RENPAR'14 / ASF / SYMPA
Hamamet, Tunisie, 10 – 13 avril 2002

Les Documents actifs basés sur une Machine Virtuelle Virtuelle

Gaël Thomas, Bertil Folliot, Ian Piumarta

Université Paris 6, LIP6/CNRS
4, place Jussieu – F-75252 Paris
gael.thomas@lip6.fr, ian.piumarta@inria.fr, bertil.folliot@lip6.fr

Résumé
Les documents numériques et les réseaux permettent d'améliorer la qualité et la diffusion des documents, mais ces progrès entraînent de nouveaux problèmes : multimédia, cohérence entre répliquats, QoS, droits d'auteur... Ces difficultés peuvent être résolues par des normes et des protocoles pour des cas génériques mais ne peuvent pas l'être pour les cas particuliers. Dans cet article, nous présentons les documents actifs qui introduisent du code exécutable dans les documents pour permettre à chaque document de choisir les solutions adéquates à ses besoins.

Mots-clés : Document Actif, Réflexivité, Machine virtuelle virtuelle, Application Active

1. Introduction

Avec l'arrivée du numérique et des échanges massifs de documents, on voit apparaître de nouveaux problèmes lors de la conception et de la réalisation de documents. Que ce soit l'hétérogénéité des documents, l'hétérogénéité des plates-formes de visualisation, les problèmes de qualité de service, la cohérence entre répliquats d'un même document ou encore les problèmes de droits d'auteur, on constate que les solutions actuelles sont insuffisantes. Le type mime (ou la signature d'un document) ne peut pas à elle seule décrire tout un document (à la fois son type, son visualisateur et la version utilisée). Les difficultés liées aux droits d'auteur posées par des protocoles de localisation et d'échanges de documents comme Gnutella [10] sont actuellement sans réponse. Les protocoles de qualité de service comme diffserv ou rsvp [14, 7] ont encore du mal à s'implanter : la qualité de service ne peut pas encore être gérée au niveau système.

Nous nous proposons de présenter une méthode pour résoudre un certain nombre de ces problèmes grâce aux documents actifs : un document actif est un document possédant du méta-code exécutable par la plate-forme hôte du document. Ce code permet au document actif de manipuler son contenu. Avec une telle méthode, le document gère sa cohérence, par exemple, en vérifiant sa validité auprès de sa copie primaire. Il se protège du piratage en s'auto-cryptant et en interrogeant un tiers de confiance pour obtenir une clé de décryptage, le document peut aussi s'adapter à des plates-formes particulières comme un téléphone portable en générant son contenu grâce à une feuille de style (tout comme avec XML [16]), mais peut aussi s'adapter à un mal voyant en changeant ses polices de caractères.

Les réalisations que nous avons effectuées sont au nombre de trois : un framework pour document actif, des post-it dans un document PostScript et un composeur pour ces notes.

Dans la section 2, nous présentons quelques solutions actives déjà proposées. Ensuite, dans la section 3, nous introduisons ce qu'est le projet Machine Virtuelle Virtuelle et quel est le prototype utilisé pour réaliser des documents actifs. Dans la section 4, nous présentons la structure globale donnée aux documents actifs, c'est ici que nous décrivons le framework de document actif. Dans la section 5 nous décrivons un premier exemple de document actif : un PostScript annoté, dans la section 6, nous décrivons un composeur pour ces notes et enfin dans la section 7, nous concluons sur les avantages et inconvénients de la solution proposée.

2. Approche Script, approche agent

La limite entre document passif et actif est difficile à donner : on ne peut que donner un degré d'activité. En effet, une expression comme : <**a HREF=mailto :gael.thomas@lip6.fr**> peut être vue comme

une marque HTML (donc passive) ou comme une commande permettant de lancer un client mail (donc active).

Plusieurs solutions actives ont déjà été proposées. On voit deux tendances se dessiner : l'approche script et l'approche plate-forme agent (ou objet).

L'approche script permet de spécifier via des scripts le comportement d'un document comme dans HTML, XML [16] ou SGML [1, 4]. Les scripts permettent aussi d'étendre l'expressivité d'un type de document, par exemple Mobisaic [9] permet à un document HTML de souscrire à certaines variables d'environnement (dont la position géographique du lecteur et l'heure courante) ce qui permet d'avoir une page HTML qui prend en compte la mobilité de la plate-forme de lecture du document.

