2ND QUARTER 2006, VOLUME 8, NO. 2

IEEE COMMUNICATIONS SURVEYS

The Electronic Magazine of Original Peer-Reviewed Survey Articles

www.comsoc.org/pubs/surveys

A DECADE OF DYNAMIC WEB CONTENT: A STRUCTURED SURVEY ON PAST AND PRESENT PRACTICES AND FUTURE TRENDS

STAVROS PAPASTAVROU, GEORGE SAMARAS, AND PARASKEVAS EVRIPIDOU, UNIVERSITY OF CYPRUS
PANOS K. CHRYSANTHIS, UNIVERSITY OF PITTSBURGH

ABSTRACT

The shift from static to dynamic Web content has been dramatic. Dynamic Web content is facilitated by specialized cooperating component systems better known as *content middlewares*. Unlike static content, the generation and delivery of dynamic Web content introduce heavy a workload on content middlewares. To address this problem, numerous research approaches have been proposed in the literature, some of which are the driving force behind popular commercial systems, a fact that stresses the importance and applicability of this research area. This article surveys the literature during the period of 1995–2005 on accelerating the generation and delivery of dynamic Web content. It classifies the proposed approaches into taxonomies based on their underlying methodologies and practices. In order to illustrate the evolution of research, we introduce a research-charting semi-formal framework called the Caching Fragmentation Polymorphism (CFP) framework, within which we relate the surveyed approaches and depict their relationships.

Since 1995, dynamic Web content technology has been facilitating the adaptation of content on demand, completely transforming the user interactions and experiences. The dynamic generation of content has allowed Web services and applications to be personalized with respect to individual users or customized with respect to groups of users. Example Web applications of the former case include Internet banking, online stock exchange, airline ticket reservations, and e-commerce. Local sport results, news sites, and discussion groups are examples of the latter case. According to [1], those two categories of dynamic Web content cover more than 20 percent of Internet traffic each at the turn of the century and are becoming predominant nowadays.

A wide range of cooperating components, better known as content middleware systems (or simply *content middlewares*), orchestrate a multitier system architecture that enables the generation and delivery of dynamic content. A typical component setup is shown in Fig. 1. Proxies intercept client requests and deliver cached content, if certain conditions hold. Otherwise, they route the requests to the appropriate Web server, which in turn invokes one or more specific application server. The latter dynamically generate content by processing template files and querying local and/or remote databases.

Driven by the massive switch of Web applications from static to dynamic content, the tasks of generating and delivering dynamic content raise the need for more computational and network resources, respectively. Work in [2] identifies various potential bottlenecks in the n-tier architecture by simulating typical dynamic content Web applications. Their findings indicate that e-commence and catalog-related Web applications create bottlenecks at the Web server as well as the database server. For less processing-hungry applications, such as regional news and media-related ones, the bottleneck shifts toward the proxy side.

Following the introduction of the Common Gateway Interface (CGI) a decade ago, the pioneer middleware that made dynamic content possible, numerous research approaches have been proposed for accelerating the generation and delivery of dynamic content. Consequently, state-of-the-art content middleware technology is found today in many commercial products.

Motivated by the lack of a comprehensive study on this research area, our work attempts a complete review and analysis of the proposed approaches since 1995, and classifies

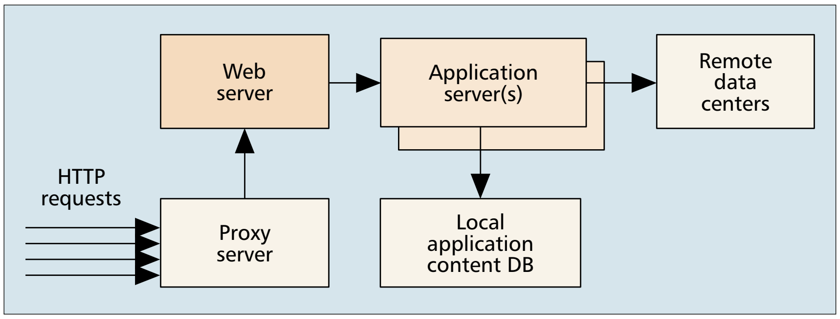


Figure 1. *The n-tier system architecture.*

them according to their underlying methodologies and practices. We identify *content caching*, *content fragmentation*, and *content polymorphism* as the three main principles on which research has evolved.

Since a performancewise comparison is rather impossible due to the tremendous number of parameters and their complexity, we compare the surveyed approaches on an abstract level by using the Caching Fragmentation Polymorphism (CFP) framework, a theoretical, semi-formal framework for charting and relating research efforts. This semi-formal framework uses the principles of caching, fragmentation, and polymorphism as its base axes by establishing approximate metrics for each one. We chart the research approaches on the framework and use it as a means of understanding the evolution of research over the past decade, and to identify current and future trends. For the rest of this article, we use the abbreviation DWC to refer to dynamic Web content, and DWP to refer to dynamic Web pages.

Given the importance and pervasiveness of the Web, and of dynamic Web content in particular, there are literally hundreds of research contributions addressing different aspects of DWC. Many of these have been published in conferences such as WWW, IEEE INFOCOM, ACM SIGMOD, ACM SIGCOMM, VLDB, IEEE ICDE, and USENIX and in journals such as IEEE TKDE. For the sake of brevity, we are citing here only those research works that either introduced key concepts for the evolution of dynamic content or had a major practical impact.

The structure of this survey is as follows: the next section presents preliminary information on modern dynamic Web content systems, which can be skipped by readers with experience on this field. We then present the comparative CFP framework and discuss its abstract metrics, followed by a taxonomy of the research approaches. We bring together and relate the surveyed approaches to the CFP framework, and then elaborate on future research trends and potential paths before we conclude the article.

Preliminaries

A chunk or fragment of dynamic Web content (DWC) is a collection of HTML code, XML data, or media content, generated on-demand by a content middleware. The generation of a DWC fragment requires the processing of a block of script code by the content middleware. The contents of the script block may also include queries on local databases and remote data retrieval. An arrangement of these fragments, according to a template file containing basic HTML, defines a dynamic Web page (DWP).

For instance, a product-customization DWP from a computer retailer site includes fragments that require the execution of thousands of lines of script code. The code issues tens of database queries and several script loops in order to build complex Web forms for product customization with price/options adjustments. The PC customization Web page of the www.higrade.com computer retailer contains approximately 2000 lines of script code with more than 20 database queries, distributed across fragments. Some script code may not produce any visible HTML output, but is necessary for performing additional background tasks such as user cart handling, credit card verification, and user session maintenance.

