An intermediary software infrastructure for edge services

Raffaella Grieco, Delfina Malandrino, Vittorio Scarano, Francesco Varriale
Dipartimento di Informatica ed Applicazioni "R.M. Capocelli",
Università di Salerno, 84081 Baronissi (Salerno), Italy.
E-mail: {rafgri, delmal, vitsca}@dia.unisa.it

Francesca Mazzoni
Dipartimento di Ingegneria dell'Informazione,
Università di Modena e Reggio Emilia, 41100 Modena, Italy.
E-mail: mazzoni.francesca@unimo.it

Abstract

We describe the goals and architecture of a new framework that aims at facilitating the deployment of adaptation services running on intermediate edge servers. The main goal is to guarantee robustness and quick prototyping of functions that should integrate mobile/fixed-network services. Moreover, we intend to design a distributed architecture with the purpose of guaranteeing efficient delivery.

1. Introduction

The World Wide Web and the services it provides are a remarkable reality in our society. In the last decade, the obscure and, somewhat, "technical" Internet, mainly used by scientists, has been diffused among all the population in the industrialized world. Many different explanations are usually brought to motivate this astonishing success, from technological to sociological to economical ones. Among the former, very important is the emphasis on the capabilities of the standards to accommodate a wide range of services, therefore, pushing the World Wide Web as a universal access portal to the information, wherever located and however accessed.

A challenge of different nature has been posed to the World Wide Web by mobile communication. In fact, the only technological trend that can be compared to the WWW in terms of growth, diffusion and size are the ubiquitous services offered by mobile terminals connected by wireless networks. Such services are growing in nature and accessibility, with more and more dynamic, multimedia and interactive content offered to mobile users. In particular, the access to the content traditionally accessed via Web comes, now, from a wide range of heterogeneous devices that are connected to the Internet through different wireless technologies, such as GPRS, UMTS, Bluetooth etc. The challenge to the World Wide Web is to be able to seamlessly integrate these services, in order to convey to the user the feeling of a unique platform that follows him and suits his needs, wherever and however located.

Wireless and mobile environments involve more challenges to end users than wired network environments. Mobile terminals are characterized by several constraints, such as battery power, device weight, graphical displays, portability, limited I/O, limited bandwidth and intermittent connection to the network. Moreover, other limitations come from servers' inability to handle software variations, such as the different data formats and protocols that clients can support. Intermediaries infrastructures represent the suitable way to address these problems, as they are able to deal with different network conditions and client heterogeneity. These systems can also efficiently provide context awareness and context adaptation, very important features of mobile and ubiquitous environments. In fact, bandwidth limitation and constraints on user interfaces require that all the information transmitted and visualized is strictly pertinent to the user's query.

Here, we present the status of an ongoing project, that is to deliver a framework for easily and efficiently deploying intermediary services on the WWW. The main overall goals of the architecture are efficiency, programmability, life-cycle support, security, scalability, persistence and personalization. Here, we show how our architecture (as current status of the project) is innovative since: it provides programmability without giving up efficiency (with respect to classical Java-based proxy environments), it offers "horizontal" users' profiles management primitives, and deployment/un-deployment mechanisms.

2. The State of the Art

Our work stems from two complementary research areas: the application servers for distributed components and the World Wide Web proxies and intermediary systems.

Application Servers. The World Wide Web was soon recognized as an ideal platform to deploy distributed applications. As a matter of fact, when the WWW was presented, one of the most exciting breakthrough for the average user (accustomed to, say, pre-WWW programs like Gopher) was that his requests on a HTML form could be answered by a server-side computation (via the well-known Common Gateway Interface Binaries (CGI-BIN) specifications). This innovation of the server-side computation was ideally complemented, at that time, by the introduction of client-side executable content (i.e. the Java applets).

The evolution of the server-side computation model of the World Wide Web was quick but complete, passing through a 2-tier model to a 3-tier model, where the presentation logic, the business logic and the data access logic was separated, respectively, on a Web browser, on a Web server equipped with executable programs and a Database Management System as a back-end.

