# Web-R : pour la mémoire exhaustive de ma Toile

Alain Lifchitz, Jean-Daniel Kant

# *Web-R* : pour la mémoire exhaustive de ma Toile

A. LIFCHITZ et J.D. KANT

*CNRS - LIP6 (Laboratoire d'Informatique - Université Paris 6),*
*8, rue du Capitaine Scott,*
*75015 Paris, FRANCE*
Mails : alain.lifchitz@lip6.fr , jean-daniel.kant@lip6.fr
Téls. : +33 1 44 27 43 32, +33 1 44 27 88 05, Fax : +33 1 44 27 70 00

**Résumé**

Dans cet article, nous présentons le système *Web-R*, un outil non-intrusif et rapide d'enregistrement qui réalise un stockage complet et systématique des pages web de l'utilisateur et de sa navigation. Il sauvegarde tous les composants qui seront nécessaires et suffisants à restituer hors-ligne la page de la même façon qu'elle l'a été vue en-ligne. Nous montrons que cet enregistrement systématique du Web personnel est non seulement techniquement réalisable, mais aussi réaliste car ne demandant qu'une faible fraction du volume de stockage des disques durs actuels. De surcroît, *Web-R* fournit un mécanisme de gestion de cet espace de stockage local incluant la comparaison du contenu de pages, évitant une inutile redondance. Enfin, puisque l'ensemble du contenu des pages visitées est stocké, *Web-R* permet à l'utilisateur d'avoir une vue globale de sa navigation personnelle par des statistiques, des outils de tri et de filtrage, pour sa réutilisation et plus tard sa structuration semi-automatique.

**Abstract**

In this paper, we present the *Web-R* system, a non-intrusive and fast recording tool that performs a systematic and complete storage of user's web pages and navigation. It saves all the components that are necessary and sufficient to visualize offline the page the same way it was displayed online. We show that a systematic storage of the personal web is non only technically feasible but also realistic since it requires only a small fraction of the storage space available in current hard disks. Moreover, *Web-R* also provides a way to manage this storage space and integrates a page comparison mechanism to avoid unnecessary redundancy. Finally, since the full content of visited pages is stored, *Web-R* allows the user to have a global view on his/her personal navigation by incorporating statistics, sorting and filtering tools and later on its semi-automated structuring.

## 1 Introduction

### 1.1 Problématique

L'expansion croissante du *World Wide Web* (WWW) a conduit récemment à une forte demande pour une interaction plus spécifique et plus personnalisée. Nous ne sommes plus

confrontés à la difficulté d'accéder à l'information en quantité, mais plutôt d'être capable d'atteindre l'information pertinente basée sur nos buts et besoins du moment. Pour faire face à cette demande, et pour éviter à l'utilisateur d'être submergé et perdu sur le WWW, un important effort a été consenti pour concevoir des applications comme les moteurs de recherche *(e.g.* [*Google*]), les services de catégorisation (*e.g.* [*Yahoo*]) ou plus généralement les outils d'aide à la navigation (*e.g. WebScout* [Milic-Frayling et al., 2002]). Pour pouvoir répondre aux rapides changements du contenu du WWW, ces applications habituellement construisent un cache local ou un entrepôt de pages web qui contribue à maintenir la cohérence des traitements suivants. Ainsi, dans chacune de ces situations, nous avons besoin d'outils spécifiques pour enregistrer les pages web qui alimenteront un traitement de plus haut niveau.

Ce travail est une partie d'un projet récent plus large, le projet [*SpiderMem*], actuellement en développement. *SpiderMem* adopte délibérément une approche centrée sur l'utilisateur, qui l'aide à concevoir et structurer son propre *Web*. Ceci signifie que ce qui peut être organisé est restreint aux seuls sites déjà visités, de telle façon qu'une cartographie de l'ensemble, ou d'une partie seulement de la navigation passée, puisse être dessinée. De plus, une gestion interactive (assistée / automatisée) de l'information, des connaissances, dont les signets, est proposée. Au cœur de l'application *SpiderMem*, l'outil *Web-R* permet à l'utilisateur d'enregistrer et de rejouer toutes séquences de ses navigations passées.

## 1.2 Les outils d'enregistrement

Cette tâche d'enregistrement semble à première vue presque triviale… mais ne l'est pas en réalité. Il y a de nombreuses familles d'outils d'enregistrement et de nombreux outils appartiennent à plus d'une famille. Pour pouvoir mieux situer la problématique de cet article, nous allons effectuer un tour rapide des outils d'enregistrement du *Web*.

