image au format JPEG) : cela permet de ne pas dupliquer inutilement la même information. De même, nous avons eu la possibilité d'ajouter quelques méthodes à certaines classes.

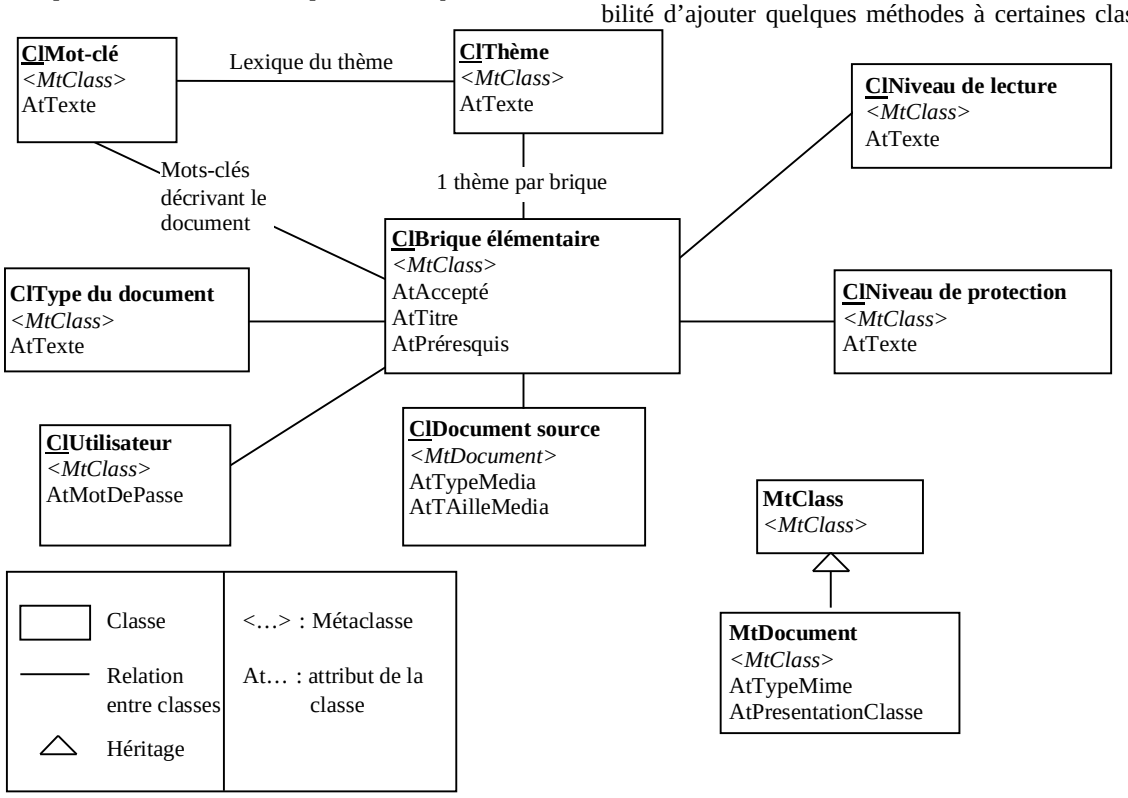

Figure 3 : Principales classes et métaclasses.

Par exemple la classe permettant de stocker des fichiers au format HTML, possède deux méthodes : l'une permettant de détecter l'ensemble des objets (images, sons, «applets», etc.) référencés dans la page et l'autre permettant de supprimer tous les liens extérieurs.

La classe *ClBriqueElementaire* constitue une classe essentielle, elle comporte un nombre important d'attributs, quelques-uns seulement sont indiqués dans la figure 3. Cette classe est en relation avec presque toutes les autres classes du serveur.

Interface CGI-MATISSE.

Cette composante de notre serveur est assez classique. En effet, aujourd'hui, la manière la plus courante de recueillir des informations provenant d'un client et de les traiter (c'est-à-dire dans notre cas d'exploiter la base de données) est d'utiliser des scripts CGI [HREF5], (Gleeson et Westaway 1995). Ces scripts sont en fait des programmes qui récupèrent de l'information par l'intermédiaire de formulaires HTML et qui exécutent des traitements spécifiques. Dans notre cas, les scripts CGI sont écrits en langage C (Goodman 1996). On en dénombre principalement six, définis de la façon suivante :

AfficheRecherche permet de communiquer au client les résultats d'une recherche,

AfficheTheme permet de communiquer les informations nécessaires à la construction de la page de recherche,

SaisieBrique permet de communiquer les informations nécessaires à la construction de la page d'ajout de brique,

AjoutBrique permet d'ajouter une brique élémentaire dans la base de données,

TéléchargerBrique permet au client de récupérer une brique,

AfficherDernièresBriques permet de communiquer aux clients des informations sur les n dernières briques.

