

Intermediary Infrastructures for the WWW*

Marios Dikaiakos

Department of Computer Science

University of Cyprus

PO Box 20537, Nicosia, Cyprus

mdd@ucy.ac.cy

Abstract

Intermediaries are software entities deployed on Internet hosts of the wireline and wireless Web that intervene to the flow of information from clients to origin servers of the WWW. Intermediaries represent a useful abstraction for the design and study of emerging software infrastructures for “next-generation” Web services. Their importance is increasing with the increasing demand for personalization, localization, and support for ubiquitous access over different physical media and protocols. It becomes apparent that intermediary systems will permeate the Internet infrastructure in the near future. In this paper, we present an overview of a wide range of systems that can be described as intermediaries, classifying them in a number of broad categories according to their basic functionalities. Going beyond simple WWW proxies, we examine the requirements arising from the need to support personalization, mobility and ubiquity under high loads. We identify and refine a set of important properties and characteristics of intermediary systems. Based on these properties, we introduce a detailed taxonomy of characteristic systems and identify a number of key component of emerging intermediary infrastructures.

1 Introduction

The World-Wide Web has become the prevailing paradigm for information dissemination on Internet and the driving force behind the popularity of Internet services. The tremendous success of the Web is changing the basic model of Web-service provision. Typically, a Web

*This work was supported in part by the Research Promotion Foundation of Cyprus under grant PENEK-No 23/2000.

service is established upon a *Web-server*, i.e., a process running on a well-defined host and providing information to *Web clients* over a specific communication protocol like HTTP. This client-server model is being subsumed by a model entailing a fully distributed and dynamic web of interacting servers and software entities, deployed throughout the Internet infrastructure [53, 27]. Already, a large number of tools, systems, services and infrastructures have been proposed in this context. These systems can be collectively described as *enhanced proxies* or *intermediaries*.

We adopt the definition of intermediaries as *software entities that intervene to the flow of information from clients to origin servers at the application level of the WWW, deployed on Internet hosts of the wireline and wireless Web, between origin servers and client systems* [8, 36]. Intervention varies from simple relaying and caching performed by Web proxies and proxy caches, to complicated transformations, such as filtering, indexing, and transcoding. Intermediary systems fall under the general term of “middleware,” as they provide a “reusable and expandable set of services and functions, commonly needed by many applications to function well in a networked environment” [3].

The importance of intermediary systems is becoming evident with the expansion of the Web and the emergence of mobile and thin clients as alternative devices for Web access. Furthermore, intermediaries represent a useful abstraction for designing, developing and analyzing the emerging software infrastructures for “next-generation” Web services, which provide personalization, customization, localization, and support ubiquitous access from various terminal devices over different physical media and protocols.

In this paper, we examine intermediaries for Internet and the World-Wide Web, going beyond “traditional” systems with minimal functionality and very wide adoption, i.e., proxy servers for the WWW like Harvest [11] and Squid [50]. We present an overview of characteristic intermediary systems introduced to cope with end-user information overloading, support for mobile Web access, personalization, and infrastructural support for ubiquitous services. Furthermore, we identify families of parameters that can be used in the classification of current and future systems.

The remaining of this paper is organized as follows: Section 2 presents a number of intermediary systems deployed near Web servers in order to improve their performance. Section 3 presents a brief overview of traditional Web proxies. Section 4 focuses on “notification

systems” for the World-Wide Web. Section 5 examines the challenges that mobile devices and wireless connections place upon intermediary systems, as well as early approaches dealing with personalization and wireless-service provision. Section 6 discusses emerging intermediary infrastructures for ubiquitous Internet services, which incorporate ideas and approaches from proxy caches, notification systems, and wireless intermediaries. In Section 7 we present a taxonomy of intermediary systems and classify characteristic systems accordingly. We conclude in Section 8.

2 From Static to Dynamic HTML Content

The architecture of the Web is based upon the traditional client-server model of distributed computing. According to this model, information resources are placed on Web servers that act as repositories of multimedia content. A Web server “listens” to its Internet connection and accepts incoming requests asking for information resources that are installed in its repository. The server retrieves the requested resources and returns them to the client that issued the request. At the client side, a user employs Web-client software (typically a Web browser) to formulate requests for information resources, identified by their location in a particular Web server. Upon arrival of the requested resources, the browser undertakes their presentation at the client’s screen. Communication between Web servers and browsers is conducted according to the HTTP protocol, which runs on top of TCP/IP. Typically, information placed on Web servers is organized and encoded in HTML format, which enables the logical interconnection of resources with hyperlinks embedded in the content, thus turning the Web into a huge hyper-media web of interconnected information. In the early days of the WWW, a typical Web service was comprised of a Web server hosting a suite of “static” Web pages (see Figure 1) [58].

2.1 Access to information resources from enhanced clients

Part of the Web’s success is due to the popular acceptance of Web browsers as a ubiquitous interface for accessing Internet and Web-based services. Through their user-interface, browsers support the conceptual “navigation” of users in the hyper-media space of documents placed at the World-Wide Web, abstracting away the details of the underlying client-server interaction. At the outset, application developers sought to extend and customize



Figure 1: Static Web Content Provision.



Figure 2: Enhanced Clients (code-on-demand).

dynamically the user-interface of typical Web browsers, according to the interactivity requirements of particular applications. Therefore, the paradigm of navigation through static HTML pages has been expanded to incorporate the extra interactivity requirements of particular applications, with technologies like JAVA Applets and client-side scripting [52].

Under this scheme, when a user accesses a particular service on the Web, its browser dynamically downloads and starts interpreting a compiled program that enhances the interaction between the client and the server (see Figure 2). This approach supports a richer interaction between users and services, and increases the flexibility of Web-service provision by allowing service-providers to ship part of the service computation from the server to the client. This comes, however, at a significant cost due to the existence of a variety of non-compatible browsers, and clients differing in Operating System, supported standards, language, and computing capabilities.

2.2 Dynamic-content provision and caching

The success of the Web-browser as a ubiquitous gateway to Internet resources created the need to expand the information resources available on the Web from collections of static Web pages to dynamic content produced by software applications, databases, legacy systems, etc.



Figure 3: Dynamic Web Content Provision.

This need is addressed by technologies that enable the use of Web servers as front-ends to back-end applications (see Figure 3); for instance, with server-side scripting following the CGI standard [58, 52], Active Server Pages (ASP) by Microsoft, Java Server Pages (JSP), Java servlets, XML-engines [40], etc.

The support for dynamic content has brought to the Web an expanding variety of services touching all aspects of human activity: personalized information provision, electronic commerce, collaboration, communication, entertainment, etc. On the other hand, the increase of dynamic content on the Web has raised numerous research questions related to the high cost of service development and maintenance, the achievement of high availability, incremental scalability, and effective load management. Furthermore, it has affected the latency experienced by Web users, since the creation and serving of dynamic content typically requires orders of magnitude more CPU time than that of static pages of comparable size [14].

To reduce Web-server loads and improve user-perceived QoS, several approaches can be applied. For example, generic techniques that distribute Web requests either to mirrored Web servers, which are geographically dispersed, or to multiple Web servers collocated on the same cluster and sharing a common file system. In these cases, request distribution can be achieved either at the TCP-router level or with round-robin DNS [14].

