Pluxy : un proxy Web dynamiquement extensible

Olivier Dedieu

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Pluxy :
un proxy Web dynamiquement extensible

Olivier Dedieu

No 3417

Mai 1998

THÈME 1

ISSN 0249-6399

Pluxy : un proxy Web dynamiquement extensible

Olivier Dedieu*

Thème 1 — Réseaux et systèmes
Projet SOR — http://www-sor.inria.fr/

Rapport de recherche n ° 3417 — Mai 1998 — 22 pages

Résumé : Pluxy est un proxy Web modulaire capable d'héberger un nombre variable et dynamiquement extensible de services. Il fournit l'infrastructure nécessaire au téléchargement, à l'exécution et à la collaboration de ces services. Pluxy est accompagné d'un jeu de services de base qui permettent aux services présents de participer à plusieurs au traitement d'une même requête HTTP, de disposer d'une interface d'interaction avec l'utilisateur et d'interagir avec des services distants. Trois applications utilisant Pluxy sont présentées : un système d'évaluation collaborative des pages Web, un cache étendu et un outil permettant de suivre l'évolution des ressources Web.

Mots-clé : proxy Web, HTTP, filtrage, composants logiciels, composition dynamique d'application, code mobile, World-Wide Web.

(Abstract: pto)

* Olivier.Dedieu@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : 01 39 63 55 11 - International : +33 1 39 63 55 11
Télécopie : (33) 01 39 63 53 30 - International : +33 1 39 63 53 30

Pluxy:
a dynamically extendable Web proxy

Abstract: Pluxy is a modular Web proxy which can receive a dynamic set of services. Pluxy provides the infrastructure to download services, to execute them and to make them collaborate. Pluxy comes with a set of basic services like collaborative HTTP request processing, GUI management and distributed services. Three Pluxy applications are introduced: a collaborative filtering service for the Web, an extended caching system and a tool to know about document changes.

Key-words: Web proxy, HTTP, filtering, software component, dynamically composed application, mobile code, World-Wide Web.

1 Introduction

Le Web est devenu un outil de communication universel, utilisé par plusieurs millions de personnes. Il a pu faire face à ce succès car ses modèles de désignation (URL), de diffusion (HTTP), d'accès (Hypertexte) et de typage (MIME) des ressources supportent une utilisation à large échelle. Ce modèle est bien adapté pour naviguer dans une masse d'informations hétérogènes. Mais les utilisateurs ont rapidement fait émerger de nouveaux besoins auxquels les chercheurs et les industriels tentent de répondre en fournissant de nouveaux services.

Les documents diffusés sur le Web peuvent évoluer (mise à jour, changement d'URL, suppression). Le modèle original du Web n'offrant pas l'infrastructure pour suivre ces évolutions, des services de suivi et de notification se sont développés. Le browser InternetExplorer propose ce service mais il ne permet pas de faire du suivi pour un groupe d'utilisateurs ni d'être utilisé par d'autres services. À l'opposé, URLMinder [21] propose ce type de service sur un site Web. L'utilisateur s'abonne en indiquant les URL qu'il veut suivre et reçoit les notifications par courrier électronique. Ce service est pratique et permet de fédérer le suivi d'un ensemble d'URL. Cependant, il n'assure pas l'anonymat des utilisateurs et la notification par courrier électronique limite son intégration avec d'autres services. Il serait, par exemple, intéressant de pouvoir précharger dans le cache les documents mis à jour ou de coupler le service de suivi avec un outil capable de mettre en évidence les changements entre deux versions.

Les techniques d'hypertexte permettent de naviguer facilement dans une grande quantité d'informations mais n'offrent pas la possibilité d'obtenir rapidement une information précise. Des sites spécialisés indexent les ressources diffusées sur le Web. À partir de ces index, ils proposent des services de recherche et diffusent des catalogues thématiques. Cependant, les techniques d'indexation employées sont mal adaptées à l'augmentation du volume d'informations et à leur hétérogénéité. Aussi, les utilisateurs ont recours à des outils de communication parallèles tels que les listes de diffusion ou les forums de discussion pour trouver des informations pertinentes. De nouveaux services apparaissent pour fournir une infrastructure à ces nouvelles méthodes de recherche. Ainsi, les techniques de *push* proposent des canaux d'informations ciblées. Mais, il s'agit d'informations fugitives et produites par une seule source. Les outils de filtrage collaboratif [3] proposent des systèmes d'évaluation des documents qui permettent aux utilisateurs de partager leur connaissance du Web. Couplés avec des outils de synthèse d'évaluations, ils peuvent recommander des documents correspondant au profil de l'utilisateur. Cependant, pour être efficaces, ces services doivent être intégrés aux outils de l'utilisateur. Par exemple, les hyperliens sur des documents recommandés doivent être mis en évidence dans le browser et ces documents doivent être automatiquement préchargés dans le cache.

L'accroissement du trafic augmente significativement les temps de réponse lors des accès aux ressources. Pour faire face à ce phénomène, des services de cache ont été conçus [19] et déployés dans les différents nœuds de l'infrastructure. Ces caches sont généralement d'une utilisation transparente. Cependant, pour les petites communautés d'utilisateurs, il serait intéressant de pouvoir contrôler la durée de vie des documents dans le cache, indexer les ressources présentes ou précharger des documents à une heure donnée. Or, les caches n'offrent

généralement pas ce type d'opérations. Pour cela, il est nécessaire de les coupler avec des services additionnels.

De plus en plus d'utilisateurs accèdent à l'Internet à partir de différents endroits (travail, foyer, école, ...) avec des environnements très variés. Ces disparités touchent, entre autres, les architectures matérielles, les systèmes d'exploitation, les logiciels et les types de connexion utilisés. Plusieurs projets de recherche sont menés pour fournir des services capables d'adapter l'accès à l'Internet à l'environnement de l'utilisateur. Par exemple, des outils permettent de réduire le trafic sur les liaisons faibles (modem ou réseau sans fil) [12], de supporter des phases de déconnection [16, 24] ou d'adapter la présentation des documents selon les clients [13].

Ces différents services Web offrent des fonctionnalités très intéressantes. Cependant, ils ont été développés indépendamment les uns des autres et pour des environnements d'utilisation particuliers. Il est donc difficile d'arriver à les assembler pour construire un ensemble cohérent, adapté à des besoins précis. Aussi, il serait opportun de disposer d'une infrastructure regroupant les tâches communes et permettant aux services présents de travailler en collaboration.

Pour répondre à ces besoins, nous avons conçu Pluxy, une plate-forme modulaire qui peut accueillir et faire coopérer un ensemble de services Web. Elle factorise les tâches communes telles que le traitement des requêtes HTTP, la gestion des interfaces graphiques ou l'accès à des services distribués. Les services hébergés sont indépendants mais peuvent s'enrichir des fonctionnalités offertes par les autres services présents. Ainsi, un système de suivi de documents peut demander au service de cache de précharger les documents nouvellement mis à jour. Un service d'évaluation peut utiliser le service de suivi pour connaître les évaluations portant sur des documents ayant subi des mises à jour (qui rendent obsolètes les évaluations). Il peut aussi utiliser le cache pour y précharger les documents fortement recommandés.