The needs for quick prototyping and deploying, as well as the increase in complexity and requirements, promoted the introduction of *distributed components* that were executed in an *application server* or container. An application server provides common middleware services, such as resource pooling, networking, security, etc. in such a way that the designer can focus on the application that is going to be run by the server. In order to allow for interoperability, the application is built as a number of distributed components, i.e. programs that implement a set of well-defined interfaces. By leveraging on portability and reusability, distributed components represent an important and durable investment for companies.

Two are the main middleware platforms that currently offer application servers: Sun Java 2 Enterprise Edition (J2EE) and Microsoft .NET. The former represents a multi-vendor standard that enrolls several major players in the Information technology (such as IBM Websphere Application Server or Oracle Application Server 10g) while the latter is mainly based on Microsoft servers.

Intermediary Systems. One of the current research trend in distributed systems is how to extend the traditional client/server computational paradigm in order to allow the provisioning of *intelligent* and *advanced* services. One of the most important motivation is to let heterogeneous devices access to WWW information sources through a variety of emerging 3G wireless technologies.

This computational paradigm introduces new actors within the WWW scene, the intermediaries [7, 17], i.e. software entities that act on the HTTP data flow exchanged between client and server by allowing content adaptation and other complex functionalities, such as geographical localization, group navigation and awareness for social navigation [6, 9], translation services [15], adaptive compression and format transcoding [2, 14, 23], etc.

Web Based Intermediaries (WBI) [15] is a programmable proxy, written in Java, developed at IBM Almaden Research Center, in order to simplify the development and the deployment of Web Intermediaries, i.e. applications that deal with HTTP information flows. The WBI approach is based on the notion of information streams [6], on which transformations can be applied in order to allow content adaptation and personalization. These functionalities are provided by WBI components, called MEGs, dynamically invoked on the HTTP request/response flow.

The BARWAN project [16] by UC/Berkeley has the goal to provide intermediary systems to support ubiquitous access to Internet services from mobile and thin clients. An important system component of this project is the proxy architecture, TACC, that acts as intermediary between servers and mobile clients. The TACC programming model provides important functionalities, such as transformation, aggregation, caching and customization of the Web content [8, 10].

iMobile architecture [21], by AT&T Research, aims to hide the complexity of multiple devices and content sources from mobile users. It provides a framework for developing and composing intermediary components in complex distributed applications. Its main component is iProxy [3], a programmable proxy server that provides personalized services, which are implemented as reusable building blocks in Java.

SEcS: Scalable Edge-computing Services architecture [13], is a programmable networking infrastructure whose main goal is to address the challenge of developing and deploying Internet services, that is, scalable, robust, highly available and, more recently, added-value services, the *Edge services*, that are able to adapt, aggregate and transform many information sources at the network *Edge*, nearer to the end users.

SEcS provides support for personalization and configuration of the services required on a per-user basis, and also supports the dynamic composition of services into a data path, as well as the adaptation along such data path.

An important direction toward standardization is being conducted by the IETF Working Group for "Open Pluggable Edge Services" (OPES) [4, 5]. Their goal is to define an open standard that allows intermediaries to provide services on the HTTP data flow exchanged between client and server. In OPES, intermediaries can also employ local or remote servers (called "callout servers") in order to facilitate the efficient delivery of complex services. OPES ruleset are applied in order to choose which service to apply to the data flow.

3. The requirements

In this section, we describe some non-functional requirements. Later, in Section 4 we provide a sketch of the overall architecture, because of the work in progress on the project, and describe the currently implemented functionalities (Section 5).

The proxy-applications server (or *proxy container*) should provide the following "horizontal" services to each proxy services infrastructure, therefore alleviating the programming effort to quickly build efficient Internet proxy services.

• **Life-cycle support:** the proxy container must fully support the deployment and un-deployment of proxy services, by making these tasks automatic and accessible by remote locations. Moreover, support for pausing and restarting services is also necessary.

• **Programmability**: programming under existing intermediaries cannot involve the same power, efficiency, expressiveness and generality like developing an open system from scratch. This means that we need a compositional framework and a programming model that allow us to realize a general-purpose programmable environment, whose functionalities will be implemented by APIs provided by the intermediary system.

• **Security**: the proxy container must be able to allow access to resources in a *trusted* way allowing both authorization and authentication for each user's session. Moreover, services management mechanisms must control that users are allowed to perform only the operations that they have rights to perform.