Les approches Scripts sont en général moins actives que les approches agents, mais permettent néanmoins de résoudre simplement un certains nombre de problèmes comme la composition de médias ou l'évolution temporelle d'un document (comme SMIL [15] dans un document XML par exemple).

Les solutions agents quant à elles sont une vision plus récente des documents actifs. Les agents semblent très bien adaptés pour rendre les documents plus autonomes et adaptatifs en y introduisant des buts et des méthodes.

Parmi les méthodes agents (ou objets répartis), on trouve Globule [3, 8] construit au dessus de Globe qui permet aux documents de s'auto-répliquer ou le serveur vidéo [13, 2] construit au dessus de Bond qui permet à un document de spécifier à son serveur la QoS dont il a besoin.

Ces solutions sont assez prometteuses car elles font intervenir plusieurs acteurs : des clients d'un document (les lecteurs) et les serveurs du document. De cette façon, le client peut agir sur le serveur.

Notre approche des documents actifs est assez différentes des scripts puisque les documents embarquent avec eux du code exécutable, mais rentre aussi difficilement dans le cadre agent puisque les documents n'ont pas de buts : nos exemples de documents actifs sont des programmes dont les données sont des documents.

Avec cette approche, on peut construire des documents actifs qui sont aussi des agents, à condition d'embarquer la plate-forme agent avec la partie active du document.

3. La machine Virtuelle Virtuelle

La machine virtuelle virtuelle [5, 6] est une plate-forme permettant d'intégrer de nouvelles machines virtuelles à la volée. Le but de ce projet est de répondre aux difficultés posées par :

- l'hétérogénéité des plates-formes (c'est une machine virtuelle),
- l'hétérogénéité des machines virtuelles en permettant une grande inter-opérabilité entre machines virtuelles (appel de méthode d'une machine à l'autre, partage de données entre machines virtuelles),
- l'aspect monolithique (donc difficilement réifiable) des machines virtuelles actuelles (Java, ghostscript etc...),

La Machine Virtuelle Virtuelle est une machine virtuelle (elle donne une abstraction de l'assembleur sous-jacent), mais c'est aussi un système d'exploitation (en particulier, la version de la MVV construite au dessus de l'exo-noyau Think [12] dans le projet Phoenix constitue un système d'exploitation) et une couche middleware (qui donne une abstraction du système sous-jacent, que ce soit Windows, Linux ou Think).

Le prototype utilisé dans le cadre des documents actifs est $\mathbf{uvm}_1$. On peut l'assimiler à une micro-machine virtuelle virtuelle : c'est à la fois une machine virtuelle réifiable et un compilateur interactif. Avec $\mathbf{uvm}_1$, toutes les étapes (de la compilation à l'exécution d'un programme) sont réifiables à l'exécution.

$\mathbf{uvm}_1$ permet aussi de fonctionner comme un éditeur de lien : elle permet de charger de nouvelles bibliothèques partagées (des .so Unix) et de lier les symboles des bibliothèques à des symboles $\mathbf{uvm}_1$.

$\mathbf{uvm}_1$ est donc une plate-forme hautement flexible, très bien adaptée aux documents actifs. En particulier, $\mathbf{uvm}_1$ permet de ne s'intéresser qu'à la partie méta-programmation en déléguant les aspects fonctionnels à d'autres langages. Un document actif n'est autre qu'un programme $\mathbf{uvm}_1$ qui apprend à $\mathbf{uvm}_1$ à charger le contenu du document. Le document actif est l'union du corps actifs (le programme $\mathbf{uvm}_1$) et de ressources (les documents).

4. Architecture globale des documents actifs

4.1. Les différents composants d'un document actif

Un document actif est composé de plusieurs entités :
- des ressources : ce sont les contenus du document actif (les doucments passifs) ;
- une partie active : c'est l'interpréteur actif du document. C'est ici qu'on manipule les ressources ;
- de pré-requis et des pos-requis : ce sont des librairies $\mathbf{uvm}_1$ qui doivent être exécutées avant et après l'exécution de l'interpréteur de documents actifs ;
- des protocoles de transfert : ce sont des protocoles qui permettent de charger les différentes entités d'un document, ces protocoles sont des exécutables $\mathbf{uvm}_1$.