An application server generates a DWP by parsing its template file and, as mentioned above, by processing its embedded script blocks that relate to the DWC fragments. Template files have extensions such as ".asp," ".cfm," or ".php" that denote different scripting languages understood by the appropriate application server. The asp extension stands for Active Server Pages (ASP), a technology developed by Microsoft Corp. to support the insertion of Visual Basic code (vbscript) blocks that may generate dynamic content. The extension ".cfm" is used by ColdFusion, a product of Macromedia Inc., which uses a tagged-based script code for templates. PHP is a project of Apache Software Foundation, that supports a C-like script code.

An expanded n-tier architecture that supports DWC is shown in Fig. 2. Serving at the front line, a Web server receives HTTP requests either from proxy servers or directly from theWeb clients. Requests for static content, such as Style Sheets, JavaScript, or existing Media files are immediately served by the Web server. Requests for files (templates) with extensions that denote a dynamic content vendor (i.e., .asp, .php, and .cfm) are tunneled to the appropriate running application server for parsing. It is a common case for a public Web hosting service provider to simultaneously run multiple middlewares in order to support a wider Web developing community.

The invoked application server parses the requested template file and runs every included script code block in it. A script code block may include access to local or remote content databases, or calls to procedures provided by third-party libraries (DLLs). The complete HTML output of a parsed template file is then transmitted to the client, always through theWeb server. The application server can refrain from parsing a template file and reuse the output of a previous parse under certain conditions (we discuss DWC caching below). For alternative application servers, such as Java Servlets, a client HTTP request refers directly to a particular "runnable" object instead of template file. In this case, the object executes to either parse an existing template file, or generate all the contents of a DWP without the existence of a template file.

The CFP Framework

The majority of the surveyed approaches each employ, to some extent, three common practices, namely *caching*, *fragmentation*, and *polymorphism*, which comprise the three principles of the CFP framework. The purpose of the CFP framework is not to pinpoint the "best" approach, but to assist the reader in understanding the evolution of research. In the CFP acronym, caching precedes the other two principles as the earlier to appear, while polymorphism is the most recent. The principle of caching suggests a multitier reuse of content on network sites such as proxies, Web servers, application servers, or even at the client browser.

The principle of fragmentation suggests the breaking of a

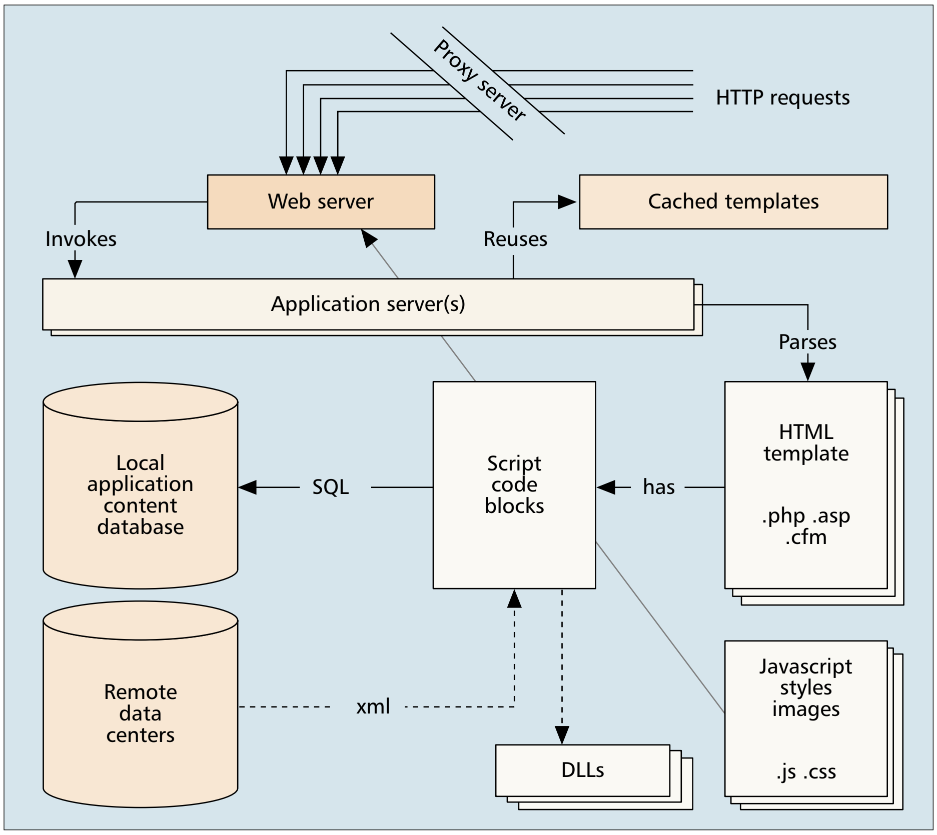


Figure 2. *Expanded n-tier architecture: cooperating system components for DWC generation.*

dynamic Web page down to computationally, but not necessarily semantically, distinct parts. This principle enables finer-grained caching and concurrency in DWC generation.

Our last principle of polymorphism allows for a dynamic Web page to be assembled in more than one way without the redundant generation of content at the server side. More specifically, the layout of the DWC fragments is decided dynamically according to, for example, the client's preferences.

In this way, polymorphism enables another dimension of content dynamism by allowing the templates to be dynamic themselves. Polymorphism can be also realized as a stand-alone practice if the script that generates a dynamic page is capable of producing the same content under arbitrary arrangements. A script (e.g., a CGI program) can achieve this by internally rearranging the layout of the content without using predefined fragments. At the absence of fragments and templates, we call this form of polymorphism *flat*.

The intuition behind the use of the CFP framework is that the three principles of the framework can be viewed as orthogonal dimensions along which different research approaches can be classified. Thus, the framework can be represented as a cube, as shown in Fig. 3. The annotations on every edge of the cube reveal the purpose of employment of each principle, as well as the outcome of combining them.

Metrics for the CFP Framework

We plot a research approach on the CFP framework, given its corresponding values for each principle. For that purpose, we define value to be the *extend of employment* of a particular principle. Since the three principles are qualitative and subjective rather than quantitative, this evaluation requires assumptions and approximations in order to define the appropriate metric for each principle. Figure 4 illustrates the basic metrics.

The metric for the principle of caching (Fig. 3) is the *proximity* of cached documents to the users. Therefore, we state that an approach which supports caching of DWC closer to Web users is evaluated higher than others that cache content closer to the Web server. The highest value for caching is received by approaches that provide support for client-side caching. The next highest value relates to proxy-side caching, and then to server-side caching.

