Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

ELSEVIER

Computer Networks 45 (2004) 421–447

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Intermediary infrastructures for the World Wide Web

Marios D. Dikaiakos *

Department of Computer Science, University of Cyprus, P.O. Box 20537, Nicosia CY 1678, Cyprus

Received 20 June 2002; received in revised form 20 October 2003; accepted 6 February 2004
Available online 26 February 2004

Responsible Editor: R. Stadler

Abstract

Intermediaries are software entities, deployed on hosts of the wireline and wireless network, that mediate the interaction between clients and servers of the World Wide Web. In this paper we present a survey of intermediaries, focusing on systems beyond simple caching proxies. We classify different intermediary systems into three categories, based on their functionality and focus: First, we investigate notification intermediaries, which are driven by end-user profiles and operate even in the absence of end-user connection. Then, we study intermediaries developed to support wireless connectivity, mobility, and ubiquity. Finally, we examine intermediary infrastructures designed to extend the support of the core network for the development and deployment of new services. Based on this survey, we propose a detailed taxonomy of intermediaries and identify key features of emerging intermediary infrastructures. Taking into account recent advances and trends in wireless and pervasive Internet technologies, we present a number of research challenges, which need to be addressed in order to integrate intermediary systems in next-generation Internet infrastructures.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Intermediary systems; World Wide Web; Proxy servers; Edge services; Personalized and mobile services

1. Introduction

The World Wide Web has become the prevailing paradigm for information dissemination on Internet and the main driving force behind the popularity of Internet services. The design of the Web was based on the client–server model of distributed computing, with servers storing data resources (content) and providing clients with data on-demand via the HTTP protocol. The tremendous success of the Web has resulted in extraordinary loads upon the Web infrastructure. Moreover, the wider deployment of wireless connectivity and the emergence of hand-held devices have created the need to support seamless Web access from mobile devices and wireless access networks. To this end, in the current context of Web use, client–server interaction is often mediated by a variety of software systems, which seek to enhance the performance, the scalability, and the ubiquity of the Web: proxies, intermediaries, mid-point servers, content service networks, etc. [26,53,55]. In this work, we adopt the general term "intermediaries" to describe such systems collectively. We define intermediaries as *software entities deployed on hosts of the wireline and wireless*

* Tel.: +1-357-2-289-2230; fax: +1-357-2-233-9062.
E-mail address: mdd@ucy.ac.cy (M.D. Dikaiakos).

network, placed in the content path between origin servers and client systems. Intermediaries intervene in the client–server interactions that take place at the application layer of the Internet [15,53,68]; *the content path is the route taken by client requests and server responses through the network* (see Fig. 1). The purpose of intermediaries is to extend the functionality and application-performance offered by the network to its end-users, without violating the end-to-end principles applied in the design of the Internet [62]. A typical intermediary operation involves the modification of client requests or origin-server responses and the creation of content in response to on-line or off-line client requests. The functionality of intermediary systems varies from message relaying, caching and replicating content, to performance enhancements and complex transformations, such as load balancing, protocol adaptation, content filtering, indexing, and virus scanning. Intermediary systems can also be classified as "middleware", since they provide a "reusable and expandable set of services and functions, commonly needed by many applications to function well in a networked environment" [8].

In this paper, we present a survey of intermediaries, focusing on systems beyond simple caching and replication proxies. We identify and refine a set of important characteristics of different intermediaries. We classify examined intermediaries into three different categories of systems, based on their functionality and focus: First, we investigate notification intermediaries, which are driven by end-user profiles and operate even in the absence of end-user connection. Second, we look at intermediaries developed to support wireless connectivity, mobility, and ubiquity. Finally, we examine intermediary infrastructures designed to extend the

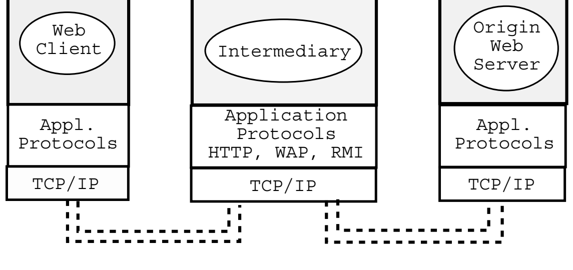

Fig. 1. Intermediary operation.

support of the core network for the development and deployment of new services. Based on this survey, we propose a detailed taxonomy of intermediaries and identify key features of emerging intermediary infrastructures. Taking into account recent advances and trends in wireless and pervasive Internet technologies, we present a number of research challenges, which need to be addressed in order to integrate intermediary systems in next-generation Internet infrastructures.

The importance of intermediary systems is increasing with the emergence of wireless connectivity, of mobile services, and the need to push dynamic content towards the edges of the network. In that context, intermediaries will operate as an overlay infrastructure, that is a virtual network of connected components layered on the existing IP network. This overlay network of intermediaries will enhance the development and deployment of "next-generation" services for the Web, supporting personalization, customization, localization, and ubiquitous access from various terminal devices over different physical media and protocols.

The remaining of this paper is organized as follows: Section 2 examines approaches to extend the client–server model of the Web. Section 3 introduces three dimensions that can be used for the characterization and analysis of intermediaries providing services beyond HTTP proxying: functionality, system architecture, and interaction support. Sections 4–6 give an overview of three different categories of intermediary systems: notification systems for the World Wide Web, intermediaries for mobile devices and wireless service, and intermediary infrastructures. In Section 7 we present a taxonomy of the reviewed intermediary systems and discuss efforts to establish a common, reference intermediary architecture. We also present open research issues. Finally, we conclude in Section 8.

2. Extending the client–server model

2.1. Extending the edges

Initial efforts to extend the functionality of the Web resulted in systems tightly integrated with

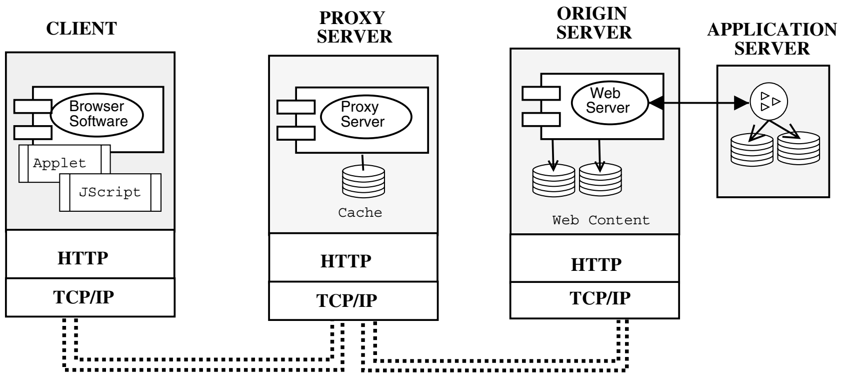

Fig. 2. Client–server model of the Web and early extensions.

Web clients and origin servers (see Fig. 2). Such systems do not comply with the definition of intermediaries given earlier; therefore, their in-depth study is outside the scope of this paper. We do provide a brief overview of such systems, however, in order to discuss their limitations and to examine how they address problems similar to those arising in the context of intermediaries.

Early on, Web developers sought to develop technologies enabling the dynamic adaptation of browser interfaces to Web-application requirements. This resulted in browser environments supporting the dynamic downloading and interpretation of compiled code and scripts, like Java Applets and client-side scripts. Hence, service-providers have been able to improve the flexibility of Web-service provision by shipping part of a service's computation from the origin server to its clients. Web-client extensions, however, come with significant costs due to the existence of numerous client environments that differ in operating system, in supported standards, and in available resources.

The success of the Web created the need to expand its information resources from collections of merely static Web-pages to dynamic content produced by software applications, databases, legacy systems, etc. This need was addressed by technologies that enabled the use of Web servers as front-ends to back-end applications; for instance, with server-side scripting following the CGI standard, Active Server Pages (ASP) by Microsoft, Java Server Pages (JSP), Java servlets, and XML engines [58]. The increase of dynamic content on the Web, however, 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 affected the latency experienced by Web users, because the creation and serving of dynamic content requires on the average orders of magnitude more CPU time than that of static pages of comparable size [24].

Several projects have addressed the management and performance improvement of dynamic-content provision (see, for instance [25,30,75]). In the following sections, we examine two projects, IBM's Websphere and INRIA's Weave, which proposed alternative approaches for specifying explicitly the assembly of dynamic content from "raw" data, for caching content while dealing with consistency with back-end databases, and for separating the concerns of data retrieval, aggregation, and presentation.

2.1.1. Composition of dynamic content out of fragments

IBM's Websphere middleware constructs dynamic Web-pages from information fragments extracted from back-end applications and/or databases [25,30]. This approach is based on the premise that Web-pages can be decomposed into a hierarchy of complex, atomic fragments, that is parts of a Web-page which change together and reside at the leaves of the hierarchy [25]. Atomic fragments can be updated either on-the-fly or via update propagation mechanisms, and stored in a software cache until their expiration. Experiments have shown that the typical overhead of composing a Web-page from simpler fragments is minor to the overhead of constructing the whole page from scratch [25].

This approach has been 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. 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 [25].

The concept of the object dependence graph is incorporated in the *Accessible Business Rules* framework (ABR) of Websphere [30], which provides application developers with access to business rules. Business rules are implemented as persistent objects encapsulating code and attributes. The ABR code contains decision points, which query back-end databases at run-time and decide which particular business logic (application code) to invoke. ABRs 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 [30]. 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 multi-threaded and replicated [30].

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.

2.1.2. Weave

A different approach for managing dynamic content was introduced in the context of the *Weave* project of INRIA, which developed a system for specifying and generating data-intensive Web-sites [75]. Weave separated the three key concerns of Web-site management, which are interwoven in many systems: (i) Web-site structure and content specification; (ii) Web-page presentation and graphical style; and (iii) 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.