Notre plateforme est matérialisée par un serveur mandataire (proxy) dans lequel il est possible d'ajouter, de retirer et de mettre à jour dynamiquement des services. Les services peuvent correspondre soit à des applications telles qu'un service de cache, soit à des composants réalisant des tâches internes (traitement des requêtes HTTP, gestion des interfaces graphiques, enrichissement des pages HTML, ...). Pour simplifier l'administration du proxy, les services peuvent être chargés et mis à jour à partir de ressources locales ou distantes. Pour cela, on utilise des techniques de code mobile et des mécanismes de sécurité capables de contrôler le comportement des services et d'assurer la pérénité du système.

Pluxy fournit l'infrastructure nécessaire pour que les services puissent collaborer entre eux. Chaque service peut connaître les autres services présents et requérir un service particulier. La gestion des dépendances entre services est assurée par le système. Ainsi, le service d'évaluation peut indiquer qu'il *souhaite* collaborer avec le service de cache et qu'il *doit* utiliser le service capable d'héberger son interface graphique.

Pluxy permet à plusieurs services de participer au traitement d'une même requête HTTP. Les points de concurrence de traitements sont réduits, localisés et contrôlés grâce à un découpage du traitement global de la requête en traitements élémentaires.

Pluxy est prévu pour être utilisé comme proxy personnel ou partagé (cf. figure 1). En utilisation personnelle, il peut être accompagné d'une interface graphique et les services peuvent y enregistrer leur propre interface pour interagir avec l'utilisateur. Pluxy a aussi été conçu pour fonctionner en mode déconnecté. Dans ce cas, seuls les services supportant cet état sont opérationnels et leur fonctionnement peut être plus ou moins dégradé.

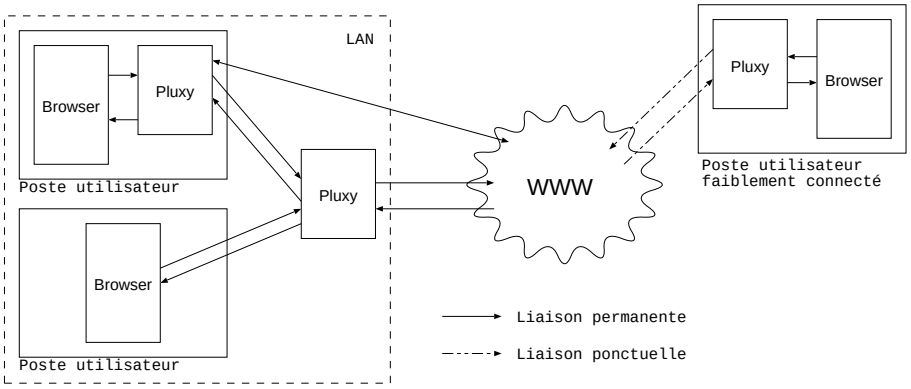

FIG. 1: *Environnement d'utilisation de Pluxy.*

Le reste de l'article s'organise comme suit. La section 2 compare Pluxy avec les autres systèmes de services additionnels pour le Web. La section 3 présente le modèle de composition de services ainsi que l'architecture du système. La section 4 décrit les services de base présents dans Pluxy et détaille en particulier le service chargé du traitement des requêtes HTTP. La section 5 présente les premières applications développées avec Pluxy. Enfin, la section 6 conclut l'article et discute des perspectives ouvertes par ce travail.

2 Comparaison avec des travaux analogues

2.1 Services additionnels pour le Web

L'architecture du Web permet de répartir les services additionnels dans les trois composantes du système : les clients (ou *browsers*), les serveurs et les mandataires (ou *proxies*).

Les browsers sont propices à l'hébergement de services dédiés à un utilisateur ou nécessitant une forte interaction. Néanmoins, la domination de ce marché par des applications propriétaires rend difficile le déploiement de nouveaux services dans les browsers. Leurs systèmes d'extensions (p. ex. *plug-ins* Netscape [22] ou composants ActiveX) sont incompatibles entre eux et dépendent du système d'exploitation. Ces extensions ne peuvent pas être ajoutées dynamiquement. L'utilisateur doit télécharger, installer et configurer les extensions manuellement. Elles s'exécutent avec les mêmes droits que le browser et peuvent remettre en cause la sécurité du système. À l'opposé, les applets Java sont portables et bien supportées

par la plupart des browsers. Cependant, elles ne sont pas destinées à accueillir de nouveaux services et elles ont des capacités réduites pour des raisons de sécurité.

Un serveur Web peut héberger des services additionnels en utilisant différents mécanismes d'extension (p. ex. CGI ou *servlet* [8]). Mais l'utilisateur ne peut généralement ni ajouter de service, ni les contrôler. D'autre part, le protocole de communication avec les serveurs (HTTP) est un protocole sans état. Les informations données par un utilisateur ne sont pas conservées entre deux requêtes. Il faut donc recourir à des techniques supplémentaires, telles que les *cookies* [17], pour construire des sessions d'utilisation. Enfin, ces services sont propres au serveur qui les diffuse et ne peuvent pas être utilisés lors de l'accès à d'autres serveurs.

Les proxies Web [19] sont une alternative intéressante aux browsers et aux serveurs. Ce sont des serveurs intermédiaires par lesquels transitent tous les messages des clients qui y sont connectés. Les proxies sont libres d'appliquer les traitements qu'ils veulent à ces messages. Par exemple, un proxy gérant un cache renvoie les documents qu'il possède dans son cache et fait suivre au serveur les requêtes sur des documents inconnus ou mis à jour. Un proxy peut être partagé par plusieurs clients. Ainsi, il est possible de factoriser les services destinés à des groupes d'utilisateurs dans un proxy commun. À l'opposé, un service spécifique à un utilisateur peut être réalisé par un proxy qui lui est dédié. Les proxies peuvent être chaînés et les messages peuvent transiter par plusieurs proxies offrant différents services. Enfin, ils sont très bien supportés par les clients ; leur configuration est simple et peut même, pour certains d'entre eux, être automatique [23].

La position privilégiée des proxies s'applique bien à tous les services qui doivent observer ou effectuer des traitements sur le trafic HTTP. Actuellement, il existe assez peu de proxies capables d'héberger et de faire coopérer un ensemble de services variés. Il est donc nécessaire de réécrire un proxy adapté au service proposé. De plus, l'utilisation cascadée de proxies augmente le temps d'accès aux documents et la consommation des ressources machines. Enfin, l'utilisateur a la charge de déterminer un ordre d'enchaînement des proxies ne provoquant pas de conflits. C'est par exemple le cas, si l'on dispose de deux proxies fournissant respectivement un service de redirection des requêtes sur des miroirs et le service d'évaluations décrit plus haut : si la redirection est placée en amont, le service d'évaluation ne pourra pas afficher les évaluation associées au document demandé car l'URL aura été modifiée par le service de redirection.