Ce qu'on appelle document actif est le fichier qui s'occupe de décrire ces différentes entités. Un document actif est interprété par le chargeur de document actif : c'est un programme $\mathbf{uvm}_1$ qui s'occupe de charger les différentes entités du document actif et de lancer l'exécution des pre/post requis et de la partie active.
Les ressources sont décrites par une URL standard, un type mime, un identifiant opaque pour le chargeur de document actif (qui permet au document actif d'identifier une ressource) et d'un booléen spécifiant si c'est au chargeur de télécharger en local la ressource. Les exécutables $\mathbf{uvm}_1$ quand à eux sont décrits par une URL.
Un protocole est une interface constituée de 4 fonctions : les fonctions get() (qui permet de charger en local un fichier), sync() (qui ne charge le fichier que si la version distante est plus récente que la version locale), open() et close() qui ouvre et ferme un descripteur de fichier (ou de socket). Les protocoles actuellement implémentés sont au nombre de trois : file, ftp et http.

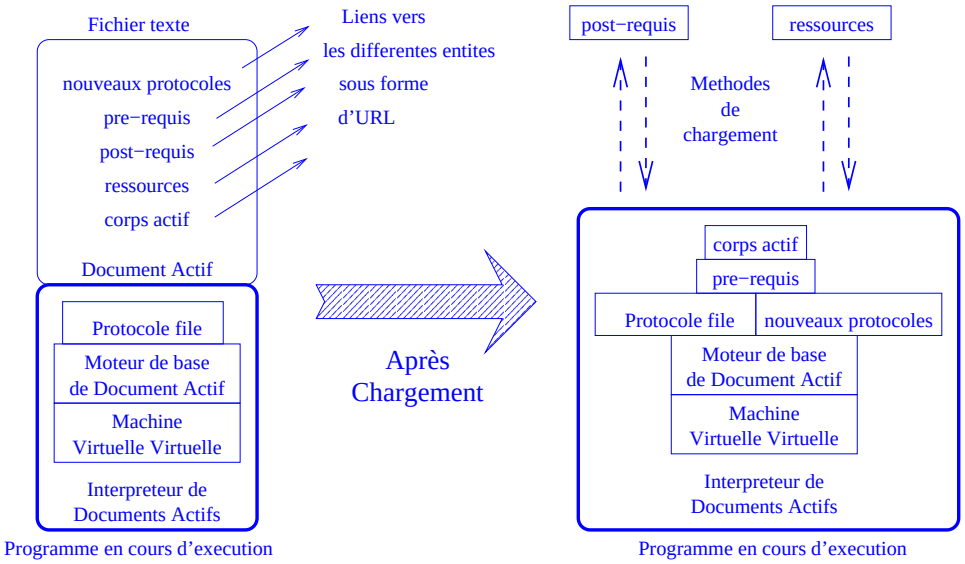

FIG. 1 – Architecture du chargeur de documents actifs

4.2. Le chargeur de documents actifs

Le chargeur s'occupe tout d'abord de synchroniser les protocoles, c'est à dire de charger les protocoles inconnus et les protocoles qui sont plus récents sur les serveurs (i.e. mise à jour automatique). Le seul protocole connu du chargeur au départ est le protocole "file" permettant de charger un fichier à partir des systèmes de fichiers montés en local.
La résolution des symboles des fonctions des protocoles n'est faîte qu'à l'exécution, ce qui permet de pouvoir charger les protocoles à la volée. Les symboles d'appel sont eux-mêmes construits à partir des URL des différentes entités du document.
Le chargeur charge ensuite les ressources nécessaires, exécute les pre-requis, le corps actifs et les post-requis.
Rien n'empêche un document actif de réifier le chargeur de document actif et de l'adapter à ses besoins. Il suffit pour cela de redéfinir les symboles utilisés dans un document actif.

4.3. Plugin Netscape/Mozilla

Pour faire fonctionner les documents actifs dans des pages HTML, on a implémenté un plugin pour les navigateurs Netscape/Mozilla. Ce plugin s'occupe de gérer le type mime **application/x-uvm**. Ce plugin lorsqu'il est lancé par Netscape lance $\mathbf{uvm}_1$ avec les bibliothèques de document actif au départ. L'entrée standard est redirigée vers un tube du plugin et le fichier chargé par Netscape est envoyé à $\mathbf{uvm}_1$ par ce biais. $\mathbf{uvm}_1$ s'occupe alors de compiler/interpréter ce fichier de document actif. Les documents actifs qui n'ont pas besoin d'interface fenêtrée n'utilisent pas cette possibilité.

4.4. Multiple Documents Actifs

Rien n'empêche de créer un document actif dans un autre document actif. Un des pre-requis peut même être un document actif (voir section 6).