A WeaveL program translates into a series of queries executed on the underlying database and producing XML fragments, which are subsequently translated into HTML; XML to HTML translation is conducted by XSLT programs representing the specification of the Web-site presentation design. WeaveL-program execution is a data materialization process 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, 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 [75].

The architecture of Weave is based on a three-tier design. At the middle tier 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 Web-interface on the front-end [75].

2.2. Proxy servers and Web caches

Proxy servers are deployed by network administrations and Internet service providers (ISPs), in order to cope with increased Web traffic and to optimize their resource use. Proxies intermediate between Web clients and origin servers, filtering or redirecting HTTP requests to reduce Web-server loads and improve user-perceived QoS. Web proxies also operate as Web-traffic caches, storing relayed responses and serving identical requests from local storage [20,49,70]. The potential gains from Web caching are significant: caching popular documents on a local network reduces 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.

In the complex hierarchy of wide-area and local networks that connect Web clients to origin servers, there are several places where proxy servers can be deployed. For example, "reverse" proxies are installed near origin Web servers and cache content served by these servers; their purpose is to reduce Web-server load. A special case of reverse proxies are *Web* or *HTTP accelerators*, which are installed "in front" of mirrored Web servers or of multiple Web servers collocated on the same cluster and sharing a common file system [24]. An accelerator receives and distributes requests, caching replies so that frequently requested pages are served from the accelerator cache rather than the Web server. The distribution of requests can be achieved either at the TCP-router level or with round-robin DNS servers, which associate a single domain name to multiple IP hosts on a round-robin basis. Implementations of Web-server accelerators run under an embedded operating system, which reduces the overhead of TCP and operating-system buffering [50]. Overall performance is improved because of the reduction of direct hits to origin servers and the serving of static and dynamic content from the streamlined accelerator cache. With the exception of reverse proxies and accelerators, most proxy servers are deployed at the relays that connect a local network to an institutional network, a local ISP network to a regional-provider network, or multiple regional networks to a national backbone [26,49]. Proxy servers placed at different points of the Internet can be configured to work as hierarchical [71] and co-operative caching infrastructures [73].

More recently, caching infrastructures have evolved into *Content Delivery* or *Content Distribution Networks* (CDNs), that is specially tuned overlays to the Internet like Akamai's network [1]. CDNs "leverage a strategically arranged set of distributed Web caching, load-balancing and Web-request redirection systems", in order to replicate content near the network's edge, to improve service availability, and to reduce service latency [66]. A typical CDN comprises a geographically distributed collection of cooperating *edge proxies*, which belong to a single administrative entity. A CDN organization offers its services to content providers wishing to improve their availability and performance. To this end, the CDN caches popular data objects of affiliated origin servers and redirects popular-object requests to the proxy which is closer to the clients issuing the requests. Several research challenges arise in the context of large-scale CDNs with hundreds or thousands of proxies: the selection of the edge proxy to which a client request is redirected, the efficient replication of content to CDN proxies, the consistency maintenance between edge proxies and origin servers hosting the same data objects, the interoperability between CDNs belonging to different organizations, etc. Several approaches have been suggested in recent literature to cope with these issues, including algorithms for redirecting client requests to nearby edge proxies, CDN topology, approaches for populating edge caches with content, and algorithms for addressing the consistency problem (see, for instance, [9,32,34,56]).

3. Beyond Web proxies

In the following sections, we examine systems extending the paradigm of typical Web intermediaries. In particular, we focus on efforts to

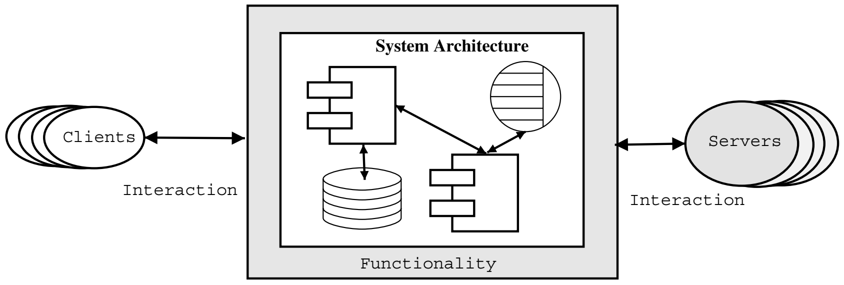

Fig. 3. Characterizing intermediaries.

enhance Web intermediaries along three main dimensions (see Fig. 3):

- The *functionality* provided by an intermediary beyond proxying and content caching. Functionality can be described in terms of the capabilities, services, and policies which are supported by the intermediary's architecture and which determine its behavior.
- The *system architecture* of an intermediary, which describes its composition in terms of individual software components, the division of roles and functions between these components, their inter-relationships, and their placement across the network [27].
- The *interaction* between intermediaries and their counterparts (intermediaries, client systems, origin servers). Interaction can be described in terms of the patterns of communication and the protocols supported by an intermediary system.

Below, we examine the particular features used to describe and classify different intermediaries.

3.1. Functionality

A number of important intermediary functions have been identified, besides the relay of messages and the replication of content in distributed caches: customization, filtering, annotation, transcoding, protocol translation, and content creation [8,19,43,53]:

- *Customization* refers to the capability of restructuring the presentation of content according to: end-user preferences, terminal-device capabilities, the context of use, the physical location of access, etc. Customization is important for systems seeking to support ubiquitous and/or location-based services.
- *Filtering* refers to the analysis of content retrieved from origin servers. The purpose of filtering is to decide whether the retrieved content matches the semantic interests of end-users or if it complies with policies for security and use. Support for filtering is important when applying intelligent techniques for service personalization and localization, protection from viruses and indecent content, etc.
- *Annotation* refers to the processing of content in order to provide users with additional information and meta-information, such as summaries, keywords, highlights, ad banners, etc. Annotations can be stored in formats different than the original content and can be dispatched to end-users via different communication mechanisms. Support for annotations can be beneficial for reducing the information overload of end-users and for extending such services to terminal devices with limited capabilities, voice interfaces, 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 across wireless access points and channels of diversified bandwidth.
- *Protocol translation* refers to the translation of application-level traffic from one application-protocol to another. Protocol translation is often performed at mobile support stations of

wireless networks, between protocols optimized for wireless media and protocols of the wireline Internet.
- *Content creation* refers to the generation of content at intermediaries systems. Usually, the content created at an intermediary results from application code off-loaded, cached, and executed at the intermediary, in coordination with one or more origin servers. Alternatively, intermediaries can produce content out of aggregated information fragments, retrieved from one or more origin servers and/or local intermediary caches.

Intermediary operation is authorized by either the origin server or the client system. Intermediaries that belong to the former category are called *surrogates* while those belonging to the later category are called *delegates* [12]. Surrogates act on behalf of origin servers whereas delegates represent the choices of end-users.

For the implementation of the functionalities described above, intermediary entities need to support the analysis and modification of client requests and server responses, and the generation of content. Client-request analysis and modification may be required in order to: (i) annotate request messages with additional information about the context of end-user connections, the properties of the terminals used, and the end-user profiles; (ii) redirect requests for load-balancing purposes; and (iii) translate requests to some other application protocol. The need to analyze and modify server responses arises in cases such as the customization, transcoding, annotation, and filtering of content. Finally, the generation of content at an intermediary site arises when the intermediary is used to compose new "value-added" services out of content retrieved from multiple origin servers. Furthermore, when parts of a dynamic-content generation process are off-loaded to the intermediary in order to reduce origin application-server load, network congestion, and network delays to the client.

3.2. System architecture

Intermediaries can be classified either as centralized or as distributed. Centralized systems are composed of tightly integrated software modules deployed at a single host. Distributed intermediaries consist of two or more software entities deployed at multiple hosts and communicating via message passing, remote procedure calls, distributed events, or shared memory. Distributed design can provide important advantages, such as performance scalability, improved robustness, and availability.

Intermediaries are also characterized by the deployment and ownership of their entities. Deployment can be at the front of origin servers, at hosts deployed in an intranet or the Internet, and at the client side. Ownership lies with client devices, ISPs, intermediary service providers, CDNs, content providers, and enterprise intranets [68].

Besides the structure and ownership of intermediary components, it is interesting to consider the complexity of intermediary implementation. Several intermediaries provide a limited functionality, such as protocol reduction and content transcoding [38,46,65]. Other systems are more complex and provide support for content caching and versioning, for collecting resources from multiple origin servers, and for indexing large sets of retrieved content [30,35,78].

Finally, a key aspect of intermediary systems is their support for configurability and programmability. This support is necessary in order to use intermediaries as programmable overlay networks, upon which application origin servers off-load parts of their business logic (application code), and service providers develop and deploy new services [13,23,77]. Tuning of an intermediary infrastructure can be achieved at different levels of abstraction and flexibility:

- Through configuration parameters, which determine the exact set of intermediary operations invoked in a given context. Different configurations can result to the adaptation of services provided via the intermediary. Configuration parameters can be either hard-wired into the intermediary implementation or extracted from meta-data files encoded in special XML syntax.
- With the employment of generic execution environments supporting the dynamic off-loading of

intermediary entities into the infrastructure. General-purpose middleware platforms of this type, such as the Java Virtual Machine, Jini [69], mobile-agent execution environments [29], provide APIs, software libraries, and design patterns for programming new services in an intermediary context.
- With compositional frameworks providing components that can be used as building blocks for defining new services according to a higher-level programming model.

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

3.3. Interaction

An 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 protocols employed, and the supported media (wireline or wireless).

Typically, two modes of communication are employed by intermediary systems: *synchronous* and *asynchronous*. 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 significant 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.