The metric for the principle of fragmentation is the degree of support for arbitrary DWC fragments in a DWP. If an approach supports the breaking of a Web dynamic page down to any number of fragments of any type, then we assert that it employs full (or arbitrary) fragmentation. If an approach allows for any number of fragments, but supports a limited number of fragments types (i.e., only XML data or only query results) then we assert that it employs "high" fragmentation. Finally, we assert that approaches which, in one way or another, do not decouple (store separately) the fragments from the templates employ "low" or "indirect" fragmentation. We base this decision on that the fragments that are not stored separately from the template cannot be reused by other clients.

The metric for polymorphism is the degree of support for alternative arrangements of the fragments in a DWP. An approach that provides flexibility and mechanisms for arbitrary arrangements of the fragments receives the highest score for polymorphism. Approaches that provide basic support for fragment arrangement are evaluated with a lower score.

Consider two examples: The approach "X" plotted on the framework in Fig. 4 supports proxy-side caching of DWC. Also, it fully supports fragmentation since DWC fragments can be of any type and number. The approach "Y" caches arbitrary DWC fragments at the client side and provides limited support for different arrangements of the fragments.

Taxonomy of Approaches

In this section we survey the proposed approaches for accelerating the generation and delivery of DWC, emphasizing only the most representative work in the literature and commercial products. We classify these in a manner that can be intuitively mapped within our CFP framework. Although not all taxonomies are directly mapped on the CFP framework, they are included here so that we can provide a broad spectrum of research in this field.

Early and Assorted Server-side Research

Following the introduction of CGI, the primary dynamic content middleware developed for the NCSA Web server, FastCGI [3] provided support for server processes to handle consequent HTTP requests (*process persistence*). CGI process-

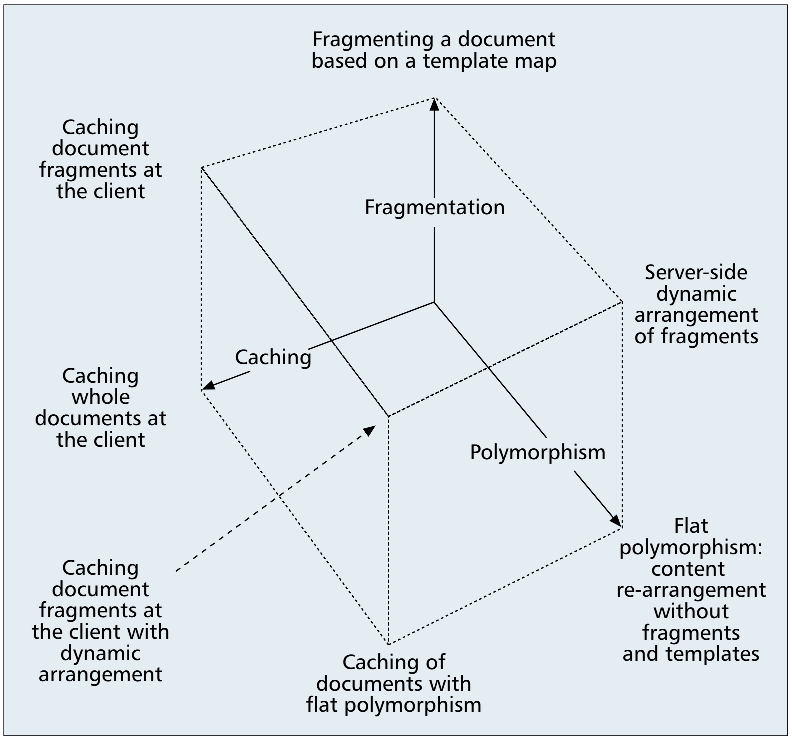

Fragmenting a document based on a template map
Caching document fragments at the client
Fragmentation
Server-side dynamic arrangement of fragments
Caching
Caching whole documents at the client
Polymorphism
Flat polymorphism: content re-arrangement without fragments and templates
Caching document fragments at the client with dynamic arrangement
Caching of documents with flat polymorphism

Figure 3. *The CFP framework.*

es were designed to terminate right after serving their first HTTP request. Similarly, [4] proposes persistent database connections for server processes to the content database.

Performance depends heavily on the middleware's implementation. The work in [5] compares various heterogeneous middlewares and reaches the conclusion that C-based middlewares outperform the Java-based ones. It verifies, however, the programmability and openness of the Java-based middlewares for generating DWC. In [6], we find a detailed study of all the Java-based middlewares for dynamic content and a thorough comparison in terms of performance and programmability.

Beyond the well-accepted and practical multithreaded architecture of Web servers, there have been significant efforts in the development of more efficient architectures to boost content generation. Flash, presented in [7], is a portable event-driven Web server that has been demonstrated to outperform both the single-process event-driven ZeusWeb server [8], and the multithreaded/multiprocess Apache Web server [9]. Its portability lies in the fact that it uses standard APIs found in any modern operating system. However, Flash was originally designed to accelerate the delivery of static content; there has been no adaptation of it for delivering dynamic content as yet.

More recently, the authors of [10] propose the use of an event-driven Web server and introduce the notion of a *stage*. According to [10], a Web application (i.e., a Web server) is built as a network of explicit computation stages connected by explicit queues whose aim is to support massive concurrency and simplify the construction of Web services. Similarly, the authors in [11] employ staged computation in Web servers, which replaces threads, and introduce a more sophisticated task scheduling mechanism. To the best of our knowledge, there is no system that uses those architectures for DWC generation.

Content Fragmentation at the Server Side

An early/indirect form of fragmentation is encountered in server-side includes (SSI). According to SSI, certain dynamic parts (fragments) of a page can be isolated and regenerated every time the page is requested. Example fragments are counters, the time at the server, and information about the requested file.

Subsequently, [12] suggested a more general form of fragmentation that allows the dissection of a dynamic page into distinct fragments that are assembled according to a template file. A fresh version of a fragment is generated every time its underlying data objects are modified, using database triggers. With the fresh fragments in place, a dynamic page can be either immediately delivered or cached (as discussed next).

More recently, [13] proposed a technique for accelerating template parsing and execution by processing the dynamic fragments of the template in a concurrent fashion. This approach achieves increased server throughput and lower client response times when the system is not fully loaded. It is worth mentioning that the identification of the fragments takes place at run time (during parsing) and requires no a priori compilation or special handling of the template. Figure 5 demonstrates a typical implementation of a dynamic document fragmentation using a template file with simple HTML commands.