Les documents sont placés dans une file d'attente lifo. Le document actif courant est le dernier arrivé. Lorsque l'interprétation du document actif courant est terminé, c'est le précédent de la file qui devient le document actif courant (et qui reprend son exécution).

5. Des PostScripts annotés

Dans cette section nous présentons un premier exemple de documents actifs : les post-it dans les PostScript. Un post-it est une note insérée dans un PostScript. Deux types de notes existent : les notes textes (qui sont insérée dans le PostScript) et les notes ayant un type mime (qui sont exécutées par l'interpréteur adéquat).

Les différents éléments constituant le document actif sont :

- un loader de post-it (en PostScript) qui s'occupe d'insérer la note dans le fichier,
- un fichier uvm2ps (en $\mathbf{uvm}_1$) qui contient toutes les fonctions de traduction de notes de $\mathbf{uvm}_1$ vers PostScript,
- le fichier d'origine (en PostScript) sur lequel on pose les notes, ce fichier n'est pas modifié par l'ajout des notes ;

La partie active se situe directement dans le fichier de document actif. C'est ici qu'on spécifie de manière (simple) en $\mathbf{uvm}_1$ quelles sont les notes.

On peut exécuter ce document actif de deux manières différentes :

- soit mixer le PostScript d'origine et les notes textes à l'exécution et afficher le résultat sur la sortie standard, ce qui rend le nouveau PostScript portable,
- soit exécuter le PostScript : avec ghostscript (qui n'analyse pas les notes mimes) ou avec une surcouche logicielle qui permet d'encapsuler ghostscript (voir section 6) et qui permet de traiter les types mimes ;

L'exécution des notes se déroule en trois étapes :

- accumulation des notes par $\mathbf{uvm}_1$ via les fonctions d'ajout,
- compilation des notes $\mathbf{uvm}_1$ en notes PostScript,
- lancement de ghostscript, de la surcouche de ghostscript ou affichage sur la sortie standard du PostScript résultat ;

L'exécution PostScript se déroule normalement sauf que le symbole **showpage** est redéfini. Ce nouveau **showpage** insère les notes de la page courante et appelle l'ancien **showpage**. La fonction **showpage** est appelée automatiquement par ghostview à chaque fin de page : c'est le symbole PostScript qui permet d'afficher une page. Les notes textes sont accumulées dans une liste PostScript grâce à des fonctions du fichier uvm2ps.

Cet exemple montre bien l'intérêt de l'activité dans un document : comme c'est le document lui-même qui spécifie sa méthode de visualisation, il peut demander des fonctionnalités exotiques à ghostscript sans que l'utilisateur ait besoin de connaître la norme PostScript.

6. Le composeur de notes

Ce deuxième exemple montre comment mélanger des langages de différentes natures dans les documents actifs. En général, chaque traitement possède un langage bien adapté. Il est donc cohérent d'utiliser des langages spécifiques pour ces traitements. Les problèmes dus au mélange de ses langages va être résolu grâce au méta-code des documents actifs.

Ce document actif est un composeur de notes graphique qui va utiliser les possibilités du plugin Netscape. Ce composeur de note permet à l'utilisateur d'un document PostScript (ou annoté) d'y ajouter des notes. Ce composeur de note peut être utilisé par exemple dans le cadre d'un work-flow : chaque lecteur du document ajoute des notes pour le lecteur suivant.

