A FRAMEWORK FOR PERVASIVE WEB CONTENT DELIVERY

HENRY NOVIANUS PALIT

(S. Kom., ITS, Surabaya; M. Kom., UI, Jakarta)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2006

Acknowledgements

Praise and glory be to the Lord, who has given me strength to pursue my purposes in life, courage to confront any challenge, perseverance to carry on in the midst of turbulence, and wisdom to keep me humble. He is my shepherd and my comfort; I shall not be in want.

I would like to take this opportunity to express my utmost gratitude to Prof. Chi Chi Hung for his inspiration, encouragement, and invaluable advice throughout the course of my research. Not only is he the best supervisor to guide me in this research field, but outside of the research work he is also a great mentor, from whom I learn a lot about important things in life. For the time he made available for research discussions, the effort he spent on reading and revising my research papers and thesis, the help he offered when I was in trouble, and the patience he showed against my late progress, I am always thankful.

I would also like to thank my replacement supervisor, Asst. Prof. Ken Sung Wing Kin, for allowing me to stay in his laboratory and for assisting me with all the administrative matters. I am sincerely grateful for his tremendous effort to help my research keep on as smoothly as possible.

This study would not have been possible without the Research Scholarship I received from the National University of Singapore. Therefore, I thank the University – and the School of Computing, in particular – for giving me the opportunity to pursue my postgraduate study.

Through my many years in Multimedia Information Laboratory, I had worked with not just colleagues but caring and hospitable friends. I have benefited from many research discussions with Hongguang, William, Junli, Li Xiang, Su Mu, and Choon Keng as much as I have enjoyed their warm and sincere friendship. My interaction with other friends like Meng, Rain Zhang, Akanksha, Wenjie, and Xiao Yan has made my stay there pleasant and lively.

Moreover, I am indebted to my brothers and sisters in Christ for their support, prayers, and companionship. In particular, I would like to thank – among others – Eni and Martin, Henny and Karim, Evelyne and Didi, Aini, and my Cell Group's and Prayer Group's friends. I thank Heng Thong, my flatmate, for the same support, prayers, and companionship he has given me. An abundance of appreciation and gratitude also goes to Dina, who kept "jia you"-ing me all the way till the completion of this thesis. I really hope one day I can repay you the same favor.

Above all, I would like to express my highest appreciation to my parents, sister, and brother for their endless love, compassion, encouragement, and persistent prayers. Forever, I owe them an immense debt of gratitude. To them, I dedicate this thesis.

Table of Contents

Summary

The integration of the Internet and the wireless network is inevitable. Consequently, Web clients become more heterogeneous, and therefore, pervasive services are required. This is one major challenge that the Web content providers face nowadays. Other challenges are, among others, increased multimedia data traffic, personalized content, and demand for efficient Web content delivery. Learning from the past researches, this thesis tries to address the challenges as a whole. In doing so, two objectives have been set out.

The first objective is to devise a fine-grained, scalable Web data model. The data model is the key factor to attain efficiency, in addition to adaptability, in Web content delivery. According to the data model, an object is heterogeneous as a whole but can be divided into homogeneous "atoms". The object can be represented by composing some of its atoms; the greater the number of atoms, the better is the object's presentation. Thus, a variety of representations – along different types of scalability, perhaps – can be generated from the object with less, or even, no complex computations.

Modulation, a novel adaptation, was proposed to exploit the data model. Modulation is characterized as fast, exclusive, and reversible. Alas, modulation can only be applied to scalable data formats such as progressive and hierarchical JPEG, MPEG-4, JPEG 2000, and H-264. Nevertheless, the multimedia trends head toward scalable data formats. To demonstrate its efficiency, modulation has been implemented in the JPEG

2000 image standard. Comparison with transcoding – the oft-cited content adaptation – in the JPEG standard confirms that modulation outperforms transcoding in processing time.

To replicate the benefits of modulation to the Web content delivery, the existing framework needs modifications. Therefore, the second objective of this thesis is to design a conceptual framework for pervasive Web content delivery. In stark contrast to the existing one, the proposed framework requires re-definition of the roles of server, proxy, and client in Web content delivery. The framework emphasizes collaboration between the origin server and the proxy, and gets benefits of both server- and proxy-based adapting approaches. Modulation is the center piece of the framework's operations. Moreover, a variety of supporting meta-data are essential for providing the best-fit presentation for each and every client. The overall goal is "on-demand" Web content delivery.

As a proof of concept, a model prototype has been developed based on the proposed framework. Two types of meta-data are involved; one is the client meta-data (CC/PP was used) and the other is the server meta-data (ADP was devised). It was found in the development that the current server application (Apache was employed) just required minor additions and some adjustments, but the proxy application (Squid was employed) had to go through quite a considerable makeover. By contrast, the client's browser only needs to add an extension header to its requests. Evaluation on the model prototype has shown that it greatly benefits from modulation and exhibits high data reuse. Some tangible benefits are improved client perceived latency, conserved Internet bandwidth, and reduced server's load.

List of Tables

List of Figures

Publications

H. Palit and C. H. Chi. Modulation for Scalable Multimedia Content Delivery. In *Proc. of the 6th International Conference on Web-Age Information Management (WAIM 2005)*, Hangzhou (China), October 2005.

C. H. Chi, H. Palit, and L. Liu. Proxy-Based Pervasive Multimedia Content Delivery. In *Proc. of 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), Chicago (IL)*, September 2006.

H. N. Palit, C. H. Chi, and L. Liu. Framework for Pervasive Web Content Delivery. In *Proc. of 7th Pacific-Rim Conference on Multimedia (PCM 2006)*, Hangzhou (China), November 2006.

Chapter 1

Introduction

This chapter starts with an overview of Web content delivery. Following the overview, the challenges in Web content delivery and the efforts to address them are explained. After that, motivation of this thesis is expressed, its objectives set out, and its contributions listed. Organization of the thesis is outlined at the end of this chapter.

1.1 Overview of Web Content Delivery

The Internet has evolved tremendously from a limited, research-oriented, hundreds-of-host network to a worldwide, multi-purpose, millions-of-host network. In fact, it is still growing at a fast pace, particularly in many developing countries. It has also been penetrating many aspects of modern civilization and becoming part and parcel of our daily activities. Owing to its instantaneity, the electronic mail (e-mail) has considerably replaced the snail mail as the medium of communication and document transfer. Chatting with distant friends and colleagues can be done economically by means of an instant messenger.

Nevertheless, the vast majority of users draw on the Internet to surf the World Wide Web (or simply the Web). Such user activities in the Web include reading news, searching for a particular subject or a product, tracking stock market performance, Internet banking, online shopping, and so forth. In the near future, more activities will be performed online through the Web. Thus, it is hardly surprising that the Web takes the lion's share of the Internet traffic. A study by Thompson et al. [ThMW97] concluded that the Web seized up to 75% of the overall bytes and 70% of the overall packets on the Internet traffic. A more thorough study by McCreary and Claffy [McC00] also affirmed the Web's dominance over other Internet applications. The Web's dominance is a fundamental reason why research on the Web is still exciting.

For the past few years we have witnessed the proliferation of mobile/wireless devices, such as cellular phones and PDAs (Personal Digital Assistants). Mobility has been the trend around the globe. Everyday we can see around us people of different ages

clutching mobile devices. Modern people crave for mobility to work and communicate with others anywhere and anytime without much restriction. Some people even feel helpless without any mobile device. Considering people's dependency on Internet applications and mobile devices, the integration of the Internet and wireless networks seems inevitable. Nowadays, many mobile devices are enabled to surf the Web. A market research report by Computer Industry Almanac[1] predicted that, by year-end 2005, 48% of Internet users would surf through wireless devices. Hence, the Web clients become more heterogeneous.

Meanwhile, the technologies behind the Internet applications keep on improving as communications and computer technologies are enhanced. The advancement of digital-imaging and digital-sound gadgets (e.g., digital camera, video camera, scanner, MP3 player, etc.), in addition to the proliferation of high speed broadband Internet connections, has bolstered multimedia data transfer over the Internet. Furthermore, Web-content providers also upgrade their sites regularly with enhanced multimedia content to attract more visitors. The content may employ a new multimedia technology with improved data compression, but it is often enlarged in spatial resolution (width and height) and may be more animated. Consequently, the overall multimedia content's data-size is increasingly large. The above factors, and many others, are the causes of an increase in multimedia data traffic observed in the Web.

The Web clients' heterogeneity and the increased multimedia data traffic are some technological factors that shape the trend of Web content delivery. There are also

---

[1] http://www.c-i-a.com/pr032102.htm

psychological factors, associated with the users' and providers' expectations, which influence the trend. Typical Web users' expectations are fast access and personalized (customizable) content, whereas Web-content providers want rich, attractive multimedia content and easy, efficient deployment of Web services. Discrepancies among the technological and psychological factors instigate challenges to the Web content delivery. Some of the challenges are mentioned in the following section.

1.2 Challenges in Web Content Delivery

The Web-content providers' desire for rich multimedia content is in line with the advancement of multimedia technologies, but may not be compatible with the users' expectation of fast Web access. As multimedia Web content grows larger and consequently multimedia data traffic increases, some Web users with a low-bandwidth Internet connection suffer slow access. Something that Web-content providers want is to be able to send the "context" of the multimedia content sooner than the content itself, regardless of the Internet traffic's condition. The context, which is much smaller than the content, could be in the form of a thumbnail or low-quality representation of the multimedia content. Digesting the context, the user may comprehend the multimedia content even before completion of the content's transfer. In this way, all users – high- and low-bandwidth-connected – can be served quite satisfactorily. This is a tough challenge facing the Web-content providers.

The problem above is further complicated by the Web clients' heterogeneity and different user preferences (the issue of personalization). There are varieties of Web-

enabled – wired and wireless – devices, such as desktop, laptop, PC tablet, PDA, and cellular phone. Besides different communication media, those client devices vary greatly in hardware (i.e., screen's resolution, color depth, processing speed, memory size, etc.) and software (i.e., operating system, browser, video/audio support, rendering applications, etc.). Presenting multimedia content to different client devices is particularly difficult since – considering each device's limitations – not all multimedia objects can be universally displayed. For instance, an image of 800×600 (width $\times$ height) pixels may be displayed properly in a desktop's monitor, but may not be in a cellular phone with a small screen, let us say, of 160×240 pixels. In addition, different clients (users) tend to have different preferences with respect to information of interest, latency time tolerance, multimedia content inclusion, and so forth. Thus, Web-content providers should no longer adopt the "single presentation for all clients" paradigm. They need to cater for different presentations if they do not intend to alienate particular clients.

The traditional method of addressing the above problems is by providing multiple prearranged versions (representations) of a Web resource. The versions are created offline before the service time (i.e., before starting to serve any client request). Each version is to serve a specific class of client devices or a certain user preference. Although this method is simple, it has several drawbacks. Firstly, it requires more disk space to store lots of resources' versions. Secondly, to reduce the disk space usage, often the number of versions is restricted. The extensibility of the Web site is also limited since the disk space may be taken up rapidly. In other words, this method is quite rigid. Lastly and more importantly, it is troublesome to maintain the resource's versions since any modification on a particular version must be disseminated to the other versions. Apparently this

method clashes with the Web-content providers' expectation of easy, efficient deployment of Web services.

Note that the challenges mentioned in the previous paragraphs are interlinked. Therefore, it is better to address the challenges as a whole instead of trying to solve them one by one. Past research efforts, coming from different research areas, have been devoted to address the challenges. Alas, each effort tried to solve one challenge at a time, independent of the other challenges. These isolated efforts may not solve the problems thoroughly. The following section highlights some of the efforts.

1.3 Efforts to Address the Challenges

The past efforts to address the Web content delivery's challenges are discussed within three research areas, namely content caching and replication, intelligent network, and multimedia standard. Each of them has made considerable contributions to the current Web content delivery.

1.3.1 Content Caching and Replication

Since the introduction of the Web proxy [LuA94], Web caching had been considered as its key feature. Indeed, the Web protocol (the latest is HTTP/1.1 [FiGM99]) has defined some headers to support Web caching, such as `Expires` and `Cache-Control`. By caching the passing Web content locally and using it to serve neighboring clients, the proxy can help reduce the client latency. Hence, the use of a Web caching proxy can meet the Web users' expectation of fast Web access.

Another but similar way of speeding up the Web content delivery is by employing a CDN (Content Distribution Network [Ver02], also known as Content Delivery Network [RaS02]). Unlike the caching proxy, which stores any passing Web content so long as it is cacheable, a CDN replicates Web content selectively – only one belonging to a paying CDN customer – and the content may be uncacheable. Furthermore, the Web content in a CDN server is fully controlled by the content provider (i.e., the CDN customer). A CDN is often employed to deliver dynamic and streaming content.

Web-content providers opt for dynamic content due to reasons like avoiding stale delivery and personalizing content for a given user. Since dynamic content is often made uncacheable, a Web caching proxy is ineffective to deal with it. By contrast, a CDN server in collaboration with the original server can deliver dynamic content effectively. In the past research efforts, some techniques (e.g., HPP [DoHR97], ESI [ESI01], and CSI [RaXD03]) have been proposed to handle dynamic content delivery. They basically divide a dynamically generated Web page into static and dynamic fragments. Forming a *template*, the static fragments are infrequently changed and therefore cacheable. When the Web page is assembled (usually at a CDN server), the dynamic fragments are requested from the origin server to fill the specified positions in the template.

Streaming content is sometimes just played partially, and as a result, may not be cached properly by a proxy. In turn, the following client requests for the same streaming content could not be served by the caching proxy. On the other hand, streaming content can be prepopulated in a CDN server before being served to the clients. In that case, the CDN server can deliver streaming content better than what the caching proxy can offer.

Although a caching proxy may prefetch streaming content, the accuracy of the prefetching is limited.

Another technique to reduce the client latency is delta-encoding [MoKD02]. A delta is the difference between old and new versions of a Web resource. When an old version is expired, instead of sending the entire new version, the delta between the old and new versions is generated and sent out. The new version can be constructed from the old version plus the delta. The delta's size is usually much smaller than the version's size, so less network traffic is required and lower client latency expected.

Studies on content caching and replication mainly focus on client latency reduction (or, fast Web access). It is understandable since the studies are mostly perceived from the Internet service providers' point of view. In general, the latency reduction can be attained by use of a caching/replication system (either a proxy or a CDN server) and fragmentation of Web content. In the former technique, the caching/ replication systems are distributed around the globe to accelerate the content distribution to the clients. In the latter technique, the fragments are grouped according to their cacheability; only stale (modified) fragments are then fetched from the original server. The studies do not pay much attention to the client devices' heterogeneity; i.e., all clients are treated equally. Although some techniques resulting from the studies can support efficient delivery of personalized and multimedia content, the construction of such content is not their main concern. It is, nevertheless, addressed more by the studies on intelligent network and multimedia standard discussed below.

1.3.2 Intelligent Network

The term "intelligent network" is associated with the network's ability to process passing data. The network here is not in its true sense, but it refers to the network's nodes which have the processing capability. In the Web, the involvement of Web proxies is again required to process content. The content processing functions include filtering, translation, adaptation, and so forth. Obviously the functions are more advanced than just caching and constructing Web content as done in the previous research studies.

ICAP (Internet Content Adaptation Protocol) [ElC03] is a lightweight protocol for executing transformation and adaptation on HTTP messages. Some value-added services supported by ICAP are virus scanning, content blocking/filtering, advertising insertion, human language translation, and markup language translation. An ICAP client may intercept and redirect a client response (or request) to an ICAP server for modification, and then send the modified response (or request) to the corresponding client (or the origin server). By off-loading these value-added services to dedicated ICAP servers, the origin server's load can be reduced. An ICAP client is often, but not always, a surrogate (i.e., a reverse proxy) acting on behalf of a user. So far, ICAP has defined the transaction semantics but it is yet to define the accompanying application framework.

In the multimedia domain, the process of converting a data object from one representation to another is called *transcoding* [HaBL98] (also known as *distillation* [FoGB96]). Transcoding is lossy (inessential or unrenderable information is removed [Mog01]), data-type specific, and irreversible (the original object cannot be recovered from the resulting representation). There are two main objectives of transcoding a

multimedia object: 1) to make the object presentable to the client, and 2) to reduce the client latency delay. Transcoding is required when a multimedia object with its original characteristics (i.e., data format, resolution, color depth, etc.) cannot be presented in the given client device. Or perhaps the multimedia object is presentable but too large and, consequently, takes too long to display; hence, transcoding is employed to reduce its data-size. Such examples of transcoding are transformations within a media data-format (e.g., quality reduction in a JPEG image) and transformations between media data-formats – either same-domain (e.g., GIF to JPEG image conversion) or cross-domain (e.g., video to images conversion).

Studies on intelligent network focus on personalized and adapted content. They try to address clients' heterogeneity in capabilities and preferences. While ICAP works on mainly textual content (especially Web pages or containers), transcoding typically works on multimedia content (especially embedded objects). Adapting multimedia content is particularly challenging considering Web-content providers' eagerness to exploit it and due to the complexity it involves. Although transcoding can reduce the client latency delay, it usually involves complex computations which may introduce another latency delay and undermine the expected reduction's benefit. Therefore, transcoding should be employed only if the expected reduction of latency delay can offset the introduced latency delay. That is why transcoding is data-type specific; understanding of the associated multimedia data-formats is needed. Unless those issues are taken into consideration, transcoding may end up with inefficiencies. Related to this, the following research topic discusses the latest development in the multimedia standards.

1.3.3 Multimedia Standard

Due to its commonly large data-size, researchers have tried to find ways to stream multimedia content. In the streaming mode, few initial segments of a multimedia object are fetched, and then displayed immediately while the following segments are fetched. Since the multimedia object is displayed before the client receives it in its entirety, the client's perceived latency delay may be cut down. In a movie clip, those initial segments normally correspond with the first few seconds of the clip; this is quite the expected result. In an image, however, they may just give the first few lines of the image. The client may still need to wait for another few segments before the image's context can be digested. To improve this, progressive data-formats have been devised. Instead of displaying the image one line after another, the progressive data-format may present it in different increasing details, such as blurred-to-clear or coarse-to-fine. This way, the client may grasp the image's context sooner. Instances of progressive data-formats are interlaced GIF, interlaced PNG, and progressive and hierarchical JPEG.

In recent years, new multimedia standards – like MPEG-4, H.264, and JPEG 2000 – have come up with better features. Compared to their predecessors, they are more advanced in data compression, error robustness, and more importantly, progressive data transmission. Accordingly, they can handle the clients' heterogeneity better. In the JPEG 2000 standard, for instance, an image can be easily streamed to clients with different screen resolutions. By exploiting the JPEG 2000's advanced progressive data-format, the image's resolution can be scaled down, if necessary, without much effort. This is because the image may be composed of some image-planes with increasing resolutions, so the

client device just needs to select the appropriate image's resolution and discard the unnecessary image data. The notion of "scalable presentation" is aptly attached to these new multimedia standards.

Studies on multimedia standards have contributed some solutions to the Web content delivery's problems. Firstly, owing to better streaming techniques, Web clients may perceive fast access. Secondly, the scalable presentation bestowed upon the new multimedia standards may meet the clients' heterogeneity fairly well. Last but not least, strong support from the multimedia standards helps Web-content providers to deliver rich multimedia content without so much taxing on the Internet bandwidth. However, there is still room for improvement to efficiently deliver the multimedia content. Since multimedia objects are typically large in data-size, it would be better if they can be fetched once but used repeatedly to serve many clients. Placing a caching system between the server and the clients may help improve the multimedia content delivery. Moreover, the delivery of unnecessary multimedia data should be avoided. For example, if the client wants a low-resolution representation of a scalable image, only the corresponding image data should be transmitted. Alas, in reality that is not always the case. Perhaps because it does not know the client's preferences or maybe due to its inability to scale down the image, the server just sends the whole image to the client and lets the client discard the unnecessary image data. Such an inefficiency wastes time and the Internet bandwidth. Collaboration between the Web-content providers and the Internet service providers may be the best way to rectify the problems.

1.4 Motivation: What Will Be the Future Web Content Delivery?

As observed above, studies on the different research areas shared a similar interest in the development of Web content delivery. Although they did not deal with the holistic problems of Web content delivery and, consequently, fell short of offering a satisfactory answer to all challenges mentioned in Section 1.2, they had contributed methods or techniques to improve Web content delivery. It is our belief that the solution should begin with the blueprint of the projected Web content delivery. The non-existence of this blueprint is the motivation of our thesis. Considering all the affecting factors, the challenges, and the previously proposed techniques, we may develop the blueprint. Below are the supposed characteristics of the future Web content delivery.

1.4.1 Pervasive or Ubiquitous Service

As the Web client base expands to include mobile users with diverse computation and/or communication appliances, content providers have to extend their services to meet the users' demands. Therefore, the Web services should be accessible for heterogeneous clients. That is why they are dubbed "pervasive services"; the services can be accessed from anywhere, at anytime, by any user.

Besides the client devices' capabilities and limitations, the pervasive services should also take the client preferences into consideration. As mentioned earlier, some examples of the client preferences are information of interest, latency time tolerance, and multimedia content inclusion. The client capabilities, limitations, and preferences are collectively labeled the client characteristics.

1.4.2 Fine-Grained Entities with Heterogeneous Properties

Techniques proposed for dynamic content delivery center on page fragmentation which partitions a Web page or container into fragments with different cacheability. In a progressive multimedia standard, an object is decomposed into several layers of presentation with increasing quality or resolution. In general, the current Web resource is no longer the smallest entity in the Web. The resource should be divisible into smaller entities (fragments, segments, or others alike), each of which has a unique combination of properties. When the resource is fetched, validated, presented, or manipulated by other means, only particular entities of the resource may really be engaged. The manipulated entities are determined by their properties' values.

Instances in Section 1.3 show that fine-grained entities of a Web resource can serve clients' heterogeneity in an efficient manner. Various representations of the resource may be constructed from its entities. In addition, the resource's entities can be streamed one by one and displayed immediately on the client side, so that the perceived latency delay can be improved.

1.4.3 On-Demand Delivery with Efficient Data Reuse

Considering its large data-size and supported by its fine-grained entities, a Web resource should be delivered on-demand. Here, "on-demand delivery" suggests that only needed entities of the resource are transmitted to the requesting client. Thus, the Internet bandwidth is consumed sensibly.

The use of a caching system can further conserve the bandwidth usage. Not only can a cached resource (i.e., a representation) be used fully to serve the same client request (requesting the same representation), but it may also be used *partially* to serve another client request (requesting another representation). This is particularly sound since entities of a resource's representation may be used to construct other representations, perhaps with additional entities of the resource. Then, the overall use – and reuse – of data in the Web content delivery will be very efficient.

1.4.4 Rich Meta-data

Deploying pervasive services requires knowledge of the client characteristics. Obtaining a suitable Web resource's representation for a given client requires information about fine-grained entities of the resource and their properties. Likewise, the on-demand delivery can only be done if information about the resource is known. All of these reveal the requirement for additional data besides the Web resources. Data that describe other data are commonly called meta-data. Clearly, the future Web content delivery will demand more and more meta-data.

There are many ways to distribute meta-data. They can be embedded in the object they describe. Most multimedia objects have meta-data within, usually dubbed "the headers". Meta-data can be attached to the protocol carrying the object. The Web protocol (HTTP) defines request, response, as well as entity headers; many are used for describing the object in the HTTP body. Lastly, meta-data can be placed in a separate document. The emerging XML format is commonly used to construct such a meta-data document. Owing

to its ease and extensibility, the last method has been widely exploited. In the near future, with the proliferation of the Semantic Web, uses of meta-data will gain more popularity.

1.5 Objectives and Contributions

Intrigued by the benefits that the future Web content delivery – specified in Section 1.4 – may bring, this thesis tries to have the future Web content delivery materialized. In this section, we declare the objectives and contributions of this thesis.

1.5.1 Objectives

The blueprint of the future Web content delivery is accomplished in two stages. In each stage, an objective is set out. The objectives of this thesis are as follows:

1. Devise a fine-grained, scalable Web data model.

   The characteristics of the future Web content delivery indicate the importance of a data model. The data model should be able to decompose a Web resource into fine-grained entities with heterogeneous properties. Of the entities, a variety of representations can be generated. The data model should also exhibit scalability, so that on-demand and efficient delivery can be attained. Studies on multimedia standards have introduced a progressive data-format which can offer scalable presentation. This will be the starting point for our data model.

2. Design a conceptual framework for pervasive Web content delivery.

   The conceptual framework should exhibit all characteristics of the future Web content delivery. Since the main purpose of improving the current Web content delivery is to

serve users better – whilst the targeted users are heterogeneous, the framework should uphold pervasive Web content delivery. The data model proposed in point 1 above will be fundamental to the framework. The main components of the framework will be outlined and their functions elaborated.

For each of the two stages, an illustration will be given to demonstrate the efficacy of our proposals. Comparison with the current practices will be conducted as well to see the improvements we may get from the proposed data model and framework.

1.5.2 Contributions

We believe that our research efforts on this thesis will enrich the knowledge base of several research areas, particularly on the field of Web content caching and distribution. Our contributions are as follows:

1. Modulation – a scalable adaptation.

   We have stated above that devising a fine-grained, scalable Web data model is our first objective. The data model also includes some transforming operations. The operations materialize into a new adaptation, called modulation. Modulation has exceptional characteristics which benefit the Web content delivery.

2. JPEG 2000 modulators.

   To show the efficacy of modulation, we give an illustration using the JPEG 2000 standard. Based on the specified modulating operations, some JPEG 2000 modulators are developed. Later on, modulation in the JPEG 2000 standard will be compared

with transcoding in the JPEG standard. Results of the comparison should affirm the benefits of modulation.

3. A framework for pervasive Web content delivery.

   Our framework is quite distinct from previously proposed frameworks. The main distinction is its holistic approach in dealing with the challenges in Web content delivery. The framework is proxy-centric. In addition to caching and adapting passing content, the proxy matches the client's characteristics to the requested resource's characteristics so that the best representation can be served to the client.

4. A model prototype of the pervasive Web content delivery.

   A model prototype is developed to show the efficacy of our proposed framework. In the development, modifications to the system's components – particularly the server and proxy applications – are detailed. The model prototype is extensible; various adaptors (transcoders and modulators) can be plugged into it quite easily. The model prototype will be evaluated and analyzed to see the improvements that it may offer. Primarily the results should exhibit a marked reduction in the client latency delay and conservation of the Internet bandwidth.

1.6 Scope and Organization of the Thesis

This thesis will not put emphasis on the widespread implementation of the framework in the Web. The emphasis should be on the efficacy and efficiency of the framework. Hence, the widespread implementation is beyond the scope of this thesis. The implemented systems – developed here for the JPEG 2000 and JPEG standards – are just

to give an illustration of their workings and to show their benefits. However, the efficacy and efficiency achieved here should be upheld for other standards alike.

The rest of this thesis is organized as follows. Literature review is given in Chapter 2. The fine-grained, scalable data model is proposed in Chapter 3. Modulation, the novel adaptation, is specified at the end of Chapter 3 and then implemented in Chapter 4, using the JPEG 2000 image standard as an illustration. In Chapter 5, modulation in the JPEG 2000 standard is compared and contrasted with transcoding in the JPEG standard. Chapter 6 proposes a framework for pervasive Web content delivery, in which modulation will be fully utilized. As a proof of concept, a model prototype based on the proposed framework is developed, and the development is elaborated in Chapter 7. Evaluation of the model prototype is presented in Chapter 8, and it will reveal the attained benefits as well as the costs. The whole thesis is concluded in Chapter 9.

Chapter 2

Literature Review

Section 1.3 of the Introduction has mentioned the efforts to address the challenges in Web content delivery. The efforts are classified into three research areas: content caching and replication, intelligent network, and multimedia standard. In this chapter, researches on those three areas are discussed in depth. Learning from the prior researches, the last section of this chapter summarizes the factors that may affect the accomplishment of pervasive Web content delivery.

2.1 Content Caching and Replication

Caching for the Web clients is analogous to the cache memory for CPU. Both are employed to speed up data access. In the Web environment, the data are the Web contents, each of which has a URI as its identity. The temporal locality [Den05], exploited by the caches, says that a resource that is referenced at one point in time will be referenced again sometime in the near future. By caching the Web contents, future client requests may be served from the cache, as opposed to from the origin server. The expected benefits are improved response time and reduced Internet traffic. Thereby, Web caching can meet the Web users' expectation of fast Web access.

In recent years, Web caching has grown along with the nature of Web contents. In the past, the contents were all static resources like Web pages, documents, images, etc. As the trend goes towards dynamic contents, the efficacy of the traditional Web caching was challenged. New methods in Web caching thus came up to address the challenge. In the following subsections, the history and development of Web caching, the proliferation of Web replication, as well as some techniques to reduce latency and to handle dynamic contents are presented.

2.1.1 HTTP and Web Caching

Caching had been incorporated in the Web protocol just after the birth of the Web. The original Hypertext Transfer Protocol (HTTP) [Ber91], known as HTTP/0.9, was very primitive and did not have any header in its messages (both requests and responses). However, Tim Berners-Lee immediately upgraded the original HTTP to include some

headers [Ber92]. Two of the request headers specified in the latter document were `Pragma` (with `no-cache` as the only defined value) and `If-Modified-Since`, which suggest the possible existence of a caching system where a copy of the requested Web object may be stored. Further, the `Expires` response header was also specified to notify when a cached Web object has to be refreshed. Those three cache-related headers were still preserved in HTTP/1.0 [BeFF96] and further expanded in HTTP/1.1 [FiGM99] with other cache-related headers, such as `Cache-Control`, `Age`, `ETag`, and `Vary` (note: `ETag` and `Vary` were classified as cache-related headers by Krishnamurthy and Rexford [KrR01]).

Web caching can be performed at the client's side, at the server's side, or at a proxy (intermediary) server. Most client browsers are equipped with a cache. This is very logical since the user may visit the same page again (temporal locality). Caching at the server's side may be useful if the contents are periodically changed and generated. Caching the generated contents, the server need not execute the generation process repeatedly and can manage its resources more efficiently. Nevertheless, Web caching is more beneficial if it is applied to a proxy server.

Luotonen and Altis [LuA94] suggested few benefits of caching Web contents at the proxy server. Firstly, the proxy can save disk space since only a single copy – as opposed to one copy per client – is cached. Secondly, serving multiple clients, the proxy can cache more efficiently Web objects that are often referenced. Thirdly, the proxy may prefetch more effectively Web objects that soon will be referenced because it has a larger sample size to base its statistics (or, other predictive algorithms) on. Lastly, to a certain extent, the caching proxy can offer availability, even if the external Internet connection is

cut off. Those benefits complement the other two benefits we have mentioned earlier; those are improved response time and reduced Internet traffic.

One of the initial evaluations on the efficacy of a caching proxy was done by Abrams et al. [AbSA95]. Some findings from that study are worth noting here. The maximum hit rate achieved by a caching proxy is around 30–50%. The caching proxy's hit rate tends to decline with time, and this may be attributed to the client browsers' cache filling over time. Caching all kinds of objects is more beneficial than caching selective objects (i.e., certain object type, object size, or domain); selective caching may drop the hit rate by 10–50%.

A study by Feldmann et al. [FeCD99] suggested that a Web proxy should not only cache data but cache connection as well; caching connection implies that the proxy uses persistent connections. They found that, in the low-bandwidth environment, data caching reduces average user-perceived latency by only 8%, whereas combined data and connection caching produces up to 28% latency reduction. Likewise, in the high-bandwidth environment, data caching improves mean latency by 38% but the combination of data and connection caching improves it by 65%. They also suggested that cookies (commonly used for personalized contents) can reduce the efficacy of a caching proxy since most cookied objects are uncacheable.

Bent et al. [BeRV04] conducted a study on commercial Websites and found that most of them use cookies indiscriminately and do not take advantage of `Cache-Control` directives. The study shows that around 66% of responses are uncacheable. A Content Distribution Network (CDN) was suggested to improve their performance.

2.1.2 Content Distribution Network

One way to speed up the delivery of Web contents is by caching them, the other way is by replicating them [KrR01, RaS02, Ver02]. Unlike Web caching, Web replication is a server-side approach employed to scale up the Web. In Web replication, the contents are copied to mirror sites. Clients can directly access or be redirected to the closest mirror site. The term "closest" may refer to geographical distance, network distance, and latency metrics. To efficiently utilize the storage capacity, replication is usually applied selectively to the most popular contents. The mirror sites, called *surrogates* or *reverse proxies*, are operated by a Content Distribution Network (CDN), also known as Content Delivery Network. For the rest of discussion, the surrogates are referred to as CDN servers. Content providers who sign up with a CDN for content delivery are called CDN customers, whereas Web clients that download contents through a CDN are called CDN clients.

Compared to forward proxies (i.e., the normal caching proxies), CDNs can offer the following benefits:

1. While deployment of forward proxies has some limitations – namely, requiring the client to explicitly configure its browser in non-transparent deployment and violating the end-to-end principle in transparent deployment – CDN servers can be deployed without such predicaments.
2. The Web contents are fully controlled by the content providers, so their consistency can be maintained effectively. In addition, access statistics can be accurately

collected. Overall, those features eliminate any reason for *cache busting* (i.e., purposely preventing responses from being cached).

3. Access to uncacheable contents – such as dynamic, streaming, and secured ones – can be improved.
4. The CDN servers can be prepopulated with precise contents from their customers. That eliminates the need to determine the contents to be prefetched.

Although there are a few techniques to direct a client to a particular CDN server, two techniques commonly employed are DNS redirection and URL rewriting [KrWZ01]. In DNS redirection, the authoritative DNS name server is controlled by the CDN. When this DNS server receives a DNS request from the client's local DNS server, it resolves the request with the IP address of one closest CDN server, depending on the availability of resources and the network conditions. There are two types of DNS redirection: full- and partial-site content delivery. The former delivers the entire contents of the CDN customer's site, the latter delivers only the embedded objects (primarily images) of Web pages. In URL rewriting, the origin server rewrites URL links as part of dynamically generating pages to redirect clients to different CDN servers.

Canali et al. [CaCC04] studied the benefits of CDNs from the client's point of view. They particularly examined partial-site content delivery provided by Akamai[2] and Speedera[3], two commercial CDN companies. Some interesting findings of their study are as follows:

---

[2] http://www.akamai.com

[3] Speedera was acquired by Akamai in June 2005.

- CDNs can offer significant performance gains and reduction in response time variance over a centralized Web server.
- CDNs show heavy time-dependent behavior, in which response times are far higher during the busiest hours of the day. Also, CDN benefits are reduced under heavy network traffic.
- Sites with a greater fraction of CDN-served objects achieve higher speedup. However, a heavy usage of CDN-enabled delivery is not sufficient to achieve high speedup.
- CDNs give better performance when only a few edge servers are used.
- DNS resolution time is a significant portion of the total response time under normal traffic condition.

More standard concepts and protocols used in Web caching and replication are discussed by Cooper et al. [CoMT01]. Some of the protocols are Cache Digests, CARP, ICP, PAC, WPAD, and WCCP. Their uses can be found in Thomas' book [Tho01]. Recently, there is also an idea to create collaboration between forward proxies and CDNs, as well as between individual CDNs, to improve access to Web contents. The collaboration is termed Content Distribution Internetworking (CDI) [RaS02], or simply, Content Internetworking [DaCT03]. Basically, it is a larger-scaled CDN, which includes components like request distribution, content distribution, and accounting. Request distribution (or, request routing) is to find the appropriate forward proxy or CDN server for a given client's request. Content distribution deals with distributing the content to CDN servers and forward proxies which eventually serve it. Finally, accounting measures and records the distribution activities, especially when the information recorded is ultimately used as a basis for the subsequent transfer of money, goods, or obligations. The

general challenge of realizing CDI is to standardize the complex interactions between multiple CDNs and ISPs (Internet Service Providers) without restricting innovation.

2.1.3 Techniques for Reducing Latency

Besides caching and replication, researches have proposed varied techniques to help reduce the user-perceived latency. Some of the techniques are data compression, prefetching, and delta-encoding.

The inclusion of data compression in the Web can be traced back to the early protocol, i.e., HTTP/1.0 [KrR01]. The `Content-Encoding` entity header can be used in end-to-end data transfer to indicate whether transformation (including compression) has been applied to a response's body. In HTTP/1.1 [FiGM99], the `Transfer-Encoding` general header has been added for the same purpose as the `Content-Encoding` header, but in host-to-host basis. Nielsen et al. [NiGB97] suggested that data compression applied to an HTML document can increase the probability of finding more embedded objects sooner, so that enough requests (for the embedded objects) can be issued immediately in a pipelined persistent connection; using `zlib` (`deflate`) compression, the resulting savings are about 16% of the TCP packets and 12% of the transmission time. Mogul et al. [MoDF97] also observed 19.8% bytes saving and 14.2% time saving when `gzip` compression was employed. As predicted, data compression is not really effective on JPEG and GIF images, since they are already compressed when generated.

Prefetching means retrieval of the Web object in advance of the client's request. This may reduce the latency delay perceived by the client at the expense of additional

load on the network and the server [KrR01]. Prefetching is not useful unless later a client requests the object and the prefetched response is still fresh. On the one hand, prefetching during periods of inactivity can make more effective use of the limited bandwidth. On the other hand, prefetching when the client is downloading another Web page would result in higher latency for the current page. Kroeger et al. [KrLM97] categorized prefetching into two types: local and server-hint. In local prefetching, the agent doing prefetching (i.e., a client's browser or a proxy) uses local information (e.g., reference patterns) to determine which objects to prefetch. In server-hint prefetching, the server uses its content specific knowledge of the requested objects and the reference patterns from a far greater number of clients to determine which objects should be prefetched. The actual prefetching, however, must be done by the agent; the server provides hints that assist the agent in prefetching. Kroeger et al.'s simulations show that a combined caching and prefetching proxy can provide at best 60% latency reduction, compared to 26% latency reduction in a pure caching proxy. Nevertheless, the cost of prefetching could be high and finding such an accurate prefetching algorithm is a difficult task.

When a Website changes its page, it is common that only some parts of the page are changed while the majority of them are still the same. Then, one may start thinking that, given that the client has a cached copy of the old page, sending the difference – or "delta" – between the old and new pages may be more efficient than sending the entire new page. That is the basic idea of delta encoding [MoKD02]. Suppose a Web resource (i.e., URI) has more than one representation at any given instant, a particular representation at a given time is called an instance. Delta encoding requires a unique identity for each instance. The `Last-Modified` or `ETag` response header defined in

HTTP/1.1 may be used as the instance's identity. A new request header, `A-IM` (stands for Accept-Instance-Manipulation), and two new response headers, `IM` and `Delta-Base`, were proposed for delta encoding. Some applications that may be employed for delta encoding are `diff`, `vcdiff` (previously known as `vdelta`), and `gdiff`. While `vcdiff` is considered the best overall delta algorithm, `diff` is relatively fast but can only be used on textual objects. Mogul et al. [MoDF97] investigated some delta encoding, data compression, and both combined algorithms applied to proxy traces. They found that `vdelta` can save 83% of the delta-eligible response-body bytes (31% of all response-body bytes) and 39% of the transfer time for delta-eligible responses (12% of the total transfer time). They suggested that delta encoding should be used when possible, and compression should be used otherwise. The added overheads for encoding and decoding are quite reasonable, but the remaining issues are which and how long deltas should be retained in the server. Finding a suitable delta-encoding algorithm for images (GIF, JPEG, etc.), which take 64% of the responses, is another open issue.

2.1.4 Techniques for Handling Dynamic Contents

The inspiration behind delta encoding is also applied to handling dynamic contents. Dynamically generated contents are generally used for news, auctions, stock quotes, and many others. Most of them are set uncacheable to enforce data integrity. Yet, not the entire page is changed at once. The frame, tables, and outline of the page are often static. Only some fragments need to be frequently changed. Instead of enforcing the whole page to be uncacheable, the page should be decomposed into fragments. Based on

their change frequencies (or, TTL/time-to-live), the fragments can be classified into groups. Hence, the changes can be applied to these small groups, as opposed to the whole page. By caching static fragments and only retrieving dynamic fragments, the dynamically generated page can be displayed fast and more efficiently. Some techniques employing this principle among others are HPP, ESI, and CSI.

HTML Pre-Processing (HPP) [DoHR97] divides an HTML document into static and dynamic portions. The static portion, called the template, contains macro-instructions for inserting dynamic information. The dynamic portion, called the bindings, contains the values of macro-variables to fill those in the template. The template can be cached, whereas the bindings are obtained for every access. After retrieving the bindings, the template is expanded by the client prior to rendering the document. The HTML syntax is extended with new tags denoting the macro-instructions. A plug-in needs be installed in the client's browser to expand the template with the bindings. Compared to the original document, HPP bindings reduce the size by factors of 4–8 without compression, or 2–4 when comparing compressed bindings to the compressed original document. In fact, the dynamic data is comparable in size to an efficient delta encoding. The end-to-end latency and the server's load also decrease.

Edge Side Includes (ESI) [ESI01] defines a simple, XML-based markup language that developers can use to identify content fragments for dynamic assembly at the network edge (i.e., a surrogate or a CDN server). ESI breaks down a Web page into a template and some fragments of differing cacheability profiles. When a page is requested, the corresponding template and page fragments are delivered to the edge server to assemble the requested page. The template can be cached at the edge server for a long

time. The page fragments may also be cached, and therefore, only uncacheable or expired fragments are fetched from the origin Website when serving subsequent requests. Unlike the template expansion in HPP, page assembly at the edge server can be conditional, based on information given in HTTP request's headers or end-user cookies. Since its release, ESI has been widely adopted by many companies, particularly CDNs.

Rabinovich et al. [RaXD03] argued that although ESI can speed up delivery of highly dynamic contents, page assembly at the edge server does not improve the response time for dial-up clients. To complement ESI, Rabinovich et al. proposed Client-Side Includes (CSI), in which page assembly occurs at the client's side. In this way, the client just needs to retrieve changed page fragments rather than download the whole Web page repeatedly. CSI does not require the presence of an edge server, although CSI can still utilize edge servers for scalable delivery of page templates and fragments. For the page assembler, CSI uses a generic JavaScript program that will download the template and any page fragments and assemble the page. Experimental results show that CSI can reduce the end-to-end (from the origin server to the client) traffic.

Compared to static contents, dynamic contents require more computational power, storage space, and Internet traffic. Most systems delivering dynamic contents employ a three-tiered architecture consisting of the Web, application, and database servers. As dynamic contents flourish and become the norm in the Web, the scalability of delivering dynamic contents is challenged. A number of approaches have been proposed in recent years to address that challenge. Basically the proposed approaches not only cache data at edge servers (this replicates the Web server's functionality only) but also try to replicate the lower levels of the three-tiered architecture, i.e., application and database servers. The

approaches can be classified into four techniques: application code replication, database engine replication, content-aware data caching, and content-blind data caching.

EdgeComputing [DaPW04] and ACDN [RaXA03] are some platforms for deploying and executing Web applications at edge servers. The data itself is still centralized, and that can become performance bottleneck and cause additional latency. Caching or replicating the data to the edge servers may address the problems.

Replicating database engines at edge servers, such as in Ganymed [PlA04], can offer scalability and reliability to dynamic Web applications. Nevertheless, since each update needs to be propagated to all other replicas to maintain data consistency, potentially enormous network traffic may be introduced.

DBCache [AlLK02, BoAM04] and DBProxy [AmPT02, AmPT03] cache database records (or tables) partially. The edge database caches are loaded with tuples resulting from queries on the central database. The cached tuples may be used to serve locally the following queries. The edge databases must be aware of the central database's data schema and are only modified through insert or update queries. While this technique can avoid the network traffic overhead yielded by database engine replication, it requires strong understanding on the central database's data schema, which requires more computations and may limit its scalability.

In contrast to content-aware data caching, content-blind data caching [OlMG05, SiPS06] stores each query result independently and does not merge (by means of insert or update queries) different query results in edge databases. This technique incurs minimal computational load but does not perform well on Web applications with poor query locality.

Researches on content caching and replication have contributed many improvements to Web content delivery. Notable improvements are improved response time, reduced Internet traffic, decreased server's load, better scalability and reliability. While uses of caching proxies and surrogates (including edge servers) can share the origin server's load in serving clients around the globe, new proposed techniques further help deliver the contents efficiently and effectively. In summary, the techniques involve data compression, prefetching, delta encoding, page fragmentation, as well as application and database replication at edge servers.

It can be noticed that the proxy's existence is very important to attain fast and efficient Web content delivery. But a proxy is not only useful for Web caching, more benefits of utilizing a proxy can be found next.

2.2 Intelligent Network

As noted in the previous chapter, the term "intelligent network" refers to the network's ability to process passing data. In the past, network was deemed passive; its sole task was to transfer packets from one host to another without knowing the packets' contents. However, the Web's proliferation has changed the old paradigm. People have come to realize that the network is a vast resource waiting to be tapped. Instead of waiting for the packets to reach the destined host before they can be processed, why could they not be processed on the network? Processing on the network can offer benefits like protection, efficiency, scalability, inter-operability, and many others. One example is a firewall, which blocks packets that may be harmful to the computers behind it. A caching

proxy, which has been thoroughly elaborated in the previous section, is another example. All of these are generally termed "active network" [TeSS97]. The intelligent network that we will discuss shortly is a set of applications dealing with Web contents. In particular, the intelligent network is employed to adapt the contents so that they can be served to heterogeneous Web clients.

To begin with, the Web protocol's support for representations of a resource is highlighted. Next, previously proposed transcoding systems are presented and analyzed. Elaboration of ICAP and OPES – two well-known adaptation architectures – follows. In the last subsection, the development of Semantic Web is summarized.

2.2.1 Web Protocol's Support

One of the key features of HTTP/1.1, compared to HTTP/1.0, is its support for representations of a Web resource [KrMK99]. In the protocol's specification [FiGM99], this feature is discussed under "Content Negotiation" (Section 12 of the document). Acknowledging different users' preferences and user-agents' capabilities, the protocol provides some mechanisms for selecting the best representation (variant) for a given response when there are multiple representations available. HTTP/1.1 provides two orthogonal forms of content negotiation:

1. Server-driven negotiation.

   The decision to select the best representation is done by the server. The client may send its preferences to the server, using request headers such as `Accept`, `Accept-Charset`, `Accept-Encoding`, `Accept-Language`, and `User-Agent`. The server,

on the other hand, can use the `Vary` response header to express the parameters it uses to select the representation. This method can avoid additional round-trips but may not accurately give what the client wants, particularly due to the limited preferences that the client may be able to use.

2. Agent-driven negotiation.

   The decision to select the best representation is done by the client manually (by clicking the selected hypertext) or automatically (by the user-agent). In this method, the client requests a varying resource, and the server replies with a 300 (Multiple Choices) response that contains a list of available representations and a description of each representation's properties (such as its content-type, language, and character set). While allowing the client to select its best available representation, this method needs a second request to fetch the representation. In addition, the HTTP working group did not complete the specification of this method, so its usability is still uncertain.

Even though the content negotiation specified in HTTP/1.1 may help address the users' expectation of personalized content, the properties used to describe a representation are very limited. That is why both forms of content negotiation above are rarely employed. Besides, the properties are only suitable to describe textual documents (e.g., HTML documents); the multimedia objects, taking the lion's share of Web objects, require a completely different set of properties. Another problem is that the protocol lacks support for describing the client itself (e.g., device's characteristics), which is more sensible than describing the content it wants.

HTTP/1.1 provides the method of communicating users' preferences, but the generation of the object's representations is another issue that needs to be addressed. The

next subsection discusses various transcoding systems that can generate different representations of a Web object to serve heterogeneous client requests.

2.2.2 Transcoding Systems

Transcoding is the process of converting a data object from one representation to another [HaBL98]. Previous researchers termed it as distillation, which is highly lossy, real-time, datatype-specific compression preserving most of the semantic content of an object [FoB96]. Hence, the characteristics of transcoding are lossy (inessential or unrenderable information is removed [Mog01]), datatype-specific, and irreversible (the original object cannot be recovered from the resulting representation). There are two objectives of transcoding a Web object:

1. To make the object presentable to the client.

   Due to limited capabilities of the client device, the object cannot be displayed on it. In other case, the object does not fit well to the client device's screen. Transcoding may convert the object to a representation of a different data-type supported by the client device, or it may reduce the object's spatial resolution for a proper presentation.

2. To reduce the client's perceived latency.

   The object's data-size may be large, and by contrast, the client's Internet bandwidth is low. Thus, the client may have to wait for a long time before it can comprehend the large object. To mitigate this discrepancy, transcoding may be employed to reduce the object's data-size.

Transcoding is often applied to multimedia objects (or conversely, multimedia objects, especially images, are often the target of transcoding). Few reasons may be suggested. Firstly, multimedia objects take the lion's share of Web access (about 58% of requests and 70% of data bytes accessed in the Web come from multimedia objects [OrCA97]). The proliferation of digital-imaging devices – such as digital camera, video camera, scanner, etc. – simply means that demands for multimedia objects are increasing. Secondly, multimedia objects are commonly large in data-size, compared to textual documents. Access to these large multimedia objects is one of the culprits of the "World Wide Wait" problem. Lastly, transcoding multimedia objects are more challenging than squeezing textual documents. Even if it is not squeezed, a textual (HTML) document can be duly displayed on a small screen, most likely still well-aligned, albeit spanning over many screen's pages. On the other hand, it is very inconvenient to see a high-resolution image displayed on a low-resolution screen.

More features of transcoding are covered in the rest of this subsection. Some past transcoding systems are mentioned and used as illustrations of the associated features.

Classification

In general, transcoding processes can be classified into two groups: 1) transformations within a media data-format, and 2) transformations between media data-formats. Examples of the first group are quality reduction in a JPEG image, color remapping or dithering in a GIF image, and so on. The second group may further be divided into same-domain and cross-domain conversions. Instances of same-domain conversions are GIF-to-JPEG, WMA-to-MP3, and PostScript-to-HTML conversions,

whereas those of cross-domain conversions are video-to-images and speech-to-text conversions.

InfoPyramid [SmML98, MoSL99] has a unique way to classify the object's representations. The representations are placed along the dimensions of modality (video, image, text, and audio) and fidelity (degree of summarization, compression, or reduction). This way, the transcoding processes to change both the modality and fidelity of an object from one representation to another can be well defined, and the cost-benefit analysis can be done.

Deployment

There are three aspects of deploying a transcoding system. The first aspect is the instantaneity of a transcoding service, whether it is executed online (on the fly) or offline (*a priori*). Online transcoding is more flexible and efficient in storage space, but it is also complex, burdensome, and sometimes time-consuming. In the offline approach, the object's representations are generated during creation time. Serving a client request, the system employed offline transcoding just needs to provide the correct object's representation; it is simple, but rigid and difficult to maintain. The majority of past transcoding systems employ the online approach; e.g., Pythia [FoB96] (later evolving into GloMop [FoGB96] and TranSend [FoGC97, FoGC98]), Mowser [JoWM96, BhJA98], SDT [Mog01, KnLM03], and TransSquid [MaSR02]. To our knowledge, InfoPyramid [MoSL99] and Quality Aware Transcoding [ChEV00] are the only offline transcoding systems.

The second aspect of deploying a transcoding system is the placement of a transcoding service, whether it is executed at the server, proxy, or client. Execution at the client causes more losses than benefits, so it should only be considered as the last alternative. Proxy-based transcoding is more scalable, cost-saving, and helpful to reduce the server's load, especially if a caching system is also employed. Server-based transcoding, however, gives more control to content providers over how the contents should appear to different clients. A proxy-based transcoding system can only use the online approach. In contrast, a server-based transcoding system usually trades the storage space for the computing resources and employs the offline approach; but the online approach is still workable. All online transcoding systems above are proxy-based, whereas InfoPyramid and Quality Aware Transcoding are both offline, server-based transcoding systems. Although implemented at the proxy, SDT also gains benefits of the server-based approach. Mogul et al. (i.e., SDT's creators) [Mog01, KnLM03] argued that transcoding may undermine the content's semantics if the server's explicit guidance to the transcoding proxy is not involved. SDT preserves the end-to-end semantics while offering more effective content transformation.