2.2 Applications à base de proxies Web

Les proxies Web sont principalement utilisés pour réaliser des services de cache [9, 20, 19] ou comme *pare-feux* en concentrant et en contrôlant les accès Web d'un réseau de machines en un seul point [10]. Cependant, le concept de proxy a aussi été repris pour filtrer à la volée les documents reçus. Par exemple, JunkBuster [1] et WebFilter [5] éliminent les images publicitaires des pages HTML. TranSend [13] réécrit les pages HTML et adapte les images pour les clients aux ressources limitées tels que les ordinateurs de poche. De nombreux projets [12, 13, 15, 27] ont aussi été menés pour utiliser les proxies comme systèmes adaptatifs dans les environnements faiblement connectés (modem ou réseaux sans fil). Par exemple,

en plaçant un proxy des deux côtés de la connection faible, il est possible d'utiliser des techniques de compression sur les flux HTTP pour réduire la consommation réseau.

La plupart de ces applications utilisent des proxies dédiés. Quelques projets de recherche proposent des outils génériques capable d'héberger un ensemble de services dans un même proxy. OreO [7] utilise le concept de *HTTP Stream Transducers*. Il s'agit d'un modèle très simple où chaque *transducer* comporte un flux d'entrée et un flux de sortie. Les *transducers* sont chaînés les uns aux autres et modifient séquentiellement les données qu'ils reçoivent. OreO ne propose aucun modèle d'agencement des *transducers* et laisse ce travail à la charge du développeur.

V6 [18] utilise un mécanisme de filtrage similaire mais il permet aux services de disposer d'une interface d'interaction en HTML (qui est accessible par une URL spécifique à chaque service). Une expression régulière est associée à chaque service. Lors d'une nouvelle requête, les services, dont l'expression régulière correspond à cette URL, fournissent un filtre. V6 cascade les filtres et les exécute séquentiellement. L'ordre d'enchaînement des filtres est statique. Chaque filtre doit participer au traitement de la requête et de la réponse. Enfin, V6 ne propose pas de modèle de collaboration entre les services.

Muffin [6] est un projet plus récent qui reprend la plupart des concepts de V6. On peut ajouter dynamiquement de nouveaux filtres. Les services peuvent disposer d'une interface graphique de configuration. Cependant, Muffin ne propose aucun modèle pour contrôler l'enchaînement des filtres (l'ordre est celui de chargement des filtres) ni de mécanisme de coopération entre les services.

3 Architecture de Pluxy

3.1 Le modèle de services de Pluxy

Les services se répartissent en trois catégories : les services noyau, les services de base et les services applicatifs.

Les services noyau sont des méta-services qui fournissent l'infrastructure nécessaire à l'exploitation des autres services. Ils réalisent des tâches telles que l'installation, le chargement et l'exécution des services. Ils sont aussi chargés de résoudre les dépendances entre services. Les autres services peuvent réutiliser tout ou partie des services noyau pour être eux-mêmes modulaires et dynamiquement extensibles.

Les services de base fournissent des fonctionnalités communément utilisées par les services applicatifs. Contrairement aux services noyau qui sont indispensables au fonctionnement du système, la présence des services de base est dictée par les besoins des services applicatifs. Pluxy est accompagné de trois services de base. HttpProxy permet de faire participer un ensemble de services au traitement des requêtes HTTP. PluxinBroker gère des services distribués. Enfin, PluxinPanelManager propose d'accueillir et de gérer les interfaces graphiques des services.

Les services applicatifs sont spécifiques à une application particulière. Ils sont gérés par les services noyau et peuvent utiliser différents services de base. Par exemple, un service de

cache participera au traitement des requêtes HTTP et disposera d'une interface graphique permettant à l'utilisateur de le configurer. Les premiers services applicatifs développés avec Pluxy sont présentés dans la section 5.

Les services de base et applicatifs sont matérialisés par des *pluxins*. Chaque pluxin regroupe un ensemble de composantes :

1. Une interface de service qui représente le service offert ;
2. Une implémentation qui réalise les fonctionnalités de ce service ;
3. Une interface de gestion du pluxin qui permet à Pluxy d'identifier ce service et de gérer son cycle de vie ;
4. Des composants respectant les interfaces nécessaires pour pouvoir collaborer avec d'autres services.

3.1.1 Identification des pluxins

Les pluxins sont identifiés de façon unique. Cette propriété permet à un pluxin d'identifier les autres pluxins avec lesquels il souhaite collaborer. L'identificateur complet se compose d'un identificateur unique de pluxin (pid) suivi d'un numéro de version optionnel : pid[:version]. Pluxy n'impose aucune structure pour le pid mais utilise, pour les pluxins de base qui l'accompagnent, un schéma similaire à celui des *packages* du langage Java. Il s'agit d'un nommage hiérarchique où chaque identificateur se compose d'un préfixe représenté par un nom de domaine Internet renversé (p. ex. fr.dyade) suivi d'une identification propre au pluxin (p. ex. fr.dyade.Pharos, fr.inria.sor.Archive, etc.)

3.1.2 Cycle de vie des pluxins

Avant d'être utilisé, un pluxin doit être installé puis chargé. L'installation consiste télécharger et à installer localement le pluxin. Le chargement a pour rôle d'ajouter le pluxin dans Pluxy, de résoudre les dépendances, de l'initialiser et de le faire démarrer. La phase d'installation n'a lieu que lors de la première utilisation du pluxin. Pour toutes les autres fois, seule la phase de chargement est répétée. Le chargement d'un pluxin est soit commandé par l'utilisateur, soit provoqué par le chargement d'un pluxin qui en dépend.

3.2 Gestion des pluxins

L'ensemble des services noyau est encapsulé dans le PluxinManager. C'est un service de façade qui agrège 4 services noyau : PluxinLoader, PluxinRegistry, PluxinClassLoader, PluxinSecurityManager. Il permet à l'utilisateur d'installer, de charger et de contrôler un pluxin. Il offre aussi une interface avec les pluxins présents qui leur permet d'obtenir une référence sur un pluxin enregistré ou d'être notifiés de l'ajout ou du retrait d'un pluxin.

3.2.1 Installation des pluxins

L'installation d'un pluxin consiste à récupérer une archive contenant les ressources locales du pluxin, identifiée par une URL. Les ressources de l'archive sont alors extraites dans un répertoire qui constitue le répertoire commun du pluxin. Un autre répertoire, correspondant au répertoire utilisateur du pluxin, lui est attribué. Cette distinction est importante pour l'utilisation partagée de Pluxy. Cela permet aux pluxins d'avoir des ressources partagées par l'ensemble des utilisateurs (p. ex. fichier de description du pluxin, code, icônes, ...) et des ressources propres à un utilisateur (p. ex. fichier de préférences).

3.2.2 Chargement des pluxins

Le chargement d'un pluxin est paramétré par un fichier de description. Ce fichier accompagne chaque pluxin. Il fournit les informations nécessaires au chargement et à l'initialisation de ce pluxin. Il contient :

1. L'identificateur du pluxin ;
2. Le nom des fichiers ou des répertoires contenant l'ensemble des classes du pluxin ;
3. Le nom de la classe représentant le pluxin ;
4. Les pluxins avec lesquels il collabore ;
5. Les *packages* Java externes requis.

Les pluxins et les *packages* Java absents sont téléchargés et installés. Si les dépendances ont pu être résolues, la classe du pluxin est chargée, instanciée puis initialisée. Lors de l'initialisation, PluxinManager fournit au pluxin un contexte d'exécution. Il permet d'obtenir la référence sur le PluxinManager et la valeur de certaines propriétés globales. C'est aussi par ce contexte que le pluxin peut accéder, en lecture et écriture, aux ressources situées dans ses parties commune et utilisateur (sous réserve de l'approbation du PluxinSecurityManager).