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 Web is pull-based with clients initiating the retrieval of content. A number of projects have investigated the use of push-based approaches, which involve servers or intermediaries dispatching content to clients according to various criteria. Moreover, many intermediaries work as user-agents, being constantly connected to the fixed network, collecting and filtering information on behalf of the end-users.

Most intermediary servers on the Web "speak" the HTTP protocol and connect to Web servers to download information. Given, however, the existence of a variety of other information sources on the 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.

The support for a wider variety of protocols is necessary in intermediary systems seeking to provide services to mobile clients, which receive information 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 [21], customize existing application-level protocols according to the requirements of the wireless channel [46], or interface with modules that "speak" wireless protocols (such as WAP/GSM, SMS/GSM) customizing their content accordingly [39,72].

4. Notification systems

Notification systems are intermediaries that monitor changes in origin Web servers on behalf of subscribed users. Whenever an update in the content of a monitored source is observed, the notification intermediary evaluates the relevance of this change with respect to stored user profiles and can notify interested subscribers accordingly. Notification systems extend the functionality of Web proxies, adding support for filtering, annotation and aggregation. In contrast to Web proxies, notification intermediaries are usually driven by user profiles or long-term queries, executed even when the end-user is off-line. Often, it is up to the notification intermediary to initiate connection with a client device, through a push model of information provision with communication protocols like SMTP or SMS over GSM.

A key issue in the comparison of different notification systems is their system architecture: its component structure, scalability, and support for expressing and processing user profiles. Another issue of interest is their support for different application-level protocols of the Internet.

4.1. SIFT

One of the first examples of an intermediary notification system is SIFT, the *Stanford Information Filtering Tool* (SIFT) [76]. SIFT was designed and developed to provide large-scale information dissemination services to users that subscribe their interests to SIFT servers. SIFT operates as a centralized server comprised of a profile-database and a tightly integrated dissemination engine.

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

Extending SIFT to support pull-based information provision over HTTP would be a straightforward task and has been implemented in many subsequent, commercial systems. Pulling resources from the Web, however, would require the addition of mechanisms to deal with old Web-pages, with different versions of the same Web-page, etc.

4.2. AIDE

AIDE, the *AT&T Internet Difference Engine*, is a notification system for Web resources. AIDE was designed to archive and handle multiple versions of changing Web resources [35]. It is comprised of a centralized notification server, a version archive and a difference engine that identifies and displays changes in Web-page content. These components are tightly integrated into the AIDE server.

Subscribers register in AIDE the list of URLs they wish to track and a few parameters configuring the degree of desired notification. AIDE supports recursive tracking and differencing of Web-pages and their descendants. Special emphasis is placed on the presentation through HTML of differences between subsequent versions. Alerts about changes in tracked pages are "pushed" to users via SMTP; additional access is provided via the Web and HTTP.

4.3. IBM's Grand Central Station

The Grand Central Station (GCS) project of IBM Almaden sought to expand and improve earlier efforts dealing with Webcasting-service provision. A result of the project was an intermediary system for the Web focusing on the description and management of user-profiles and the support of diversified Internet sources [52,64]. The GCS intermediary is a centralized server which: (i) speaks multiple application-level protocols (HTTP, NNTP, SMTP) and collects content from different Internet sources (Web, news, email); (ii) introduces the *Grand Central Station Profile Language* (GCSPL), a boolean-structured predicated language for specifying personalized channels of information; (iii) incorporates an engine to process user profiles expressed in the GCSPL; and (iv) disseminates content to subscribers via Webcasting channels.

The profile engine of GCS merges 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 [52]. Experiments with this approach showed that profile merging into a common boolean-tree representation results to effort-duplication avoidance in the presence of users with similar profiles. This led to good performance scalability with increasing numbers of subscribers.

4.4. FIGI

SIFT, AIDE and the GCS are "server-based" and centralized tools (according to the taxonomy in [35]) as they rely on tightly integrated components running on a central location and not on users' machines or nodes of the networking infrastructure. An approach for building 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 [31].

Similarly to SIFT, FIGI is user-profile driven. A FIGI-profile is a set of long-term, continuously evaluated queries, which 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 FIGI server is organized as a distributed, two-tier architecture. The first tier comprises 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 and connection management. The alerter retrieves information pertinent to a user profile from the FIGI cache and re-directs it to the user through his personalized Web interface. The profile registry is a server where users can register or update their interests. The second tier of FIGI comprises 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. FIGI-modules are developed on top of the Concordia Mobile-Agent middleware, as stationary or mobile agents [74]. Therefore, the components of the system can be distributed dynamically to different hosts residing at different Internet hosts.

5. Intermediaries for mobility and ubiquity

As resource-limited client machines become more popular, the use of wireless connectivity, the heterogeneity of client devices, and the capacity mismatch between clients and servers are expected to grow [36,72]. To cope with these trends, system infrastructures for Internet services have to support ubiquitous service provision over both wireless and wireline connections on diversified devices and client systems, in the presence of personal or physical mobility of end-users. [1] To this end, infrastructures have to: (a) optimize client–server communication over the wireless medium; (b) support both synchronous (on-demand) and asynchronous modes of interaction with users, thus coping with frequent disconnections of wireless connections and user mobility; (c) support seamless access from a variety devices; (d) customize content to adapt to different terminal devices; (e) enable the provision of multiple formats (HTML, WML, XML) to the same device over the same link; (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 of the World Wide Web. Earlier projects have addressed the problem of providing Internet applications over wireless connections by deploying intermediary components ("agents") on the wireline network

[1] Personal mobility: the ability of an end-user to move from one device to another without perceiving a significant change in the context of his application. Physical mobility: the user moves around with his terminal.

and/or the mobile host. Such components mediate between origin servers and mobile terminals, optimizing communication, dealing with disconnections, etc. To this end, they provide the minimal functionality required to do protocol translation between application-level protocols for the wireline Internet and streamlined versions for wireless.

Nevertheless, to support personalization, content customization, ubiquity and mobility, proxy functionality has to be extended over the optimization and management of wireless communications. This raises the need for increased computational resources becoming available at the intermediary's host in order to maintain its high-performance operation. A single-component approach would deploy an enhanced intermediary in some host residing at the content 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 [47].

End-to-end solutions represent a possible alternative, with origin servers either storing content in formats compatible with wireless clients or adapting it on-the-fly according to client-system, connection, or end-user profiles [47]. Under the end-to-end approach, however, adapted content or adaptation software must be inserted at each origin server. Consequently, updates have to be propagated to all origin servers whenever new terminal devices, wireless access protocols, and content-formats emerge. Furthermore, on-the-fly adaptation of content can be very time-consuming, leading to a deterioration of user experience.

One approach for coping with these problems suggests the deployment of powerful, client-specific adaptation intermediaries in the network, acting as delegates for a specific family of terminal devices; this is the case of the Blazer system by Handspring [44]. Another approach, followed by WebExpress [46] and the Web Stream Customizers [67], suggests the separation of adaptation concerns between multiple cooperating intermediaries. A third option, implemented in the context of WAP and Palm services [45,65], is to combine end-to-end with proxy-based solutions. In the following sub-sections we examine proposed solutions targeting the requirements of wireless access and resource-poor clients.

5.1. Handspring's blazer

Blazer is a general-purpose micro-browser developed for hand-helds running the PalmOS operating system [44]. The micro-browser supports text, links, icons, and colors. Communication between the micro-browser and origin servers goes through a proprietary, centralized adaptation intermediary. The Blazer intermediary supports HTTP and WAP in its interaction with the micro-browser, depending on the connection established by the hand-held. On the origin-server side, the Blazer intermediary speaks HTTP to communicate with origin servers. If the hand-held is connected via WAP, the intermediary translates WAP requests into HTTP.

In contrast to Wap gateways and Web clipping proxies, the Blazer intermediary can transcode a variety of formats (HTML, WML/HDML, cHTML) into a Blazer-specific, stripped-down HTML format accepted by the micro-browser. Possible adaptations conducted in this process include the removal of unnecessary or unsupported tags, the reduction of the size of Web-pages and images, and the adaptation of Web-page size to the screen of the hand-held [44].

5.2. Distributed intermediaries for wireless Web access

IBM's *WebExpress* is a characteristic example of a proxy-based approach seeking to optimize Web access from resource-poor clients connected via wireless connections [46]. WebExpress is comprised of two intermediary entities ("agents"): the *server-side intercept* (SSI), dispatched on the wireline network, and the *client-side intercept* (CSI), residing on the end-user's mobile device. The two agents communicate using a stripped-down version of the HTTP/1.0 protocol, on top of a TCP/IP connection established over the wireless link. Both agents support HTTP header reduction, simple caching of content, and run a simple differencing protocol between the client and server-side caches to reduce the data exchanged over the low-bandwidth, wireless link.

An extension of the WebExpress approach is proposed in the context of the *Web Stream Customizer* (WSC) project [67]. The WSC system customizes Web traffic on-the-fly as it flows along the content path between a client and an origin server. The goal of the WSC is to support Web access for resource-limited clients and to cope with wireless bandwidth fluctuations, disconnections, etc. To this end, customizers provide redirection of HTTP traffic, protocol and content adaptation, compression, and content caching.

A customizer comprises two intermediary entities: a *local component* (LC), residing at a *local component server*, and a *remote component* (RC), residing at a *remote component server*. Each LC server is dedicated to one client machine and is deployed inside or near that machine. An LC server hosts a number of LC entities, with each LC managing the traffic that flows between its corresponding client and a single origin server. An RC server is deployed near one or more associated origin servers; the RC server hosts a number of RCs customizing the content provided by these origin servers.