The last aspect of deploying a transcoding system is the architecture of a transcoding service. Most of the proposed transcoding systems run in a single machine, although they may be extended to multiple machines. Fox et al. [FoGC98] proposed a cluster-based architecture to give scalability, availability, and cost effectiveness to its transcoding service. Canali et al. [CaCC03], in contrast, favored a distributed architecture to prevent network bottlenecks. Further, Canali et al. examined a few cooperative schemes among transcoding proxies. Some of their findings are: 1) flat (peer-to-peer)

topologies give better hit-rates than hierarchical topologies, 2) query-based discovery protocols (e.g., ICP) offer better performance than summary-based discovery protocols (e.g., Cache Digests), and 3) employing a load-aware algorithm among cooperative transcoding proxies can yield smaller response times than employing a load-blind algorithm, particularly if the client load is unevenly distributed among the edge servers. In recent work, Canali et al. [CaCL05, CaCL06b] also incorporated CDN architecture in their two-level topology for content adaptation services. The two-level topology comprises internal nodes (located in the network core) and edge nodes (located on the borders of the Internet). By distributing adaptation load between the two nodes (i.e., assigning adaptation services requiring sensitive information on the internal nodes and those not requiring sensitive information on the edge nodes), not only can it preserve a high level of user privacy, but it can also increase scalability. In addition, they found that collaboration between the content provider and adaptation service provider can improve the performance further.

Client Profile

To serve the client with the best-fit representation, a transcoding system needs to know the client profile consisting of the client device's capabilities and the client's preferences. As stated before, the current Web protocol lacks support for describing the client profile. The past transcoding systems have proposed some alternatives to obtain the client profile.

In Pythia and Mowser, each client has to store its profile in the transcoding proxy. Each client profile contains an IP address so that, for every request received by the proxy,

the corresponding profile can be determined. The drawback of this mechanism is that the client is confined to its IP address for obtaining the service.

In GloMop, the client sets the desired options – e.g., download time, resolution of the image, and color depth of the image – provided by the specially built image browser. Similarly, TranSend embeds a Java dashboard (i.e., a Java Applet) to the HTML browser, by which the client can specify its quality preference. In the proxy, the preferences are converted into parameters used to execute transcoding.

In TransSquid, the client specifies its profile using CC/PP (Composite Capability / Preference Profiles) and registers the profile to the proxy. Similar to the use of IP address above, here a unique key is used in every sent request to link the request to the respective CC/PP.

Media Feature Sets is employed in SDT to express the client device's characteristics. The device's characteristics are added to each request's header. The proxy matches the device's characteristics with the requested content's characteristics – also expressed using Media Feature Sets – and invokes a particular applet, if necessary, to transcode the content.

Canali et al. use *requester-specific capability information* (RCI), attached to the client request, to convey information describing the capabilities of the requesting client.

Caching

A caching system is employed by most proxy-based transcoding systems. The proxy fetches the original object from the origin server and stores it in the cache, in addition to transcoding it according to the client profile. Serving future client requests for

the same object, the proxy just fetches the cached object and transcodes it promptly. A server-based transcoding system commonly does not need a caching system but provides various representations of an object (note that the offline approach is employed). Nevertheless, InfoPyramid still uses a caching system to store client-specific versions of the object's container (e.g., the generated HTML documents); hence, the cache can improve response times.

Quite a tricky issue is whether the transcoded results should also be cached. SDT seems to support that idea since caching the transcoded results avoids the need to execute the costly transcoding operations repeatedly. Differentiating one transcoded result from another can be done by the use of an HTTP extension header registered with the `Vary` header. However, selecting the transcoded result that matches a given client profile requires an additional match-making process. TransSquid tries to simplify the match-matching process by limiting the number of transcoded results. It classifies the client devices into certain categories and divides the cache into several levels according to the number of categories. Thus, for every request it receives, TransSquid firstly determines the device's category, and then fetches the cached transcoded result in the respective cache's level. If the object's representation is not found in the cache, then it is generated and stored in the appropriate level of the cache. In spite of that, caching multiple representations can affect the cache replacement policy.

Canali et al. [CaCL06a] put an additional string to the resource's URL to differentiate one version to another. The added string is extracted from the HTTP `ETag` header of a client's request; in this case, the `ETag`'s content indicates the resource's version requested by the client. Thus, multiple versions of a resource can be present at the

same time in the proxy's cache. The downside is that the client must specify the requested version in the `ETag` header, thus the mechanism is not so transparent to the client.

Some caching policies had been studied in the past. Cardellini et al. [CaYH00] examined whether caching the retrieved (more detailed) version, the transcoded version, or both versions was more beneficial. Their study found that caching the transcoded version gives shorter response time and higher cache hit than the other two caching policies. Similar study was done by Shen et al. [ShLB04] on caching transcoded streaming video clips. They concluded that caching a single version of video content is desirable in a less heterogeneous environment (e.g., in a corporation), whereas caching multiple versions of video content is beneficial in a heterogeneous environment showing strong temporal locality in the access pattern. Chang and Chen [ChC03] proposed an efficient cache replacement algorithm for transcoding proxies based on a generalized profit function. The function considers the reference rate of each version, the delay of fetching the original object, the delay of transcoding, the size of each version, and the aggregate effect of caching multiple versions of the same object. PTC [SiTR04] employs a similar cache replacement algorithm to Chang and Chen's, but it also takes dynamic factors – such as the current proxy load and network traffic – into account.

Other Features

Most transcoding systems only deal with the responses carrying the documents or multimedia objects to be transformed. Mowser may modify the requests, as well, to be HTTP/1.1 compliant and to append an `Accept` header. Used in content negotiation, the `Accept` header field contains data-types (i.e., MIME types) that a client is able to

display; the list of data-types is already predefined in the client's profile. A few MIME types are further specified in Mowser to indicate special data-types.

Besides transcoding multimedia objects, some transcoding systems also modify the container (i.e., HTML document), particularly if it contains embedded objects requiring transcoding. Pythia, for an instance, modifies the URL of an embedded image, and therefore, will recognize the modified URL's request as belonging to an embedded image. Next to the modified image tag, Pythia also inserts a hyperlink that, if it is clicked, will request the original image. Mowser, on the other hand, replaces the URL of an embedded image with a local URL that refers to the transcoded image in the proxy. URICA (Usage-awaRe Interactive Content Adaptation) [MoCC06] can change the layout of images on a Web page according to the client's preferences.

Cardellini et al. [CaYH00] suggested the idea of multiple transcoding, in which a transcoded object may be further transcoded to yield a less detailed (lower in quality and/or resolution) object. TransSquid allows inter-cache transcoding, in which a high-fidelity representation stored in a high cache's level can be transcoded into a low-fidelity representation stored in a low cache's level. Later transcoding systems [ChC03, SiTR04, ShLB04, CaCC03, CaCL06a] also support the idea of multiple transcoding.

Ihde et al. [IhMM01] proposed multiple transcoding in the form of a chain. Rather than creating complex, unlimited number of transcoders, Ihde et al. championed the idea of simple, modular transcoders. The modular transcoders may compose a chain (pipeline) of transformations if a single transcoder cannot give the expected result. Suppose three modular transcoders – JPEG-to-GIF, GIF-to-PNG, and PNG-to-TIFF converters – are available and the original image to be transcoded is in JPEG. The image's representations

in GIF, PNG, and TIFF can be attained by executing one or a chain of the three converters; e.g., converting the JPEG image to a PNG image can be done by composing the JPEG-to-GIF and GIF-to-PNG converters. Although their idea may reduce the number of transcoders, its success in reality is doubtful. Note that transcoding is a lossy process. Multiple conversions may cause a large amount of loss on the transcoded result.

Odyssey [NoSN97] supports application-aware adaptation, which emphasizes a collaborative partnership between the operating system and applications, for mobile information access. On the one hand, the operating system monitors resource levels (network bandwidth, computing cycles, memory usage, battery power, etc.) and notifies applications of relevant changes; on the other hand, each individual application adapts to the changes accordingly when notified.

Wijnants et al. [WiMQ05] integrated transcoding-enabled proxies to a Networked Virtual Environment framework. Each proxy is capable of transcoding video streams (i.e., avatars) in real time. Without the proxy, each client had to send three versions (high, medium, and low quality) of the same video stream; now only a single video stream is required. Intelligence is embedded in the proxy, so that it can monitor the network condition and consequently transcode the video stream to a suitable version.

URICA [MoCC06] collects a client's preferences through an interactive learning process. When a client is unsatisfied with the current adaptation result, he/she can interactively make changes to the adaptation process until the content is suitably adapted. URICA then stores the client's preferences and uses them for future adaptation, not only for the particular client but also for others having the same context (e.g., device type, screen size, network bandwidth, and user location).

BlackBerry[4] is a commercial Internet service, provided by Research In Motion (RIM), for wireless devices. Services like Web content filtering and compression as well as Web content transcoding are done by the BlackBerry Internet Service, acted as an HTTP gateway and placed at the other end of the wireless network. Through the services, HTML content is parsed and stripped of extraneous tags and unrenderable content, images are scaled down and converted to PNG file format, the whole page is partitioned and compressed for delivery, and other format translation activities may be performed for the purpose of fast and efficient delivery.

Transcoding can help reduce the client's perceived latency and make unpresentable content presentable. However, a transcoding process is often complex and time-consuming; hence, it introduces another latency delay. Han et al. [HaBL98] developed an analytical framework to determine whether and how much to transcode an image. In brief, to reduce response time, transcoding should be employed only if the expected reduction of latency delay in delivery can offset the introduced latency delay of executing the transformation.

Interested readers are referred to Colajanni and Lancellotti's survey paper [CoL04] for a range of solutions, issues, as well as research directions in Web content adaptation services. In addition, Colajanni et al. [CoLY05] presented the trend of distributed architectures for content generation, adaptation, and delivery services. The four identified architectures are progressively improved from a centralized, cluster-based system to a geographically replicated, multi-cluster system.

---

[4] http://www.blackberry.com

In connection with transcoding, the next subsection elaborates two well-known adaptation architectures: ICAP and OPES. While transcoding focuses more on the adaptation processes, the architectures deal with a broad range of issues, including protocol, components, and procedure calls.

2.2.3 ICAP and OPES

In this subsection, ICAP and its prospective benefits are discussed first. Discussion on OPES, which is a more general architecture, follows shortly.

ICAP

ICAP (Internet Content Adaptation Protocol)[5] is a lightweight protocol for executing a "remote procedure call" on HTTP messages [ElC03]. It is designed to off-load specific Internet-based content to dedicated servers, thereby freeing up resources and standardizing the way in which features are implemented [ICAP01]. The dedicated ICAP servers may be focused on a specific function like virus scanning, markup language translation, advertising insertion, human language translation, content filtering, or data compression. An ICAP client is often, but not always, a surrogate acting on behalf of a user. To a certain extent, ICAP is very similar to HTTP, as we will see shortly.

There are two major components in ICAP architecture: 1) transaction semantics, and 2) control of policy. So far, ICAP just defines the transaction semantics, which specifies the communication between an ICAP client and an ICAP server, the URI of an

---

[5] http://www.i-cap.org

ICAP resource, and the format of ICAP messages. This wire-protocol is of limited use without the second part, an accompanying application framework in which it operates. The second issue is beyond the scope of the current ICAP protocol, but is planned in future work.

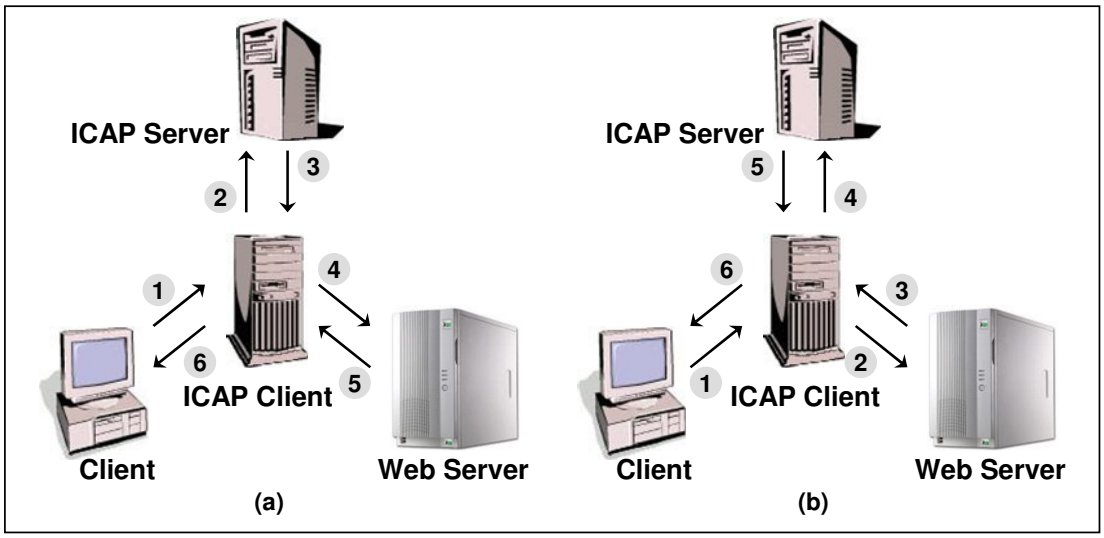


Figure 2.1 ICAP data flow for (a) request modification and (b) response modification

In a very general sense, ICAP can be used to modify both HTTP requests and responses. In the request modification [Figure 2.1(a)], the ICAP client passes a client request to the ICAP server, which in turn replies either a modified request or a response to the request back to the ICAP client. If a modified request is received, the ICAP client passes it on to the Web server and gets the response. The response – either from the ICAP server or the Web server – is returned to the requesting client. An example use of the request modification is content filtering. In the response modification [Figure 2.1(b)], the ICAP client passes the Web server's response to the ICAP server, which processes the response and returns the possibly modified response to the ICAP client. Then, the ICAP client sends the response – maybe different from the original – to the requesting client.

Instances of utilizing the response modification are markup language translation, human language translation, and virus checking. Readers interested in the details are referred to the ICAP specification [ElC03].

OPES

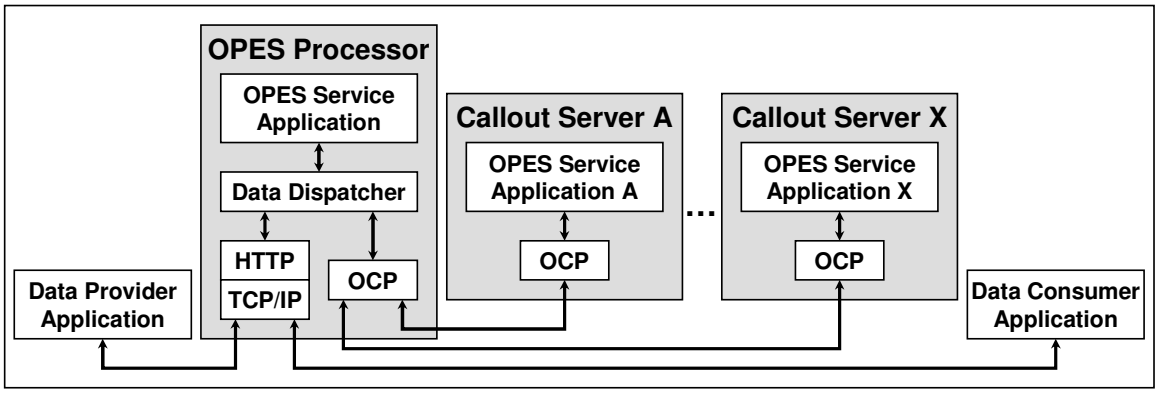


Figure 2.2 OPES architecture

OPES (Open Pluggable Edge Services) provides an architecture that enables the creation of an application service in which a data provider, a data consumer, and zero or more application entities cooperatively implement a data stream service [BaPC04]. OPES offers a bigger picture of content services than ICAP does. Its architecture, as depicted in Figure 2.2, comprises three interrelated concepts:

1. OPES entities

   An OPES entity is an application that operates on a data flow between a data provider application and a data consumer application. There are two forms of OPES entities: 1) an OPES service application, and 2) a data dispatcher. An OPES service application analyzes and possibly transforms messages exchanged between the data provider application and the data consumer application. A data dispatcher invokes an OPES service application based on an OPES ruleset and application-specific knowledge. In

the network, OPES entities reside inside OPES processors, which are explicitly addressable at the IP layer by the end user (data consumer application) and consented to by either the data consumer or data provider application. Every OPES processor must include a data dispatcher.

2. OPES flows

   An OPES flow is a cooperative undertaking between a data provider application, a data consumer application, zero or more OPES service applications, and one or more data dispatchers. At least one data dispatcher, which enforces policies, is required in the OPES flow.

3. OPES rules

   Policy regarding the OPES services and the data provided to them is determined by a ruleset consisting of OPES rules. The OPES ruleset, installed in a data dispatcher, indicates which service applications will operate on a data stream.

In some cases, the OPES processor may distribute the responsibility of service execution by communicating with one or more callout servers, each of which has an OPES service application. A data dispatcher invokes the service of a callout server by using the OPES callout protocol (OCP).

There are three types of OPES services [BaBC04]: 1) services performed on requests, 2) services performed on responses, and 3) services creating responses. An OPES service performed on HTTP requests may occur when a request arrives at an OPES processor or when it is about to leave the OPES processor. The service may or may not modify the requests on behalf of the data consumer or the data provider. Such services are content filtering, redirection, preferences addition, usage tracking, user profiling, etc. An

OPES service performed on HTTP responses may occur when a response arrives at an OPES processor or when it is about to leave the OPES processor. This service, too, may or may not modify the responses. Instances of the service are content adaptation, language translation, logging, usage billing, and so on. OPES services may create responses by dynamically assembling Web pages based on the context of the data consumer application; e.g., generating a local weather forecast Web page.

OPES services network can be deployed in two scenarios: surrogate overlays and delegate overlays. Surrogate overlays act on behalf of data provider applications (one or more origin servers), so the elements of surrogate overlays logically belong to the authoritative domain of the respective origin server. Delegate overlays act on behalf of one or more data consumer applications, and therefore, the elements of delegate overlays logically belong to the authoritative domain of the respective data consumer application. Within an enterprise environment, those two scenarios can be combined under the same administrative domain. More details about OPES and its specifications can be found in the official Website[6].

ICAP and OPES frameworks are quite complete and detailed. Nevertheless, they are on-going projects, and their adoption and implementation remain to be seen. An emerging technology for supporting intelligent network that we will discuss shortly is Semantic Web. It is dubbed the next-generation Web, in which meta-data play an important role. Although Semantic Web does not have a direct association with content

---

[6] http://www.ietf-opes.org

adaptation, it provides a mechanism to convey the client's context seamlessly to the adaptation services; the mechanism of which the current Web protocol is lacking of.

2.2.4 Semantic Web

The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation [BeHL01]. Web information varies along many axes. One of these is the difference between information produced primarily for human consumption and that produced mainly for machines. To date, the Web has developed most rapidly as a medium of documents for people rather than for data and information that can be processed automatically. The Semantic Web aims to make up for this. It provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C (World Wide Web Consortium) with participation from a large number of researchers and industrial partners.

For the Semantic Web to function, computers need structured collections of information and sets of inference rules to conduct automated reasoning; this technique is known as knowledge representation. Traditional knowledge-representation systems typically have been centralized, but central control is stifling and inflexible. Moreover, these systems usually limit the questions that can be asked so that the computer can answer reliably – or not answer at all. Semantic Web researchers, in contrast, want versatility at the price of unanswerable questions. The challenge of the Semantic Web,

therefore, is to provide a language that expresses both data and rules for reasoning about the data and that allows rules from any existing knowledge-representation system to be exported onto the Web. In other words, the task is to add logic to the Web so that rules can be used to make inferences, choose courses of action, and answer questions.

The first two technologies for developing the Semantic Web are the eXtensible Markup Language (XML) and the Resource Description Framework (RDF). XML lets everyone create or add arbitrary structure to their documents but says nothing about what the structures mean. The meaning is expressed by RDF, which encodes it in sets of triplets; the triplets themselves can be written using XML tags. In RDF, a document makes assertions that particular things (i.e., the subjects) have properties (the verbs) with certain values (the objects). Subjects and objects are each identified by a Universal Resource Identifier (URI), just as used in a link on a Web page. The verbs are also identified by URIs, enabling anyone to define a new concept – a new verb – just by defining a URI for it somewhere on the Web. Because RDF uses URIs to encode this information in a document, the URIs ensure that concepts are not just words in a document but are tied to a unique definition that everyone can find on the Web.

A problem may later come up. Two databases may use different terms for what is in fact the same concept, so a program that wants to compare or combine information across the two databases has to know that these two terms are being used to mean the same thing. A solution to this problem is provided by collections of information called ontologies; hence, the third basic component of the Semantic Web is the Web Ontology Language (OWL). An ontology is a document or file that formally defines the relations among terms. The most typical kind of ontology for the Web has a taxonomy and a set of

inference rules. The taxonomy defines classes of objects and relations among them. The inference rules empower the program to make deductive reasoning. OWL builds on RDF and adds more vocabulary for describing properties and classes, like relations between classes, cardinality, equality, richer typing of properties, characteristics of properties, and enumerated classes.

In short, each of the three standards in the Semantic Web has its own function; i.e., XML is for syntax, RDF semantics, and OWL domain specific vocabularies. A range of meta-data platforms have been developed using the Semantic Web's standards, particularly the RDF/XML syntax, and many are still to come in the near future. Some of the meta-data platforms are CC/PP (describing client profile), P3P (expressing privacy), PICS (associating meta-data with Internet content), RSS (summarizing a Website's channel), and so forth. They essentially have brought the existing Web closer to the projected Semantic Web, albeit more efforts and collaboration among developers are needed for greater proliferation of their uses.

Researches on intelligent network aim at exploiting the network resources for providing value-added services to the users. The services discussed in this section are mainly confined to delivering the best-fit representation of Web content to heterogeneous clients. Briefly, the current Web protocol is inadequate to describe the content's and the client's characteristics. That is why researchers have come up with new methods, either specific (as found in many transcoding systems) or general (e.g., ICAP, OPES, and Semantic Web), to address the protocol's inadequacies. Studies on past transcoding systems have demonstrated the viability of content transformation on the network (i.e., at

an intermediary or proxy). The ICAP and OPES frameworks further affirm the benefits of processing on the network. The intelligent network can offer personalized/customizable content while easing the origin servers' burden. While past research has contributed tremendously to the advancement of Web content adaptation, we observe that new multimedia standards championing scalable presentation open a new way of delivering content more efficiently. We also observe that an important element for content transformation is meta-data, which is used among others to describe the client profile, specify content's properties, and convey directives/instructions. Even the Semantic Web suggests more data and information that can be processed automatically. These observations plus others will be revealed further in Section 2.4.

As noted before, the multimedia objects have been dominant in the Web, yet many of them are large in data-size and not really presentable to varied client devices. Thereby, they are the common target of content transformation. In recent years, new multimedia standards have been devised and equipped with advanced features like superior bit-rate performance, improved data robustness, and progressive transmission. Learning about the multimedia standards may give us insight on how to deliver the multimedia objects better in the Web. The next section summarizes some features of the emerging multimedia standards.

2.3 Multimedia Standards

In the past, a multimedia object had to be downloaded completely before it can be viewed by a user. It may take a long while to download a large multimedia object. To

mitigate the long-winded display of multimedia presentation, people have come up with the streaming technique. While the multimedia object's data is transmitted bit by bit at the sender's side, it is immediately decoded and displayed at the receiver's side even though the data transmission is not complete. The decoding process takes place repeatedly every time a certain part of the multimedia object's data is received. As a result, the user views the multimedia presentation gradually. In case of an image file, the user can see the image presentation line after line.

Although the streaming technique can display the multimedia object gradually, sometimes it does not help reduce the user's perceived latency. For example, if the important image's region is at the bottom part, then the user has to wait for the image to be fully displayed. Then, researchers devised the progressive refinement technique, by which the image can be displayed at full size but initially blurred and becoming clearer with time. Some image standards employing the progressive refinement technique are interlaced GIF, interlaced PNG, and progressive and hierarchical JPEG. [Note: In the interlaced GIF and PNG standards, the image display may give a "venetian blind" effect due to the respective four-pass and Adam7 interlacing techniques they adopt.]

New multimedia standards, like JPEG 2000, MPEG-4, and H.264, have emerged in recent years. Compared to their old predecessors, they have more sophisticated features that are favorable to Web access. This section outlines two emerging multimedia standards: JPEG 2000 and MPEG-4.

2.3.1 JPEG 2000

JPEG2000 is the latest digital image standard developed by JPEG (Joint Photographic Experts Group)[7]. Superior low bit-rate performance, progressive transmission by pixel accuracy and resolution, and robustness to bit-errors are some of the JPEG2000's features [ChSE00] that are beneficial for pervasive Internet access. Due to progressive transmission, user may receive a JPEG2000 image with increasing pixel accuracy (known as *SNR scalability*), from a blurred image to a completely clear image. Alternatively, the image may be reconstructed with increasing resolution (*spatial scalability*), from a coarse, blocky image to a completely fine, smooth image.

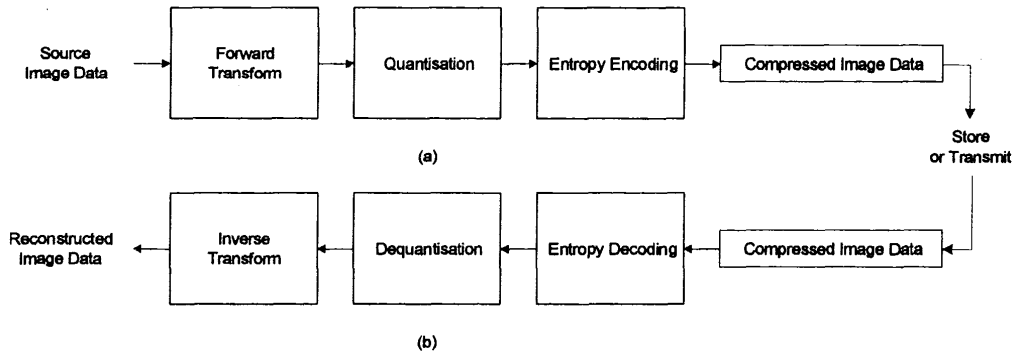


Figure 2.3 Block diagram of the JPEG 2000 (a) encoder and (b) decoder [ChSE00]

Figure 2.3 illustrates the encoding and decoding procedures of the JPEG 2000. In the encoding procedure, the source image data is firstly discrete wavelet transformed (DWT). The resulting transform coefficients are then quantized and entropy encoded to form the output codestream. Reversely, the codestream is entropy decoded, dequantized, and inverse discrete transformed by the decoder to obtain the reconstructed image data. The procedures are applied to image tiles. The term 'tiling' refers to the partition of the

[7] http://www.jpeg.org/jpeg2000/index.html.

original (source) image into rectangular non-overlapping blocks (tiles), which are compressed independently, as though they were entirely distinct images.

To recapitulate, the detailed, step-by-step encoding procedure works as follows:

- The source image is decomposed into *components*.
- The image and its components are decomposed into rectangular *tiles*. The *tile-component* is the basic unit of the original or reconstructed image.
- By means of DWT, each tile-component is decomposed into different *decomposition levels*. These decomposition levels can create components with different resolutions.
- The decomposition levels are made up of *sub-bands* of coefficients describing the frequency characteristics of local areas of the tile-component.
- The sub-bands of coefficients are quantized and collected into rectangular arrays of *precincts*. Each precinct is further divided into non-overlapping *code-blocks*.
- The bit-planes of the coefficients in a code-block are entropy encoded in three coding passes: significance propagation, magnitude refinement, and cleanup. Some of the coefficients can be coded first at a higher quality than the background to provide a region of interest (ROI).
- The coding passes from the code-blocks are collected in *layers*.
- *Packets*, the basic units of the compressed data, are composed of one partition (precinct) of a single layer of a single decomposition level of a single tile-component.
- All the packets from a tile are interleaved in one of several progression orders and placed in one or more *tile-parts*, which have a descriptive *tile-part header* and can be interleaved in any order.

- *Markers* are added in the codestream to allow error resilience.
- The codestream has a *main header* at the beginning that describes the original image and the various decomposition and coding styles used to locate, extract, decode, and reconstruct the image with the desired resolution, fidelity, region of interest, and other characteristics.
- The optional file format, describing the meaning of the image and its components in the context of the application, may be applied to the codestream.

Thus, the data representing a specific tile, component, resolution, precinct, and layer appears in the codestream in a contiguous segment called a packet. Packet data is aligned at 8-bit (one-byte) boundaries. The order in which these packets are interleaved is called the progression order. The interleaving of the packets can progress along four axes: layer (L), component (C), resolution (R), and precinct/position (P). There are five built-in progression orders defined in the JPEG 2000 standard [ISO15444-1]: 1) LRCP, 2) RLCP, 3) RPCL, 4) PCRL, and 5) CPRL.

The SNR scalability can be demonstrated if the image's packets are transmitted from the lowest to the highest layers. The transmission of an image codestream employing the LRCP progression order may have the same effect. The spatial scalability, on the other hand, can be demonstrated if the image's packets are transmitted from the lowest to the highest resolutions. Employing the RLCP or RPCL progression order for the image codestream may generate the same effect in transmission. Overall, the employed progression order determines how the JPEG 2000 image is gradually displayed.

2.3.2 MPEG-4

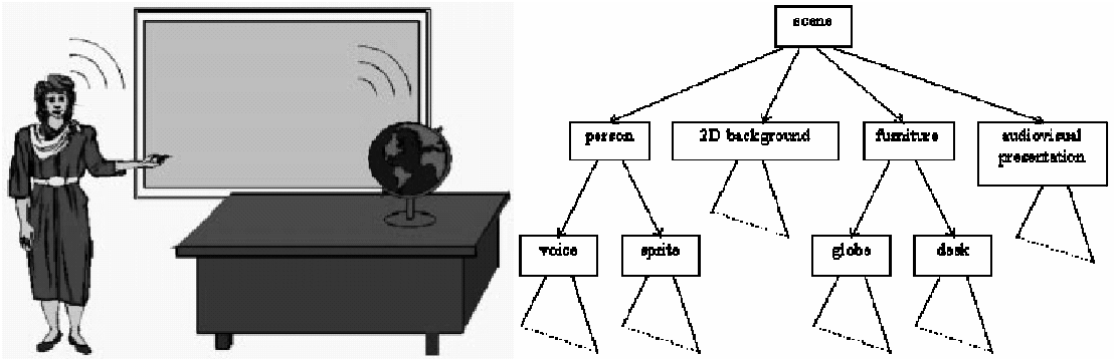
MPEG-4 is a new standard for digital audio and video developed by MPEG (Moving Picture Experts Group)[8]. In addition to the frame-based functionalities of the previous standards (MPEG-1 and MPEG-2), MPEG-4 also supports object-based manipulation. Officially called "Coding of Audio-Visual Objects", MPEG-4 is the first audio-visual representation standard that understands an audio-visual scene as a composition of objects (audio, video, or audio-visual/AV), according to a script that describes their spatial and temporal relationship [PeA97]. This object-based approach is also motivated by the increasing convergence between the telecommunications, computer, and TV/film technologies, leading to the mutual exchange of elements, formerly typical for each one of these areas. Other important features of MPEG-4 are improved coding efficiency, robustness in error-prone environments, and content-based scalability. MPEG-4's bit rates targeted for the video standard are between 5-64 kb/s for mobile applications and up to 2 Mb/s for TV/film applications [Sik97].

MPEG-4 supports *quality scalability*, which can be achieved by scaling the spatial or temporal resolutions. The spatial scalability is associated with the representation of an object at a certain moment in time; this means it may be associated with the number of pixels in the object, the accuracy of these pixels, the discrete cosine transform (DCT) coefficients, or other parameters. On the other hand, the frame/display rate is the issue of the temporal scalability. In addition, MPEG-4 supports *object scalability*, which is associated with the capability to control the number of simultaneous objects decoded and

---

[8] http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm.

displayed. A user with a low bandwidth Internet connection, for example, may request only the most important objects and drop the rest.



(a) **(b)**

Figure 2.4 (a) An MPEG-4 scene and (b) its tree structure [Lia99]

The basic building block of a multimedia presentation in MPEG-4 is object [Lia99]. There are many basic object types supported in MPEG-4: video objects, audio objects, texts, graphics, 2-D meshes, 3-D meshes, HTML objects, etc. These objects are placed inside a scene with a description describing their location, orientation, and other necessary properties. A scene in MPEG-4 is described using *Binary Format for Scenes* (BIFS), an efficient syntax for describing the composition of a scene with binary data. An example of an MPEG-4 scene is given in Figure 2.4(a). In this scene, the teacher is a moving video object. An audio object that stores the speeches spoken by the teacher is also associated with the video object. There are also graphic objects, such as the globe, the desk, and the white board. Simultaneously, a dynamic web page object – the presentation material for the course – is displayed on the white board. Each object has a built-in timing attribute, which can be utilized to synchronize the presentation timing of different objects. For instance, the synchronization of the lip movement of the video object and the audio object, the synchronization of the audio object and the page changes

on the white board, and so forth. BIFS describes an MPEG-4 scene with a hierarchical structure, which can be represented with a tree, as depicted in Figure 2.4(b). Each node of the tree is an AV object. In addition, the scene can be dynamic; that is, BIFS supports streaming delivery, so that the scene description can be updated dynamically. This feature allows the creation of a very complicated and lively presentation with little overhead.

MPEG-4 also supports user interactivity at the object level. A user can click on any object or move the mouse inside the perimeter of an object. This event may trigger a signal sent back to the server/encoder side. The property or composition of the scene can be changed according to this event. This object-based interactivity is very valuable for multimedia learning application, where users can explore on their own terms and the scene is adapted according to the user preferences.

To better understand the MPEG-4 mechanism, let us look into the MPEG-4 Video standard. The MPEG-4 Video Verification Model [Sik97] introduces the concept of *video object plane* (VOP) to enable the content-based interactive functionalities. It is assumed that each frame of an input video sequence is segmented into a number of arbitrarily shaped image regions (VOPs); each VOP may possibly cover particular image or video content of interest. Successive VOPs belonging to the same physical object in a scene are referred to as *video objects* (VOs), i.e., a sequence of VOPs of possibly arbitrary shape and position. The shape[9], motion, and texture[10] information of the VOPs belonging to the same VO is encoded and transmitted or coded into a separate *video object layer* (VOL).

---

[9] The VOP shape matrix is also referred as the *alpha plane*.

[10] Commonly each VOP has *Y*, *U*, and *V* texture matrices.

In addition, relevant information needed to identify each of the VOLs – and how the various VOLs are composed – is also included in the bitstream. This allows the separate decoding of each VOP and the required flexible manipulation of the video sequence.

The MPEG-4 compression algorithm employed for coding each VOP image sequence is based on the successful block-based hybrid DPCM (Differential Pulse Code Modulation) / DCT (Discrete Cosine Transform) coding technique. The coding algorithm encodes the first VOP in intraframe VOP coding mode (*I*-VOP). Each subsequent frame is coded using interframe VOP prediction (*P*-VOPs), in which only data from the nearest previously coded VOP frame is used for prediction. In addition, the coding of bi-directionally predicted VOP (*B*-VOPs) is also supported. After coding the VOP shape information, each color input VOP image in a VOP sequence is partitioned into non-overlapping "macroblocks". Each macroblock contains blocks of data from both luminance and chrominance bands – four luminance blocks (Y_1, Y_2, Y_3, Y_4) and two chrominance blocks (U, V) – each with size 8×8 pixels.

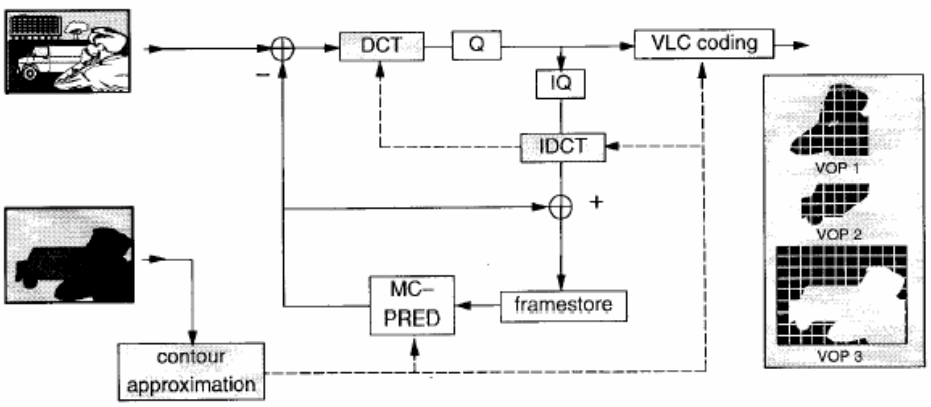


Figure 2.5 Block diagram of the basic MPEG-4 hybrid DPCM/DCT encoder and decoder [Sik97]

The basic diagram of the hybrid DPCM/DCT encoder and decoder structure for processing single Y, U, or V blocks and macroblocks is depicted in Figure 2.5. The

previously coded VOP frame *N*-1 is stored in a VOP framestore in both encoder and decoder. Motion estimation is performed on a block or macroblock basis; only one motion vector is estimated between VOP frame *N* and VOP frame *N*-1 for a particular block or macroblock to be encoded. The motion-compensated prediction error is calculated by subtracting each pixel in a block or macroblock belonging to the VOP frame *N* with its motion shifted counterpart in the previous VOP frame *N*-1. An 8×8 DCT is then applied to each of the 8×8 blocks followed by quantization (Q) of the DCT coefficients with subsequent run-length coding and entropy coding (VLC/Variable Length Coding). A video buffer is needed to ensure that a constant target bit rate output is produced by the encoder. The quantization stepsize for the DCT-coefficients can be adjusted for each macroblock in a VOP frame to achieve a given target bit rate and to avoid buffer overflow or underflow. The decoder uses the reverse process to reproduce a macroblock of VOP frame *N* at the receiver. After decoding the variable length words contained in the video decoder buffer, the pixel values of the prediction error are reconstructed. The motion-compensated pixels from the previous VOP frame *N*-1 contained in the VOP frame store are added to the prediction error to recover the particular macroblock of frame *N*.

The techniques adopted for the MPEG-4 Video Verification Model allow the "content-based" access or transmission of arbitrarily-shaped VOPs at various temporal or spatial resolutions. Receivers either not capable or willing to reconstruct the full resolution arbitrarily-shaped VOPs can decode subsets of the layered bitstream to display the objects at lower spatial or temporal resolution. A multiresolution representation can be achieved by downscaling the input video signal into a lower resolution video. The

downscaled version is encoded into a base layer bitstream with reduced bit rate. The prediction error (between an upscaled version – or the original input video signal – and the downscaled version) is encoded into an enhancement layer bitstream. Downscaled VOP signals can be reconstructed by only decoding the lower layer bitstream. It is important to notice that the display of the VOP at the highest resolution with reduced quality is also possible by only decoding the lower bit-rate base layer(s). Similarly, different frame rates can also be supported with a layered bitstream. Layering is achieved by providing a temporal prediction for the enhancement layer based on coded video from the lower layers. It is also possible to provide different display rates for different VOLs within the same video sequence (i.e., a foreground person of interest may be displayed with a higher frame rate than the remaining background). Last but not least, object scalability is supported as well. Some objects may not be decoded and used for reconstruction, while others are decoded and displayed using subsequent scaling or rotation. Moreover, new objects that do not belong to the original scene may be included. Since the bitstream of the sequence is organized in an object-layered form, the manipulation is performed on the bitstream level – without the need for further transcoding.

A few years back, a new scalable coding mechanism, called FGS (Fine Granularity Scalability), was amended to the MPEG-4 standard as the coding scheme for the streaming video profile [Li01]. Similar to the previous scalabilities, FGS also codes a video sequence into a base layer and an enhancement layer (some researchers considered the enhancement layer to be multiple layers). The base layer uses nonscalable coding to reach the lower bound of the bit-rate range. The enhancement layer is to code the

difference between the original picture and the reconstructed picture using bit-plane coding of the DCT coefficients. After encoding is completed, the bitstream of the FGS enhancement layer may be truncated into any number of bits per picture. The enhancement-layer video quality is proportional to the number of bits decoded by the decoder for each picture. Hence, FGS is capable of achieving continuous rate control for the enhancement layer.

A variation of FGS, labeled as progressive fine granularity scalability (PFGS), has been proposed to improve the efficiency of scalable video coding [WuLZ01]. PFGS has all the features of FGS, such as fine granularity bit-rate scalability, bandwidth adaptation, and error recovery. Unlike FGS, which only uses the base layer as a reference for motion prediction, PFGS uses several high-quality references to reduce the prediction error, resulting in higher coding efficiency.

The new, emerging multimedia standards – such as JPEG 2000, MPEG-4, and H.264 – give more advanced features than their predecessors. The most promising feature is their support for multi-scale presentation. Employing this feature, a multimedia object is transformed into a datastream that supports a range of scalabilities, such as quality, resolution, temporal, and object scalabilities. A user that is not capable or willing to view the full-quality or full-resolution of the object may opt for its representation, which is lower in quality or resolution and can be easily obtained from the datastream without further transcoding. Thus, the new multimedia standards can suit the user's constraints and preferences, in addition to reducing the user's perceived latency.

Nevertheless, an interactive application – involving continuous communication between the sender and the receiver – is often required to select the best-fit object's representation for a certain user. In an environment without such interactivity, like the Web, the server may send out the entire object and let the client selectively get the representation it wants. Alternatively, the server may give several options – corresponding to some predetermined constraints and preferences – for the client to explicitly choose one of them. While the former is wasting the Internet bandwidth, the latter is rigid and requires user's involvement perhaps for every visited Website. Use of client and content profiles, as suggested in the previous section, may help address the problem. Yet, it also asks for an automatic system that can understand both profiles and process the client requests accordingly.

2.4 Concluding Remarks

Researches on content caching and replication, intelligent network, and multimedia standards face the same challenges, which have been revealed in the previous chapter. Some of the challenges are clients' heterogeneity, increased multimedia data traffic, reduced user's perceived latency, personalized/customizable content, and easy, efficient deployment of Web services. Although each research area has its own emphases in dealing with the challenges, there is some common ground in their efforts. Some techniques are shared by researchers in different research areas. Some are unique to one research area but possibly can be applied to other research areas, as well. Overall, their

contribution has improved the Web content delivery considerably, in comparison with the Web in its early years.

Based on the findings of the past researches, we may deduce the essential factors that can improve the Web content delivery (some factors may overlap with the characteristics of future Web content delivery, listed in Section 1.4):

1. Use of Web intermediaries.

   Proxies and surrogates are greatly utilized in content distribution and intelligent network. Web intermediaries can reduce the origin servers' burden as well as offer technical and economical benefits.

2. Use of a caching system.

   Equipped with a caching system, a Web intermediary can further reduce Internet traffic and improve response time.

3. Efficiency of compressed data.

   Transferring compressed data can reduce Internet traffic and improve response time, too. Data compression has long been accommodated in the Web protocol. One important feature of the new multimedia standards is better coding efficiency, which means more advanced data compression.

4. Decomposition of a Web object.

   Dynamic content is decomposed into fragments so that it can be dealt with faster and more efficiently. Similarly, a multimedia object is decomposed and transformed into smaller packets or units, which can support multi-scale presentation well. This factor can directly address the problems of clients' heterogeneity and personalized / customizable content.

5. Delivery of deltas.

   A technique in content distribution, delta encoding was proposed to reduce data transfer. In the multimedia domain, motion compensation and prediction error are kinds of deltas. Moreover, the progressive display of a multimedia object is, in fact, the result of adding deltas to the presentation of the base layer. Delivery of deltas supports high data reuse, and consequently, reduces Internet traffic and user latency.

6. Use of meta-data.

   Intelligent network requires client's profile and content's description to be able to serve heterogeneous clients properly. Objects of the new multimedia standards are also rich of meta-data. A JPEG 2000 image has main, tile-part, and packet headers, each of which describes a specific part of the image data. Likewise, an MPEG-4 video object contains many meta-data, e.g., the shape, motion, and texture information of a VO (Video Object), the description of a VOL (Video Object Layer), and so forth.

Three out of six factors above are related to the Web object's data; those three factors are points 3–5. Thus, we may conclude that the object's data-type plays a vital role in the Web content delivery. If Web objects share the same data-type, surely they will be easy to maintain and efficient to process. However, it is very difficult, if not impossible, to enforce all Web objects to have a single data-type. Yet, we may suggest a data model, which all Web objects may refer to, to support fast, efficient, pervasive Web content delivery. The proposed data model is discussed shortly in the following chapter. We will return to the remaining factors (i.e., points 1, 2, and 6) in the later chapters.

Chapter 3

A Fine-Grained, Scalable Data Model

A fine-grained, scalable data model is presented in this chapter. It begins with an observation on some well-known multimedia standards. Based on the observation, the concept of object decomposition and construction is drafted. Then, the fine-grained, scalable data model is formally specified. Bringing out the data model's benefits into the Web content delivery, we devise a new adaptation called modulation. It is fast, reversible, yet exclusive content adaptation.

3.1 Background

If we observe most of the well-known multimedia data-formats, there exists the smallest unit of coded data which is quite recognizable and usually aligned to a particular number of bits boundary. For simplicity, the smallest unit is called an *atom* for the rest of this chapter. The atom usually cannot be partitioned anymore without firstly being decoded. For instance, let us look into three multimedia standards, namely JPEG, JPEG 2000, and MPEG-4.

JPEG [ITU92]

In the JPEG standard, compressed image data consists of exactly one image. The image contains one frame (in case of sequential and progressive modes) or more (for the hierarchical mode). Further, a frame contains one or more scans. If all components are interleaved together, just one scan is required for the given frame; by contrast, one scan is needed for each non-interleaved component. The progressive mode requires several scans, each of which improves the image presentation. A scan comprises *minimum coded units* (MCUs). An MCU in the JPEG standard can be regarded as an atom. [Note: Although an MCU may contain some data units, the data units are partial in the progressive mode so that they are less recognizable than the MCU.]

JPEG 2000 [ISO15444-1]

An image in the JPEG 2000 standard can be divided into tile-components. Each tile component is decomposed into several decomposition levels, each of which is made up of sub-bands of coefficients. The individual sub-bands are further divided into code-blocks. Compressed data of associated code-blocks is grouped in layers. Packets

of bit-stream data are then formed. A packet is a particular partition of one layer of one decomposition level of one tile-component. Hence, in the JPEG 2000 standard, a packet is an atom.

MPEG-4 [PeE02]

In MPEG-4 Visual, a video scene can be composed of some *visual/video objects* (VOs). A video object is an area of the video scene that may occupy an arbitrarily-shaped region and exist for an arbitrary length of time. Each VO comprises one or (in case of scalable video coding) more *video object layers* (VOLs). Each VOL contains some *video object planes* (VOPs). Each VOP is basically an instance of the video object at a particular time. When a VOP is coded and a time stamp is attached to it, it becomes an *access unit* (AU). Similarly, the MPEG-4 Audio decomposes an audio signal into frames, and the compressed data of one audio frame forms an *access unit* (AU). In both cases, an AU is an atom.

In a non-scalable multimedia data-format, only a single presentation can be constructed from those atoms. Therefore, all of the atoms must be obtained and decoded altogether to display the multimedia presentation. In a scalable multimedia data-format, however, more than one presentation can be constructed. In that case, some atoms represent the additional information which makes the multimedia presentation clearer (in case of SNR/signal-to-noise-ratio scalability) or finer (in case of spatial scalability). Exploiting this feature, we may generate different representations of a multimedia object without the need for converting the coded data. The only required processes are determining and collecting the necessary atoms and perhaps – if it is to be stored in a file format – modifying the meta-data embedded in the object so that the representation can

be displayed correctly. The simplicity of these processes should result in a minimal delay. And more importantly, it offers high data reuse. The representation can be used to get an improved representation by raising its atoms. Even the original object can be recovered.

Realizing the benefits above, we specify a fine-grained, scalable data model that matches well with many scalable multimedia data-formats. By formally specifying the data model, we can define the adapting processes applied to the scalable data model and later convey the benefits to Web content delivery. Following this idea, let us explain how an object should be decomposed and later be constructed in a scalable manner.

3.2 Concept of Object Decomposition and Construction

As mentioned in the previous section, an atom is the smallest unit of an object. Each atom should be distinct. Therefore, the atoms have attributes (properties) that differentiate one atom from another. In a scalable multimedia object, the attributes may correspond with the types of scalability. A scalable multimedia object may support more than one type of scalability. For an instance, a multimedia object may support quality (SNR) and resolution (spatial) scalability. For each atom of the object, we may say that it contributes to a certain level of quality scalability and a certain level of spatial scalability; those are the atom's attribute values.

Atoms with the same attribute values can be grouped together. The resulting groups are called *segments*. Referring back to the example above, the object's atoms can be grouped based on quality or resolution scalability. Different attributes employed in the grouping may result in different segments. The reason of introducing the segment is to

balance fine granularity and easy data processing. The granularity of an atom is so fine that handling and processing it may be troublesome and inefficient.

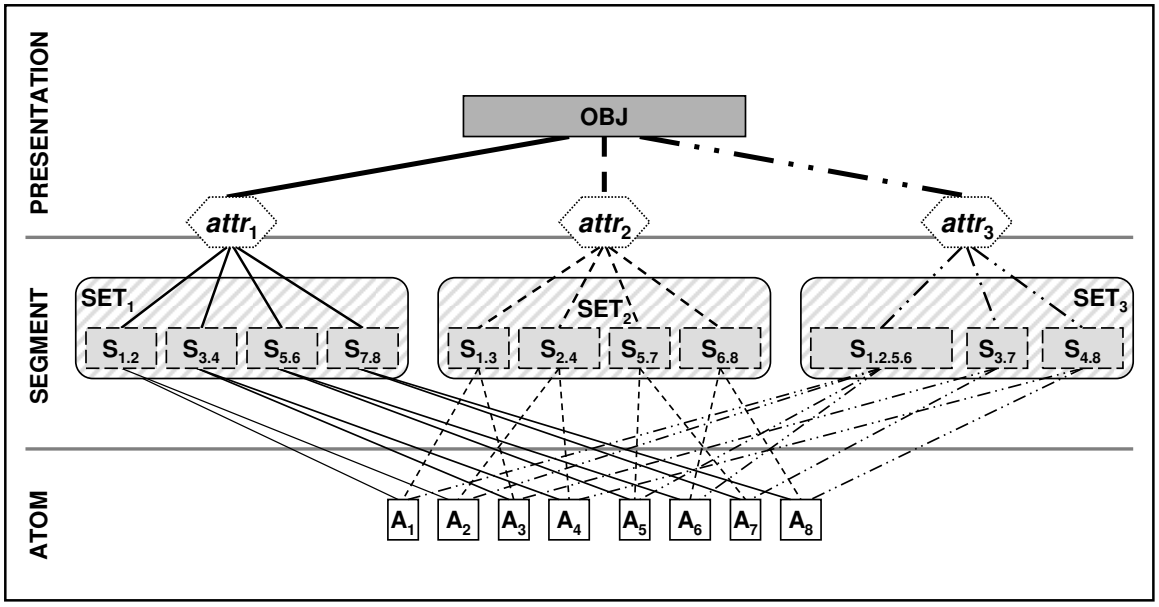


Figure 3.1 Object decomposition

Consequently, the data model comprises three data layers: presentation, segment, and atom. The presentation layer is where an object and its representations (also called versions or variants) are sorted into. Obviously, segments and atoms of the object are placed in the segment and atom layers, respectively. Figure 3.1 depicts the organization of the data layers. Entities in each data layer have specific characteristics. An entity of the presentation layer is heterogeneous since it contains fragments with diverse attribute values. As indicated by the data layer's name, the entity is presentable to the clients. On the contrary, an entity of the segment layer cannot be presented. However, homogeneity starts to appear in the segment layer; that is, the entity's fragments share one or more attribute values. An entity of the atom layer is homogeneous and has a unique combination of attribute values. Clearly, the entity is very primitive and far from being

presentable. In general, the entity becomes finer in granularity and more homogeneous in attribute values as it goes from the presentation layer to the atom layer.