The approach in [12] supports arbitrary fragmentation of DWC; however, it provides caching only at the granularity of a page. The approach in [13] supports arbitrary fragmentation and immediate execution of fragments with no caching. The former approach is more suitable for less interactive Web applications such as portals and news sites, since the generation of content is data driven (i.e., triggered by database changes). The later approach better suits interactive Web applications, such as e-commerce, where content generation is

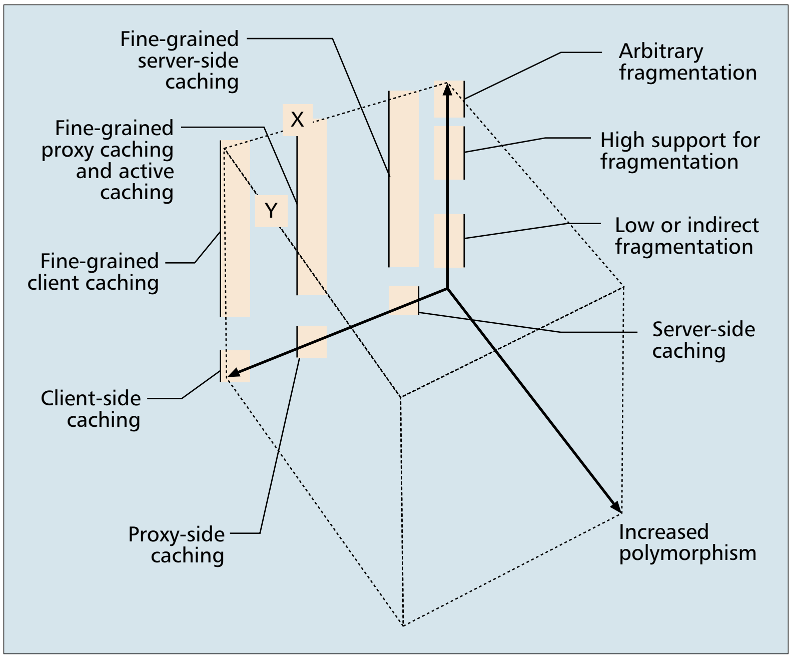
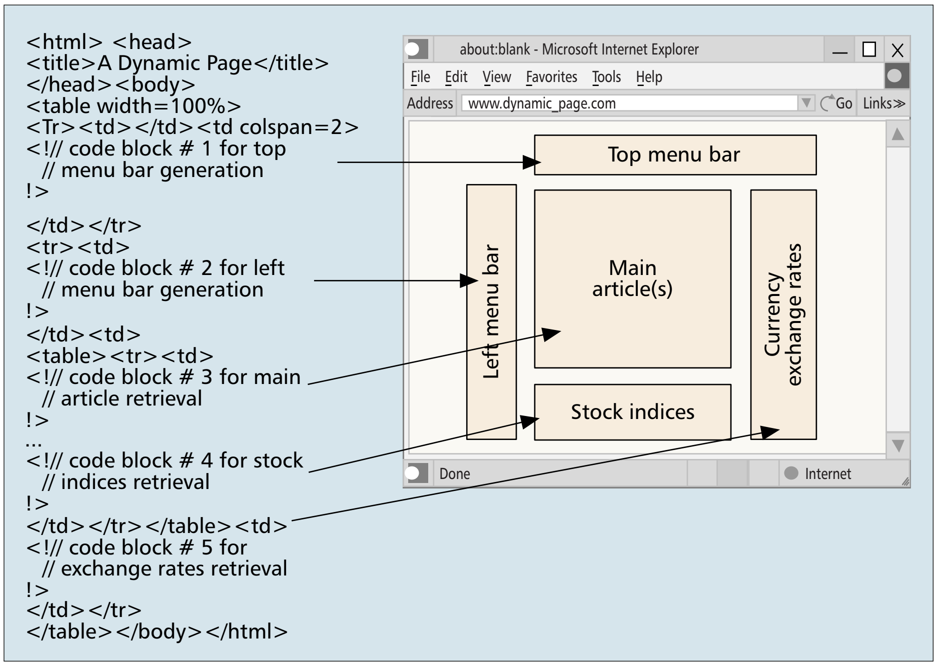

Fine-grained server-side caching
Arbitrary fragmentation
X
Fine-grained proxy caching and active caching
High support for fragmentation
Y
Low or indirect fragmentation
Fine-grained client caching
Server-side caching
Client-side caching
Proxy-side caching
Increased polymorphism

Figure 4. *Approximate metrics on the CFP framework.*



```
<html> <head>
<title>A Dynamic Page</title>
</head><body>
<table width=100%>
<Tr><td></td><td colspan=2>
<!// code block # 1 for top
  // menu bar generation
!>
</td></tr>
<tr><td>
<!// code block # 2 for left
  // menu bar generation
!>
</td><td>
<table><tr><td>
<!// code block # 3 for main
  // article retrieval
!>
...
<!// code block # 4 for stock
  // indices retrieval
!>
</td></tr></table><td>
<!// code block # 5 for
  // exchange rates retrieval
!>
</td></tr>
</table></body></html>
```

about:blank - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address www.dynamic_page.com
Go Links
Top menu bar
Left menu bar
Main article(s)
Currency exchange rates
Stock indices
Done
Internet

Figure 5. *Implementing a fragmentation.*

user driven.

Content Caching at the Server Side

Server-side content caching boosts the generation DWC by eliminating redundant server load. There are many interesting approaches for server-side caching that vary mostly on the granularity and level of caching. In [14, 15], the caching of dynamic documents at the granularity of a page was proposed for early content middlewares such as CGI, FastCGI, ISAPI, and NSAPI.

Extending their work in [15], the authors in [16] propose a DWC Caching Protocol that can be implemented as an extension to HTTP. This protocol allows for content middlewares, such CGI and Java Servlets, to specify full or partial equivalence among different URIs (HTTP GET requests). The equivalence information is inserted by the content middleware into the HTTP response header of a dynamically generated page, and stored at the caching module along with the cached page. For example, the URI "www.server.com/LADriveTo.php?DestCity=newyork" instructs the content middleware to generate a page with driving directions from Los Angeles to New York. Prior to transmitting the result page, the middleware inserts the "cache-control: equivalent result=Dest=queens" attribute in the HTTP response header. The caching module will cache the page, transmit it to the client, and store the cache-control directive for future use. A subsequent client request for the same URL, but for a different DestinationCity value, will be evaluated by the cache module for a possible match with the value of "queens" or "newyork". If a match is found, then the cached page is transmitted to the client.

Caching Content at Finer Granularities

To achieve greater reuse of cached content across both time and multiple users, caching at finer granularities is proposed. The authors in [17] suggest the caching of static HTML fragments, XML fragments, and database query results. This approach, however, applies to Web applications that follow a strict declarative definition and follow a certain implementation only. In addition, caching cannot be applied to random parts of the dynamic page and, in extend, does not allow for arbitrary fragmentation.