2.2.1 Web accelerators

An alternative solution to expedite the provision of dynamic data is *Web* or *HTTP acceleration* [14]. A Web accelerator is a streamlined system installed “in front” of a number of collocated Web servers. The accelerator receives and distributes requests to the Web servers, caching replies so that frequently requested pages are served from the accelerator’s cache rather than the Web server. The implementation of a Web server accelerator runs

under an embedded operating system, which reduces the overhead of TCP and operating-system buffering [34]. Overall performance is improved because of the reduction of direct hits to origin servers and the serving of content (static and dynamic) from the streamlined accelerator cache.

2.2.2 Composition of dynamic content out of fragments

Another approach for improving the performance of dynamic-content provision is the construction of Web pages from individual information fragments, which are computed from back-end applications and/or databases. It is based on the premise that Web pages can be decomposed into a hierarchy of complex and atomic fragments. Atomic fragments are “parts of a Web page that change together” and reside at the leaves of this hierarchy [15]. Fragments can be updated on-the-fly or via update propagation mechanisms and stored in a software cache until their expiration. The basic idea behind this approach is the observation that the overhead of composing an object from simpler fragments is usually minor if compared to the overhead of constructing the whole object from scratch [15].

This approach was proposed by IBM Research and used in a number of very popular Web sites including the 2000 Olympic Games Web. The system works by taking objects from one or more sources, constructing pages and writing them to one or more “sinks,” which can be the file systems of local or remote Web servers or Web accelerators. Efficient construction of Web pages and consistency with back-end-database updates are achieved via a persistent *object dependence graph* structure and a trigger monitor integrated in the publishing system.

The notion of the object dependence graph is also incorporated in the *Accessible Business Rules* framework (ABR) of IBM’s Websphere [17]. This is a middleware system which presents application developers with business rules (ABR’s). ABR’s are persistent objects that encapsulate code, which implements a variable behavior, and attributes defining the business context in which this behavior applies. Variable behavior is defined through “variability” or “decision points” corresponding to code which decides at run time the particular business logic to be executed via a query to a back-end database. ABR’s represent a useful abstraction for high-level programming of Web-based applications serving dynamic content. The performance of content generation, however, faces significant overhead due to the high

cost of back-end-database connection and query execution [17].

This problem is addressed by caching query results (i.e., dynamic content-fragments) in a general-purpose software cache, and maintaining the consistency of cached data with the back-end database through update propagation mechanisms. Furthermore, by making the middleware servers multithreaded and replicated [17]. The ABR framework is organized as a three-tier architecture. At the middle layer, application servers provide support for managing business rules and caching query results. The bottom layer hosts data and resources (back-end database) and the top layer deals with presentation of dynamic content and user interaction.

2.2.3 Weave

Another approach for managing dynamic content was introduced in the context of the *Weave* project at INRIA, which developed a system for specifying and generating data-intensive Web sites [56]. The Weave project separated the three key concerns of Web-site management which are interwoven in many systems: Web-site structure and content, Web-page presentation and graphical style, and implementation, i.e., content-assembly and HTML creation.

Structure and content are specified in the WeaveL language, which provides a declarative approach for specifying the mapping between raw data (captured in databases) and the logical model of a dynamic Web-site. Each WeaveL program consists of a set of “site class” specifications. A site class represents a set of homogeneous pages in a Web site and determines the hyperlinks emanating from its instances (i.e., its Web pages), the parameters that identify instances of the class, and the SQL queries whose results provide all possible values for the site-class parameters.

The execution of a WeaveL program translates into a series of database queries that are executed on the underlying database and produce XML fragments, which are subsequently translated into HTML. XML to HTML translation is conducted by XSLT programs that represent the specification of the Web-site graphical presentation design. WeaveL-program execution is essentially a *data materialization process*, which is driven by a declarative specification of runtime and caching policies described in the *WeaveRPL* language. WeaveRPL provides abstractions for specifying complex runtime policies for database materialization,

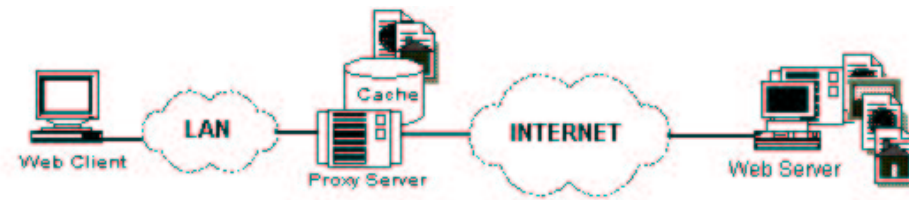


Figure 4: Proxy Servers.

such as the timing of materialization, the storage of materialized intermediate results from SQL-query execution and XML-fragment generation, and the policy for database-update propagation. The goal of the WeaveRPL approach is to enable the reuse of intermediate results, through caching as a means to improve Web-site performance [56].

The architecture of Weave is based on a 3-layer design. At the middle layer lies a customizable cache system, which caches database data, XML fragments and HTML files. This is connected to the underlying database, XML and HTML repositories on the back-end, and to the interface to the Web/HTTP on the front-end [56].

3 Web Proxies

Network administrations and ISP's employ proxy servers to cope with increased Web traffic and optimize the use of their computing, storage and networking resources. Proxy servers intermediate between Web clients and origin servers (see Figure 4). Typically, they filter or redirect HTTP requests either to enable caching or to enforce a security policy, relaying requests through firewalls. Web proxies can be easily configured to function as Web traffic caches, storing relayed responses locally and serving identical requests that come from their internal networks. Therefore, whenever a user seeks a particular Web page, instead of automatically connecting to the origin Web server designated by the corresponding URL, his Web-client software checks if the Web page is cached in its vicinity. In that case, the user can be provided with a cached copy of the page.

In the complex hierarchy of wide-area and local networks that connect Web clients to information sources, there are several places where proxy servers can be deployed: in the relay connecting a local network to an institutional network, a local ISP network to a regional-provider network, multiple regional networks to a national backbone, etc. The

potential gains from Web-caching adoption are obvious: caching popular documents on a local network reduces significantly the incoming traffic and the load imposed on origin Web servers. Furthermore, users experience much shorter response times when receiving documents from a nearby cache than from a distant origin server [58].

The proxy-caching paradigm has been expanded into complex caching infrastructures deployed on Internet and offering hierarchical [16, 51] and co-operative [54] caching, dissemination of Web resources through terrestrial [9] and satellite [44, 19] links, as well as more generic, dissemination-based systems [4]. Lately, the efforts to distribute dynamic and interactive multimedia content over the Internet has turned the industry's focus on *Content Delivery Networks* or *Content Distribution Networks* (CDNs), that is, specially-tuned overlays to the Internet, such as Akamai's network [1]. CDNs "leverage a strategically-arranged set of distributed Web-caching, load-balancing and Web-request redirection systems," seeking to push content towards the network's edge, where most end-users reside [47].

4 Notification Systems

Notification systems are intermediaries that monitor changes in information sources on behalf of subscribed users. Whenever an update in the content of a monitored source is observed, the notification service evaluates the relevance of this change with respect to stored user profiles, and notifies accordingly interested subscribers.

4.1 SIFT

One of the first examples of an intermediary notification system for Internet is SIFT, the *Stanford Information Filtering Tool* (SIFT) [57]. SIFT has been developed to provide large-scale information dissemination services to users that subscribe their interests to SIFT servers.