• **Persistence:** services must be guaranteed to keep a per-service status as well as a per-user and per-transaction status. Per-user and per-transaction status allow, e.g., personalization services while per-service status offers cooperation and sharing of resources. In general, persistence should ensure that status is also kept in case of a server shutdown and restart.

• **Scalability:** the proxy container could be (seamlessly) run on a cluster. Several instances of the proxy container would transparently collaborate over a LAN to offer a single execution environment for proxy services. The location transparency requirement implies that neither the proxy programmer (during the development of the proxy service) nor the user (during his interaction with the proxy services) are aware of the fact the service is provided by (any node of) a cluster. It also means that services can be dynamically located in the cluster and that some level of fault-tolerance should be allowed (transparent fail-over). Notice that scalability can heavily impact on persistence; in fact, HTTP requests from the same user (or within the same transaction) are guaranteed to reach the same node of the cluster and, therefore, additional mechanisms are to be employed.

• **Back-end integration**: our proxy container should be also accessible as a traditional Web server in order to enhance traditional applications. As application servers provide an easy interaction with DBMS and Naming server, each proxy service should be easily integrated with the other services provided by the Web server. A classical example could be the integration with portals and cooperative systems.

• **Logging and auditing**: the proxy container should provide mechanisms to record security-related events (logging) by producing an audit trail that makes possible the reconstruction and examination (auditing) of a sequence of events. The process of capturing user activity and other events on the system, storing this information and producing system reports is important for understanding and recovering from security attacks. Logging is also important to provide billing support, since services can be offered with different price models (flat-rate, per-request, per-byte billing options). The container should offer the possibility to manage accounting to users for each service, as specified by the proxy service manager.

• **Inter-services communication:** when deploying different proxy services on a WAN, the programmer can cascade by chaining them with HTTP, therefore, there is no (strong) need for an additional communication medium among proxy services that are run on different container. Different is the situation when proxy services, run on the same container, need to exchange information. In that case, a communication protocol (necessarily asynchronous, as any message-oriented middleware) is needed.

4. The architecture and the implementation

Here, we provide a sketch of the architecture and provide some details of what we already implemented.

As preliminary step, our decision was to base our work on top of existing open-source, mainstream applications, such as Apache Web server. First of all, it will make our work widely usable because of the popularity of the Apache Web server [22]. Then, our results will be released as open source (by using some widely accepted open-source license) so that it will be available for improvements and personalizations to the community. Then, last but not least, Apache is a high quality choice, since it represents one of the most successful open-source projects (if not the most successful) that delivers a stable, efficient and manageable software product to the community of Web users.

4.1 Apache and mod_perl

Apache [1] is recognized as the world's most popular Web server (HTTP server) [22]. It provides a full range of Web server features, including CGI, SSL, and virtual domains. Apache also supports plug-in modules for extensibility and is reliable and relatively easy to configure. Finally, it is a free software distributed by the Apache Software Foundation, that promotes various open source advanced Web technologies.

mod_perl [19] represents the perfect marriage of the Perl programming language [20] and the Apache Web server [1]. It brings together the power of these two powerful technologies by providing a programmable framework for building and accelerating dynamic content, providing mechanisms for database integration, allowing a simple customization of new modules that can be directly and easily integrated into Apache, managing the Apache configuration file, etc.

The most important characteristic of the Apache Web server is that it can be extended with new modules, commonly written in C programming language, but since it is a hard work, the new opportunity is to write such modules entirely in Perl programming language and integrate them in Apache through mod_perl. This leads to extend the behavior of the Apache Web server by taking advantage of the power and flexibility of the Perl language.

Having a persistent Perl interpreter embedded in Apache Web server allows to avoid the overhead of starting an external interpreter for any HTTP request which needs to run Perl code. An important feature is the code caching: the modules and scripts are loaded and compiled only once, when the server is first started. Then for the rest of the server's life the scripts are served from the cache, so the server only has to run the pre-compiled code.