A more general and flexible method for fragment caching that is also easier to use was introduced in [18] and studied more thoroughly in [19]. According to this method, caching can be applied to an arbitrary fragment of a template by first wrapping it around with the appropriate tags (explicit tagging). XCache [20] is a commercial product that installs as a plug-in on popular dynamic content middlewares, and supports fragment caching of any type using explicit tagging. Also, the ColdFusion content middleware provides tags for explicitly defining the page fragment to be cached. For example, the coding `<cf_cache refresh-rate=60>` ...some script code or HTML... `</cf_cache>` caches an arbitrary fragment that refreshes every 60 seconds. In addition, other scripting languages, such as ASP and PHP, provide programming-level support for fragment-level caching at the server.

Proxy Caching

Proxy caching, better known today as Edge caching, is the most popular approach for faster delivery of reusable static content such as static HTML pages and media files [21]. A Proxy degrades bandwidth consumption by eliminating unnecessary traffic between clients and servers, given that it is strategically located. It has been identified that the usual hit ratio for proxy caches is around 40 percent [22]. Another study finds out that even when proxies are employed, approximately 40 percent of the original volume of Web traffic is still unnecessarily generated [23].

Despite the location of cached content, server-side and proxy-side caching have their major implementation difference in how data consistency between the cached content and the underlying database objects is enforced. For server-side caching, consistency is more easily enforced because the caching module is local to the content middleware (as seen in [12]). For proxy-side caching, efficient cache invalidation techniques are required, as discussed below.

Early research presented in [16] proposed the caching of dynamic content at the granularity of a page by using the abovementioned Dynamic Content Caching Protocol. The caching protocol is applicable for both server-side and proxy-based caching, and works by allowing the manipulating of HTTP header information and URL query string parameters (GET variables).

Another interesting approach for caching complete dynamic pages is found in [24]. Analogous to the caching protocol approach discussed above, this one suggests that the proxy server be allowed to examine the HTTP POST variables that are submitted as part of a client HTTP request for a URI. In brief, the proxy server attempts to reuse cached SQL query results by looking up a predefined mapping. This mapping relates the HTML form fields that are submitted with a URI request with the SQL query that uses those form fields as input. Two strong points of this work are as follows:

• The proxy can extract and reuse portions of cached query

results, if necessary, to satisfy future requests.
- It can add-on to a cached query result on demand by negotiating with the Web server.

Since the HTTP post variables are generated from HTML form fields, this approach is called *form-based*.

FINE-GRAINED PROXY CACHING

Following proxy-caching content fragmentation, caching at the granularity of a fragment is proposed for proxy caches. According to fine-grained proxy caching, the template file is cached at the proxy server, whereas its dynamic fragments are either reused from the proxy cache or fetched fresh from the Web server.

Edge side includes (ESI) was introduced as a standard for caching page templates along with their fragments on proxy servers [25]. According to ESI, the dynamic fragments of a page are explicitly marked using tag-based macrocommands inside the page's template file. An ESI-compliant proxy server must provide support for parsing the cached template file and executing macros that dictate whether a fragment should also be retrieved from the cache, or pulled from the original server. ESI macros have access to a client's HTTP request attributes (cookies, URL string, and browser used) in order to choose between fragment alternatives. An example of this would be the identification of the client's browser version or vendor in order to pick the appropriate fragment that meets the browser's capabilities.

ESI is a key component for Content Distribution Networks (CDNs), a popular caching approach that supports the leasing of cache space on a service-based network of interconnected proxy servers. A typical CDN employs a set of proxy servers strategically arranged by geographical or network location. It is noteworthy that for a Web site to be registered and served by a CDN network, an offline procedure of updating the templates of the Web site is required. A thorough survey on the procedures, practices, and performance of CDNs can be found in [26].

We state that ESI extends the approaches that support arbitrary server-side caching ([18, 19] and scripting languages such as PHP, ASP, and ColdFusion) by moving fragment caching from servers to proxies. ESI also provides some basic support for dynamic fragment arrangements through the use of a tag-based scripting language, a practice that we identify subsequently as polymorphism.

A more recent study in [27] proposes a different approach to content fragmentation and its caching. Instead of using explicit fragmentation techniques such as tagging (ESI, ColdFusion), it proposes an automatic fragment detection framework which isolates the "most beneficial" content in terms of caching. More specifically, the fragmentation is based on the nature and the pattern of the changes occurring in dynamic Web pages, and its potential use across consecutive accesses by all users. According to the authors, this approach improves the efficiency of disk-space utilization at the proxies and reduces the load on the network and the origin server.

FINE-GRAINED CACHING AT THE CLIENT SIDE

Surprisingly, the notion of assembling a dynamic page away from the original content middleware was firstly introduced in [28] not for proxy caches, but for client browsers. The proposed technique, called HPP (HTML preprocessing), requires from the client browsers the extra functionality of caching and processing a template file, containing blocks of macro-commands, prior to rendering a dynamic page. Each macro-command block generates from scratch a page fragment by manipulating local variables and strings. This idea can be viewed as the client-side equivalent to the SSI discussed above.

Extending their early work in [28], the authors of [29] propose the client-side includes (CSI) by merging HPP and ESI. In order to provide support for CSI in the Internet Explorer Web browser, the authors propose a generic downloadable wrapper (plug-in) that uses JavaScript and ActiveX. The wrapper pulls and caches at the client side the template and fragments that are associated with a requested DWP, assembles them together according to the ESI directives in the template, and finally renders the page. According to the authors, CSI is suitable for "addressing the last mile" and it better suits low-bandwidth dial-up users.

The original CSI approach, as proposed in [28], employs full caching and targets low-bandwidth clients. However, this approach does not provide full fragmentation, since the cached parts of a document cannot be reused by other ones. The improved version, as proposed in [29], supports full fragmentation by allowing arbitrary content fragments to be cached and reused by any templates at the client browser.

POLYMORPHISM: A SECOND DIMENSION OF CONTENT DYNAMISM

As discussed above, caching at the fragment level requires the existence of a page layout/template that dictates a strict arrangement for cached DWC fragments. If we loosen up this restriction by allowing for arbitrary arrangements of fragments, we introduce polymorphism (in Greek: the ability of something to show different phases/morphs) in DWC caching.