At subscription, a user provides the system with an information-retrieval-style profile and additional parameters that declare the desired frequency of updates and expected amount of information to be received. SIFT employs the NNTP protocol to collect news articles published over USENET News. Collected content is indexed and filtered according to profiles registered in a SIFT database. Based on filtering results, SIFT produces notifications, which are delivered to interested subscribers via email.

4.2 AIDE

Another example of a notification system for WWW resources is AIDE, the *AT&T Internet Difference Engine*. AIDE was designed to archive and handle multiple versions of changing WWW resources [21]. AIDE supports recursive tracking and differencing of Web pages and their descendants. Special emphasis is placed on the graphical viewing of changes taking place between subsequent versions.

AIDE is comprised of a centralized notification server, a version archive and a difference engine to find and display changes of pages on the WWW. Subscribers register in AIDE the list of URLs they wish to track, and a few parameters configuring the degree of desired notification. Alerts about changes in tracked pages reach the users via email; additional access is provided via the Web.

4.3 IBM's Grand Central Station

The issue of profile-driven filtering intermediaries was addressed in the context of the Grand Central Station project of IBM Almaden [45, 35]. The GCS system sought to expand and improve earlier efforts dealing with Webcasting-service provision. To this end, the project focused on: (i) collecting content from different Internet sources (Web, news, email) and making it available through Webcasting channels; (ii) introducing the *Grand Central Station Profile Language* (GCSPL), a boolean-structured predicated language for specifying personalized channels of information, and (iii) building an engine to process user profiles expressed in the GCSPL. The profile engine of GCS coalesces different user profiles into a single, boolean-tree representation with internal nodes corresponding to boolean operators and leaf-nodes storing the predicates of different profiles. A number of algorithms have been proposed for “resolving” this profile and generating content for personalized channels [35]. Experiments with this approach showed that its performance scales well with increasing number of users subscribing to different channels.

Both SIFT, AIDE and the GCS are “server-based” and centralized tools (according to the taxonomy in [21]), since they rely on monitoring software running on a centralized server and not on a user’s machine.

4.4 FIGI

An approach for building decentralized, distributed intermediaries has been explored with the *Financial Information Gathering Infrastructure* (FIGI), which is a notification system that retrieves, caches, filters and serves financial data [18]. The FIGI server is organized as a distributed, two-tier architecture. The first tier is comprised of three servers that receive user input and act upon it: a Login Server, an Alert Server (Alerter), and a Profile Registry. The Login Server deals with user authentication. It notifies other FIGI components of user connections. Upon user connection, the Alerter starts retrieving information pertinent to the particular user profile from the FIGI Cache. Retrieved information is forwarded to the user through her personalized Web interface. The Profile Registry is a server where users can register or update their interests.

Similarly to SIFT [57], a FIGI user profile is a set of long-term, continuously evaluated queries. These queries may include typical queries to Web databases, HTTP requests for World-Wide Web resources, access to general-purpose Search Engines or Subject Cataloging Sites, subscription to Usenet News, etc. Each profile is annotated by the user with a number of *data* and *control* parameters. Data parameters are query arguments (e.g., a stock symbol of interest), whereas control parameters determine the frequency of query execution, the expected amount of information gathered from queries (e.g., summary vs. full results), the priority of notification for a given query, etc.

The second tier of FIGI is comprised of a Profile Database, the FIGI Cache, and a Proxy Server. The Profile Database stores user profiles. The Proxy Server scans continuously the Profile Database, schedules and issues requests for information to Wide-area network services. The results of these requests are tagged with information denoting the corresponding user-profile and are stored in the FIGI Cache. When a user logs in to FIGI, these results are extracted from the Cache by the Alerter and forwarded to him. FIGI-modules are developed on top of the Concordia Mobile-Agent middleware [55], as stationary or mobile agents.

5 Intermediaries for Mobility and Ubiquity

As PDAs, thin clients, mobile phones and the like become more popular, the heterogeneity of client devices and the capacity mismatch between clients and servers are expected to grow [22]. To cope with these trends, software infrastructures for Web services have to:

- (a) Optimize client-server communication over the wireless medium;
- (b) Support seamless access from a variety of devices;
- (c) Customize content to adapt to different terminal devices;
- (d) Enable the provision of multiple formats (HTML, WML, XML) to the same device over the same link;
- (e) Support both synchronous (on-demand) and asynchronous modes of interaction with users, thus coping with frequent disconnections of wireless connections and user mobility.
- (f) Optimize the amount of useful content that reaches users through client devices with limited resources and restricted interfaces, by enabling service personalization, localization and filtering of information.
- (g) Guarantee high availability and robustness, as well as incremental performance and capacity scalability with an expanding user base.

Clearly, these requirements cannot be met by traditional proxy infrastructures.

5.1 Copying with wireless connectivity

A number of projects, products and protocols have addressed issues arising in the presence of wireless connectivity by deploying proxy-agents on the wireline network and/or the mobile host [30, 38, 46]. Such agents mediate between origin servers and mobile terminals, optimizing communication, dealing with disconnections, etc. Two characteristic examples are the WebExpress system of IBM and the WAP technology.

WebExpress is a characteristic example of a proxy-based approach, where the proxies optimize wireless communication [30, 31]. *WebExpress* is a *client/intercept*-based system that optimizes Web browsing on resource-poor clients connected over wireless connections. To this end, it dispatches two agents: the *Server Side Intercept*, on the wireline network and the *Client Side Intercept* on the end-user's mobile device. The two agents run a stripped-down version of the HTTP/1.0 protocol, optimized for wireless communication. Both agents provide caching of content, and run a simple differencing protocol between the client-side and the server-side cache to optimize the provision of dynamic content.

A similar approach for optimizing Web-access provision over wireless links has been adopted by the architecture of the *Wireless Application Protocol* (WAP), a layered suite

of standards defining how wireless devices communicate over the wireless network [23, 46]. WAP is the protocol adopted by numerous mobile-telephony operators for providing Internet connectivity to cellular GSM and GPRS mobile phones. Consequently, it is one of the major vehicles for establishing mobile services over wireless links [22]. A WAP-enabled handset can establish a connection to a WAP-compliant wireless data infrastructure, request content and services, and present them to the user via a WAP micro-browser running on the handset.

One of the layers of WAP corresponds to the *Wireless Session Protocol* (WSP) that manages the establishment of a long-term, wireless session between a WAP micro-browser running on a mobile phone, and a *Wap gateway* running on the wireline network. The WAP gateway is an intermediary installed “near” the local base station of the wireless devices it serves [46]. The WAP gateway manages WSP sessions between wireless devices and the wireline network: it optimizes communication, provides header caching, supports session resumption following disconnections, etc. Furthermore, it translates WSP requests received over the air into HTTP requests sent over the wireline network, and HTTP replies received from Internet into WSP packets sent to the wireless device [46].

5.2 Single-proxy vs. End-to-end Approaches

To support personalization, content customization, ubiquity and mobility, proxy functionality has to be extended over the optimization and management of wireless communication. This raises the need for increased computational resources becoming available at the proxy’s host in order to maintain its high-performance operation. A single-proxy approach would deploy an enhanced proxy at some host residing in the path from the server to the mobile client, typically at the mobile support station. In the context of ad hoc networks supported by technologies like Bluetooth, however, it is questionable whether mobile support stations would have the computing and storage resources to host such proxies [32].