3.2.3 Référencement des pluxins

Après avoir été chargés, les pluxins sont enregistrés dans le PluxinRegistry. Tous les pluxins à l'écoute de l'arrivée de nouveaux pluxins sont notifiés de cet enregistrement. Tout pluxin peut obtenir une référence sur un pluxin enregistré à partir de son identificateur. Si le pluxin demandeur ne précise aucune version, le PluxinRegistry lui renvoie la dernière version connue. L'actuel PluxinRegistry ne permet d'obtenir que des références de pluxins locaux. Dans la prochaine version, il est prévu de permettre l'accès à un pluxin hébergé dans un Pluxy distant.

3.2.4 Contrôle de la sécurité

Le chargement dynamique de code étranger peut remettre en cause la perénité du système. Pour éviter cela, Pluxy utilise les trois couches de sécurité qu'offre Java : vérification du *byte code*, contrôle du chargement des classes et restriction des accès aux ressources. La

première couche est prise en charge par la machine virtuelle Java. La seconde est réalisé par PluxinClassLoader qui empêche qu'un service puisse invoquer des classes des services noyau. Enfin, toute tentative d'accès aux ressources du système est soumise à l'approbation du PluxinSecurityManager. Les ressources comprennent l'environnement d'exécution (arrêt, accès aux classes, introspection, *thread*, processus, chargement de librairies natives), le système de fichiers et les couches réseaux. PluxinSecurityManager peut appliquer différentes politiques sécuritaires selon l'identité du pluxin, sa provenance ou sa signature. Pour ce dernier cas, Pluxy utilise le mécanisme d'authentification proposés par Java [14].

Certains pluxins peuvent connaître, dès leur initialisation, leurs besoins d'accès aux ressources et les indiquer au PluxinSecurityManager. Par exemple, un pluxin gérant un cache peut indiquer la taille de l'espace disque dont il a besoin. Si le PluxinSecurityManager refuse la requête du pluxin et qu'il s'agit d'un besoin vital, ce dernier peut rester dans l'état initialisé et requérir l'intervention de l'utilisateur pour passer dans l'état actif.

4 Les services de base

4.1 Le traitement des requêtes HTTP

HttpProxy est un service de base qui accompagne Pluxy. Il permet à plusieurs services applicatifs de participer au traitement d'une requête HTTP. Il utilise le modèle d'enregistrement de Pluxy (i.e. : PluxinRegistry) pour gérer les pluxins participant aux traitements des requêtes. Ces pluxins doivent respecter l'interface HttpPluxin et s'inscrire auprès de HttpProxy.

4.1.1 Fonctionnement général

Le pluxin HttpProxy comporte une partie serveur, qui reçoit des requêtes HTTP et délègue leur traitement aux pluxins qui souhaitent y participer. HttpProxy supporte les protocoles HTTP/1.0 [4] et HTTP/1.1 [11]. La choix des participants a lieu à chaque requête. Pour cela, HttpProxy notifie cette requête à tous les pluxins inscrits. S'ils sont intéressés, ils fournissent en retour un ou plusieurs composants capables de réaliser un traitement élémentaire sur la requête. HttpProxy décompose le traitement complet d'une requête en huit étapes. Chaque étape réalise un traitement élémentaire :

1. Observation de la requête originale ;
2. Modification de la requête originale ;
3. Observation de la requête modifiée ;
4. Obtention d'une réponse correspondant à cette requête ;
5. Observation de la réponse originale ;
6. Modification de la réponse originale ;
7. Observation de la réponse modifiée ;

8. Renvoi de la réponse au client.

Chaque composant est capable d'effectuer un et un seul traitement élémentaire. HttpProxy répartit ces composants dans huit groupes selon le type de traitement qu'ils réalisent. HttpProxy ordonne les groupes et les composants au sein des groupes. L'agencement des composants d'un groupe est spécifique à ce groupe. L'agencement des groupes est représenté dans la figure 2.

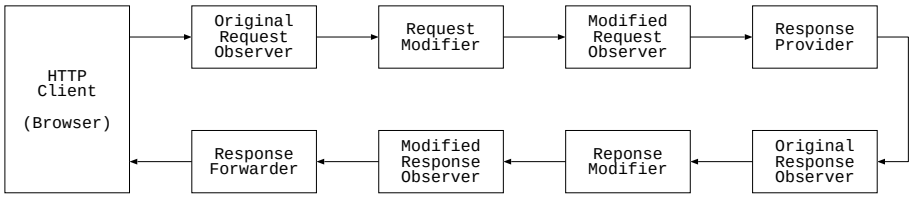

FIG. 2: *Enchaînement des traitements élémentaires sur une requête HTTP.*

Nous avons retenu cette décomposition car elle permet de localiser les points de concurrence de traitement entre les différents pluxins. De plus, grâce à ce découpage, certains groupes peuvent mener leurs traitements en parallèle. La figure 3 représente le graphe des exécutions des traitements élémentaires d'une requête. La justification de ce graphe est présentée dans les paragraphes suivants.

4.1.2 Détail du traitement complet d'une requête

HttpProxy démarre le traitement en transmettant la requête originale, en parallèle, aux groupes OriginalRequestObserver et RequestModifier. Les OriginalRequestObserver ne font qu'observer les requêtes reçues par HttpProxy. Ils ne la modifient pas et ne fournissent rien en retour. Leur exécution ne requiert donc pas d'ordre particulier et ils peuvent effectuer leurs traitements en parallèle. Le groupe RequestModifier travaille aussi sur la requête originale. Son exécution ne dépend pas de celle des OriginalRequestObserver, il peut donc effectuer son traitement en parallèle.

Les composants du groupe RequestModifier modifie la requête (en-tête et corps). Tous ces composants doivent être exécutés. Le groupe reçoit une requête et doit fournir une seule requête en retour. Une exécution parallèle des composants nécessiterait de fusionner les requêtes modifiées par chacun des composants, ce qui pourrait engendrer des conflits. C'est pour cette raison que nous avons retenu une solution plus simple en imposant une exécution séquentielle cascadée. La requête originale est donnée au premier composant qui effectue ses modifications et passe la requête modifiée au second qui fait de même. Dans ce modèle, si c_k représente le $k$ième composant de la chaîne et si $i < j$ alors l'exécution de c_j peut avoir un effet de bord sur le résultat produit par c_i. La requête résultante dépend donc de l'ordre d'enchaînement des composants. Dans cette version de HttpProxy, nous ne disposons pas de modèle permettant de formaliser les contraintes d'ordre entre les RequestModifier. Nous nous en remettons à l'utilisateur qui peut préciser dans la configuration de HttpProxy