The approach found in [30] provides full support for proxy-side arbitrary polymorphism by switching between a list of available templates for a specific dynamic page. According to this approach, a client request for a dynamic page (e.g., www.server.com/page1.php?id=2) is always routed to the origin Web server and causes the execution of the original script (in this case the homepage.php). This execution is necessary for determining the desired template for page1.php at run time. The selected template is then pushed to the proxy server (if not cached there) and parsed for identifying which fragments should be reused from cache and which ones should be requested fresh from the server. The performance analysis conducted in [30] demonstrated solid bandwidth reductions when applying fragment caching; however, performance analysis for other critical metrics, such as scalability and responsiveness, remains to be seen. We believe that both the necessary routing of each request to the origin content server and the invocation of the original script can hurt client response time and server scalability, respectively. Nevertheless, the techniques introduced in this approach are an excellent starting point for further research.

We also find limited support for polymorphism in ESI. Instead of choosing from the template pool, basic ESI branching commands can reorganize parts of the layout inside a template, according to client preferences. In [30], ESI is extended by providing arbitrary support for polymorphism at the proxy. This support, however, is achieved through vendor-specific code.

As we stated earlier, polymorphism can be realized as a standalone practice without the use of content fragments with a dynamic arrangement, a practice that we call *flat polymorphism*. In this case, the script that generates an entire dynamic page must provide the right mechanisms that internally rearrange the layout of the content according to, for example, the client's needs and expectations. Unlike the approach discussed in [30], flat polymorphism is not suitable for content caching

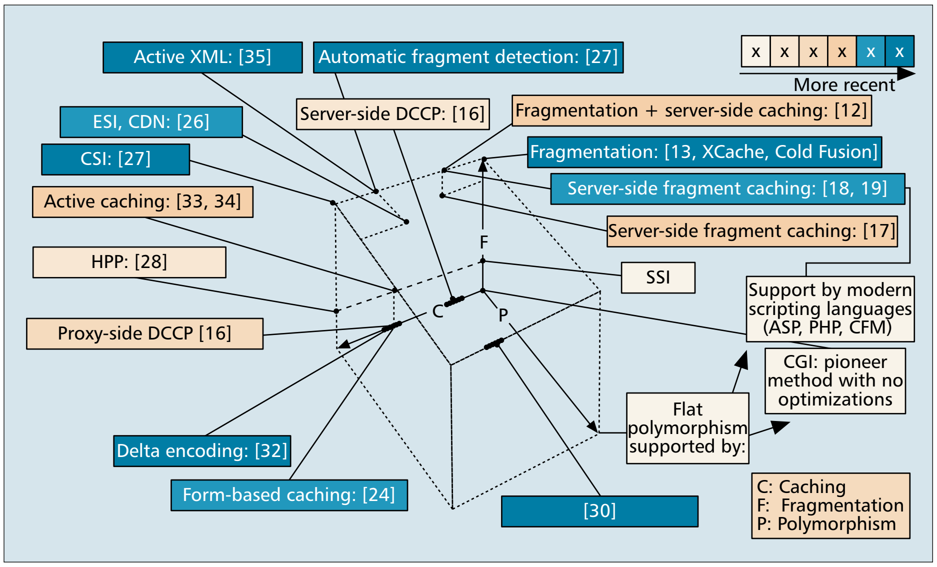

Active XML: [35]
Automatic fragment detection: [27]
x x x x x x
More recent
ESI, CDN: [26]
Server-side DCCP: [16]
Fragmentation + server-side caching: [12]
CSI: [27]
Fragmentation: [13, XCache, Cold Fusion]
Active caching: [33, 34]
Server-side fragment caching: [18, 19]
Server-side fragment caching: [17]
F
HPP: [28]
SSI
Support by modern scripting languages (ASP, PHP, CFM)
C
P
Proxy-side DCCP [16]
CGI: pioneer method with no optimizations
Flat polymorphism supported by:
Delta encoding: [32]
Form-based caching: [24]
[30]
C: Caching
F: Fragmentation
P: Polymorphism

Figure 6. *The CFP framework with the plotted proposed approaches and technologies. The numbers relate to the reference numbers in the bibliography.*

and reuse, since the entire script must execute each time a client request arrives and content chunks are not stored separately. Support for this form of polymorphism is provided by any scripting language.

Caching with Delta Encoding

Delta encoding is a popular technique for efficiently compressing a file relatively to another one called the "base" file [31]. This is achieved by computing and storing the difference between the file being compressed and the base file. Streaming media compression, displaying differences between files (the UNIX diff command) and backing-up data are common applications of Delta encoding.

Under the assumption that consecutive client requests for a specific URI would generate a sequence of moderately different dynamic pages, Delta encoding can be exploited as an alternative for caching dynamic content. In [32], the caching of a base file for each group (also called Class) of correlated documents (i.e., pages that share a common layout) is proposed. With the base file cached, the next client request would force the content middleware to compute the Delta between the new dynamic page that the client would normally receive and the base file. The computed Delta is then transmitted from the content middleware to the side where the base file is cached for computation of the new dynamic page. Eventually, the result is transmitted to the client. An interesting feature of this "class-based delta-encoding" approach is that the base file can be cached either at the server-side, at the proxy-side, or even at the client browser itself, as long as the required infrastructure exists. In the latter case, Delta encoding could benefit low-bandwidth users. Solid bandwidth savings and reduced client perceived latency are demonstrated in [32]; however, those performance gains reduce the average system throughput to 75 percent due to the increase of the CPU overhead for computing the Deltas. Nevertheless, we consider Delta encoding, in association with fine-grained caching, for caching DWC as an exciting open topic of research.

Active Caching

The notion of *active caching* refers to the ability possessed by a caching middleware to manipulate cached content instead of requesting fresh versions of it from the server. The approach found in [33] piggybacks a Java object into a dynamically generated document, which is then cached at the proxy. The proxy provides a Java runtime environment in which that object executes in order to modify the dynamic parts of the cached document according to a client's preferences. Examples of document modifications include advertising banner rotation, logging user requests, SSI execution and even Delta compression. Besides these general types of modification, the Java object can personalize cached documents by retrieving personal information from the application database at the server side. Data chunks of personal information are kept by the object for future reuse.

Building upon this approach, a more general form of DWC caching using active caching is suggested in [34]. This one is very similar to the "form-based" approach discussed above in the sense that the Java object manipulates the HTTP post variables (the Form input) for filling the dynamic parts of the cached document.

Both active-caching approaches combine the advantages of proxy-side caching while providing some support for fragmentation. They do not employ full fragmentation, since the fragments are not decoupled from the template (i.e., are not stored separately), and therefore cannot be cached and reused.

Active XML

An approach similar to active caching is Active XML (AXML) [35]. A template file designed according to AXML employs calls/references to Web Services, that look like `<sc>rentdvd.com/getPoPularDvdList()</sc>`, and return the dynamic parts of the template. A runtime is required at the location where the templates are cached that parses the templates and triggers the calls to Web services.