Typically, for every client of the WSC system, there are more than one customizers active simultaneously, each being a different (LC, RC) pair [67]. A customizer operates on requests to and responses from a specified set of Web sites which belong to its *domain of applicability*. The LC server of a client re-directs any requests for Web-pages outside that client's domains of applicability to the corresponding origin servers.

Customizers are implemented as Java classes, stored in jar files. Customizer programming is done with a callback-based programming model that facilitates the development of new customizer entities. The selection, download, installation, and configuration of customizers can be specified by the end-user through a Web-based interface.

5.3. Wap gateways and Web clippings

Wap gateways and Web clipping proxies are intermediaries concerned with the translation between HTTP and wireless application protocols for WAP-enabled mobile phones and Palm handheld devices, respectively. Both systems handle content encoded in formats optimized for wireless access, assuming the availability of this kind of content at the origin servers.

5.3.1. Wap gateways

The *Wireless Application Protocol* (WAP) is a layered suite of standards defining how wireless devices communicate over the wireless network and providing lower-level optimizations for the provision of Web access over wireless devices [37,65]. WAP has been widely adopted for providing Internet connectivity and mobile Web services over cellular GSM and GPRS mobile phones [36]. WAP-enabled handsets can establish connections to WAP-compliant wireless data infrastructures, request content from HTTP servers, and present it 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), which manages the establishment of long-term, wireless sessions between WAP micro-browsers running on mobile phones and Wap gateways running on the wireline network. The *WAP gateway* is an intermediary installed "near" the local base-station of the wireless devices it serves [65]. The 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 [65].

WAP carries content encoded in WML (wireless markup language) format. Consequently, the deployment of WAP applications requires either the full transformation of existing Web applications from HTML into WML or the translation of HTML content into WML on-the-fly. This translation can be conducted by intermediary servers installed between the WAP gateway and origin Web servers. Developing intermediaries to translate automatically arbitrary HTML pages into WML is not an easy task [65]. Moreover, both approaches have obvious drawbacks due to cost and performance considerations.

5.3.2. Web clippings

Web Clipping is an architecture proposed and implemented by Palm Inc. to support Web content retrieval from Palm devices connected via wireless links [45]. To fetch content from a Web-site via the Web clipping system, a user has to install a site-specific *Web Clipping Application* on his hand-held device. The clipping application works as a special-purpose, site-specific browser: it provides a navigation interface to the content of a particular Web-site, translates user clicks into query messages, and displays fetched content (the "clipping") on the hand-held screen.

Typically, clippings are small, "Palm friendly", HTML 3.2 pages generated dynamically by CGI scripts or application programs, deployed at selected origin servers. An intermediary 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. A Palm device encodes its requests into a proprietary, compressed format (the Palm query format), and sends them to the intermediary using UDP. The intermediary translates incoming requests into HTTP request messages dispatched to the "Palm friendly", origin server. Upon the origin server's reply, the intermediary translates the HTML 3.2 content into a compressed format known as Compressed Markup Language (CML), and wraps it into a compressed Palm Proxy Format (PPF). Subsequently, the intermediary sends the PPF file back to the clipping application via the hand-held's local base station using UDP [45]. The intermediary server provides limited caching of static HTML resources and takes care of encryption–decryption using SSL and the HTTPS protocol.

6. Intermediary infrastructures

Performance scalability problems are growing 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 [48]. High loads result in poor end-to-end performance and low quality of service [18]. Therefore, service infrastructures will have to accommodate millions of simultaneous end-users connecting from a large heterogeneity of end-user devices and resulting in 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 24×7 availability and robustness, and be scalable in terms of capacity and performance [22,54].

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 [21,61]. Such an approach would result to the deployment throughout the network of distributed, programmable and possibly mobile *intermediary servers*, mediating between primary sources and various client systems.

In this section, we describe a number of systems seeking to comply with these characteristics. All systems described offer a limited level of programmability or configurability by: (i) providing a number of software components that represent the building blocks of new intermediary systems; (ii) giving guidelines on how these components can be programmed, configured, deployed, and how they exchange information; and (iii) providing a framework within which these components can be executed.

6.1. The TACC model

The BARWAN project at UC/Berkeley developed a system to support service access from mobile clients that roam across a collection of heterogeneous wireless networks [21,39]. A key component of the BARWAN system is an intermediary server that customizes content retrieved from origin servers before feeding it to heterogeneous mobile clients.

The architecture of the BARWAN intermediary consists of three software layers with different roles: The lower layer, *scalable network services* (SNS), provides guarantees for high availability, scalability, and robustness of the intermediary operation. The middle layer, *TACC*, provides a compositional programming model designed to simplify the creation and deployment of services by the intermediary. The top layer handles the actual presentation of data to various client terminals.

The main component of the TACC compositional framework is the *worker*. Workers are the building blocks of TACC applications. They are combined together according to a programming model that enables chaining (similarly to the chaining of processes in a Unix pipe) and the invocation of one worker from another as a subroutine or a coroutine. The TACC programming model provides a set of generic functionalities: transformation, aggregation, caching and customization of content [21,39]. The *transformation* functionality deals with various changes done by the proxy to the content of a single data object, such as 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, transformed data objects, 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 TACC software is instantiated in the context of TACC servers, which run on clusters of workstations and provide support for inter-worker calling and for chaining APIs. 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 [39].

The TACC infrastructure has been used to develop and deploy services providing Web access in resource-poor devices [38], combining searching and browsing facilities [21], providing distributed, collaborative repository access to digital music resources [39], etc.

6.2. The Ninja architecture

An evolution of the TACC model was provided in the context of the Ninja project from UC/Berkeley [43]. This project developed a robust infrastructure for Internet-scale systems and services in Java. Ninja separates the functionality required for building and deploying scalable Internet services into four types of components: bases, active proxies, units, and paths.

A *base* is the platform upon which Ninja services are deployed. It consists of a local cluster, a software environment, and a cluster-based execution environment. The software environment is designed to support high-concurrency, robustness, and the transparent distribution of data to cluster-nodes [42]. Furthermore, it provides a programming model that consists of four design patterns: *wrap*, *pipeline*, *combine* and *replicate* [43]. Service programmers can use these patterns to compose different stages of a single service. The execution environment of a Ninja base is called *vSpace*. vSpace provides facilities for service component replication, load-balancing, and fault-tolerance [43].

Active proxies are fine-grain intermediaries providing transformational support between services running on Ninja bases and terminal devices. Software running in the context of active proxies performs dynamic service adaptation, data distillation, protocol adaptation, caching, encryption, etc. Examples of active proxies include wireless base-stations, network gateways, firewalls, and caching proxies.

Paths represent an abstraction that facilitates service composition out of existing service components. A path is comprised of a sequence of intermediary entities: *operators* and *connectors*. Operators process data and connectors pass data between operators, conducting protocol translation when necessary. Operators come with a strict definition of the input they accept and the output they provide. There are two types of operators in the Ninja architecture: long-lived and dynamically created. Long-lived operators are standard Ninja services deployed at a Ninja base. Dynamically created operators run in active proxies and implement the various transformational operations required to adapt the data into a format acceptable by the next service or device along the path. 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.

6.3. WBI

A programmable framework for building intermediary applications is the *Web Browser Intelligence* or *Web Intermediaries* (WBI) by IBM Almaden [13,14,53]. The design of WBI focuses on providing a simple approach for assembling complex intermediary systems from simpler components. The goal is to enable application developers to focus on the programming of each individual component's functionality, by providing the glue that will assemble those components together [13]. 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 [14].

The WBI approach 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). WBI components operate on information as it flows along the information stream, performing various transformations. WBI building blocks fall into five basic categories: request editors, generators, document editors, monitors, and autonomous functions [14]: *Request editors* receive and modify requests before forwarding them toward 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. Finally, *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 the data paths that transform information flowing from origin servers to users and implement different application scenarios. WBI constructs data paths dynamically, using a *rule-based* approach that determines which MEGs must be invoked and in what sequence.

WBI plugins can be installed both on client machines and on any other networked machine, possibly near the origin servers. Multiple client-side and server-side WBI intermediaries can cooperate to establish one WBI service. Since WBI components are programmable, the intermediaries developed with WBI can support both synchronous and asynchronous interactions and various application-level communication protocols.

6.4. eRACE

The *extensible retrieval*, annotation and caching engine (eRACE) is an infrastructure designed to support the development of intermediaries that provide personalized services over a variety of devices (mobile, thin clients, desktops) [5,33]. eRACE is driven by XML-encoded *eRACE profiles*, maintained within its infrastructure. eRACE profiles represent the personal interests and service characteristics of each user, or the structure of a portal-service built on top of eRACE and made available to its end-users. Information collected by eRACE is stored in a software cache for further processing (filtering, aggregation, etc.), personalized dissemination to subscribed users, and wide-area dissemination on the wireline or wireless Internet. eRACE supports ubiquitous service provision thanks to the decoupling of content publishing and distribution from information retrieval, storage, and filtering. Because of the explicit management of eRACE profiles, which comprise service-related information, the infrastructure can also incorporate mechanisms for providing subscribed users with differentiated service-levels.

eRACE is organized as a two-tier, distributed architecture. The first tier includes components that manage services provided to users: the *service manager*, *content-distribution agents*, and *personal information roadmap* servlets (PIR). The second tier of eRACE consists of a number of protocol-specific *agent-proxies*, an *object cache* that stores multiple versions of retrieved resources, and an *annotation engine* that indexes collected resources, executes user-queries, and produces user alerts, encoded in XML. eRACE comprises agent-proxies like WebRACE, mailRACE, newsRACE and dbRACE that retrieve and cache information from the Web, POP3 email-accounts, USENET NNTP-news, and Web-database queries respectively. The most important proxy is WebRACE, the agent-proxy that deals with information sources on the Web and is accessible through the HTTP protocol [33]. WebRACE is developed in Java and consists of a distributed crawler [78], and an object cache. Other proxies have the same general architecture with WebRACE, differing only in the implementation of their protocol-specific proxy engines.

eRACE's structure supports information provision according to different access models (pull or push), protocols (HTTP, SMTP, WAP, GSM/SMS), and tailored to various client-device capabilities (PC, PDA, mobile phone, thin clients).