Bien que les scripts CGI présentent de nombreux avantages comme la facilité de développement et de test, la connexion du SGBD au poste client n'est pas optimale (Calabretto et Rumpler 1997). En effet, la liaison avec le client n'est pas permanente, c'est-à-dire qu'il y a autant d'exécution du script CGI, qu'il y a de requêtes. De plus, une fois le résultat renvoyé, il y a destruction du script CGI. Cela oblige donc le serveur à exécuter autant de CGI qu'il y aura de connexions. Par conséquent, le suivi de l'utilisateur est très difficile à réaliser. Ces dernières années, plusieurs solutions ont été proposées, par exemple les scripts FastCGI [HREF8], sans toutefois répondre totalement aux besoins des développeurs. Nous verrons cependant par la suite une architecture innovante qui permet de résoudre

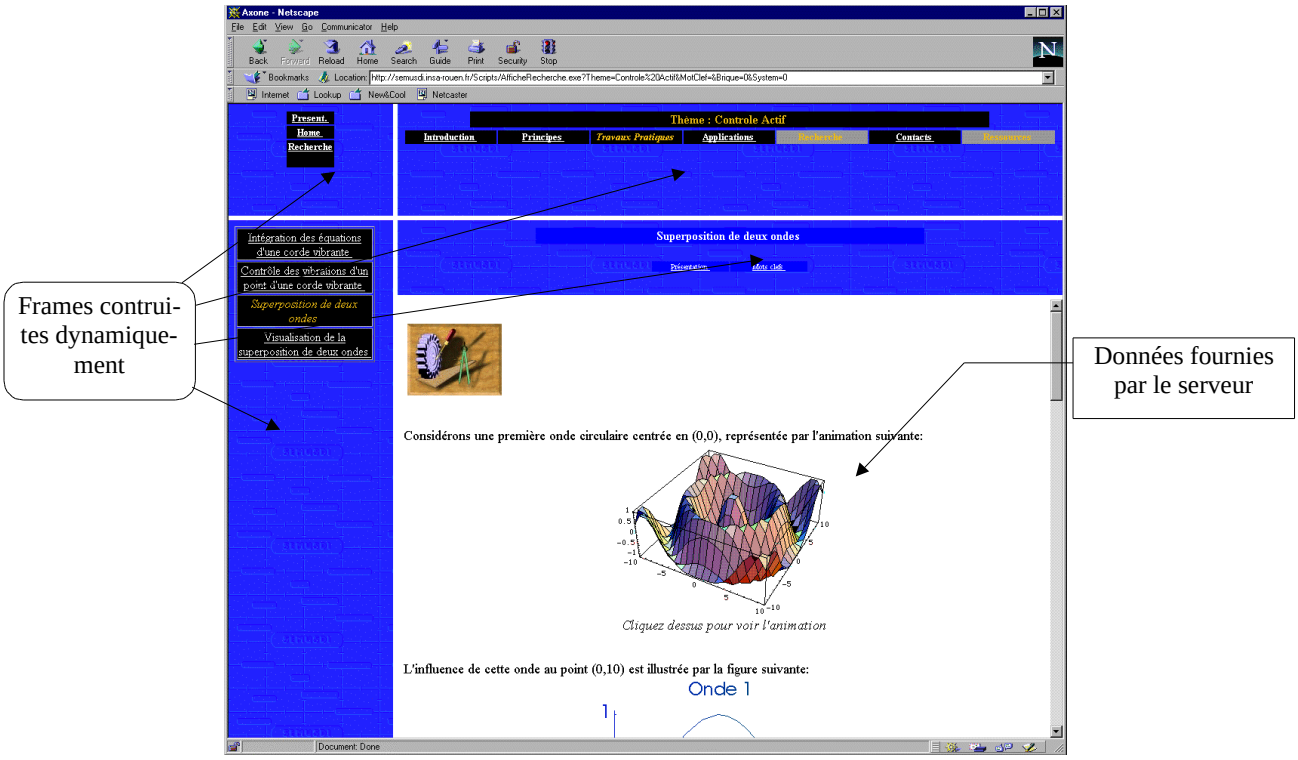

Figure 4 : Visualisation d'une brique élémentaire.

ces problèmes.