AXML is an alternative form of fragmentation in which the template of the dynamic page is cached at the proxy, but the fragments themselves are substituted by function calls. Another strong point of AXML is that the references to XML services embedded in a template can be reused by other templates, in this wayallowing for arbitrary fragmentation.

Capturing DWC Approaches within the CFP Framework

We plot on the CFP framework the surveyed approaches and technologies that employ at least one of the three principles (Fig. 6). This allows for a high-level comparison, as well as identification of the evolution of research.

In order to put these approaches in a chronological perspective, a relative time dimension is indicated through a gray-scale background. The darker the background of an approach, more recent the approach is. At the heart of the framework, with null values for each principle and a white background, we place CGI as the primary middleware that employs no optimizations.

An immediate observation from Fig. 6 is that the evolution of research with regard to DWC has been toward refining the employment of the three principles, and finding ways of combining them. In other words, DWC tends to be cached closer to clients (for faster response times) at finer granularities (for higher caching reuse), and served to clients under various possible arrangements (to support personalization). It seems that this evolution parallels that of other interactive, online systems such as operating systems and database servers.

The evolution in caching arises first in response to the increasing processing bottleneck caused by redundant generation of content at the server side, and then the increasing number mobile users with limited bandwidth. Advances in fragmentation arise mainly in response to the diversity and complexity of content that modern Web applications support. For example, a typical online booking Web page may include information for all alternative flights itineraries, recommended hotels and upcoming events at the destination, currency exchange rates and car rentals for the country being visited, and excursion packages. Consequently, this diversity requires the dissection of a dynamic page down to content fragments with different caching characteristics. Another important reason for the introduction of fragmentation is performance, since the fragments can be concurrently processed at the server side. Finally, polymorphism is driven by the need to further personalize dynamic pages with fragments of personal interest under user-selected layout. Examples can be found in the Yahoo! Web site and, more recently, in Google's News section.

WHAT'S NEXT? FUTURE TRENDS

Over the past ten years, DWC technology has evolved across the three dimensions of caching, fragmentation, and polymorphism. During the early stages, the need for increased response times and reduced network utilization motivated the evolution of caching strategies, while the diversity of modern Web applications stimulated the research on fragmentation and polymorphism. Modern Web applications, however, are always in need of more efficient and sophisticated means of DWC delivery. Two open challenges that can stimulate further research, in our opinion, are the following:

- *More Efficient User Authentication*: The procedure of user authentication is a typical application of DWC that has the overhead of secured server-side communication and computation. A possible optimization would be the migration of the authentication procedure from the server side closer to the client side. Still, this requires the secure shipping and hosting of vendor code and application data from the server to (for example) the proxy side. As we have seen in the "Taxonomy of Approaches" section, there exist a number of approaches that are a few steps away from achieving this. Nevertheless, an open-standard middleware that migrates both code and data to proxies, or even to the client itself, remains to be seen.
- *Cached Web Transactions*: Since data replication at the proxy side is currently exploited for read-only purposes, client requests for data updates (i.e., an order placement or a message post) require server-side processing, proxy cache invalidation, and database update. The open challenge of supporting cached transactions without the immediate intervention of the server and the database extends the user-authentication example by requiring that the cached data be writable and that cached vendor code implements the appropriate database consistency model between cached and server-side data. Potential solutions must challenge important issues, such as limited service availability, imposed by a strong consistency model.

Other future research paths are identified by combining or refining the proposed techniques. For instance, we have seen above that Delta encoding is a promising approach toward DWC caching, especially for low-bandwidth users; still, it does not support content fragments. Consequently, a fine-grained adaptation of Delta encoding to meet modern Web applications should prove to be an interesting research topic.

CONCLUSION

This survey has brought together a decade of research on dynamic Web content (DWC), and compared it in a structured and abstract manner with the use of the CFP framework. The framework is based on the principles of DWC caching, fragmentation, and polymorphism as the three main paths on which research has evolved. Based on our survey findings, future trends and paths in this research have been discussed and two open challenges, namely, efficient user authentication and caching of Web transactions, have been identified. In the near future, we expect to see a new wave of research proposals on this very interesting topic of dynamic Web content acceleration. We hope that this survey can serve as a point of reference for these proposals and new research studies.

REFERENCES

[1] A. Feldmann *et al.*, "Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments," *IEEE INFOCOM Conf.*, 1999, pp. 107–16.
[2] C. Amza *et al.*, "Specification and Implementation of Dynamic Web Site Benchmarks," *IEEE 5th Annual Wksp. Workload Characterization*, 2002.
[3] Fastcgi white paper from open market, Inc, http://www.fastcgi.com/devkit/doc/fastcgi-whitepaper/fastcgi.htm
[4] Y.-H. Liu *et al.*, A Distributed Scalable Web Server and Its Program Visualization in Multiple Platforms," *IEEE ICDCS Conf.*, 1996, pp. 665–72.
[5] E. Cecchet *et al.*, "Performance Comparison of Middleware Architectures for Generating Dynamic Web Content," *Middleware Conf.*, 2003, pp. 282–304.
[6] S. Papastavrou *et al.*, "An Evaluation of the Java-Based Approaches to Web Database Access," *Int'l. J. Cooperative Info. Sys.*, vol. 10, no. 4, 2001, pp. 401–22.
[7] V. S. Pai, P. Druschel, and W. Zwaenepoel, "Flash: An Efficient and Portable Web Server," *Proc. USENIX 1999 Annual Technical Conf.*, 1999, pp. 199–212.
[8] The NCSA Zeus Web server, http://www.zeus.com
[9] The Apache Web server, http://www.apache.org
[10] M. Welsh *et al.*, "SEDA: An Architecture for Well-Conditioned, Scalable Internet Services," *Symp. Operating Systems Principles*, 2001, pp. 230–43.
[11] J. R. Larus and M. Parkes, "Using Cohort Scheduling to Enhance Server Performance," *LCTES/OM*, 2001, pp. 182–87.
[12] J. Challenger *et al.*, "A Publishing System for Efficiently Creating Dynamic Web Content," *IEEE INFOCOM Conf.*, 2000, pp. 844–53.
[13] S. Papastavrou *et al.*, "Fine-Grained Parallelism in Dynamic Web Content Generation: The Parse Dispatch and Approach," *CoopIS/DOA/ODBASE Conf.*, 2003, pp. 573–88.
[14] A. Iyengar and J. Challenger, "Improving Web Server Performance by Caching Dynamic Data," *USENIX Symp. Internet Technologies and Systems*, 1997.
[15] V. Holmedahl, B. Smith, and T. Yang, "Cooperative Caching of Dynamic Content on A Distributed Web Server," *IEEE Int'l. Symp. High-Performance Distributed Computing*, 1998, p. 243.
[16] B. Smith *et al.*, "Exploiting Result Equivalence in Caching Dynamic Web Content," *USENIX Symp. Internet Technologies and Systems*, 1999.
[17] K. Yagoub *et al.*, "Caching Strategies for Data-Intensive Web Sites," *VLDB*, 2000 pp. 188–99.