In the illustration (see Figure 3.1), object OBJ can be decomposed into segments $S_{1.2}$, $S_{3.4}$, $S_{5.6}$, and $S_{7.8}$ (note: in reality the segments are often imaginary, that is why they are enclosed with dotted line) based on a set of attributes $attr_1$. Further, the segments can be decomposed into atoms A_1–A_8. This is a two-way transformation. We can regard it as a top-down transformation (from the presentation to atom layer). Likewise, we can also view it as a bottom-up transformation (from the atom to presentation layer). Hence, we can say that segments $S_{1.2}$, $S_{3.4}$, $S_{5.6}$, and $S_{7.8}$ are constructed by collecting the atoms according to the shared values of attributes in set $attr_1$. Since there may be more than one alternative to decompose the object, different sets of segments may be produced. Of object OBJ in the illustration, three sets of segments – SET_1, SET_2, and SET_3 – can be produced. Each set is associated with a distinct set of attributes, respectively $attr_1$, $attr_2$, and $attr_3$.

Continuing the previous illustration, Figure 3.2 demonstrates how various object's representations can be constructed. An object's representation is constructed from one or more segments. For examples: employing SET_1, representation $P_{1.2}$ is made up of segment $S_{1.2}$, while representation $P_{1.2.3.4}$ is made up of segments $S_{1.2}$ and $S_{3.4}$. Note that segment $S_{1.2}$ itself is not presentable; some headers (meta-data) need be added to the segment to become representation $P_{1.2}$. Adding one or more segments to a representation can give a higher-fidelity representation, e.g., adding segment $S_{5.6}$ to representation $P_{1.2.3.4}$ gives representation $P_{1.2.3.4.5.6}$. On the other hand, eliminating one or more segments from a representation can give a lower-fidelity representation, e.g., eliminating segment $S_{3.4}$

from representation $P_{1.2.3.4}$ gives representation $P_{1.2}$. When the segments of SET_1 are combined altogether, the resulting representation is indeed the original object, OBJ. We may also work on segments of the other set (SET_3, for example) to construct the object's representations.

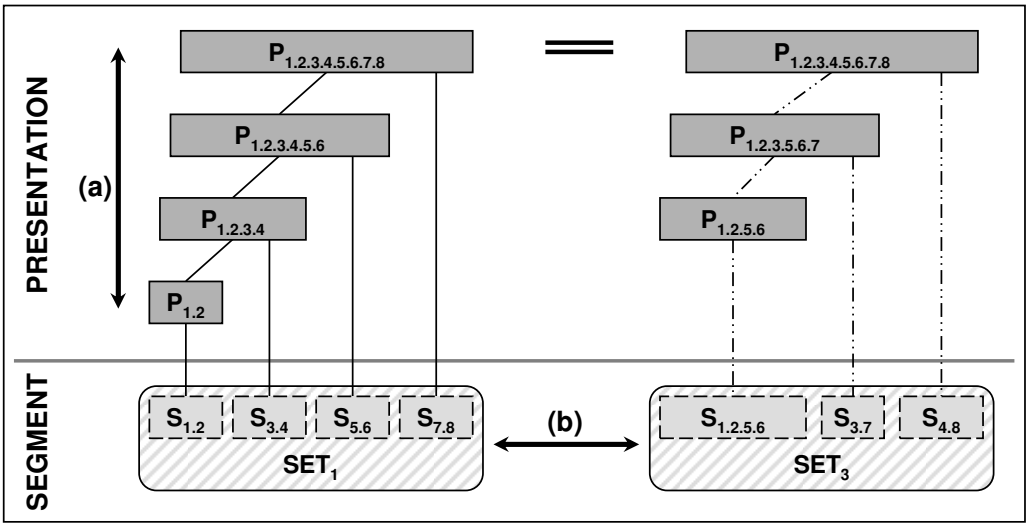


Figure 3.2 Construction of representations
Adaptability aspects: (a) Scalability and (b) Variability

Analyzing the construction of various object's representations, we can observe two kinds of adaptability in the data model. Figure 3.2 shows the two adaptability aspects: *scalability* and *variability*. The scalability aspect (the vertical dimension in Figure 3.2) deals with the variety of representations constructed from a particular set of segments. The variability aspect (the horizontal dimension in Figure 3.2) deals with the variety of decomposing alternatives supported by the object. Elaboration on the adaptability aspects can be found in the discussion about the data model.

Now we are ready to formally specify the fine-grained, scalable data model. The concept of object decomposition and construction discussed here can help to understand the proposed data model.

3.3 Specifications of Data Model

The fine-grained, scalable data model is specified in this section using the set theory. We begin with some definitions.

3.3.1 Definition 1: Object

An object comprises segments. The decomposition of an object into segments depends on one or more attributes. Moreover, there may be more than one alternative to decompose an object into segments. A set of attributes determining a decomposing alternative is called a *variation*[11]. The object decomposition can be formulated as follows

$$OBJ\begin{cases} O^A = \{S_1^A \cup S_2^A \cup \ldots \cup S_a^A\} \\ O^B = \{S_1^B \cup S_2^B \cup \ldots \cup S_b^B\} \\ \vdots \quad \vdots \quad \vdots \quad \vdots \\ O^V = \{S_1^V \cup S_2^V \cup \ldots \cup S_v^V\}, \end{cases}$$

where superscript upper-case letters $A,B,\ldots,V$ denote the variations employed by sets (better called *obsets*[12]) $O^A, O^B, \ldots, O^V$ respectively, and subscript lower-case letters $a,b,\ldots,v$ are arbitrary numbers, each of which signifies the number of segments (or, the *scale*) in the respective obset.

To present the object appropriately, each segment of an employed obset requires the preceding segments. Suppose a variation (e.g., V) is employed for the presentation, the

---

[11] A variation in the data model corresponds to a type of scalability in the common term (e.g., quality scalability, resolution scalability, object scalability, etc.).

[12] Derived from the words "object" and "set" since the whole set is indeed the object itself.

first segment of the respective obset (S_1^V) must be present early on, the second segment (S_2^V) must be present if the third segment (S_3^V) is present, the third segment must be present if the fourth segment (S_4^V) is present, and so forth. Let $\mathbb{V}\mathtt{ar}$ be the universe of all variations and Present(*S*) be a function returning true when segment *S* is present in the presentation, the condition above can be stated as follows

$$\forall V : \mathbb{V}\mathtt{ar} \bullet \forall i : \mathbb{N}_1 \bullet \exists j : \mathbb{N}_1 \bullet (j < i) \wedge \text{Present}(S_i^V) \Rightarrow \text{Present}(S_j^V).$$

In the above definition, two terms – "variation" and "scale" – have been introduced. These two terms are closely related to "variability" and "scalability"[13], introduced in the earlier section. The variability of an object is determined by the number of variations supported by (or, the number of obsets possibly generated from) the object. On the other hand, the scale (number of segments) in a given obset (e.g., the scale of obset O^A is *a*) determines the scalability of the object with respect to the obset's variation (variation *A*). Note that a variation must be specified when we talk about scale and scalability. We can compare the adaptability of two objects in these two aspects, namely variability and scalability. For an example, suppose we compare the adaptability between objects OBJ_A and OBJ_B. Object OBJ_A supports two variations, `qty` and `res`, with scales 4 and 3 respectively; whereas object OBJ_B just supports variation `qty`, and its scale is 5. We may say that the variability of OBJ_A is better than that of OBJ_B since OBJ_A has more variations than OBJ_B. However, the scalability of OBJ_A with respect to variation `qty` is inferior to that of OBJ_B; OBJ_A's scale is 4 while OBJ_B's scale is 5, with respect to

---

[13] Scalability here is not to be confused with scalability in the usual sense. We refer to the types of scalability as "variations" (look at footnote 11 in this chapter).

variation `qty`. The object's scalability and variability will be exploited later in the adapting operations.

3.3.2 Definition 2: Segment and Atom

A segment is a collection (set) of atoms. An atom is the smallest unit of an object. Atoms are indivisible, and they are the basic building blocks of an object. The relation between segments and atoms is depicted as follows

$$\begin{aligned} S_1^V &= \{A_1^V, A_2^V, \ldots, A_i^V\} \\ S_2^V &= \{A_{i+1}^V, A_{i+2}^V, \ldots, A_j^V\} \\ &\vdots \quad \vdots \quad \vdots \quad \vdots \\ S_v^V &= \{A_{m+1}^V, A_{m+2}^V, \ldots, A_n^V\}, \end{aligned}$$

where $\{S_1^V \cup S_2^V \cup \ldots \cup S_v^V\} = \text{obset } O^V = OBJ$, and n is the total number of atoms in object *OBJ*. Here, *V* is the employed variation. Replacing all segments of obset O^V with the atoms, we get

$$\{\{A_1^V, A_2^V, \ldots, A_i^V, A_{i+1}^V, A_{i+2}^V, \ldots, A_j^V, \ldots, A_{m+1}^V, A_{m+2}^V, \ldots, A_n^V\}\} = O^V = OBJ .$$

The subscript ordered numbers 1,2,…,*n* attached to the atoms signify the atoms' identity with respect to variation *V*. Employing different obsets in the replacement should give exactly the same atoms, even though they may not be in the same order and may have different identity numbers. In other words, of an object, the composition and the number of segments vary with respect to the employed variation (or, obset), but the atoms contained in the entire segments as well as the total number of atoms remain the same. Notice also that a segment is a set of atoms, whereas an object is a set of a single set of

atoms. To put it simply, there is just one entity in an object, and that entity is the set of atoms. Thus, it shows that the concept of segments can be imaginary.

Within a particular obset, there are no overlapping segments; that is, an atom can only be contained in one segment. This condition can be expressed as follows

$$\forall V : \mathbb{V}\mathrm{ar} \bullet \forall i, j : \mathbb{N}_1 \bullet i \neq j \Rightarrow \left(S_i^V \cap S_j^V = \varnothing\right).$$

3.3.3 Definition 3: Representation

A representation is part of an object which can represent the object well. A representation is constructed from one or more sequential segments of a particular obset. The scale in the obset determines the number of representations that can be constructed. Given that object OBJ = obset $O^V = \left\{S_1^V \cup S_2^V \cup \ldots \cup S_v^V\right\}$, there are exactly v representations that can be constructed from the obset. Those are,

$$\begin{aligned} P^{(V/1)} &= \left\{S_1^V\right\} \\ P^{(V/2)} &= \left\{S_1^V \cup S_2^V\right\} \\ &\vdots \qquad \vdots \\ P^{(V/v)} &= \left\{S_1^V \cup S_2^V \cup \ldots \cup S_v^V\right\}. \end{aligned}$$

In general, the symbol $P^{(V/s)}$ denotes a representation with V as its variation and s as its scale in the respective variation (where $1 \leq s \leq v$, and v is the number of segments in obset O^V). Simply, the symbol $P^{(V/s)}$ means a representation containing segments of obset O^V, the indices of which are lower than or equal to s. Since the number of segments in obset O^V is v, representation $P^{(V/v)}$ is identical to obset O^V. Alternatively, the variation can be ignored if its scale is at the fullest; that is $P^{(V/v)} = OBJ$ (not P, since it is unusual to use

the representation's symbol without at least a variation and its corresponding scale). Although object *OBJ* may have other variations, they are ignored (not printed) in the representation since their corresponding scales are at the fullest.

Since a representation is part of an object, the atoms of the representation are also part of the object's atoms. The condition can be depicted as follows

$$\exists!T_1 : OBJ \bullet \exists!T_2 : P^{(V/s)} \bullet P^{(V/s)} \subseteq OBJ \Rightarrow T_2 \subseteq T_1 .$$

It means: given that T_1 is the only entity of object *OBJ* and T_2 is the only entity of representation $P^{(V/s)}$ (recall that an object – and a representation, too – is a set containing just one entity, i.e., the set of atoms), $P^{(V/s)}$ is a representation of object *OBJ* if $T_2 \subseteq T_1$.

3.3.4 Definition 4: Supplement

Similar to a representation, a supplement is also part of an object. However, a supplement cannot represent the object and often is not presentable. It is used to enhance a representation to be more presentable. There are more varieties of supplements than those of representations. This is reasonable since there are many representations and each representation can be enhanced differently by some supplements. Given that object *OBJ* = obset $O^V = \{S_1^V \cup S_2^V \cup \ldots \cup S_v^V\}$ and representation $P^{(V/1)} = \{S_1^V\}$, possible supplements to enhance the representation are

$$\begin{aligned}
\tilde{P}^{(V/1\sim 2)} &= \{S_2^V\} \\
\tilde{P}^{(V/1\sim 3)} &= \{S_2^V \cup S_3^V\} \\
&\vdots \qquad\quad \vdots \\
\tilde{P}^{(V/1\sim v)} &= \{S_2^V \cup S_3^V \cup \ldots \cup S_v^V\}.
\end{aligned}$$

In general, the symbol $\tilde{P}^{(V/s\sim t)}$ denotes a supplement that can enhance the scale of representation $P^{(V/s)}$ with respect to variation V from s to t (where $1 \le s < t \le v$, and v is the number of segments in obset O^V). Supplement $\tilde{P}^{(V/s)}$ implicitly has the same meaning as supplement $\tilde{P}^{(V/s\sim v)}$, which can enhance the scale of representation $P^{(V/s)}$ with respect to variation V to its fullest (provided that v is the number of segments in obset O^V).

Since a supplement is also part of an object, the atoms of the supplement are part of the object's atoms, as well. Further, the condition can be depicted as follows

$$\exists! T_1 : OBJ \bullet \exists! T_2 : \tilde{P}^{(V/s\sim t)} \bullet \tilde{P}^{(V/s\sim t)} \subseteq OBJ \Rightarrow T_2 \subseteq T_1 .$$

Referring back to the object decomposition in Figure 3.1, a supplement cannot be placed in any layer. On the one hand it is not in the presentation layer since it is not presentable; on the other hand it cannot be classified as a segment due to its heterogeneity. Perhaps it should be placed in between the presentation and segment layers. Anyway, we prefer not to place a supplement in any data layer since it is a sideline in the object decomposition. The role of a supplement, however, is quite important to achieve efficiency.

In addition to the basic entities specified above, the data model also involves some adapting operations to transform one entity to another. The adapting operations are specified shortly. For the next discussion, assume that

$$\begin{aligned} OBJ = O^V &= \left\{ S_1^V \cup S_2^V \cup \ldots \cup S_v^V \right\} \\ &= O^W = \left\{ S_1^W \cup S_2^W \cup \ldots \cup S_w^W \right\}, \end{aligned}$$

where V and W are variations supported by object *OBJ*.

3.3.5 Operation 1: Selection

Selection ($F^{V=s}$) is an operation to derive a representation from an object or another representation. Selection is defined as follows

$$F^{V=s}(OBJ) = F^{V=s}(O^V) = \left\{ \bigcup_{i=1}^{s} S_i^V \right\},$$

where V is the reference variation and s is the reference scale ($1 \le s \le v$, v is the number of segments in obset O^V). The example below shows how to get a representation of an object by a selection operation.

$$\begin{aligned} P^{(V/2)} &= F^{V=2}(OBJ) \\ &= F^{V=2}(O^V) \\ &= F^{V=2}(\{S_1^V \cup S_2^V \cup \ldots \cup S_v^V\}) \\ &= \left\{ \bigcup_{i=1}^{2} S_i^V \right\} \\ &= \{S_1^V \cup S_2^V\} \end{aligned} ,$$

where $P^{(V/2)}$ is a representation of object *OBJ* with the scale in variation V equal to 2.

3.3.6 Operation 2: Inverse-Selection

Inverse-selection ($\tilde{F}^{V=s\sim t}$) is an operation to derive a supplement from an object or a representation. Inverse-selection is defined as follows

$$\tilde{F}^{V=s\sim t}(OBJ) = \tilde{F}^{V=s\sim t}(O^V) = \left\{ \bigcup_{i=s+1}^{t} S_i^V \right\},$$

where V is the reference variation, s the reference scale, and t the target scale ($1 \le s < t \le v$, v is the number of segments in obset O^V). If the target scale is not given, the default

target scale – which is the number of segments in the corresponding obset – is assumed. Thus, operation $\tilde{F}^{V=s}$ implicitly means

$$\tilde{F}^{V=s}(OBJ) = \tilde{F}^{V=s\sim v}(OBJ) = \tilde{F}^{V=s\sim v}(O^V) = \left\{ \bigcup_{i=s+1}^{v} S_i^V \right\}.$$

The next example shows how to get a supplement of an object by an inverse-selection operation.

$$\begin{aligned}
\tilde{P}^{(V/2\sim 4)} &= \tilde{F}^{V=2\sim 4}(OBJ) \\
&= \tilde{F}^{V=2\sim 4}(O^V) \\
&= \tilde{F}^{V=2\sim 4}\left(\left\{S_1^V \cup S_2^V \cup S_3^V \cup S_4^V \cup \ldots \cup S_v^V\right\}\right) \\
&= \left\{ \bigcup_{i=2+1}^{4} S_i^V \right\} \\
&= \left\{S_3^V \cup S_4^V\right\}
\end{aligned} ,$$

where $\tilde{P}^{(V/2\sim 4)}$ is a supplement of object *OBJ* which can enhance the scale of representation $P^{(V/2)}$ with respect to variation *V* from 2 to 4.

3.3.7 Operation 3: Join

Join (F^{V+}) is an operation to construct a representation by combining a lower-fidelity representation and a suitable supplement. Join is defined as follows

$$F^{V+}\left(P^{(V/s)}, \tilde{P}^{(V/s\sim t)}\right) = F^{V+}\left(\left\{\bigcup_{i=1}^{s} S_i^V\right\}, \left\{\bigcup_{i=s+1}^{t} S_i^V\right\}\right) = \left\{\bigcup_{i=1}^{t} S_i^V\right\},$$

where *V* is the reference variation, $P^{(V/s)}$ and $\tilde{P}^{(V/s\sim t)}$ the relevant representation and supplement, *s* the reference scale, and *t* the target scale ($1 \le s < t \le v$, *v* is the number of segments in obset O^V). Notice that the reference variation and scale of both the

representation and the supplement must match each other for the operation to work properly. The target scale of the supplement also becomes the scale of the resulting representation. In the example below, representation $P^{(V/2)}$ and supplement $\tilde{P}^{(V/2\sim4)}$ – both are the results of the selection and inverse-selection operations in the previous examples – are combined by a join operation.

$$\begin{aligned} P^{(V/4)} &= F^{V+}\left(P^{(V/2)}, \tilde{P}^{(V/2\sim4)}\right) \\ &= F^{V+}\left(\left\{\bigcup_{i=1}^{2} S_i^V\right\}, \left\{\bigcup_{i=2+1}^{4} S_i^V\right\}\right) \\ &= \left\{\bigcup_{i=1}^{4} S_i^V\right\} \\ &= \left\{S_1^V \cup S_2^V \cup S_3^V \cup S_4^V\right\} \quad . \end{aligned}$$

The result, $P^{(V/4)}$, is a representation with scale 4 in variation *V*.

3.3.8 Operation 4: Translation

Translation ($F^{V\to W}$) is an operation to change the variation of a representation. Translation is defined as follows

$$F^{V\to W}\left(P^{(V/s)}\right) = O^W \cap P^{(V/s)} = \left\{\bigcup_{i=1}^{w} S_i^W\right\} \cap P^{(V/s)} = \left\{\bigcup_{i=1}^{w} S_i^{W(V/s)}\right\},$$

where *V* and *W* are the reference and target variations, respectively. The operation neither drops nor adds the representation's atoms, but merely rearranges them into different segments (i.e., changes the grouping of the atoms). Translation is never employed alone, since it does not change the representation in any way. It is employed only if the representation's current variation does not match the variation exploited by the operation being applied to the representation. In the example below, a selection operation in

variation W is applied to representation $P^{(V/4)}$, the result from the previous example. Since the selection operation exploits a different variation (W) to the representation's currently-employed variation (V), a translation operation is required to change the representation's variation.

$$\begin{aligned} P^{(W/2;V/4)} &= F^{W=2}\left(P^{(V/4)}\right) = F^{W=2}\left(F^{V\to W}\left(P^{(V/4)}\right)\right) \\ &= F^{W=2}\left(O^{W} \cap P^{(V/4)}\right) \\ &= F^{W=2}\left(\left\{\bigcup_{i=1}^{w} S_i^{W}\right\} \cap \left\{S_1^{V} \cup S_2^{V} \cup S_3^{V} \cup S_4^{V}\right\}\right) \\ &= F^{W=2}\left(\left\{\bigcup_{i=1}^{w} S_i^{W(V/4)}\right\}\right) \\ &= \left\{\bigcup_{i=1}^{2} S_i^{W(V/4)}\right\} \\ &= \left\{S_1^{W(V/4)} \cup S_2^{W(V/4)}\right\} \end{aligned} \quad ,$$

where $P^{(W/2;V/4)}$ is a representation with scale 2 in variation W and scale 4 in variation V. Translation operation $F^{V\to W}$ changes the variation of representation $P^{(V/4)}$ (and its underlying segments) from V to W, whilst selection operation $F^{W=2}$ obtains the first two segments of the translated representation.

3.3.9 How Is It Useful?

Selection, inverse-selection, and join are operations exploiting the scalability of an object in a particular variation. Applying these operations to a given object's representation, not only may we obtain a lower-fidelity representation, but we may also obtain a higher-fidelity representation. Suppose this feature is implemented in Web content delivery, a cached object's representation may be used to serve client requests for

the same, lower-fidelity, as well as higher-fidelity representations. The high data reuse portrayed here offers efficiency greatly needed in Web content delivery.

Translation, on the other hand, is an operation exploiting the variability of an object. Translation makes inter-variation operations possible. Combining translation with the other operations, we may obtain varied representations (in different variations) from a given object's representation. In Web content delivery, this feature even bolsters high data reuse.

Exploiting this benefit, a new content adaptation is devised. Compared to transcoding – the oft-cited content adaptation – the new adaptation should be faster since its adapting operations are simpler, as noticed in the above specifications. The following section elaborates modulation, the new adaptation, in detail.

3.4 Modulation – A Scalable Adaptation

The Oxford English Dictionary[14] defines modulation as the action of forming, regulating, or varying according to due measure and proportion. Here, we define *modulation* as the process to obtain an object's representation by means of adjusting (dropping and/or adding) the building blocks of the object. The building blocks (atoms) of an object could be fragments, layers, packets, units, or whatever applicable. Modulation has the following characteristics:

---

[14] The Oxford English Dictionary, 2nd edition; prepared by J. A. Simpson and E. S. C. Weiner; Vol. IX, page 955; Oxford University Press, 1989.

1. Since processes in modulation are basically dropping atoms from and/or adding ones to an object's representation without involvement of any complex computation, the demand for computing resources is expected to be minimal. The overall processes should be *fast*. Thereby, it can be carried out anywhere, even at the Web server, without noticeable decrease in performance. By contrast, transcoding is quite resource consuming and, therefore, rarely executed at the Web server.
2. Modulation is an *exclusive* process. It is an adaptation within a data-format. It means the modulation's result always has the same data-format as the original object. Scalable data-formats like progressive JPEG, JPEG 2000, and MPEG-4 are potential targets for modulation. If a client device does not support the scalable data-format, then transcoding is required for conversion between data-formats. This may be the only drawback of modulation.
3. Unlike transcoding, which just transforms a high-fidelity representation to a low-fidelity one, modulation is *reversible*[15]. A representation can be obtained by dropping a few atoms from an object. Conversely, the original object can be retrieved by adding the missing atoms to the representation. As mentioned before, this reversible property causes high data reuse possible in modulation.

An illustration is given here to demonstrate modulation's benefits. Suppose an image supports two types of scalability, namely quality (`qty`) and resolution (`res`) scalability. The image comprises 8 (eight) atoms. Classification of the image's atoms follows the illustration in Figure 3.2. It is reproduced in the top-right box of Figure 3.3;

---

[15] Another definition of modulation in the Oxford English Dictionary:
[*Biol.*] Reversible variation in the activity or form of a cell in response to a changing environment.

the bottom set consists of 4 (four) segments of quality scalability, whereas the top set consists of 3 (three) segments of resolution scalability. The image resides in a Web server and is served to 4 (four) clients with different devices. Moreover, there is a caching proxy, placed between the server and clients, to store passing data locally for future reuse. Then, the following events occur in sequence (please refer to Figure 3.3).

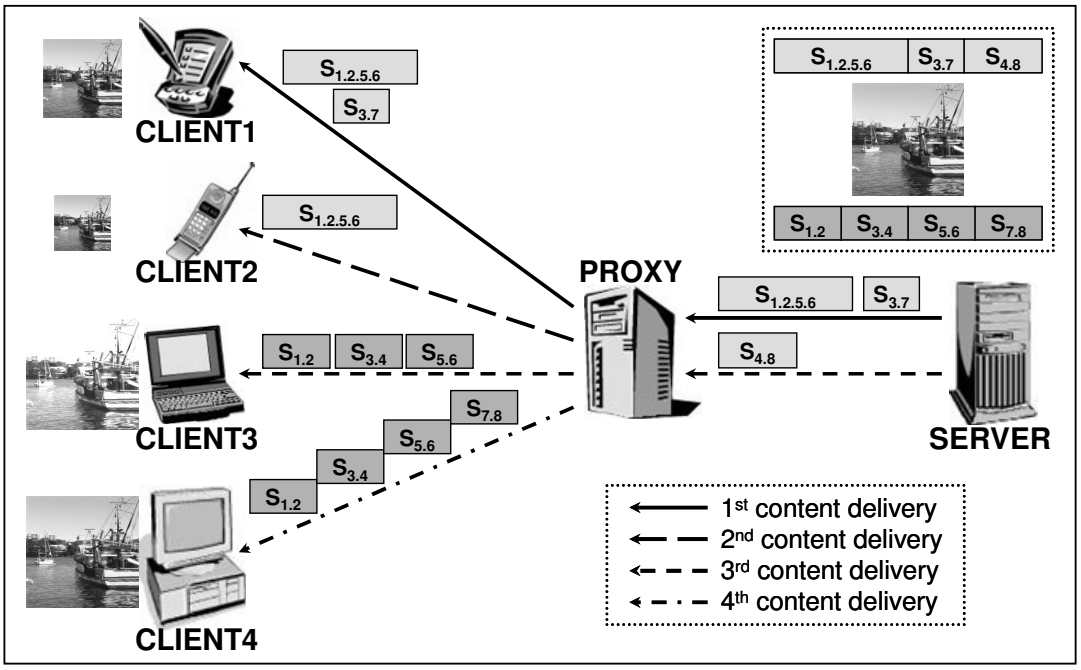


Figure 3.3 Modulation on an image

1. The first client, using a PDA, requests the image through the proxy. A medium-sized representation of the image is considered suitable for the PDA's screen. Not having the image in its cache, since this is the first request, the proxy passes on the request to the Web server. The server replies the request by modulating the image (dropping a segment with respect to resolution scalability by applying a selection operation $F^{res=2}$ to the image) and sending out the resulting image (the medium-sized representation) comprising segments $S_{1.2.5.6}$ and $S_{3.7}$. Receiving the representation, the proxy stores it

in the proxy's cache while simultaneously sending it to the first client (note: observe arrows of the 1st content delivery in the figure).

2. The second client requests the image from a cellular phone. Due to its smaller screen, the cellular phone can only display a small-sized representation of the image. The proxy can reply this request – without the server's involvement – by modulating the cached representation (dropping another segment with respect to resolution scalability by applying a selection operation $F^{\mathtt{res=l}}$ to the representation) to obtain the small-sized representation comprising just segment $S_{1.2.5.6}$. Hence, no data transfer from the server is required for the second request (note: there is no arrow from the server to the proxy for the 2nd content delivery in the figure).
3. Using a notebook connected to the Internet via a modem, the third client requests the image. Because of its sophisticated device, the client can display the full-sized representation of the image. However, its slow bandwidth connection only allows the client to obtain a less-quality representation of the image. There are two separate modulating operations at the server and proxy. Since the proxy only has a medium-sized representation in its cache, it requests for a supplement comprising just segment $S_{4.8}$ from the server. The server, in turn, modulates the image to obtain the requested supplement (by applying an inverse-selection operation $\tilde{F}^{\mathtt{res=2}}$ to the image) and returns the result to the proxy. Receiving the server's reply, the proxy joins the supplement with the cached representation (by applying a join operation $F^{\mathtt{res+}}$). Up until this point, the proxy has a full-sized image. It then modulates the image to obtain a less-quality representation (by applying a translation operation $F^{\mathtt{res}\rightarrow\mathtt{qty}}$, followed

by a selection operation $F^{qty=3}$). The end result (corresponding with a representation comprising segments $S_{1.2}$, $S_{3.4}$, and $S_{5.6}$ with respect to quality scalability) is transferred to the third client. The server-to-proxy link here is used to transfer segment $S_{4.8}$ only (note: observe the arrow from the server to the proxy for the 3rd content delivery in the figure).

4. The fourth client requests the image from a desktop PC connected to the Internet via a high-speed broadband. The client can be served with a full-sized, full-quality image. Since the proxy has the requested image fully in its cache, it just sends the image out to the client without any adaptation. Once again, the server is not involved in serving the client (note: no arrow from the server to the proxy for the 4th content delivery in the figure).

The third client may be served differently and more efficiently. Instead of requesting a supplement comprising segment $S_{4.8}$, the proxy may request for a supplement comprising segment S_4 from the server. Later on when serving the fourth client, the proxy then requests for a supplement comprising segment S_8 from the server. In that case, the term "on-demand delivery" can be aptly applied. Another thing, while waiting for the server's reply, the proxy can transfer the cached segments to the client, if possible. For an example, when serving the third client, the proxy may transfer segment $S_{1.2}$ to the client; segment $S_{5.6}$ cannot be sent out because the preceding segment $S_{3.4}$ – which has not been constructed at that time – must be sent out first. By employing this non-delaying data transfer, the client's perceived latency may be kept low.

The illustration shows that modulation can serve every single client with the best-fit representation of the image. The image's representations can be generated in a simple

and fast way. Clients would be able to access the image within a short delay. To gain the most benefits, modulation should be employed at the proxy as well as at the origin server. This way, on-demand delivery – which does not strain the Internet traffic – can be attained. The illustration also demonstrates the reversible property of modulation, which offers high data reuse. Any cached representation often can be used – either fully or partially – to serve subsequent client requests.

Emerging video standards like MPEG-4 and H.264 support scalable presentation. To a certain extent, modulation can also help in delivering video objects of those standards in a heterogeneous environment. There are two ways of transmitting a video object from a server to a client. First, the video object can be sent out just like a normal Web object. Second, it can be streamed by means of an RTSP (streaming) server. Modulation is definitely beneficial in the former case. Since the video object is in a scalable data format, modulation can adapt it to fit the client's characteristics in a fast and efficient manner. For example, modulation may drop some enhancement layers of the video object corresponding to high quality presentation, drop some enhancement layers corresponding to high resolution presentation, or drop some frames to reduce the video presentation's frame rate. As for the latter case, modulation's benefit is limited since some RTSP servers can stream the video object in multiple channels – each of which contains just a layer – which the client can selectively subscribe according to its needs. However, if the RTSP server is unable to split the video object into multiple layers (and stream them in multiple channels), or if the client is unable to subscribe to multiple channels, a proxy with modulation's capability can be placed between the RTSP server and the client to provide the service.

3.5 Related Work

The idea of partitioning a multimedia object and delivering the part(s) to represent the object on the Internet is not entirely ours. Some previous work on this research aspect is outlined in this section.

Kangasharju et al. [KaKO98] proposed a Soft Caching Proxy, in which a progressive JPEG image can be served to the clients in different qualities. The purposes of serving a range of image qualities to the clients are to reduce the delivery time, in particular for the clients with a low-bandwidth Internet connection, and to allow more efficient cache usage. The rationale behind the second purpose is as follows: a cached large image (>15KB), which is being evicted to make space for new objects, is recoded by the proxy into a progressive JPEG image, and then, one or two layers of the progressive image are discarded. Since the evicted image is just partially discarded from the proxy's cache, it can still be served to clients requesting the image. So, the system can reduce the requirement for fetching images from the origin servers and can keep more objects in the proxy's cache.

Smith et al. [SmCL99] partitioned large images (> 10K × 10K pixels) in the spatial and frequency domains for fast image retrieval. Partitioning an image in the spatial domain creates equal-sized tiles, each of which corresponds to a spatial portion (i.e., a certain region) of the image. On the other hand, partitioning an image in the frequency domain – usually by means of wavelet – creates subbands that are logarithmically spaced. The low-frequency wavelet subband serves as a coarse, low-resolution version of the image. The result of partitioning the image in both domains is a set of view elements,

which are smaller, easier to maintain, and cheaper to process than the original large image. Later on, a particular image view (representing a certain region and resolution of the image) can be synthesized from the view elements.

Rejaie et al. [ReYH00] employed hierarchical encoding to split a video stream into a base layer, containing the most essential low quality information, and higher layers, providing optional quality enhancement information, so that quality adaptive streaming can be delivered through the Internet. Further, each layer of the encoded stream is divided into equal-sized pieces called segments to allow fine-grained prefetching and cache replacement in the multimedia proxy cache. Initially, the proxy cache only fetches segments of the base layer of the video stream. When there is spare bandwidth, due to less traffic or a cache hit, the proxy cache may prefetch segments of the higher layers to improve the video stream's quality. Cache replacement is done by discarding segments of the higher layers from the end; hence, it can maximize the cache's efficiency.

It is obvious that previous work had exploited inherent scalability in some multimedia data-formats. Alas, study on the general data model of the scalable multimedia data-formats is still lacking. A formal data model is necessary to understand clearly the concept of object decomposition and construction in the scalable multimedia data-formats. Moreover, the operations specified in the data model can help develop the appropriate applications to adapt the scalable multimedia objects. Actually, Chi and Cao [ChC02] had proposed a scalable data model. However, their proposal does not enlighten the heterogeneous attributes inherent in a scalable multimedia object. In addition, it only considers a single scalability's type, i.e., the quality scalability. Our proposed data model

offers a more extensive perspective of the characteristics of a scalable multimedia data-format.

Modulation, an adapting mechanism directly derived from the data model, has been introduced and further illustrated with a scalable image. Modulation is not just limited to an image adaptation; it can be applied to any scalable multimedia data-format. Actually, it should be extended to all kinds of Web contents – including textual content, if possible – since it could bring a dramatic improvement in the overall Web content delivery. However, before we could come to that supposition, let us first investigate its benefits in a smaller scope. The next chapter discusses modulation's implementation in the JPEG 2000 standard.

Chapter 4

Modulation in JPEG 2000

In this thesis, modulation – specified at the end of the previous chapter – is implemented in the JPEG 2000 standard. This chapter elaborates on the implementation. To begin with, the reasons of using the JPEG 2000 standard to illustrate modulation are explained. Three JPEG 2000 modulators – namely `JP2Selector`, `JP2Joiner`, and `JP2Converter` – are then presented in detail.

4.1 Why Use JPEG 2000?

In a scalable multimedia data-format, an object's coded data has been placed in a particular arrangement so that the object can be displayed progressively, resulting in a multimedia presentation with increasing clarity, resolution, detail, or others alike. In the Web environment, the progressive multimedia presentation is preferable to the line-by-line or, worse, the late-appearing multimedia presentation. It is true that the progressive multimedia presentation demands more computing resources at the client appliances for repeated decoding. Yet, it may cut down the client's perceived latency since the client may grasp the object's context sooner. Although there are not many scalable multimedia data-formats currently, the figure will surely improve in the near future as client appliances become more sophisticated.

In the previous chapter, modulation has been specified and explained in detail. Modulation works exclusively on scalable multimedia data-formats, such as progressive-and-hierarchical JPEG, JPEG 2000, MPEG-4, and H.264. As revealed in the previous chapter, modulation can offer efficiency to Web content delivery. Short latency offered by scalable multimedia data-formats plus efficiency offered by modulation is the ideal combination for the future Web content delivery.

In this chapter, the JPEG 2000 still image standard is exploited to illustrate modulation. The JPEG 2000 image standard is selected due to several reasons. Firstly, exploiting a still image is simpler than exploiting a video stream, which involves motion compensated prediction and time dimension. Thus, at the moment we leave out the video coding standards (MPEG-4 and H.264), but we may consider them in the future study.

Secondly, the JPEG 2000 standard is based on DWT (Discrete Wavelet Transform), which provides better coding efficiency than DCT (Discrete Cosine Transform) employed by the JPEG standard. In addition, since DWT is applied to the entire image, rather than to blocks of usually 8 x 8 pixels, it does not exhibit the characteristic blocking artefacts found in a DCT-based compression method. Lastly, JPEG 2000 supports several types of scalability, such as quality, resolution, and component scalability. Moreover, exploiting the ROI (region of interest) feature in JPEG 2000, we may display only a particular region of an image. The rich scalability in JPEG 2000 plus its fine granularity gives us a strong reason to select it over other multimedia data-formats.

4.2 JPEG 2000 Modulators

As indicated in the previous chapters, in the JPEG 2000 standard, an image's coded data is arranged into packets. Each packet contains data of a specific quality *layer*, a specific *position* (or, precinct), a specific *resolution*, and a specific color *component*. The layer, position, resolution, and component are thus the attributes (properties) that can differentiate one packet from another. In a codestream, packets of the image are interleaved along the four attributes. The interleaving of the packets follows one of the five progression orders predefined in the JPEG 2000 standard.

Packets of a JPEG 2000 image correspond to the atoms of an object in the proposed data model. The four attributes governing the interleaving of the packets can be used as variations to classify the packets and build the imaginary segments. Since the image's entities fit in nicely with the data model's entities, modulation can be applied to

the JPEG 2000 image. Modulation can generate various representations of the image by simply dropping some packets of the image.

4.2.1 General Issues

Modulation in the JPEG 2000 still image standard was done in three types of scalability; those of quality, resolution, and component scalability. They can be associated with the respective layer, resolution, and component – three out of four attributes used to interleave packets. To modulate a JPEG 2000 image correctly, some information (meta-data) about the image must be retrieved. The information is available in the image's main header, where markers and marker segments are held. To be precise, the information we need is stored in the `SIZ` and `COD` marker segments. Their specifications [ISO15444-1] are reproduced below in more concise forms.

Image and tile size (`SIZ`)

Function: Provides information about the uncompressed image such as the width and height of the reference grid, the width and height of the tiles, the number of components, component bit depth, and the separation of component samples with respect to the reference grid.

Usage: Main header. There shall be one and only one in the main header immediately after the `SOC` marker segment. There shall be only one `SIZ` per codestream.

Length: Variable depending on the number of components.

Table 4.1 Image-and-tile-size marker segment

Parameter	Size (bits)	Values	Notes
SIZ	16	0xFF51	Image and tile size marker.
Lsiz	16	41 – 49 190	Length of marker segment in bytes (not including the marker).

Parameter	Size (bits)	Values	Notes
Rsiz	16	0000 0000 0000 0000 (no restrictions) 0000 0000 0000 0001 (Profile-0 compliant) 0000 0000 0000 0010 (Profile-1 compliant)	Capabilities of the codestream.
Xsiz	32	$1 – (2^{32} - 1)$	Width of the reference grid.
Ysiz	32	$1 – (2^{32} - 1)$	Height of the reference grid.
XOsiz	32	$0 – (2^{32} - 2)$	Horizontal offset from the origin of the reference grid to the left side of the image area.
YOsiz	32	$0 – (2^{32} - 2)$	Vertical offset from the origin of the reference grid to the top side of the image area.
XTsiz	32	$1 – (2^{32} - 1)$	Width of one reference tile with respect to the reference grid.
YTsiz	32	$1 – (2^{32} - 1)$	Height of one reference tile with respect to the reference grid.
XTOsiz	32	$0 – (2^{32} - 2)$	Horizontal offset from the origin of the reference grid to the left side of the first tile.
YTOsiz	32	$0 – (2^{32} - 2)$	Vertical offset from the origin of the reference grid to the top side of the first tile.
Csiz	16	1 – 16 384	Number of components (N_{cmp}) in the image.
$Ssiz^i$	8	0000 0000 – 0010 0101 or 1000 0000 – 1010 0101	Precision (depth) in bits and sign of the i^{th} component samples; depth = value + 1. 0xxx xxxx unsigned values 1xxx xxxx signed values
$XRsiz^i$	8	1 – 255	Horizontal separation of a sample of i^{th} component with respect to the reference grid.
$YRsiz^i$	8	1 – 255	Vertical separation of a sample of i^{th} component with respect to the reference grid.

Coding style default (COD)

Function: Describes the coding style, number of decomposition levels, and layering that is the default used for compressing all components of an image (if in the main header) or a tile (if in the tile-part header). The parameter values can be overridden for an individual component by a COC marker segment in either the main or tile-part header.

Usage: Main and first tile-part header of a given tile. It shall be one and only one in the main header. Additionally, there may be at most one for each tile. If there are multiple

tile-parts in a tile, and this marker segment is present, it shall be found only in the first tile-part.

Length: Variable depending on the value of Scod.

Table 4.2 Coding-style-default marker segment

Parameter	Size (bits)	Values	Notes
COD	16	0xFF52	Coding style default marker.
Lcod	16	12 – 45	Length of marker segment in bytes (not including the marker).
Scod	8	(see Table 4.3)	Coding style for all components.
SGcod	32	Defined below	
Progression order	8	(see Table 4.4)	Progression order.
Number of layers	16	1 – 65 535	Number of layers (N_{lyr}).
Multiple component transformation	8	0000 0000 (no MCT used) 0000 0001 (MCT used)	Multiple component transformation usage. If used, irreversible component transformation used with 9-7 irreversible filter and reversible component transformation used with 5-3 reversible filter. [Note: refer to the Transformation field of SPcod]
SPcod	variable	Defined below	
Number of decomposition levels	8	0 – 32	Number of decomposition levels (N_{lvl}); zero implies no transformation. [Note: number of resolutions = N_{lvl} + 1]
Code-block width	8	xxxx 0000 – xxxx 1000	Code-block width exponent offset value; xcb = value + 2.
Code-block height	8	xxxx 0000 – xxxx 1000	Code-block height exponent offset value; ycb = value + 2. (xcy + ycb ≤ 12)
Code-block style	8	(see Table 4.5)	Style of code-block coding passes.
Transformation	8	0000 0000 – 0000 0001	Wavelet transformation used. 0 = 9-7 irreversible filter 1 = 5-3 reversible filter
Precinct size	variable	0000 0000 – 1111 1111	Precinct size (only if Scod = xxxx xxx1). 4 LSBs are the precinct width exponent, PPx = value; and 4 MSBs are the precinct height exponent, PPy = value.

Table 4.3 Coding style parameter values for Scod parameter

Values (bits) MSB LSB	Coding style
xxxx xxx0	Entropy coder, precinct with PPx = 15 and PPy = 15
xxxx xxx1	Entropy coder with precinct defined below
xxxx xx0x	No SOP marker segments used
xxxx xx1x	SOP marker segments may be used
xxxx x0xx	No EPH marker used
xxxx x1xx	EPH marker shall be used
	All other values reserved

Table 4.4 Progression orders for SGcod parameter

Values (bits) MSB LSB	Progression order
0000 0000	Layer – resolution level – component – position progression
0000 0001	Resolution level – layer – component – position progression
0000 0010	Resolution level – position – component – layer progression
0000 0011	Position – component – resolution level – layer progression
0000 0100	Component – position – resolution level – layer progression
	All other values reserved

Table 4.5 Code-block style for SPcod parameter

Values (bits) MSB LSB	Code-block style
xxxx xxx0	No selective arithmetic coding bypass
xxxx xxx1	Selective arithmetic coding bypass
xxxx xx0x	No reset of context probabilities on coding pass boundaries
xxxx xx1x	Reset context probabilities on coding pass boundaries
xxxx x0xx	No termination on each coding pass
xxxx x1xx	Termination on each coding pass
xxxx 0xxx	No vertically causal context
xxxx 1xxx	Vertically causal context
xxx0 xxxx	No predictable termination
xxx1 xxxx	Predictable termination
xx0x xxxx	No segmentation symbols are used
xx1x xxxx	Segmentation symbols are used
	All other values reserved

When a representation of the image is constructed, besides dropping some packets of the image, some parameter values in the marker segments may need to be modified so that they describe the resulting image correctly. Thereby, common rendering applications can properly display the resulting image. This is also in agreement with one of the rules[16] specified in the JPEG 2000 standard (Annex A, Subsection A.1.4 [ISO15444-1]). For instance, to construct a representation with one-quarter resolution (half width and half height) of the original image, packets associated with the highest decomposition level are dropped from the image; along with the action, the width and height of the resulting image (in the `SIZ` marker segment) as well as the number of decomposition levels (in the `COD` marker segment) are modified. Detailed modifications on the image's header will be explained later when we discuss the modulators.

In order to improve a representation, up to the original image, a supplement containing some missing packets is required. The representation and the suitable supplement are then joined together to yield an improved representation. A supplement should also include some meta-data (information) to help the joining operation. There are two things that the meta-data can help. Firstly, the meta-data can indicate which marker segments of the current representation shall be replaced to reflect the improved representation's characteristics. Note again that if a JPEG 2000 image is modified, the marker segments shall be updated accordingly. Secondly, the meta-data can guide how to insert the supplement's packets into the representation. Correspondingly, we introduce

---

[16] The marker segments shall correctly describe the image as represented by the codestream. If truncation, alteration, or editing of the codestream has been performed, the marker segments shall be updated accordingly.

two marker segments, namely RPL and FLG, to be used by a supplement for carrying the meta-data. The proposed marker segments are specified below.

Replacement indicator (RPL)

Function: Indicates marker segments – in this codestream – which shall replace the respective ones in the resulting (joined) codestream. It just contains a list of replacement markers; the indicated marker segments shall be found in the latter part of the header.

Usage: It is strictly used in a supplement's header. Optional in the main and tile-part headers. If it is present in the main header, there shall be only one immediately after the SOC marker segment (note: it shall precede the SIZ marker segment, as well). Likewise, if it is present in a tile-part header, there shall be only one immediately after the SOT marker segment.

Length: Variable depending on the number of replacement markers.

Table 4.6 Replacement-indicator marker segment

Parameter	Size (bits)	Values	Notes
RPL	16	0xFF80	Replacement indicator marker.
Lrpl	16	4 – 65 534	Length of marker segment in bytes (not including the marker).
$Mrpl^i$	16	0xFFXX	The i^{th} replacement marker. The first byte is always 0xFF (a marker indicator)

Insertion flag (FLG)

Function: Guides the insertion of packets – in this codestream – into the resulting (joined) codestream.

Usage: It is strictly used in a supplement's header. There shall be one and only one in a tile-part header.

Length: Variable depending on the number of packets in the resulting tile-part.

Table 4.7 Insertion-flag marker segment

Parameter	Size (bits)	Values	Notes
FLG	16	0xFF81	Insertion flag marker.
Lflg	16	4 – 65 535	Length of marker segment in bytes (not including the marker).
LBflg	8	1 – 8	Number of real bits in the last data-byte of Fflg.
Fflg	variable	N/A	A sequence of flag bits. The number of flag bits (Nbits) corresponds to the number of packets in the resulting tile-part. Nbits = (Lflg − 4) × 8 + LBflg bit 0 = a packet from the representation to be improved (enhanced) bit 1 = a packet from this supplement

As mentioned earlier, packets in the JPEG 2000 codestream are interleaved along layer, position, resolution, and component attributes. The "Progression order" field of the SGcod parameter (see Table 4.2) determines the interleaving of the packets. The data-sizes of the packets are variable. If the `SOP` marker segments (see Table 4.3) are not used, the boundaries between two adjacent packets are undetected. Nevertheless, each packet has a packet header which can tell, among others, the size of the packet body. Yet, the packet header has to be decoded to get the information. To sum up, in the modulators, iterations based on the progression order's attributes are done to determine which layer, which position, which resolution, and which component each packet contributes to. Once a packet's contribution is determined, a decision can be made whether to keep or drop the packet. The packet header is then decoded to get the size of the packet body, and thereby, the whole packet can be correctly determined. Although the modulators involve decoding the packet headers, the process just comprises some addition and bit-shifting for each

traversed code-block; moreover, the decoding process is passed if there are no data carried by the code-block.

Based on the adapting (modulating) operations specified in the previous chapter, three JPEG 2000 modulators were devised; those are `JP2Selector`, `JP2Joiner`, and `JP2Converter`. Each modulator is detailed in the following subsections.

4.2.2 Modulator 1: JP2Selector

`JP2Selector` is used to generate both representations and supplements of a JPEG 2000 image. So, it emulates the selection, inverse-selection, and – to a certain extent – translation operations (referring to the data model in Section 3.3). Since `JP2Selector` can produce two different types of results – i.e., representations and supplements – the processes within it are explained along the results' types.

A. Generating Representations

Of a JPEG 2000 image, various representations may be generated. As mentioned earlier, there are three attributes (variations) that can be exploited in generating representations. In general, a representation is generated by dropping some packets corresponding to particular layers, particular decomposition levels (resolutions), and/or particular components. [Note: With respect to the JPEG 2000 standard, we may use the terms "resolution" and "decomposition level" interchangeably. "Resolution" is more commonly used; "decomposition level" is often used when we refer to the number of decomposition levels (N_{lvl}) in the image. The number of resolutions = the number of decomposition levels + 1; see Table 4.2.]

Dropping some layers of the image gives a lower-quality representation. The layers to be dropped must be in sequence starting from the highest layer. If the number of layers in the image is reduced by l, the associated action is the dropping of packets corresponding to the l highest layers.

Dropping some decomposition levels of the image gives a smaller-resolution representation. Similar to the above condition, the decomposition levels to be dropped must be in sequence starting from the highest level. If the number of decomposition levels in the image is reduced by r, the associated action is the dropping of packets corresponding to the r highest levels.

Dropping some components of the image gives a less-color representation. Any components may be dropped at will. If component c is removed, the associated action is the dropping of packets corresponding to the c^{th} component. Most images have three color components: one luminance (luma) and two chrominance (chroma)[17]. However, there is a condition attached to the component removal in the JPEG 2000 standard that it requires the CPRL (component–position–resolution–layer) progression order to be employed; otherwise, the resulting image cannot be displayed. Here is the plausible explanation. Our preliminary study revealed that the number of components (N_{cmp}) in the image (the Csiz parameter of Table 4.1) cannot be changed without re-coding since that information has been integrated into the image's coded data. Therefore, the dropping of components

---

[17] In the frequency domain, an image can be separated into luminance and chrominance components. The luminance (symbolized by Y) component determines the brightness/darkness (grayscale) of the image, while the chrominance (Cb and Cr) components carry the color (saturation and hue) of the image. The human eye is less sensitive to chrominance than to luminance. Thus, we can afford to lose a lot more information in the chrominance components than we can in the luminance component.

without changing the number of components may cause inconsistency in fetching the image's packets. For instance, suppose the LRCP progression order is employed for the component removal and at this point the number of components is inconsistent; if the iteration of the component attribute (in LRCP, the third iteration after the iterations of the layer and resolution attributes) goes beyond the actual number of components, the next fetched packet will fall in the wrong attribute values. The problem does not occur when the CPRL progression order is employed, because there is no packet after the actual number of components is surpassed (notice that, in CPRL, the component attribute is the outer-most iteration). `JP2Converter` (described in Subsection 4.2.4) is used to change the progression order of a JPEG 2000 image.

In generating a representation, the three attributes – layer, resolution, and component – can be exploited concurrently, so more varieties of representations can be obtained from the combination of the three attributes. This is a major advantage gained from the JPEG 2000 data format. In theory (i.e., in the data model), those three attributes signify distinct variations of the object decomposition. The generation of a representation, involving modifications on scales of different variations, requires several selection and translation operations. But in practice (in this case), those several operations can be performed in a single execution. By the iterations of progression order's attributes, the image's packets can be easily identified. Hence, the dropping of packets – corresponding to different attributes – can be executed in one scan.