Le « *Suivi d'URL* » est le mécanisme qui sous-tend les fonctionnalités de signets (favoris) et d'historiques des navigateurs actuels. Cela signifie que dans chacun de ces cas, la seule information enregistrée est l'adresse de la page, complétée le plus souvent par l'horodatage du chargement. Par exemple, c'est le choix de conception fait dans l'outil d'aide à la navigation [*Internet Cartographer*], bien qu'il analyse au vol (au chargement) les mots-clés contenus dans les métadata de la page web. L'URL (*Uniform Resource Locator*) est une information pertinente et compacte, aisée à stocker, mais une page web est parfois fugace, et quand nous marquons son URL grâce au signet, nous ne pouvons assurer, spécialement en raison de la nature dynamique croissante du *Web* vivant, que cette page sera disponible dans le futur et/ou que son contenu sera identique à ce qu'il est en cet instant. Pour prendre en compte pleinement cette situation notre outil doit non seulement stocker l'URL mais aussi l'intégralité du contenu d'une page ou plus exactement de ses composants.

Chaque navigateur web possède une fonctionnalité « *Enregistrer sous…* » qui devrait réaliser cette tâche. Cependant la sauvegarde des pages doit être automatisée, car bien sûr il est trop fastidieux pour l'utilisateur de cliquer sur « *Enregistrer sous…* » chaque fois qu'il parcourt une page particulière... De plus, la nécessaire redirection vers les fichiers locaux pour la commutation du contexte en-ligne / hors-ligne est rarement complètement et correctement réalisée. Par exemple, quand nous essayons de sauvegarder une page qui inclut une animation *Flash Macromedia®*, avec la version la plus récente (6) de *Microsoft Internet Explorer®* - après avoir pris la précaution de vider les caches mémoire et disque – nous obtenons un rectangle blanc frustrant à la place de l'animation attendue lorsque nous affichons la page enregistrée sans connexion Internet. Cela est dû au fait que des hyperliens pointent encore sans objet vers le *WWW* au lieu d'être des chemins valides vers des fichiers locaux existant effectivement.

Un *butineur* web (*crawler* appelé aussi *webbot* ou *spider*) est un programme configuré pour parcourir et traiter automatiquement, à un moment prédéterminé, les pages, en traversant

systématiquement une topologie web statique. Clairement, ni un *butineur* web personnel (*e.g.* [*WebSPHINX*], [Miller and Bharat, 1998]) ni son pendant global (*e.g.* [*Googlebot*], [*Slurp*]) sont les outils que nous recherchons puisque leur but est de recueillir l'intégralité des fichiers de tout ou partie du *Web* visible. Au contraire, nous souhaitons seulement recueillir les pages visitées par l'utilisateur actuel.

L'*aspirateur* de site (*site ripper*) est utilisé pour dupliquer (aspirer) un site entier sur le disque local. Il s'active à la demande spécifique d'un utilisateur. Certains *aspirateurs* (*e.g.* [*BlackWidow*], [*Wget*]) intègrent des fonctionnalités de visualisation / parcours du web et de gestion. Par rapport aux visées de *Web-R*, il manque à ces outils de l'automatisation et de ne pas être « *centré utilisateur* » (pas d'adaptation, ne fonctionnent qu'à la demande explicite).

Les « *navigateurs hors-ligne* » (*offline browsers*) sont apparus dans les années 1990 pour réduire / rentabiliser la durée de connexion et économiser la bande passante, denrée rare à l'époque. Les versions les plus récentes de ces « *navigateurs hors-ligne* » intègrent d'intéressants outils de parcours et de gestion (*e.g.* [*SurfSaver*]) mais le plus souvent la redirection *WWW* / locale est imparfaite et rarement automatisée.

La dernière famille d'outils d'enregistrement est celle que nous avons dénommée la « famille *Web-R* ». Nous avons découvert deux très récents systèmes, qui ont des caractéristiques proches de celles que nous souhaitons pour la conception de *Web-R* : [*Keepoint*] et *WebScout* [Milic-Frayling et al., 2002]. Comme *Web-R*, *Keepoint* offre le stockage automatique et complet des pages web parcourues au travers d'*Internet Explorer* (*IE*). Il intègre aussi des outils de gestion qui vont au-delà de ce que nous proposons dans la version courante de *Web-R*, puisqu'un utilisateur de *Keepoint* peut notamment annoter les pages stockées et les classer automatiquement par domaines, organisations ou mots-clés. En revanche, son moteur d'enregistrement souffre – du moins dans la version actuelle (1.0) – de deux inconvénients majeurs qui épargnent *Web-R*. D'une part, le mode d'enregistrement automatique réclame encore une intervention systématique de l'utilisateur : celui-ci doit spécifier au système si la page visitée courante, qui partage le même URL qu'une page déjà stockée, doit-être ajoutée à la base de données ou bien substituée à celle-ci. Si l'utilisateur choisi l'automatisation complète, cette décision (duplication ou substitution) sera alors systématiquement appliquée quelle que soit la situation rencontrée pour les pages suivantes avec le risque de redondance ou à l'opposé la perte de pages importantes. La raison profonde de cette difficulté est que *Keepoint*, au contraire de *Web-R*, n'intègre pas un mécanisme de comparaison de pages. D'autre part, le second inconvénient – moins fondamental parce qu'il peut être aisément contourné (cf. 2.3.1) – est que les pages en mode sécurisé (`https://…`) soient incomplètement stockées, ce qui n'est pas le cas de notre système.