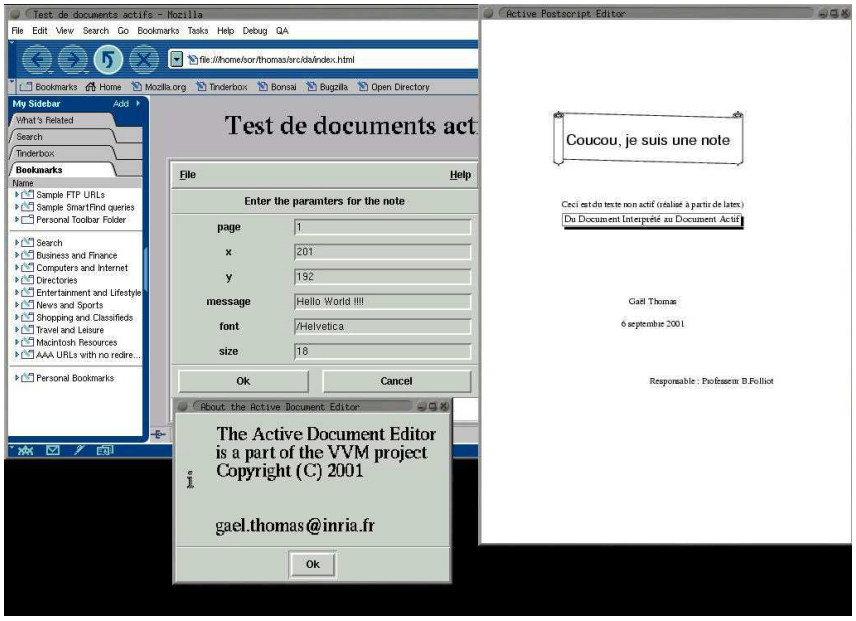

FIG. 2 – Capture de composeur

La figure 2 présente une capture du domposeur de note : on voit à droite le document post-script avec une note et à gauche le plugin netscape permettant de d'ajouter des notes graphiquement dans le document.

6.1. Les langages

Les langages utilisés sont Tcl/Tk (445 lignes) pour la partie graphique, C (490 lignes) pour interfacer ghostview avec $\mathbf{uvm}_1$ ghostscript pour les notes (174 lignes) et HTML (22 lignes) pour en faire un document Web. Le corps en $\mathbf{uvm}_1$ fait, quand à lui, 1100 lignes pour faire les différentes interfaces entre langages et executer le document actif.

On trouve 4 pre-requis pour interfacer les différents langages avec $\mathbf{uvm}_1$ (une par langage). On a aussi besoin de 3 ressources en Tcl/Tk, une ressource en PostScript (celle des notes PostScript) et une en source C pour le mini-viewer. On considère que l'exécutable ghostview est sur la plate-forme destination (on pourrait tout à fait faire un pre-requis qui s'occupe de télécharger et d'installer ghostview le cas échéant).

6.2. Interface Tcl/$\mathbf{uvm}_1$

Comme Tcl/Tk a été développé en C, il existe des bibliothèques Unix pour ce langage. La construction de l'interface entre les deux langages a été assez simple : il suffit d'ouvrir ces bibliothèques et de lier certains symboles avec des symboles $\mathbf{uvm}_1$.

Cette interface est minimaliste. En effet le but n'était pas d'écrire du Tcl/Tk en $\mathbf{uvm}_1$ mais bien de fabriquer une interface entre les deux langages.

6.3. Le mini-viewer PostScript

Toute la documentation concernant Ghostscript a été trouvée dans les sources de ghostview [11]. Dans le mini-viewer, on lance Ghostscript dans une de nos fenêtre et on envoie des blocs de PostScript à ghostscript via son entrée standard (préalablement redirigée vers un de nos tubes).

Le mini-viewer est capable de trouver les fins de pages dans un document PostScript (il suffit de trouver les marques "%%Page : %d %d" dans le source PostScript). Il est écris en C. Trouver ces fins de pages est la seule analyse qui a été faite dans ce projet pour interpréter les PostScript.

Le but du mini-viewer est de pouvoir sélectionner graphiquement une position pour une note et de capturer les événements de saut de page. Ces informations sont transmises à $\mathbf{uvm}_1$.
Le mini-viewer peut être utilisé pour d'autres activités autour des PostScript : on trouve toutes les fonctions nécessaire pour piloter Ghostscript en interne. On ne se sert pas directement de ces fonctions, mais de fonctions dédiées au composeur de notes. Le mini-viewer constitue une interface entre $\mathbf{uvm}_1$ et l'exécutable Ghostscript.
Le mini-viewer est lui même définit dans un document actif qui permet de ne charger que le code source et de compiler ce code sur la plate-forme de destination. Après exécution de ce document actif, les fonctions d'accès à ghostview sont définis en $\mathbf{uvm}_1$.

6.4. Chargement d'un PostScript actif

On peut aussi chargé un PostScript actif (un document actif de note). Pour se faire, on utilise toute la mécanique de chargement des notes (voir premier exemple), mais on modifie la fonction d'exécution des post-it (après chargement) de manière à ne pas exécuter le document actif mais da manière à remplir les structures de description du document actif.
Cette méthode permet de réutiliser la mécanique mise en place pour les notes : le document PostScript de base est bien téléchargé en local et les notes sont bien représentée dans la liste de notes $\mathbf{uvm}_1$. On voit là un exemple de document actif qui réifie son chargeur de document actif.