End-to-end solutions represent a possible alternative, with origin servers adapting content on-the-fly according to the terminal device and network connection involved in a user session [32] or user profiling [39]. Under the end-to-end approach, however, adaptation software has to be inserted at each origin server. Consequently, software updates have to be propagated to all origin servers whenever new terminal devices and content-encoding protocols emerge. Furthermore, on-the-fly adaptation of content can be very time-consuming,

leading to a deterioration of user experience.

Suggested approaches for coping with these problems propose the deployment of proxy servers at powerful machines near the origin servers [24, 28] or the combination of end-to-end with proxy-based solutions [32, 29]. In the following sections we examine proxy-based and hybrid approaches, which have been adopted and used in commercial products and services mainly targeting the information and connectivity requirements of handheld devices.

Handspring's Blazer is a general-purpose light-weight browser developed for handhelds running the PalmOS Operating System [28]. The end-user can fetch content on his handheld by typing a URL of a Web or WAP page on the Blazer's GUI. Blazer accepts content in multiple formats (HTML, WML/HDML, cHTML), implements content streaming and progressive rendering for faster presentation of Web pages, and supports icons and colors. All communication between the Blazer browser and origin servers passes through a proprietary proxy server that customizes content. This proxy employs the appropriate protocol (HTTP or WAP) in order to fetch content from the origin server of choice, transcodes the retrieved content into a stripped-down HTML format accepted by the browser, removes unnecessary or unsupported tags, reduces the size of Web pages and images, and adapts a Web page's size to the screen of the handheld [28].

Web clippings is an architecture proposed and implemented by Palm Inc. to support Web content retrieval from Palm devices connected via wireless links [29]. To fetch content from a Web site via the Web clipping system a user has to employ a *Web clipping application* developed for that particular site and pre-installed on a handheld device. Web clipping applications are developed with a proprietary tool of Palm, which creates a site-specific interface with hard-coded links and translates it into a Palm-specific format before installing it on the handheld. The clipping application works as a special-purpose, site-specific browser: it provides the interface for navigating the content of a particular Web site, translates user clicks into queries sent to the Internet and displays fetched content (the "clipping") on the handheld screen.

Typically, clippings are small, "Palm friendly," HTML 3.2 pages generated dynamically by CGI scripts or application programs, deployed for that reason in an origin Web server. A proxy server installed at the Palm.Net data center mediates the interaction between the local base station of the Web clipping client, and that client's corresponding origin server.

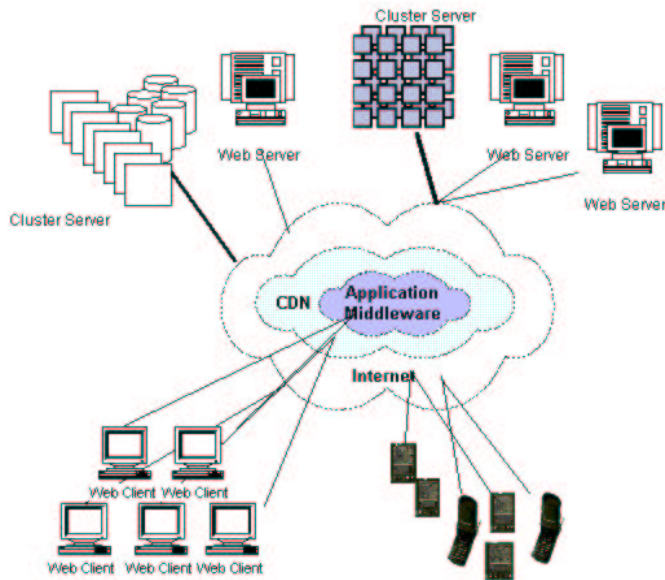


Figure 5: The context of next-generation Internet services.

The proxy receives requests from the PDA that are encoded into a proprietary, compressed format (the Palm Query Format) via the UDP protocol. Subsequently, it translates them into standard HTTP requests and dispatches them to the “Palm friendly,” origin server. Upon the server’s reply, it receives the content in HTML 3.2, translates it into a compressed format known as Compressed Markup Language (CML) and wraps the content into a compressed “Palm Proxy Format.” Subsequently, the proxy sends the PPF file back to the clipping application via the handheld’s local base station [29]. The proxy server provides limited caching of static HTML resources and takes care of encryption-decryption, if necessary using SSL and the HTTPS protocol.

6 Intermediary Infrastructures

Performance scalability problems are expected to grow with the wide spread of Internet use: popular portals like Yahoo receive more than half a billion page-views per day; during major events, popular Web-site hit rates approach 1 million per minute [33]. High loads result to poor end-to-end performance and low Quality of Service [10]. Therefore, service infrastructures will have to accommodate millions of simultaneous end-users connecting from a large heterogeneity of end-user devices and resulting to highly bursty workloads.

At the same time, service infrastructures are required to sustain a very high throughput of service requests, support ubiquitous access through different client-terminals, exhibit 24x7 availability and robustness, be scalable in terms of capacity and performance [13, 37].

These requirements suggest the shift of computation, storage and complexity from centralized proxies, mobile devices, and mobile base stations into the networking infrastructure, in order to achieve performance scalability, better sharing of resources, higher cost efficiency and a streamlining of new service provision [12]. Such an approach would result to the deployment throughout the network of distributed, programmable and possibly mobile *intermediary servers*, mediating between primary information sources and various client systems (see Figure 5). In this section we describe a number of systems seeking to comply with these characteristics.

6.1 WBI

A programmable framework for building various intermediary services is the *Web Browser Intelligence* or *WeB Intermediaries* (WBI) by IBM Almaden [6, 7, 36]. WBI is a programmable proxy server designed for the development and deployment of intermediary applications. The design of WBI is based on the notion of “information streams” that convey data from information providers (e.g., a Web server) to information consumers (e.g., a Web browser), and can be classified into unidirectional and bidirectional messages or transaction streams [8].

WBI provides an approach to develop intermediaries for the unidirectional transaction streams of the WWW. To this end, it provides five basic building blocks: *request editors*, *generators*, *document editors*, *monitors* and *autonomous functions* [7]. Request editors receive and modify requests before forwarding them towards their destination. Generators are abstractions of information sources that produce documents in response to requests. Editors receive and modify responses before passing them down to their destination. Monitors observe transactions without interfering. Autonomous functions run independently of information processing transactions and perform background tasks.

These blocks are called collectively *MEG's* (Monitor/Editor/Generator) and can be assembled together into *WBI plugins*, which are used to construct data paths that transform information flowing from origin servers to users and implement different application scenar-

ios. WBI constructs data paths dynamically, using a rule-based approach that determines which MEGs must be invoked and in what sequence.

WBI has been used to develop a number of applications such as a manager-repository for *cookies* and a Web-browsing service for mobile devices [7]. WBI plugins can be installed both on a client machine and on any other networked machine, possibly near an origin server. Multiple client-side and server-side WBI intermediaries can cooperate to establish one WBI service.

6.2 iMobile

Building mobile services from proxy components is the main goal of the *iMobile* project of AT&T Research [42]. The iMobile proxy maintains user and device profiles, accesses and processes Internet resources on behalf of the user, keeps track of user interaction and performs content transformations according to device and user profiles. The architecture of iMobile is established upon the infrastructure of *iProxy*, a programmable proxy server designed to host agents and personalized services developed in Java [43]. iMobile consists of three main abstractions: *devlets*, *infolets* and *applets*.