Le projet *Microsoft WebScout* semble être très ambitieux et est en fait plus proche de *SpiderMem* que de *Web-R*, puisqu'il vise à intégrer l'annotation de page, le traitement du langage naturel, l'analyse d'image, l'indexation et la recherche d'information textuelle et visuelle, ainsi que la visualisation de données. Jusque-là, trois composants ont été conçus, parmi lesquels le *History Explorer* qui partage les mêmes buts que *Web-R*. Cependant, nous n'avons aucun détail sur cette implantation, et il est donc difficile de la comparer à la solution que nous proposons.

## 1.3 Objectifs

Pour résumer, nous souhaitons réaliser une application compagne du navigateur, intégrant navigation, enregistrement de celle-ci et des contenus, visualisation, et qui puisse fonctionner aussi bien hors-ligne qu'en-ligne. Nous proposons donc de sauvegarder automatiquement l'intégralité des pages que l'utilisateur parcourt pour lui permettre d'organiser à tout moment son *Web* personnel. Pour être concrètement utilisable cette application devra être non-intrusive, rapide et capable de fournir un service de gestion de l'espace pour éviter une éventuelle saturation du disque local. Elle devra aussi fournir un stockage, précis et complet,

des pages web vues par l'utilisateur, en sauvegardant seulement les composants nécessaires et suffisants à la restitution hors-ligne des pages conforme à ce qu'elles étaient en-ligne.

Cet article est organisé comme suit. Nous décrivons les bases principales de conception du système *Web-R*, puis détaillerons son implémentation en section 2. Les travaux à poursuivre dans un futur proche et moyen, en section 3, concluront ce document.

# 2 Conception et implémentation

## 2.1 Communication / contrôle concernant le navigateur

Afin de pouvoir stocker les pages web, systématiquement et automatiquement, pendant que l'utilisateur navigue, nous avons à choisir le mode d'interaction le plus approprié avec le navigateur envisagé. Cette interaction devra, non seulement ne pas consommer trop de temps et autres ressources, mais aussi être suffisamment efficace pour supporter des pages complexes, comme celles comportant du *JavaScript* ou des *frames*. Il y a au moins, de base, quatre façons d'interagir avec un navigateur : Le *proxy*, l'*applet*, le *plugin* et l'API.

On peut, par exemple, prévoir un serveur *proxy* entre le *WWW* et le navigateur de l'utilisateur, fonctionnant comme un cache pour les pages entrantes. Nous rejetons cette architecture « serveur », parce qu'elle est très lourde et peut conduire notamment à des failles concernant la confidentialité des données personnelles.

Les trois solutions restantes sont des architectures « client », bien plus adaptées à l'outil personnel que nous souhaitons concevoir. Ces architectures mettent à profit les fonctionnalités du [*Document Object Model*] (DOM). La première solution « client » est d'utiliser une *applet Java* conjointement avec le navigateur. Dans le système *WebVCR* [Anupam et al., 2000], cette solution s'est montrée efficace pour enregistrer les URLs visités et les informations associées. Une deuxième solution est de réaliser un *plugin* qui prend le contrôle interne du navigateur utilisé. Enfin, une troisième solution « client » est de prendre le contrôle externe du navigateur : sous les environnements *Microsoft Windows*®, nous pouvons utiliser *OLE Automation*® pour contrôler et interagir avec *IE*[1]. C'est la solution que nous avons adoptée pour *Web-R*, puisque nous souhaitions doter cet outil de fenêtres et d'interfaces spécifiques alors qu'un plugin ne peut qu'ouvrir des fenêtres du navigateur sous contrôle.