Each HTTP request is processed in sequential phases, and at each phase different decisions can be taken about the request (processed, rejected, or simply forwarded to the succeeding phase). HTTP requests can be managed by the standard Apache core, or by new and external modules. With mod_perl all phases of the HTTP request cycle can be accessed and controlled, allowing to enhance and personalize the behavior of the Web server. Each HTTP request goes through twelve phases, and on each phase it is possible to provide a specialized handler, such as a PerlTransHandler to manipulate the requested URI, the PerlAccessHandler to provide restrictions, the PerlAuthenHandler and PerlAuthzHandler to provide authentication and authorization mechanisms, the PerlHeaderParserHandler to inspect the HTTP requests and perform tasks according to conditions, the PerlResponseHandler to produce HTTP responses, etc.

4.2 Architecture Overview

Our server is entirely written in Perl and utilizes mod_perl within Apache to speed up performance. Our framework consists of several modules: each of them acts in a specific phase of the Apache HTTP Request Lifecycle (see Fig. 1). In particular the *ProxyPerl* and the *Authorization* Handlers have been implemented as standard Apache/mod_perl 2.0 HTTP Handlers, the *Registry* and the *Deploy* Modules as standard Apache/mod_perl 2.0 HTTP Response Handlers, and finally the *FilterPlugin* Module as the standard Apache Filter Handler.

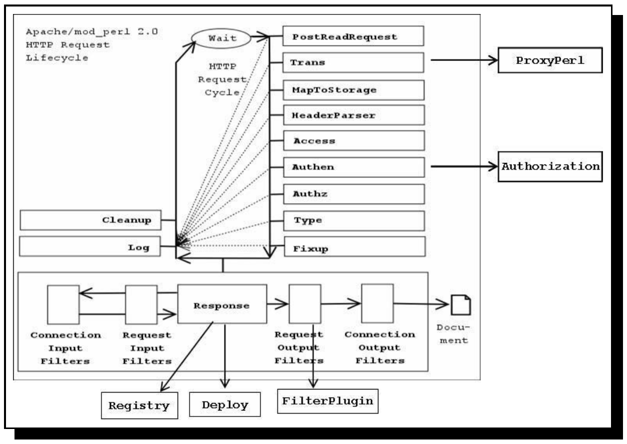

Figure 1. How our modules fit in the HTTP Request Lifecycle of Apache.

Moreover, the ProxyPerl and FilterPlugin Modules are part of a what we call the *Proxy View*, that is the functionalities offered by these modules are all implemented proxy-side. On the other hand, the Registry and Deploy Modules are part of what we call the *Server View* since all functionalities offered by these modules are server-side, offering users/administrator an interaction to manage (personal/services) configurations (see Fig. 1). It must be emphasized that the Authorization Module is a hybrid module since any functionality is usually managed with a server view control and then provided as proxy view.

Let us start with a detailed description of the functionalities offered by these modules.

The ProxyPerl Module. The *ProxyPerl Module* is a Perl Module that represents the proxy component of our architecture. It intercepts all clients requests and initializes the transaction process. Its most important task is to fetch the requested URLs manipulating, if necessary, HTTP headers. If no transformation must be applied on the HTTP flow, the requested document will be returned back to the client, else the transaction will be managed by the handlers invoked according to users' profiles.

When the client issues an HTTP request, the ProxyPerl Module identifies the user (or provides to initiate a challenge-response authentication), and then loads the configuration of the services that the user selected during the configuration. Once the client is identified by the ProxyPerl Module, the *personalized* navigation begins: the services selected by the user will be applied to the HTTP flow of requests/responses.

The Authorization Modules. Each proxy service must be able to rely on a module to easily and safely authenticate the proxy user ("*Are you who you say you are?*") as well as to check the operations that the user is performing on the component ("*Are you allowed to do what you are asking for?*").

This means that the proxy must verify that the user issuing a request is authorized to do so, or that user is charged for a specified operation. Then, the Authorization service is useful both for restricting access to a proxy server as well as to distinguish between users. If we are interested in services that treats users differently, e.g., per-user settings, it is important provide a mechanism that is able to distinguish between them. An example is the creation of user profiles that necessarily requires a mechanism to identify Web users.

The Apache Authentication phase is called whenever the requested file or directory is password protected. Our *Authentication Module*, that acts in this phase, is used to verify user's identification credentials. If the user is authenticated, the handler loads the user's profile and starts the navigation, otherwise it provides a challenge-response authentication mechanism asking for the credentials by using the Proxy Authorization HTTP header `Proxy-Authorization` as detailed in [11].