7. Conclusion et perspectives

La solution proposée convient très bien à des besoins spécifiques d'un document, mais ne permet pas de résoudre de manière globale les problèmes. En effet si de telles solutions existaient, il suffirait de modifier les interpréteurs. Le coût de développement reste lourd puisqu'à chaque type de document et à chaque besoin d'activité il faut écrire la partie active du document actif. Cette partie peut être légère (comme dans le cas des notes) mais demande d'avoir une très bonne connaissance du type de document traité. Par rapport à des technologies comme XML, le coût de développement d'un document actif semble assez similaire : construire un nouveau type de document actif revient à écrire l'équivalent d'un DTD et d'une feuille de style.
Le framework de document actif est quant à lui suffisamment ouvert pour permettre de rendre actif n'importe quel type de document : il n'est fait aucune hypothèse à ce niveau sur les caractéristiques du document passif (en particulier, le document passif peut être une application ou un document actif).
Les deux exemples illustrent bien les possibilités de cette approche : les documents embarquent avec eux les outils permettant d'adapter leur plate-forme de visualisation à leurs besoins. Seuls les problèmes liés à l'adaptation du document et de la plate-forme, à la gestion des répliquats d'un document et à la composition de différents types de documents ont été abordés dans ces exemples, mais l'approche document actif permet aussi de résoudre les autres difficultés évoquées en introduction.
Cette approche offre les avantages de l'approche agent puisque le document actif est un processus à part entière, mais offre aussi les avantages de l'approche script : la partie active du document permet d'adapter le contenu passif du document.
Cependant, aucune approche globale du système n'a encore été réalisée : les interactions entre processus actifs sont inexistantes. Une prochaine étape sur les documents annotés serait par exemple de donner une vision globale du document actif à chaque utilisateur du document (lecteur ou écrivain) en ajoutant dynamiquement les nouvelles notes à tous les répliquats et en donnant des méthodes pour sélectionner les notes visibles pour un utilisateur (en fonction de sa demande et de ses privilèges).
La partie interaction entre langages (et/ou machines virtuelles) semble la plus prometteuse de ce projet. En effet le fait de pouvoir mélanger plusieurs machines virtuelles pour les faire fonctionner en collaboration permet de réduire le temps de développement en utilisant des langages adaptés aux traitements donnés.

Bibliographie

1. A. Eliëns, J. van Ossenbruggen, and B. Schönhage. Animating the web – an sgml-based approach. *In International Conference on 3D and Multimedia on the Internet*, 1996. WWW and Networks British Computer Society.

2. A. Kendel and H. Hoffmann. An object oriented framework for building collaborative network agents in intelligent systems and interfaces. *Kluewer Publishing House*, pages 31–64, 2000.
3. A. Tannenbaum, M. Van Steen, and P. Homburg. Globe : A wide-area distributed system. *IEEE'99*, pages 70–78, janv-mars 1999.
4. A.Eliëns, J. van Ossenbruggen, and B. Schönhage. Web applications and sgml. *In Sixth International Conference on Electronic Publishing*, 1996. Document Manipulation and Typography.
5. B. Folliot, I. Piumarta, and F. Riccardi. Virtual virtual machines. *Cabernet Radical Workshop*, 1997.
6. B. Folliot, I. Piumarta, and F. Riccardi. A dynamically configurable, multi-language execution platform. *Proc. of 8th ACM SIGOPS European Workshop,*, 1998.
7. Differentiated services. http ://www.gta.ufrj.br/diffserv/.
8. G. Pierre and M. van Steen. Globule : a platform for self-replicating web documents. *Proc. 7th annual ASCI Conference*, may 2001.
9. G. Voelker and B. Bershad. Mobisaic : An information system for a mobile wireless computing environment. *In Mobile Computing Workshop*, septembre 1994.
10. The gnutella protocol specification v0.4. www.clip2.com/gnutellaprotocol104.pdf.
11. Interface de ghostscript. http ://siag.nu/latest-src/gvu/gs.interface.
12. J.P. Fassino and J.B. Stéfani. Think : un noyau d'infra-structire répartie adaptable. *CFSE'02*, Avril 2001.
13. K. Jun, L. Boloni, and D. Yau. Intelligent qos support for an adaptive video server. *IRMA 2000*, Octobre 2000.
14. Rsvp. http ://www.isi.edu/div7/rsvp/rsvp.html.
15. Smil. http ://www.w3.org/AudioVideo/.
16. Xml home page. www.w3c.org/XML/.