6.5. iMobile

Building mobile services from proxy components is the main goal of the *iMobile* project of AT&T Research [59]. Similarly to WBI, iMobile provides a framework for assembling and configuring intermediary components into new, distributed intermediary applications. These applications are executed in the context of *iProxy*, a programmable proxy server designed to host agents and personalized services developed in Java [60]. Additional programmability is provided by iMobile's support for user and device profiles. In a nutshell, 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 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 [46]. The devlet sends iMobile-specific commands to the iMobile server (the "let engine"), to invoke an iMobile applet that represents 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, iMobile *applets* are modules that process and aggregate content retrieved by different sources and relay 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, re-directs them to the right infolet or applet, transcodes the result to an appropriate terminal-device format and re-directs it to the terminal device via the proper devlet.

6.6. Intermediaries for dynamic content provision

The emphasis of the intermediary infrastructures presented above is on providing mechanisms for the specification and execution of intermediary tasks that involve the assembly, caching, customization, presentation, and delivery of content. These mechanisms, however, have limited applicability when it comes to origin servers that provide dynamic content extracted and assembled from back-end databases and application servers, according to proprietary business rules.

Typically, dynamic content is non-cacheable. Also, as described earlier, the process of dynamic-content creation can lead to an overloading of origin servers and to a deterioration of user-perceived quality of Web-service provision. To cope with these problems, several projects have proposed the off-loading of application-server logic onto intermediary systems deployed throughout the network [6,10,23,51].

An early example of an intermediary system that provides application off-loading is the *active cache* intermediary [23]. The active cache is a Web proxy that caches and executes code supplied from origin servers. To this end, the system employs a construct called *cache applet*, that is a server-supplied software component written in Java and attached to a Web resource and a URL. Whenever the active cache intermediary caches a document, it also fetches and stores the corresponding cache applet. When a user-request hits on a cached document, the intermediary can invoke the associated cache applet. The cache applet runs and determines what the intermediary will send back to the user: either a new document created on-the-fly by the applet, or a document cached at the intermediary's storage, or a new document fetched by the applet from the origin server.

The active cache intermediary has special provisions for security and resource management in order to avoid potential problems from malicious cache applets [23]. In particular, cache applets implement a Java interface with minimum functionality that allows them to check the availability of a document in the cache, to read and write a document, and to communicate with their origin server. Security restrictions are enforced with the general security mechanisms of Java, through static checking of cache-applet bytecodes by the active proxy, and with monitoring of an applet's resource consumption.

Using the active cache, an application server can off-load part of its processing to intermediary nodes close to the end-user. The active-cache approach, however, does not provide any general abstractions for specifying which parts of a Web service will be encapsulated into a cache applet, the cache consistency policies that a cache applet should follow, or the off-loading of business logic to more than one cooperating cache applets.

More recent research efforts focus on Web applications with a three-tier architecture, which consists of a presentation, a business logic, and a database tier [77]. Several projects study the off-loading of different parts of these tiers to the intermediary infrastructure [10,41,77]. To this end, they adapt techniques applied previously in the performance enhancement of dynamic-content generation at origin Web servers; for example, through the specification of Web-pages as structured collections of information fragments. These fragments are associated with policies for the retrieval, caching, and consistency-maintenance of Web-page content. The assembly of fragments into dynamic Web-pages and the implementation of policies regulating the storage of fragment-contents is conducted at the intermediaries. Such an approach is implemented in commercial systems like the *Websphere Edge Services* of IBM [3] and the *EdgeSuite* content distribution network of Akamai [6], which place portions of the presentation and business logic tiers to intermediary servers near the network's edge.

The EdgeSuite network of Akamai is based on the *Edge Side Includes* (ESI) specification accepted by the World Wide Web consortium [2]. The ESI specification consists of: (i) an XML-based markup language, which is used to specify Web-page structure and contents; (ii) the content invalidation specification, which defines the rules used to invalidate content stored at ESI intermediaries; (iii) the architecture specification, which provides directives on the use of HTTP headers for the control of ESI intermediaries; and (iv) the Java ESI (JESI) Tag Library Specification, which provides a Java-based API for manipulating ESI content. The ESI markup language specifies dynamic Web-pages as templates that consist of hierarchies of fragments [2]. To this end, it provides tags that specify inclusion of fragments within other fragments, conditional inclusion, cache and access profiles of fragments, and revalidation rules. The core functionality expected from ESI-compliant intermediaries comprises the ability to retrieve and include fragments that make up a Web page and the processing of rules that specify conditions under which fragment retrieval and assembly occur. Also, the access and use of environmental variables supported by the CGI and cookie specifications and the handling of exceptions and errors.

The EdgeSuite network consists of ESI-enabled origin application servers and intermediaries (edge servers). EdgeSuite intermediaries act as surrogate proxies with a functionality that can be configured via the ESI mechanisms. An EdgeSuite intermediary operates as follows: Upon receipt of an end-user request for some Web-page, the intermediary fetches and caches the corresponding Web-page template from its origin server; subsequent requests for the same page that are intercepted by the intermediary, trigger an interpretation of the cached template according to its associated cacheability profile. Communication between EdgeSuite origin servers and intermediaries uses an optimized HTTP with persistent connections and traffic compression. Furthermore, EdgeSuite implements the content invalidation protocol provided by ESI, which allows ESI-enabled origin servers to send invalidation messages and overwrite information replicated in the intermediaries.

7. Taxonomy and open research issues

7.1. A classification of intermediary systems

From the discussion of intermediary systems presented in previous sections, we can observe that existing intermediaries lack the necessary semantics, and have limited programming and system support, to be used as open overlay networks. Furthermore, they do not provide adequate support for adaptability and reconfigurability, which are key requirements in the context of emerging pervasive Internet systems and telecommunication technologies beyond 3G [63].

In particular, proxy intermediaries are centralized software modules that relay HTTP messages and cache static Web pages, without processing or modifying the relayed content. In essence, they operate as special-purpose systems with a "hard-wired" functionality that does not provide any substantial support for reconfiguring or adapting their operation, for instance through configuration parameters or programming constructs.

Notification intermediaries are driven by end-user profiles and operate even when the end-user is disconnected. They provide support for the semantic analysis and filtering of relayed content. Their implementation incorporates algorithms for information retrieval, personalization, content ranking, etc. Most notification intermediaries operate as tightly integrated, centralized servers. Some systems provide a limited support for a reconfigurable behavior, through configuration parameters or policy metadata.

Intermediaries for wireless access and mobile clients typically implement a common set of functionalities: they operate as gateways between the HTTP and wireless protocols, they customize content for resource-limited client devices and wireless access, and they provide basic support for disconnected operation of mobile clients. However, most systems are tailored to the needs of a particular wireless protocol or of a family of mobile devices. Their implementation relies either on tightly integrated servers with a complex, proprietary architecture, or on lighter cooperating agents with a limited functionality. Their design, provides limited support for adapting to changing conditions of use, for operating with multiple wireless protocols, for customizing the content relayed according to the needs of different terminal devices, and for sustaining peak loads. Intermediaries for mobility and wireless access are installed usually under the administrative domain of the wireless service provider. They do not cooperate with intermediary entities of other domains in order to establish overlay networks of service modules in an open network setting.

Recent research efforts sought to develop intermediaries with provisions for performance scalability, high availability, support for dynamic content, and a programmable functionality. The long-term goal of these efforts has been to develop open overlay networks, deploy them on top of the Internet, and provide infrastructural support for the implementation of new services. In order to provide performance scalability and high availability, intermediary infrastructures have modular system architectures, comprising components that can be distributed in a cluster environment or in wide-area networks. A number of intermediary infrastructures support some level of dynamic service adaptability by maintaining and managing metadata about content structure, user profiles, policies of use, service characteristics, etc. Additional programmability is supported with the specification of core software components, and of programming abstractions enabling the composition thereof into commonly used patterns. The exact functionality of individual components can be programmed with a programming language like Java. In some cases, the choice of compositional patterns takes place at run-time, based on the evaluation of rules described in some configuration language. Finally, communication between intermediary components, clients, and servers relies upon application-level protocols and does not allow the definition of messages with richer semantics.