The Registry Module. The Apache Registry Handler is the standard Apache module that allows to run CGI scripts very fast under mod_perl, by compiling all scripts once and then caching them in main memory.

Our architecture uses the Apache Registry Handler to handle the user's and services data. In particular the scripts are part of three different categories: the first gives information about the available services; in the second one the administrator manages the user's home and, finally, the last category of scripts is used by users to modify their own data and profiles. In order to obtain a services' list, the user must enter its credentials (login and password), and access the *Change Profile Section*. At this point the user can choose among the offered services. The page representing the list of services is composed of a set of HTML forms each of them is used to set the parameters of a specific service. In addition, the administrator, after the authentication phase, can perform some tasks such as adding or removing users from the system. After the initialization phase, any user can access the system (providing username and password) and then perform some operations such as change the password, create, modify or change his profile.

The FilterPlugin Module. The *FilterPlugin Module* acts as dispatcher within our architecture. A list of available services is deployed into Apache and it can be accessed by the dispatcher in order to activate the services according to the users' preferences (users' profiles). To this end, the FilterPlugin Module uses an internal parser to dynamically invoke the requested services. Services are applied on the HTTP requests/responses only if all prerequisites are satisfied.

Modules that implement the services are preloaded in main memory, but only the modules that correspond to the requested services will be allocated. Each service is identified by a task-based service name, with a set of parameters that each user can modify according to his preferences.

The Deploy Module. Application deployment is an important system functionality that allows client anytime-anywhere access to provided services and applications. It consists of an automatic modules generation process that implements intermediary services starting from simple Perl files (i.e. with `.pl` extension).

If the programmer follows some easy and well defined rules (templates), his Perl programs can be used to build the *body core* of the service modules to be loaded into Apache. In detail, the *Deploy Module* is an Apache module activated via URI. It uses an XML file (.reg) that defines the parameters needed in the dynamic creation and installation of the service.

In order to make a service available and to show its system properties to the users, is necessary adding it to the services' list. This happens inserting an HTML file in the location designed to contain all services. The HTML file contains the service name, the activation parameters and other values characterizing the service.

5. Functionalities

Here we describe some of the functionalities that are actually implemented by our architecture. In particular, what we have addressed in the current version of our framework is personalization, programmability, efficiency and easiness of management.

User & Profiles. User profiling is becoming more and more important in the future with heterogeneous devices used to access all kinds of information and services. Within ubiquitous computing user profiling and their management is a valid research subject, since it is the only efficiently way to provide personalized content for each type of mobile and ubiquitous client. The devices that characterize the ubiquitous Web show significant differences regarding the storage, display, processing power and connectivity capabilities. This drives the providers to offer adapted contents. That is, when a user connects with a cellphone he may be given black and white images or not given images at all, according to his preferences.

All the user preferences are kept in a user profile. A single user may have various profiles, according to the fact he may use many devices and with a click he can switch from a profile to another.

When a new user is added to the system, a default profile is automatically generated. When the user logs on the proxy for the first time, he has to modify the default profile, in order to capture his own needs. This can be accomplished by filling an on-line form. In the same way the user is also able to create new profiles, to modify or delete existing ones, to change the current profile and password. The form is generated on the fly by parsing several files that refer to each module and filling the fields with the current user parameters. In this way the user can see his current choices and can change them easily.

In our implementation, all information is coded in XML. Each piece of information translates in a tuple like (parameter name, parameter value).

HTML/HTTP Parsing Libraries. This library, implemented in Perl, realizes an efficient parsing of HTML pages. Several methods have been provided to programmers, such as methods for searching all links (HTML A tag), images, scripts, etc. in a Web page.

HTMLParser is a real-time parser for HTML. It is designed to be used as a base class for Perl modules in order to add the required functionalities.

The two fundamental cases handled by the parser are the extraction and the transformation of the HTML stream.

The extraction allows to take very useful information from an HTML page, such as: text extraction, to be used, for example, for text search engine databases; link extraction, for crawling through Web pages or harvesting email addresses; resource extraction, like collecting images or sound embedded in HTML pages; link checking, that is ensuring links are valid; site monitoring, that is checking for significant page differences i.e. beyond a simplistic `diff`.