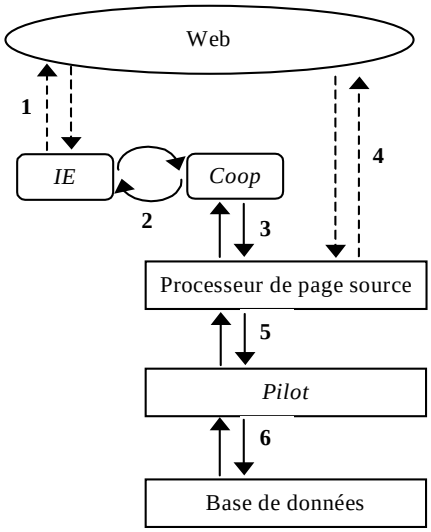
## 2.2 Architecture

*Web-R* est une application qui prend le contrôle de son instance d'*IE* exécutée enfant. Seules des instances qui ont été créées au travers de cette interface racine de *Web-R*, seront sous contrôle. Cette interface utilisateur, de niveau le plus élevé, est représentée Figure 1, ci-dessous.


**Figure 1. L'interface racine de *Web-R* (échelle réelle).**

L'enregistrement des pages est réalisé comme représenté Figure 2 ci-dessous. L'instance d'*IE* obtient la page à partir du *Web* (1), et interprète son code pour affichage. Le chargement de la page par *IE*, est sous le contrôle du module *Coop* qui attend qu'*IE* ait fini son traitement (2). *Coop* récupère le résultat de ce traitement (*i.e.* la page traitée) et l'envoie au module *Processeur de page source* (PPS) (3) qui peut profiter du pré-traitement d'*IE* au travers de DOM.

[1] Nous avons dû choisir pour *Web-R* un développement spécifique à ce navigateur pour deux raisons : 1) *OLE Automation* était disponible et bien documenté ; 2) Il est de loin le navigateur le plus utilisé.




Le but de ce module est de réaliser tout le traitement nécessaire pour s'assurer que le stockage local de la page courante soit « parfait ». Par stockage parfait, nous entendons sauvegarde de toutes les données nécessaires et suffisantes (composants et chemins) pour ré-afficher la page hors-ligne conformément à ce qu'elle était en-ligne. Ainsi, le PPS devra collecter depuis le *Web* (4) les fichiers additionnels nécessaires (*e.g.* images, sons, animations *Macromedia Flash*®, code *Javascript*, etc.). Une fois le traitement de la page réalisé exhaustivement, celle-ci et ses composants sont transmis au module *Pilot* (5) pour le stockage de tous ces fichiers et de leur description dans la *Base de données* (6).

**Figure 2. Organigramme de l'enregistrement des pages.**

Un des points essentiels de la conception réside dans le module *Coop* qui doit gérer les événements *IE*. *IE* active des évènements de la même façon que tout objet *Component Object Model* (COM). Chaque fois que *IE* souhaite fournir une information à ses clients sur son activité courante, il active un évènement par son point de connexion. Pour l'essentiel, le module *Coop* est un *handler* d'événement de *IE*. Il fournit l'information sur la progression du traitement de la page réalisé par *IE*.

## 2.3 Processeur de page source

Du fait du changement de contexte, du *Web* en-ligne au stockage local, les sources de pages doivent être éditées : c'est le rôle attribué au *Processeur de page source*.

### 2.3.1 Chemins de fichiers dans les tags HTML

La fonction essentielle du PPS est de prendre en charge correctement la redirection des noms de fichiers des composants de page, du *Web* en-ligne au disque dur local. Pour pouvoir déterminer où ces liens doivent être modifiés dans le code HTML de la page, il faut intervenir sur les libellés des attributs pertinents en localisant les tags potentiellement concernés, c'est-à-dire ceux qui peuvent comporter un lien vers un fichier externe. Ces tags sont localisés dans le code source en utilisant DOM. Dans sa version courante, *Web-R* prend en charge la redirection locale automatique des couples `<tags> attributs` (HTML 4.0 & 4.01) suivants : `<BODY> BACKGROUND` (URL d'image), `<FRAME>` et `<IFRAME> SRC` (URL de *frame*), `<IMG> SRC` (URL d'image), `<INPUT> SRC` (URL d'image), `<LINK> HREF` (URL de fichier ressource), `<EMBED> SRC` (URL de données de *plug-in*), `<SCRIPT> SRC` (URL de fichier script)[2]. Il faut remarquer que les pages sécurisées sont traitées de la même façon en redirigeant simplement toutes les entrées `"https://…"` du code HTML vers les fichiers locaux appropriés.