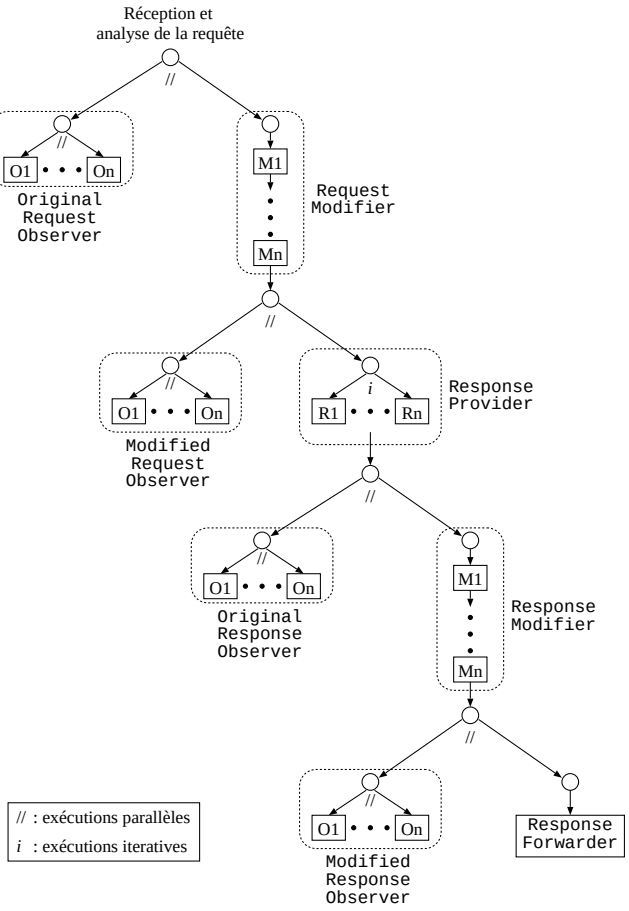

FIG. 3: *Graphe des exécutions des traitements élémentaires.*

(via son interface graphique) l'ordre global d'intervention des différents pluxins inscrits (si l'utilisateur ne précise rien, c'est l'ordre d'inscription qui est retenu).

La requête produite par le dernier composant du groupe RequestModifier est ensuite fournie, en parallèle, aux groupes ModifiedRequestObserver et ResponseProvider. Le groupe ModifiedRequestObserver effectue la même tâche que le groupe OriginalRequestObserver mais sur la requête modifiée. L'exécution des ResponseProvider ne dépend pas de celle des ModifiedRequestObserver.

Les composants du groupe ResponseProvider ont en charge de fournir une et une seule réponse à la requête qu'ils reçoivent. Contrairement au traitement de la requête par les RequestModifier, la réponse doit être consistante et ne peut pas être un amalgame des réponses

proposées par les différents composants. L'algorithme retenu consiste à ordonner les composants puis à faire une exécution itérative de chaque composant jusqu'à obtenir une réponse. Dès qu'une réponse est obtenue, l'itération est arrêtée et la réponse est transmise au groupe suivant (OriginalResponseObserver). Comme pour les RequestModifier, la réponse obtenue est fonction de l'agencement des composants. L'ordre est celui précisé dans la configuration de HttpProxy. Il se peut qu'aucun pluxin n'ait fourni de composant pour ce groupe ou qu'aucun des composants présents dans le groupe n'ait pu fournir de réponse. Aussi, afin de pouvoir toujours fournir une réponse à une requête valide, HttpProxy insère en dernière position un HttpFetcher. Ce composant est un client HTTP capable de faire une requête sur le réseau (directement ou via un proxy) pour obtenir une réponse à une requête HTTP valide. Lors des phases de déconnection, si HttpFetcher est invoqué, celui-ci retourne une page signalant que la requête ne peut aboutir.

Les groupes traitant la réponse sont symétriques à ceux traitant la requête. On trouve les groupes OriginalResponseObserver et ModifiedResponseObserver qui effectuent des traitements similaires aux groupes OriginalRequestObserver et ModifiedRequestObserver. Les composants du groupe ResponseModifier peuvent modifier la réponse. Leur exécution est séquentielle et leur ordre est celui précisé dans la configuration de HttpProxy.

Le groupe ResponseForwarder est un groupe réduit à un seul composant fourni par HttpProxy. Le rôle de ce composant est de retourner la réponse reçue au client. Les pluxins ne peuvent pas fournir de composants pour ce groupe.

Pour illustrer le fonctionnement de HttpProxy, considérons un Pluxy hébergeant un service de cache, un service de redirection de requêtes et un service d'évaluation de documents. Le service de cache doit pouvoir répondre à une requête dont il possède la réponse. Il réalise cette fonction avec un ResponseProvider. Il doit aussi stocker les documents qu'il voit passer. Pour cela, il utilise un OriginalResponseObserver. Le service de redirection modifie la requête afin de la rediriger sur une autre ressource. Il effectue cette tâche avec un RequestModifier. Enfin le service d'évaluation affiche les évaluations disponibles pour le document consulté et enrichit les pages HTML en marquant les liens intéressants. Il réalise la première opération avec un OriginalRequestObserver et la seconde avec un ResponseModifier.

4.1.3 Détails sur l'implémentation des composants de traitement

Les composants des groupes observateurs et des groupes modificateurs sont traités dans des *threads* distincts. Ce choix est nécessaire pour (i) permettre les exécutions parallèles et (ii) pour réduire la latence et la consommation mémoire. Le traitement d'une requête ou d'une réponse nécessite d'analyser l'en-tête et le corps du message. La taille des en-têtes est toujours faible (quelques centaines d'octets) tandis que la taille des corps est très variable : la taille d'une page HTML est en moyenne de 6 ko [25] mais les autres types de ressources (images, PDF, postscript, ...) peuvent atteindre plusieurs centaines de ko. Le chargement en mémoire d'une réponse au travers d'un objet spécifique permet d'analyser une et une seule fois l'en-tête. Les traitements peuvent alors se faire par invocation de méthodes avec cet objet en paramètre. Cependant, cette solution peut engendrer un forte consommation mémoire et augmenter considérablement la latence (i.e. : le temps que met le premier octet du document

à arriver). En effet, le premier ResponseModifier ne commencera son traitement que lorsque l'objet représentant la réponse sera disponible ; c'est-à-dire lorsque tout le document aura été chargé. Pour certains types de ressources, l'augmentation de latence ne gêne pas l'utilisateur. Par exemple, un fichier PDF ou une applet Java ont besoin d'être chargés en totalité avant de pouvoir être exploités). Par contre, pour les ressources interprétables à la volée (telles des ressources de type MIME 'text/*' et 'image/*'), l'augmentation de la latence nuit au confort de l'utilisateur.

Pour réduire la latence et la consommation mémoire, une solution consiste à faire transiter les requêtes et les réponses au travers de flux. Dans ce modèle, les composants sont chaînés les uns aux autres, dans un *pipeline*, et interagissent selon le paradigme *producteur/consommateur*. Le flux de sortie du composant c_i est connecté au flux d'entrée du composant c_{i+1}. Les composants lisent le message d'entrée sur leur flux d'entrée et écrivent leur message résultat sur leur flux de sortie. Ils peuvent commencer à produire un résultat partiel, disponible pour le composant suivant, en n'ayant lu qu'une partie du message. Ce modèle impose que les composants s'exécutent en parallèle et qu'ils soient hébergés chacun dans un *thread*.