A devlet is an agent-abstraction for supporting the provision of iMobile services to different types of mobile devices connected through various access networks. A devlet-instance communicates with a device-specific “driver” that is either co-located in the iProxy server or resides at a remote mobile support station. This driver “speaks” the protocol of the mobile device’s access network and communicates with the devlet via TCP, in a scheme similar to the client-intercept-server model of WebExpress [31]. The devlet sends iMobile-specific commands to the iMobile server (the “let engine”), to invoke an applet representing the implementation of a particular iMobile service. The server’s output is encoded in a MIME type appropriate for devlet’s terminal device, and is passed back to the terminal device by the devlet.

The infolet abstraction provides a common way of expressing the interaction between the iMobile server and various information sources or information spaces (in iMobile terminology) at a level higher than the HTTP protocol and the URI specification. Different sources export different interfaces to the outside world: JDBC and ODBC for corporate databases, the X10 protocol for home networks, IMAP for email servers, etc. Finally, an

applet is a module that processes and aggregates content retrieved by different sources and relays results to various destination devices.

At the core of an iMobile server resides the “let engine,” which registers all devlets, infolets and applets, receives commands from devlets, forwards them to the right infolet or applet, transcodes the result to an appropriate terminal-device format and forwards it to the terminal device via the proper devlet.

6.3 The TACC Model and the Ninja architecture

The BARWAN project at UC/Berkeley developed an architecture for the seamless provision of services over an overlaid collection of heterogeneous wireless networks [12, 25]. The goal of the project was to support ubiquitous access of roaming users that access services via thin clients. BARWAN focuses on issues, such as: (i) dynamic adaptation to varying network conditions and client heterogeneity; (ii) optimizations of cross-layer hand-offs at the networking level; (iii) the infusion of agents in the network infrastructure to cope with problems arising from mobility, wireless connectivity, and device and network heterogeneity, and (iv) the exploitation of “soft” (non-critical) state to improve service performance.

An important component of the BARWAN system is its proxy architecture, which functions as a smart intermediary between traditional servers and heterogeneous mobile clients. The proxy design is organized in three layers. The lower layer, *Scalable Network Services* (SNS), seeks to guarantee high availability, scalability and robustness of the proxy. The top layer handles the actual presentation of data to the various client terminals. The middle layer, *TACC*, provides a programming model seeking to simplify service creation and deployment.

TACC is a general model for intermediaries, that provide *transformation*, *aggregation*, *caching* and *customization* of content [12, 25]. The transformation functionality deals with various changes done by the proxy to the content of a single data object, to achieve filtering, format conversion and compression. Aggregation enables the collection of data from different origin servers, and the combination thereof in a pre-specified way that adds value to the collected information. Caching provided by TACC allows for the caching of original Internet content, and of post-transformation data, aggregated information and intermediate results. Finally, customization of services according to user requirements is achieved via the

parameterization of services according to user preferences.

The main component of the TACC infrastructure is the *worker*. Workers are the building blocks of TACC applications. They are combined together according to a general programming model that enables the chaining of workers (similarly to the chaining of processes in a Unix pipe), and the invocation of one worker from another as subroutine or coroutine. TACC workers are instantiated in the context of TACC servers, which run on clusters of workstations and provide support for inter-worker calling and chaining API's. Besides a pool of available TACC-workers, a TACC server hosts a front-end module for interfacing with client systems, a customization database storing user profiles, a load balancing/fault tolerance manager and a system monitor [25].

The TACC infrastructure has been used to develop and deploy services providing Web access to resource-poor devices [24], combining searching and browsing facilities [12], providing distributed, collaborative repository access to digital music resources [25], etc.

An evolution of the TACC model is provided in the context of the Ninja project from UC/Berkeley [27]. This project seeks to develop a robust infrastructure for Internet-scale systems and services in Java. The architecture of Ninja consists of *bases*, *active proxies*, *units* and *paths*. Bases are scalable platforms designed to host Internet services. They consist of a programming model and I/O substrate designed to provide high-concurrency, robustness, and transparent distribution of data to cluster-nodes [26]. Moreover, they include a cluster-based execution environment (*vSpace*) that provides facilities for service component replication, load-balancing and fault-tolerance [27]. The programming model of Ninja consists of four *design patterns* that service programmers can use to compose different stages of a single service: wrap, pipeline, combine and replicate [27].

Active proxies are fine-grain intermediaries providing transformational support between Ninja services and terminal devices. Active proxies perform data distillation, protocol adaptation, caching, encryption, etc. Examples of active proxies include wireless base-stations, network gateways, firewalls, and caching proxies. In Ninja terminology, a path is a flow of typed data through multiple proxies across a wide-area network; each proxy performs transformational operations to adapt the data into a form acceptable by the next service or device along the path. Similarly to WBI plugins, Ninja-paths can be established dynamically. Finally, units are abstractions for the client devices attached to the Ninja

infrastructure, which range from PC's and laptops to mobile devices, sensors and actuators.

6.4 eRACE

The *extensible Retrieval, Annotation and Caching Engine* (eRACE) is a middleware infrastructure designed to support the development and deployment of intermediaries on Internet [20, 2]. eRACE is a modular, configurable and distributed proxy infrastructure that collects information from heterogeneous Internet sources and protocols according to XML-encoded *eRACE profiles*, registered within the infrastructure. Collected information is stored in a software cache for further processing, personalized dissemination to subscribed users, and wide-area dissemination on the wireline or wireless Internet.

eRACE supports personalization by enabling the registration, maintenance and management of personal profiles representing the interests of individual users. Furthermore, its structure allows the easy customization of service provision according to parameters, such as information-access model of choice (pull or push), client-proxy communication (wireline or wireless; email, HTTP, WAP), and client-device capabilities (PC, PDA, mobile phone, thin clients). Ubiquitous service-provision is supported by eRACE thanks to the decoupling of content publishing and distribution from information retrieval, storage and filtering. The eRACE infrastructure can also incorporate easily mechanisms for providing subscribed users with differentiated service-levels at the middleware level. Finally, the design of eRACE is tuned for providing performance scalability, which is an important consideration given the expanding numbers of WWW users, the huge increase of information sources available on the Web, and the need to provide robust services.

eRACE is organized as a two-tier architecture (see Figure 6). The first tier includes modules that manage services provided to users: the Service Manager, Content-Distribution Agents, and Personal Information Roadmap (PIR) Servlets. The second tier of eRACE consists of a number of protocol-specific *Agent-Proxies* like WebRACE, mailRACE, newsRACE and dbRACE that retrieve and cache information from the WWW, POP3 email-accounts, USENET NNTP-news, and Web-database queries respectively.