### 2.3.2 Frames

Un *frame* inclus dans une page déclenche un traitement spécifique, puisque la page maîtresse est en fait une page composite de pages filles. Cette imbrication de pages filles au sein d'une page maîtresse devra être suivie récursivement sur plusieurs niveaux, puisqu'une page fille d'un *frame* peut elle-même comporter un *frame*, et ainsi de suite. Par conséquent, quand la présence d'un *frame* a été détectée, le PPS démarre le traitement des pages filles de chaque *frame*, et le poursuit pour tous les *frames* englobés, récursivement, jusqu'à ce

[2] Il n'y a aucun obstacle conceptuel à prendre en charge de la même façon le code XML (XHTML, SVG) des pages du *Web* les plus récentes. Cela n'est pas encore réalisé faute de moyens humains/matériels plus importants…

qu'aucun *frame* ne soit plus détecté. Parallèlement, les différents composants sont stockés localement jusqu'à ce que le niveau le plus profond soit atteint. Une des difficultés rencontrées dans le traitement des *frames* provient du mécanisme de sécurité implanté dans *IE*, où l'accès à un *frame* fils, dont le domaine diffère du *frame* père, n'est pas évident.

### 2.3.3 Pages dynamiques

Nous avons conçu le PPS pour qu'il prenne en charge, aussi correctement que possible, les pages dynamiques. Le premier cas de pages locales dynamiques sont les *scripts* notamment *JavaScript*. Quand ces scripts sont des fichiers externes, nous les traitons par une redirection locale décrite en 2.3.1 ci-dessus.

Avec le code embarqué (*embedded*) / externe, deux situations fréquentes sont actuellement prises en charge :

- Le script modifie la page à l'affichage initial. Dans ce cas, *IE* exécutera le script, modifiera la page en conséquence, et alors transformera en commentaire le script pour éviter sa re-exécution intempestive. Par conséquent, aucun traitement additionnel n'est nécessaire puisque nous récupérons la page déjà pré-traitée par *IE*.
- Le script est interactif et s'active lorsque certains évènements se produisent (*e.g. OnMouseOver*, *OnMouseOut*). Il s'exécutera correctement hors-ligne, pour autant que tous les hyperliens soient relatifs à la page maîtresse, et là encore aucun traitement additionnel n'est requis.

Un autre cas de page locale dynamique est l'objet animation (*e.g. Flash Macromedia*®). Il sera traité par une simple redirection de lien vers un fichier local dans le tag `EMBED`, comme décrit dans 2.3.1 ci-dessus.

Le cas des pages dynamiques coté serveur (*e.g.* ASP, PHP, CGI, etc.) ne requièrent aucun traitement additionnel, puisque instanciées par *IE* comme des pages HTML statiques relevant du traitement générique de ces pages.

## 2.4 Stockage

### 2.4.1 Contenu de page

Le contenu de page est stocké dans un sous-répertoire d'un répertoire unique dédié à l'utilisateur courant. Ce sous-répertoire contient toutes les pages qui ont été visitées, par l'utilisateur, ce jour courant. Ce contenu est constitué du fichier source HTML d'origine (celui reçu du *WWW* avant tout traitement, pour mémoire), du fichier source HTML transformé (sortie du PPS), et de tous les composants de page nécessaires et suffisants pour pouvoir rejouer la page hors-ligne de façon fiable (fichiers images, fichiers sons, frames, fichiers embarqués, fichiers scripts, fichiers de style (CSS), etc.).

### 2.4.2 Visite de page

Pour refléter la visite d'une page, des informations additionnelles sont stockées dans une base de données. La version courante de *Web-R* utilise le moteur *Microsoft Jet*® pour cela. Nous stockons pour chaque page, un identifiant numérique unique (*id*), l'emplacement des composants sur le disque local, le nombre d'accès, la durée cumulée de visite, la taille logique totale et la taille physique totale (sur le disque). Pour la dernière visite de chaque page, nous stockons la date et l'heure de début de visite, la durée de celle-ci, la taille et la position de la fenêtre *IE*. Les composants ont aussi chacun un descripteur stocké dans la base, qui comporte l'identifiant de la page à laquelle ils appartiennent, un identifiant de composant, la localisation sur le disque et les tailles logiques / physiques. Pour finir, nous ajoutons un champ de validité (si une erreur s'est produite pendant le chargement de page et dans ce cas le code d'erreur) pour chaque entrée (aussi bien pour la page que pour le composant).

## 2.5 Au-delà de l'enregistrement

### 2.5.1 Redondance (contenus identiques)

Pour éviter l'essentiel de la redondance, nous avons implanté un algorithme simple de comparaison de surface pour déterminer si la page courante est déjà stockée. Cet algorithme se déroule en 3 étapes pour déterminer si la page *P* a besoin d'être stockée :

1. Soit {*P*} un ensemble de pages stockées avec le même URL que *P*. Si {*P*} est vide, stocker *P*;
2. Sinon, rechercher une page *P'*∈{*P*} avec le même code HTML que *P*. Si une telle page n'existe pas, stocker *P*;
3. Sinon, comparer les tailles individuelles respectives des composants homologues dans *P'* et *P*. S'ils diffèrent, stocker *P*; sinon ne pas stocker *P*.