The transformation includes all processing where the input and the output are HTML pages. Some examples are: URL rewriting, that is modifying many or all links on a Web page; site capture, e.g. moving content from the Web to local disk; censorship, that is removing offending words and phrases from Web pages; HTML cleanup, correcting erroneous or non-standard Web pages; ad removal, by eliminating URLs that reference advertising.

Image filter Libraries. PerlMagick is an objected-oriented Perl interface to ImageMagick. It is used to read, manipulate, or write an image or image sequence within Perl scripts. In our architecture we use PerlMagick to realize intermediary services working with images, an example is the FilterImg Service. It provides functionalities that match both device limitations and user's preferences This service retrieves the image from the Web, and performs some adjustments, such as changing colors and contrast, rotating, cropping, and/or resizing.

Deployment. The deployment of services simply occurs by uploading a Perl program by using an HTML form. It allows a quick and effective life-cycle management of the services, since a service can be developed offline, as a traditional Perl program, accessing locally stored HTML files that act as testbed for the filtering that is required, and, when ready, the Perl program can be simply deployed on the proxy-applications server.

Module generation is critical since it requires that the input Perl file is written according to a template. Once the module is ready to be used, it must be loaded into Apache, and a new information (LoadModule <MyApache::ModuleName>) must be added into the Apache configuration file (`httpd.conf`). After Apache server restarts, the added service is available for dynamic invocation by the FilterPlugin Module according to user's profile and service's constraints.

6. Conclusions

In this section we conclude with a comparison of our work with relevant literature. The objectives of our project, in this phase, were (a) to ensure programmability (b) efficiency, (c) lifecycle support and (d) easy management of users' profiles. In literature, several highly programmable proxy systems can be found, such as, e.g., WBI [6], SEcS [13] and ALAN [12, 18]. Unfortunately, all of them are based on Java environments which, in terms of efficiency, are not as efficient as the highly optimized framework that we used (i.e., Apache + mod_perl). In spite of not having conducted formal benchmarks, the clear perception is that our services are significantly more efficient than other (Java based) environments. At the same time, our framework preserves the programmability that is offered by the other systems (see Sec. 5). In fact, several proof-of-concept services have been realized and deployed on the architecture and the preliminary feedback on usability and efficiency is very positive. Finally, all the other proxy environments do not support lifecycle management of proxy applications; users' profiles management is also not offered: any application has to manage its own profiles (if necessary).

The next steps in our project will be tackling scalability (i.e. porting the server on a cluster in such a way that it is transparent both to users and to the service programmer) and persistence (ensuring a persistent status to services that is easily usable to service programmers).

The overall objective is to provide a framework that offers to intermediary services the same robustness, efficiency and quick prototyping provided by the application server in the world of distributed computing. The role of such a framework can be crucial to safely and easily employ intermediaries into a smooth mobile/fixed-network integration of services (see Fig. 2).

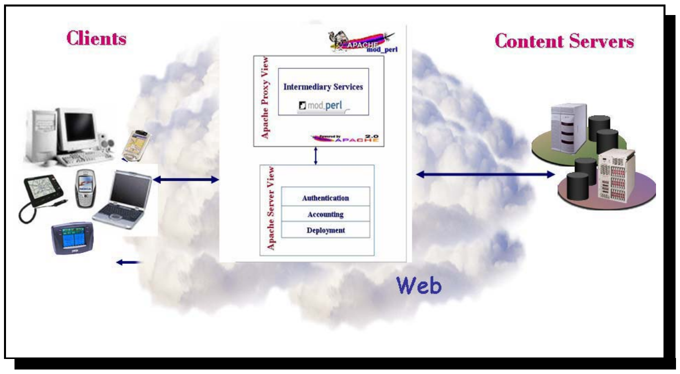

Figure 2. How a proxy-applications server lies between heterogeneous terminals and content providers.

Acknowledgments: The authors gratefully acknowledge several stimulating discussions with Michele Colajanni. This research work was financially supported by the Italian FIRB 2001 project number RBNE01WEJT "WEB–MiNDS" (Wide-scalE, Broadband MIddleware for Network Distributed Services) `http://web-minds.consorzio-cini.it/`.

References

[1] The Apache Software Foundation. http://www.apache.org.