At the core of the second tier lies WebRACE, the Agent-Proxy that deals with information sources on the WWW, accessible through the HTTP protocols (HTTP/1.0, HTTP/1.1) [20]. WebRACE is developed in Java and consists of a Distributed Crawler [59],

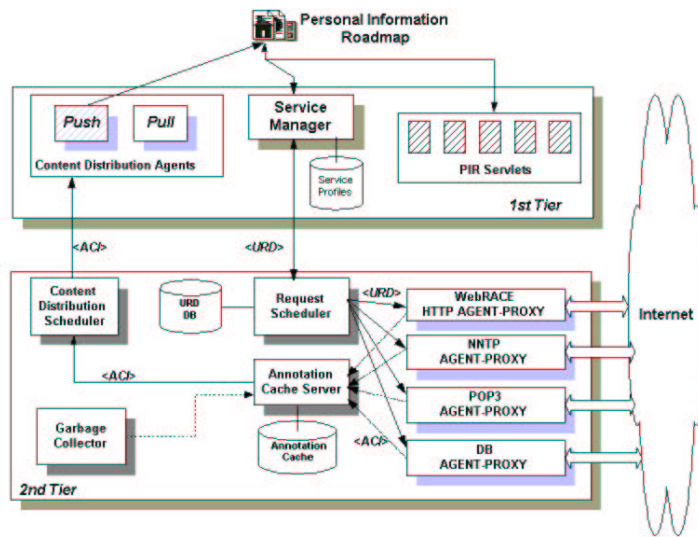


Figure 6: eRACE System Architecture.

an Object Cache storing multiple versions of retrieved resources, and an Annotation Engine that indexes collected resources, executes user-queries, and produces “user alerts,” encoded in XML. Other proxies have the same general architecture with WebRACE, differing only in the implementation of their protocol-specific proxy engines.

7 A Taxonomy of Intermediaries

In the previous sections we presented an overview of typical intermediary systems developed to provide various kinds of Web services over fixed and wireless connections. Besides their differences, all these systems exhibit many common features, which can be analyzed along three main dimensions:

1. The software architecture of the intermediary;
2. The interaction of the intermediary with clients and origin servers;
3. The functionality provided by the intermediary over and above simple client proxying.

Classification of intermediaries along these three dimensions can be further refined according to characteristics specifying them in more detail. These traits can be taken into consideration when classifying existing and emerging systems. Below, we examine the particular features used to describe and classify different intermediaries.

7.1 Software Architecture

An important aspect that characterizes the structure of an intermediary system is whether it is composed of a “centralized” software module running on one host, or of a number of distributed software modules residing at different hosts and communicating via message exchanges, distributed events, or shared memory. Distributed design has obvious advantages as it supports load distribution to multiple processors, scalability of performance, and improves system robustness and availability [13]. Another characteristic of the structure of an intermediary system is the location of its software modules, which can reside in front of origin servers, in intermediate hosts residing on the network between origin servers and clients, and on the client side.

Besides distributed design, of particular importance is the incorporation in an intermediary system of software-cache and high-performance-crawling modules. Software caches have already been identified as a necessary module in many middleware systems [17, 21, 59]. The importance of crawlers, on the other hand, is increasing with the remarkable increase of origin servers and the need to achieve resource sharing, cost efficiency and high-performance while reaping content from the Web on behalf of large user communities [59].

Many intermediaries act as “user-agents,” i.e. they collect and customize Internet content on users’ behalf. Users receive this content asynchronously, following a customization conducted according to situational parameters and user preferences. In these cases, information gathering and user notification could take place at different time scales, and the intermediary may retrieve different versions of the same resource between subsequent user accesses. Maintenance and management of these versions require particular care [21] and represent another feature for characterizing the design and architecture of intermediary systems.

Finally, an important aspect of recent intermediary systems is their support for configurability and/or programmability. This is necessary for using intermediaries as infrastructures upon which various services can be developed and deployed. Tuning of an intermediary infrastructure can be provided at different levels of abstraction and flexibility:

- Through configuration parameters, which are defined in the information architecture of an intermediary system and guide the information retrieval, transformation and dissemination that the system makes. Different configurations can result to new services

provided via the intermediary.

- With the employment of generic middleware platforms, such as Jini [49] and mobile agents [41], which provide the substratum for extending intermediary systems [18].
- Through API's, software libraries and design patterns developed for programming new services in the intermediary's context [27].
- With components that can be used as building blocks according to a higher-level programming model, combined in different ways and extended to define new services [6, 8, 42].
- With different combinations of the options described above.

The capability of tuning the behavior and services of intermediaries results to a separation of deployment from development issues and the reduction of programming effort and maintenance cost [48]. Tuning by configuration, however, does not usually offer the same scale of flexibility as a programming environment or API, unless configuration itself becomes quite complicated.

7.2 Interaction between Intermediaries, Clients and Origin Servers

Another important dimension in the classification of different intermediaries is their interaction with origin servers and client systems. Different interaction approaches can be characterized by the mode of communication, the access model, the communication protocol employed, and the supported media (wireline or wireless).

Typically, two modes of communication are employed by intermediary systems: *synchronous*, which is implemented via blocking messages or remote procedure calls, and *asynchronous*, which is implemented via non-blocking messages. Proxy servers and transcoding intermediaries typically perform their intermediation activities in a synchronous (on demand) manner: the intermediary is activated upon receipt of a user request, interacts with origin servers and returns a reply “synchronously,” while the user remains connected to the system. There is also a large number of “asynchronous” intermediaries, which perform operations on behalf of users on a longer-term basis, or perform complicated operations on-demand, providing users with a result at a later time [5].

	Squid	Palm Clippings	SIFT	AIDE	WBI	TACC	eRACE
<i>Software Architecture</i>							
Structure	centralized	central.	central.	central	distr.	distr.	distr.
Component Location	network	network & client	network	network	network client, server	network server	network
Caching	✓	limited	✓	✓	✓	✓	✓
Crawling support	-	-	-	✓	-	-	✓
Archiving	-	-	-	✓	-	-	✓
Programmability	-	-	-	-	✓	✓	✓
Configurability	-	✓	-	-	✓	✓	✓
<i>Client-Intermediary-Server Interaction</i>							
Proxy-Server Protocol	HTTP	HTTP	NNTP	HTTP	HTTP	HTTP	HTTP, NNTP SMTP
Client-Proxy Protocol	HTTP	UDP and compressed msgcs.	SMTP HTTP	HTTP SMTP	HTTP	wireless protocols	HTTP, SMTP GMS/SMS
Medium	wireline	wireless	wireline	wireline	wireline wireless	wireless wireline	wireline wireless
Access Model	pull	pull	pull/push	pull/push	pull/push	pull	pull/push
Communication Mode	synch.	synch.	asynch.	asynch.	synch. asynch.	synch.	asynch. synch.
<i>Intermediary Functionalities</i>							
Customization	-	-	-	-	✓	✓	✓
Filtering	-	-	✓	✓	✓	-	✓
Annotation	-	-	-	✓	✓	✓	✓
Transcoding	-	✓	-	-	✓	✓	✓
Aggregation	-	-	✓	✓	✓	✓	✓

Table 1: Important Features of Intermediary Systems.

The access model depends on the part that initiates an interaction and can be characterized either as *push*-based or *pull*-based. Interaction on the WWW is essentially pull-based. A number of research and commercial systems, however, have investigated the advantages of push-based approaches in wide-area networks. Most intermediaries work as user-agents, being constantly connected on the fixed network, collecting and filtering information on behalf of users. Therefore, intelligent content-push from intermediaries to their users can be combined with synchronous content provision, employing various protocols enabling information push, such as SMS/GSM and Java RMI over TCP/IP.

Most intermediary servers on the Web “speak” the HTTP protocol and connect to Web servers and download information. Given, however, the existence of a variety of other information sources on Internet (email-lists, newsgroups, Web databases, WML sites), it can be useful for an intermediary system to support other popular application-level protocols, such as SMTP, IMAP, NNTP, WAP, and to have an extensible architecture that could easily incorporate new protocols [20]. The support for a wider variety of protocols is necessary in intermediary systems seeking to provide services to mobile clients, which receive informa-

tion from the network typically through protocols streamlined for low-resource devices and wireless connectivity. Therefore, some intermediaries implement customary protocols for communication with particular mobile terminals [12], customize existing application-level protocols according to the requirements of the wireless channel [31], or interface with modules that “speak” wireless protocols (such as WAP/GSM, SMS/GSM), customizing their content accordingly [25].