Cet algorithme est rapide et fonctionne dans la très grande majorité des situations concrètes rencontrées dans une navigation web. Une variante maîtrise la prolifération de pages non-identiques, par le contenu, avec le même URL (ce qui peut se produire aisément dans le cas d'accès répétitifs au même site, *e.g.* dans le suivi en temps réel des cours de bourse).

### 2.5.2 Gestion de l'espace de stockage

Puisque toutes les pages visitées seront stockées sur le disque local durant une session *Web-R*, on peut se préoccuper de l'évolution de l'espace occupé. Nous pouvons prévoir que la disponibilité et la banalisation rapide de très grand volume de stockage ainsi que de la puissance de traitement ne pourront qu'aider… mais bien sur ce n'est pas suffisant.

En fait, nous estimons que la taille totale du contenu web personnel ne dépassera pas 5 Go / an pour un utilisateur typique. Des études récentes confortent cette estimation.

Une première expérience conduite par [Huberman et al, 1998] sur un échantillon de 23962 utilisateurs AOL donne un total de 3247054 pages visitées qui conduit à 147 pages visitées / utilisateur / jour. Cela donne un total d'approximativement 50000 pages visitées / utilisateur / an. De plus, l'index [Nielsen//NetRatings, 2002] qui mesure chaque semaine et mensuellement l'audience *Web* sur plusieurs pays montre un maximum de l'ordre de 150 sites visités par mois (*e.g.* pour octobre 2002 : 147 aux USA, 52 en Angleterre, 59 en France, 82 en Allemagne). Le nombre observé de pages visitées par site étant approximativement de 20 (il était de 16 pour l'utilisation privée aux USA en juin 2002, où 735 pages ont été visitées pour un total de 47 sites). Ainsi, les statistiques de l'index Nielsen//NetRatings donnent un maximum de 150x20=3000 pages visitées mensuellement, soit 36000 pages visitées annuellement par utilisateur. Étant donné que la taille moyenne de la page ne doit pas excéder 100 Ko (la troisième *State of Web Survey* [SOW Survey, 1999] rapporte une taille moyenne de la page web de 60 Ko en 1999, images et tous autres composants compris), nous arrivons à la conclusion que la taille maximum web personnel annuel est compris entre 3,6 Go (d'après Nielsen//NetRatings) et 5 Go (d'après l'étude de Huberman *et al.*). Dans tous les cas, cette taille est largement compatible avec les capacités courantes des disques durs actuels et leurs coûts relativement faibles.

Quoi qu'il en soit, nous devons fournir un outil de gestion de l'espace de stockage comme dans tout système de cache. Bien sûr, l'algorithme de maîtrise de la redondance que nous avons présenté en section 2.5.1 ci-dessus ne peut qu'aider. Cependant, l'utilisateur de *Web-R* accède à un mécanisme de limitation de la taille réservée au stockage de la navigation passée, au moyen de l'interface de configuration : Cette limite peut être inexistante, fixe ou un pourcentage de la taille totale du disque local. Quand cette limite est atteinte, un mécanisme de purge est déclenché, qui peut être manuel ou bien automatique.

Dans le cas de la purge automatique, les pages destinées à disparaître sont automatiquement sélectionnées, sur la base d'une combinaison de trois critères : taille physique, date de dernière visite et nombre de visites. Ce calcul de score vise à effacer en

priorité les pages qui auront la plus grande taille, la date de dernière visite la plus ancienne et le moins grand nombre de visites. L'importance accordée à chaque critère est fixée par l'utilisateur pour obtenir le score global. Pour la purge manuelle, l'utilisateur sélectionne, depuis la liste précédente, les pages qu'il choisit d'effacer du disque local.

### 2.5.3 Gestion d'utilisateurs multiples

Bien que *Web-R* soit un outil délibérément personnel, il peut être utile qu'il soit mis à la disposition, successivement, de plusieurs utilisateurs sur la même machine. Pour ce faire, *Web-R* est capable de gérer plusieurs utilisateurs, en limitant le risque d'interférence.

Cependant, on peut souhaiter fusionner les navigations partielles provenant d'utilisateurs différents en une unique base de données. C'est le cas, par exemple, lorsqu'on veut incrémenter une navigation personnelle par celle d'un autre utilisateur, ou reporter sur l'ordinateur du domicile les pages issues de l'ordinateur du bureau. Un autre exemple est celui où l'on souhaite conduire une étude sur l'usage du *Web* : on a un *panel* d'utilisateurs et l'on souhaite collationner toutes les pages de leur navigation.