Utilisation de Javascript.

Cette dernière composante permet d'améliorer l'interactivité et les vitesses de chargement, et d'augmenter l'homogénéité du site SEMUSDI. En effet, en règle générale, ce sont les programmes CGI qui mettent en forme les pages HTML que peut visualiser le client. Sur notre serveur, ce travail est effectué au niveau du poste client, déchargeant dès lors plus rapidement le serveur : il a donc, le cas échéant, la possibilité d'effectuer d'autres tâches. Cette délégation de tâche est réalisée grâce à la transmission, parallèlement aux résultats des programmes CGI, de librairies Javascript [HREF3]. Le navigateur du client reçoit donc, d'une part les résultats du script CGI sous la forme d'une page HTML simple incluant les données sous forme de tableau Javascript, et d'autre part des librairies Javascript contenant des fonctions de construction de page et de gestion d'interaction homme machine. Ainsi, seules les données essentielles sont transmises constamment, d'où un gain dans la vitesse de transmission de l'information. Ce processus est particulièrement sensible lors de la visualisation d'une brique. Comme le montre la figure 4, l'ensemble des « frames » qui composent cette page de visualisation est construit chez le client. Ainsi les modifications visuelles de l'interface, issues des actions de l'utilisateur, sont effectuées au niveau du poste client, le serveur intervenant seulement lors de la transmission de la nouvelle brique à visualiser.

L'homogénéité du serveur est de ce fait grandement améliorée puisque pratiquement toutes les pages du serveur utilisent les mêmes fonctions Javascript. Ainsi, si nous devions modifier l'aspect visuel de nos pages, voire modifier totalement l'organisation de ces dernières, nous n'aurions alors «qu'à» modifier les fonctions Javascript.

Toutefois, l'utilisation de ces librairies Javascript nous a posé quelques problèmes. En effet, les navigateurs Internet sont légions, et seulement quelques-uns intègrent des interpréteurs Javascript. Nous avons donc dû nous restreindre aux deux navigateurs les plus utilisés que sont Netscape Navigator (version supérieure à la 3.03) et Internet Explorer (version 4.0). Malheureusement, la concurrence entre ces deux produits empêche l'obtention d'une compatibilité parfaite entre le Javascript des navigateurs Netscape et le Jscript [HREF6] d'Internet Explorer. Ainsi, deux alternatives sont étudiées en ce moment. Tout d'abord, il est possible d'envoyer au client la librairie adaptée, contenant soit des fonctions Javascript soit des fonctions Jscript, ce choix est effectué en fonction du navigateur et du système d'exploitation que possède l'utilisateur. Cette solution peut être totalement transparente pour l'utilisateur mais nécessite un travail de développement et de test assez conséquent. Ensuite, comme nous allons le voir dans le paragraphe suivant, nous étudions en parallèle une évolution architecturale de notre serveur en essayant de minimiser l'utilisation du Javascript en utilisant conjointement une ou plusieurs « applets » Java.

Pour finir, l'architecture actuelle de notre serveur peut être résumée par le schéma ci-dessus (Figure 5). Par expérience, nous pouvons affirmer que ce type d'architecture permet d'obtenir un serveur Internet interactif, rapide et uniforme.

Utilisation possible des «applets» Java.

Les problèmes que nous avons rencontrés nous ont amenés à commencer une étude impliquant l'utilisation d' «applets» Java, [HREF4] (Duan et Atlantic 1996), afin de minimiser l'utilisation des librairies Javascript. Cette nouvelle architecture peut être résumée par le schéma de la figure 6.