L'utilisation unique des flux imposerait à chaque composant d'analyser le message. Une solution intermédiaire consiste à séparer le traitement de l'en-tête de celui du corps du message. L'en-tête est lu et analysé une seule fois. Un objet représentant cet en-tête est produit. Le premier composant du *pipeline* reçoit cet objet et une référence sur le flux du corps du message. Ce composant peut modifier l'objet en-tête. S'il souhaite modifier le corps, il construit un flux de sortie dans lequel il produira son résultat. Il invoque le composant suivant avec l'en-tête et la référence sur l'entrée de ce flux (ou du flux original s'il ne fait pas de modification sur le corps du message), puis il peut commencer son traitement sur le corps du message en lisant sur le flux d'entrée fourni et en écrivant le résultat sur le flux de sortie créé. Ce modèle n'est efficace que si les composants ne bloquent pas le *pipeline*. Si un composant attend la fin d'un document pour produire sa réponse, la latence augmente à nouveau.

Le modèle de flux engendre des complications pour gérer des accès concurrents à un même flux d'entrée. C'est le cas des observateurs qui doivent lire concuramment dans le même flux. La lecture dans le flux étant destructive, il est indispensable de conserver les données lues par le lecteur le plus en avance, pour les autres lecteurs. Pour cela, on gère une fenêtre glissante de mémorisation dont les extrémités sont désignées par les positions, dans le flux, du lecteur le plus en avance et de celui le plus en retard (cf. figure 4). La lecture d'un octet par le lecteur le plus en avance (R4 sur la figure) provoque l'incrémentation de la taille du buffer et l'ajout de cet octet. La lecture d'un octet par le lecteur plus en retard (R1 sur la figure) retourne le dernier octet du buffer et décrémente la taille du buffer.

Ce modèle peut dégénérer si un lecteur stoppe sa lecture. La fenêtre de bufferisation continue à s'agrandir mais ne se réduit plus. Dans le cas où un lecteur ne lit aucun octet, la fenêtre atteint alors la taille du document à lire ce qui nous ramène à la situation précédente. Pour éviter une telle situation, chaque composant peut indiquer qu'il ne souhaite plus lire le flux. On peut aussi être confronté au même phénomène si les vitesses de lecture sont très

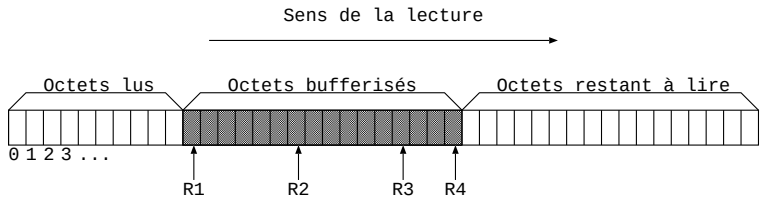

FIG. 4: *Fenêtre glissante de mémorisation d'un flux lu par plusieurs lecteurs.*

différentes. Pour pallier à ce problème, un objet supervise le buffer et a la responsabilité de réattribuer les priorités des *threads* afin de réduire les écarts. Le superviseur peut aussi éliminer un lecteur qui n'a pas effectué de lecture depuis un temps déterminé.

4.2 Service d'accès distant à un pluxin

PluxinBroker propose un service d'accès distant pour les pluxins. Ce service permet à un pluxin d'invoquer un pluxin situé dans une autre machine virtuelle locale ou distante. Pour cela, les pluxins doivent fournir un objet (éventuellement eux-mêmes) respectant l'interface RemotePluxin. PluxinBroker repose sur le mécanisme d'objets distribués RMI [26]. Chaque RemotePluxin est associé à une URL. Ces URL respectent le schéma suivant : `pluxy://host[:port]/pid[:version]` où `pid` représente l'identificateur du pluxin. PluxinBroker fournit les méthodes `bind` et `lookup`. La méthode `bind(url, pluxin)` permet d'associer un pluxin à une URL. La méthode `lookup` permet d'obtenir un représentant local (*stub*) d'un pluxin distant à partir de son URL. PluxinBroker se charge d'appliquer au RemotePluxy les propriétés liées au modèle de service de pluxy (i.e. : cycle de vie du pluxin et sécurité des accès). Pour l'instant, PluxinBroker n'offre pas de fonctionnalité de distribution en plus de celle de RMI. Plusieurs extensions sont envisagées :

- Couplage avec PluxinRegistry pour obtenir des références sur des (*stub* de) pluxins distants de la même manière que pour les pluxins locaux ;
- Couplage avec un service de migration de pluxins pour permettre la répartition de charge sur un ensemble de Pluxies ;
- Support pour la déconnection. Pour cela, le protocole de communication doit être étendu pour supporter la liaison flexible et pour différer les invocations lors des période de déconnection. Nous menons actuellement une étude pour intégrer à RMI les apports de travaux similaires réalisés sur les RPC [16, 2].

4.3 Gestion des interfaces graphiques des pluxins

PluxinPanelManager est un service de base qui permet aux autres services de disposer d'une interface graphique. Ces interfaces graphiques doivent respecter l'interface PluxinPanel

qui permet d'obtenir des informations liées à l'interface graphique (p. ex. son nom, l'identificateur du service qu'elle représente) et de contrôler son cycle de vie. PluxinPanelManager ne gère pas d'interface graphique mais sert d'intermédiaire entre une interface graphique et les services. L'interface graphique doit s'enregistrer au PluxinPanelManager. Lors de chaque ajout de PluxinPanel, PluxinPanelManager notifie l'interface graphique de la présence de ce composant. Une interface graphique par défaut, capable d'encapsuler un ensemble de PluxinPanel, est fournie avec ce service. La figure 5 illustre cette interface graphique hébergeant trois PluxinPanel.

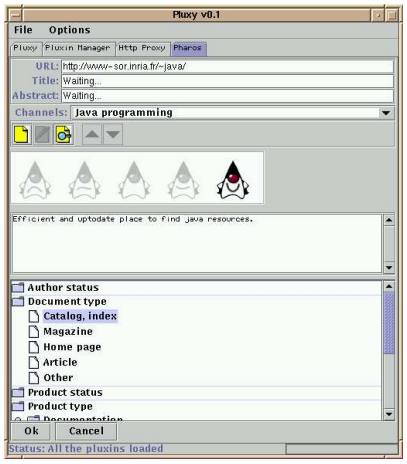

FIG. 5: *Interface graphique du service Pharos hébergée dans PluxinPanelManager.*

PluxinPanelManager a été conçu de manière à ce que les PluxinPanel puissent être gérés localement ou à distance. Cette distinction est nécessaire pour permettre à Pluxy d'être utilisé soit en proxy personnel soit en proxy partagé. Dans les deux cas, les pluxins disposant d'un PluxinPanel doivent le fournir au PluxinPanelManager. Pour une utilisation en proxy personnel, Pluxy démarre avec une interface graphique et insère les PluxinPanel enregistrés dans le PluxinPanelManager.

Lors d'une utilisation en proxy partagé, Pluxy s'exécute sans interface graphique. PluxinPanelManager dépose les PluxinPanel dans un PluxinPanelRepository. PluxinPanelManager exporte une partie de son interface via PluxinBroker. L'interface graphique peut être déclenchée à distance par l'application RemotePluxy. Elle se connecte au PluxinPanelManager, récupère et affiche les PluxinPanel disponibles. Aucun protocole n'est imposé pour le dialogue entre un PluxinPanel hébergé dans le RemotePluxy et le pluxin qu'il représente. Cependant, les pluxins peuvent utiliser le service PluxinBroker pour réaliser cette tâche. RemotePluxy peut aussi être utilisé sous forme d'applet, ce qui permet de piloter Pluxy depuis tout browser supportant les applets Java.