Pour répondre à ces besoins, nous avons prévu une fonctionnalité d'importation/exportation de navigation. Un utilisateur peut exporter toute sélection de navigation en un unique fichier d'échange. Ce fichier d'échange est une archive compressée (*zip*) qui contient la base de données, tous les répertoires et leurs fichiers de pages visitées. Quand l'utilisateur choisit d'importer un fichier d'échange, sa base de données personnelle est fusionnée avec celle contenue dans le fichier d'échange, les répertoires et les fichiers de pages sont ajoutés à ceux du répertoire de données *Web-R*, et les statistiques mises à jour.

## 2.6 Gestion du *Web* personnel

Dans sa version actuelle, *Web-R* offre aussi un outil de base pour le visionnage et la gestion des pages stockées.

Quand le Gestionnaire / Visionneuse de Pages est activé, il affiche une fenêtre comportant une liste multi-colonnes qui contient les descripteurs de toutes les pages stockées, comme on le voit sur la Figure 3 ci-dessous :

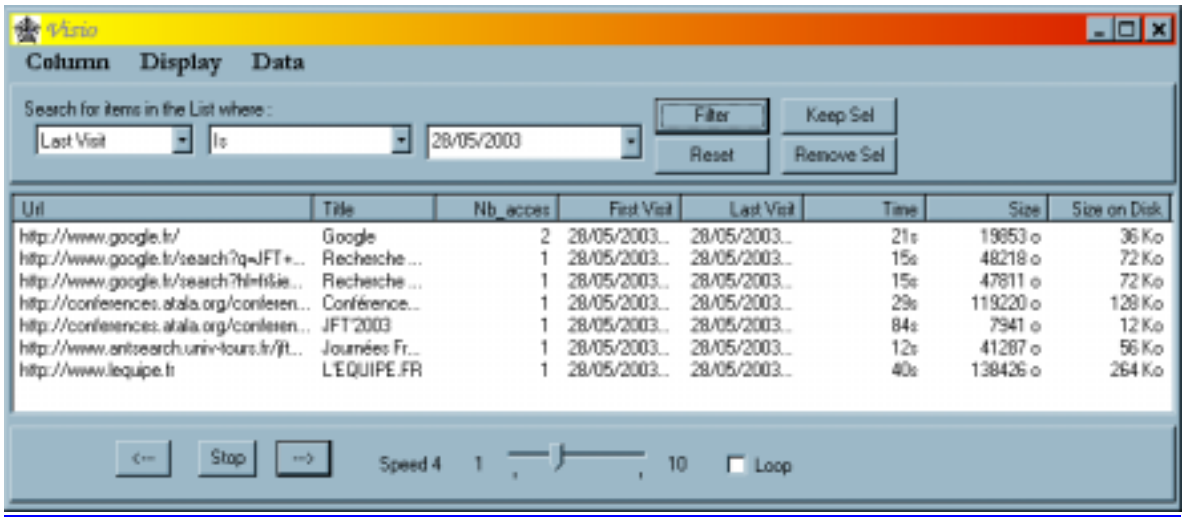



**Figure 3. Visionneuse de pages avec filtre.**

Ces entrées peuvent être classées selon les attributs suivants : URL, titre, nombre d'accès, date et heure de la première visite, date et heure de la dernière visite, taille logique / physique. De plus, en haut de la fenêtre du Gestionnaire / Visionneuse de Pages, un filtre est prévu, pour ne lister que des ensembles de pages satisfaisant certaines combinaisons de valeur des attributs ci-dessus. Par exemple, on peut sélectionner les pages dont l'URL contient la chaîne « 2003 » postérieures à juin 2002. Remarquons que chaque attribut sélectionne

automatiquement son type de filtre (e.g. pour une valeur numérique, le filtre propose : égal / moins que / plus grand que / différent). Cet outil de gestion permet aussi de re-visionner chaque page stockée. Son contenu (hors-ligne) sera affiché quand l'utilisateur double-clique sur un item de la liste, comme on le voit sur la Figure 4 ci-dessous :


**Figure 4. Visualisation hors-ligne d'une page enregistrée.**

# 3 Conclusion et projets futurs

Dans cet article, nous avons montré que le stockage systématique du *Web* personnel est non seulement techniquement réalisable, mais réaliste avec un espace de stockage maîtrisé. Nous avons présenté l'outil *Web-R*[3], un moteur d'enregistrement non-intrusif, rapide et fiable qui peut sauvegarder systématiquement, au cours du temps, le contenu de l'ensemble des pages visitées par un utilisateur. De plus, *Web-R* fournit aussi des informations sur la navigation de l'utilisateur, et intègre des outils pour gérer cette navigation web stockée. Ces pages stockées peuvent être visionnées hors-ligne, en étant reproduites conformément à ce qu'elles étaient, en-ligne, lors de leur visite.