Cette architecture permet d'éliminer les problèmes exposés précédemment. Tout d'abord la disparition des scripts CGI permet de stabiliser le serveur et d'améliorer le suivi de l'utilisateur (avec éventuellement une adaptation du système en fonction des caractéristiques ou des actions de l'utilisateur). A ce niveau, deux choix s'offrent à nous :

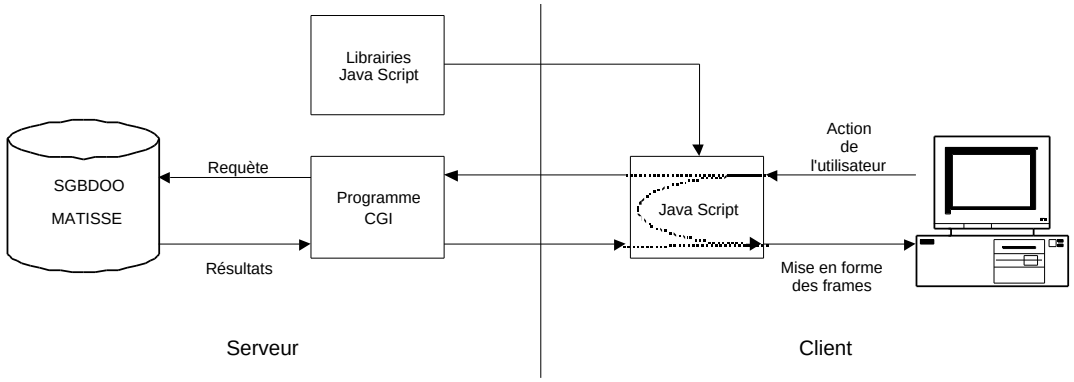

Figure 5 : Architecture générale de SEMUSDI

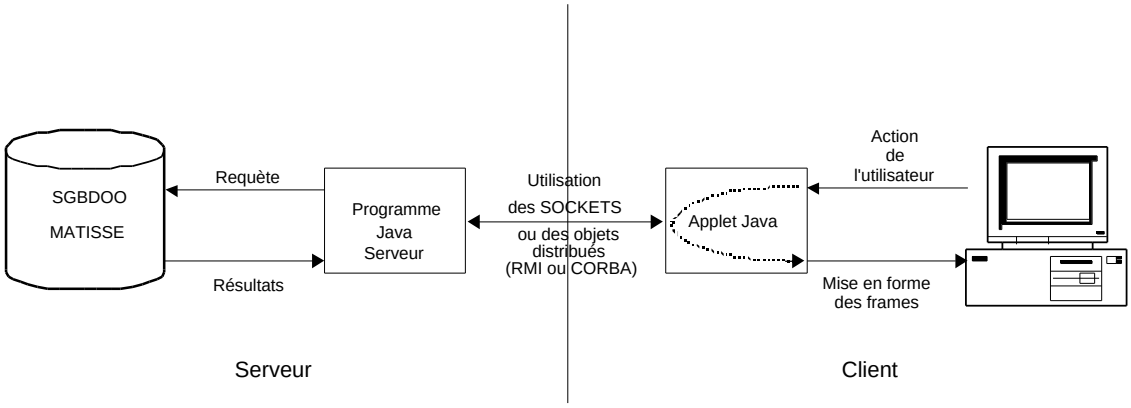

Figure 6 : Nouvelle architecture étudiée

- Nous pouvons tout d'abord utiliser les sockets (Niemeyer et Peck 1996) permettant ainsi faire de transiter les données du client vers le serveur (les requêtes), ou du serveur vers le client (les résultats des requêtes). Cette architecture, bien que ressemblant à la précédente, est plus stable. En effet, chaque nouvelle connexion entraîne la création d'un processus «fils» du programme Java qui n'est détruit qu'au moment de la déconnexion du client, c'est-à-dire au moment de la destruction de « l'applet » exécutée sur le poste client. La communication entre le serveur et le client est donc continue et non plus discrète.
- L'autre solution est de ne plus faire transiter les données mais plutôt de les partager, de les distribuer. CORBA[1] (Siegel 1996) permet de distribuer des objets entre plusieurs applications. Mais JAVA propose son propre protocole de distribution d'objet, RMI[2], beaucoup plus simple à mettre en œuvre. En simplifiant, RMI permet d'activer les méthodes d'objets distants. Dans notre cas, cela permet à l'applet d'utiliser des objets se trouvant sur le serveur.