7.3 Intermediary Functionalities

Finally, different intermediaries can be classified according to the functionality they provide. A number of important intermediary functions have been identified in the literature [27, 36]: *customization, filtering, annotation, transcoding* and *aggregation*. Customization refers to the restructuring of content-presentation according to end-user preferences, context, location, etc. Filtering is defined as the pruning of the collected information according to the “semantic” interests of individual end-users, before dissemination. Annotation is the processing of collected and filtered content, in order to provide users with additional, useful meta-information, such as summaries, keywords, highlights, etc. Transcoding is the transformation of the content from one format to another, to make it deliverable to terminal devices that support different formats or to optimize its transportation via wireless channels. Finally, aggregation means the capability to integrate content from multiple origin servers into a single new service.

A detailed classification of intermediary systems described earlier is given in Table 1.

8 Conclusions

The tremendous success of the Web, the explosion of information available on Internet, and the emergence of mobile and thin clients have rendered the archetypal client-server model of the Web obsolete. Nowadays, numerous intermediaries intervene between origin servers and client systems, as information flows from one end to the other during a simple Web interaction. The common goal of intermediaries is to improve the quality of end-user’s Web experience by improving the performance of Web requests, by coping with information overloading, and by supporting seamless access to Web services via different terminal devices and physical connections. Their intervention ranges from very simple chores, like relaying

requests and replies and transcoding content-formats, to more complicated tasks such as caching, filtering, personalization, and crawling. Intermediaries represent a useful abstraction for designing, developing, analyzing and comparing emerging software infrastructures for “next-generation” Web services.

In this paper we presented an overview of a wide range of systems that can be described as intermediaries, classifying them in a number of broad categories according to their basic functionality: Web proxies, notification systems, wireless-Web proxies, infrastructural intermediaries. We examined the requirements arising from the need to support personalization, mobility and ubiquity under high loads. We identified and refined a set of important properties and characteristics, which can be used for: (i) the classification of existing systems and the analysis of their capabilities; (ii) the comparative study of different systems; (iii) the design of new intermediary systems. Based on this set of properties, we introduced a detailed taxonomy of characteristic intermediary systems (Table 1), identifying and investigating important features. From this taxonomy, it becomes evident that more recent systems typically consist of distributed software modules, which support a wider variety of client devices and protocols. Furthermore, that emerging intermediary systems have infrastructural characteristics as they provide abstractions and modules for the development and deployment of new applications and services.

It becomes apparent that intermediary systems will permeate the Internet infrastructure, supporting personalized and mobile services, ubiquitous portals, etc. Distributed data structures, software caches, garbage collectors, high-performance crawlers, indexers, transcoders, and user-interface agents will be some of the key components in these systems. These components will provide mechanisms for combining different pieces and formats of information to provide added-value services. To this end, a major challenge would be the development of high-level programming abstractions that would enable the fast definition and composition of new, high-performance services.

References

- [1] Akamai Technologies Inc. <http://www.akamai.com>, 2001.
- [2] eRACE Project. <http://www.cs.ucy.ac.cy/Projects/eRACE/>, 2001.

- [3] R. Aiken, M. Carey, B. Carpenter, I. Foster, C. Lynch, J. Mambreti, R. Moore, J. Strasner, and B. Teitelbaum. Network Policy and Services: A Report of a Workshop on Middleware. Technical Report RFC 2768, IETF, 2000. <http://www.ietf.org/rfc/rfc2768.txt>.
- [4] D. Aksoy, M. Altinel, R. Bose, U. Cetintemel, M.J. Franklin, J. Wang, and S.B. Zdonik. Research in Data Broadcast and Dissemination. In *Proceedings of the First International Conference on Advanced Multimedia Content Processing, AMCP '98, Lecture Notes in Computer Science*, pages 194–207. Springer Verlag, 1999.
- [5] AvantGo. Help Guide for AvantGo (PalmOS). <http://www.avantgo.com>, 2001.
- [6] R. Barrett and P. Maglio. Intermediaries: New places for producing and manipulating Web content. In *Proceedings of the Seventh International World Wide Web Conference (WWW7)*, 1998.
- [7] R. Barrett and P. Maglio. Intermediaries: New Places for Producing and Manipulating Web Content. *Computer Networks and ISDN Systems*, 30(1–7):509–518, April 1998.
- [8] R. Barrett and P. Maglio. Intermediaries: An approach to manipulating information streams. *IBM Systems Journal*, 38(4):629–641, 1999.
- [9] A. Bestavros and C. Cunha. Server-initiated Document Dissemination for the WWW. *IEEE Data Engineering Bulletin*, 19(3):3–11, September 1996.
- [10] N. Bhatti, A. Bouch, and A. Kuchinsky. Integrating user-perceived quality into Web server design. In *Proceedings of the 9th International World-Wide Web Conference*, pages 1–16. Elsevier, May 2000.
- [11] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest Information Discovery and Access System. In *Proceedings of the Second International WWW Conference*, 1995.
- [12] E. Brewer, R. Katz, E. Amir, H. Balakrishnan, Y. Chawathe, A. Fox, S. Gribble, T. Hodes, G. Nguyen, V. Padmanabhan, M. Stemm, S. Seshan, and T. Henderson. A Network Architecture for Heterogeneous Mobile Computing. *IEEE Personal Communications Magazine*, 5(5):8–24, October 1998.

- [13] E. A. Brewer. Lessons from Giant-Scale Services. *Internet Computing*, 5(4):46–55, July-August 2001.
- [14] J. Challenger, A. Iyengar, P. Danzig, D. Dias, and N. Mills. Engineering Highly Accessed Web Sites for Performance. In S. Murugesan and V. Deshpande, editors, *Web Engineering. Managing Diversity and Complexity of Web Application Development*, volume 2016 of *Lecture Notes in Computer Science*, pages 247–265. Springer, 2001.
- [15] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed. A Publishing System for Efficiently Creating Dynamic Web Content. In *Proceedings of the IEEE Infocom 2000 Conference*, pages 844–853. IEEE, 2000.
- [16] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A Hierarchical Internet Object Cache. In *Proceedings of the 1996 USENIX Technical Conference*, January 1996.
- [17] L. Degenaro, A. Iyengar, I. Lipkind, and I. Rouvellou. A Middleware System which Intelligently Caches Query Results. In *Proceedings of the IFIP/ACM International Conference on Distributed systems platforms (Middleware '00)*, pages 24–44. Springer-Verlag, 2000.
- [18] M. Dikaiakos and D. Gunopulos. FIGI: The Architecture of an Internet-based Financial Information Gathering Infrastructure. In *Proceedings of the International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems*, pages 91–94. IEEE-Computer Society, April 1999.
- [19] M. Dikaiakos and A. Stassopoulou. Content-Selection Strategies for the Periodic Prefetching of WWW Resources via Satellite. *Computer Communications*, 24(1):93–104, 2001.
- [20] M. Dikaiakos and D. Zeinalipour-Yazti. A Distributed Middleware Infrastructure for Personalized Services. Technical Report TR-01-4, Department of Computer Science, University of Cyprus, December 2001.
- [21] F. Douglis, T. Ball, Y.-F. Chen, and E. Koutsofios. The AT&T Internet Difference Engine: Tracking and Viewing Changes on the Web. *World Wide Web*, 1(1):27–44, January 1998.