*Web-R* est en cours de développement, ce qui explique certaines de ses limitations[2]. Plusieurs questions techniques doivent cependant être approfondies. A court terme, nous devons consolider le *Processeur de page source* pour le rendre plus robuste. De plus, nous souhaitons faciliter la navigation hors-ligne, en distinguant les liens vers les pages déjà stockées (disque local) et liens vers des pages non-stockées (*WWW*). Dans ce dernier cas, notre système devra demander à l'utilisateur s'il souhaite stocker cette page externe. Nous projetons aussi de faire un meilleur usage des ressources du navigateur, comme l'utilisation de son cache pour éviter de multiples - et donc inutiles - accès aux mêmes composants du

[3] Une version de démonstration de *Web-R* est disponible à l'adresse : http://webia.lip6.fr/~lifchitz/SpiderMem/download
Une version en-ligne de ce document est disponible à l'adresse : http://webia.lip6.fr/~lifchitz/SpiderMem/papers

*Web* en-ligne. Nous devrions aussi utiliser un algorithme de compression pour les fichiers de contenu stockés pour optimiser l'espace de stockage, et encrypter toutes les données pour garantir une totale confidentialité. Enfin, pour améliorer l'ergonomie de notre logiciel, nous envisageons une hybridation des techniques de contrôle d'*IE*, où un plugin et une application externe se partageront le contrôle du navigateur.

Enfin l'aspect validation/expérimentation auprès des utilisateurs ne pourra être considéré que sur une version suffisamment stable et ergonomique de cet outil.

# 4 Remerciements

Les auteurs aimeraient remercier le LIP6 pour son soutien financier à ce projet. Nous remercions aussi Nicolas Limare, Arnaud Joly et Violaine Ruffié pour leur participation au développement de la version préliminaire du logiciel *Web-R*.

# Références

[Anupam et al., 2000] Anupam, V., Freire, J., Kumar, B., and Lieuwen, D. (15-19 May 2000). Automating Web Navigation with the WebVCR. In *Proceedings of WWW9*, Amsterdam (The Netherlands). http://www9.org/w9cdrom/208/208.html

[*BlackWidow*] *BlackWidow*. http://sbl.net/BlackWidow

[*Document Object Model*] *Document Object Model*. http://www.w3.org/DOM/

[*Google*] *Google*. http://google.com/

[*Googlebot*] *Googlebot*. http://www.googlebot.com/bot.html

[Huberman et al, 1998] Huberman, B., Pirolli, P., Pitkow, J., and Lukose, R. (3 April 1998). Strong regularities in World Wide Web Surfing. *Science*, 280(5360): 95–97. http://external.nj.nec.com/~giles/huberman/98.huberman.pdf

[*Internet Cartographer*] *Internet Cartographer*. http://www.inventix.com/

[*Keepoint*] *Keepoint*. http://www.keepoint.com/

[Milic-Frayling et al., 2002] Milic-Frayling, N., Sommerer, R., and Tucker, R. (7-11 May 2002). MS WebScout: Web Navigation Aid and Personal Web History Explorer. In *Proceedings of WWW2002*, Honolulu, (Hawaii, USA). http://www2002.org/CDROM/poster/170/index.html

[Miller and Bharat, 1998] Miller, R., and Bharat, K. (14-18 April 1998). SPHINX: A Framework for Creating Personal, Site-Specific Web Crawlers. In *Proceedings of WWW7*, Brisbane (Australia). In *Computer Network and ISDN Systems* 30: 119-130. http://www7.scu.edu.au/programme/fullpapers/1875/com1875.htm

[Nielsen//NetRatings, 2002] http://www.nielsen-netratings.com/hot_off_the_net.jsp

[*Slurp*] *Slurp*. http://www.inktomi.com/slurp.html

[SOW Survey, 1999] SOW Survey. http://www.pantos.org/atw/35654.html

[*SpiderMem*] *SpiderMem*. http://www-connex.lip6.fr/~lifchitz/SpiderMem

[*SurfSaver*] *SurfSaver*. http://www.surfsaver.com/

[*WebSPHINX*] *WebSPHINX*. http://www-2.cs.cmu.edu/~rcm/websphinx

[*Wget*] *Wget*. http://www.gnu.org/directory/wget.html

[*Yahoo*] *Yahoo*. http://yahoo.com/