Corresponding to the three exploited attributes, there are three input parameters involved in generating a representation. The input parameters – namely R_{lyr}, R_{res}, and R_{cmp} – are specified in Table 4.8. Constants N_{lyr} and N_{lvl} are respectively the number of layers

and the number of decomposition levels in the input image; both values can be obtained from the image's `COD` marker segment (refer to Table 4.2). Constant N_{cmp} is the number of components in the input image; it can be obtained from the image's `SIZ` marker segment (refer to Table 4.1). A specific example of using parameter R_{cmp}: if components 1 and 2 (chroma) are to be removed, then R_{cmp} = 6 (= $2^1 + 2^2$); component 0 (luma) will be retained, and the result should be a grayscale image.

Table 4.8 Input parameters for generating a representation or a supplement

Input	Value Range	Notes
R_{lyr}	$\{\forall R_{lyr} : \mathbb{N} \bullet R_{lyr} < N_{lyr}\}$	Number of layers to be reduced. If it is not provided, the default is $R_{lyr} = 0$ (all layers retained in the representation).
R_{res}	$\{\forall R_{res} : \mathbb{N} \bullet R_{res} \leq N_{lvl}\}$	Number of resolutions to be reduced. If it is not provided, the default is $R_{res} = 0$ (all resolutions retained in the representation).
R_{cmp}	$\{\forall R_{cmp} : \mathbb{N} \bullet R_{cmp} < 2^{N_{cmp}}\}$	Flag bits of components (starting from LSB) to be removed. If it is not provided, the default is $R_{cmp} = 0$ (all components retained in the representation).

The following pseudo-code snippet shows how `JP2Selector` determines packets for generating a representation.

```
/*********************************************************/
/* JP2Selector: Determining packets for a representation  */
/*********************************************************/
// Input parameters:
// FD_img => the image's file descriptor
// N_lyr  => number of layers (from the image's COD)
// N_lvl  => number of decomposition levels (COD)
// N_cmp  => number of components (SIZ)
// R_lyr  => number of reduced layers
//           [default: R_lyr = 0]
// R_res  => number of reduced resolutions
//           [default: R_res = 0]
// R_cmp  => flag-bits of removed components
//           [default: R_cmp = 0]
```

```
// S_lyr denotes the reference number of layers
S_lyr = N_lyr - R_lyr;

// S_res denotes the reference number of resolutions
S_res = N_lvl - R_res + 1;

// S_cmp denotes the flag-bits of retained components
S_cmp = (1 << N_cmp) - R_cmp - 1;

WHILE (NOT End_Of_File(FD_img)) {
    // Packet denotes an image's packet
    Packet = Read_Packet(FD_img);
    IF ((Packet.lyr < S_lyr) AND
        (Packet.res < S_res) AND
        ((1 << Packet.cmp) & S_cmp)) {
            // Output packets of the representation
            PRINT Packet;
    }
}
```

The code basically selects packets, the layer's identity of which is less than the reference number of layers (referring to the reference scale in Subsection 3.3.5), the resolution's identity of which is less than the reference number of resolutions, and the component's identity of which is in the retained components. Besides the packets, the resulting representation also inherits the image's header but with some modifications, particularly on the `SIZ` and `COD` marker segments. The modifications are necessary for the representation to be properly displayed. For each affected attribute, the necessary modifications are given in Table 4.9.

A representation may be generated from another higher-fidelity representation. In that case, no special treatment is required. The higher-fidelity representation is just considered "the original image", and all the procedures can be applied without any alteration. In fact, the above modifications on the image's header keep all information consistent, so successive generation of representations will have no problem.

Table 4.9 Header modifications in a generated representation

Attribute	SIZ Marker Segment	COD Marker Segment
Layer	No modification required.	"Number of layers" in SGcod is reduced. $N_{lyr}^{REP} = N_{lyr} - R_{lyr}$
Resolution	Xsiz, Ysiz, XOsiz, YOsiz, XTsiz, YTsiz, XTOsiz, and YTOsiz are corrected.§ $X^*siz^{REP} = \lceil X^*siz / 2^{R_{res}} \rceil$ $Y^*siz^{REP} = \lceil Y^*siz / 2^{R_{res}} \rceil$	"Number of decomposition levels" in SPcod is reduced. $N_{lvl}^{REP} = N_{lvl} - R_{res}$ Unused (the last R_{res}) precinct sizes in SPcod, if defined, are discarded.
Component	No modification required.	No modification required.

Note: § In the formula, Xsiz, XOsiz, XTsiz, and XTOsiz are symbolized by X*siz, whereas Ysiz, YOsiz, YTsiz, and YTOsiz are symbolized by Y*siz.

B. <u>Generating Supplements</u>

Generating supplements of a JPEG 2000 image is very similar to generating representations. `JP2Selector` also needs to drop some packets corresponding to particular layers, particular decomposition levels, and/or particular components to generate a supplement. However, a supplement usually does not include packets of the lowest layer or the lowest decomposition level, therefore, it cannot be displayed. Furthermore, many marker segments of the image are stripped off in a supplement. A supplement is only useful for enhancing a certain representation.

Like the process in generating a representation, the three attributes – layer, resolution, and component – can be exploited concurrently when generating a supplement. So, the supplement's generation can be executed in one scan, too. The input parameters specified in Table 4.8 are also used for generating a supplement. Besides, four more input parameters are added; the additional parameters are specified in Table 4.10. The most important parameter is *Inv*, which determines whether a representation or a

supplement is to be generated. The other input parameters – namely T_{lyr}, T_{res}, and T_{cmp} – are non-zero and meaningful only if a supplement is to be generated ($Inv = 1$). Notice also that additional conditions take effect when the parameters are used. The conditions regulate the relation between the reduction (prior) and the target (additional) parameters.

Table 4.10 Additional input parameters for generating a supplement

Input	Value Range	Notes
Inv	$0 \mid 1$	0 = generating a representation (selection) 1 = generating a supplement (inverse-selection)
T_{lyr}	$\{\forall T_{lyr} : \mathbb{N}_1 \bullet T_{lyr} \le N_{lyr}\}$	Target number of layers (only if $Inv = 1$). If it is not provided, the default is $T_{lyr} = N_{lyr}$. Additional condition: $R_{lyr} < T_{lyr}$
T_{res}	$\{\forall T_{res} : \mathbb{N}_1 \bullet T_{res} \le N_{lvl} + 1\}$	Target number of resolutions (only if $Inv = 1$). If it is not provided, the default is $T_{res} = N_{lvl} + 1$. Additional condition: $R_{res} < T_{res}$
T_{cmp}	$\{\forall T_{cmp} : \mathbb{N}_1 \bullet T_{cmp} < 2^{N_{cmp}}\}$	Target flag bits of components (only if $Inv = 1$). If it is not provided, the default is $T_{cmp} = 2^{N_{cmp}} - 1$. Additional condition: $(R_{cmp} \mid T_{cmp}) = T_{cmp}$

The pseudo-code snippet below is an extension to the previous `JP2Selector`'s code. Besides determining packets for generating a representation, the code now accommodates the supplement's generation.

```
/*********************************************************/
/* JP2Selector: Determining packets for a representation  */
/*              or a supplement                          */
/*********************************************************/
// Input parameters:
// FD_img => the image's file descriptor
// N_lyr  => number of layers (from the image's COD)
// N_lvl  => number of decomposition levels (COD)
// N_cmp  => number of components (SIZ)
// R_lyr  => number of reduced layers
//           [default: R_lyr = 0]
// R_res  => number of reduced resolutions
//           [default: R_res = 0]
// R_cmp  => flag-bits of removed components
//           [default: R_cmp = 0]
```

```
// Inv    => 0 - selection; 1 - inverse-selection
// T_lyr  => target number of layers
//           [default: T_lyr = N_lyr]
// T_res  => target number of resolutions
//           [default: T_res = N_lvl + 1]
// T_cmp  => target flag-bits of components
//           [default: T_cmp = (1 << N_cmp) - 1]

// S_lyr denotes the reference number of layers
S_lyr = T_lyr - R_lyr;

// S_res denotes the reference number of resolutions
S_res = T_res - R_res;

// S_cmp denotes the reference flag-bits of components
S_cmp = T_cmp - R_cmp;

// Re-set values of T_lyr, T_res, and T_cmp, if (Inv == 0)
// [This is to optimize the selection operation]
IF (NOT Inv) {
    T_lyr = S_lyr;
    T_res = S_res;
    T_cmp = S_cmp;
}

WHILE (NOT End_Of_File(FD_img)) {
    // Packet denotes an image's packet
    Packet = Read_Packet(FD_img);
    IF ((Packet.lyr < T_lyr) AND
        (Packet.res < T_res) AND
        ((1 << Packet.cmp) & T_cmp)) {
        IF (NOT Inv) {
            // Output packets of the representation
            PRINT Packet;
        } ELSE {
            // Flag "0" denotes a representation's packet
            Flag = 0;
            IF ((Packet.lyr >= S_lyr) OR
                (Packet.res >= S_res) OR
                NOT ((1 << Packet.cmp) & S_cmp)) {
                    // Flag "1" denotes a supplement's packet
                    Flag = 1;
                    // Output packets of the supplement
                    PRINT Packet;
            }
            // This function accumulates all flag-bits
            Accumulate_Flags(Flag);
        }
    }
}
```

When generating a supplement, the code filters the packets in two steps. The outer filter (condition) selects packets, which are within the target number of layers, the target number of resolutions, and the target components (referring to the target scale in Subsection 3.3.6). The packets construct the targeted representation that will be obtained if the supplement is joined with the reference representation. The inner filter selects packets, which are missing from the reference representation; those packets construct the supplement. This filter is also applied to the layer, resolution, and component attributes. Based on the attributes involved in generating the targeted representation, the modified `SIZ` and/or `COD` marker segments are created (following the directions in Table 4.9) and included in the supplement's header. Later the marker segments will replace ones belonging to the reference representation when the targeted representation is generated. Information about the modified marker segments are noted in the supplement's `RPL` marker segment (refer to Table 4.6). Lastly, the accumulated flag-bits are stored in the supplement's `FLG` marker segment (refer to Table 4.7). The flag-bits will help in joining packets for generating the targeted representation.

A supplement may be generated from a representation, too. Again, no special treatment is required. However, a supplement cannot be generated from another supplement. The reason is because much information – i.e., marker segments – has been stripped off in the supplement, so inconsistency may occur if a successive supplement is to be generated.

The representation's generation and the supplement's generation are similar in procedure but different in result. Both processes select certain packets of a JPEG 2000 image and drop the rest to get the sought results. But the selected packets are

contradictory between the two processes. While the former selects packets which construct a representation, the latter select packets which do not construct but can enhance a representation. Owing to their similarity, we can combine the two processes in a single modulator, which is the `JP2Selector`. As seen in the pseudo-code, the processes are quite simple. No complex computations are required. Therefore, fast results are very much expected.

4.2.3 Modulator 2: JP2Joiner

As described by its name, `JP2Joiner` is used to join a JPEG 2000 representation and its suitable supplement to construct an improved (enhanced) representation, with respect to quality, resolution, and/or color. Clearly it emulates the join operation of the data model (refer to Subsection 3.3.7). `JP2Joiner` works in tandem with `JP2Selector`, in particular to construct an enhanced representation. `JP2Joiner` does not need any input parameter besides the representation to be enhanced and its suitable supplement.

The pseudo-code snippet below demonstrates how `JP2Joiner` joins packets from the representation and the supplement to construct the enhanced representation.

```
/*********************************************************/
/* JP2Joiner: Joining packets to create a representation  */
/*********************************************************/
// Input parameters:
// FD_rep => the representation's file descriptor
// FD_sup => the supplement's file descriptor
// F_bits => sequence of flag-bits (from the supplement's FLG)

// Looping until the end of bit-sequence F_bits
WHILE (NOT End_Of_Sequence(F_bits)) {
    // Flag denotes a flag-bit
    Flag = Get_Bit(F_bits);
```

```
        IF (Flag == 0) {
            // Get a packet from the representation
            Packet = Read_Packet(FD_rep);
        } ELSE {
            // Get a packet from the supplement
            Packet = Read_Packet(FD_sup);
        }
        // Output packets of the enhanced representation
        PRINT Packet;
    }
```

The code reads the sequence of flag-bits – obtained from the supplement's `FLG` marker segment – one bit at a time. If the flag is zero, a packet from the representation is fetched; otherwise, a packet from the supplement is fetched. In this way, the representation's and the supplement's packets are perfectly blended. The resulting sequence of packets constructs a new representation, i.e., an enhanced one. The enhanced representation inherits the old representation's header but some marker segments may be replaced. The replacement of the marker segments is directed by the supplement's `RPL` marker segment.

The simplicity of modulating operations is shown again here. `JP2Joiner`'s process is even much simpler than `JP2Selector`'s. By employing `JP2Selector` and `JP2Joiner`, any representation can be improved to any level, even to its original image. And more importantly, it is lossless; no information loss occurs in the process. This verifies the reversible property of modulation.

4.2.4 Modulator 3: JP2Converter

`JP2Converter` is used to change the progression order of a JPEG 2000 image (or its representation). It neither drops nor adds the representation's packets, but merely rearranges the packets according to the sought progression order. There is a strong

resemblance between the `JP2Converter`'s process and the translation operation in the data model (refer to Subsection 3.3.8). However, there is also a marked difference between the two. It is stated before that the translation operation is never employed without another adapting operation, but here `JP2Converter` is a stand-alone application. The reasons of creating `JP2Converter` are given in the following paragraphs.

A change of progression order can have an effect on the image's progressive display. For an instance, if the LRCP (layer–resolution–component–position) progression order is employed, the layer (L) attribute will be dominant since it is the outer-most attribute in the progression order's iterations. Hence, the progressive display of the image will go from blurred to clear presentation; it exhibits the quality (SNR) scalability. On the other hand, if the RLCP (resolution–layer–component–position) progression order is employed, the resolution (R) attribute will be dominant and, therefore, the progressive display of the image will go from coarse (due to its low resolution) to fine presentation; it exhibits the resolution (spatial) scalability. So, the first reason of creating `JP2Converter` is to give users the option to choose the preferable progressive display of a JPEG 2000 image.

In the JPEG 2000 standard, generating representations and supplements hardly requires an explicit translation application since `JP2Selector` can perform the translation operation implicitly. An exception is if the component attribute is exploited. In that case, the CPRL (component–position–resolution–layer) progression order must be employed. Changing progression order involves processes – as we will see shortly – that are completely different from the `JP2Selector`'s processes. Therefore, it would be

better to build a separate application to change the progression order. This is the second reason of creating `JP2Converter`.

Besides the image's codestream, `JP2Converter` needs just one input parameter; that is, the new (targeted) progression order. The following pseudo-code snippet outlines its process.

```
/*********************************************************/
/* JP2Converter: Re-arranging packets in different       */
/*               progression order                       */
/*********************************************************/
// Input parameters:
// FD_img => the image's file descriptor
// FD_tmp => temporary file descriptor
// PO_new => new progression order (PO)

WHILE (NOT End_Of_File(FD_img)) {
    // Packet denotes an image's packet
    Packet = Read_Packet(FD_img);

    // With respect to the packet's attributes, note its size
    // and the position of the temporary FD's pointer
    Note_Size_And_Position(Packet, FD_tmp);

    // Output packets to temporary file descriptor
    Print_To_File(Packet, FD_tmp);
}

// Initialize variables for the new progression order
Initialize_New_PO(PO_new);

// Iterate the PO_new's attributes
WHILE (Next_Iteration(PO_new)) {
    // With respect to the PO_new's attributes,
    // get the associated packet's size and position
    Size = Get_Packet_Size(PO_new);
    Position = Get_Packet_Position(PO_new);

    // Fetch the packet from temporary file descriptor
    Packet = Read_Packet_From_File(Size, Position, FD_tmp);

    // Output packets in new PO
    PRINT Packet;
}
```

There are two iterations observed in the above code. The first iteration reads all of the image's packets and stores them to a temporary file. For each of the packets, the iteration also takes notes of its size and its position in the temporary file and stores the information in the memory. The sequence of packets in the first iteration is based on the old progression order, stated in the `COD` marker segment (see Table 4.2). Employing the new progression order, the second iteration determines one associated packet at a time, retrieves its information (size and position) from the memory, fetches it from the temporary file based on the information, and promptly outputs it. In addition to rearranging the packets, `JP2Converter` also rectifies the progression order in the `COD` marker segment with the new one. `JP2Converter` can also be applied to any image's representation; however, it cannot change the progression order of an image's supplement.

4.3 Related Work

Since the JPEG 2000 standard is a relatively new standard for image compression, there are only a few JPEG 2000 applications available in the public domain. Commonly, the applications deal with the creation or conversion of an image from one standard (out of a range of standards) to the JPEG 2000 standard, and vice versa. Hence, the main JPEG 2000 applications are the encoder (compressor) and decoder (decompressor), also known as the JPEG 2000 codec. Raw, uncompressed image standards, such as PPM (Portable Pixel Map) and PGM (Portable Grey Map), are often used as intermediaries in

the image conversion. Instances of such applications can be found in JasPer[18], JJ2000[19], and Kakadu[20]. They are software packages, each of which contains a collection of applications mainly related to the JPEG 2000 standard. JasPer and JJ2000 are the formal JPEG 2000 reference software, specified in Part 5 of the JPEG 2000 standard [ISO15444-5]. The former is written in C, and the latter in Java. They are both available under open-source type licensing, and their source code can be retrieved from their respective Websites. Kakadu, on the other hand, is proprietary freeware, and therefore a license is required to access its source code.

Alas, the collections of applications mentioned above do not support packet-based processing, which can selectively retrieve certain packets of a JPEG 2000 image. With respect to modulation, the packet-based processing is required to generate the JPEG 2000 image's representations in a fast and efficient way. Actually, Kakadu includes an application, called `kdu_transcode`, that can generate different image's representations in quality, resolution, and component aspects. However, the application seems to combine and rearrange data in the image's packets, particularly when it reduces the image's quality. Such kind of adaptation obviously takes more processing time. Our preliminary study found that, compared to `kdu_transcode`, our modulator (`JP2Selector`) can achieve 83–92% processing-time improvement in quality aspect, 44–70% in resolution aspect, and 15–35% (64–71%, if CPRL is used as the progression order) in component aspect.

---

[18] http://www.ece.uvic.ca/~mdadams/jasper

[19] http://jpeg2000.epfl.ch

[20] http://www.kakadusoftware.com

Nevertheless, the major benefit of our JPEG 2000 modulators is the ability to improve an image's representation. This also demonstrates the reversible property of modulation. To improve the representation, the targeted representation is determined, a suitable supplement containing the missing image's packets is then generated, and eventually the entire packets of the representation and supplement are joined together to construct the targeted representation. In the context of Web delivery, this feature may reduce the bandwidth consumption and improve the client perceived latency.

This chapter has described the development of the JPEG 2000 modulators, which closely follow the concepts and operations specified in the fine-grained, scalable data model (in Chapter 3). The three resulting modulators are `JP2Selector` (used to generate various representations and supplements of a JPEG 2000 image), `JP2Joiner` (used to construct an improved representation by joining the prior representation and its suitable supplement), and `JP2Converter` (used to change the progression order of a JPEG 2000 image or its representation). In the next chapter, modulation (the fast and scalable adaptation) is evaluated in comparison with transcoding (the traditional adaptation). The JPEG 2000 modulators developed in this chapter are employed in the evaluation, along with some JPEG transcoders.

Chapter 5

Evaluation: Modulation vs. Transcoding

Modulation is characterized by fast adaptation, thanks to the simplicity of its process. To verify this, in this chapter, modulation is compared and contrasted with transcoding, the oft-cited multimedia adaptation. Modulation in the JPEG 2000 still image standard is performed on one side, while transcoding in the JPEG image standard is performed on the other. The results beyond doubt substantiate the superiority of modulation.

5.1 Rationale of Using Two Different Image Standards

JPEG is comparable to JPEG 2000 in several aspects. Firstly, JPEG is by far the commonly used standard in the Web to present natural images, and JPEG 2000 has recently put a strong challenge to JPEG. Secondly, both standards support lossy compression that reduces not only the image's data-size but its quality as well. The quality reduction in both standards can be adjusted easily, so they are suitable for differentiated services in the Web. Chandra and Ellis [ChE99] utilized the quality value to quantify the loss of information in transcoding a JPEG image. We will see shortly how the quality reduction is done in JPEG 2000. Last but not least, both standards are royalty and license-fee free – but not patent-free – so their proliferating use can be assured.

In the previous chapter, we have mentioned the reasons of implementing our modulators in JPEG 2000. One of the reasons is JPEG 2000's support of multiple types of scalability. JPEG also supports scalability in quality and resolution, but its progressive and hierarchical formats lack public support. We could not get the source code implementing the JPEG hierarchical format; perhaps, because there are some patents associated with it. So, we are more comfortable with modulating JPEG 2000. However, in this evaluation we do not use JPEG 2000 for their transcoding counterparts. We have two reasons to back our decision. Firstly, DWT (Discrete Wavelet Transform), the compression method employed by JPEG 2000, is a complex process, much more complex than DCT (Discrete Cosine Transform) which is employed by JPEG. Our preliminary study shows that transcoding in JPEG 2000 is 7–8 times more delaying than that in JPEG; in the study, transcoding JPEG images took between 0.5 and 2 seconds, whereas

transcoding JPEG 2000 images took 4 to 16 seconds. Secondly, whatever image standard is used will not change much the evaluation's results here. What we want to evaluate is mainly the contrasting techniques used in transcoding and modulation. While modulation just drops parts of the image data with minimal computation, transcoding involves complex computations like encoding and decoding, quantization and de-quantization, compression and decompression, and the reduction of quality and/or resolution applied to the raw image. In the end, we decided to use the JPEG 2000 image standard for modulation and the JPEG image standard for transcoding.

5.2 Experimental Setup

In this first section, the adaptors (modulators and transcoders) used in the experiments are discussed. After that, the creation of image test data is described.

5.2.1 Experimented Adaptors

The JPEG 2000 modulators – the development of which has been discussed in the previous chapter – are employed for modulation. `JP2Selector` is the main application to generate representations of a JPEG 2000 image. `JP2Joiner` is a supporting application used to construct a high-fidelity representation from a low-fidelity one. `JP2Converter` is used, only if necessary, to change the image's progression order.

As for the counterpart, two different JPEG transcoding methods are employed. The particulars of the two methods are as follows:

1. Spatial-Domain Transcoding (SDT) – using the `djpeg` and `cjpeg` applications of the Independent JPEG Group (IJG) library[21] release 6b. The `djpeg` application is used to decompress a JPEG image to a non-compressed image; in this case, to a PNM (Portable aNyMap) image. Conversely, the `cjpeg` application is used to compress a non-compressed image to a JPEG image.
2. Frequency-Domain Transcoding (FDT) – using our developed application called `jpegfdt`. The core of this application is two transcoding modules. One is a module[22], devised by Surendar Chandra, which changes the JPEG compression metric (determining the image's quality) in the frequency domain. The other is a module which scales down the image's resolution also in the frequency domain; the module is based on the approximate algorithm proposed by Natarajan and Vasudev [NaV95].

The former method operates in the spatial domain; it is a naive approach that decompresses the image into the spatial domain, and later, compresses the spatial image data back to the JPEG format after/with adaptation. The latter method operates in the compressed (frequency) domain directly; the image is entropy decoded and de-quantized to obtain its coefficients, the equivalent frequency-domain adaptation is then applied to the coefficients, and finally, the modified coefficients are quantized and entropy encoded.

Supported adapting processes in the adaptors are reductions in quality, resolution, and (color) component aspects. Not only can the adaptors execute adaptation in a particular aspect at a time, but they too can execute it in multiple aspects simultaneously.

---

[21] http://www.ijg.org

[22] Source code: http://www.cse.nd.edu/~csesys/qat/source/transcode.c

5.2.2 Image Test Data

Boat (actual res. 1976×2960)

Hawaii (actual res. 2097×1391)

Venice (actual res. 1055×1568)

Figure 5.1 The reference images for test data

Three images (in Figure 5.1) were employed in the experiments. The source images are lossless, JPEG 2000 images obtained from the CD-ROM accompanying Taubman and Marcellin's book [TaM02]; those are `boat4_2100.jp2` (1976×2960), `hawaii1_1500.jp2` (2097×1391), and `venice1_1500.jp2` (1407×2091). The last two images are quite similar in number of pixels. In the beginning, the images were transformed into PPM (Portable Pixel Map) images using the `jasper` application of the JasPer software package version 1.700.2. The last image (`venice`) was further downsized 25% horizontally and vertically, hence its resolution became 1055×1568. The

resulting images were called `boat.ppm`, `hawaii.ppm`, and `venice.ppm`. They were the reference images for the subsequent image creation and processes.

Three JPEG images (i.e., `boat.jpg`, `hawaii.jpg`, and `venice.jpg`) were created from the reference images using the `cjpeg` application of the above-mentioned IJG library. To create the JPEG images, the *quality* parameter of the application was set to 100 (the highest quality). The data-sizes of the resulting images are about 5 MB (to be exact, 5,065,493 bytes), 2.4 MB (2,455,504 bytes), and 1.2 MB (1,258,420 bytes), respectively.

Three JPEG 2000 images (i.e., `boat.jp2`, `hawaii.jp2`, and `venice.jp2`) were also created from the reference images. This time, the `kdu_compress` application of Kakadu software package version 3.4 was employed. The number of decomposition levels, which determines the resolution (spatial) scalability, and the number of layers, which determines the quality (SNR) scalability, can be specified during the images' creation. The application provides parameters *Clevels* and *Clayers* to specify the number of decomposition levels and the number of layers, respectively. The JPEG 2000 images were created with the respective parameters set to 5 (the default value) and 10. The default progression order, LRCP (layer–resolution–component–position), was not changed. The intended data-size of a JPEG 2000 image can also be specified in the application through parameter *rate*, which is given in bpp (bits per pixel). We tried to make the JPEG 2000 images as comparable in data-size as possible to their JPEG counterparts. Hence, the data-sizes of the previously created JPEG images were used as a reference for governing the JPEG 2000 images' data-sizes. For an instance, the

5,065,493-byte data-size of `boat.jpg` is equal to 6.9284 bpp (= 5065493 bytes × 8 bits/byte ÷ [1976 pixels-width × 2960 pixels-height]). This value was set as the *rate* for creating image `boat.jp2`. The *rate* values for creating images `hawaii.jp2` and `venice.jp2` – 6.7345 and 6.0858 bpp, respectively – were determined in the similar way. Further, the *rate* parameter can be used to control the quality rates – and the associated data-sizes – of the image's presentations having different numbers of layers. This is a simple way to specify the quality reduction in a JPEG 2000 image: every time a layer is removed from the image, the image presentation's data-size is reduced by a certain factor. For image `boat.jp2`, the reduction factor is 2; it means removal of a layer will reduce the image's data-size by half. The initial *rate* value, 6.9284 bpp, is the quality rate of the full presentation (containing 10 layers) of image `boat.jp2`. We then need to specify the quality rate of the lowest presentation (containing one layer only). Its quality rate is 0.01353 bpp (= 6.9284 bpp ÷ 2^9); where 2 is the reduction factor and 9 is the number of layers that can be removed. Thus, we fed the *rate* parameter with the intended highest and lowest quality rates of presentations in image `boat.jp2`, which were 6.2984 and 0.01353 bpp. [Note: the application can automatically determine the rates of other presentations in between.] For images `hawaii.jp2` and `venice.jp2`, the reduction factors are 1.85 and 1.7, respectively. Correspondingly, the *rate* parameter was set with 6.7345 and 0.02653 bpp for image `hawaii.jp2`, and 6.0858 and 0.05132 bpp for image `venice.jp2`. The data-sizes of the respective JPEG 2000 images are 5,065,561 bytes, 2,455,540 bytes, and 1,258,403 bytes.

The JPEG and JPEG 2000 images were the test data used in the following experiments. JPEG images were used as the test data for transcoding, while JPEG 2000 images the test data for modulation. In the subsequent section, the generation of both sets of images' representations by transcoding and modulation, respectively, is explained.

5.3 Generating Image Representations

As described earlier, there are two methods of transcoding a JPEG image. We employed the two methods to generate representations of the JPEG images. Employing the first method (SDT), each image was decompressed by `djpeg` and then re-compressed by `cjpeg` with different *quality* parameter values. We used nine different *quality* values, ranging from 90 to 10 (with a down step of 10), to generate the representations. The second method (FDT) uses the `jpegfdt` application to transcode the images. Similar to the first method, each image was transcoded with nine different *quality* parameter values, from 90 to 10. So, for each of the experimented JPEG images, there were ten representations – including the original image – resulting from each exercised method.

The JPEG images' representations – generated by means of SDT and FDT – and their corresponding data-sizes are listed in Table 5.1. In general, the lower the *quality* value of the representation, the smaller is the representation's data-size. Furthermore, the representations resulting from FDT are smaller (by 15–26%) in data-size than those resulting from SDT.

Table 5.1 Representations of the JPEG images and their data-sizes

Representation	boat.jpg		hawaii.jpg		venice.jpg	
	Data-Size* (SDT)	Data-Size* (FDT)	Data-Size* (SDT)	Data-Size* (FDT)	Data-Size* (SDT)	Data-Size* (FDT)
quality = 100	5,065,493	5,065,493	2,455,504	2,455,504	1,258,420	1,258,420
quality = 90	1,483,904	1,197,587	794,689	673,854	374,434	310,868
quality = 80	958,115	737,940	537,540	438,997	249,615	196,903
quality = 70	746,605	559,725	429,022	343,943	196,693	151,332
quality = 60	614,404	454,139	361,932	285,065	163,776	124,761
quality = 50	530,833	390,703	318,645	248,358	142,838	108,445
quality = 40	453,216	335,622	278,035	214,266	123,573	93,831
quality = 30	376,508	284,239	235,565	179,578	104,395	79,258
quality = 20	292,435	225,800	185,112	139,452	82,315	62,866
quality = 10	202,689	159,873	124,298	93,606	56,633	44,479

Note: * in bytes

`JP2Selector` was employed to generate representations of the JPEG 2000 images. As a matter of fact, the representations have been predetermined during the image's creation; that is, by specifying the number of layers and the quality rates of the resulting presentations. Since the number of layers in the images was ten, there were nine representations we could generate when each image was modulated along quality scalability. Accordingly, `JP2Selector` was applied to the images with different *R_lyr* (number of reduced layers) parameter values, ranging from 1 to 9. In total, for each of the experimented JPEG 2000 images, there were ten representations – including the original image – resulting from this exercise.

The JPEG 2000 images' representations and their corresponding data-sizes are given in Table 5.2. As designed during the creation of the images and as seen in the table, every time a layer was removed from the three JPEG 2000 images, the resulting representations' data-sizes were reduced by approximately 50%, 46%, and 41%.

Table 5.2 Representations of the JPEG 2000 images and their data-sizes

Representation	boat.jp2	hawaii.jp2	venice.jp2
	Data-Size*	Data-Size*	Data-Size*
$R\_lyr = 0$	5,065,561	2,455,540	1,258,403
$R\_lyr = 1$	2,529,546	1,327,471	740,500
$R\_lyr = 2$	1,264,125	717,521	434,745
$R\_lyr = 3$	633,431	388,177	256,639
$R\_lyr = 4$	316,433	210,156	150,657
$R\_lyr = 5$	158,748	113,824	89,009
$R\_lyr = 6$	79,570	61,554	52,603
$R\_lyr = 7$	40,024	33,561	31,001
$R\_lyr = 8$	20,256	18,338	18,435
$R\_lyr = 9$	10,362	10,135	10,993

Note: * in bytes

In the next two subsections, the image quality exhibited by both sets of images – JPEG and JPEG 2000 – is assessed to verify JPEG 2000's superiority over JPEG. Two methods, which involve the test images' representations, are employed for the assessment. The first method measures the bit-rate performance of the test images. The second method visually contrasts the low-quality representations of the test images.

5.3.1 Bit-Rate Performance

JPEG 2000 is claimed to offer better bit-rate performance than other existing image standards [ChSE00], including JPEG. An image standard is superior in bit-rate performance to another standard if it can carry more information than what the other can have within the same amount of data. It also means that the superior standard gives more compressed (smaller) data-size than what the other standard may produce without compromising the image quality. In this sense, the bit-rate performance has a parallel

meaning with the data compression; that is, better bit-rate performance implies better data compression. Both terms may be used interchangeably for the rest of the section. In this subsection, we examine the bit-rate performance of the representations resulting from transcoding (SDT and FDT) the JPEG images and modulating the JPEG 2000 images.

The bit-rate performance of an image is determined by two factors, namely bit-rate and quality. In fact, we have used the image bit-rate when creating the JPEG 2000 images in Subsection 5.2.2 (i.e., parameter *rate*). The bit-rate of an image can be attained by dividing the data-size of the image (in bits) by the number of pixels in the image. The quality of an image, on the other hand, can be attained by measuring the quality difference (PSNR / peak signal to noise ratio) between the assessed image and its reference image. Hence, to measure the image quality, we need the reference images (i.e., `boat.ppm`, `hawaii.ppm`, and `venice.ppm`), which are the origins of all images and representations in the experiments. Note that the image quality measured here is different from the *quality* value specified in generating a JPEG representation. The former results from the actual pixel-by-pixel comparison between the assessed image and the reference image, while the latter is the value used to determine the scale factor applied to the quantization table – the effect of which is quality reduction – in JPEG compression.

For each image representation – of both JPEG and JEPG 2000 images – the image bit-rate and image quality were assessed. The image bit-rate could be determined easily since the data-size of each representation was known (refer to Table 5.1 and Table 5.2) and so was the number of pixels in the representation. In contrast, the `imgcmp` application (also from the JasPer software package version 1.700.2) had to be employed to determine the image quality. The application compared each image representation with the reference

image. Three PSNR values resulted from each assessment: one for the luminance component (Y) and two for the chrominance components (Cb and Cr).

Figure 5.2 shows the results for the luminance component only. The horizontal axis is the image data-size (in bits per pixel), and the vertical axis is the image quality (in decibels). As shown in the figure, images of `venice` have higher quality than images of `hawaii` and `boat`. Also, the JPEG images' representations resulting from FDT have slightly lower quality than those resulting from SDT. If the comparison is applied to every two corresponding representations generated with the same *quality* value (i.e., representations of the same row in Table 5.1), the quality of FDT-based results is lower by 0.5–3.2 dB than that of SDT-based results. However, if the evaluation is based on the normalized data-size, it can be inferred that the quality of FDT-based results is lower by 0.1–0.9 dB. Nevertheless, FDT produces a smaller data-size than SDT.

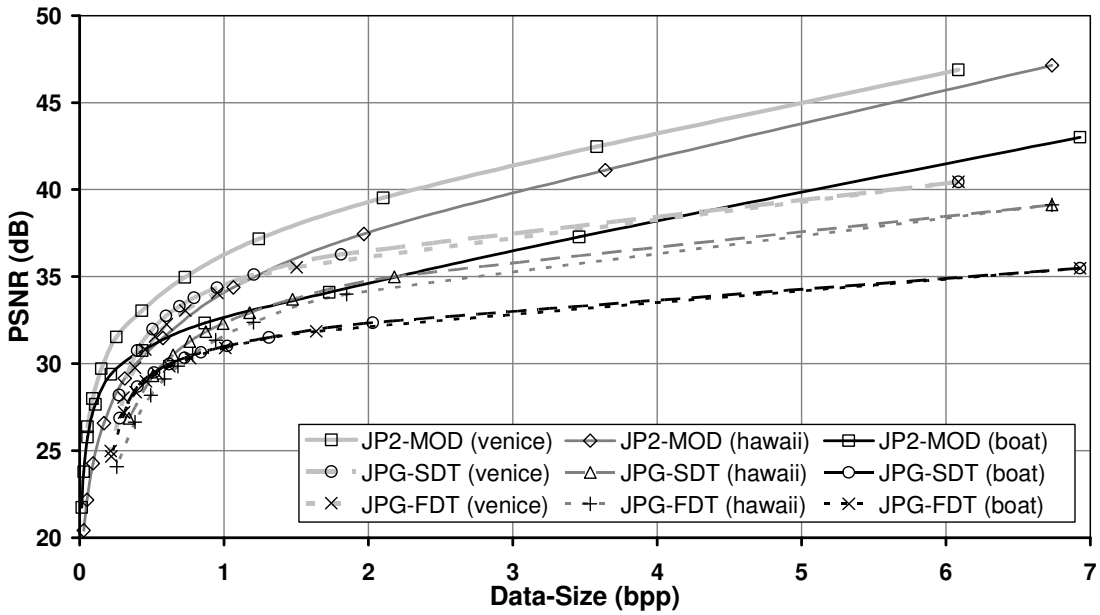
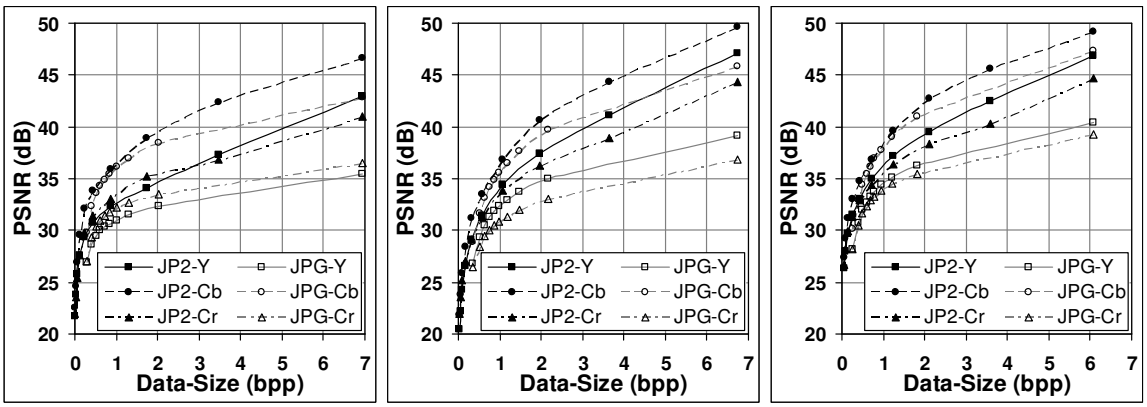


Figure 5.2 Bit-rate performance (luminance only)

Clearly, representations of the JPEG 2000 images have better bit-rate performance than those of the JPEG images. For the same normalized data-size, the quality of JPEG 2000 representations is 1.5–8.0 dB higher than that of JPEG representations. The JPEG 2000 standard is really a remarkable feat of image data compression technique. Further, the bit-rate performance for all color components is detailed for each image in Figure 5.3(a)–(c).



(a) Boat **(b) Hawaii** **(c) Venice**

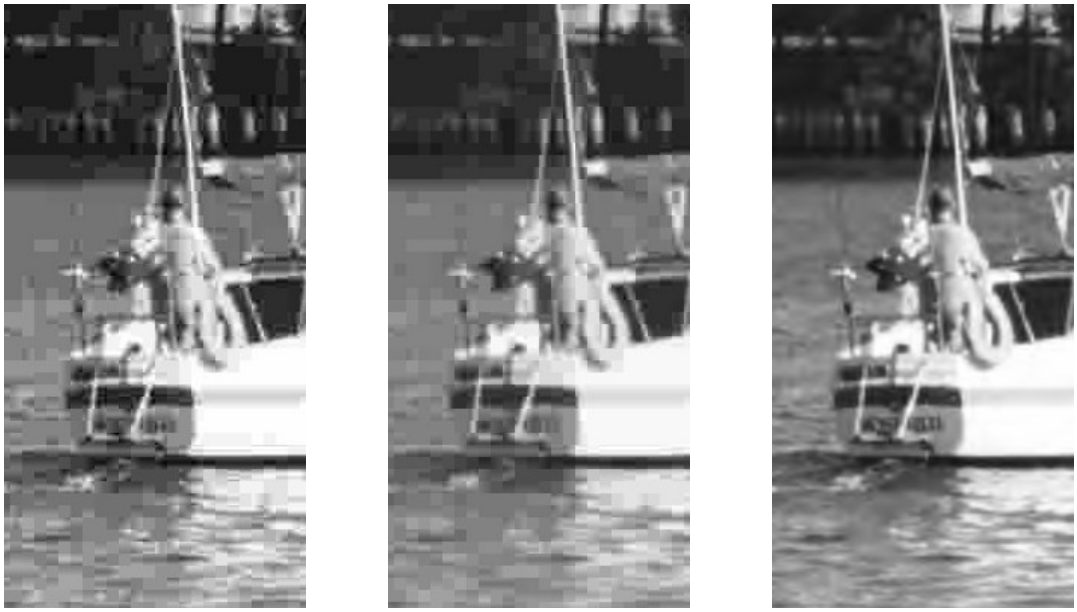
Figure 5.3 Bit-rate performance (all color components)

As seen in the latter figures, curves of the JPEG 2000 representations with respect to the three color components (Y, Cb, and Cr) are all above those of the JPEG representations. The JPEG 2000 luminance component (Y) takes about 35–55% data-size of what the JPEG luminance component requires to acquire certain image quality. Likewise, around 50–70% and 30–50% data-sizes of the JPEG chrominance components (Cb and Cr, respectively) are required by the JPEG 2000 chrominance components to bring in the same image quality.

The results are in agreement with the previous study [ChSE00]. The previous study found that the JPEG 2000 image standard significantly outperforms the JPEG

image standard in bit-rate performance. Likewise, here the images (representations) resulting from modulating the JPEG 2000 images give better quality than those resulting from transcoding the JPEG images. Further, we split each of the three JPEG 2000 images into its components and found that the Y, Cb, and Cr components on average take 49%, 19%, and 32% of the image's data-size. Based on the components' shares and their above-mentioned quality's advantage over JPEG components, we may conclude that to achieve the same image quality, we just need a JPEG 2000 image having at most 60% data-size of its JPEG counterpart; often, less than a half data-size is adequate.

5.3.2 Visual Comparison



(a) JPEG (SDT): 160,107 bytes (b) JPEG (FDT): 159,873 bytes (c) JPEG 2000: 158,748 bytes

Figure 5.4 Representations of `boat.jpg` and `boat.jp2` at 0.22 bpp (partial images)

It can even be visually evaluated in Figure 5.4(a)–(c) that, with a similar data-size (i.e., 0.22 bpp), the modulated JPEG 2000 image offers better image quality than the

transcoded JPEG images. The JPEG image on the left (a) resulted from spatial-domain transcoding image `boat.jpg` with *quality* = 6, whereas the one on the centre (b) resulted from frequency-domain transcoding the same image with *quality* = 10. The JPEG 2000 image (c) resulted from modulating image `boat.jp2` with *R_lyr* = 5 (i.e., five layers removed). For comparison's sake, the resulting data-sizes of the corresponding representations are provided below the images.

To show the details, only a part (200×400 pixels) of each representation is presented in the figures. The partial images are focused on the same region; that is, the farther boat on the left. As seen in the figure, the left and centre images exhibit the characteristic blocking artefacts commonly found in a low-quality DCT-based image. In addition, the centre image seems more blurred than the left image although it resulted from transcoding with a higher quality value. This corroborates the previous finding that, with respect to the same resulting data-size, FDT produces representations with lower quality than those produced by SDT. The right image, the JPEG 2000 representation, does not exhibit the blocking artefacts, and it is obviously the clearest among the three representations. The data compression in the JPEG 2000 standard is indeed more advanced than that employed by the JPEG standard.

The superior data compression in JPEG 2000 is one of the main reasons why it has proliferated rapidly in recent years. Aside from its superior data compression, we can be benefited from its scalability. As we will demonstrate shortly, using modulation's techniques to adapt a JPEG 2000 image, we can get much shorter processing times. Please be mindful, however, that JPEG 2000 is used in the experiments just to illustrate

modulation's work and to reveal modulation's benefits. Other scalable data formats can surely get the same benefits from using modulation.

5.4 Processing Time

The objective of the following experiments is to corroborate our previous claim that modulation is a fast adaptation. The processing time of executing modulation is thus compared to that of executing transcoding. The same JPEG transcoders and JPEG 2000 modulators as mentioned earlier were employed in the experiments. All experiments were executed on a 1.3 GHz Pentium 4 system, with 128 MB of RAM, and Fedora Core 2 Linux is used as the operating system. To obtain the processing time, the standard `time`[23] command of the Linux OS was put in front of every execution of transcoding and modulating instructions. The experiments were conducted along three adapting attributes: quality, resolution, and component. The next subsections detail the experimental results.

5.4.1 Adaptation in Quality Aspect

The two transcoding methods – SDT and FDT – were employed to generate representations of images `boat.jpg`, `hawaii.jpg`, and `venice.jpg` with different quality. For each representation's generation, the processing time was determined. Table 5.3 presents the resulting processing times of transcoding (SDT and FDT) the JPEG images in quality aspect. The results have been averaged from several runs.

---

[23] http://directory.fsf.org/GNU/time.html

Table 5.3 Processing times of transcoding the JPEG images in quality aspect

Representation	boat.jpg		hawaii.jpg		venice.jpg	
	Prc.Time* **(SDT)**	**Prc.Time*** **(FDT)**	**Prc.Time*** **(SDT)**	**Prc.Time*** **(FDT)**	**Prc.Time*** **(SDT)**	**Prc.Time*** **(FDT)**
quality = 90	2,318.1	1,234.0	1,168.3	612.3	630.8	328.0
quality = 80	2,187.2	1,155.1	1,107.2	573.4	594.6	308.3
quality = 70	2,131.9	1,123.6	1,082.4	557.6	582.2	300.4
quality = 60	2,092.0	1,104.3	1,065.7	549.2	571.2	295.8
quality = 50	2,070.8	1,093.0	1,053.7	540.9	565.7	293.2
quality = 40	2,048.4	1,082.4	1,042.5	535.6	561.2	290.3
quality = 30	2,024.7	1,073.5	1,031.8	528.7	556.1	287.7
quality = 20	2,001.1	1,061.7	1,016.7	520.9	548.8	285.1
quality = 10	1,977.5	1,047.6	998.7	513.0	541.0	281.0

Note: * in milliseconds

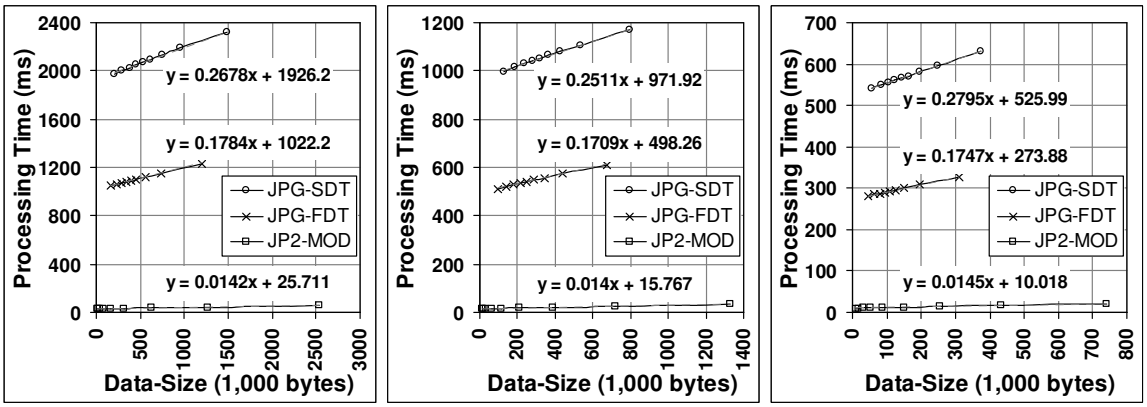
The quality reduction in the JPEG standard is done by changing the quantization tables as well as the image coefficients. In the SDT method, the image coefficients are converted to the spatial domain, and then converted back to the frequency domain with the new quantization tables. In the FDT method, the image coefficients are directly multiplied by the ratio of the new to old quantization tables, so no conversion is required. As a result and as noticed in the table, the processing times of generating FDT-based representations are just about a half of generating their SDT-based counterparts.

Similarly, the JPEG 2000 modulator (`JP2Selector`) was used to generate the `boat.jp2`'s, `hawaii.jp2`'s, and `venice.jp2`'s representations with different reduced layers, and the processing times were recorded as well. In stark contrast to the transcoding results above, modulation took much less time; at least, 97% and 94% faster than the processing times of SDT and FDT, respectively. The complete processing times of modulating the JPEG 2000 images in quality aspect are listed in Table 5.4. Like the previous results, here the processing times have been averaged from several runs, too.

Table 5.4 Processing times of modulating the JPEG 2000 images in quality aspect

Representation	boat.jp2	hawaii.jp2	venice.jp2
	Prc. Time*	Prc. Time*	Prc. Time*
$R\_lyr = 1$	61.6	34.3	20.7
$R\_lyr = 2$	43.5	26.0	16.3
$R\_lyr = 3$	34.8	21.1	14.0
$R\_lyr = 4$	30.1	19.0	12.2
$R\_lyr = 5$	28.0	17.1	11.4
$R\_lyr = 6$	26.9	16.7	11.0
$R\_lyr = 7$	26.2	16.1	10.4
$R\_lyr = 8$	26.0	16.0	10.1
$R\_lyr = 9$	25.9	16.0	10.0

Note: * in milliseconds



(a) Boat **(b) Hawaii** **(c) Venice**

Figure 5.5 Data-size vs. processing time of the three adapting methods in quality aspect

To further contrast the processing times of transcoding and modulation, we put them in the same graphs. For each respective image, both processing times (in Table 5.3 and Table 5.4) are plotted against their corresponding data-sizes (presented earlier in Table 5.1 and Table 5.2, respectively) and the resulting graphs are depicted in Figure 5.5(a)–(c). As shown in the graphs, the processing times of modulation (curves *JP2-MOD*) are well below the processing times of transcoding (curves *JPG-SDT* and *JPG-FDT*). The graphs also show that the representations of the JPEG 2000 images are more

distributed in data-size than those of the JPEG images; actually the distribution could be predetermined when the images were created (i.e., the setting of the *rate* parameter).

Nevertheless, the most interesting finding is that all curves appear almost linear. Therefore, we can assert that the processing times of generating the representations – either by transcoding or modulation – correspond directly with the resulting data-sizes. Due to their linearity, a trend-line can be fitted to the data of each curve. The equations of the trend-lines are displayed next to the respective curves in Figure 5.5(a)–(c). The slope of the trend-line indicates the increase rate of processing time. As an example, for the image of `boat` (a), the increase rate of SDT's processing time is 0.2678 milliseconds/ 1,000 bytes. It means an increase of 1,000 bytes in the resulting data-size corresponds with an increase of 0.2678 milliseconds in the processing time. To reduce the fractional part, the increase rate can be normalized to a better measurement unit. The nanoseconds per byte is used to denote the increase rate; thus, 0.2678 milliseconds/1,000 bytes = 267.8 nanoseconds/byte (ns/byte). In the figures, the increase rates of SDT's processing time range from 250 to 280 ns/byte, whereas the increase rates of FDT's processing time are between 170 and 180 ns/byte. The increase rates of modulation's processing time, by contrast, are just 14.0–14.5 ns/byte, which are 5.4% and 8.2% (on average) of the respective increase rates of SDT's and FDT's processing times.

The trend-lines' equations may also be used to predict the processing time of an adaptation (either by a transcoding or modulating process), if the expected data-size is known. For an instance, the processing times of generating representations of image `boat` with the same data-size of 2,500,000 bytes by SDT, FDT, and modulation are about 2,596, 1,468, and 61 milliseconds, respectively. Prediction of the processing time may be

useful for deciding whether an adaptation is beneficial or for selecting the best-fit (i.e., the most suitable) representation among several alternatives.

Recovering the JPEG 2000 images

Besides generating representations with reduced quality, modulation can be employed to recover the original JPEG 2000 images. Recall that modulation is reversible. Two operations are involved for the recovery. The first operation is to generate a suitable supplement, and the second is to construct the original image by joining the representation and the supplement. `JP2Selector` and `JP2Joiner` were employed for the respective operations. We ran the experiments to recover images `boat.jp2`, `hawaii.jp2`, and `venice.jp2` from their representations, and the processing times are presented in Table 5.5.