[2] S. Ardon, P. Gunningberg, B. LandFeldt, M. P. Y. Ismailov, and A. Seneviratne. MARCH: a distributed content adaptation architecture. *Intl. Jour. of Comm. Systems, Special Issue: Wireless Access to the Global Internet: Mobile Radio Networks and Satellite Systems.*, 16(1), 2003.

[3] AT&T Labs-Research. iProxy: a Programmable Proxy. http://www.research.att.com/sw/tools/iproxy/.

[4] A. Barbir, E. Burger, R. Chen, S. McHenry, H. Orman, and R. Penno. Open Pluggable Edge services (OPES) Use Cases and Deployment Scenarios, April 2004. http://www.ietf.org/rfc/rfc3752.txt.

[5] A. Barbir, R. Penno, R. Chen, H. Hofmann, and H. Orman. An Architecture for Open Pluggable Edge services (OPES), August 2004. http://www.ietf.org/rfc/rfc3835.txt.

[6] R. Barrett and P. P. Maglio. Adaptive Communities and Web Places. In *Proceedings of* 2^{nd} *Workshop on Adaptive Hypertext and Hypermedia, HYPERTEXT 98.*, Pittsburgh (USA), 1998. ACM Press.

[7] R. Barrett and P. P. Maglio. Intermediaries: An approach to manipulating information streams. *IBM Systems Journal*, 38(4):629–641, 1999.

[8] E. Brewer, R. Katz, E. Amir, H. Balakrishnan, Y. Chawathe, A. Fox, S. Gribble, T. Hode, G. Nguyen, V. Padmanabhan, M. Stemm, S. Seshan, and T. Henderson. A Network Architecture for Heterogeneous Mobile Computing. *In IEEE Personal Communication Magazine*, 5(5):8–24, October 1998.

[9] M. G. Calabrò, D. Malandrino, and V. Scarano. Group Recording of Web Navigation. In *Proceedings of the HYPERTEXT'03*. ACM Press, August 2003.

[10] A. Fox, Y. Chawathe, and E. A. Brewer. Adapting to Network and Client variation using active proxies: Lessons and perspectives. *IEEE Personal Communications*, 5(4):10–19, 1998.

[11] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Peach, A. Luotonen, and L. Stewart. "HTTP Authentication: Basic and Digest Access Authentication"., June 1999. RFC 2617.

[12] M. Fry and A. Ghosh. Application level active networking. *Comput. Networks*, 31(7):655–667, 1999.

[13] R. Grieco, D. Malandrino, and V. Scarano. "SEcS: Scalable Edge-computing Services". In *Proceedings of the* 20^{th} *Annual ACM Symposium on Applied Computing (SAC 2005).*, March, 13 -17. 2005.

[14] M. Hori, G. Kondoh, K. Ono, S. Hirose, and S. Singhal. Annotation-Based Web Content Transcoding. In *Proceedings of the* 9^{th} *International World Wide Web Conference*, Amsterdam (The Netherland), 2000. ACM Press.

[15] Web Based Intermediaries (WBI). http://www.almaden.ibm.com/cs/wbi/.

[16] R. H. Katz, E. A. Brewer, E. Amir, H. Balakrishnan, A. Fox, S. Gribble, T. Hodes, D. Jiang, G. T. Nguyen, V. Padmanabhan, and M. Stemm. The Bay Area Research Wireless Access Network (BARWAN). In *Proceedings of the* 41^{st} *IEEE International Computer Conference*, page 15. IEEE Computer Society, 1996.

[17] A. Luotonen and K. Altis. World-Wide Web proxies. *Computer Networks and ISDN Systems*, 27(2):147–154, 1994.

[18] G. MacLarty and M. Fry. Policy-based content delivery: an active network approach. *Computer Communications*, 24(2):241–248, 2001.

[19] mod_perl. http://www.perl.apache.org.

[20] The Perl Programming Language. http://www.perl.com.

[21] C. Rao, Y. Chen, D.-F. Chang, and M.-F. Chen. imobile: A proxy-based platform for mobile services. In *Proceedings of the First ACM Workshop on Wireless Mobile Internet (WMI 2001).* ACM Press, 2001.

[22] December 2004 web server survey. http://news.netcraft.com/archives/web_server_survey.html.

[23] IBM Websphere Transcoding Publisher. http://www-3.ibm.com/software/webservers/transcoding.