5 Applications

Nous présentons ici les premières applications conçues avec Pluxy. Ces applications visent à améliorer la facilité d'utilisation du Web. Grâce à Pluxy, elles peuvent coexister dans le même proxy, disposer de leur propre interface graphique et effectuer des traitements concurrents sur une même requête HTTP. Elles sont partiellement dépendantes, ce qui signifie qu'elles peuvent fonctionner indépendamment les unes des autres, mais qu'elles sont aussi capables de collaborer entre elles pour offrir une meilleure qualité de service.

5.1 Service de cache étendu

Ce service a été développé afin d'offrir un support pour la navigation en mode déconnecté. La solution retenue consiste à offrir un service permettant, à l'utilisateur et aux autres services, de stocker localement des documents. Ce service se comporte donc comme un cache mais dont les actions d'alimentation et d'élimination sont contrôlables. Ainsi l'alimentation peut être soit implicite (i.e. : mise en cache de tous les documents consultés satisfaisant certains critères), soit explicite (i.e. : demande de mise en cache d'une racine de documents selon une certaine profondeur). L'élimination des documents peut être aussi implicite (selon la durée de vie attribuée aux documents) ou explicite (en choisissant les documents à éliminer). Dans les deux cas, le cache assure la cohérence des documents : une ressource ne peut être éliminée que si elle n'est plus référencée par aucune autre ressource présente dans le cache. Dans le cas de pages HTML, les références correspondent soit à des documents embarqués (images, applet, ...) soit à des hyperliens.

Ce service est abonné à HttpProxy. Il fournit deux composants à chaque requête : un ResponseProvider et un OriginalResponseObserver. Le composant ResponseProvider retourne, le cas échéant, les documents disponibles dans le cache. Le composant OriginalResponseObserver stocke dans le cache les réponses reçues selon la politique d'alimentation choisie. Ce service dispose aussi d'une interface graphique qui permet à l'utilisateur de configurer le cache et de gérer son contenu.

5.2 Pharos : un système de filtrage collaboratif du Web

Le projet Pharos a pour but d'améliorer la recherche d'informations sur le Web. Pharos utilise pour cela les agents intelligents du réseau : les utilisateurs. Lorsqu'un utilisateur cherche une information, il existe toujours au moins une personne qui connaît une ou des pages pouvant correspondre à sa recherche. Pharos fournit une infrastructure pour que ces personnes puissent partager leurs connaissances du réseau. Pour ce faire, il propose un système d'évaluation collaboratif des ressources disponibles sur le Web. Les critères d'évaluations pouvant varier d'un domaine à l'autre, les évaluations sont réparties dans des canaux thématiques (dans le même esprit que les *news groups*). Une évaluation est caractérisée par le canal dans lequel elle est diffusée, l'URL du document qu'elle référence, son auteur et ses valeurs. Les valeurs correspondent aux valeurs possibles des différents champs de la grille d'évaluation du canal. Cette grille est propre au canal. Néanmoins, Pharos propose un canal

de base qui peut être enrichi. La grille de ce canal comporte trois champs : l'intérêt du document (une note de 1 à 5), les mots-clés du document (choisis dans un thésaurus propre au canal) et un commentaire en texte libre.

Les membre du canal peuvent consulter ces bases d'évaluation pour rechercher des documents correspondant à certains critères. En plus de ces possibilités de recherche explicite, Pharos est en mesure de faire des recommandations adaptées pour chaque membre. Ces recommandations peuvent être calculées selon différents algorithmes de synthèse. Nous proposons des algorithmes simples calculant la moyenne des évaluations, éventuellement pondérée par certains critères tels que l'auteur ou la date, ainsi qu'un algorithme de prédiction selon le passé. Dans cet algorithme, on essaie d'établir des corrélations entre les évaluations précédemment posées par les membres pour déterminer des profils proches. À partir de ces calculs, Pharos est alors en mesure de recommander à un membre des documents évalués par d'autres membres ayant des profils similaires.

Le modèle des canaux est très proche de celui de pluxins. Pharos gère ses canaux (installation, chargement, retrait, contrôle de l'accès aux ressources, etc.) de la même manière que PluxinManager gère les pluxins. Cependant, contrairement aux pluxins, les canaux ne collaborent pas entre eux.

Pharos utilise PluxinPanelManager pour gérer son interface graphique. Les interfaces graphiques des canaux sont prises en charge par Pharos. Pharos est abonné au pluxin HttpProxy et fournit un OriginalResponseObserver. Il diffuse chaque nouvelle réponse aux canaux afin qu'ils affichent la liste des évaluations et des recommandations disponibles pour ce document et qu'il permettent, le cas échéant, de saisir une nouvelle évaluation pour ce document. Si le service de cache étendu est présent, les canaux peuvent collaborer avec lui pour précharger les documents recommandés.

5.3 Suivi des mises à jour des ressources Web

Le service de notification permet d'être informé des mises à jour survenues sur un document. Contrairement aux deux précédents, ce service n'est pas abonné à HttpProxy. Il interagit avec un gestionnaire de notification distant qui fédère les demandes de notification pour réduire le trafic. Afin de préserver l'anonymat des utilisateurs, le gestionnaire ne garde aucune trace des utilisateurs. Lorsqu'un utilisateur désire être notifié des mises à jour, il interroge le gestionnaire en lui fournissant la liste des URL des documents suivis avec leur date de dernière modification. Le gestionnaire lui retourne l'état de chacun de ces documents. Ce service dispose d'une interface graphique pour indiquer les documents à suivre et prendre connaissance des mises à jour. Il fonctionne en collaboration avec le service de cache pour précharger les documents mis à jour avant de les notifier à l'utilisateur.

5.4 Redirection des requêtes

Ce service redirige les requêtes HTTP sur des sites miroirs. La liste des site miroirs est indiquée par l'utilisateur sous la forme d'une série de couples {site original, site miroir}. Ce

service est abonné à HttpProxy. Pour chaque requête destinée à un site dont il connaît un miroir, il fournit un RequestModifier qui redirige la requête sur ce miroir.

5.5 Caractérisation des liens

Ce service permet aux autres services d'enrichir les liens présents dans une page HTML. Les services souhaitant faire des enrichissements doivent s'abonner à ce service. Il est lui-même abonné à HttpProxy et fournit un composant ResponseModifier qui extrait les liens contenus dans une page HTML. Pour chaque lien rencontré, il demande une caractérisation aux services abonnés. Ces services renvoient un texte HTML qui est mis à la suite du lien dans la page. Il est, en particulier, utilisé par le service de cache pour indiquer si un lien pointe sur un document présent dans le cache et par Pharos pour afficher l'évaluation du document pointé par le lien.

6 Conclusion et perspectives

Pluxy offre une plate-forme générique et extensible de services Web hébergés dans un proxy. Il peut être utilisé en proxy personnel ou partagé. Il propose un système de composition dynamique de services ainsi que l'infrastructure nécessaire pour qu'ils puissent se découvrir et coopérer. Il permet une administration simplifiée des services en supportant le téléchargement et la gestion des versions. Enfin, Pluxy a été concu pour héberger aussi bien des filtres HTML simples que des applications plus ambitieuses nécessitant un environnement d'exploitation complet.