As shown in Table 5.5, the processing times of generating the supplements are mostly higher than those of generating the representations (compare the processing times in Table 5.5 column 4 and Table 5.4) since the data-sizes of the supplements are generally bigger than those of the representations (compare also the data-sizes in Table 5.5 column 3 and Table 5.2). The more the number of removed layers in a representation (from the representation at the top of the table to that at the bottom), the smaller is the data-size of the representation, but the bigger is the data-size of the suitable supplement required to recover the original image. This is reasonable and quite self-explanatory. On the other hand, the processing times of constructing the original images from various representations and supplements are quite steady at 37–38 ms for `boat.jp2`, 21 ms for `hawaii.jp2`, and 13 ms for `venice.jp2` (see the last column of Table 5.5).

Table 5.5 Processing times of recovering the JPEG 2000 images in quality aspect

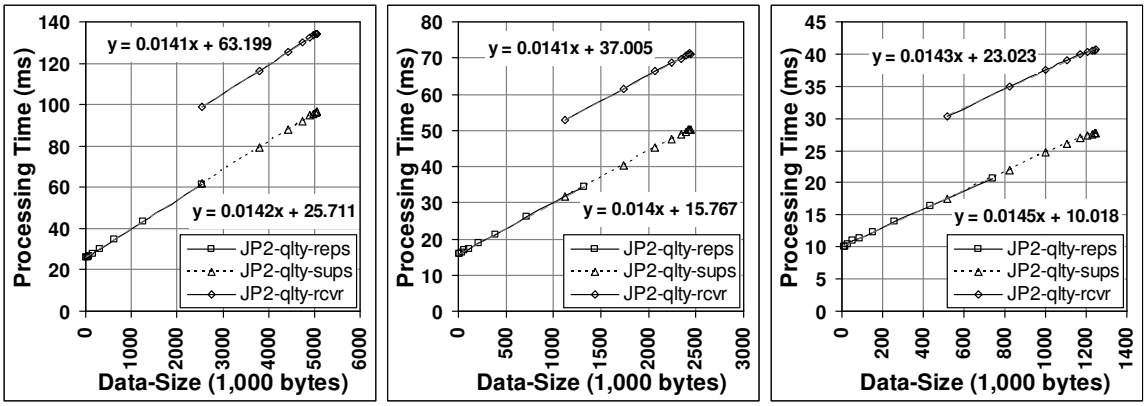
Image	Representation	Supplement's Data-Size*	Supplement's Prc. Time+	Construction's Prc. Time+
boat.jp2	$R\_lyr = 1$	2,536,081	61.7	37.4
	$R\_lyr = 2$	3,801,502	79.1	37.4
	$R\_lyr = 3$	4,432,196	87.7	38.0
	$R\_lyr = 4$	4,749,194	92.2	37.8
	$R\_lyr = 5$	4,906,879	94.6	38.0
	$R\_lyr = 6$	4,986,057	95.7	37.8
	$R\_lyr = 7$	5,025,603	96.1	37.9
	$R\_lyr = 8$	5,045,371	96.4	38.0
	$R\_lyr = 9$	5,055,265	96.6	37.9
hawaii.jp2	$R\_lyr = 1$	1,128,135	31.8	21.0
	$R\_lyr = 2$	1,738,085	40.3	21.1
	$R\_lyr = 3$	2,067,429	45.3	21.1
	$R\_lyr = 4$	2,245,450	47.6	21.0
	$R\_lyr = 5$	2,341,782	48.9	20.9
	$R\_lyr = 6$	2,394,052	49.6	21.2
	$R\_lyr = 7$	2,422,045	50.1	21.0
	$R\_lyr = 8$	2,437,268	50.2	21.2
	$R\_lyr = 9$	2,445,471	50.2	20.8
venice.jp2	$R\_lyr = 1$	517,969	17.4	12.9
	$R\_lyr = 2$	823,724	21.9	13.0
	$R\_lyr = 3$	1,001,830	24.7	12.9
	$R\_lyr = 4$	1,107,812	26.0	13.0
	$R\_lyr = 5$	1,169,460	26.9	13.1
	$R\_lyr = 6$	1,205,866	27.3	13.0
	$R\_lyr = 7$	1,227,468	27.6	13.0
	$R\_lyr = 8$	1,240,034	27.7	12.9
	$R\_lyr = 9$	1,247,476	27.7	13.0

Note: * in bytes; + in milliseconds

The data overhead in the JPEG 2000's modulation can be calculated by subtracting the data-size of the original image from the total data-sizes of the coupled representation and supplement. Of interest to note is that the data overheads for the three

experimented JPEG 2000 images are the same, i.e., 66 bytes. The data overhead comes from the supplement's `FLG`, `RPL`, and `COD` marker segments.

For each image, we further analyzed the results in a graph by plotting the processing times against the data-sizes. The graphs of analyzing different images are shown in Figure 5.6(a)–(c). For comparison's sake, the processing times of generating the representations are included in the graphs; curves *JP2-qlty-reps* are re-drawn from curves *JP2-MOD* of the graphs in Figure 5.5(a)–(c). The processing times of generating the supplements are depicted by curves *JP2-qlty-sups* (in dotted line). The last curves, *JP2-qlty-rcvr*, are the total processing times of recovering the images from different representations. The total processing times come from adding the processing times of generating the supplements and the processing times of constructing the original images (the fourth and fifth columns of Table 5.5).



(a) Boat **(b) Hawaii** **(c) Venice**

Figure 5.6 Data-size vs. processing time of modulating the JPEG 2000 images in quality aspect

The graphs in Figure 5.6 show that curves *JP2-qlty-reps* and *JP2-qlty-sups* lie on the same line. Indeed, both resulted from the same modulator, i.e., `JP2Selector`. Once again they demonstrate that the processing times in modulation correspond directly with

the resulting data-sizes. Curves *JP2-qlty-rcvr* are quite parallel to the other two curves. Remind that these curves resulted from the sequential executions of two modulators: `JP2Selector` and `JP2Joiner`. The trend-lines applied to curves *JP2-qlty-rcvr* have similar slopes to the trend-lines applied to curves *JP2-qlty-reps*. Thus, the increase rates of processing time in recovering the JPEG 2000 images (i.e., 14.1–14.3 ns/byte) are equivalent to those in generating the images' representations (i.e., 14.0–14.5 ns/byte). In summary, the overall results verify the linearity of modulating operations, and in turn, their simplicity (i.e., no involvement of any complex computation).

5.4.2 Adaptation in Resolution Aspect

The same three methods were compared and contrasted in adapting images in resolution aspect. The `djpeg+cjpeg` and `jpegfdt` applications were used to adapt the JPEG images, whereas the `JP2Selector` application was used to adapt the JPEG 2000 images. Table 5.6 shows the data-sizes and processing times of transcoding (SDT and FDT) the JPEG images in resolution aspect, while Table 5.7 shows the data-sizes and processing times of modulating the JPEG 2000 images in resolution aspect.

Different from the finding in quality aspect, here the representations resulting from FDT are generally bigger in data-size than those resulting from SDT. The reason is because the `jpegfdt` application employs an approximate downscaling algorithm which may result in bigger image coefficients, and consequently, give a larger data-size. The approximate algorithm is quite fast since it involves addition and shift operations only; any multiplication operation is purposely avoided. The downside is that the image quality

is compromised. In the experiments, we found that the quality of FDT-based representations was up to 6.5 dB lower than that of SDT-based representations.

Table 5.6 Processing times of transcoding the JPEG images in resolution aspect

Image	Representation	Data-Size* (SDT)	Prc.Time+ (SDT)	Data-Size* (FDT)	Prc.Time+ (FDT)
boat.jpg	*scale* = 1/2	1,340,018	1,299.6	1,557,164	964.4
	scale = 1/4	362,128	786.9	403,494	712.6
	scale = 1/8	98,052	485.3	92,727	643.2
hawaii.jpg	*scale* = 1/2	742,132	637.1	845,026	467.0
	scale = 1/4	218,079	381.0	236,306	341.2
	scale = 1/8	60,663	232.9	56,607	305.8
venice.jpg	*scale* = 1/2	349,702	346.0	406,760	252.0
	scale = 1/4	95,620	207.2	107,414	184.2
	scale = 1/8	27,813	126.5	26,971	165.4

Note: * in bytes; + in milliseconds

Table 5.7 Processing times of modulating the JPEG 2000 images in resolution aspect

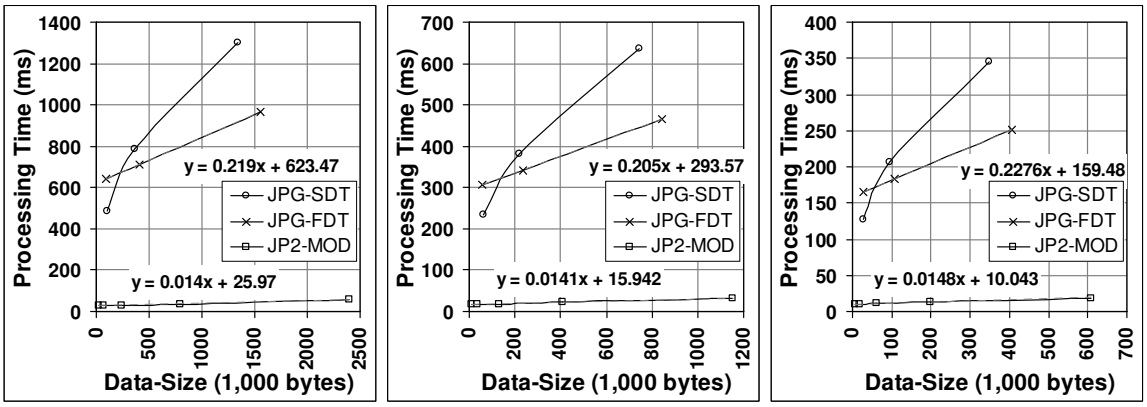
Representation	boat.jp2		hawaii.jp2		venice.jp2	
	Data-Size*	Prc.Time+	Data-Size*	Prc.Time+	Data-Size*	Prc.Time+
R_res = 1	2,403,166	59.5	1,150,789	32.1	607,484	19.0
R_res = 2	790,186	37.0	409,569	22.0	198,265	13.0
R_res = 3	236,283	29.3	129,049	18.0	61,478	11.0
R_res = 4	70,624	27.0	38,654	16.2	19,337	10.3
R_res = 5	21,923	26.2	11,436	16.0	6,262	10.1

Note: * in bytes; + in milliseconds

The results in Table 5.6 also show that the processing times of SDT are just a little higher, on average, than those of FDT. Actually, the SDT method employed by the `djpeg` application is not entirely run in the spatial domain. The downscaling process in `djpeg` is carried out simultaneously with the inverse-DCT process; in fact the inverse-DCT with downscaling process has less number of operations than the normal inverse-

DCT. As a result, it just occupies a short delay to downscale a JPEG image. It is even very simple to downscale the image by a scale of 1/8 since the necessary operation is just to get the DC coefficient and divide it by 8 (eight). As seen in the table, for downscaling by a scale of 1/8, the SDT method takes shorter processing times than the FDT method. After the image data is downscaled and converted to the spatial domain, it is converted back to the frequency domain by the `cjpeg` application. The largest part of the SDT processing time is used to compress the image (i.e., execute the `cjpeg` application).

Once again, modulation demonstrates its superiority over transcoding. The processing times of executing modulation (in Table 5.7) are much lower than those of executing transcoding (in Table 5.6). The cost of modulation is 91–95% smaller than the cost of transcoding. Figure 5.7(a)–(c) reveals more the modulation's superiority.



(a) Boat **(b) Hawaii** **(c) Venice**

Figure 5.7 Data-size vs. processing time of the three adapting methods in resolution aspect

Curves *JPG-SDT* appear non-linear in Figure 5.7(a)–(c). Different downscaling processes for different scaling factors in the `djpeg` application may cause the non-linearity. In contrast, the other two curves – *JPG-FDT* and *JP2-MOD* – are quite linear. Hence, the trend-lines can be fitted to both curves' data. The equations of the trend-lines

are displayed next to the respective curves. Similar to the previous graphs, the slope of the trend-line indicates the increase rate of processing time. Thereby, the increase rates of FDT's processing time are 200–230 ns/byte, whereas those of modulation's processing time are just 14–15 ns/byte. Notice that the increase rates of modulation's processing time are consistent with the previous findings. The slopes of curves *JP2-MOD* in Figure 5.7(a)–(c) are comparable to those in Figure 5.5(a)–(c). Based on the equations, it may be asserted that both curves are alike. This corroborates that the modulating processes in the `JP2Selector` application for quality and resolution reduction are exactly the same.

Recovering the JPEG 2000 images

The JPEG 2000 images can also be recovered from the representations given in Table 5.7. The `JP2Selector` and `JP2Joiner` applications were employed respectively for generating the supplements and constructing the original images from the corresponding representations and supplements. The supplements' data-sizes and the processing times resulting from the experiments are presented in Table 5.8. The processing times of generating the supplements are also higher than those of generating the representations due to the supplements' bigger data-sizes. The processing times of constructing the original images from various representations and supplements are also steady, as seen in the last column of Table 5.8. The results are quite consistent with the previous results in quality aspect (see Table 5.5). The data overhead for the modulation is 117 bytes, coming from the supplement's `FLG`, `RPL`, `SIZ`, and `COD` marker segments.

When the processing times were plotted against the data-sizes, we discovered the same findings as the previous analyses in quality aspect. It also proves the linearity and

simplicity of modulating operations. The plotted graphs can be found in Figure 5.8(a)–(c) (in the next subsection). In the graphs, curves *JP2-resl-reps* are reproduced from curves *JP2-MOD* of the graphs in Figure 5.7(a)–(c), curves *JP2-resl-sups* (in dotted line) denote the processing times of generating the supplements, and curves *JP2-resl-rcvr* denote the total processing times of recovering the original images. Notice the similar slopes of curves *JP2-resl-reps* (14 ns/byte) and *JP2-resl-rcvr* (14–15 ns/byte). Notice also the closeness of curves *JP2-resl-rcvr* in the graphs to those in Figure 5.6(a)–(c). This shows the similarity of their processes, albeit the adapting attributes are different.

Table 5.8 Processing times of recovering the JPEG 2000 images in resolution aspect

Image	Representation	Supplement's Data-Size*	Supplement's Prc. Time+	Construction's Prc. Time+
boat.jp2	$R\_res = 1$	2,662,512	62.7	38.0
	$R\_res = 2$	4,275,492	85.5	38.1
	$R\_res = 3$	4,829,395	93.3	38.2
	$R\_res = 4$	4,995,054	95.4	38.0
	$R\_res = 5$	5,043,755	96.1	38.2
hawaii.jp2	$R\_res = 1$	1,304,868	33.9	21.0
	$R\_res = 2$	2,046,088	44.2	21.1
	$R\_res = 3$	2,326,608	48.3	21.0
	$R\_res = 4$	2,417,003	49.1	21.0
	$R\_res = 5$	2,444,221	50.0	21.2
venice.jp2	$R\_res = 1$	651,036	19.2	13.0
	$R\_res = 2$	1,060,255	25.0	13.0
	$R\_res = 3$	1,197,042	26.9	13.0
	$R\_res = 4$	1,239,183	27.4	13.0
	$R\_res = 5$	1,252,258	27.6	13.0

Note: * in bytes; + in milliseconds

5.4.3 Adaptation in Component Aspect

Finally, we come to the last adapting attribute, the color component. The three adapting methods – SDT, FDT, and modulation – were also experimented in component aspect. There was only one representation resulting from transcoding (SDT or FDT) a JPEG image. The resulting image was grayscale since the luminance (luma) component was kept but the chrominance (chroma) components were dropped. The resulting data-sizes and processing times of generating the JPEG images' representations by SDT and FDT are presented in Table 5.9. In contrast, two representations resulted from modulating a JPEG 2000 image. One representation was obtained by dropping one chroma component, the other was obtained by dropping two chroma components. The resulting data-sizes and processing times of modulating the JPEG 2000 images in component aspect are presented in Table 5.10. Remind that the *R_cmp* parameter contains the flag-bits of the removed components.

Table 5.9 Processing times of transcoding the JPEG images in component aspect

Image	Representation	Data-Size* (SDT)	Prc.Time+ (SDT)	Data-Size* (FDT)	Prc.Time+ (FDT)
boat.jpg	*grayscale*	3,537,044	1,891.0	3,525,507	1,348.8
hawaii.jpg	*grayscale*	1,776,010	930.2	1,765,952	642.4
venice.jpg	*grayscale*	907,397	510.0	902,729	351.0

Note: * in bytes; + in milliseconds

Table 5.10 Processing times of modulating the JPEG 2000 images in component aspect

Representation	boat.jp2		hawaii.jp2		venice.jp2	
	Data-Size*	Prc.Time+	Data-Size*	Prc.Time+	Data-Size*	Prc.Time+
$R\_cmp = 2^2$	3,312,387	194.5	1,728,796	103.2	871,215	58.7
$R\_cmp = 2^2+2^1$	2,431,367	182.1	1,221,116	96.1	614,940	55.1

Note: * in bytes; + in milliseconds

As shown in Table 5.9, the processing times of FDT are lower than those of SDT. Actually the process of transcoding a JPEG image in component aspect is quite simple. In the SDT method, the `djpeg` application just decompresses the luma component and omits the chroma components, and then the `cjpeg` application re-compresses the image data. Likewise, in the FDT method, the `jpegfdt` application selects the image coefficients associated with the luma component and encodes them directly. The difference between the two transcoding methods is that the FDT method need not convert the image coefficients to the spatial domain; as a result, the FDT method is faster by 30%.

Modulating a JPEG 2000 image in component aspect requires two operations: 1) converting the image's progression order to CPRL (component–position–resolution–layer), and 2) removing some color components from the image. Recall that there is a condition attached to modulating a JPEG 2000 image in component aspect that the CPRL progression order must be employed. The first operation was carried out by `JP2Converter`, while `JP2Selector` was employed for the second operation. Although two modulating applications were used to generate the representations, modulation still took shorter processing times than transcoding (SDT and FDT alike). Comparing the results in Table 5.10 (the last row) and Table 5.9, the modulation's processing times are 90% and 85% faster than the SDT's and FDT's counterparts, respectively. The modulation's processing times here are quite high compared to the same processing times in quality and resolution aspects. This is mainly attributed to the `JP2Converter`'s process, which scans all of the image's packets twice to rearrange them. If the progression order of the image is already CPRL, of course, the `JP2Converter`'s involvement will not be required, and the resulting processing times will be much lower. In the

experiments, the progression orders of all JPEG 2000 images were deliberately set to LRCP (layer–resolution–component–position) in order to test all of the modulators.

Recovering the JPEG 2000 images

The original JPEG 2000 images can also be recovered from representations resulting from modulation in component aspect. For this purpose, in the experiments, all three JPEG 2000 modulators were used. Firstly, `JP2Converter` was used to convert the progression order of the images to CPRL. `JP2Selector` was then used to generate the suitable supplements. Lastly, `JP2Joiner` was used to construct the original images from the corresponding representations and supplements. The supplements' data-sizes and the processing times resulting from the experiments are presented in Table 5.11. In the table, the supplement's processing times (the fourth column) signifies the execution time of `JP2Converter` and `JP2Selector` together.

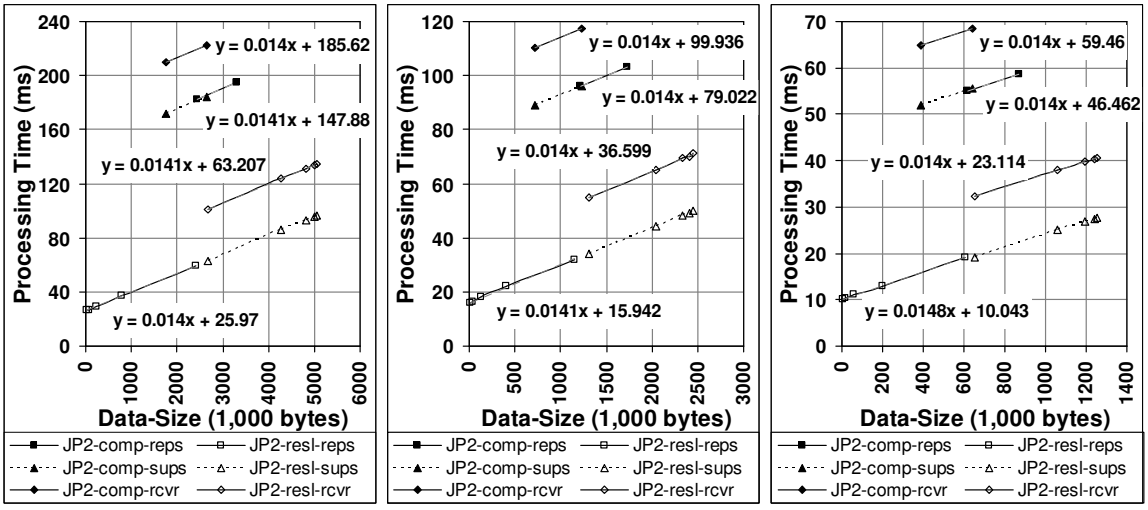
Table 5.11 Processing times of recovering the JPEG 2000 images in component aspect

Image	Representation	Supplement's Data-Size*	Supplement's Prc. Time+	Construction's Prc. Time+
boat.jp2	$R\_cmp = 2^2$	1,753,220	172.0	38.1
	$R\_cmp = 2^2+2^1$	2,634,240	184.3	38.1
hawaii.jp2	$R\_cmp = 2^2$	726,790	89.1	21.0
	$R\_cmp = 2^2+2^1$	1,234,470	96.2	21.0
venice.jp2	$R\_cmp = 2^2$	387,234	51.9	13.0
	$R\_cmp = 2^2+2^1$	643,509	55.5	13.0

Note: * in bytes; + in milliseconds

Compared to the previous results in quality and resolution aspects, the supplements' processing times in component aspect are also higher (compare Table 5.11 with Table 5.5 and Table 5.8). Again, this is attributed to the `JP2Converter`'s process.

On the contrary, the construction's processing times are equal to the corresponding results in quality and resolution aspects. The data overhead for the modulation in component aspect is 46 bytes, which comes from the supplement's `FLG` marker segment.



(a) Boat **(b) Hawaii** **(c) Venice**

Figure 5.8 Data-size vs. processing time of modulating the JPEG 2000 images in resolution and component aspects

The plotting of the processing times against the data-sizes is displayed in Figure 5.8(a)–(c). The plotted graphs of modulating the images in component aspect are shown at the top of the figures (note: at the bottom are the plotted graphs of modulating the images in resolution aspect). As noticed in the graphs, curves *JP2-comp-reps* and *JP2-comp-rcvr* have similar slopes; the indicated increase rates of processing time of the two curves are about 14 ns/byte. In addition, the curves appear quite parallel to the curves belonging to the modulation in resolution aspect (notice their similar slopes); it also means that they are also parallel to the curves belonging to the modulation in quality aspect (see Figure 5.6(a)–(c)). Thus, the previous assertion is still upheld that modulating operations are quite linear, simple, and apparently fast.

5.5 Concluding Remarks

The JPEG 2000 standard is superior to the JPEG standard in data compression. The experimental results show that, to attain the same image quality, a JPEG 2000 image needs at most 60% the data-size of a JPEG image. Further, JPEG 2000's superiority over JPEG can be visibly observed in a low-quality image. While the JPEG low-quality image exhibits the characteristic blocking artefacts, the JPEG 2000 counterpart does not.

Executing modulation takes much less time than executing transcoding. Some experiments were conducted by measuring the processing times required to execute modulation and transcoding (SDT and FDT) in quality, resolution, and component aspects. In quality aspect, executing modulation is at least 30 and 16 times faster than executing transcoding – SDT and FDT, respectively. In resolution aspect, the processing time of modulation is 11–20 times smaller than that of transcoding. In component aspect, modulation is faster than transcoding by a factor of 10 and 6.5 – SDT and FDT, respectively – albeit several modulators were involved.

The increase rates of processing time may further indicate the characteristics of modulation and transcoding. The increase rates of transcoding's processing time vary in different aspects and are found to be over 170 ns/byte in the experiments. By contrast, the increase rates of modulation's processing time are steady at 14–15 ns/byte. This is attributed to the linearity and simplicity of modulating operations. Modulation just selects or drops the image's packets, and hence, no complex computation is involved.

More importantly, it has been demonstrated that, by modulation, the original images can be recovered from their representations. The recovery process is efficient (i.e.,

the data overhead is minimal) and consistent with modulation's linearity. Also, this corroborates our claim that modulation is reversible.

This chapter has shown the benefits of modulation, which is characterized as fast, efficient, and reversible. These characteristics are very beneficial if applied to the Web content delivery. Just imagine the gains that we may obtain if any Web object can be modulated easily. Meanwhile, the multimedia trend also tells us that more scalable (modulation-friendly) data-formats will come out in the near future. To serve a client's request, an object's representation can be generated as easily as taking part(s) of the object. Later on, if the client wants a better representation of the object, only the necessary part(s) of the object is (are) delivered. Putting a caching proxy between the server and the client can further increase the object's reusability since the object now may be shared among many clients. Thereby, the Internet bandwidth can be reasonably consumed and the client perceived latency can be much improved. However, a new framework in Web content delivery is required to exploit modulation's benefits. The following chapters discuss how to accommodate modulation in the existing Web content delivery.

Chapter 6

Framework for Pervasive Web Content Delivery

The fine-grained, scalable data model, proposed in Chapter 3, is the fundamental aspect of the future Web content delivery. As indicated in Chapter 5, modulation – the content adaptation based on the data model – is very beneficial to the delivery of Web content. However, modulation requires some modifications on the existing Web content delivery. This chapter presents the framework which can exploit modulation's benefits to its fullest. It begins with a comparison of two adapting approaches: proxy- and server-based adaptation. Different kinds of scenarios are further evaluated. At the end of the chapter, the proposed framework is discussed and its elements are detailed.

6.1 Proxy- vs. Server-Based Adaptation

To deploy pervasive Web content delivery, the content providers have to deal with heterogeneous clients. For a Web object, various representations (variants) are needed to serve the clients. The representations can be generated either offline or online. As exposed in Section 1.2, the offline approach is space-consuming, rigid, and difficult to maintain. It is unfavorable to the content providers. The online approach, on the other hand, involves on-demand adaptation to serve each client with a particular representation of the Web object. Since the online approach is the ultimate choice, adaptation is essential for deploying pervasive Web content delivery.

Now we face another decision-making task with regard to the location where adaptation should take place. Initially we have three options for the location: server, proxy, or client. Although currently some browsers can perform adaptation, many client devices do not have the capability – due to lack of resources or applications – to adapt a Web object. Moreover, if the Web objects are adapted at the client side, the client will suffer not only the transmitting time of the full objects but also the delay of adapting the objects; in short, no time gain is obtained. Therefore, we set aside the option of adapting the Web objects at the client side.

Proxy-based (both forward and reverse proxies) adaptation offers technical and economical benefits. The proxy can help reduce the server's load particularly in adapting Web objects. In addition, adaptation can be extended more easily at the proxy than it is at the server. Even if a new adapting method cannot be installed in the existing proxy, a new proxy can always be set up without much difficulty. It is also more efficient to apply

adaptation at the proxy because the proxy runs and adapts on behalf of many servers, and therefore, a less number of installations are required. Furthermore, the cost of running the adapting proxy may be shared by many parties. Due to the proxy's dependency on other servers and proxies, running adaptation at the proxy may also speed up the standardization drive. More benefits may be attained if the proxy is equipped with a caching system. In that case, some Internet traffic may be avoided and the response time may be shortened.

On the other hand, employing server-based adaptation, the content provider can maintain its control over its Web objects. Since the content provider knows exactly which aspects of each object need to be presented to the clients, it can adapt the objects without losing essential information. Thereby, the end-to-end semantics of the Web objects can be preserved between the server and the clients. In addition, by performing adaptation at the server, the gain resulting from the reduced response time may be higher than the gain obtained if the adaptation is done at the proxy. Adaptation usually has the effect of reducing the object's data-size, so that the transmitting time – and consequently, the response time – can be cut down. It is plausible that transmitting a smaller, adapted object consumes less time than transmitting the original object, and certainly, the earlier the adaptation takes place (at the server rather than at the proxy) is the better. However, some adapting processes are quite complex and resource-consuming, so they may cause a burden to the server. Alternatively, a surrogate may be placed in front of the server to help with the adaptation.

Clearly, each of the two approaches above has its advantages as well as disadvantages. The server-directed transcoding (SDT) [Mog01, KnLM03] has been

proposed to get benefits of both the proxy-based and server-based approaches. In SDT, the adaptation is done by the proxy under the server's direction. In this fashion, the technical and economical benefits from the proxy-based approach are attained, while the end-to-end semantics of the Web objects is duly preserved.

6.2 Evaluation of Adapting Approaches

In order to find out which one of the approaches is best suited to deploy pervasive Web content delivery, let us evaluate analytically all of the possible approaches. The analytical model used here follows closely the model proposed by Han et al. [HaBL98], and it is presented in Figure 6.1. Suppose the Web object to be distributed is an image of data-size S. When the image is adapted, the resulting representation has the data-size of S_{rep}. The original image resides in the Web server. The image or its representation is transmitted to the Web client via the caching proxy. The effective bandwidth on the server-proxy link is B_{sp}, whereas the effective bandwidth on the proxy-client link is B_{pc}.

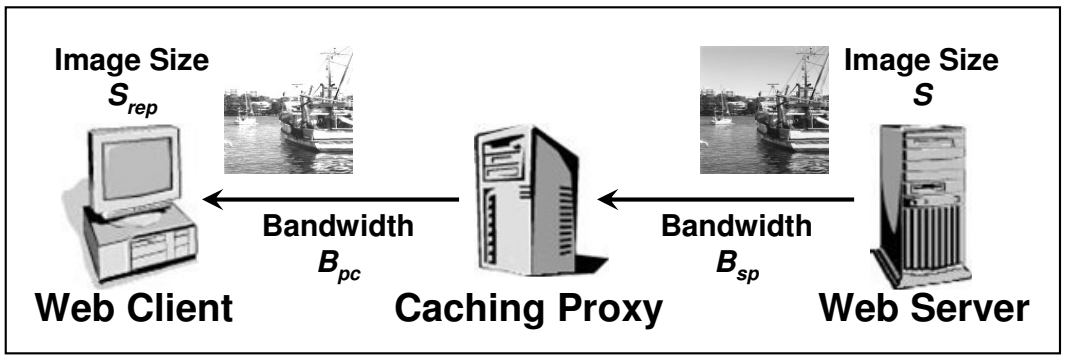


Figure 6.1 Analytical model of pervasive Web content delivery

As mentioned before and also in the previous chapters, the objective of adaptation is to reduce the Web object's data-size, in addition to addressing the client device's

limitations. The reduction is sometimes necessary to overcome different bandwidths and delays of the communication media stretching between the server and the client. By reducing the object's data-size, the overall response time is expected to be shorter than the time required to fetch the original object. Hence, the response time of fetching the original image from the server becomes the benchmark that other response times have to beat. The response time measured here starts the moment a TCP connection is initiated and ends when the last TCP packet is received by the client.

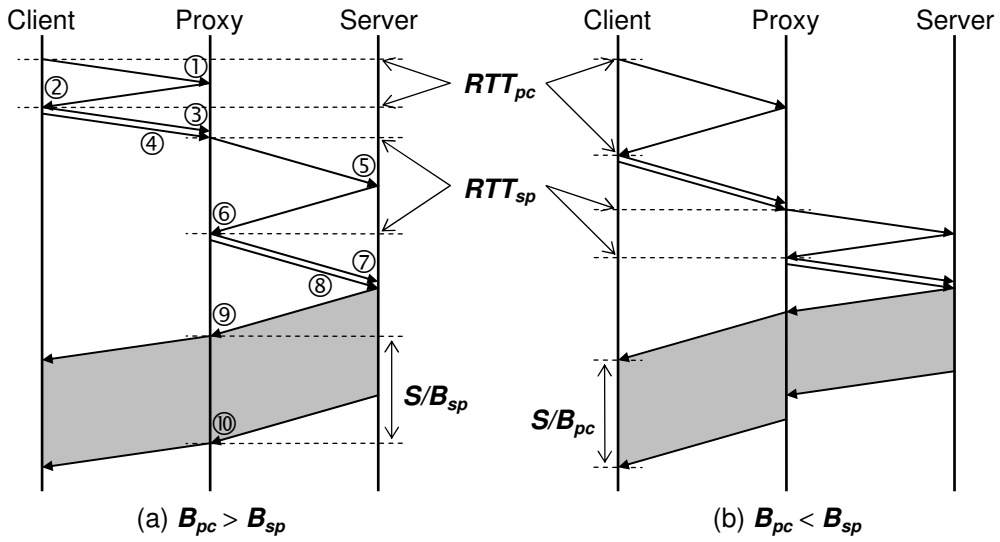


Figure 6.2 Timeline for fetching the original image from the server

Figure 6.2(a) – taken from Chi et al.'s paper [ChC02] – shows the TCP packets transmitted among the client, the proxy, and the server. Packets 1–3 are respectively SYN, SYN/ACK, and ACK packets, which constitute the three-way handshake in establishing a TCP connection between the client and the proxy. Packet 4 is the client's HTTP request for the image. Packets 5–7 again constitute the three-way handshake, but now between the proxy and the server. Packet 8 is the client's HTTP request that the proxy passes on to the server. Packets 9 and 10 are the first and the last packets of the

server's HTTP response, which carries the requested image; there are likely other packets between packets 9 and 10.

Suppose R_o is the response time of fetching the original image from the server without adaptation and assuming that the server-proxy link is slower than the proxy-client link (which is a typical case), we may formulate R_o – based on the timeline presented in Figure 6.2(a) – as follows

$$R_o^{(\mathrm{i})} = 2 \cdot RTT_{pc} + 2 \cdot RTT_{sp} + \frac{S}{B_{sp}} \qquad (1)$$

RTT_{pc} is the roundtrip time between the client and the proxy; that is, the time required to send a packet back and forth between the client and the proxy. Likewise, RTT_{sp} is the roundtrip time between the proxy and the server. Since B_{sp} (the bandwidth on the server-proxy link) is less than B_{pc} (the bandwidth on the proxy-client link), the proxy receives the image's packets more slowly than it may send. As depicted in Figure 6.2(a), the overall speed of transmitting the packets is determined by B_{sp}. On the other hand, if B_{sp} is greater than B_{pc}, as depicted in Figure 6.2(b), the proxy receives the entire image's packets much earlier and needs to buffer them. So, the transmission of some of the packets is delayed at the proxy due to the slower proxy-client link. The overall speed of transmitting the packets is instead determined by B_{pc}, and therefore, R_o needs be re-formulated as follows

$$R_o^{(\mathrm{ii})} = 2 \cdot RTT_{pc} + 2 \cdot RTT_{sp} + \frac{S}{B_{pc}} \qquad (2)$$

Combining equations (1) and (2), we get the generic response time of fetching the original image from the server without adaptation; that is,

$$R_o = 2 \cdot RTT_{pc} + 2 \cdot RTT_{sp} + \frac{S}{\min(B_{pc}, B_{sp})} \quad (3)$$

Now we are ready to analyze the two adapting approaches: proxy- and server-based. In the following subsections, two different scenarios are examined. The first scenario is when the image is requested for the first time; that is, no representation of the image can be found in the proxy's cache. The second scenario is when a particular representation of the image exists in the proxy's cache, and the cached representation may be used to serve the current client request, either fully or partly.

6.2.1 Scenario 1: First-Time Delivery

In the following analyses, assume that the adaptation takes the same amount of time whether it is executed at the proxy or at the server. When a client requests for the image from the proxy and the proxy does not have any representation of the image in its cache, the request is passed on to the server. The server may send the original image to the proxy for adaptation, and the proxy sends the adapted image to the client. Alternatively, the server may adapt the image by itself and send the representation to the client via the proxy. These two different approaches are depicted in Figure 6.3.

If the adaptation is done at the proxy, the transmission of the original image – from the server to the proxy – may take a long time. However, the long transmission of the image may be compensated by the following transmission of the adapted image – supposedly smaller in data-size than the original image – from the proxy to the client. As depicted in Figure 6.3(a), the response time of delivering for the first time the image adapted by the proxy is as follows

$$R_p^{fst} = 2 \cdot RTT_{pc} + 2 \cdot RTT_{sp} + \frac{S}{B_{sp}} + D_d\left(S, S_{rep}\right) + \frac{S_{rep}}{B_{pc}} \qquad (4)$$

S_{rep} is the data-size of the adapted image, which is smaller than S. The delay of adapting the image is indicated by $D_d(S, S_{rep})$; the subscript 'd' signifies a downscaling operation, by which a lower-fidelity representation can be obtained. In modulation, this is done by the selection operation. As denoted by the operation's input parameters, the adaptation delay depends on the input and output images' data-sizes. The smaller the input and/or output image's data-size, the smaller is the adaptation delay, and vice versa. As a matter of fact, the adaptation delay also depends on many other factors such as the image's data-type, resolution, number of blocks, and the adapting parameters applied to the image. The later section will discuss in more detail how to predict the adaptation delay.

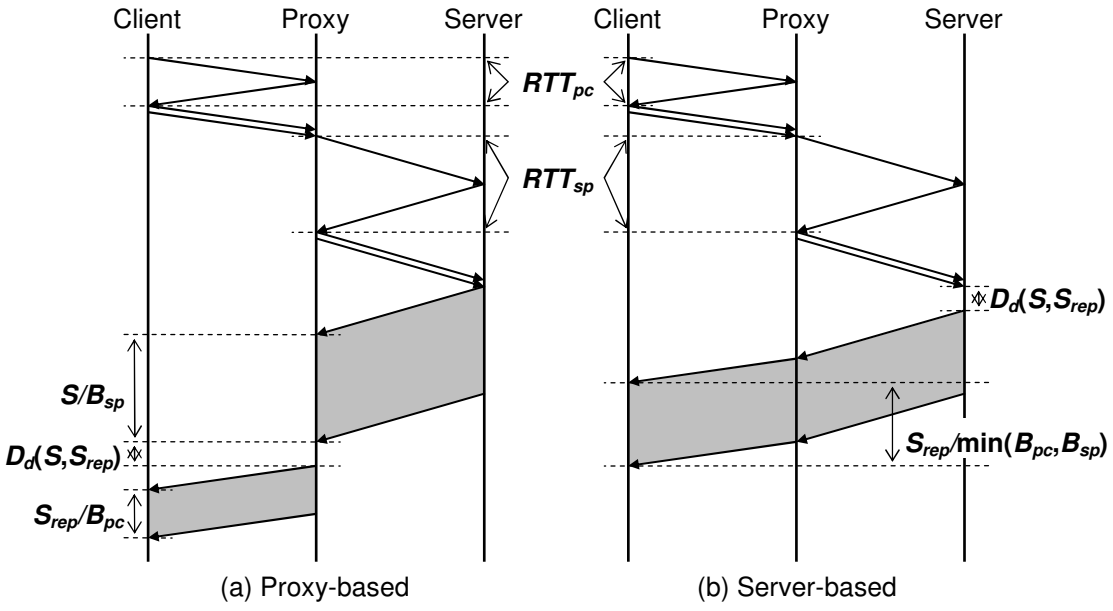


Figure 6.3 Timeline for the first-time delivery of the adapted image

If shortening the response time is the objective of the adaptation, then $R_p^{fst} < R_o$ must be satisfied. As asserted by Han et al. [HaBL98], the condition may be fulfilled only

if $B_{pc} < B_{sp}$. [Comparing equations (1) and (4), when $B_{pc} > B_{sp}$, it is always the case that $R_p^{fst} > R_o$.] Hence, the adaptation's objective can be attained if

$$D_d(S, S_{rep}) < \frac{S - S_{rep}}{B_{pc}} - \frac{S}{B_{sp}} \tag{5}$$

The first term on the right-hand side of inequality (5) denotes the benefit of transmitting the adapted image over transmitting the original image on the proxy-client link. The second term on the right-hand side, by contrast, denotes the cost of transmitting the original image on the server-proxy link. Thus, the inequality basically says that, in order to attain the adaptation's objective, the delay of adapting the image should be less than the benefit gained from transmitting the adapted image.

If the adaptation is done at the server, the object transmitted to the proxy, and subsequently to the client, is indeed the adapted image. Since the adapted image is smaller in size than the original image, the response time here is correspondingly smaller than the response time of delivering the original image (R_o). As depicted in Figure 6.3(b), the response time of delivering for the first time the image adapted by the server is as follows

$$R_s^{fst} = 2 \cdot RTT_{pc} + 2 \cdot RTT_{sp} + D_d(S, S_{rep}) + \frac{S_{rep}}{\min(B_{pc}, B_{sp})} \tag{6}$$

In this case, if the same objective is to be attained, then $R_s^{fst} < R_o$ must be satisfied. Replacing R_s^{fst} and R_o with equations (6) and (3), respectively, the objective's inequality becomes as follows

$$D_d(S, S_{rep}) < \frac{S - S_{rep}}{\min(B_{pc}, B_{sp})} \tag{7}$$

Similar to inequality (5), this inequality also says that the adaptation's objective can be attained if the delay of adapting the image is less than the benefit gained from transmitting the adapted image. While the benefit in inequality (5) only comes from transmitting the adapted image on the proxy-client link, the benefit in inequality (7) comes from the overall transmission, not just on the particular proxy-client link. In addition, inequality (5) has a condition attached that $B_{pc} < B_{sp}$ must be satisfied, whereas inequality (7) does not have any other condition. Apparently, inequality (7) is easier to attain than inequality (5).

We may further evaluate the proxy-based and server-based approaches by comparing equations (4) and (6). If $B_{pc} < B_{sp}$, it is always the case that $R_p^{fst} > R_s^{fst}$ since

$$\frac{S}{B_{sp}} + D_d(S, S_{rep}) + \frac{S_{rep}}{B_{pc}} > D_d(S, S_{rep}) + \frac{S_{rep}}{\min(B_{pc}, B_{sp})}$$

$$\frac{S}{B_{sp}} + \frac{S_{rep}}{B_{pc}} > \frac{S_{rep}}{B_{pc}}$$

$$\frac{S}{B_{sp}} > 0$$

Likewise, if $B_{pc} > B_{sp}$, then

$$\frac{S}{B_{sp}} + D_d(S, S_{rep}) + \frac{S_{rep}}{B_{pc}} > D_d(S, S_{rep}) + \frac{S_{rep}}{\min(B_{pc}, B_{sp})}$$

$$\frac{S}{B_{sp}} + \frac{S_{rep}}{B_{pc}} > \frac{S_{rep}}{B_{sp}}$$

$$\frac{S - S_{rep}}{B_{sp}} + \frac{S_{rep}}{B_{pc}} > 0$$

Thus, R_p^{fst} is always greater than R_s^{fst} in any condition. In conclusion, the server-based approach is more beneficial than the proxy-based one in delivering the adapted image for the first time.

6.2.2 Scenario 2: Subsequent Delivery

For subsequent delivery, assume that a representation of the image has been stored in the proxy's cache. The cached image's representation may be used to serve a subsequent client request for the same image, particularly in the case of modulation. It is also possible to serve the request directly from the server like what is done before in the first-time delivery, but serving the request with the cached representation may further shorten the response time since it may exclude the server in the image delivery, as shown in Figure 6.4(a).

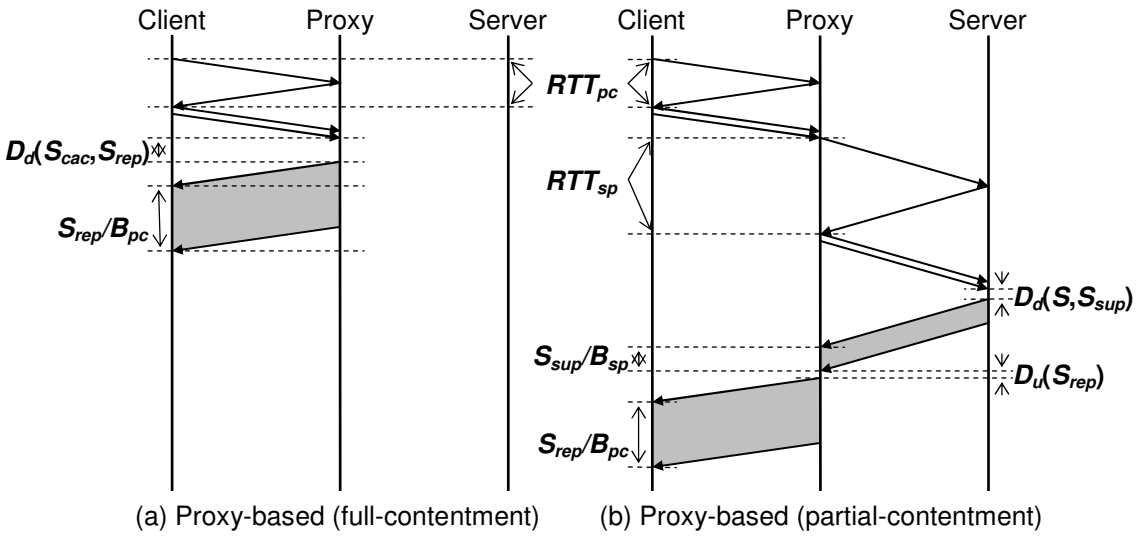


Figure 6.4 Timeline for the subsequent delivery of the adapted image

Actually there are two possibilities of making use of the representation cached in the proxy. The first possibility is when the cached representation can fully serve the client request. This is indeed the situation depicted in Figure 6.4(a). In this case, the response time can be expressed as follows

$$R_p^{ssq(\text{full})} = 2 \cdot RTT_{pc} + D_d\left(S_{cac}, S_{rep}\right) + \frac{S_{rep}}{B_{pc}} \tag{8}$$

Since the server is not involved in the delivery, no connection is set up between the proxy and the server. Notice also that the delay of adapting the image depends on the data-sizes of the cached representation (S_{cac}) as the input and the resulting representation (S_{rep}) as the output. This adaptation delay is less than the delay of generating the requested representation from the original image (S), since $S_{cac} \leq S$. Moreover, the adaptation delay can be discarded if the cached representation is actually the requested representation; that is, no adaptation is required. To attain the adaptation's objective, the response time of this full-contentment scheme ($R_p^{ssq(\text{full})}$) should be less than the response time of fetching the original image (R_o). The relevant inequality is expressed as follows

$$R_p^{ssq(\text{full})} < R_o \begin{cases} \text{if } B_{pc} < B_{sp}, \quad D_d(S_{cac}, S_{rep}) < \dfrac{S - S_{rep}}{B_{pc}} + 2 \cdot RTT_{sp} \\ \text{if } B_{pc} > B_{sp}, \quad D_d(S_{cac}, S_{rep}) < \dfrac{S}{B_{sp}} - \dfrac{S_{rep}}{B_{pc}} + 2 \cdot RTT_{sp} \end{cases} \tag{9}$$

As mentioned earlier, one of the possible challenges that the Web content delivery has to overcome is the different media bandwidths. However, the challenge may bring benefit, too, as demonstrated here in this scheme. In the upper case of inequality (9), if $B_{pc} << B_{sp}$ (B_{pc} is very much less than B_{sp}), the value on the right-hand side of the inequality gets bigger. Similarly, in the lower case, if $B_{pc} >> B_{sp}$ (B_{pc} is very much greater than B_{sp}), the value on the right-hand side of the inequality gets bigger. Either way, the smaller the representation's data-size (S_{rep}), the bigger is the value on the right-hand side of the inequality. When the value on the right-hand side of the inequality gets bigger, the scheme's benefit increases. Apparently, the adaptation's objective is pretty attainable in the full-contentment scheme, so long as the adaptation itself is not really delaying.

The second possibility of reusing the representation cached in the proxy is when the cached representation can only partly serve a subsequent client request. This is a unique case found only in modulation. In this case, the requested representation is more sophisticated than the cached one. A supplement is thus required from the server to enhance the cached representation. Receiving the supplement, the proxy joins it with the cached representation to obtain the requested representation. So, there are two adapting processes involved. The first adapting process is the generation of the supplement at the server, and the second is the join of the cached representation and the supplement at the proxy. As depicted in Figure 6.4(b), the response time of this scheme is as follows

$$R_p^{ssq(\text{part})} = 2 \cdot RTT_{pc} + 2 \cdot RTT_{sp} + D_d(S, S_{sup}) + \frac{S_{sup}}{B_{sp}} + D_u(S_{rep}) + \frac{S_{rep}}{B_{pc}} \qquad (10)$$

$D_d(S, S_{sup})$ is the delay of generating the supplement; remind that in modulation the delay of generating a supplement is exactly the same as that of generating a representation. Since $S_{sup} < S_{rep}$, this adaptation delay is less than the delay of generating the requested representation. On the other hand, $D_u(S_{rep})$ is the delay of joining the cached representation and the supplement; the subscript '*u*' denotes an upscaling operation, by which a higher-fidelity representation can be obtained. The adaptation delay of the join operation is determined by the data-size of the resulting representation (S_{rep}); this will be elaborated in the later section. Employing the partial-contentment scheme is more beneficial than fetching the original image if the following conditions are satisfied.

$$R_p^{ssq(\text{part})} < R_o \begin{cases} \text{if } B_{pc} < B_{sp}, \quad D_d(S, S_{sup}) + D_u(S_{rep}) < \dfrac{S - S_{rep}}{B_{pc}} - \dfrac{S_{sup}}{B_{sp}} \\ \text{if } B_{pc} > B_{sp}, \quad D_d(S, S_{sup}) + D_u(S_{rep}) < \dfrac{S - S_{sup}}{B_{sp}} - \dfrac{S_{rep}}{B_{pc}} \end{cases} \qquad (11)$$

Similar to the full-contentment scheme before, the partial-contentment scheme benefits from the different bandwidths between the proxy-client and server-proxy links. In the upper case of inequality (11), the scheme's benefit (i.e., the value on the right-hand side of the inequality) increases if $B_{pc} << B_{sp}$. Likewise, the scheme's benefit increases in the lower case if $B_{pc} >> B_{sp}$. When the data-size of the requested representation or that of the supplement gets smaller, the scheme's benefit increases, as well. Although it is not as good as the full-contentment scheme, the partial-contentment scheme can still be beneficial.

Both schemes depicted in Figure 6.4 employ the proxy-based approach. If the server-based approach is employed to serve the subsequent client request, the situation is not different from the server-based, first-time delivery of the adapted image; the timeline of which is shown in Figure 6.3(b) and its response time is expressed in equation (6). Now let us compare the response times of the proxy-based (full- and partial-contentment) and server-based approaches in serving the subsequent client request. Comparing equations (8) and (6), we get

$$D_d(S_{cac}, S_{rep}) + \frac{S_{rep}}{B_{pc}} < D_d(S, S_{rep}) + \frac{S_{rep}}{\min(B_{pc}, B_{sp})} + 2 \cdot RTT_{sp}$$

$$\text{if } B_{pc} < B_{sp} \begin{cases} D_d(S_{cac}, S_{rep}) + \dfrac{S_{rep}}{B_{pc}} < D_d(S, S_{rep}) + \dfrac{S_{rep}}{B_{pc}} + 2 \cdot RTT_{sp} \\ 0 < (D_d(S, S_{rep}) - D_d(S_{cac}, S_{rep})) + 2 \cdot RTT_{sp} \end{cases}$$

$$\text{if } B_{pc} > B_{sp} \begin{cases} D_d(S_{cac}, S_{rep}) + \dfrac{S_{rep}}{B_{pc}} < D_d(S, S_{rep}) + \dfrac{S_{rep}}{B_{sp}} + 2 \cdot RTT_{sp} \\ 0 < (D_d(S, S_{rep}) - D_d(S_{cac}, S_{rep})) + \left(\dfrac{S_{rep}}{B_{sp}} - \dfrac{S_{rep}}{B_{pc}} \right) + 2 \cdot RTT_{sp} \end{cases}$$

$$\left[\text{Note}: D_d(S_{cac}, S_{rep}) \le D_d(S, S_{rep}), \text{ since } S_{cac} \le S\right]$$

The evaluation shows that $R_p^{ssq(\text{full})}$ is always less than R_s^{fst}. It means that the full-contentment, proxy-based approach is more beneficial than the server-based approach.