[18] A. Datta *et al.*, "A Comparative Study of Alternative Middle Tier Caching Solutions to Support Dynamic Web Content Acceleration," *VLDB*, 2001, pp. 667–70.
[19] A. Datta *et al.*, "Dynamic Content Acceleration: A Caching Solution to Enable Scalable Dynamic Web Page Generation," *SIGMOD Conf.*, 2001, p. 616.
[20] Xcache: The Cache Management Solution, http://www.xcache.com
[21] J. Wang, "A survey of Web Caching Schemes for the Internet," *ACM Comp. Commun. Review*, vol. 25, no. 9, 1999, pp. 36–46.
[22] A. Wolman *et al.*, "On the Scale and Performance of Cooperative Web Proxy Caching," *Symp. Operating Systems Principles*, 1999, pp. 16–31.
[23] N. T. Spring and D. Wetherall, "A Protocol-Independent Technique for Eliminating Redundant Network Traffic," *ACM SIGCOMM Conf.*, 2000, pp. 87–95.
[24] Q. Luo and J. F. Naughton, "Form-Based Proxy Caching For DatabaseBacked Web Sites," *VLDB*, 2001, pp. 191–200.
[25] The Edge-Side Includes Initiative, http://www.esi.org
[26] B. Krishnamurthy, C. E. Wills, and Y. Zhang, "On the Use and Performance of Content Distribution Networks," *Internet Measurement Wksp.*, 2001, pp. 169–82.
[27] L. Ramaswamy *et al.*, "Automatic Fragment Detection in Dynamic Web Pages and Its Impact on Caching," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, 2005, pp. 859–74.
[28] F. Douglis, A. Haro, and M. Rabinovich, "HPP: HTML Macro-Preprocessing To Support Dynamic Document Caching," *USENIX Symp. Internet Technologies and Systems*, 1997.
[29] M. Rabinovich *et al.*, "Moving Edge-Side Includes to the Real Edge — The Clients," *USENIX Symp. Internet Technologies and Systems*, 2003, pp. 87–95.
[30] A. Datta *et al.*, "Proxy-based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation," *SIGMOD Conf.*, 2002, pp. 97–108.
[31] J. J. Hunt, K.-P. Vo, and W. F. Tichy, "Delta Algorithms an Empirical Analysis," *ACM Trans. Software Eng. and Methodology*, vol. 7, no. 2, 1998, pp. 192–214.
[32] K. Psounis, "Class-Based Delta-Encoding: A Scalable Scheme for Caching Dynamic Web Content," *IEEE ICDCS Wksps.*, 2002, pp. 799–805.
[33] P. Cao, J. Zhang, and K. Beach, "Active Cache: Caching Dynamic Contents on the Web," *Distributed Systems Eng.*, vol. 6, no. 1, 1999, pp. 43–50.
[34] Q. Luo *et al.*, "Active Query Caching for Database Web Servers," *WebDB*, 2000, pp. 92–104.
[35] S. Abiteboul *et al.*, "Active XML: Peer-to-Peer Data and Web Services Integration," *VLDB*, 2002, pp. 1087–90.

BIOGRAPHIES

STAVROS PAPASTAVROU (stavrosp@ucy.ac.cy) is a full-time teacher at St Barnabas School for the Blind in Nicosia and a Ph.D. Student at the University of Cyprus. His current research focuses on accelerating Dynamic Web Content practices, Proxy Caching, Mobile Agents and Java Middleware Technology. In 1999, his Bachelors Thesis received the Best Paper Award in IEEE's International Conference in Data Engineering. He has also published a number of papers in journals and peer-reviewed conferences. He has worked as a senior Web developer for the United States Chamber of Commerce. He is currently developing new techniques for making computer technology and programming interfaces accessible to visually impaired people.

GEORGE SAMARAS (cssamara@ucy.ac.cy) undertook undergraduate studies at the University of Athens (B.Sc. in Mathematics, 1982)and graduate studies at the Rensselaer Polytechnic Institute, USA (M.Sc., 1982, and Ph.D., 1989, in Computer Science). He has worked in the applied research program of IBM,s Communications and Networks Centre at Research Triangle Park, North Carolina (1990–1993), and has taught at the University of North Carolina at Chapel Hill (Visiting Assistant Professor, 1990-1993). He was also a member of IBM's International Standards Committees for issues related to distributed transaction processing (OSI/TP, XOPEN,OMG). His research interests are concerned with relational and object orient ed databases, distributed transaction processing, commit protocols and mobile computing.

Paraskevas Evripidou (skevos@ucy.ac.cy) undertook undergraduate studies in Electrical Engineering at the Higher Technical Institute of Cyprus (H.N.D. in 1981) and University of Southern California (B.Sc. in 1985). He undertook graduate studies at the University of Southern California (M.Sc. in 1986 and Ph.D in 1990 in Computer Engineering). He has taught at the University of Southern California, USA (part-time faculty, 1989-1990) and at the Southern Methodist University, USA (Assistant Professor, 1990-1994). His research interests are: Parallel Processing and Computer Architecture, Data-Flow Systems, and Functional Programming and Parallelizing Compilers.

PANOS K. CHRYSANTHIS (panos@cs.pitt.edu) is a Professor of Computer Science at the University of Pittsburgh and an Adjunct Professor at Carnegie Mellon University. He received his B.S. from the University of Athens, Greece and his M.S. and Ph.D. from the University of Massachusetts at Amherst. His current research focus is on mobile and pervasive data management including sensor networks and data streams. In 1995, he was a recipient of the National Science Foundation CAREER Award for his investigation on the management of data for mobile and wireless computing. His research accomplishments have been published in over 100 papers in journals and peer-reviewed conferences and workshops in the field of data management. In addition, his publications include a book and book chapters on transaction processing and data access in distributed, mobile and web databases. He is on the editorial board of the VLDB Journal, and was program chair of several workshops and conferences related to data management and databases. More recently, he was the IEEE ICDE 2004 PC Chair for the area of distributed, parallel and mobile databases, the PC Co-chair for ACM MobiDE 2006 and the General Chair of MobiDE 2003 and MDM 2005. He is a member of ACM, IEEE and USENIX.