Pour finir, l'utilisation d'«applet» Java pour le cheminement de l'utilisateur dans le site et la création des pages HTML permet d'éviter ou du moins de réduire l'utilisation du Javascript, minimisant dès lors les problèmes exposés précedemment.

Toutefois cette proposition n'est pas exempte de tout défaut. En effet, les normes Java sont en ce moment en constante évolution et les navigateurs n'acceptent pas tous la même version du langage Java. De plus, une utilisation brève du Web permet de se rendre rapidement compte que la bande passante est très faible, il faut donc absolument utiliser des «applets» de petite taille pour préserver les performances.

[1] Common Object Request Broker Architecture
[2] Remote Method Invocation.

Conclusion.

Cet article présente l'architecture SEMUSDI, un serveur multimédia pour les sciences de l'ingénieur, SEMUSDI. Ce serveur se distingue des sites déjà existants sur Internet, d'une part par son contenu, caractérisé par des documents techniques très courts et bien ciblés, d'autre part par l'utilisation de techniques et d'outils qui n'ont pas jusqu'à présent été utilisés conjointement. Après moins d'un an d'exploitation, certaines difficultés nous sont apparues, principalement dues à la non normalisation des navigateurs disponibles en ce moment : différentes solutions sont alors à l'étude. Toutefois cette phase d'exploitation a permis de mettre en valeur les qualités intrinsèques de ce serveur, tout particulièrement les qualités d'homogénéité, d'interactivité et de performance du site.

Références bibliographiques.

Bergel, H. 1996. The client's side of World Wide Web. *Communication of the ACM*. 39(1) :30-40.

Calabretto, S. Rumpler, R. 1997. WWW access to a philological workstation. *In: Proceedings of ADBIS'97, ACM SIGMOD St. Petersburg* (Russia), Ed. Nevsky Dialect, ISBN5 7940-004-X, pp. 326-330. Edited by Springer Verlag.

Delestre, N. Gréboval, C. Pécuchet, J.P. 1997. METADYNE, a Dynamic Adaptive Hypermedia for Teaching. *3rd ERCIM Workshop*, pp143-149.

Duan, N.N. Atlantic, B. 1996. Distributed Database Access in a Corporate Environnement Using Java. *Fifth International World Wide Web Conference* Paris, France

Gleeson, M. Westaway, T. 1995. Beyond Hypertext *: Using the WWW for Interactive Applications. AusWeb 95*.

Goodman, D. 1996. Programmer avec JavaScript . *Ed. Sybex, ISBN: 2-7361-2212-7*.

Hadjiefthymiades, S. Martakos, D. 1996. A generic Framework for the Deployment of Structured Databa-

ses on the WWW. *Fifth International World Wide Web Conference Paris, France.*

Kueng, P. 1994. Comparison of ten ooDBMS. Institute of Computer Science, *University of Fribourg, Switzerland*.

Lange, D. 1993. Object-Oriented Hypermodeling of Hypertext Supported Information Systems. *Proceedingd of HICSS'93.*

Muller, P.A. 1997. Modélisation objet avec UML. *Edition Eyrolles*.

Niemeyer, P. Peck, J. 1996. JAVA par la pratique. *Edition O'Reilly International Thomson*.

Rumbaugh, J. 1994. Modélisation et conception orientées objet. *Edition MASSON*.

Siegel, J. 1996. CORBA Fundamentals and Programming. *Wiley computer publishing*.

Références hypertextes.

[HREF1] SEMUSDI *Un Serveur Multimédia pour les Sciences de l'Ingénieur*, http://semusdi.insa-rouen.fr

[HREF2] *Le SGBDOO MATISSE*, http://www. adb.fr

[HREF3] *Spécifications du langage Javascript*, http://home.netscape.com/eng/ mozila/2.0/handbook/javascript/index.html

[HREF4] *Spécification du langage Java*, http://www.sun.com/java

[HREF5] Fonctionnement des scripts CGI, http://hoohoo.ncsa.uiuc.edu/cgi/interface.html

[HREF6] Didacticiel JScript, http://www. microsoft.com/france/jscript/jstutor/jstutor.htm

[HREF7] V. Balasubramanian, State of the Art review on Hypermedia Issues and Applications,. http://cbl.leeds.ac.uk/nikos/tmp/hypemedia/hypemedia.html.

[HREF8] FastCGI specifications, http://www. fastcgi.com