However, it is totally different when equations (10) and (6) are compared; the result below may not be as obvious as above.

$$D_d(S, S_{sup}) + \frac{S_{sup}}{B_{sp}} + D_u(S_{rep}) + \frac{S_{rep}}{B_{pc}} \rhd\lhd D_d(S, S_{rep}) + \frac{S_{rep}}{\min(B_{pc}, B_{sp})}$$

$$\text{if } B_{pc} < B_{sp} \begin{cases} D_d(S, S_{sup}) + D_u(S_{rep}) + \frac{S_{sup}}{B_{sp}} + \frac{S_{rep}}{B_{pc}} \rhd\lhd D_d(S, S_{rep}) + \frac{S_{rep}}{B_{pc}} \\ \qquad \frac{S_{sup}}{B_{sp}} + D_u(S_{rep}) \rhd\lhd (D_d(S, S_{rep}) - D_d(S, S_{sup})) \end{cases}$$

$$\text{if } B_{pc} > B_{sp} \begin{cases} D_d(S, S_{sup}) + D_u(S_{rep}) + \frac{S_{sup}}{B_{sp}} + \frac{S_{rep}}{B_{pc}} \rhd\lhd D_d(S, S_{rep}) + \frac{S_{rep}}{B_{sp}} \\ \qquad \frac{S_{sup}}{B_{sp}} - \left(\frac{S_{rep}}{B_{sp}} - \frac{S_{rep}}{B_{pc}} \right) + D_u(S_{rep}) \rhd\lhd (D_d(S, S_{rep}) - D_d(S, S_{sup})) \end{cases}$$

$$\left[\text{Note}: D_d(S, S_{sup}) < D_d(S, S_{rep}), \text{since } S_{sup} < S_{rep}\right]$$

As revealed by the evaluation, $R_p^{ssq(\text{part})} \rhd\lhd R_s^{fst}$ means that the response time of the partial-contentment, proxy-based approach could be less than, equal to, or greater than that of the server-based approach. In the upper case ($B_{pc} < B_{sp}$), the partial-contentment, proxy-based approach will be more beneficial than the server-based approach if the data-size of the supplement gets smaller. The smaller supplement decreases the delay of generating the supplement ($D_d(S, S_{sup})$), and therefore, increases the total benefit on the right-hand side of the inequality. In the same time, the smaller supplement also decreases the cost of delivering the supplement from the server to the proxy (S_{sup}/B_{sp}), as suggested by the left-hand side of the inequality. Similarly, in the lower case ($B_{pc} > B_{sp}$), the smaller the supplement's data-size, the more beneficial is the partial-contentment, proxy-based approach than the server-based approach. In addition, if $B_{pc} >> B_{sp}$ (B_{pc} is very much

greater than B_{sp}), the terms in the brackets at the left-hand side of the inequality gets bigger, and hence the benefit of the proxy-based approach increases, too. Comparing the left-hand sides of both upper and lower cases, we can assert that the partial-contentment, proxy-based approach is more beneficial when $B_{pc} > B_{sp}$ (the lower case).

Considering the overall possible schemes (full- and partial-contentment), it can be inferred that the proxy-based approach is more beneficial than the server-based one in serving the subsequent client request, although in some cases the server-based may be more beneficial.

The evaluations presented in this section show that the server-based approach is beneficial in some adaptation's cases, but in other cases the proxy-based approach is more beneficial. Generally, the server-based approach is more beneficial in serving the first-time client request for the Web object, whereas the proxy-based approach is more beneficial in serving the subsequent client request (assuming that there is an object's representation available and reusable in the proxy's cache). This finding is a key determinant in building the proposed framework for pervasive Web content delivery. Before discussing the proposed framework, as indicated before, the next section discusses how the delay of adapting an image is predicted.

6.3 Prediction of Adaptation Delay

The delay of adapting a Web object may contribute considerably to the overall response time in serving a client request. That is why predicting the adaptation delay is

important. In agreement with the model shown in Figure 6.1, this section tries to predict the delay of the image adaptation. The JPEG transcoding (FDT) and the JPEG 2000 modulation – described and exploited in the previous chapters – are used as illustrations. The prediction is confined to the image adaptation in quality aspect.

As mentioned earlier, the delay of adapting an image depends on many factors, which can be classified into four groups as follows:

1. The input image's attributes, such as: data-size, data-type, resolution (width × height), number of color components, number of blocks, etc.
2. The output image's attributes, such as: data-size, data-type, resolution, etc.
3. The adapting parameters, such as: type of adaptation, expected quality of the result, downscaling factor, etc.
4. The implementation issues, such as: efficiency and optimization of the implemented algorithm, CPU's speed of the machine where the adaptation is executed, etc.

Among the factors mentioned above, we are interested in examining some of them which can significantly affect the adaptation delay. The factors mentioned in point 4 are disregarded since they vary widely; instead we evaluate the same adapting applications in a single machine without considering their efficiency or whether they are optimal. Since only the image adaptation in quality aspect is considered in this examination, most of the adapting parameters (in point 3) can be dropped, too. In the previous chapter, the results of transcoding a JPEG image and modulating a JPEG 2000 image have shown that there is a strong correlation between the output data-size and the adaptation delay (i.e., processing time). This is indicated by the linear appearances of the resulting curves. Other significant factors may be the input image's attributes such as its data-size, resolution,

and number of blocks; the number of color components is not affected by the image adaptation in quality aspect.

Two types of adapting operations are analyzed in the following subsections. The first is the downscaling operation (denoted as D_d in the previous section); this can be analyzed in transcoding as well as in modulation. The second is the upscaling operation (denoted as D_u in the previous section), which is exclusive to modulation only.

6.3.1 Adaptation Delay in a Downscaling Operation

The following experiments were conducted to understand which factors considerably determine the delay of the image adaptation in a downscaling operation (with respect to modulation, it is also known as the selection operation). Using some images obtained from the CD-ROM accompanying Taubman and Marcellin's book [TaM02], the JPEG and JPEG 2000 images were created for the test data. The same methodology explained in Subsection 5.2.2 was used to create the JPEG and JPEG 2000 images. For each of the image standards, ten different images – varied in width and height – were created. The JPEG images were created first, and then, the JPEG 2000 images were created using the associated JPEG images' data-sizes as references; therefore, the associated JPEG and JPEG 2000 images had similar data-sizes. Table 6.1 shows the particulars of the test data.

For each of the JPEG images, the number of DCT blocks is calculated and presented under the JPEG header in Table 6.1. The number of DCT blocks is determined by counting the number of 8×8-pixel blocks in a JPEG image. Likewise, for each of the

JPEG 2000 images, the number of packets and the number of code-blocks are calculated and displayed under the JPEG 2000 header in the table. There is not much variation in the number of packets since the JPEG 2000 images have the same numbers of color components and layers, so only the different numbers of decomposition levels determine the number of packets. The number of code-blocks in a JPEG 2000 image, on the other hand, is more complex to calculate. As the default resolution of the code-block is 64×64 pixels, the number of code-blocks is determined by counting the number of 64×64-pixel blocks in each decomposition level.

Table 6.1 Test data for predicting the adaptation delay in a downscaling operation

Image	Width (pixels)	Height (pixels)	JPEG		JPEG 2000		
			Data-Size (bytes)	DCT Blocks	Data-Size (bytes)	Packets	Code-Blocks
Boat	1,976	2,960	5,065,493	91,390	5,065,266	180	46,140
Wharf	2,944	1,966	4,689,377	90,528	4,689,348	180	44,700
Flower	1,418	1,825	2,653,184	40,762	2,653,069	180	22,080
Hawaii	2,097	1,391	2,455,504	45,762	2,455,510	180	23,820
Yosemite	1,551	1,045	1,544,959	25,414	1,544,947	180	15,240
Venice	1,055	1,568	1,258,420	25,872	1,258,309	180	15,240
Bath	1,054	704	786,216	11,616	786,083	150	6,990
NewYork	706	1,029	574,491	11,481	574,541	150	6,930
Rome	353	528	181,841	2,970	181,917	120	2,130
Eiffel	351	526	163,095	2,904	163,077	120	2,130

The experiments were executed by reducing the JPEG images' quality and reducing the JPEG 2000 images' number of layers. For each adaptation, its processing time was duly noted. Furthermore, each adaptation was executed several times and the recorded processing times were averaged from those several runs.

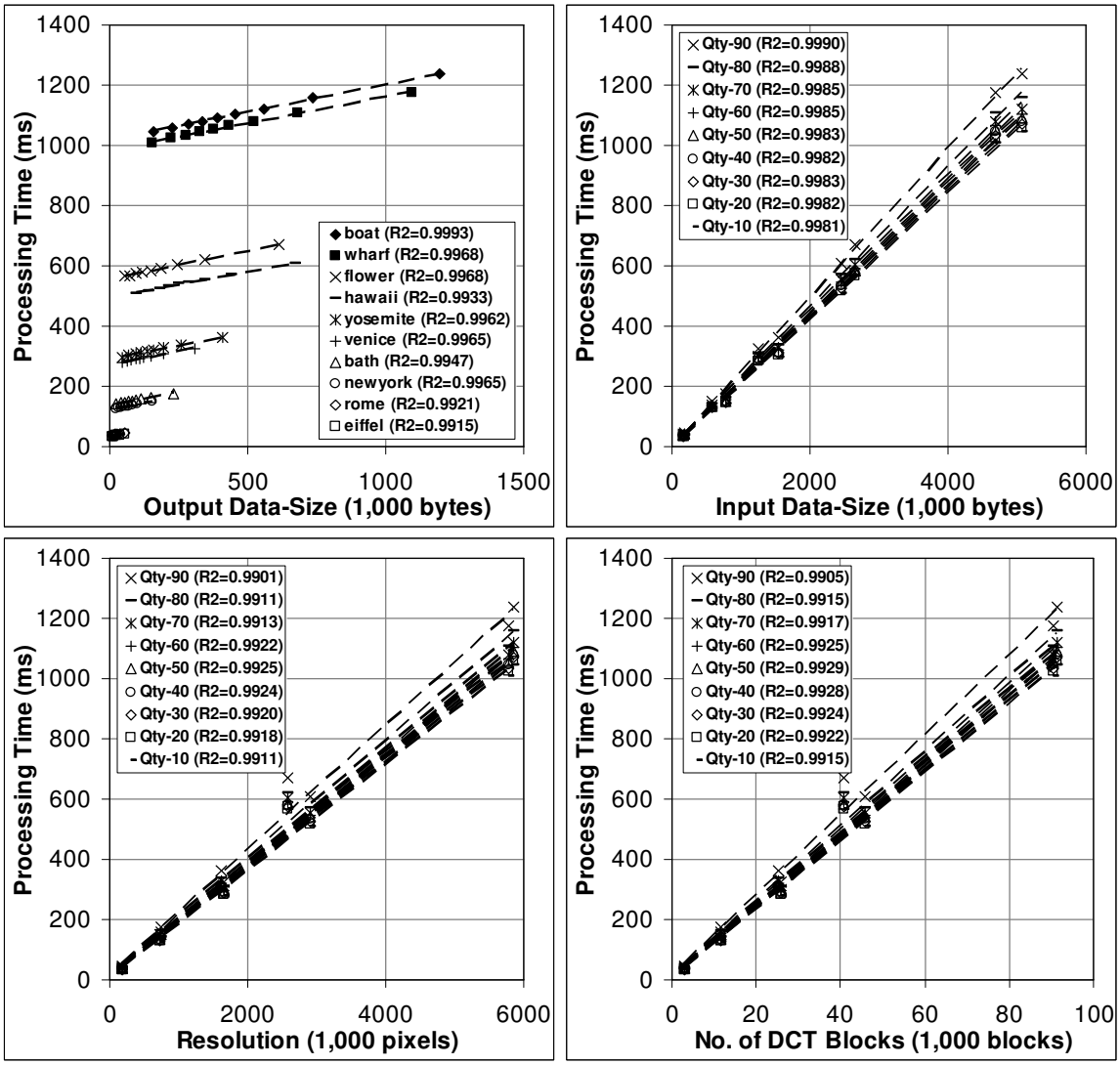


Figure 6.5 Processing-times of transcoding JPEG images in quality aspect vs. indicated factors

For the experiments with the JPEG images, the outcomes are drawn against four indicated factors: output data-size, input data-size, resolution (width × height), and number of DCT blocks. The resulting graphs are shown in Figure 6.5. For each group of data-points in the graphs, a trend-line is plotted and its coefficient of determination (R^2) is figured out. The coefficient of determination gives us the percentage of the explained variation (i.e., the trend-line) compared to the total variation of the data-points; quite

simply, it indicates the linearity of the data-points (scale 0 to 1, with 1 indicating very linear). The coefficients of determination are attached to the respective data-point groups in the graph's legend. Now let us analyze one by one the graphs in Figure 6.5.

In the first graph (top-left), the processing times are drawn against the output data-sizes. The data-points are grouped according to their input images (e.g., boat, wharf, flower, and so on). As seen in the graph, the trend-lines have quite similar slopes; they appear parallel to each other. The bigger the input image's data-size, the higher is its trend-line's position in the graph. In the remaining graphs, the processing times are drawn against the input data-sizes (top-right), the resolutions (bottom-left), and the numbers of DCT blocks (bottom-right). The data-points are grouped according to their targeted quality values, ranging from 90 to 10. The trend-lines in the graphs seem converging, where the trend-line of a higher quality value's group has a steeper slope. In all four graphs, the coefficients of determination are considerably high (above 99%), suggesting a strong correlation between the processing time and each of the four indicated factors. The input data-size and the output data-size seem to be the most influential factors since their average coefficients of determination are higher than those belonging to the other two factors (i.e., resolution and number of DCT blocks).

The processing times of modulating the JPEG 2000 images are also drawn against four indicated factors: output data-size, input data-size, resolution, and number of code-blocks. Figure 6.6 shows the resulting graphs. Like what is done to the previous graphs, the trend-lines are plotted and the coefficients of determination are figured out in the graphs.

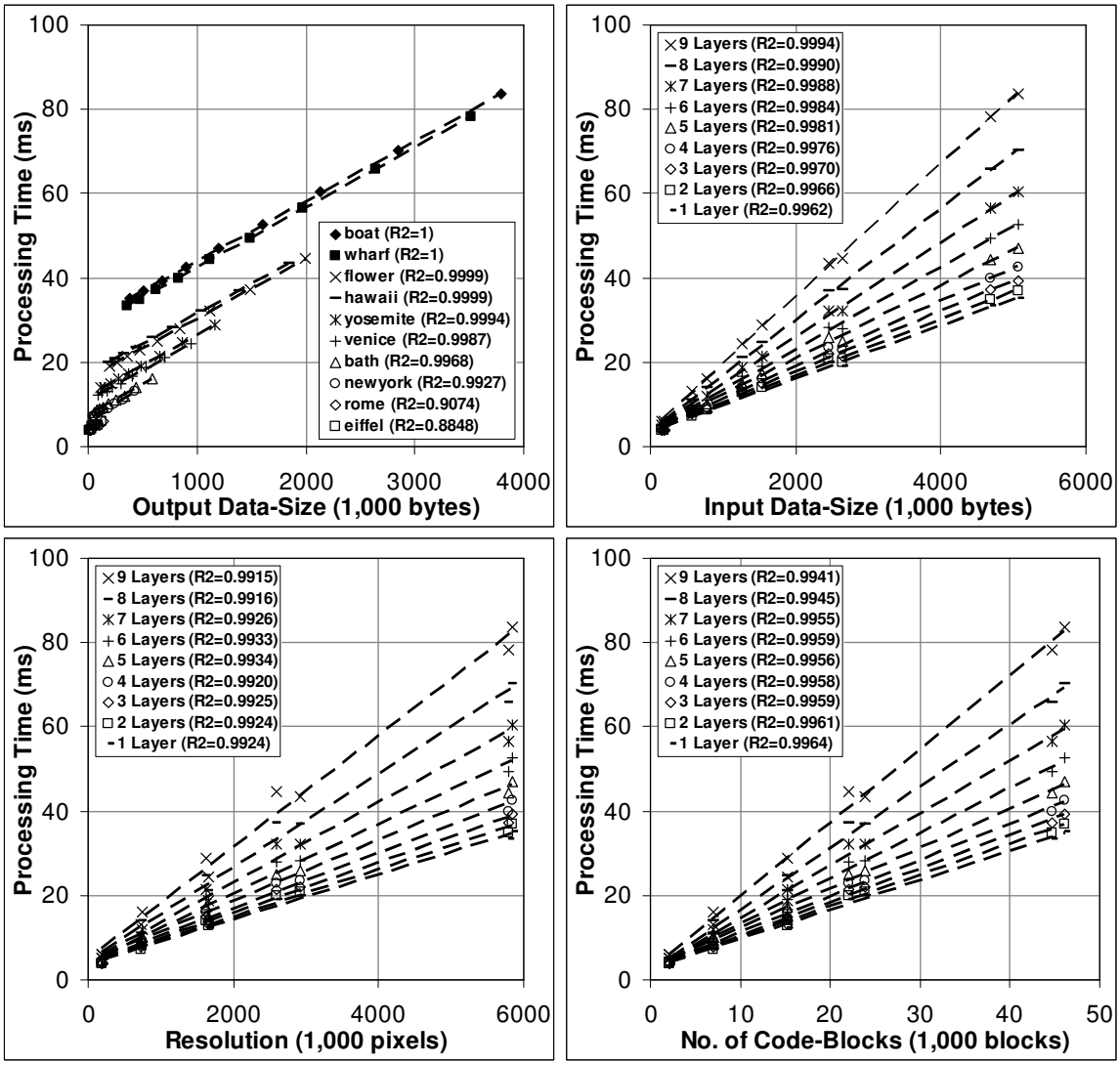


Figure 6.6 Processing-times of modulating JPEG 2000 images in quality aspect vs. indicated factors

Drawing the processing times against the output data-sizes (the top-left graph) gives parallel trend-lines, too. While the trend-lines of the first four images have very high coefficients of determination (almost 100%), the coefficients of determination deteriorate in the trend-lines of the last four images. This is because in the smaller-sized images the differences of processing times between two adjacent data-points are so small (i.e., less than 1 ms) that they cannot be distinguished definitely. Also of interest to note is

that the trend-line of image `hawaii.jp2` is above that of image `flower.jp2`, albeit their input data-sizes should indicate the opposite tendency. It seems there is another factor affecting the results. We speculate that the other factor is the number of code-blocks since image `hawaii.jp2`'s number of code-blocks is greater than that of image `flower.jp2` (kindly refer to the last column of Table 6.1). In the remaining graphs, the data-points are grouped according to the resulting numbers of layers, ranging from 9 to 1. Drawing the processing times against input data-sizes (the top-right graph), resolutions (the bottom-left graph), and the numbers of code-blocks (the bottom-right graph) results in convergent trend-lines. Generally the coefficients of determination in all four graphs are very high (above 99%), except for the trend-lines of the last two images in the top-left graph. Thus, there is a strong correlation between the processing time and each of the four indicated factors. The most influential factors seem to be the input data-size, the output data-size, and the number of code-blocks.

It can be inferred from the results of the JPEG transcoding (FDT) and the JPEG 2000 modulation above that the delay (i.e., processing time) of adapting an image by means of a downscaling operation in quality aspect is mainly determined by the input image's and output image's data-sizes, besides other lesser factors like the image's number of blocks (either DCT blocks or code-blocks) and resolution. This corroborates our claim in the previous section (notice the input parameters of operation D_d).

6.3.2 Adaptation Delay in a Upscaling Operation

The analysis here is exclusive to modulation since transcoding operations are not reversible. Modulation in the JPEG 2000 standard is again used for illustration purpose. To enhance a JPEG 2000 image's representation, application `JP2Joiner` – which carries out the upscaling operation – is employed. Image `boat.jp2`, which creation has been described in Subsection 5.2.2, was used as the test data in the experiments. Of the image, nine representations could be generated, each of which had a particular number of (quality) layers ranging from 1 to 9. Each representation was then enhanced by different but fitting supplements, and as a result, different representations were produced. The representation and each of the supplements were combined by application `JP2Joiner`, and the processing times were noted. The resulting adaptation delays (i.e., processing times) presented in Table 6.2 have been averaged from several runs.

Table 6.2 Processing times of enhancing various representations of image `boat.jp2`

Origin Rep.	Targeted Rep.								
	L=2	L=3	L=4	L=5	L=6	L=7	L=8	L=9	L=10
Layer = 1	3.0 ms	3.3 ms	4.0 ms	5.0 ms	6.4 ms	9.0 ms	14.1 ms	22.0 ms	37.9 ms
Layer = 2	N/A	3.3 ms	4.0 ms	5.0 ms	6.5 ms	9.0 ms	14.1 ms	22.1 ms	38.0 ms
Layer = 3	N/A	N/A	4.0 ms	5.0 ms	6.3 ms	9.1 ms	14.0 ms	22.1 ms	37.9 ms
Layer = 4	N/A	N/A	N/A	5.0 ms	6.6 ms	9.0 ms	14.1 ms	22.1 ms	37.8 ms
Layer = 5	N/A	N/A	N/A	N/A	6.4 ms	9.1 ms	14.0 ms	22.0 ms	38.0 ms
Layer = 6	N/A	N/A	N/A	N/A	N/A	9.0 ms	14.0 ms	22.0 ms	37.8 ms
Layer = 7	N/A	N/A	N/A	N/A	N/A	N/A	14.0 ms	22.0 ms	38.0 ms
Layer = 8	N/A	N/A	N/A	N/A	N/A	N/A	N/A	22.0 ms	37.4 ms
Layer = 9	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	37.4 ms

Note: N/A = Not Applicable

The first column of Table 6.2 lists the image `boat.jp2`'s representations that were enhanced by the suitable supplements. Each representation is identified by the number of layers it has. The remaining columns are the resulting representations, each of which is identified by the contained number of layers, as well. Hence, the join operations are to enhance the representations on the first column (labeled as the origin representations) to become the representations indicated by the headers of the remaining columns (labeled as the targeted representations). Some operations are marked "not applicable" in the table as they cannot be done.

As observed in the table, the adaptation delays are similar for the join operations in the same column. It means that the adaptation delays depend on the targeted representations. In other words, the delay of adapting an image by means of an upscaling operation in quality aspect is determined by the output data-size. Once more, this finding confirms our claim in the previous section (notice the input parameter of operation D_u).

Although adapting an image in other aspects (i.e., resolution, color component, etc.) is not examined here, by learning from the experiments above and in the previous chapter, it is likely that the image adaptation in other aspects would yield similar results. Understanding the determining factors can help predict the adaptation delay. More realistically, if the trend-line's equation of an adapting process can be figured out, the adaptation delay can be easily determined.

6.4 Proposed Framework

In order to serve Web clients with heterogeneous devices and varied preferences, a supplementary framework to the existing Web content delivery is required. The framework should be able to deliver Web contents to the heterogeneous clients effectively and efficiently. The fundamental element of the framework is the fine-grained, scalable data model, which has been discussed and detailed in Chapter 3. Employing the data model, a Web object can be adapted in a simple manner and in a short time. In addition, the data model supports high data reuse, in which not only can the object's representation – resulting from an adaptation – be used to serve the subsequent client requests for either the same or a lower-fidelity representation but it can also be used to construct a higher-fidelity representation. Based on the data model, the novel adaptation – called modulation – has further been illustrated and evaluated using the JPEG 2000 image standard in Chapters 4 and 5.

The proposed data model and adaptation might not be that useful unless they are supported by a new paradigm in the Web content delivery. The new paradigm involves re-defining the server's and proxy's roles in serving the client requests. Thus, modifications on the current system architecture are inevitable.

6.4.1 System Architecture

The findings in Section 6.2 have revealed that the server-based adaptation is more beneficial in serving the first-time client request, whereas the proxy-based adaptation is more beneficial in serving the subsequent client request. To gain the most benefits for all

parties involved, it is best if the origin server and the proxy can collaborate in delivering the adapted Web objects. If the adaptation is done by the origin server only, the delivery of the adapted Web object may not be efficient since the adaptation's result may not be used to serve other client requests and, therefore, the adaptation may have to be done for every request. On the other hand, adapting a Web object at the proxy without the original server's involvement may diminish the object's semantics. Hence, collaboration between the server and the proxy is the best way to address the problems. Depending on the circumstances, the adaptation may be carried out either by the server or by the proxy.

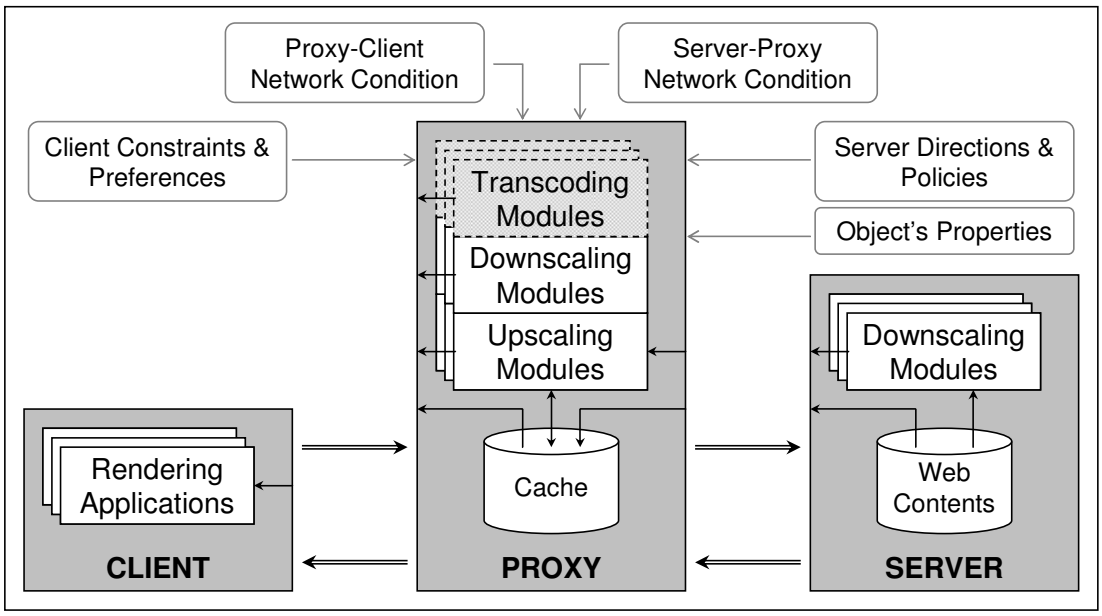


Figure 6.7 System architecture of pervasive Web content delivery

Figure 6.7 shows the proposed system architecture for delivering Web contents to heterogeneous clients. The supporting factors of pervasive Web content delivery, listed in Section 2.4, are taken into consideration in building the system architecture. In general, it does not differ much from the current system architecture. The client sends a request for a particular representation of a Web object to the original server via the proxy. Initially

there is no representation of the Web object found in the proxy's cache. Hence, the proxy passes the client request on to the server. The server replies to this first-time client request by generating the requested representation of the Web object and sending it back to the proxy. The proxy then stores the representation in its cache while simultaneously passing the representation on to the requesting client. For the subsequent client requests, asking for the same Web object, the representation in the proxy's cache can be used to serve them, perhaps with additional data from the server to enhance the representation.

The challenge in the system architecture is how and which party determines the best-fit representation of a Web object for a particular client. To address this challenge, Knutsson et al. [KnLM03] have clarified the roles of the three parties involved in the content adaptation; their roles are as follows

- The origin server controls the semantics of the adapted content.
- The client controls the presentation of the adapted content.
- The proxy controls the resources used for adaptation, caching, and network traffic.

Slightly different from the Server-Directed Transcoding proposed by Knutsson et al., in our proposed system architecture, the proxy – as opposed to the server – is the decision maker, although the server still gives its directives. The proxy collects the server's and the client's directives – in addition to other information like network traffic's condition – and then decides on the best-fit representation of the Web object for the corresponding client. Based on the decision, the proxy subsequently serves the client by reusing the cached representation, if one exists and if possible, or by consulting the server. This is where the collaboration between the server and the proxy is brought into play.

The adapting modules and the meta-data involved in the system architecture are elaborated further in the following subsections.

6.4.2 Adapting Modules

Since the server is involved in the adaptation, it is equipped with downscaling modules. The downscaling modules are used to reduce the building blocks (i.e., fragments, layers, packets, or units) of a Web object. Referring to the data model in Chapter 3, the downscaling modules comprise the selection, inverse-selection, and translation operations. Hence, the server can generate various representations as well as supplements of the Web object. Some Web objects may not need to be adapted and can be sent out directly, as depicted by the left arrow coming out from the Web Contents in the system architecture.

The proxy stores the representation of the Web object in its cache (depicted by the right arrow coming in to the proxy's Cache in the system architecture). The cached representation may be used to serve the following requests for the same Web object. There are three possibilities of serving the cached representation to the clients:

1. The cached representation may be sent out as it is. In that case, the cached representation is already the best-fit representation for the requesting client (depicted by the left arrow coming out from the Cache).
2. The cached representation may need to be enhanced to fulfill the client request. Hence, upscaling modules are required by the proxy to construct the enhanced representation. The upscaling modules comprise the join and, perhaps, translation

operations. As depicted in the system architecture, the associated supplement – generated by the server's downscaling modules – is fetched from the origin server and combined with the cached representation using the proxy's upscaling modules.

3. A lower-fidelity representation than the cached one may be requested by the client. The cached representation, therefore, is downscaled by executing one or more downscaling modules at the proxy.

Remind that modulation is an exclusive process (kindly refer to Section 3.4). It is possible that the client device is not equipped with the application to render the scalable data-format, so the proxy may still need transcoding modules to convert the data-format. However, the transcoding modules are not our main concern, and therefore, they may be set aside (shaded in grey) in the system architecture.

6.4.3 Supporting Meta-Data

As mentioned earlier, the server's and the client's directives are required by the proxy to select the best-fit representation of a Web object for the particular client. The directives come in the form of meta-data. The meta-data may be separated from the Web (HTTP) requests and responses, so that they can be stored in the proxy's cache and used in subsequent requests. The system architecture in Figure 6.7 displays different types of meta-data, which are detailed below.

Clients may access the Web objects with heterogeneous devices. Each device has particular characteristics in terms of dimensions, number of colors, multimedia support, and many others. These characteristics, known as the client device's constraints, should

be sent to the proxy to assist it in making decision. Besides those constraints, sometimes the client has certain preferences like acceptable latency delay, multimedia interest, and so on. The client's preferences may as well be conveyed to the proxy.

Every Web object residing in the server has particular properties such as its width and height, supported color space, number of bits per pixel, and other else. The object's properties are required by the proxy to properly adapt the object. Some of the object's – perhaps intangible – properties may need to be preserved during adaptation, such that the semantics of the object is duly preserved. For that purpose, the server (i.e., the content provider) should give its directions regarding how the object should be adapted. For an instance, the server may dictate the maximum amount of quality reduction that can be applied to an image to maintain the image's clarity. Also, the server may have policies that the proxy must/should follow. The policies often help maintain the object's integrity. Examples of the policies are the adapting modules that must/should be employed, the time-to-live (TTL) of the resulting representation that must/should be observed, and so forth. All of these are the meta-data typically gathered from the server's side.

The last type of meta-data is the network traffic's condition. Since there are two network segments connecting the client to the server, the network traffic's condition can be observed on the proxy-client and server-proxy links. The network traffic's condition – comprising, among others, the bandwidth (i.e., network throughput) and the roundtrip time (i.e., network latency) – is useful for predicting the overall latency delay (response time) of the Web access, as already illustrated in Section 6.2. Based on the predicted latency delay, the proxy may determine the best-fit representation to meet client preferences like the acceptable latency delay.

6.5 Related Work

Previous work has proposed varied frameworks to address the issue of Web clients' heterogeneity. Since adaptation is the ultimate answer to the issue, the proposed frameworks cover a range of adaptation and transcoding systems. Most of them have been mentioned and discussed in the previous chapters. Here, some of them are highlighted and, if necessary, contrasted with our solution.

Han et al. [HaBL98] and Chi and Cao [ChC02] have previously analyzed the conditions when it is beneficial to perform transcoding. Both focused their analyses on a proxy-based transcoding system. While the former analyzed the benefit of a traditional transcoding proxy, the latter evaluated that of an improved transcoding proxy which can store and reuse partial objects. Compared to the previous studies, our evaluation was broader, taking both proxy- and server-based approaches into consideration. Moreover, various scenarios were also involved in our evaluation. This extensive study makes us see the benefits of each approach. Based on the evaluation, a framework that can utilize both approaches' benefits was proposed. The framework emphasizes the collaboration between the server and the proxy as the condition to attain efficient Web content delivery. Apparently, modulation – the novel, scalable adaptation – can fit well into the framework.

The need for information about the client's constraints and preferences was also recognized by previously proposed transcoding systems. GloMop [FoGB96] requires the client device to be equipped with a special browser, through which the client can specify its preferences. In Mowser [JoWM96, BhJA98], the client stores its preferences directly in the transcoding proxy. TransSquid's [MaSR02] client uses CC/PP (Composite

Capability/Preference Profiles) to communicate its capabilities and preferences to the proxy. Lastly, Server-Directed Transcoding (SDT) [KnLM03] adopts Media Feature Sets, which the client may employ in its requests to express its capabilities and preferences. Our framework was not confined to any specific method. That issue is better addressed in the implementation.

Although information about the client's constraints and preferences helps the transcoding system in serving the client with the best-fit presentation, the transcoding system should also acquire information about the requested object so that it can transform the object properly without destroying essential information. Mogul [Mog01] observed that existing transcoding systems relied on implicit information, such as the HTTP `Content-Type` header, to transform the requested object. Such transformation may undermine the object's semantics and disturb end-to-end information transfer from the server to the client. To address this problem, Mogul proposed the use of explicit server's directions to transform the object. The directions are put in the responses' header. Sharing Mogul's idea, we have included two kinds of meta-data – namely the object's properties and the server directions and policies – in the framework. The specific method employed is again up to the implementer.

Most transcoding systems face the predicament of storing the original or transcoded objects in their cache. If it is cached, an original object can be reused many times and transcoded into various representations. Caching different representations, on the other hand, may avoid the need to run the delaying transcoding processes repeatedly, but they occupy a large amount of the cache's space. In addition, a representation is usually so specific to particular client characteristics that it is difficult to reuse the

representation for serving other clients; in other words, caching it may be less beneficial. TransSquid has a unique way to improve the cache's performance. It divides its cache's space into several levels in accordance with the client devices' categories. A higher-fidelity representation, stored in a higher-level cache, can be transcoded into a lower-fidelity representation and the result is stored in a lower-level cache. Our framework does not need such division in the cache's space, and only one representation of a Web object is kept in the cache. The cached representation may get bigger – and better – when a more sophisticated client requests it, yet the cache still keeps the one and only representation, thanks to the reversible property in the adaptation's mechanism.

ICAP [ElC03], a well-known adaptation framework, is a lightweight protocol for executing a "remote procedure call" on HTTP messages. An ICAP client (usually a surrogate) may redirect a client's request or a server's response to an ICAP server for some sort of transformation or adaptation. By doing this, the ICAP server can off-load the burdens of the origin servers. Alas, the current ICAP only offers a means for communication between the ICAP client and the ICAP server. An application framework, which is another important component in an adaptation, is yet to be defined. Our proposed framework can be considered as an application framework, which is specific to dealing with the clients' heterogeneity. So, ICAP may be used to realize our framework.

A more general framework for edge services, OPES [BaPC04] offers a standard, cooperative way to perform a data stream service between a provider (i.e., the origin server) and a consumer (i.e., the client). It supports a wide range of transformation services. Its architecture includes components like OPES processors and callout servers. Some cooperative OPES processors may be put in between a provider and a consumer to

perform various transformation services; each OPES processor may distribute the responsibility of service execution to one or more callout servers. Communication between the two components is through the OPES Callout Protocol (OCP). Similar to the previous assertion, our system architecture may be implemented as a transformation service in the OPES framework.

Compared to the past adaptation and transcoding systems, our proposed framework has the following advantages:

1. "On-demand" Web content delivery.

   The adaptation can be carried out at the server's or the proxy's side, wherever it is more efficient. As the analyses in Section 6.2 have revealed, the server-based adaptation is more beneficial in serving the first-time client request, whereas the proxy-based adaptation is more beneficial in serving the subsequent client request. The on-demand Web content delivery, suggested by the framework, bases its actions on those analyses. In the first-time client request, the proxy determines the best-fit representation of a Web object for the client and asks the server to adapt the Web object accordingly. The proxy serves subsequent client requests with the cached representation and only consults the server if more data (supplements) are required. Thus, only required data are delivered from the server to the proxy and from the proxy to the client. The outcome is fast and efficient Web content delivery.

2. Scalable service.

   The adaptation service provided by the framework is greatly scalable. It can be implemented in multiple proxies, even if they are arranged in a hierarchy. The adaptation service will not tax the proxies' performance. On the contrary, the on-

demand Web content delivery – which also can be applied to the proxy-to-proxy data transfer – may improve their performance in serving heterogeneous Web clients.

3. Detailed meta-data.

   The framework involves a variety of meta-data that are needed to correctly provide the best-fit presentation for a particular client and to effectively preserve the end-to-end semantics of the Web objects. As mentioned earlier, the rich meta-data – unsupported by the current Web protocol, i.e., HTTP/1.1 – may be detached from the client's request and the server's response and be fetched separately. To circumvent repeated access to the server where the meta-data resides in, the meta-data may be stored in the cache, too.

The framework specified in this chapter is still very general. Many issues – such as how the meta-data look like, what modifications required in the server application, and how to apply modulation to the proxy application – are left unanswered. The next chapter details our experience in developing a model prototype based on the framework. Realistically, developing the model prototype helps us address those issues.

Chapter 7

Model Prototype of Pervasive Web Content Delivery

The development of a model prototype of the pervasive Web content delivery is discussed in this chapter. The prototype was built based on the framework proposed in the previous chapter. This chapter begins with assessment of the incomplete elements in constructing the prototype. The necessary specification, development, and enhancement to complete the model prototype are then discussed in detail. The prototype serves as a proof of concept of the proposed framework. Instead of emphasizing the completeness of the framework's features, we emphasize its efficacy. The main contribution of the prototype is the efficiency in pervasive Web content delivery.

7.1 What Do We Have So Far?

As mentioned before, the fundamental element of the framework is the data model which, if it is employed in the Web objects' data-formats, can help generate representations of the Web objects in a fast and efficient manner. Modulation, a novel adaptation based on the data model, has been devised and further implemented. For the implementation, the JPEG 2000 standard has been used as an illustration in this thesis, as described in Chapter 4. Thus, the adapting modules of the framework have been materialized.

The adapting modules need be integrated into the origin server and the proxy. Apart from the integration, the server's and the proxy's behavior in response to the client requests may need to be modified, as well. Using the right adapting modules, the server should be able to generate on-the-fly either a representation or supplement of the requested Web object. Besides passing on the representation – received from the server – to the client, the proxy should be able to reduce or enhance a representation stored in its cache, depending on the circumstances. In short, the current server and proxy applications have to be customized to incorporate the new framework.

The last but equally important element is the meta-data, which help the proxy in choosing the best-fit Web object's representation for a particular client. As described in the proposed framework's system architecture, there are a variety of meta-data required by the proxy. The meta-data are collected from the origin server, the requesting client, and maybe other sources on the Internet. However, the meta-data may come in different

formats and that needs to be addressed properly. Unless the meta-data are standardized, the three parties (i.e., client, server, and proxy) may be lost in communication.

To sum up, things that need to be done – in orderly fashion – to complete the model prototype are: 1) the supporting meta-data have to be standardized and clearly defined; 2) the server application has to be extended to integrate the adapting modules; and 3) the proxy application has to be extended to integrate the adapting modules, accommodate the decision-making process, and improve the cache's efficacy. The rest of this chapter discusses the development of these three elements.

7.2 Meta-Data Specifications

Basically there are three types of meta-data required by the proxy for the decision-making process. The meta-data originating from the client informs the proxy about the client device's constraints and the client's preferences. The meta-data gathered from the server gives information about the object's properties as well as the server's directions and policies. Another meta-data conveys the network traffic's condition on the proxy-client and server-proxy links. The last meta-data is discussed first in the following paragraphs.

The network traffic's condition primarily provides information about network latency and network throughput. The network latency may be determined in a few ways:

1. By using the roundtrip time (RTT), which is the amount of time required by a packet to travel from the sender to the receiver and then back to the sender. The RTT can be measured by running the `ping` utility, which is supported by almost every known

operating system. The ICMP (Internet Control Message Protocol) packet sent by `ping` is time-stamped. When the sender receives back the packet, the RTT is calculated as the elapsed time between the sending and receiving events. The network latency in one direction (from the sender to the receiver, or vice versa) is more or less half of the RTT.

2. By using the packet's time stamp. This may be done in the protocol layer above the transport layer (TCP/UDP). For instance, by calculating the time difference between the current time and the time stated in the `Date` header of an HTTP packet, the latency time required to deliver the packet from the sender to the receiver can be determined. However, the two systems – sender and receiver – should synchronize their time, so that an accurate result can be obtained; NTP (Network Time Protocol) application can be used to synchronize their time to a particular NTP server or to the global time. Another drawback is that it can only estimate with a 1-second precision.
3. By an assist from a global infrastructure comprising distributed servers. The idea is to have the dedicated servers measure each other distances[24]; this may also be pre-determined. The servers are, in turn, used as a reference to measure the distance between two hosts. Some proposed systems that follow this idea are IDMaps [FrJJ01] and NPS [NgZ04] (previously known as GNP [NgZ01]). In IDMaps, the nearest server is determined for each host. The distance between two hosts can be calculated as the sum of the distances from the hosts to their nearest servers and the distance between the two servers. In NPS, the host probes some reference servers to determine

[24] Here, the distance means the latency time on the network connection between two hosts/servers.

its position in the Euclidean space. By obtaining the other host's position, the (former) host can calculate the distance between them.

4. By an assist from an existing global infrastructure. This is similar to the method in point 3. The only difference is, instead of creating a new infrastructure, it exploits an existing one. King [GuSG02] estimates the latency between two hosts by measuring the latency between nearby DNS servers. Like GNP and NPS, Vivaldi [DaCK04] also uses the Euclidean space model to estimate the latency between two hosts. Vivaldi, however, can base its estimation on the RTT data gathered from any existing infrastructure, like DNS and peer-to-peer systems.

The first method requires an additional packet (i.e., an ICMP packet) to determine the network latency, whereas the second method does not since the network latency can be determined from the HTTP packet (either a request or a response) itself. Yet, the second method is imprecise and requires time synchronization between the sender and the receiver. The last two methods can estimate the network latency almost immediately but by means of indirect measurement; the error rate is still relatively high. Either way, it is difficult to obtain the network latency accurately in real time due to several reasons. Firstly, the packet routing on the Internet is rather dynamic. Two packets sent from one node to another may take different routes. Therefore, the network latency between two nodes on the Internet may vary over time. Furthermore, the sending and receiving packets may also take different routes, causing different latency in opposite directions. Secondly, depending on network load, the queuing delays in network routers are varied. When the network load is high, a packet may stay for a longer time in the router's queue buffer, resulting in high network latency. The more router-hops a packet traverses, the greater is

the variability of the network latency. Lastly, packet loss may contribute to high network latency, too. If the network traffic is too congested, the router starts discarding some packets at random to ease the congestion. A replacement packet then may need to be sent, and as a result, the overall network latency may increase. Since the network latency changes frequently, it would be better if the network latency is measured based on a number of packets. The downside is that the cost is too high for a short-time connection.

The network throughput refers to the rate of data transfer – in bits per second (bps) – between two nodes on the Internet. The network throughput is different from the bandwidth capacity. Often the network throughput – also known as the effective bandwidth – is much lower than the bandwidth capacity due to several reasons. Firstly, the protocol overhead, such as the header's data-size and control overhead, makes the actual data being delivered smaller. Since the protocol overhead is generally not considered in the evaluation, the network throughput is less than the bandwidth capacity. Secondly, packet collision and packet loss can further reduce the network throughput. Lastly, the network latency may affect the network throughput, too. This is true in the case of TCP, which requires acknowledgement for the packets sent. As the network latency increases, the sender may spend lots of time waiting on acknowledgements instead of sending packets. Apparently, the network throughput is even more difficult to measure than the network latency above. There are two ways of measuring the network throughput:

1. By querying network devices for stored information. Most network devices, like routers, store basic network information about the traffic on the device. The information includes the data rate capacity of each interface, the number of data-bytes

transmitted and received per interface, the number of erroneous and discarded packets on the interfaces, and so forth. The information may be queried using SNMP (Simple Network Management Protocol) periodically, so that the network throughput can be deduced. The drawback is that many network providers do not allow us to access this information due to security concern. Moreover, this information is just from a network segment, whereas a packet may have to traverse many segments of the Internet before reaching its destination. Querying all the information from many network devices at a time is quite strenuous. Even if all of the network information is obtained, the network throughput is anything but difficult to deduce.

2. By sending some test packets and measuring their transmission delays. The network throughput is then determined from the packets' sizes and the associated transmission delays. A variety of techniques have been proposed based on this idea. Generally they can be classified into two categories: one-packet and packet-pair. The one-packet technique assumes that transmission delay varies linearly with packet size while the latency remains constant for different packet sizes. By sending a large number of packets of different sizes along the network path, their transmission delays will approximate a line whose slope is the inverse of the network throughput. Some tools based on the one-packet technique are pathchar [Jac97], clink [Dow99], and pchar [Mah99]. The packet pair technique sends two large same-sized packets consecutively with the expectation that they are positioned one after another at the bottleneck queue. After traversing the bottleneck link, the time dispersion between the two packets linearly corresponds with the bottleneck link bandwidth (i.e., the network throughput). Some tools employing the packet-pair technique are bprobe/cprobe [CaC96], nettimer

[LaB01], pathrate [DoRM01], and SProbe [SaGG02]. Besides the two main categories, other tools worth noting here are NWS [WoSH99] and pathload [JaD02]. NWS measures the link bandwidths among machines in a distributed system through active sensors; each sensor resides in a single machine and probes other sensors (in other machines) periodically. Similarly, pathload also sends a fleet of packets regularly to measure the network throughput between two end-hosts. Further information about tools for throughput measurement can be found in CAIDA's Webpage[25].

Either way, the network throughput can only be obtained after some processes are done and calculations made. Thus, there may be a delay in obtaining the network throughput.

Both network latency and throughput are helpful for the decision-making process to select the best-fit representation of a Web object. However, it is extremely difficult to obtain their precise values due to the dynamic nature of the Internet traffic. In addition, there is no way to obtain their values before hand; that is, they can only be evaluated after the associated sender and receiver are determined. Another unresolved issue is which party (or, parties) – client, server, or proxy – is (are) supposed to provide the network information, since different parties may supply different information. Owing to these problems, the meta-data carrying information about the network traffic's condition is excluded from the model prototype. Taking its place in the decision-making process, other properties (such as the maximum object's size) from the client meta-data may be used in the model prototype, albeit they are different.

---

[25] http://www.caida.org/tools/taxonomy/performance.xml#bw

The two remaining meta-data are those coming from the client and the server. The client meta-data is detailed in the next subsection, followed by description of the server meta-data.

7.2.1 Client Meta-Data

As mentioned earlier, the client meta-data should carry information about the client device's capabilities (or perhaps, constraints) and the client's preferences. The purpose of sending the meta-data is to help the proxy select the best-fit representation of a Web object for the respective client. The HTTP/1.1 standard [FiGM99] has introduced some request-header fields such as `Accept`, `Accept-Charset`, `Accept-Encoding`, and `Accept-Language` for indicating the client's preferences. However, the request-header fields are basically used to indicate a preference for a particular object over other objects having the same context but different data-formats, character sets, or languages. They do not give a clear description of the client device's capabilities. To resolve this problem, IETF (Internet Engineering Task Force) proposed Media Feature Sets [Kly99a, Kly99b]. They provide an extensible way of describing the client device's capabilities. The syntax of Media Feature Sets also allows a complex expression using Boolean operators to combine individual predicates.

Meanwhile, W3C (World Wide Web Consortium) released CC/PP (Composite Capability/Preference Profiles)[26], which is a profile describing device capabilities and user preferences. CC/PP is developed based on RDF (Resource Description Framework),

---

[26] http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/

a general purpose meta-data description language. The XML syntax for RDF is called RDF/XML. Commonly, CC/PP is expressed in the RDF/XML syntax. CC/PP is compatible with IETF's Media Feature Sets in the sense that all media feature tags and values can be expressed in CC/PP, but not all CC/PP profiles can be expressed as media feature tags and values. However, CC/PP does not have mechanisms matching those in the IETF media feature framework to express certain comparisons (e.g., `pix-x<=640`) and complex expressions (e.g., `pix-x=640 & pix-y=480 | pix-x=800 & pix-y=600`).

Owing to its extensibility and user-friendliness, CC/PP was preferred to Media Feature Sets for expressing the client meta-data. The CC/PP schema can be found in site http://www.w3.org/2002/11/08-ccpp-schema. According to the schema, a CC/PP profile has one or more components, and each component contains one or more attributes. Examples of the client device's components are hardware platform, software platform, and individual application (like browser). The client capabilities and preferences are described by the components' attributes. The CC/PP schema has a mechanism where the attributes of a component may be specified by reference to a default profile, which may be stored separately and accessed using its specified URI. When an attribute is specified in the component, but is also specified in the referenced default profile, the directly defined attribute value takes precedence. If necessary, new attributes can be added easily to CC/PP by specifying an extension schema.

Each client should specify its capabilities and preferences in a CC/PP document and deliver it to the proxy. There are several ways for the client to deliver the CC/PP document; those are:

1. The client attaches the CC/PP document (in its entirety) to the request's header. However, the XML-based document may need to be translated into a compact form (like the compact policy in P3P) to suit the HTTP header field's style; otherwise, the HTTP header may become too bulky.
2. Instead of attaching the CC/PP document to the request's header, the client embeds it in the request's body. In that case, POST – rather than GET – would be better used as the request's method since the action is pretty similar to the POST-based form submission. The downside is that the request's data-size is bigger than usual, and since the CC/PP document is embedded in every request, it may slowdown the requests' delivery.
3. The client just puts a reference to the CC/PP document (i.e., its URI) in the request's header, whereas the CC/PP document itself can reside in any Website (perhaps, of the client device's vendor). Different from the other two alternatives before, the client's meta-data cannot be found in the client's request. Only the URI of the client's meta-data is available in the request's header. While this alternative offers the least modifications to the HTTP request, there is an extra roundtrip required by the proxy to fetch the client's meta-data.

Among the three alternatives, we chose the last alternative to be implemented in the prototype. Four reasons can be suggested here. Firstly, it is simple and requires just a small modification to the client's browser. Secondly, by placing the CC/PP document in a public Website (let us say, the device vendor's site), the device's characteristics specified in the document can be shared with many clients. If a user wants to specify his/her preferences, he/she can create a new CC/PP document, in which he/she may indicate the

vendor's CC/PP document as the default and change any property as he/she likes (if there is an overlapping property, the user-specified one will take precedence over the default). He/she then needs to place the new CC/PP document in a public Website and refers to it in the request's header. So, the third reason is that the chosen alternative is also flexible. And lastly, since the CC/PP document is separated from the client's request, it may be cached in the proxy and used for subsequent requests, either of the same client or of other clients having the same device's characteristics. This way, the extra roundtrip is required only for the first request. URI of the CC/PP document is given in an HTTP extension header called `CCPP`.