In the intermediary infrastructures examined above, the location of components is determined mainly at deployment time and confined to one

Table 1
Key features of intermediary systems

	Web proxies	WAP gateway	Palm clippings	AIDE	WBI	TACC	eRACE	EdgeSuite
Functionality								
Customization	–	–	–	–	✓	✓	✓	✓
Filtering	–	–	–	✓	–	–	✓	–
Annotation	–	–	–	–	✓	–	✓	✓
Transcoding	–	–	✓	–	✓	✓	✓	–
Protocol translation	–	✓	✓	–	✓	✓	–	–
Content creation	–	–	–	✓	✓	✓	✓	✓
Authorization	Client or server	Client	Client	Client	Client or server	Server	Client	Server
System architecture								
Structure	Centralized	Centralized	Centralized	Centralized	Distributed	Distributed	Distributed	Distributed
Component deployment	Network	Network	Network and client	Network	Network, client, server	Network and server	Network	Network
Content caching	✓	✓	limited	✓	✓	✓	✓	✓
Crawling support	–	–	–	✓	–	–	✓	–
Archiving	–	–	–	✓	–	–	✓	–
Configurability	–	–	✓	–	✓	–	✓	✓
Programmability	–	–	–	–	✓	✓	–	–
Interaction								
Wireless support	–	✓	✓	–	✓	✓	✓	–
Proxy-server protocol	HTTP	HTTP	HTTP	HTTP	HTTP	HTTP	HTTP, NNTP, SMTP	Optimized HTTP
Client-proxy protocol	HTTP	WSP	UDP and compressed messages	HTTP, SMTP	HTTP	Wireless protocols	HTTP, SMTP, GMS/SMS	HTTP
Access model	Pull	Pull/push	Pull	Pull/push	Pull/push	Pull	Pull/push	Pull/push
Communication mode	Synchronous	Synchronous	Synchronous	Asynchronous	Synchronous, Asynchronous	Synchronous	Asynchronous, Synchronous	Synchronous, Asynchronous

administrative domain. Possibilities for dynamic migration, replication, or off-loading of components to the network are limited and of a narrow scope. Components do not encode their interfaces in some standardized description or publish them in directories. Therefore, they cannot be located by third parties searching for service bindings. Moreover, available programming constructs are tailored to the scope of each different intermediary system, rather than being general-purpose. Most intermediaries studied do not explicitly manage information and metadata about their execution status, i.e., request rates, pending operations, service throughput, available bandwidth, response times of origin servers, network latency, resource utilization, etc. Any descriptions of profiles, configuration parameters, and component-structuring specifications are system-specific and do not follow common and widely adopted standards. There is an absence of interfaces for exporting services to other intermediary systems and of primitives for specifying arbitrary data aggregations and transformations out of third-party origin servers or service components. Furthermore, existing intermediaries have limited interaction with lower layers of the communication-protocol stack, their underlying operating system, and execution environment. Consequently, it is difficult to specify, using programming abstractions, algorithms for the dynamic adaptation of intermediary behavior to changing infrastructure conditions.

A summary and classification of the characteristics of selected intermediaries is presented in Table 1. The classification is organized along three dimensions: intermediary functionality, architecture, and interaction with clients and origin servers. Each dimension is refined further according to particular features of relevance.

7.2. Research issues

The future of intermediary systems for the Web is driven by "next-generation" Internet infrastructures that emerge from the convergence of Internet with 2.5–4G wireless systems. These infrastructures entail heterogeneous access networks, network overlays providing caching and content distribution, and a variety of client devices (see Fig. 4). Behind the push towards these infrastructures lies the vision of providing end-users with any service through any client, anytime and anywhere. To meet this vision, we need to establish service-development frameworks and deployment infra-

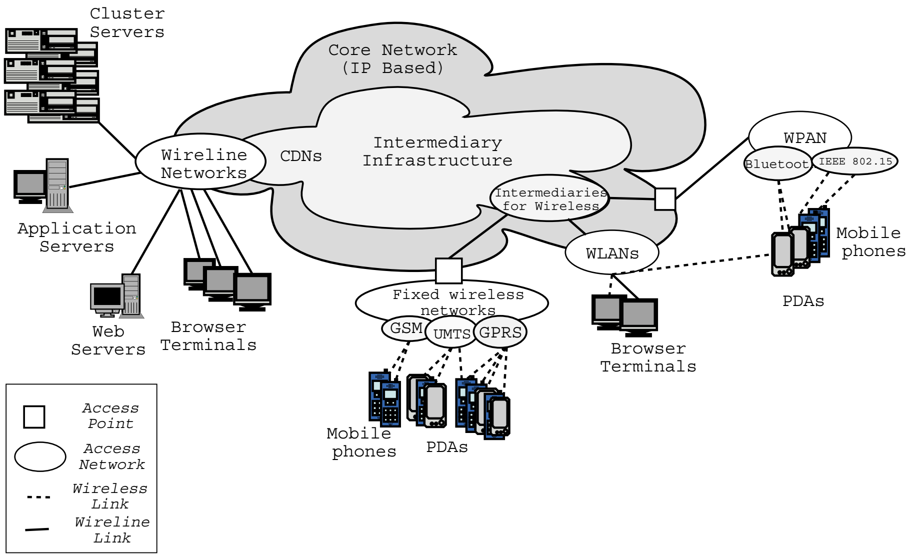

Fig. 4. The communication infrastructure of next-generation Internet services (adapted from [57]).

structures that [40,55,72]: (i) Enable the production and/or assembly of new services from existing, basic services and "off-the-shelf" components. (ii) Support services that are dynamically adaptable to personal interests, context, location, terminal device, access network, and bandwidth. Moreover, to changing conditions, such as resource-demand and capacity allocation, deployment of new services or service components, failures of software and hardware. (iii) Provide services seamlessly to terminal devices in the presence of low wireless coverage and high mobility, and maintained during hand-overs even across heterogeneous access networks. Many research challenges need to be addressed in order to develop such infrastructures. Some of these are described below.

Programmability: Programming constructs of existing intermediaries do not have the necessary generality, expressiveness, and power to enhance the development and maintenance of adaptable, high-performance, scalable, and open systems. Therefore, new programming models, languages, APIs, and generic compositional frameworks are required for the programmatic description, at a reasonably high level, of new intermediaries. Such programmatic descriptions should be amenable to compilation into executable code, which will be deployed at execution environments distributed throughout the intermediary infrastructure (e.g., virtual machines). Intermediary components should also have well-defined interfaces registered with public registries, to enhance their discovery and composability, possibly following Web services standards like WSDL and UDDI [28]. The programming of generic components could be performed using existing programming languages, such as Java, with calls to service APIs of the system infrastructure.

Communication support: Interaction between existing intermediaries, origin servers, and client systems employs application-level protocols, such as HTTP, SMTP, WAP, and Java RMI. As we move towards intermediaries defined out of interacting components in an open, distributed setting, we need to come up with general mechanisms for specifying and standardizing richer interaction semantics between intermediary components, origin servers, and diverse clients. These semantics must be independent of the various underlying communication protocols. The XML-based SOAP protocol adopted by Web services represents a step towards this direction, as it supports the definition of arbitrary message-types carrying messaging or remote procedure-call information, on top of existing application-layer protocols (HTTP, SMTP, etc.) [28].

Adaptability: Dynamic (run-time) adaptation to changing conditions in the service infrastructure and the context of use is one of the major challenges for next-generation Internet infrastructures [63]. Various issues arise in this context: monitoring the performance of service components and adapting accordingly the resources used, enabling the dynamic off-loading of service components to the infrastructure, moving execution state across diverse platforms, migrating services within the infrastructure, dynamic adaptation to different client devices, etc. [55,63].

Advances in pervasive computing and wireless Internet will shift the scope of intermediary systems from adapting and aggregating content retrieved from origin servers of the Web, towards dynamic mediation between service providers and diverse client devices over wireline and wireless connections. Consequently, intermediary infrastructures will be transformed into distributed environments with capabilities that can be located and retrieved dynamically according to changing application needs, to available resources, etc. These infrastructures will support the discovery of service components, the dynamic composition of new services on-demand, and the adaptation to changing conditions of use, placement, behavior, resource consumption, etc.

Support from the middleware: To cope with adaptability requirements, future programming frameworks for intermediary systems must provide support for the explicit description and management of adaptation conditions, such as user context, resource availability, performance, and measured QoS. Additional support will be required at the programmatic level for requesting resources according to usage needs, authenticating third parties that request service, using persistent components that provide standard data management facilities (caching, indexing, filtering, metadata extraction, garbage collection), implementing policies for QoS, security, access, etc. In summary, future intermediary systems will require extensive support from emerging middleware infrastructures. Providing this support is a difficult challenge as it presumes:

- The capability of the middleware to collect, manage, and export a variety of information collected from different layers of the communication protocol stack [63].
- The standardization of middleware APIs through which intermediary components can exchange information with the middleware, locate service components, retrieve code and data, etc.
- The integration of key middleware components with the telecommunication infrastructure at a wide scale and the deployment of middleware services and intermediary execution frameworks to nodes of the infrastructure.

7.3. A general framework for intermediary infrastructures

Several of the open issues mentioned above can be addressed in the context of the framework proposed by the IETF *open pluggable edge services* (OPES) working group, which focuses on the specification of intermediary services in open, distributed settings [4,7].

The OPES group is introducing a general reference architecture for distributed intermediary (edge) services [11]. The basic concepts of this architecture are: entities, flows, and rules. An *OPES flow* is a sequence of message exchanges between an origin server and a client system at the application layer of the Internet. An *OPES entity* is a process operating on OPES flows. OPES entities reside in *OPES processors* deployed at hosts throughout the Internet [12]. There are two types of entities: *service applications* or *proxylets* and *data dispatchers* or *OPES engines*. A proxylet encapsulates the transformation logic applied by an OPES intermediary to messages exchanged between clients and servers passing through that intermediary. A data dispatcher is a policy enforcement point that decides which proxylet to invoke on a particular flow of messages. To make decisions about proxylet invocations, data dispatchers maintain state information, use application-specific knowledge, and evaluate a set of *OPES rules* that consist of conditions and related actions. OPES rules must be encoded according to a standardized schema expressed in some common policy language [17]. In some cases, the functionality provided by an OPES intermediary can be implemented outside its application-service entities, through remote-procedure calls initiated by the data dispatcher to one or more remote *call-out servers*. Communication between the data dispatcher and call-out servers must be carried according to the *OPES call-out protocol* (OCP), which specifies the requirements for the communication protocol between a data dispatcher and a call-out server [16].