Le service applicatif de cache étendu permet d'utiliser Pluxy dans les environnements faiblement connectés . Cependant, ce service sert uniquement à la consultation de ressources Web. Il ne fournit aucun support pour les services interagissant avec des services distants selon d'autres protocoles. C'est par exemple le cas des canaux Pharos qui dialoguent avec leur dépôt d'annotations distant. Un service plus général de support de la déconnexion est en cours d'étude.

Pluxy a été conçu pour héberger des services Web. Cependant, son architecture flexible lui permet d'accueillir des services pour d'autres protocoles de communication. Par exemple, on pourrait utiliser Pluxy pour faire des traitements particuliers sur les systèmes de messageries SMTP, POP ou IMAP, tels que l'indexation des messages reçus, la gestion de réplicats de boîtes aux lettres ou l'utilisation en mode déconnecté.

Pluxy est partiellement implémenté. La gestion des services et les services de bases HttpProxy et PluxinPanelManager sont disponibles. Les autres composants du système sont en cours de réalisation. Les deux services applicatifs, Pharos et le cache contrôlable, sont aussi en phase de développement. Nous avons l'intention de rendre disponible Pluxy pour encourager les utilisateurs à développer de nouveaux services.

Remerciements

Je tiens à remercier les membres de l'action Webtools et ceux du projet SOR qui ont fait de nombreux commentaires sur les versions préliminaires de cet article, notamment Vincent Bouthors, Mesaac Makpangou et Guillaume Pierre.

Références

[1] JunkBuster, 1996. `http://www.junkbusters.com/`.

[2] Bakre, A., and Badrinath, B. M-RPC : a remote procedure call service for mobile clients. Tech. rep., Rutgers University, May 1995. `ftp://paul.rutgers.edu/pub/badri/mrpc.ps.Z`.

[3] Balabanovíc, M., and Shoham, Y. Fab : content-based, collaborative recommendation. *J. ACM 40*, 3 (Mar. 1997), 66–72. `http://www.acm.org/pubs/citations/journals/cacm/1997-40-3/`.

[4] Berners-Lee, T., Fielding, R. T., and Nielsen, H. F. Informal RFC 1945 - Hypertext Transfer Protocol – HTTP/1.0, May 1996. `ftp://ftp.inria.fr/rfc/rfc19xx/rfc1945.Z`.

[5] Boldt, A. WebFilter, 1997. `http://math-www.uni-paderborn.de/~axel/NoShit/`.

[6] Boyns, M. Muffin, 1997. `http://muffin.doit.org/`.

[7] Brooks, C., Mazera, M. S., Meeks, S., and Miller, J. Application-specific proxy servers as HTTP stream transducers. In *Proceedings of the 4th international WWW conference* (Boston, MA, Dec. 1995). `http://www.w3.org/pub/Conferences/WWW4/Papers/56`.

[8] Chang, P. I. Inside the Java Web server, 1996. `http://java.sun.com/features/1997/aug/jws1.html`.

[9] Chankhunthold, A., Danzig, P. B., Neerdaels, C., Schwartz, M. F., and Worrel, K. J. A hierarchical Internet object cache. In *Prooceedings of the Winter Usenix Conference* (San Diego, (USA), Jan. 1996). `http://catarina.usc.edu/danzig/cache.ps`.

[10] Chapman, D. B., and Zwicky, E. D. *Building Internet Firewalls*. O'Reilly & Associates Inc., 1995.

[11] Fielding, R. T., Gettys, J., Mogul, J. C., Nielsen, H. F., and Berners-Lee, T. RFC 2068 - Hypertext Transfer Protocol – HTTP/1.1, Jan. 1997. `ftp://ftp.inria.fr/rfc/rfc20xx/rfc2068.Z`.

[12] Fox, A., and Brewer, E. A. Reducing WWW latency and bandwidth requirements by real-time distillation. In *Proceedings of the 5th International Conference on the World-Wide Web* (Paris (France), Mar. 1996). `http://www5conf.inria.fr/fich_html/papers/P48/Overview.html`.

[13] Fox, A., Chawathe, Y., Brewer, E. A., and Gauthier, P. Cluster-based scalable network services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (St Malo (France), Oct. 1997). http://www.cs.berkeley.edu/~fox/papers/sosp16.ps.

[14] Gong, L. Java security : Present and near future. *IEEE Micro 17*, 3 (May/June 1997), 14–19. http://www.computer.org/micro/mi1997/m3014abs.htm.

[15] Housel, B. C., and Lindquist, D. B. WebExpress : A system for optimizing Web browsing in a wireless environment. In *Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking* (Nov. 1996), ACM. http://www.networking.ibm.com/art/artwewp.htm.

[16] Joseph, A. D., deLespinasse, A. F., Tauber, J. A., Gifford, D. K., and Kaashoek, M. F. Rover : a toolkit for mobile information access. In *Proceeding of the Fifteenth Symposium on Operating Systems Principles* (MIT, Dec. 1995), USENIX. http://www.pdos.lcs.mit.edu/rover/.

[17] Kristol, D. M., and Montulli, L. RFC 2109 - HTTP state management mechanism, Feb. 1997. ftp://ftp.inria.fr/rfc/rfc21xx/rfc2109.Z.

[18] Lang, B., and Rouaix, F. The V6 engine. In *Proceedings of the 5th International Conference on the World-Wide Web* (Paris (France), Mar. 1996). http://pauillac.inria.fr/~rouaix/V6/.

[19] Luotonen, A., and Altis, K. World-Wide Web proxies. In *Proceedings of the 1rst International Conference on the World-Wide Web* (Geneva, May 1994). http://www.cern.ch/PapersWWW94/luotonen.ps.

[20] Makpangou, M., and Éric Bérenguier. Relais : un protocole de maintien de cohérence de caches Web coopérants. In *Proceeding of the 1997 NoTeRe '97 symposium* (Pau (France), Nov. 1997). http://www-sor/publi/RPMCCWC_notere97.html.

[21] NetMind. Url-minder, 1996. http://www.netmind.com/html/url-minder.html.

[22] Netscape. Netscape plug-in guide, 1996. http://developer.netscape.com/docs/manuals/communicator/plugin/index.htm.

[23] Netscape. Netscape proxy automatic configuration, 1996. http://home.netscape.com/eng/mozilla/2.02/relnotes/unix-2.02.html.

[24] Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J., and Hauser, C. H. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (Cooper Mountain Resort, Colorado, Dec. 1995).

[25] Touch, J., Heidemann, J., and Obraczka, K. Analysis of HTTP performance, Aug. 1996. http://ntrg.cs.tcd.ie/4ba2.96/group6/performance.htm.

[26] Wollrath, A., Riggs, R., and Waldo, J. A distributed object model for the Java system. In *Proceeding of the USENIX 1996 Conference on Object-Oriented Technologies (COOTS)* (Toronto, June 1996), USENIX. http://www.usenix.org/publications/library/proceedings/coots96/wollrath.html.

[27] ZENEL, B. *A proxy based filtering mecanism for the mobile environment.* PhD thesis, Columbia University, Mar. 1996. `http://www.mcl.cs.columbia.edu/~baz/thesis.baz.ps.gz`.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399