7.2.2 Server Meta-Data

The server meta-data needs to be generated for every Web object residing in the server. In the designed prototype, it basically tells two things to the proxy. The first is the object's description containing the attributes (properties) of all alternative representations, such as content-type, height, width, data-size, supported color, quality, and so on. Based on this detailed information about the object's representations, the proxy may select the best-fit representation for a particular client. The second thing that the meta-data tells the proxy is the adapting module(s) used to generate the object's representations. The adapting parameters applied to the module(s) for each generated representation may also be included in the meta-data.

The HTTP/1.1 standard [FiGM99] does not say much about the Web object's description. Maybe it is because the standard focuses more on the server-driven

negotiation, in which only the client's characteristics are required by the server for choosing the best-fit representation. With respect to the object's description, the standard just specifies status code "`300 Multiple Choices`", which should include an entity (i.e., the response's body) containing a list of resource characteristics and locations (URIs) from which the client can select the one most suitable; this mechanism is called the user-driven negotiation as opposed to the server-driven negotiation earlier.

The Media Feature Sets [Kly99a, Kly99b] can also be used to specify the object's description. The attributes used to characterize the client device can be squarely used to describe the object's representations. As there are a number of object's representations, the data-size of the object's description is definitely larger, or even much larger, than that of the client device's characteristics. If the object's description is to be attached to the server's response, it is better placed in the response's body; placing the large description in the response's header would look awkward. Nevertheless, Media Feature Sets is basically predicate-based processing and cannot be used to describe the content adaptation; that is, the adapting modules and parameters have to be described elsewhere.

Since there is no fitting meta-data to fulfill our needs, we devised a meta-data profile – called ADP (Adaptation Profiles) – to describe the server meta-data. Like CC/PP, ADP is based on RDF/XML. The ADP schema is presented in Appendix A. In ADP, a document can have several object-profiles (or simply, profiles), one of which should be the main object-profile. The main profile describes the original Web object, whereas the other profiles describe its representations. The schema specifies four initial profile types corresponding to four object-types: image, video, audio, and text. Each profile contains attributes for describing an object's or a representation's characteristics.

One of the attributes is `adp:representation`, which is used to list representations of the respective object. Hence, the representations' profiles – along with their attributes – should be nested within the main object's profile and neatly enclosed by the `<adp:representation>` and `</adp:representation>` tags. Alternatively, the representations' profiles can be placed outside at the same level as the main object's profile, but their URIs are still listed in the main object's profile, also enclosed by the designated tags. Another important attribute is `adp:adaptedBy`, used to refer to URI of the adapting module employed for generating the representation. The schema further specifies a resource of type `adp:Adaptor` for indicating an adapting module. Some attributes are also specified for describing the adapting module's characteristics. Just like CC/PP, the ADP schema can be extended to include more profile types and attributes.

Since every adaptable Web object should have its own ADP document – detailing its representations and characteristics – the document is better placed in the server. There are some alternatives for the proxy to fetch the ADP document:

1. URI of the ADP document is derived from that of the Web object it describes. So, the ADP document is placed in the same directory as the Web object. Suppose the Web object's URI is http://www.foo.com/images/picture.jpg, then the ADP document's URI should be http://www.foo.com/images/picture.jpg.adp. This method is simple since the proxy can make a request for a definite resource, i.e., the ADP document's URI. However, it may not work if the Web object's URI contains a query string.
2. The proxy negotiates the ADP document's URI with the server. So, the server points out where the proxy can fetch the ADP document. This may solve the problem in point 1, but it suffers from extra roundtrips for the negotiation.

3. The proxy sends the usual request for the Web object, but the server replies it with part of the object and piggybacks the ADP document with the response. The difficulty is to determine how big and which part of the object should be sent out. Furthermore, the penalty is almost the same as the first method since a second request may still be needed after the proxy makes a decision on the best-fit representation.

For the development of the model prototype, we opted for the first method. Again, the reason is because it is much simpler than the other two methods. By clearly separating the ADP document from the HTTP response, it is also easier to cache the meta-data in the proxy and reuse it for serving subsequent requests.

7.3 Modifications in Server Application

The server application employed in the model prototype is Apache[27] version 2.0.51. It is the most popular Web server on the Internet. In September 2005, the Netcraft Web Server Survey[28] found that almost 70% of the Websites on the Internet were using Apache, making it more widely used than all other Web servers combined.

Most adapting modules (adaptors), available in the public domain, are stand-alone applications, and so are our already-built adaptors (i.e., JPEG transcoders and JPEG 2000 modulators). To utilize these adaptors directly in the server, we need to develop some interfaces to link the server application with the adaptors; one interface may be needed for each adaptor. There are two options that can be considered to develop the interfaces. The

---

[27] http://www.apache.org

[28] http://news.netcraft.com/archives/2005/09/05/september_2005_web_server_survey.html

first is to use Apache API to create an adaptation's module, and the second is to use a scripting language (i.e., PHP, Perl, JavaScript, etc.) to create a dynamic Web resource. Although it may not be optimal, we opt for the second method for simplicity's sake.

We developed three interfaces – two for the JPEG transcoders (SDT and FDT) and the other for the JPEG 2000 modulators – using PHP (Hypertext Preprocessor) scripting language. The interfaces (i.e., `jpgtranscoder-sdt.php`, `jpgtranscoder-fdt.php`, and `jp2modulator.php`, respectively) can be located by certain URIs, which are accessible to the Web users. Each interface requires two inputs, namely the image to be adapted and the adapting parameters. The interface's tasks are to translate the received inputs into the adaptor's input parameters, execute the adaptor with the corresponding input parameters, and output the resulting image.

The image to be adapted can be determined from the extra path information found in the request's URI. When a client requests the interface's service, it directs the request to the interface's URI with some extra path information. For instance, the interface's URI is http://www.foo.com/adaptor/jpgtranscoder-sdt.php and the client may request its service at http://www.foo.com/adaptor/jpgtranscoder-sdt.php/images/picture.jpg, hence the extra path information is /images/picture.jpg. This extra path information can be obtained from the server-supplied environment variable `PATH_INFO` by accessing PHP predefined variable `$_SERVER`. Obtaining the extra path information, the interface in turn can determine the exact location of the image in the server.

The adapting parameters may be delivered to the interface in two ways: 1) using the query string of the request's URI, and 2) using an HTTP extension header in the request. Both methods were implemented in the interface. In the first method, the

adapting parameters are given in the request's query string (the string following '?' and containing 'param=value' pairs separated by '&'). For instance, the client requests for http://www.foo.com/adaptor/jpgtranscoder-sdt.php/images/picture.jpg?s=2&q=60&g, so the request's query string is "s=2&q=60&g". PHP automatically splits the query string into parameter names and their corresponding optional (may or may not exist) values. So, the interface just needs to access PHP predefined variable `$_GET` to obtain these passing adapting parameters. In the second method, an HTTP extension header called `ADP-Qry` is defined and used to deliver the adapting parameters. The format of the passing adapting parameters is the same as the query string's format; that is, 'param=value' pairs separated by '&'. But in this case, the interface itself has to split the content of the `ADP-Qry` header field into parameter names and their corresponding values.

Once the interface attains the requested image's location and the adapting parameters, it can translate and arrange them into the input parameters of the associated adaptor. Then, the interface can execute the adaptor and return the resulting image's representation accordingly. Thus, the interface's tasks are rather simple; in fact, each of our developed interfaces comprises less than a hundred lines of PHP instructions.

It can be noticed that the client's request looks quite awkward with the exposed interface's URI. Actually, the interface's URI can be concealed using the `mod_rewrite` Apache module. This way, the client can just request an image as normally as it requests the image's URI. If the `mod_rewrite` module determines that the requested image needs to be adapted (by checking the image extension), it will rewrite the requested image's URI with the interface's URI plus the image's path as the extra path information. As an

illustration, let us say that the image's URI (including the query string) requested by the client is http://www.foo.com/images/picture.jpg?s=2&q=60&g. Since the image extension is '.jpg', the `mod_rewrite` module then rewrites the requested URI with http://www.foo.com/adaptor/jpgtranscoder-sdt.php/images/picture.jpg?s=2&q=60&g, that is the URI of the JPEG-SDT transcoder's interface plus the image's path. Consequently, the requested image will be adapted by the JPEG-SDT transcoder according to the adapting parameters given in the query string.

Last but not least, the server should supply the correct `Content-Type` for the ADP documents it delivers. As indicated earlier, each adaptable Web object – in this case an image – is accompanied by an ADP document describing the object's characteristics and guiding the adaptation. The ADP document can be fetched from the server using the object's URI plus the '.adp' extension. For an example, an adaptable image at http://www.foo.com/images/picture.jpg is accompanied by an ADP document at http://www.foo.com/images/picture.jpg.adp. Since the content of an ADP document begins with "`<?xml`", by default the server will supply `text/xml` as its `Content-Type`. To help the proxy or the client recognize the ADP document easily, we added an *AddType* directive in the Apache configuration so that the server would instead supply `text/xml+adp` as the document's `Content-Type`.

7.4 Modifications in Proxy Application

As for the proxy application in the model prototype, we employ Squid[29] version 2.5.STABLE9. Squid is an open-source Web proxy cache, designed to run on Unix systems (including Linux). It supports FTP, Gopher, Wais, and HTTP data objects.

Among the Squid's components, there are three basic components determining its main workflow. Those components are client side, server side, and storage manager. The client side is where requests are accepted, parsed, processed, and replied. This component determines whether a request is a cache hit or miss, and takes actions accordingly. The server side is responsible for forwarding cache misses to other proxy caches or origin servers. It can serve requests of different protocols such as FTP, Gopher, Wais, but primarily HTTP. The storage manager is the glue between client and server sides. It handles objects stored in the cache. Figure 7.1 depicts the three basic components, together with some of their important routines. In the server side component, only routines for HTTP are shown. As seen in the figure, the client side component is quite complicated since there are many possibilities that a request may go into. Therefore, it needs elaboration in the following paragraph; the other two components – server side and storage manager – are quite self-explained in the figure.

Every request initially comes to routine *clientReadRequest*. After traversing some checking routines, it will arrive at routine *clientProcessRequest*, which determines whether a corresponding cached object exists. If the cached object does not exist, routine

---

[29] http://www.squid-cache.org

clientProcessMiss is called to trigger the object's retrieval from another proxy cache or the origin server. If it does exist, the cached object is fetched from the storage manager, and the control is passed to routine *clientCacheHit*. In this routine, the cached object is assessed. If it is valid and fresh, the cached object is sent to the requesting client via routine *clientSendMoreData*. Otherwise, depending on the circumstances, routine *clientProcessExpired* (if it is not fresh), *clientProcessRequest* (if the `Vary` header has to be included in the object's retrieval), or *clientProcessMiss* (for other reasons) is called. If routine *clientProcessExpired* is elected, later – after retrieving a fresh object from the server – the control is passed to routine *clientHandleIMSReply* to determine whether the old (i.e., cached) or the new object should be sent back to the client. Any replied object is sent to the client bit by bit, so routine *clientSendMoreData* may be called repeatedly during delivery of the object.

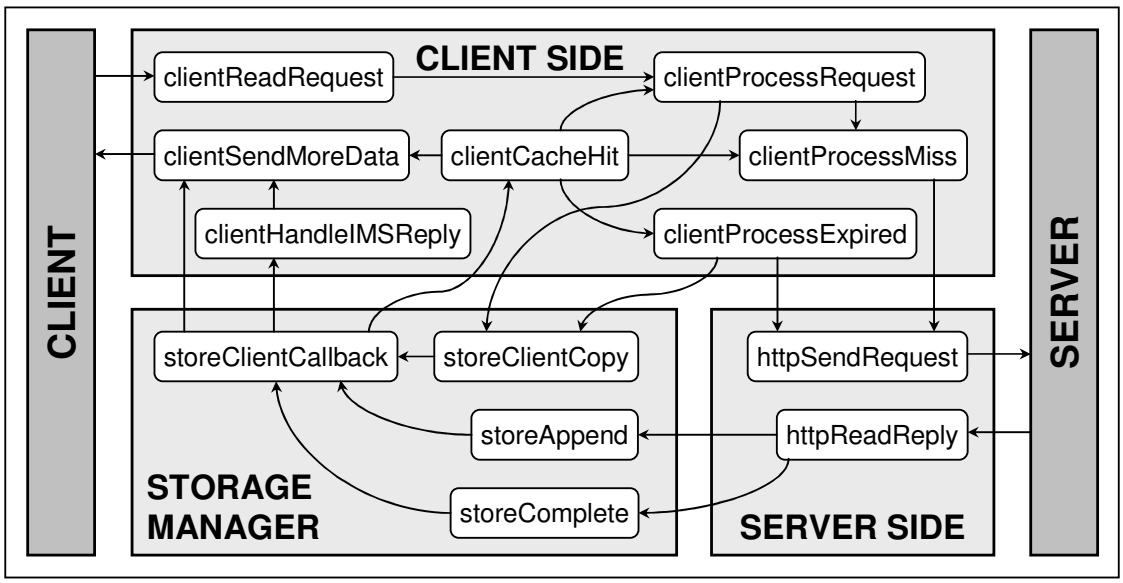


Figure 7.1 Three basic components of Squid

The "brain" of our model prototype, the proxy application had to go through modifications to support our proposed framework. Some notable modifications are:

1. The proxy's workflow was modified to accommodate retrieval of the meta-data documents, execution of the decision-making process, and adaptation to the object stored in the proxy's cache.
2. A number of external modules were attached to the proxy. They were CC/PP parser, ADP parser, and interfaces for accessing the adaptors.
3. A rule-based decision maker was embedded to the proxy to select the best-fit object's representation for a particular client.
4. Simple adapting commands called *apcoms*[30] – a kind of API – were devised to accommodate the adaptation into the proxy's cache.

The next subsections discuss the four modifications in detail.

7.4.1 Modified Workflow

As stated before, the Squid's workflow was extended with three additional processes: meta-data retrieval, decision-making process, and adaptation process. Those processes are important to support our proposed framework. They are discussed and detailed in this subsection.

Meta-data retrieval

The meta-data retrieval should be done before the requested object is fetched either from the local cache or from an external server. Referring to the workflow in Figure 7.1, the meta-data retrieval should be placed after a request is received (routine

[30] Stands for "Adapting Proxy Commands"

clientReadRequest) but before the request is processed (routine *clientProcessRequest*). In fact, a few checking processes take place between the two routines; those are access check (whether the request is allowed), redirection check (whether the request needs to be redirected), and no-cache check (whether the request must not be satisfied from the proxy's cache). We decided to place the meta-data retrieval just after the access check. This is to make sure that the request is already allowed to continue. Figure 7.2 shows the modified workflow after the meta-data retrieval is put in. The dotted-line arrow in the figure is the original workflow, the diagonal-striped box (*clientAccessCheckDone*) the affected routine, and the shaded, dotted-line box (*clientRedirectParser*) the new routine for retrieving the meta-data.

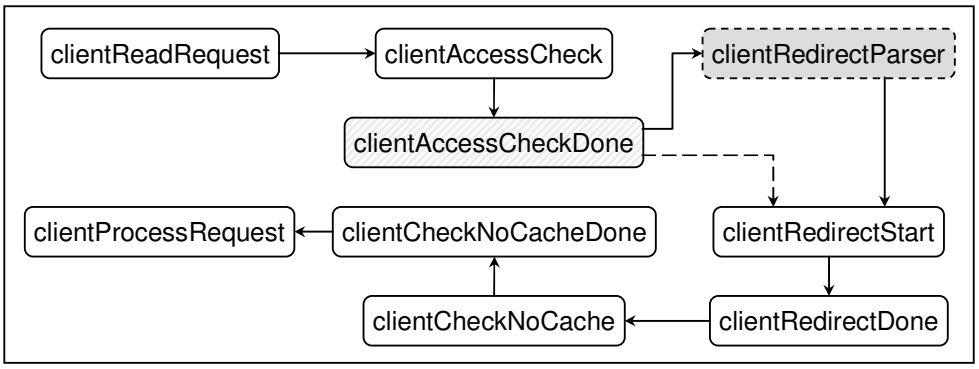


Figure 7.2 Modified Squid's workflow to include the meta-data retrieval

Initially, routine *clientRedirectParser* contains three execution phases. Phase 0 is the initial phase; that is when the routine is called for the first time. Phase 1 is when the client's CC/PP document is retrieved. URI of the CC/PP document can be found in the request's `CCPP` header and is conveyed to the CC/PP parser, an external module analyzing the CC/PP document. Phase 2 is when the requested object's ADP document is retrieved. URI of the ADP document is determined by adding '.adp' to the object's URI

and is conveyed to the ADP parser, another external module analyzing the ADP document. The CC/PP and ADP parsers will be discussed shortly in Subsection 7.4.2. After the meta-data retrieval, the process returns to the normal workflow, i.e., executing routine *clientRedirectStart*.

Decision-making process

The decision-making process is carried out to select the best-fit object's representation for a particular client. It is done by matching the client's characteristics (in the CC/PP document) with the object's characteristics (in the ADP document). Thus, this process can only take place after both meta-data are retrieved. We opted to extend routine *clientRedirectParser* to include the decision-making process. We added the fourth phase (Phase 3) to the routine to call routine *clientSelectRepresentation*, where the decision-making process takes place. Once routine *clientSelectRepresentation* is completed, the representation for that client is already decided, and the control is returned to routine *clientRedirectParser*. Figure 7.3 depicts the new addition to the Squid's workflow.

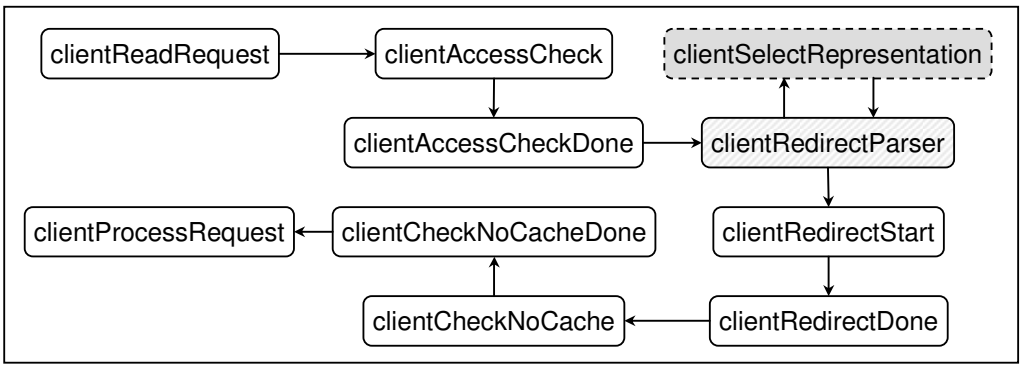


Figure 7.3 Modified Squid's workflow to include the decision-making process

In the decision-making process, to match the client's characteristics to the object's characteristics, a list of rules is needed. It is very likely that both meta-data have different

semantics. Often they do not share the same attributes' names, or worse, they may share some attributes' names but those names have different meanings. Hence, the rules help indicate which attributes of the client's characteristics are to be matched with which attributes of the object's characteristics. In addition, the list of rules offers us a sense of priority. More important attributes are to be matched first, whereas less important attributes are compared later. More discussion on this decision-making process can be found in Subsection 7.4.3.

Adaptation process

The adaptation process includes retrieval of the requested object (whether from the proxy's cache or from an external server), communication with a variety of adaptors, and construction of the object's representation that fits the client's characteristics. Different from the previous two additional processes, the adaptation process could not be added to the Squid's workflow in a single location since the adaptation might involve modifications to the requests as well as the responses. Instead, it was spread over various places in the client side and storage manager components.

First of all, once the decision-making process selects the fitting object's representation, the adaptation process has to modify the request to get that particular representation. In other words, it has to determine the adapting parameters needed to obtain the representation. Due to this task's closeness to the decision-making process, we assigned the task to routine *clientRedirectParser*, right after completion of decision-making process (in Phase 3). An appointed adaptor is required to help generate the adapting parameters. The upper left part of Figure 7.4 illustrates this task.

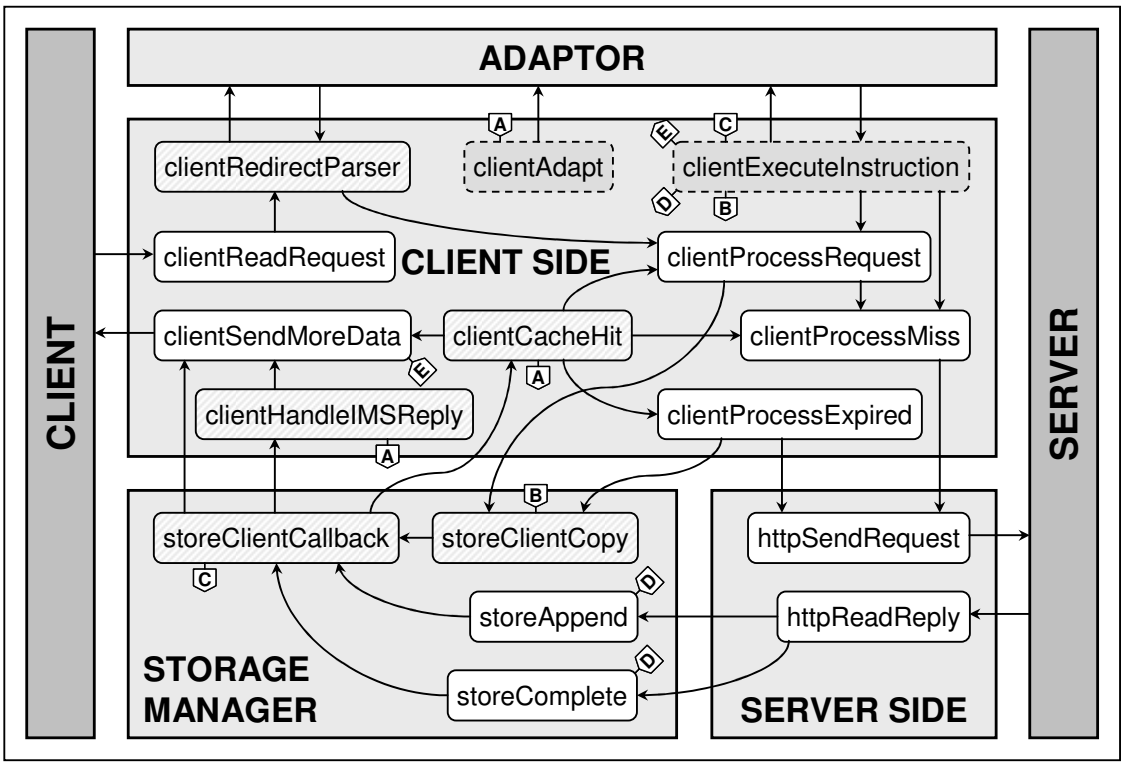


Figure 7.4 Modified Squid's workflow to include the adaptation process

Next, the adaptation process may have to transform the cached representation into the best-fit one, if the two are different. This transformation is particularly applied to modulation, where high data reuse is possible. To carry out the transformation, again the adaptation process needs assistance from the appointed adaptor. This task is performed by a new routine, *clientAdapt*. The routine can only be called after the cached representation has been determined, and there are two places where the routine may be needed. The first is in routine *clientCacheHit* and the second is in routine *clientHandleIMSReply*. The added and modified routines are depicted in Figure 7.4 with shaded and striped boxes, respectively. Connector A in the figure represents the routine calls from routines *clientCacheHit* and *clientHandleIMSReply* to routine *clientAdapt*. Up till this point, the adaptor has given instructions how the adaptation should be done, but the process itself

has not begun yet. The adapting instructions are given in the form of a sequence of apcoms, discussed later in Subsection 7.4.4.

Lastly, the adapting instructions given by the adaptor need to be executed in a controlled manner. Another new routine, *clientExecuteInstruction* was created to handle the execution. Routine *clientExecuteInstruction* gets the control for the first time after the adaptor conveys the adapting instructions. It executes one instruction at a time – perhaps by calling another routine. Every time the execution of one instruction is completed, it gets the control back to execute the next instruction, and this goes on until all instructions are executed. Depending on the cached and targeted representations, there are many ways that the execution may take up. Therefore, routine *clientExecuteInstruction* may have wide access to many routines. It may fetch the cached representation (Connector B and C), output the cached representation to and ask for service from the adaptor, or replace the cached representation with another one (Connector D). It may also send a request to an external server (i.e., another proxy cache or the original server), either by calling routine *clientProcessRequest* or routine *clientProcessMiss*. And of course, it can send a response to the corresponding client (Connector E); usually, this is done after all adapting instructions are completely executed.

The proxy application may have a variety of adaptors attached to it. Our modified Squid communicates with standalone adaptors – like JPEG transcoders and JPEG 2000 modulators – through some interfaces. Actually, the adaptor depicted in Figure 7.4 refers to the standalone adaptors and the corresponding interfaces as one package. There are three different times during the adaptation process that the proxy application may need the adaptor's assistance. The proxy application has to send different directives to indicate

the kind of assistance it needs. The first is when it needs the adapting parameters of a particular object's representation; routine *clientRedirectParser* will send a `QRY` directive to the adaptor. The second is when it needs specific instructions to transform one representation to another; routine *clientAdapt* will send an `ADP` directive to the adaptor. The last is when it asks the external adaptor to execute the adaptation; routine *clientExecuteInstruction* will send an `EXE` directive to the adaptor. The interfaces and their supporting roles are further discussed in the next subsection.

7.4.2 External Modules

To understand the client's characteristics (i.e., its capabilities and preferences) as well as the requested object's characteristics, the proxy cache has to be equipped with respective parsers: CC/PP and ADP parsers. We developed the parsers as external modules to the proxy application, so that it is possible to run several parsing processes concurrently. Both meta-data documents (CC/PP and ADP) are based on RDF/XML. In order to speed up and simplify the parsers' development, we had better utilize available RDF libraries. Therefore, libraries from Redland RDF Application Framework[31] version 1.0.0 were employed to develop the parsers.

The CC/PP and ADP schemas were hard-coded into the respective parsers. So, the parsers need not read the schemas any more. The input of each parser is just the URI of a meta-data document. The parser will fetch the meta-data document with help from

---

[31] http://librdf.org

Redland and cURL[32] libraries. In the header of the meta-data document, some extension schemas may be declared. The parser will then fetch the extension schemas and extend its default schema with those schemas. Therefore, during parsing the meta-data document, it can understand the properties already specified in its extended schema. Any unknown property may generate an error. The output of the parser is the attribute's names and values in a certain format. The CC/PP and ADP parsers generate output in slightly different formats, which can be understood by the respective internal modules in the proxy cache [Note that, in Squid, every external module must be coupled with an internal module. Those two modules communicate each other through two channels; one – connecting the internal module's `stdout` with the external module's `stdin` – is used by the internal module to send instructions and the other – connecting the internal module's `stdin` with the external module's `stdout` – is used by the external module to send results.]

The temporal and spatial locality in Web traces tells us that the same client may request again and the same object may be requested again soon. Considering this premise, it would be better if the client's CC/PP document and the Web object's ADP document are cached in the proxy. It can save time and bandwidth. Coincidentally, the CC/PP and ADP parsers use cURL to fetch the meta-data documents, and cURL may use the *http_proxy* environment variable, if defined, to direct its requests. To store the meta-data documents in the local proxy's cache, the *http_proxy* environment variable is set to "localhost[:*port*]", where *port* – in a default Squid installation – is 3128.

---

[32] http://curl.haxx.se

Other external modules required by the modified proxy application are interfaces to link the proxy application with the adaptors. These interfaces are comparable to those implemented in the server application. However, here each interface has three tasks to deal with, and the tasks are initiated by Squid. The tasks, corresponding to the Squid's adaptation process, are as follows:

1. Generating the adapting parameters used to get a particular representation of a Web object. This task is run when the interface receives a `QRY` (referring to the URI's query string) directive. The combination of the adapting parameters is unique and can be used as the representation's identity when it is stored in the cache.
2. Determining the kind of adaptation needed to get a representation from another (i.e., cached) representation. This task is run when the interface receives an `ADP` directive. Completion of this task produces a sequence of apcoms employed to obtain the targeted representation.
3. Executing an appointed adaptor with particular parameters. This task is run when the interface receives an `EXE` directive. This is the primary task of the interface, which conveys the proxy's executing instructions to the associated adaptors.

Like the parsers, these interfaces are also coupled and communicate with internal modules, but here the internal modules are uniform and standardized to provide extensibility and modularity.

All external modules can be controlled in the Squid configuration. Each of them can easily be included in or excluded from Squid. Of each module, the number of children (i.e., concurrently running processes) and other settings, if any, can be specified in the configuration, too. In addition, use of external modules makes the proxy cache

extensible. New adaptors can be easily attached to the proxy cache by creating the corresponding interfaces and registering them in the Squid configuration.

7.4.3 Rule-Based Decision Maker

As indicated before, our modified Squid includes a decision-making process to assist in selecting the best-fit representation of a Web object for a particular client. Two kinds of meta-data are involved in the process; those are the client's and the object's characteristics. To select the representation, the client's characteristics are matched against the characteristics of each object's representation. The matching process is directed by a list of rules. As explained earlier, these rules serve two purposes: 1) pointing out the attributes to be matched against, and 2) bringing in prioritization to the matching process. The following paragraphs elaborate how this rule-based decision maker works in our modified Squid.

After analyzing the client's CC/PP document, the external CC/PP parser returns the client's characteristics to Squid, which in turn stores them in CC/PP components (i.e., *CCPP_comp* structures). Based on the examples presented in the CC/PP structure and vocabularies[33], there are three predefined types of CC/PP components: hardware, software, and browser. Any CC/PP component that cannot be classified into one of the three predefined types will be marked unknown. Similarly, the external ADP parser analyzing the object's ADP document returns the object's characteristics, which are then stored in ADP objects (i.e., *ADP_obj* structures). Five types of ADP objects are

[33] http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/

predefined; those are general, image, video, audio, and text. Besides that, the adaptors – which are also stored in ADP objects – have a type of their own. Both, CC/PP components and ADP objects, are kept in the client request's data object (i.e., *clientHttpRequest* structure). But without any help, those meta-data are just meaningless. Hence, external assistance is required to explain the meta-data.

Attributes of the client's and the object's characteristics that will be used in the matching process need to be declared. A declared attribute of the client's characteristics is represented by a CC/PP variable, whereas that of the object's characteristics represented by an ADP variable. Thus, there are two things done in a single action. When a variable is linked to a particular attribute, the variable being defined can be used by and the attribute being declared recognized by the Squid's decision maker. Definition of the CC/PP and ADP variables are done in the Squid's configuration. Below is the definition's example:

```
ccpp_def dev_width H –i pix-x width
ccpp_def dev_height H –i pix-y height
adp_def rep_width GEN –i width
adp_def rep_height GEN –i height
```

The first line defines a CC/PP variable called *dev_width*. The variable is linked to any CC/PP hardware (H) attribute, which contains strings "pix-x" or "width" in its name. The "-i" option means that the name matching is case insensitive. The name matching is preferred here since different content providers may employ different semantics for the meta-data. Similarly, the second line defines another CC/PP variable, *dev_height*, which is linked to any CC/PP hardware attribute containing case-insensitively strings "pix-y" or "height" in its name. The last two lines define ADP variables, i.e., *rep_width* and *rep_height*. They are linked to ADP general attributes containing case-insensitively

strings "width" and "height", respectively. Hence, attributes `presentationWidth` and `presentationHeight`, which are specified in the ADP schema (see Appendix A), will be linked to respective variables *rep_width* and *rep_height*.

Once they are defined, the CC/PP and ADP variables can be matched against each other under direction of some matching rules. The matching rules have to be specified in the Squid's configuration, as well. An example of the matching rules is presented below:

```
obj_match 0 [rep_width + 10 <= dev_width]
obj_match 1 [rep_height <= dev_height]
```

Each matching rule has an identity number, which also determines the rule's priority; the smaller the rule's identity number, the higher is its priority. In the example, the first rule (Rule 0) has a higher priority than the second (Rule 1). All rules must be Boolean expressions consisting of operands (i.e., CC/PP and ADP variables) and operators. The list of supported operators for the matching rules is shown in Table 7.1.

Table 7.1 Matching rule's operators in precedence order

Operators	Operand(s)	Description
()	(B/I/R/S)	Parentheses
!	(B)	Unary logical negation
* / %	(I/R, I/R)	Multiplication, division, and modulus Exception: operands for modulus are (I, I)
+ −	(I/R, I/R)	Addition and subtraction
$	(S, I)	Substring
< <= > >= = !=	(I/R/S, I/R/S)	Relational operators
&	(B, B)	Logical AND
|	(B, B)	Logical OR

Note: B = Boolean, I = Integer, R = Real, S = String

For the decision maker, the rule-matching process is the means of selecting the best-fit representation for a particular client. Simplistically, the overall decision-making process looks as follows:

```
/*********************************************************/
/* Decision Maker: Selecting the best-fit representation  */
/*********************************************************/
// Variables:
// Rules => a list of matching rules
// Rule  => a matching rule
// Reprs => a list of ordered representations (ADP_obj struct)
// Repr  => a single representation (ADP_obj struct)
// Comps => a list of CC/PP components (CCPP_comp struct)
// Prsvd => the list of preserved representations
// Dropd => the list of dropped representations

WHILE (Rules != NULL) {
    // Get the top matching rule in list Rules
    Rule = Dequeue(Rules);

    // Initially, Prsvd and Dropd are empty
    Prsvd = NULL;
    Dropd = NULL;

    WHILE (Reprs != NULL) {
        // Get the top representation in list Reprs
        Repr = Dequeue(Reprs);

        // The rule-matching process =>
        // a Boolean value results from the execution
        IF (Match_Repr_and_Comps(Repr, Comps, Rule)) {
            // Add Repr to the end of list Prsvd
            Enqueue(Prsvd, Repr);
        } ELSE {
            // Add Repr to the end of list Dropd
            Enqueue(Dropd, Repr);
        }
    }

    // Determine Reprs for the next rule-matching process
    IF (Prsvd != NULL) {
        Reprs = Prsvd;  /* Preserved representations */
    } ELSE {
        Reprs = Dropd;  /* Dropped representations */
    }
}

// Returning the top-most representation
RETURN Pop(Reprs);
```

The pseudo-code above shows that each matching rule is applied to filtering out some representations. The matching rule employs specific representation's and client's attribute

values in its matching process. Once it is dropped, a representation may not be processed further. For each matching rule being executed, the preserved representations are collected and used in the execution of the following matching rule. However, if there is no preserved representation, then the dropped representations are used in the next execution; the associated matching rule is just ignored as if it does not exist. After the matching process is completed, one of the remaining representations (i.e., the top-most) is selected. Since the representations in the list have been arranged in a particular order, which can be determined by the content provider, the selected representation implies the best-fit one in the client's and the content provider's points of view.

The priority given to a matching rule determines its execution order and, consequently, the end result. Different arrangements for the rules may yield different results. For a fast example, if the two rules in the earlier example are combined into a single rule, the results may be different. The two-rule and one-rule cases look as follows:

```
/* Case 1 */
obj_match 0 [rep_width + 10 <= dev_width]
obj_match 1 [rep_height <= dev_height]

/* Case 2 */
obj_match 0 [rep_width + 10 <= dev_width &
    rep_height <= dev_height]
```

Assume that a Web object initially has 5 representations. Also assume that in the first case, Rule-0 filters out 3 representations and Rule-1 filters out the remaining 2 representations. Accordingly, Rule-0 in the second case will drop all representations. In the first case, Rule-0 holds but Rule-1 is ignored; hence, one of the 2 remaining representations will be selected. In the second case, Rule-0 is promptly ignored, and therefore, one of the 5 representations will be selected. Depending on the representations

being filtered out, the selected representations in the first and second cases may be different.

There are some hidden, but somehow beneficial, features that this decision maker may offer. It has been suggested before that the content provider can determine the representations' arrangement in the list. Actually, the representations are arranged in an ascending order based on their identities (stated in attribute `rdf:about` or `rdf:ID`). So, indirectly the content provider may indicate the arrangement through the representations' identities. Another useful feature is the rule-based nature in the decision-making process. It offers flexibility to the network providers (i.e., the proxy's owner) to prioritize the matching rules. Different network providers may have different ways to specify the rules, resulting in different representations being selected. Yet, semantics' loss in the end-to-end Web content delivery does not occur since the selection is made on the predefined list of representations given by the content provider through the ADP document.

7.4.4 Adapting Proxy Commands

To perform transcoding, Squid requires some modifications to the client side component (please refer to Figure 7.1), particularly to the routines handling responses. However, Squid performing modulation requires further modifications not only to the routines handling requests but also to those managing cached objects (in the storage manager component). This is because modulation may need to change the requests (e.g., to fetch a supplement instead of a representation) and the cached objects (e.g., to replace a representation with an improved one).

To offer flexibility to Squid in adapting Web objects, a set of adapting instructions (commands) was devised to accommodate various adaptors with different behavior. As indicated earlier, this set of adapting commands is called *apcoms*. An adapting interface, linking Squid to the associated adaptors, should give a sequence of apcoms when it receives an `ADP` directive (the interface's second task). The apcoms are then executed in Squid by routine *clientExecuteInstruction* (please refer to Figure 7.4). The list of apcoms that are currently supported by our modified Squid is presented in Table 7.2.

Table 7.2 Adapting proxy commands (apcoms)

Apcom	Longform	Param(s)	Description
FTO	FTOUCH	(flh)	Touch (create) a temporary file
FST	FSTORE	(flh)	Store *entry*'s object to a temporary file
FLO	FLOAD	(flh)	Load *entry*'s object from a temporary file
SES	SESHFT	N/A	Shift (move) *entry*'s object to *old-entry*
SEC	SECLR	N/A	Clear *entry*
SEN	SENEW	N/A	Create new *entry*
SEU	SEUPDT	N/A	Update *entry*'s info with its *reply*'s info
SER	SEREL	N/A	Mark *entry*'s object released (deleted)
OER	OEREL	N/A	Mark *old-entry*'s object released (deleted)
RPC	RPCOPY	N/A	Copy (duplicate) *old-entry*'s *reply* to *entry*'s *reply*
RPD	RPDATE	N/A	Store *reply*'s `Date` header to variable *tmp_date*
RPS	RPSWOT	N/A	Swap *reply* out (to *entry*'s object)
CHA	CHADPQ	(str)	Change *reply*'s `ADP-Qry` header
CHC	CHCLEN	(flh)	Change *reply*'s `Content-Length` header
CHD	CHDATE	N/A	Change *reply*'s `Date` header with variable *tmp_date*
RQC	RQCHG	(str)	Change *request*'s query string and send it out
RQD	RQDEL	N/A	Delete *request*'s query string
LOG	LOGPAR	(flh'/'flh)	Calculate partial hit info for logging
EXE	EXEC	(str)	Execute adaptation in an appointed adaptor

Note: flh = file handle, str = string, N/A = not applicable

The apcoms listed in Table 7.2 are the commands required to accommodate JPEG transcoders and JPEG 2000 modulators that have been earlier developed. It is not our objective to develop a complete and powerful proxy's API, but it may be considered for

future work. To understand clearly what each apcom does, it would be better to get familiar with the terms used in Squid. An object in the Squid's cache can only be created, added, and modified if it is linked to a *StoreEntry* structure. A *request*'s data object (of *clientHttpRequest* structure) has two pointer variables (of *StoreEntry* structure) to handle cached objects; one is called *entry* (the main pointer variable) and the other called *old-entry*. The cached object itself contains a variable, called *reply*, to hold the response. Now let us examine closely the adapting commands.

The first three apcoms handle an external temporary file. The channels connecting Squid and an interface are used merely for communication; they cannot be used to transfer data. Hence, a temporary file is the alternative to transfer data between Squid and an interface (and, inevitably, the adaptors). The parameter of the three apcoms is a file handle in the form of "`$`*n*", where *n* is a number starting from 1. The actual temporary file will be created and maintained by Squid.

The next six apcoms (`SES–OER`) handle the *request*'s *entry* and *old-entry* variables. Of interest to note is that apcom `SEC` clears *entry* without releasing (deleting) the cached object; the apcom just unlinks *entry* from the cached object. Instead, apcom `SER` should be used to release *entry*'s object; the cached object is not immediately released but later when the *request* is released. These six apcoms do not need any parameter.

The next six apcoms (`RPC–CHD`) handle *reply* of the *entry*'s cached object. The last three, in particular, modify the *reply*'s header fields: `CHA` modifies the `ADP-Qry` header (i.e., the adapting parameters), `CHC` modifies the `Content-Length` header (with

data-size of the given temporary file), and `CHD` modifies the `Date` header. Apcom `RPD` is the opposite of apcom `CHD`; while `RPD` stores the *reply*'s `Date` header to variable *tmp_date*, `CHD` updates the header with that variable. Apcom `RPS` is employed to write *reply* to the *entry*'s object; the apcom updates the cached object and usually is called if *reply* has been modified.

The next two apcoms (`RQC` and `RQD`) deal with the *request*'s query string. Note that `RQC` not only changes the query string but also sends the *request* to the server by calling routine *clientProcessRequest*. On the other hand, `RQD` does the opposite operation to `RQC`. It deletes the *request*'s query string, if one exists, and does not pass on the *request* to the server.

Apcom `LOG` is used to determine the percentage of partial hit. Two file handles, separated by '/', are needed. The partial hit's percentage is the ratio of the first temporary file's data-size to the second temporary file's data-size. Finally, apcom `EXE` is used to execute a particular adaptor. The accompanied string parameter is the complete executing command to run the adaptor, including the adapting parameters, input file, and output file.

7.5 Implemented Architecture

CC/PP and ADP, the meta-data to support Web content adaptation has been specified. The server application (Apache) has been extended to integrate the adapting modules. Enhancement is also done to the proxy application (Squid) to integrate the adapting modules, accommodate a decision-making process, and improve the cache's efficacy. Thus, all necessary elements to complete the model prototype have been detailed

and developed. The overall system architecture of the model prototype can be depicted as shown in Figure 7.5.

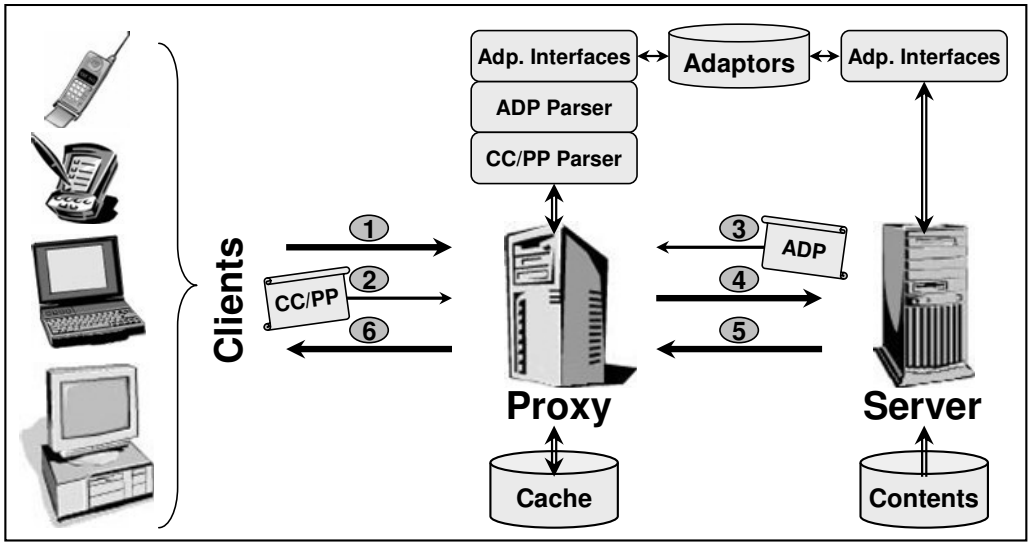


Figure 7.5 System architecture of model prototype

As seen in the figure, the proxy undergoes the most number of changes in the model prototype. The enhanced proxy includes CC/PP parser, ADP parser, and some adapting interfaces. The CC/PP parser is to fetch and analyze clients' CC/PP documents, whereas the ADP parser is to fetch and analyze objects' ADP documents. The adapting interfaces help in bridging the proxy and adaptors. The enhanced server also includes adapting interfaces. But the server's adapting interfaces are much simpler than those installed in the proxy. Both, proxy's and server's adapting interfaces often access the same collection of adaptors (i.e., transcoders and modulators), mostly standalone and provided by many parties. The client just goes through a very minor change, which is adding the `CCPP` header – containing the URI of its CC/PP document – in its requests.

The step by step data transfer among the three parties – client, proxy, and server – in the Web content delivery is as follows:

1. The client sends a request for a particular object to the server via the proxy. The request's header contains the URI of the client's CC/PP document.
2. Receiving the client's request, the proxy parses the request's header, gets the URI of the client's CC/PP document, and passes on the URI to the CC/PP parser. The CC/PP parser then fetches the CC/PP document (usually from a public Website referenced by the URI), parses it, and returns the client's characteristics to the proxy.
3. Based on the request's URI, the proxy determines the URI of the requested object's ADP document and passes on the URI to the ADP parser. The ADP parser fetches the ADP document (from the server), parses it, and returns the object's characteristics to the proxy.
4. Matching the client's and the object's characteristics, the proxy selects the best-fit representation of the object, revises the request accordingly, and sends the modified request to the server.
5. Receiving the proxy's request, the server fetches the requested object, adapts it accordingly, and returns the resulting representation to the proxy.
6. Receiving the server's response, the proxy stores the object's representation in its cache and passes on the response to the client.

All of the above steps have to be executed if the client makes a request for the first time or if the object is requested for the first time. For the following requests, steps 2 and 3 may not be needed. Steps 4 and 5 may be skipped too, if the proxy can satisfy the client's request with its cached objects. Compared to the original system architecture, the new one suffers additional roundtrips owing to the processes in steps 2 and 3. However, as mentioned earlier, the additional roundtrips only occur in the first time requests.

Moreover, the new system architecture may reduce data transfer in step 5, particularly if modulation is employed for adapting the requested object.

7.6 Related Work

Although ICAP [ElC03] or OPES [BaPC04] framework may be employed for developing the model prototype – as indicated in the previous chapter – we opted to apply modifications directly to the server and proxy applications. A few reasons may be suggested. Firstly, ICAP and OPES are ongoing projects. Many things are still to be completed, and wider adoption and implementation remain to be seen. Secondly, both ICAP and OPES involve quite complex procedures. A lot of work is required to develop the model prototype using either of them. Anyway, the existing server and proxy applications do not support ICAP or OPES, so modifications to those applications are still needed. Lastly, separating the adaptation server(s) from the proxy – as what ICAP and OPES suggest – may incur another latency delay due to distant calls and additional network hops. Moreover, we believe that modulation comprises lightweight adapting operations that will not burden the proxy (or, even the server) so much.

Up till now, the only transcoding system that considers the client's as well as the server's directives is the Server-Directed Transcoding (SDT) [KnLM03]. SDT's client and server employ Media Feature Sets in the header of HTTP messages to convey their directives, whereas those in our model prototype use RDF/XML-based documents separated from the HTTP messages. While the former is standardized, the latter offers more extensibility and user-friendliness. In addition, separated from the HTTP messages,

the meta-data documents can be cached and reused for future references. The SDT's server uses applets to send transcoding instructions. The applets are comparable to the interfaces in our model prototype since the real adaptors are still needed in the transcoding proxy; SDT employs ImageMagick software package as the image adaptors. But SDT requires the applets or their parameters – although the applets are cacheable – to be fetched from the server for every different request, either because each applet may perform a specific transcoding operation or because a different set of parameters is required to fulfill a different request. That is not the case in our model prototype; all adapting instructions and information are already in the plugged-in interfaces or the cacheable meta-data documents. Last but not least, SDT stores multiple transcoded results in its cache. Employing modulation, our modified proxy may store just one representation for every object.

We are not the first to exploit the JPEG 2000 image standard in Web content delivery. Deshpande and Zeng [DeZ01] proposed an architecture for streaming JPEG 2000 images using HTTP. In their proposal, each JPEG 2000 image is accompanied by an index file, containing information about the image's structure and the URL of the image itself. The client firstly accesses the index file. Based on the MIME type of the index file, a helper application is invoked. The helper application uses the information in the index file to send one or more requests for parts of the image, and later, decodes the image's parts and displays the resulting image. The HTTP `Range` header is used to request the image's parts. A user's action – such as zooming and panning – will generate more requests to obtain the relevant image's parts. The index file here is comparable to the object's meta-data in our framework. Deshpande and Zeng's proposal expects the client

to be the active party making some specific requests based on the given information. Also, their architecture does not involve a caching proxy (anyway, partial contents commonly are uncacheable), so the server's responses cannot be shared with other clients.

Chi and Cao [ChC02] also used the HTTP `Range` header to request part of a JPEG 2000 image, but no knowledge about the image's structure is required. The image data presented to the client is always contiguous and may be truncated at an arbitrary point. As a result, their system can only offer quality scalability. However, a caching proxy that can store partial contents is involved in their system; thus, the image data can be shared among many clients.

Li and Sun [LiS03] and Taubman and Prandolini [TaP03] proposed new protocols – namely Vmedia and JPIP, respectively – for browsing JPEG 2000 images. Both protocols support interactivity between a client application and a remote server. The client makes a request for a JPEG 2000 image based on the current spatial region and resolution of interest. During the image data transfer, the client may revise its interest by user's zooming and/or panning, which generates a new request to replace the previous one. The server may send the image data out-of-order, and it may keep track the sent data to avoid retransmission. On the other hand, the client caches the image data – either in memory or in disk – to prevent repeated requests. Understandably, the image data needs to be arranged in a certain data format; Vmedia uses MU/SMU structures while JPIP uses data-bins. Both protocols may be run on top of HTTP (TCP) or UDP. While acknowledging the excitement of interactive access to the images, we think that caching the image data in the proxy to be shared with others is more beneficial to the Web community. The interactive nature of the new protocols, on the contrary, may cause

partial responses that cannot be cached at all in the Web proxy. Although JPIP's deployment may involve a proxy server, it is a special-purpose proxy – not a Web proxy – that can only serve JPIP requests. The interactivity between the client and the server also generates more data overheads owing to the multiple requests and responses. In addition, implementation of Vmedia and JPIP requires special applications at the server as well as at the client, which may consume quite a lot of resources. By contrast, our framework can be implemented easily to the existing server's application and client's Web browser without taxing much on their performance. Our framework requires a number of modifications on the proxy's application, yet its functionality is not reduced but improved. And more importantly, our framework can offer high data reuse in delivering Web contents.

In this chapter, a working model prototype – based on our proposed framework – has been developed. Development of the simple model prototype involves meta-data specifications and modifications on the current server and proxy applications. To verify the framework's benefits, the next chapter describes some evaluations on the model prototype.

Chapter 8

Performance Evaluation on Proposed Pervasive Web Content Delivery

In this chapter, the model prototype – the development of which is detailed in the previous chapter – is evaluated. Since the framework for pervasive Web content delivery – on which the model prototype is based – expects collaboration between the origin server and the proxy, the evaluation was conducted in two stages. In the first stage we evaluated the adaptation at the server only, whereas in the next stage we evaluated the adaptation at both the server and the proxy. Through the evaluation, the benefits as well as costs of our proposed framework will be revealed, analyzed, and discussed.

8.1 Experimental Setup

In this section, the adapting applications, image test data, and meta-data documents used in the experiments are detailed. The adapting applications and image test data, in particular, have been created and used previously in Chapter 5.

8.1.1 Experimented Adaptors

All experiments in this chapter employed the same sets of adaptors as those already used before. There are three sets of adaptors:

1. JPEG spatial-domain transcoders (SDT);
2. JPEG frequency-domain transcoders (FDT); and
3. JPEG 2000 modulators.

Readers are referred to Subsection 5.2.1 for the detailed particulars of the respective sets of adaptors. All sets of adaptors were installed at the server and the proxy. The server's specification is a 650 MHz Pentium III system, with 128 MB of RAM and Fedora Core Linux 2 as the OS. The machine employed as the proxy, on the other hand, is a 1.3 GHz Pentium 4 system, with 128 MB of RAM; Fedora Core 2 Linux is also used as the operating system.