One of the key aspects of the OPES architecture is to enable the deployment of OPES intermediaries at network hosts that belong to different administrative domains. Consequently, the OPES architecture has to address issues such as verifiability, security, authentication, authorization, and accounting. To ensure the verifiability of OPES operations, the architecture requires the support of operation tracing. To this end, each data dispatcher must support the annotation of messages exchanged in the context of an OPES flow with information about the OPES services operating on that flow. Moreover, the architecture requires that operations conducted upon a given flow are explicitly authorized by either the origin server or the end-user (client system) involved. The architecture also requires that an OPES intermediary is not hidden from the end-user or his client system. Thus, the IP address of an OPES processor must be known to and be directly accessible by the end-users invoking its services. This requirement, however, does not preclude the chaining of OPES processors with only the first one in the chain being exposed to the end-user [11].

The OPES architecture dictates that OPES entities and call-out servers implement a trust policy describing which parties are trusted to operate on data and what security requirements are required for communication. Trust can be delegated for various levels of data granularity.

Delegation starts at either an origin server or a client system and moves to other entities in a stepwise manner, creating "trust domains" that expand across different administrative domains. OPES processors are required to maintain an explicit representation of their trust domain and to report it for tracing purposes.

The OPES working group has also proposed a reference system architecture for creating OPES overlay infrastructures on top of the Internet and for supporting the provisioning of added-value services [68]. According to this proposal, an OPES intermediary comprises three basic components: the data dispatcher (OPES engine), the *proxylet run-time system*, and the *remote call-out system*. A typical dispatcher must include a message parser, a rule processor, and rule modules storing applicable policies.

The proxylet run-time system is the environment that executes proxylet entities; it also provides a number of libraries offering core intermediary functionalities and a mechanism for hosting loadable proxylets. Typical proxylet-library functions include HTML parsing, crawling, caching, archiving, logging, etc. In addition to the proxylet library functions, the OPES system architecture supports the downloading and installation of proxylets from remote proxylet providers into a local run-time system. This process is controlled by an *OPES administration server*, which handles security issues of authentication and sandbox validation of proxylet codes. Finally, the remote call-out system is the environment used for invoking services of remote call-out servers and for handling their responses.

In summary, a typical operation of an OPES intermediary entails the invocation of a proxylet upon an OPES flow; the proxylet may in turn invoke a number of functions from the intermediary's proxylet-library, from other proxylets of the intermediary, or from remote call-out servers. The invocation of a particular proxylet upon some OPES flow is decided by the intermediary's OPES engine, which parses messages of that flow and evaluates applicable policy rules.

The OPES working group has identified key issues that need to be addressed in order to establish an IP-based, open infrastructure for general-purpose intermediary systems that span across different administrative domains. In particular, the OPES architecture:

- Incorporates provisions for the reconfigurability of intermediary behavior at run-time, based on applicable policies expressed in a commonly accepted metadata format, in combination with application-specific knowledge and state information. Metadata-mediated reconfigurability is supported by the concept of OPES processors, which evaluate OPES rules and invoke intermediary actions accordingly.
- Supports the dynamic reconfigurability and adaptability of intermediary functionality through the downloading of rule sets and code modules on demand. This is possible thanks to concepts like the proxylet run-time system, the OPES administration server, and proxylet and policy-rule providers. The proxylet run-time system supports the dynamic installation and execution of proxylet modules retrieved from local storage or from the network. The OPES administration server provides the functionality required to cope with security issues when retrieving code and rules from remote providers.
- Supports the availability of core intermediary services through the provision of proxylet libraries installed on each OPES intermediary and being accessible to the proxylet source code through API calls.
- Offers provisions for off-loading complex functionality to remote call-out servers. The implementation of a remote call-out system in each OPES intermediary facilitates the integration of remote functionality in the proxylet programming model. Furthermore, call-out functionality enables the deployment of notification intermediaries in the context of an OPES-compliant infrastructure.
- Incorporates provisions for mechanisms that manage trust and security in intermediary architectures and ensure end-to-end data integrity and end-user privacy protection.
- Enables the use of higher-level communication protocols (like OCP and SOAP), which support richer message semantics than HTTP.

Evidently, the implementation and deployment of OPES-compliant overlay infrastructures is a challenging endeavor. Many research questions have to be addressed in this context, such as: the naming and discovery of available service entities; the design of programming models for the composition of new services specified at a high level of programming abstraction; the provision of high-availability and incremental scalability through mechanisms for network-scale resource and QoS management that integrate monitoring information from different layers of the software and communication protocol stack with knowledge about applicable policies for resource-use and QoS. Also, the creation and management of dynamic, virtual organizations involving cooperating intermediary entities that belong to different administrative domains.

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. Intermediary intervention ranges from very simple chores, like relaying requests and replies and trans coding 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 system 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. Nevertheless, many open challenges have to be addressed in order to make intermediary functionalities seamlessly available, easily programmable and adjustable to the needs of service providers and end users of next-generation Internet infrastructures. These challenges arise from the need to make intermediary systems more open, flexible and fully integrated with the adaptable and reconfigurable networks of the future.

Acknowledgements

This work was supported in part by the European Union under project ANWIRE (IST-2001-38835). The author wishes to thank the anonymous reviewers for their helpful comments and suggestions.

References

[1] Akamai Inc., http://www.akamai.com.
[2] Edge Site Includes, http://www.esi.org.
[3] IBM WebSphere Application Server, http://www-3.ibm.com/software/webservers/edgeserver/.
[4] Open Pluggable Edge Services, http://www.ietf-opes.org.

[5] eRACE Project, http://www.cs.ucy.ac.cy/Projects/eRACE/, 2001.

[6] Turbo-charging dynamic Web sites with Akamai EdgeSuite, Akamai White Paper, 2001.

[7] Open Pluggable Services Working Group Charter, http://www.ietf.org/html.charters/opes-charter.html, October 2003.

[8] R. Aiken, M. Carey, B. Carpenter, I. Foster, C. Lynch, J. Mambreti, R. Moore, J. Strasnner, B. Teitelbaum, Network Policy and Services: A Report of a Workshop on Middleware, RFC 2768, IETF, 2000, http://www.ietf.org/rfc/rfc2768.txt.

[9] L. Amini, A. Shaikh, H. Schulzinne, Modeling redirection in geographically diverse server sets, in: Proceedings of the Twelfth International World Wide Web Conference, ACM Press, 2003, pp. 472–481.

[10] A.A. Awadallah, Mendel Rosenblum, The vMatrix: a network of virtual machine monitors for dynamic content distribution, in: Proceedings of the 7th International Workshop on Web Content Caching and Distribution (WCW 2002), 2002.

[11] A. Barbir et al., An architecture for open pluggable edge services (OPES), draft-ietf-opes-architecture-04.txt (work in progress), December 2002.

[12] A. Barbir et al., OPES use cases and deployment scenarios, draft-ietf-opes-scenarios-01.txt (work in progress), February 2002.

[13] R. Barrett, P. Maglio, Intermediaries: new places for producing and manipulating Web content, in: Proceedings of the Seventh International World Wide Web Conference (WWW7), 1998.

[14] R. Barrett, P. Maglio, Intermediaries: new places for producing and manipulating Web content, Computer Networks and ISDN Systems 30 (1–7) (1998) 509–518.

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

[16] A. Beck et al., Requirements for OPES callout protocols, draft-ietf-opes-protocol-reqs-03.txt (work in progress), December 2002.

[17] A. Beck, M. Hofmann, IRML: a rule specification language for intermediary services, draft-beck-opes-irml-03.txt (work in progress), June 2003.

[18] N. Bhatti, A. Bouch, A. Kuchinsky, Integrating user-perceived quality into Web server design, in: Proceedings of the 9th International World Wide Web Conference, Elsevier, Amsterdam, 2000, pp. 1–16.

[19] J. Border, M. Kojo, J. Griner, G. Montenegro, Z. Shelby, Performance Enhancing Proxies Intended to Mitigate Link-related Degradations, IETF, RFC 3135, 2001, http://www.ietf.org/rfc/rfc3135.txt.

[20] C.M. Bowman, P.B. Danzig, D.R. Hardy, U. Manber, M.F. Schwartz, The Harvest information discovery and access system, in: Proceedings of the Second International WWW Conference, 1995.

[21] E. Brewer, R. Katz, E. Amir, H. Balakrishnan, Y. Chawathe, A. Fox, S. Gribble, T. Hodes, G. Nguyen, V. Padmanabhan, M. Stemm, S. Seshan, T. Henderson, A network architecture for heterogeneous mobile computing, IEEE Personal Communications Magazine 5 (5) (1998) 8–24.

[22] E.A. Brewer, Lessons from giant-scale services, Internet Computing 5 (4) (2001) 46–55.

[23] P. Cao, J. Zhang, K. Beach, Active cache: caching dynamic contents on the Web, in: Proceedings of the IFIP International Conference on Distributed System Platforms and Open Distributed Processing (Middleware 98), 1998, pp. 373–388.

[24] J. Challenger, A. Iyengar, P. Danzig, D. Dias, N. Mills, Engineering highly accessed Web sites for performance, in: S. Murugesan, V. Deshpande (Eds.), Web Engineering Managing Diversity and Complexity of Web Application Development, Lecture Notes in Computer Science, vol. 2016, Springer, Berlin, 2001, pp. 247–265.

[25] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, P. Reed, A publishing system for efficiently creating dynamic Web content, in: Proceedings of the IEEE Infocom 2000 Conference, IEEE, Rockville, MD, 2000, pp. 844–853.

[26] I. Cooper, I. Melve, G. Tomlinson, Internet Web replication and caching taxonomy, IETF, RFC 3040, January 2001, http://www.ietf.org/rfc/rfc3040.txt.

[27] G. Coulouris, J. Dollimore, T. Kindberg, Distributed Systems. Concepts and Design, Pearson Education, Upper Saddle River, NJ, 2001.

[28] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, S. Weerawarana, Unraveling the Web services Web: an introduction to SOAP, WSDL, and UDDI, Internet Computing 6 (2) (2002) 86–93.