- [22] Directorate General for the Information Society. Digital Content for Global Mobile Services: Executive Summary. European Commission, 2002.
- [23] Wireless Application Protocol Forum. *Official Wireless Application Protocol: The Complete Standard with Searchable CD-ROM*. John Wiley & Sons, 1999.
- [24] A. Fox, I. Goldberg, S. Gribble, D. Lee, A. Polito, and E. Brewer. Experience with Top Gun Wingman: A Proxy-based Graphical Web Browser for the 3Com PalmPilot. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, pages 407–426, 1998.
- [25] A. Fox, S.D. Gribble, 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, August 1998.
- [26] S. Gribble, E. Brewer, J. Hellerstein, and D. Culler. Scalable, Distributed Data Structures for Internet Service Construction. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000)*, 2000.
- [27] S. Gribble, M. Welsh, R. von Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R.H. Katz, Z.M. Mao, S. Ross, and B. Zhao. The Ninja Architecture for Robust Internet-scale Systems and Services. *Computer Networks*, 35:473–497, 2001.
- [28] Handspring. Blazer. http://www.handspring.com/software/blazer_overview.jhtml.
- [29] Gary Hillerson. *Web Clipping Developer's Guide*. Palm Inc., May 2001.
- [30] B.C. Housel and D.B. Lindquist. WebExpress: A System for Optimizing Web Browsing in a Wireless Environment. In D. Milojevic, F. Douglis, and R. Wheeler, editors, *Mobility. Processes, Computers and Agents*, chapter 9.5, pages 353–362. ACM Press, 1999.
- [31] B.C. Housel, G. Samaras, and D.B. Lindquist. WebExpress: A Client/Intercept Based System for Optimizing Web Browsing. *ACM/Baltzer Journal of Mobile Networking and Applications (MONET)*, 3:419–431, 1998.

- [32] A. Joshi. On proxy agents, mobility, and web access. *Mobile Networks and Applications*, 5:233–241, 2000.
- [33] K. Kant and P. Mohapatra. Scalable Internet Servers: Issues and Challenges. In *Workshop on Performance and Architecture of Web Servers (PAWS), held in conjunction with the ACM SIGMETRICS 2000*. ACM, June 2000.
- [34] E. Levy, A. Iyengar, J. Song, and D. Dias. Design and Performance of a Web Server Accelerator. In *Proceedings of IEEE Infocom 1999 Conference*. IEEE, 1999.
- [35] Q. Lu, M. Eichstaedt, and D. Ford. Efficient profile matching for large scale Webcasting. In *Proceedings of the 7th International World Wide Web Conference*, 1998.
- [36] P. Maglio and R. Barrett. Intermediaries Personalize Information Streams. *Communications of the ACM*, 43(8):96–101, August 2000.
- [37] D. Milojevic. Internet Technology. *IEEE Concurrency*, pages 70–81, January-March 2000.
- [38] J. Pasquale, E. Hung, T. Newhouse, J. Steinberg, and N. Ramabhadran. Improving Wireless Access to the Internet by Extending the Client/Server Model. In *Proceedings of the European Wireless 2002 Conference*, volume 2, pages 670–676, February 2002.
- [39] M. Perkowitz and O. Etzioni. Towards adaptive Web sites: Conceptual framework and case study. *Artificial Intelligence*, 118:245–275, 2000.
- [40] The Apache XML Project. Introduction to Cocoon Technologies. White Paper, 1999-2001. <http://xml.apache.org/cocoon/technologies.html>.
- [41] P-A. Queloz and A. Villazon. Composition of Services with Mobile Code. In *Proceedings of the Joint Symposium ASA/MA '99. First International Symposium on Agent Systems and Applications (ASA '99). Third International Symposium on Mobile Agents (MA '99)*, pages 176–189. IEEE-Computer Society, October 1999.
- [42] H. Rao, Y. Chen, D. Chang, and M. Chen. iMobile: A Proxy-based Platform for Mobile Services. In *The First ACM Workshop on Wireless Mobile Internet (WMI 2001)*, 2001.

- [43] H. Rao, Y. Chen, and M. Chen. A Proxy-based Web Archiving Service. In *Middleware Symposium*, 2000.
- [44] P. Rodriguez and E.W. Biersack. Bringing the Web to the Network Edge: Large Caches and Satellite Distribution. In *Proceedings of WOSBIS '98, ACM/IEEE MobiCom Workshop on Satellite-based Information Services*, October 1998.
- [45] B. Schechter. Information on the fast track. *IBM Research Magazine*, 35(3):18–21, 1997.
- [46] S. Singhal, T. Bridgman, L. Suryanarayana, D. Mauney, J. Alvinen, D. Bevis, J. Chan, and S. Hild. *The Wireless Application Protocol*. Addison Wesley, 2001.
- [47] Stardust.com. The ins and outs of Content Delivery Networks. White Paper, December 2000. <http://events.stardust.com/cdn/resources.htm>.
- [48] S. Vinoski. Where is Middleware? *IEEE Internet Computing*, 6(2):83–85, March-April 2002.
- [49] Jim Waldo. Jini Architecture Overview. White Paper. <http://www.sun.com/products/jini/whitepapers>.
- [50] D. Wessels. Squid internet object cache. Technical report, National Lab for Applied Network Research, August 1999. <http://squid.nlanr.net/Squid/>.
- [51] D. Wessels and K. Claffy. Evolution of the NLANR Cache Hierarchy: Global Configuration Challenges. Technical report, NLANR, October 1996. <http://www.nlanr.net/Papers/Cache96/>.
- [52] E. Wilde. *Wilde's WWW: Technical Foundations of the World Wide Web*. Springer, 1999.
- [53] A. Wolisz, C. Hoene, B. Rathke, and M. Schlage. Proxies, active networks, reconfigurable terminals: The cornerstones of future wireless internet. In *Proceedings of the IST Mobile Communications Summit*, pages 795–803, 2000.
- [54] Alec Wolman, Geoff Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry Levy. On the scale and performance of cooperative Web proxy caching. In *Proceedings*

of the 17th ACM Symposium on Operating Systems Principles (SOSP'99), pages 16–31, December 1999.

- [55] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet. Concordia: An Infrastructure for Collaborating Mobile Agents. *Lecture Notes in Computer Science*, 1219, 1997. <http://www.meitca.com/HSL/Projects/Concordia/>.
- [56] K. Yagoub, D. Florescu, V. Issarny, and P. Valduriez. Caching Strategies for Data-Intensive Web Sites. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 188–199, 2000.
- [57] T. W. Yan and H. Garcia-Molina. SIFT - A Tool for Wide-Area Information Dissemination. In *Proceedings of the 1995 USENIX Technical Conference*, pages 177–186, 1995.
- [58] N. J. Yeager and R. E. McGrath. *Web Server Technology*. Morgan Kaufmann, 1996.
- [59] D. Zeinalipour-Yazti and M. Dikaiakos. Design and Implementation of a Distributed Crawler and Filtering Processor. In A. Halevy and A. Gal, editors, *Proceedings of the Fifth Workshop on Next Generation Information Technologies and Systems (NGITS 2002)*, volume 2382 of *Lecture Notes in Computer Science*, pages 58–74. June 2002.