[29] D. Lange, M. Oshima, Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley, Reading, MA, 1998.

[30] L. Degenaro, A. Iyengar, I. Lipkind, I. Rouvellou, A middleware system which intelligently caches query results, in: Proceedings of the IFIP/ACM International Conference on Distributed systems platforms (Middleware '00), Springer, Berlin, 2000, pp. 24–44.

[31] M. Dikaiakos, 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, IEEE Computer Society, Silverspring, MD, April 1999, pp. 91–94.

[32] M. Dikaiakos, A. Stassopoulou, Content-selection strategies for the periodic prefetching of WWW resources via satellite, Computer Communications 24 (1) (2001) 93–104.

[33] M. Dikaiakos, D. Zeinalipour-Yazti, A distributed middleware infrastructure for personalized services, Technical Report TR-01-4, Department of Computer Science, University of Cyprus, December 2001.

[34] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl, Globally distributed content delivery, IEEE Internet Computing 6 (5) (2002) 50–58.

[35] F. Douglis, T. Ball, Y.-F. Chen, E. Koutsofios, The AT&T Internet difference engine: tracking and viewing changes on the Web, World Wide Web 1 (1) (1998) 27–44.

[36] Directorate General for the Information Society, Digital content for global mobile services: executive summary, European Commission, 2002.

[37] Wireless Application Protocol Forum, Official Wireless Application Protocol: The Complete Standard with Searchable CD-ROM, Wiley, New York, 1999.

[38] A. Fox, I. Goldberg, S. Gribble, D. Lee, A. Polito, 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), 1998, pp. 407–426.

[39] A. Fox, S.D. Gribble, Y. Chawathe, E.A. Brewer, Adapting to network and client variation using active proxies: lessons and perspectives, IEEE Personal Communications 5 (4) (1998) 10–19.

[40] V. Friderikos, E. Tsontis, et al., User requirements for next generation networks, Deliverable D1.1.1, November 2002, ANWIRE Project, IST-2001-38835, http://www.anwire.org.

[41] L. Gao, M. Dahlin, A. Nayate, A. Iyengar, Application specific data replication for edge services, in: Proceedings of the Twelfth International World Wide Web Conference, ACM Press, New York, 2003, pp. 449–460.

[42] S. Gribble, E. Brewer, J. Hellerstein, 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.

[43] 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, B. Zhao, The Ninja architecture for robust Internet-scale systems and services, Computer Networks 35 (2001) 473–497.

[44] Handspring, Blazer, http://www.handspring.com/software/blazer_overview.jhtml.

[45] G. Hillerson, Web Clipping Developer's Guide, Palm Inc., 2001.

[46] B.C. Housel, G. Samaras, D.B. Lindquist, WebExpress: a client/intercept based system for optimizing Web browsing, Mobile Networking and Applications 3 (1998) 419–431.

[47] A. Joshi, On proxy agents, mobility, and Web access, Mobile Networks and Applications 5 (2000) 233–241.

[48] K. Kant, 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, New York, June 2000.

[49] B. Krishnamurthy, J. Rexford, Web Protocols and Practice, Addison-Wesley, 2001.

[50] E. Levy, A. Iyengar, J. Song, D. Dias, Design and performance of a WEB server accelerator, in: Proceedings of IEEE Infocom 1999 Conference, IEEE, Rockville, MD, 1999.

[51] W.-S. Li, W.-P. Hsiung, D.V. Kalashnikov, R. Sion, O. Po, D. Agrawal, K.S. Candan, Issues and evaluations of caching solutions for Web application acceleration, in: VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, Hong Kong, China, August 20–23, 2002.

[52] Q. Lu, M. Eichstaedt, D. Ford, Efficient profile matching for large scale Webcasting, in: Proceedings of the 7th International World Wide Web Conference, 1998.

[53] P. Maglio, R. Barrett, Intermediaries personalize information streams, Communications of the ACM 43 (8) (2000) 96–101.

[54] D. Milojicic, Internet technology, IEEE Concurrency 8 (1) (2000) 70–81.

[55] D. Milojicic, A. Messer, P. Bernadat, I. Greenberg, O. Spinczyk, D. Beuche, W. Schroeder-Preikschat, Ψ-pervasive services infrastructure, in: M.-C. Shan, F. Casati, D. Georgakopoulos (Eds.), Technologies for E-Services, Proceedings of the Second International Workshop, TES 2001, Lecture Notes in Computer Science, vol. 2193, Springer, Berlin, 2001, pp. 187–199.

[56] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamrinam, R. Tewari, Cooperative leases: scalable consistency maintenance in content distribution networks, in: Proceedings of the Eleventh International World Wide Web Conference, ACM Press, New York, 2002, pp. 634–646.

[57] C. Pinart, F. Bader, C. Christophi, E. Tsiakkouri, I. Ganchev, V. Friderikos, C. Bohoris, L. Ferreira, User-centric analysis of perceived QoS in 4G IP mobile/wireless networks, in: Proceedings of the 14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC2003), September 2003.

[58] The Apache XML Project, Introduction to cocoon technologies, White Paper, 1999–2001, http://xml.apache.org/cocoon/technologies.html.

[59] H. Rao, Y. Chen, D. Chang, M. Chen, iMobile: a proxy-based platform for mobile services, in: The First ACM Workshop on Wireless Mobile Internet (WMI 2001), 2001.

[60] H. Rao, Y. Chen, M. Chen, A proxy-based Web archiving service, in: Middleware Symposium, 2000.

[61] J. Rolia, R. Friedrich, C. Patel, Service centric computing—next generation Internet computing, in: M. Calzarossa, S. Tucci (Eds.), Performance Evaluation of Complex Systems: Techniques and Tools: Performance 2002, Lecture Notes in Computer Science, Tutorial Lectures, vol. 2459, Springer, Berlin, 2002, pp. 463–479.

[62] J. Saltzer, D. Reed, D. Clark, End-to-end arguments in system design, ACM Transactions on Computer Systems 2 (4) (1984) 277–288.

[63] M. Satyanarayanan, Pervasive computing: vision and challenges, IEEE Personal Communications 8 (4) (2001) 10–17.

[64] B. Schechter, Information on the fast track, IBM Research Magazine 35 (3) (1997) 18–21.

[65] S. Singhal, T. Bridgman, L. Suryanarayana, D. Mauney, J. Alvinen, D. Bevis, J. Chan, S. Hild, The Wireless Application Protocol, Addison-Wesley, Reading, MA, 2001.

[66] Stardust.com, The ins and outs of content delivery networks, White Paper, December 2000, http://events.stardust.com/cdn/resources.htm.

[67] J. Steinberg, J. Pasquale, A Web middleware architecture for dynamic customization of content for wireless clients, in: Proceedings of the Eleventh International World Wide Web Conference, ACM Press, New York, 2002.

[68] G. Tomlinson et al., A model for open pluggable edge services, draft-tomlinson-opes-model-01.txt (work in progress) , November 2001.

[69] J. Waldo, Jini architecture overview, White Paper, http://www.sun.com/products/jini/whitepapers.

[70] D. Wessels, Squid Internet object cache, Technical report, National Lab for Applied Network Research, August 1999, http://squid.nlanr.net/Squid/.

[71] D. Wessels, K. Claffy, Evolution of the NLANR cache hierarchy: global configuration challenges, Technical Report, NLANR, October 1996, http://www.nlanr.net/Papers/Cache96/.

[72] A. Wolisz, C. Hoene, B. Rathke, M. Schlaeger, Proxies, active networks, re-configurable terminals: the cornerstones of future wireless Internet, in: Proceedings of IST Mobile Communications Summit, Galway, Ireland, October 2000, pp. 795–803.

[73] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, H. Levy, On the scale and performance of cooperative Web proxy caching, in: Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99), December 1999, pp. 16–31.

[74] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, B. Peet, in: Concordia: An Infrastructure for Collaborating Mobile Agents, Lecture Notes in Computer Science, vol. 1219, Springer, Berlin, 1997.

[75] K. Yagoub, D. Florescu, V. Issarny, P. Valduriez, Caching strategies for data-intensive Web sites, in: VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, 2000, pp. 188–199.

[76] T.W. Yan, H. Garcia-Molina, SIFT—A tool for wide-area information dissemination, in: Proceedings of the 1995 USENIX Technical Conference, 1995, pp. 177–186.

[77] C. Yuan, Y. Chen, Z. Zhang, Evaluation of edge caching/offloading for dynamic content delivery, in: Proceedings of the Twelfth International World Wide Web Conference, ACM Press, New York, 2003, pp. 461–471.

[78] D. Zeinalipour-Yazti, M. Dikaiakos, Design and implementation of a distributed crawler and filtering processor, in: A. Halevy, A. Gal (Eds.), Proceedings of the Fifth International Workshop on Next Generation Information Technologies and Systems (NGITS 2002), Lecture Notes in Computer Science, vol. 2382, Springer, Berlin, 2002, pp. 58–74.

Marios D. Dikaiakos received his M.A. and Ph.D. degrees in Computer Science from Princeton University, in 1991 and 1994, respectively. Before that, he studied at the National Technical University of Athens, where he received a Dipl.-Ing. in Electrical Engineering (Summa cum Laude, 1988). He is currently an Assistant Professor at the Department of Computer Science, University of Cyprus, where he heads the High-Performance Computing Laboratory. Before joining the Department of Computer Science at the University of Cyprus in 1998, he held a Research Associate position at the University of Washington, Seattle (2/1994–12/1995) and a Visiting Assistant Professor position at the University of Cyprus (1996). His research interests include Parallel and Distributed Systems, with an emphasis on Middleware, Web Technologies, and Performance Engineering.