8.1.2 Image Test Data

Two images – in the JPEG and JPEG 2000 standards – were used in the experiments. The experimented images had been used in the previous experiments, as

well. They are `boat.jpg` and `boat.jp2`, the creation of which can be found in Subsection 5.2.2. To refresh our memory, both images originated from the same source image (i.e., `boat.ppm`), shared the same resolution of 1976 × 2960 (width × height) pixels, and had similar data-sizes – almost 5 MB. The progression order of image `boat.jp2` used here has been changed to CPRL (the original employs LRCP), so that it can be directly modulated along component aspect without changing the progression order. Both images resided in the server.

For each experimented image, six representations with different characteristic settings were determined. Particulars of the six representations resulting from SDT, FDT, and modulation are listed in Table 8.1, Table 8.2, and Table 8.3, respectively. The resolution reduction was restricted until a one-eighth resolution of the original image since it is the maximum limit that the JPEG transcoders (SDT and FDT) can do.

As shown in the tables, the six predetermined representations vary in quality, resolution, color, and – more importantly – data-size; they are to serve different client preferences. Two of them have a half resolution of the original image, two have a quarter resolution of the original image, and two have a one-eighth resolution of the original image. Three of them have full color components, whereas the other three are grayscale (having the luminance component only). All of the JPEG 2000 representations (Table 8.3) contain five quality layers; it means the top five quality layers were dropped when the original image was modulated. The quality values of the JPEG representations (Table 8.1 and Table 8.2) resulting from SDT and FDT were determined such that their data-sizes are comparable to the data-sizes of the corresponding JPEG 2000 representations.

Table 8.1 Representations of `boat.jpg` adapted by JPEG transcoders (SDT)

Representation	Data-Size (bytes)	Resolution (width × height)	Color/Gray	Qty Value
1/2-scaled	157,673	988 × 1480	Color	47
1/2-scaled, gray	131,034	988 × 1480	Grayscale	41
1/4-scaled	114,579	494 × 740	Color	88
1/4-scaled, gray	86,810	494 × 740	Grayscale	84
1/8-scaled	57,329	247 × 370	Color	96
1/8-scaled, gray	37,473	247 × 370	Grayscale	93

Table 8.2 Representations of `boat.jpg` adapted by JPEG transcoders (FDT)

Representation	Data-Size (bytes)	Resolution (width × height)	Color/Gray	Qty Value
1/2-scaled	157,538	988 × 1480	Color	57
1/2-scaled, gray	130,310	988 × 1480	Grayscale	50
1/4-scaled	115,150	494 × 740	Color	90
1/4-scaled, gray	87,382	494 × 740	Grayscale	87
1/8-scaled	62,054	247 × 370	Color	98
1/8-scaled, gray	37,114	247 × 370	Grayscale	95

Table 8.3 Representations of `boat.jp2` adapted by JPEG 2000 modulators

Representation	Data-Size (bytes)	Resolution (width × height)	Color/Gray	No. of Qty Layers
1/2-scaled	158,733	988 × 1480	Color	5
1/2-scaled, gray	130,454	988 × 1480	Grayscale	5
1/4-scaled	115,217	494 × 740	Color	5
1/4-scaled, gray	86,958	494 × 740	Grayscale	5
1/8-scaled	58,439	247 × 370	Color	5
1/8-scaled, gray	36,860	247 × 370	Grayscale	5

As noticed, the quality values of SDT-based representations are lower than those of their FDT-based counterparts. However, PSNR measurement reveals that the qualities of SDT-based representations are 1–6 dB (in case of grayscale, 2–11.5 dB) higher than those of FDT-based representations. This finding is in agreement with the previous finding in Subsection 5.3.1. The previous finding gives lower quality difference than the present finding perhaps because the reduction of resolution and component, which is done here but not in the previous experiments, further increases the quality gap between the SDT and FDT results.

The qualities of JPEG 2000 representations, by contrast, are further lower by up to 6 dB (in case of grayscale, up to 7 dB) when compared to those of FDT-based, JPEG representations. This finding is in stark contrast to the JPEG 2000's superiority over JPEG in bit-rate performance – as noted in Subsection 5.3.1 – and may be explained by the multi-scale presentation within the JPEG 2000 image. Since multiple layers, resolutions, and components are bestowed on the image, removal of a single layer, resolution, or component – by means of modulation – always corresponds with a decline in the image's quality. Moreover, the quality decline is consistent in all adaptation aspects of the image; it means that combined removal of layer, resolution, and component further decreases the image's quality. This is different from transcoding, which does not suffer much from multi-aspect reduction.

Nevertheless, visual comparison reveals that the representations do not differ so much. Figure 8.1 shows the 1/8-scaled, gray representations (the last rows of the above tables) of SDT-ed `boat.jpg`, FDT-ed `boat.jpg`, and modulated `boat.jp2`. Please note that representation (a) is 11.5 dB better than representation (b), whereas

representation (b) is 7 dB better than representation (c). But visually representation (c) is better than representation (b) and is almost comparable to (perhaps slightly brighter than) representation (a), albeit there is 18.5 dB quality gap between them. [Note: observe particularly the boat's mast and rigging to contrast the images' difference in quality]

(a) **(b)** **(c)**

Figure 8.1 1/8-scaled, gray representations of `boat.jpg` and `boat.jp2` (partial images) (a) SDT-JPEG; (b) FDT-JPEG; and (c) Modulated JPEG 2000

8.1.3 Server Meta-Data Documents

Each experimented image is accompanied by an ADP document describing the image's characteristics and listing the suggested representations of the image. The ADP schema extension and ADP documents used in the experiments are attached at Appendix

B. To differentiate images for SDT and FDT processes, a *symbolic link*[34] named `boat.jpeg` was created at the server and pointed to image `boat.jpg`; hence, accessing any of them (`boat.jpeg` or `boat.jpg`) gives the same image data. Image `boat.jpeg` was used for SDT, and image `boat.jpg` for FDT. The associated ADP documents for the transcoding processes are `boat.jpeg.adp` and `boat.jpg.adp`, respectively. Each of the ADP documents has size of about 6.3 KB and contains 15 representations (including the original and the six predetermined representations) of image `boat.jpg`. The ADP document for modulating `boat.jp2` is `boat.jp2.adp`. It has size of about 8.8 KB and contains 21 representations (also including the original and the six predetermined representations) of image `boat.jp2`. Thus, generally the description of an image's representation takes around 420–430 bytes.

Each representation in the ADP documents has a unique identity (`rdf:ID`) so that it can be accurately addressed. Notice also that the naming of the representations' identities has been purposely arranged. As noted in the previous chapter, the proxy's decision-making process sorts the representations based on their identities. In case there is more than one representation matching a client's profile, the representation on top of the list will be selected as the best-fit representation for the client.

Among the image's attributes, the most difficult one to obtain is probably the data-size of a representation. For the `boat.jpg`'s representations, each of them had to be created (by SDT and FDT) to obtain the data-size. The data-sizes of the `boat.jp2`'s representations are easier to obtain. Since modulation basically just drops the image's

---

[34] In Linux OS, it is created by executing command "`/bin/ln -s`".

packets, we can devise an application to get the bytes of all packets of the image and calculate the exact data-sizes of all possible representations.

8.1.4 Client Meta-Data Documents

In line with the six predetermined representations described in Subsection 8.1.2, we also created six client profiles, each of which should get one of the six representations when used in a request for the experimented images. The client profiles were specified in the form of CC/PP documents, which are attached at Appendix C. The six client profiles are called `Client1` to `Client6`. If a client holding profile `Client1` requests the experimented images, the 1/2-scaled representations should be returned. A client holding profile `Client2` should get the 1/2-scaled, gray representations, and the arrangement continues until a client holding profile `Client6` gets the last representations, i.e., the 1/8-scaled, gray ones. These client profiles – CC/PP documents – were stored in a separate server and could be referred to by their URIs (i.e., the server's domain name plus the location path of respective documents). Remind that the CC/PP document can be attached to the client's request by putting the document's URI in the `CCPP` header.

The matching of a client's profile and a particular image's representation cannot be separated from the decision-making process at the proxy. Considering the client's profile and the image's characteristics, the proxy has to decide which image's representation is served to the client. Some matching rules have to be specified at the proxy so that the expected results can be precisely achieved. The employed matching rules will be revealed later when we discuss the experiments at the proxy. For now, it is

enough to say that the client profile's attributes, used in the decision-making process, were the display screen's dimensions and supported color, as well as the availability of the browser's scrollbars.

All applications, image test data, and meta-data documents have been described in this section. Now, we are ready to begin with the first evaluation concerning adaptation at the server.

8.2 Evaluating Adaptation at Web Server

We start with an assertion that adaptation at the proxy alone may not be beneficial and even causing an increase in the client's response time. This has been analytically studied in Section 6.2. Here, let us verify that assertion. Consider the following illustration. Suppose a client requests one of the experimented images above (`boat.jpg` or `boat.jp2`) via a proxy. Due to the client device's constraints, the requested image needs to be adapted. For instance, the best-fit representation for the client is the 1/2-scaled representation. If the adaptation is done at the proxy, the original 5 MB-sized image has to be fetched by the proxy from the server. Assuming that the server-proxy link has a bandwidth of 1 Mbps (considered quite a fast connection by today's standard), it takes about 40.8 seconds in our preliminary test to deliver the image data from the server to the proxy. We have not considered the adaptation delay yet, but the delivery time has already been beyond most clients' patience. And surely it is not acceptable for pervasive Web content delivery. In contrast, if the image is adapted at the server, the total response time

of adapting the image and delivering the adapted result is less than 3 seconds. This preliminary study corroborates that adaptation at the server is necessary, especially if the server-proxy link has a low bandwidth.

In the following subsections, the objectives of evaluating adaptation at the server are specified and the conducted experiments are detailed. Two sets of experiments have been conducted: 1) response time analyses, and 2) stress test. Explanation of the experiments is smoothly intermingled with the results and discussion.

8.2.1 Experimental Objectives

Some objectives of evaluating adaptation at the server have been specified. Firstly, the costs and benefits of the server-based adaptation can be identified. As noted earlier, one clear benefit has been the reduction in the client response time. Details of the benefit will be exposed by the experiments. At the same time, the experiments are expected to uncover the involved costs, as well.

Secondly, through the experiments, the two adaptation methods – transcoding and modulation – can be compared and contrasted. Chapter 5 has compared and contrasted the JPEG transcoders and JPEG 2000 modulators in a secluded environment. Here, we want to compare and contrast them in a near-real environment, where their differences may be further highlighted.

Lastly, the experiments may indicate some problems, inefficiencies, or other unexpected findings on the proposed server-based adaptation. They can be used as feedback to improve it in the future work.

8.2.2 Response Time Analyses

The response time of the Web server's service was measured for three different schemes. Each scheme corresponded with the employed sets of adaptors. The first scheme was transcoding (SDT) `boat.jpeg`, the second was transcoding (FDT) `boat.jpg`, and the last was modulating `boat.jp2`; respectively, they are referred to as JPG-SDT, JPG-FDT, and JP2-MOD schemes, for the rest of this section. All adaptation processes were executed at the server, and the proxy was not involved at all in the adaptation. The experiments simply measured the time required to get an adapted image directly from the server. The measured time started when a request was sent out and ended when the last byte of the response was received.

As the experiments did not involve adaptation at the proxy, which is the decision maker in our proposed framework, the meta-data documents (i.e., CC/PP and ADP documents) could not be used. Instead, each request included a specific query string to indicate the wanted image's representation. For an example, to get the 1/2-scaled representation of image `boat.jpeg` (stored at http://svr.my-dom.org/images/boat.jpeg), the corresponding request would be http://svr.my-dom.org/images/boat.jpeg?s=2&q=47. Readers may refer back to Section 7.3 for the format of the request's query string.

In addition to requests for the images' representations resulting from adaptation, for comparison purpose, direct requests for the representations without adaptation should be made possible. Hence, the images' representations were also generated and stored as resources at the server; these are referred to as the pre-generated representations. Later on,

the response times of requesting the representations *with adaptation* will be compared with those of requesting the representations *without adaptation*.

To simulate heterogeneous network environments, the experiments were executed on different bandwidth constraints, i.e., 56 Kbps (the common modem speed), 128 Kbps, 256 Kbps, 512 Kbps, and 1 Mbps. An Apache module, bw_mod[35] version 0.6, was installed at the server to limit each connection's bandwidth as wanted.

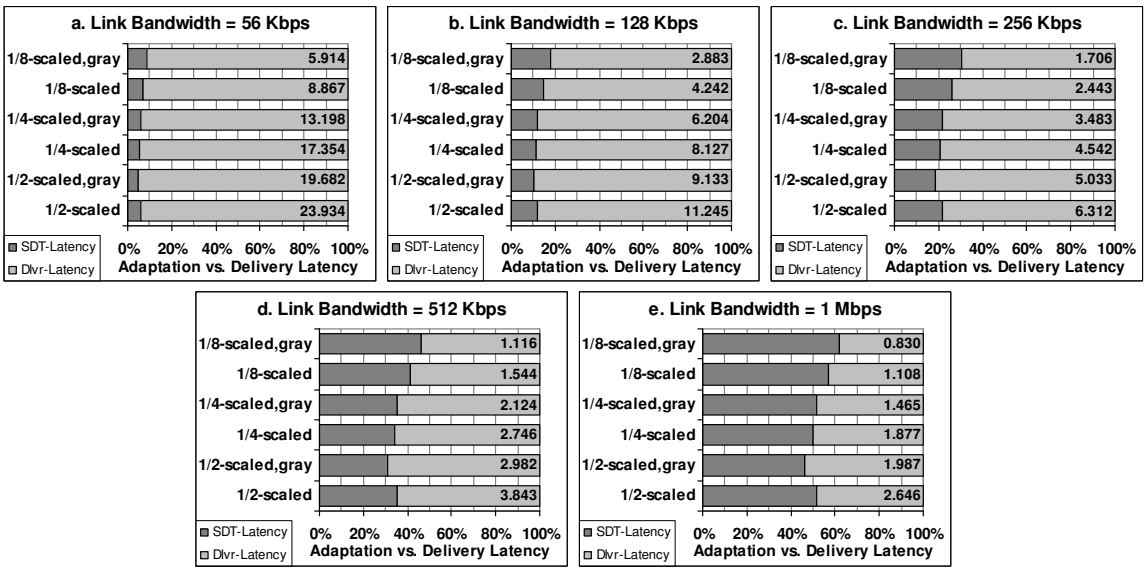


Figure 8.2 Response times (in seconds) of requesting `boat.jpeg`'s SDT representations from the server

The results of requesting `boat.jpeg`'s SDT representations (as listed in Table 8.1) in different network environments are presented in Figure 8.2. The corresponding response times, in seconds, are shown at the right-hand side of the bars. Each response time is composed of the transcoding (SDT) latency and the delivery latency. The delivery latency is actually the response time of requesting the pre-generated representation (without adaptation) from the server. The latency difference between the response time of

---

[35] http://ivn.cl/apache/

requesting a representation resulting from SDT and that of requesting the associated pre-generated representation is considered as the SDT latency. In the figure, the SDT latency and the delivery latency are depicted as percentages of the response time.

There are two trends that we may observe in Figure 8.2. The first trend is that the percentage of SDT latency increases as the bandwidth constraint is relaxed. On the 56 Kbps connection, the percentages of SDT latency are between 4.7% and 8.7% (on average, 6.2%), whereas those on the 1 Mbps connection are between 46.3% and 62.2% (on average, 53.1%). This trend is reasonable, since the SDT latency is relatively unchanged but the delivery latency decreases when the bandwidth constrain is relaxed. From this trend, again we may assert that the server-based adaptation is more beneficial when the bandwidth on the server-proxy connection is restricted.

Also observed is that the percentage of SDT latency tends to increase as the resulting representation's data-size decreases. This second trend may be attributed to the different data throughputs (in bytes/second) in generating the representations. If we refer back to the processing times of generating the representations in Section 5.4, many of them can be formulated in the form of $y = mx + c$, where m is the increase rate of processing time and c is a constant indicating the minimum processing time. If c is zero, then the data throughput will be steady (which is equal to $1/m$). But since c is a positive number, the bigger the resulting representation's data-size, the higher is the data throughput, and consequently, the lower is the percentage of SDT latency here.

However, while the 1/8-scaled, gray representation (the smallest in data-size) is taking the highest percentage of SDT latency as compared to other representations, the 1/2-scaled, gray representation takes the lowest percentage of SDT latency; if it follows

the pattern, the 1/2-scaled representation – which is the biggest in data-size – should take the lowest percentage of SDT latency. This phenomenon may be caused by the distinct performance of SDT processes in different aspects. Results in Section 5.4 have indicated that SDT processes in resolution and component aspects perform better than those in quality aspect. These collective factors make the SDT latency's percentage of 1/2-scaled, gray representation lower than that of 1/2-scaled representation.

Next, the response times of requesting `boat.jpg`'s FDT representations (as listed in Table 8.2) in different network environments are shown in Figure 8.3. Each response time is composed of the transcoding (FDT) latency and the delivery latency, both of which are determined in the same way as the previous results.

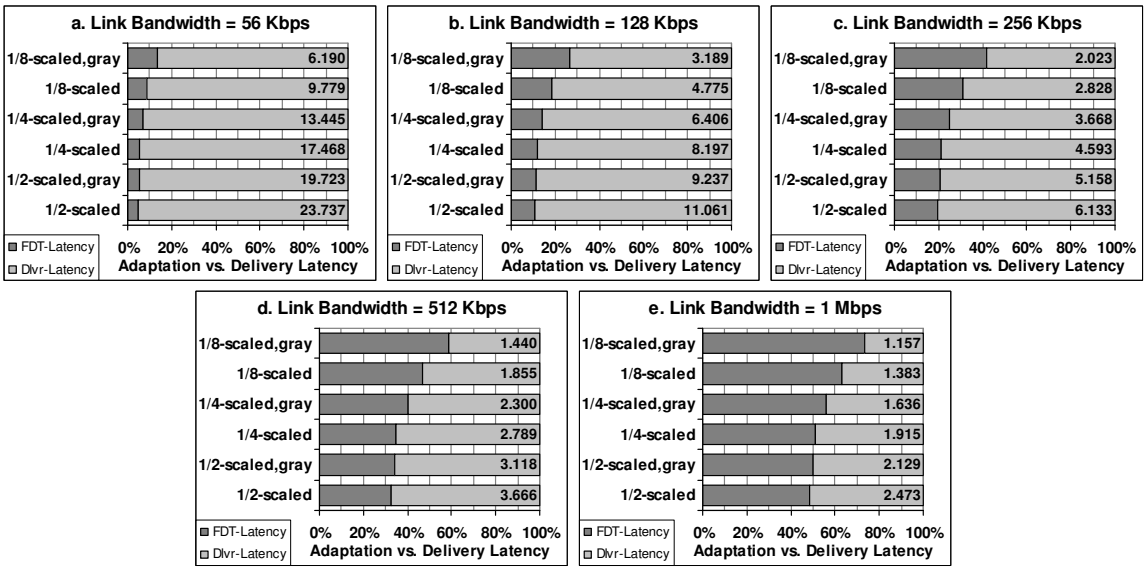


Figure 8.3 Response times (in seconds) of requesting `boat.jpg`'s FDT representations from the server

The two trends observed in the previous results are also found in Figure 8.3. The percentage of FDT latency increases from 5.0–13.7% (on average, 7.6%) on the 56 Kbps connection to 48.2–73.1% (on average, 56.9%) on the 1 Mbps connection. Overall, this JPG-FDT scheme takes higher percentages of adaptation latency than the JPG-SDT

counterpart; in fact, most of JPG-FDT's response times are also higher than JPG-SDT's response times, even though the data-sizes of representations in both schemes are comparable. As concluded in Section 5.4, FDT processes are superior to SDT processes for adaptation in quality aspect, but not so for adaptation in resolution and component aspects. This may explain the slightly inferior performance of the JPG-FDT scheme to the JPG-SDT scheme.

The percentage of FDT latency also increases when the resulting representation's data-size decreases, i.e., from the 1/2-scaled representation to the 1/8-scaled, gray representation. The pattern here is even sounder than that of the previous results, and this may be attributed to the more linearity of FDT processes in all adaptation aspects.

With respect to the last scheme, Figure 8.4 presents the response times of requesting `boat.jp2`'s representations (as listed in Table 8.3) in different network environments. Like the previous results, each response time is composed of the modulation latency and the delivery latency.

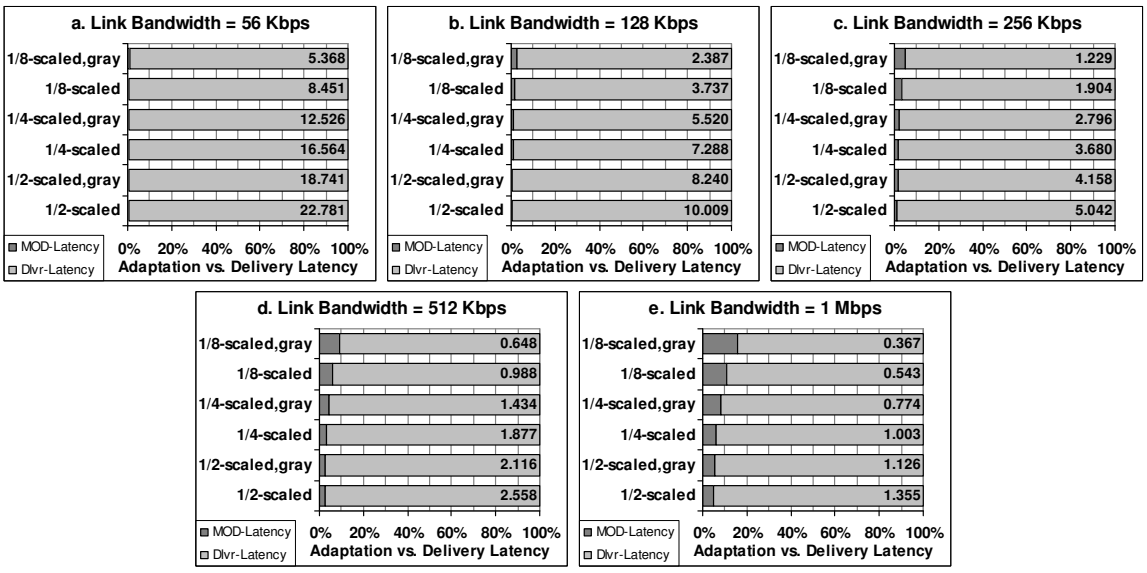


Figure 8.4 Response times (in seconds) of requesting `boat.jp2`'s representations from the server

The response times of the JP2-MOD scheme exhibit the two trends observed in the previous results. The percentage of modulation latency increases from 0.3–1.1% (on average, 0.5%) to 4.7–15.9% (on average, 8.6%) as the connection's bandwidth improves from 56 Kbps to 1 Mbps. The percentage of modulation latency also increases as the resulting representation's data-size decreases.

Compared to the previous results, the JP2-MOD scheme's response times give the lowest percentages of adaptation latency; that is, 4–20 times better than the percentages of adaptation latency in the JPG-SDT scheme and 5–18 times better than those in the JPG-FDT scheme. This is due to the fast and efficient processes performed by modulation. In addition, the response times resulting from modulation give the clearest trends since modulating processes are very linear and consistent in all adaptation aspects, as already demonstrated in Section 5.4.

8.2.3 Stress Test

The server was further examined through a stress test to figure out its performance while adaptation was being carried out. It is common knowledge that a running adaptation application requires system resources like CPU cycles and memory. If adaptation is executed at the server, each service to a client may consume more resources than usual (i.e., the server without adaptation), and therefore, the overall server's performance is expected to decrease. The purpose of the experiments is to reveal how much performance decrement that the server may suffer. The server's number of maximum clients was left unchanged, which was 150. [Note: we used Apache distributed by Fedora Core Linux.]

The stress test was done by executing a benchmarking application on a machine acting as a client. ApacheBench[36] version 0.63 was used here. It is a Perl API for Apache benchmarking and regression testing. ApacheBench was inspired by an application called `ab`, which is bundled in the Apache Web server package. Within the ApacheBench package, there is a ready Perl application called `ab`, too. For our stress test purpose, the `ab` application was slightly modified so that it could send requests for multiple Web resources concurrently. Besides executing the `ab` application, a monitoring application was devised and used to collect the periodic numbers of concurrent connections.

The three adaptation schemes used in the previous experiments (i.e., JPG-SDT, JPG-FDT, and JP2-MOD) were employed while the server was being stress-tested. In addition, requests for the pre-generated image's representations were also made to see the server's normal performance; this is called the no-adaptation scheme. [Please refer to the previous subsection for explanation about the pre-generated representations.] Thus, totally there were four schemes used for the stress test. For each scheme, there were six image's representations (i.e., from 1/2-scaled to 1/8-scaled, gray representations) requested from the server. A thousand requests were generated for each image's representation, so totally there were six thousand requests. The number of concurrent connections at any time was restricted to three hundred (300). Also, the server limited the bandwidth to 1 Mbps/connection.

Three schemes – i.e., no-adaptation, JPG-SDT and JP2-MOD – were successfully experimented, but the JPG-FDT scheme caused the server to crash due to its high demand

---

[36] http://adiraj.org/sw/ApacheBench/

for resources, particularly memory. If we examine closely the two types of JPEG transcoders, JPG-SDT comprises two applications (`cjpeg` and `djpeg`) whereas JPG-FDT comprises just one application (`jpegfdt`). Application `cjpeg` compresses a raw image into the JPEG format, and conversely, application `djpeg` decompresses a JPEG image into the raw format. In application `jpegfdt`, the decompressing and compressing data structures are allocated in the same time and additional buffer is also needed for the adaptation processes; that is why it demands a lot of memory space. We will return to experimenting with JPG-FDT later. Now let us analyze the results of the three successful schemes, as shown in Table 8.4.

Table 8.4 Results of stressing the server running adaptation [max. concurrent connections = 300]

Scheme	No. of Reqs*	Attained Req Rate (req/sec)	Attained Xfer Rate (KB/sec)	Conn. Time+ (milliseconds)	Resp. Time+ (seconds)
No-adapt	6000 (6000)	58.21	5,571.08	825.00 (3)	3.18 (2.47)
JP2-MOD	6000 (5988)	13.69	1,298.42	5,447.18 (1)	16.57 (12.29)
JPG-SDT	6000 (5621)	1.09	103.18	5,289.49 (2)	244.80 (240.57)

Note: * values are *sent* (*completed*); + values are *mean* (*median*)

The second column of Table 8.4 indicates the numbers of sent and completed requests; the number of completed requests is in the brackets. All requests were successfully served in the no-adaptation scheme. But in the JP2-MOD and JPG-SDT schemes, 0.2% and 6.3%, respectively, of the total requests could not be accomplished due to connection timeout. Without adaptation, the server could handle 58 requests per second, yield an average connection time of 825 milliseconds, and give an average response time of 3.18 seconds. With adaptation, the server's attained request rates were dropped to 14 and 1 requests per second, its average connection times delayed by 6.6 and

6.4 times, and its average response times slowed down by factors of 5.2 and 77.0 in the respective JP2-MOD and JPG-SDT schemes. Clearly, under a heavy load (about 300 concurrent connections), the server employing the no-adaptation scheme could serve all requests well. The server employing the JP2-MOD scheme could serve the majority of the requests but with increased response times. The server employing the JPG-SDT scheme, by contrast, suffered a higher percentage of failed requests and a much bigger drop in response times. The server employing the JPG-FDT scheme performed the worst since it crashed due to the scheme's high demand for memory space.

Further, for each successful scheme, the number of *established* concurrent connections to the server was periodically noted. The cumulative distributions of the collected data (i.e., periodic numbers of concurrent connections) are plotted in Figure 8.5. The figure reveals different characteristics of the three schemes: no-adaptation, JP2-MOD, and JPG-SDT.

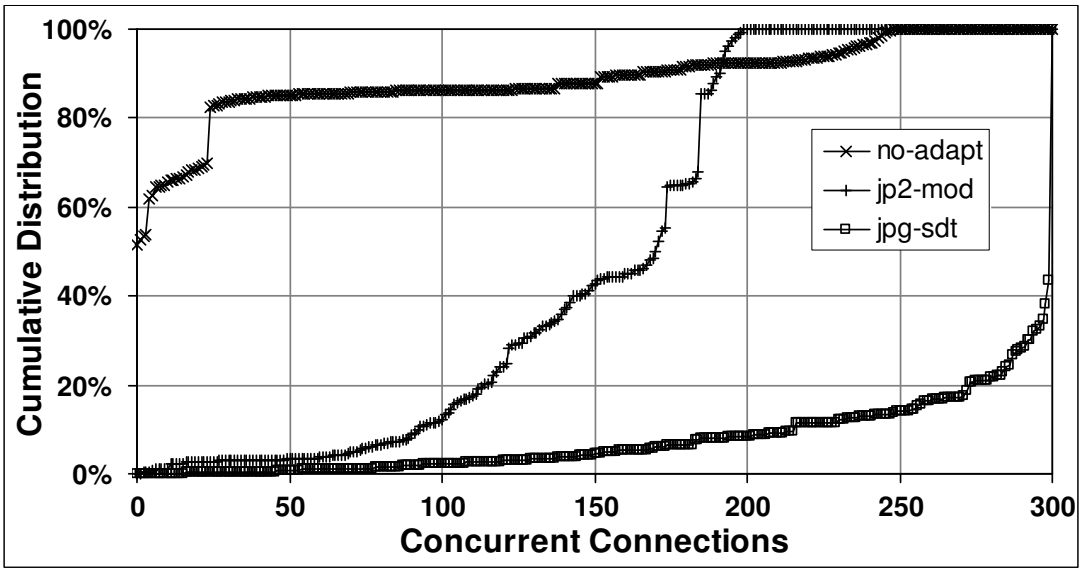


Figure 8.5 CDFs of periodic numbers of concurrent connections (max. 300) while stressing the server

Although the maximum possible number of concurrent connections was set to 300, the no-adaptation scheme (curve *no-adapt*) used at most about 250 established connections; close observation revealed that the remaining connections were in synchronization and closing states. Since the pre-generated representations could be directly delivered without adaptation, the connections were relatively short, and therefore, some of them were found not in the established state. The server's performance, in turn, was determined more by the connections' availability rather than by the availability of the server's other resources. As noticed in the curve, around half of the periodic numbers of concurrent connections are zero. This happened because the server ran out of sockets, so new connections could not be set up. The TIME_WAIT timeout may be the reason of this connection blockage. For every connection being closed, the server has to wait for some time to handle packets still in the network. These slow-to-be-released connections reduce the number of active sockets and, if there is no available socket left, may block or delay new connections. The curve also shows that most of the time (more than 80% of the collected data) the number of concurrent connections is 25 or less.

The number of concurrent connections in the JP2-MOD scheme (curve *jp2-mod*) topped at 200. Examining the data traces reveals that the number of concurrent connections increased gradually and reached 200 before it declined. The modulation processes running at the server may be the plausible explanation. The multiple connections caused the server's load to increase, and as a result, the server could not set up new connections fast enough and less number of established connections resulted; this is in agreement with the increased average connection time. At the same time, the fast modulation processes also restrained the number of established connections from going

beyond 200. About 10% of the periodic numbers of concurrent connections are less than 100, 30% of them between 100 and 150, and 60% of them in the range of 150 to 200.

In the JPG-SDT scheme (curve *jpg-sdt*), most of the time (about 60% of the collected data) the number of concurrent connections reached 299–300, which is the maximum possible number. This was due to the transcoding processes taking quite some time to complete so that the number of concurrent connections rapidly increased to the maximum. Only 10% of the periodic numbers of concurrent connections have values less than 200, another 10% between 200 and 275, and the remaining 20% in the range of 275 to 299.

This preliminary stress test corroborates our early assertion that the server employing modulation performs much better than the server employing transcoding (either SDT or FDT). Under a heavy load, the average response time in the JP2-MOD scheme is at least 14 times faster than that in the JPG-SDT scheme. More over, sixty percent of the time, the number of concurrent connections in the JP2-MOD scheme is between 150 and 200, but that in the JPG-SDT counterpart is close to 300 – the maximum possible number of concurrent connections.

In the next experiments, all schemes – including the JPG-FDT – were examined again under a normal condition, meaning that the server's resources are not stretched to their limits. The JPG-FDT scheme, which seemed to perform the worst, was used to determine the load applied to the server. After several trials, we found that the JPG-FDT scheme could only succeed if the maximum number of concurrent connections was limited to 30. Accordingly, the number of requests was reduced to 600; that means 100 requests for each image's representation. The bandwidth for each connection was still

restricted to 1 Mbps. Even under this condition, the JPG-FDT scheme took about 5.5 hours to complete; for comparison, the no-adaptation, JP2-MOD, and JPG-SDT schemes took roughly 0.5, 1, and 9 minutes to complete. The test's results are shown in Table 8.5.

Table 8.5 Results of stressing the server running adaptation [max. concurrent connections = 30]

Scheme	No. of Reqs*	Attained Req Rate (req/sec)	Attained Xfer Rate (KB/sec)	Conn. Time+ (milliseconds)	Resp. Time+ (seconds)
No-adapt	600 (600)	22.65	2,168.02	1.86 (1)	0.84 (0.94)
JP2-MOD	600 (600)	10.84	1,037.61	1.84 (1)	1.99 (1.75)
JPG-SDT	600 (600)	1.14	108.98	1.85 (1)	22.42 (22.15)
JPG-FDT	600 (600)	0.03	2.97	4.96 (1)	938.98 (977.40)

Note: * values are *sent* (*completed*); + values are *mean* (*median*)

As shown in the table, under the normal condition, all requests were successfully served. Compared to the previous results, the attained request rate in the no-adaptation scheme dropped 60% (i.e., from 58 to 23 requests/second), that in the JP2-MOD scheme dropped 21% (i.e., from 14 to 11 requests/second), and that in the JPG-SDT scheme remained unchanged (at 1 request/second). The attained request rate in the JPG-FDT scheme was extremely low, just 2 requests/minute. Logically, the attained transfer rate always follows the request rate; thus, the no-adaptation and JPG-FDT schemes respectively had the highest and lowest transfer rates. The no-adaptation, JP2-MOD, and JPG-SDT schemes yielded similar average connection time, around 1.85 milliseconds. The average connection time in the JPG-FDT scheme was slightly higher at about 5 milliseconds. While the average response time in the no-adaptation scheme was reasonably low (0.84 seconds), that in the JP2-MOD scheme was just 2.4 times higher (about 2 seconds). By contrast, the average response times in the JPG-SDT and JPG-FDT

schemes were 26.6 and 1114.8 times higher than that in the no-adaptation scheme. The JPG-FDT scheme, in particular, performed very badly. Twenty two seconds (i.e., the average response time in the JPG-SDT scheme) is already considered too long to display an image's representation, not to mention 15–16 minutes (i.e., the average response time in the JPG-FDT scheme). The low average connection times in all schemes meant that the server did not experience much trouble in setting up 30 concurrent connections. However, the high average response times in the JPG-SDT and JPG-FDT schemes meant that the server was heavily loaded. Again, the results prove that transcoding is resource-consuming. Modulation, on the other hand, is less burdensome due to the nature of its processes, which involve much less or no complex computations.

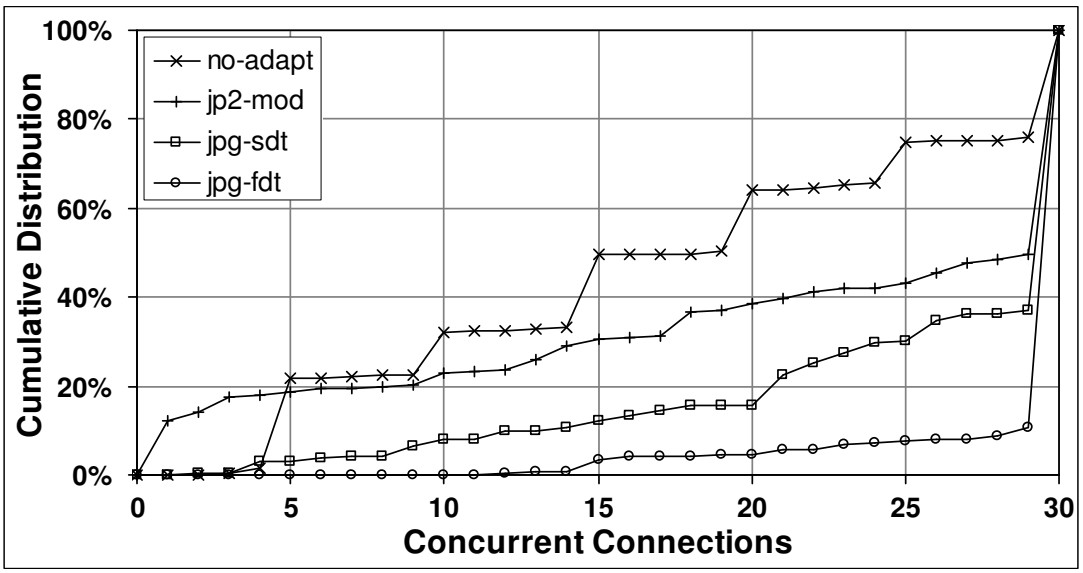


Figure 8.6 CDFs of periodic numbers of concurrent connections (max. 30) while stressing the server

The cumulative distributions of the periodic numbers of concurrent connections in different schemes are depicted in Figure 8.6. In the no-adaptation scheme (curve *no-adapt*), the data are mainly distributed on numbers which are products of five, such as 5, 10, 15, and so forth. The plausible explanation can be found in how the benchmarking

application, `ab`, operates. To reach 30 concurrent connections, we had configured the `ab` application to send five requests for the first image's representation, followed by five requests for the second image's representation, and the pattern continues until the last – the sixth – image's representation. Since no adaptation was involved, those five requests – being sent out concurrently – might be completed in the same time, causing the periodic numbers of concurrent connections to be crowded by products of five.

The cumulative distributions in Figure 8.6 can be contrasted by evaluating the percentages of data representing the maximum number of concurrent connections, which is 30 by design. Those percentages are roughly 25%, 50%, 60%, and 90% for the respective no-adaptation, JP2-MOD, JPG-SDT, and JPG-FDT schemes. The higher is the percentage, the heavier is the server's load. Unsurprisingly, the no-adaptation scheme gives the best performance, followed by the JP2-MOD and JPG-SDT schemes. The JPG-FDT scheme exhibits the worst performance.

To sum up, we have discussed experiments on the server-based adaptation to figure out its costs and benefits. Adaptation at the server is particularly beneficial if the bandwidth of the server-proxy link is restricted. One cost can be measured from the resulting response time, which can be broken down into the delivery latency and the adaptation latency. The delivery latency is unavoidable, whereas the adaptation latency is the cost of performing server-based adaptation. The other cost comes in form of decrement in performance.

Three distinct adaptation schemes had been examined in the experiments; they are JPEG spatial-domain transcoding (JPG-SDT), JPEG frequency-domain transcoding (JPG-

FDT), and JPEG 2000 modulation (JP2-MOD). Analyses on those three adaptation schemes show that the JP2-MOD scheme gives the lowest cost; its ratio of adaptation latency to total response time ranges from 0.3–1.1% on 56 Kbps connection to 4.7–15.9% on 1 Mbps connection. Comparing that ratio to those resulting from other schemes, the JP2-MOD scheme is 4–20 times better than the JPG-SDT scheme and 5–18 times better than the JPG-FDT scheme.

With regard to the server's performance under a heavy load (300 concurrent connections), the JP2-MOD and JPG-SDT schemes managed to serve 14 and 1 requests/second, which were 76.5% and 98.1% drops from 58 requests/second in the no-adaptation scheme (i.e., the server without adaptation). However, under a normal load (30 concurrent connections), the JP2-MOD and JPG-SDT schemes were able to serve 11 and 1 requests/second, representing 52.1% and 95.0% drops from 23 requests/second in the no-adaptation scheme. Surprisingly, the JPG-FDT scheme performed very badly due to its high demand for memory space; it crashed under a heavy load and just managed to serve 2 requests/minute under a normal load. While the average response time in the JP2-MOD scheme was quite fast (i.e., 2 seconds), those in the JPG-SDT and JPG-FDT schemes were considered too long (22 and 939 seconds, respectively). Moreover, analyses on the periodic numbers of concurrent connections reveal that the JPG-SDT and JPG-FDT schemes required 30 concurrent connections (the maximum possible number) 60% and 90% of the time, but the JP2-MOD topped at that number half of the time.

Evidently, the costs of implementing modulation at the server are lower than those of implementing transcoding. Although the benefits of server-based adaptation are scarce, it is easier to manage (less number of Web resources) and to extend (adding more types of

adaptation) than providing multiple pre-generated versions of Web resources. In our framework, the server-based adaptation is coupled with the proxy-based adaptation to give greater benefits. The benefits will be demonstrated shortly in the following section.

8.3 Evaluating Adaptation at Web Server and Proxy

Our framework suggests greater collaboration between the origin server and the proxy in delivering the best-fit representations for heterogeneous clients. The proposed system is proxy-centric, in which the proxy decides the representation to be served to a particular client. For the decision-making process to succeed, the proxy requires information about the object's characteristics and some guidance from the server, in addition to information about the client's characteristics. All of this information is available in form of the server and client meta-data documents.

The current evaluation tested and analyzed the performance of our model prototype, which was built according to the proposed framework. Each experimented image was accompanied by an associated ADP document (i.e., the server meta-data document) residing in the server. The three ADP documents used in the experiments – namely, `boat.jpeg.adp`, `boat.jpg.adp`, and `boat.jp2.adp` – have been described in Subsection 8.1.3. Each client's request carried a reference (URI) to the client's CC/PP document (i.e., the client meta-data document) in its `CCPP` header. The experiments used six CC/PP documents (`Client1`–`Client6`), each of which corresponded with a particular image's representation. Details about the CC/PP documents can be found in Subsection 8.1.4.

Aside from the meta-data documents, some matching rules had to be configured in the proxy so that the expected representation was correctly selected by the decision-making process. The defined variables and installed matching rules are as follows:

```
# CC/PP Variables
ccpp_def dev_width H -i pix-x
ccpp_def dev_height H -i pix-y
ccpp_def dev_color H -i color
ccpp_def dev_scrollbar B -i scrollbar

# ADP Variables
adp_def rep_width GEN -i width
adp_def rep_height GEN -i height
adp_def rep_gray IMG -i grayscale

# Matching Rules
obj_match 0 [rep_width + 10 <= dev_width]
obj_match 1 [(rep_height <= dev_height) |
  (dev_scrollbar $ 4 = "vert")]
obj_match 2 [rep_gray | (dev_color = "full")]
```

The top four lines define CC/PP variables – namely *dev_width*, *dev_height*, *dev_color*, and *dev_scrollbar* – corresponding with attributes of a CC/PP document which names contain words "pix-x" (display width), "pix-y" (display height), "color" (supported color), and "scrollbar" (browser's scrollbars). The next three lines define ADP variables – namely *rep_width*, *rep_height*, and *rep_gray* – corresponding with attributes of an ADP document which names contain words "width" (presentation width), "height" (presentation height), and "grayscale" (grayscale image). At last, using the defined variables, three matching rules are specified. Rule 0 says that the addition of the representation's width and 10 should be less than or equal to the display width. Rule 1 says that representation's height should be less than or equal to the display height, but the rule may be ignored if the browser has a vertical scrollbar. The last rule basically prevents a color representation to be displayed on a device supporting grayscale only.

In the following subsections, three sets of experiments are discussed. The first is response time analyses between server/proxy-based (meaning a combination of server- and proxy-based) adaptation and merely server-based adaptation. The second is a stress test applied to the systems running server-based and server/proxy-based adaptation. Finally, the third set of experiments demonstrates the benefits of data reuse. Before discussing the experiments, let us start with the objectives of this evaluation.

8.3.1 Experimental Objectives

Similar to the previous evaluation, the primary objective of evaluating adaptation at the server/proxy is to analyze its costs and benefits. We believe that the benefits of server/proxy-based adaptation should be greater than those of server-based adaptation. The experiments here will elaborate those benefits. Pragmatically, we may have to trade the benefits in certain aspects for some overheads in other aspects. This evaluation is expected to unveil those overheads, too.

The secondary objective is to understand the distinct characteristics of server/proxy-based adaptation in our proposed framework. These experiments may reveal invaluable information that can be used to improve the framework and further open up new research ideas, particularly in pervasive Web content delivery.

8.3.2 Response Time Analyses

Our proposed framework greatly exploits data reuse of Web objects. A typical caching proxy stores the object received from the origin server and uses it to respond

subsequent client requests for the same object. The stored object may be the product of adaptation, reusability of which is usually low. However, with the proliferation of scalable data formats such as MPEG-4, JPEG 2000, and H-264 standards, data reusability can now be improved. Our caching proxy may reuse the cached representation – of a particular object, for instance – to serve not only requests for the same object's representation but perhaps also requests for other representations of the object. In our implementation (model prototype), the reuse of a cached object's representation can be specified in the proxy. As noted in the previous chapter, any adaptor can be plugged into our enhanced proxy so long as it is coupled with a suitable adapting interface (readers are referred to Subsections 7.4.2 and 7.4.4 for development of such an adapting interface). The adapting interface determines whether or not a cached object's representation can be reused for serving a request for a different representation of the same object.

Three adapting interfaces had been devised for the model prototype. They were developed to communicate with the external JPEG transcoders (SDT and FDT) and JPEG 2000 modulators. For simplicity, again we may refer them by the schemes' names; those are JPG-SDT, JPG-FDT, and JP2-MOD. With regard to data reuse in case the requested image's representation is different from the cached image's representation, the JPG-SDT's and JPG-FDT's adapting interfaces can only reuse the cached representation if it satisfies the following three conditions:

1. It must have a higher resolution than or equal to the requested representation;
2. It must have full color components if the requested representation is also in full color;
3. It must have a higher quality value than or equal to the requested representation.

Actually, any representation resulting from transcoding should not be transcoded again since the quality of the resulting representation decreases by 0.4–2 dB, but we trade the image's quality for more data reuse. In contrast, the JP2-MOD's adapting interface generally can reuse any cached representation, either partly or fully, without sacrificing the image's quality by means of downscaling and/or upscaling operations.

The six predetermined representations of images `boat.jpeg` (for SDT), `boat.jpg` (for FDT), and `boat.jp2` – as listed in Table 8.1 to Table 8.3 – were used as the experimental test cases. By employing the above mentioned JPG-SDT's adapting interface, possible data reuses among the predetermined representations are as depicted in the diagram of Figure 8.7(a). The 1/2-scaled representation, if cached, can only be reused to generate the 1/2-scaled, gray representation. The reverse data reuse is not allowed since it would violate conditions 2 and 3 above. Likewise, the cached 1/4-scaled and 1/8-scaled representations can be reused to generate the 1/4-scaled, gray and 1/8-scaled, gray representations, respectively. The 1/2-scaled representation has a higher resolution than the 1/4-scaled and 1/8-scaled representations, but its quality value is lower than theirs; due to the third condition, it cannot be reused to generate the other two representations. The JPG-FDT's adapting interface behaves exactly the same as the JPG-SDT counterpart.

On the other hand, the JP2-MOD's adapting interface may reuse any cached representation to generate other representations, as depicted in the diagram of Figure 8.7(b). Note that the transitive law is in effect here; it means that since the 1/2-scaled representation can be reused to generate the 1/4-scaled representation and the 1/4-scaled representation can be reused to generate the 1/8-scaled representation, then the 1/2-scaled representation can also be reused (directly) to generate the 1/8-scaled representation. If a

requested representation is partly satisfied by the cached representation, additional image data is required and an upscaling operation is involved. The upscaling operations are indicated by the up-pointing arrows in the diagram.

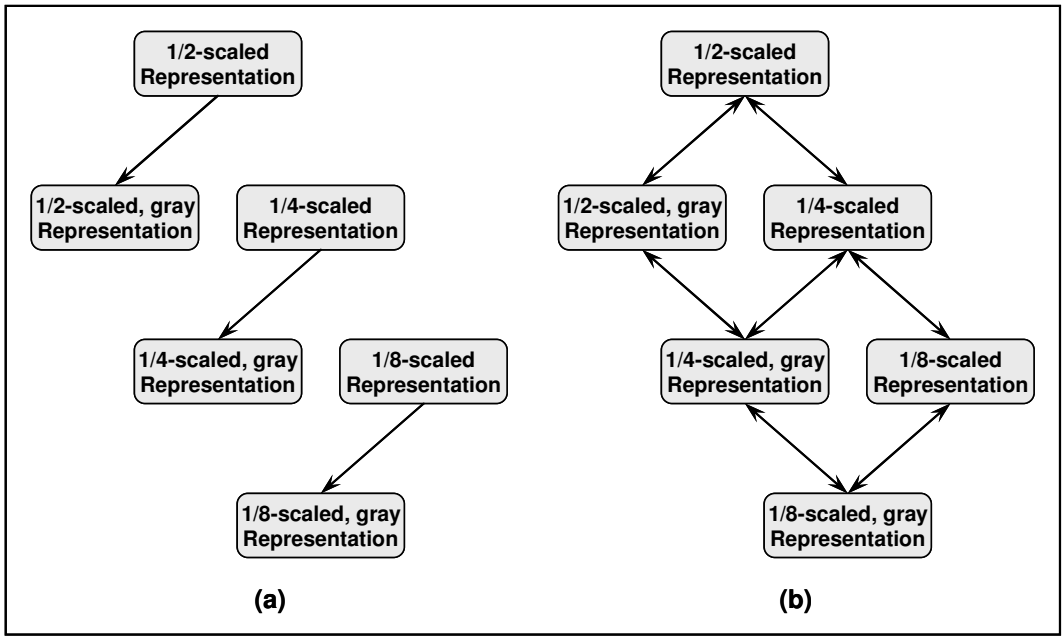


Figure 8.7 Data reuses among image representations in (a) JPG-SDT & JPG-FDT, and (b) JP2-MOD

Some experiments were conducted to analyze the costs and benefits between the system running server/proxy-based adaptation and the system only running server-based adaptation. To be fair, both systems involved a server and a caching proxy. The former employed our enhanced server and enhanced proxy; the latter employed our enhanced server and a normal proxy. The cost-benefit analyses are in term of the response times of requesting the images' representations. The analyses are carried out in two scenarios: first-time delivery and subsequent delivery. [To some extent, we are trying to corroborate our analytical evaluation in Section 6.2.] The experiments were tested on the JPG-SDT, JPG-FDT, and JP2-MOD schemes, which have also been used in the prior experiments.

For the experiments with the system running server-based adaptation, each generated request included a specific query string to indicate the wanted image's representation. This is because the proxy was not involved in the adaptation (see also Subsection 8.2.2). However, the proxy could still cache the images' representations.

In contrast, for the experiments with the system running server/proxy-based adaptation, the image's resource was directly requested without an extra query string, but each request carried a reference (URI) to one CC/PP document – out of the six – in its `CCPP` header to indicate the client's profile. Using the specified matching rules, the enhanced proxy would make a decision about the best-fit representation according to the given client's profile and the image's characteristics (retrieved from the server).

All response times in the experiments were measured at the proxy. We did not dedicate a special machine to be a client. Besides, the striking contrast between the server-based and server/proxy-based adaptation takes place along the server-proxy link, not on the proxy-client link. It was also assumed in the experiments that there might be only one representation of an image in the cache at any given time; this condition was necessary to make the analyses possible and comparable. The experimental results and discussion are given in the following paragraphs. The first-time delivery's results are discussed first, and then, the subsequent delivery's results.

A. First-time delivery

Figure 8.8 compares the response times of requesting the images' representations for the first time between the server-based adaptation (*s-adp*) and the server/proxy-based adaptation (*sp-adp*). The results are classified according to the requested representations

(notice the horizontal axis). For each representation, there are two collections of bars; the left-hand side is the results of server-based adaptation and the right-hand side the results of server/proxy-based adaptation. Each collection comprises three bars corresponding with the employed schemes – i.e., JPG-SDT, JPG-FDT, and JP2-MOD.

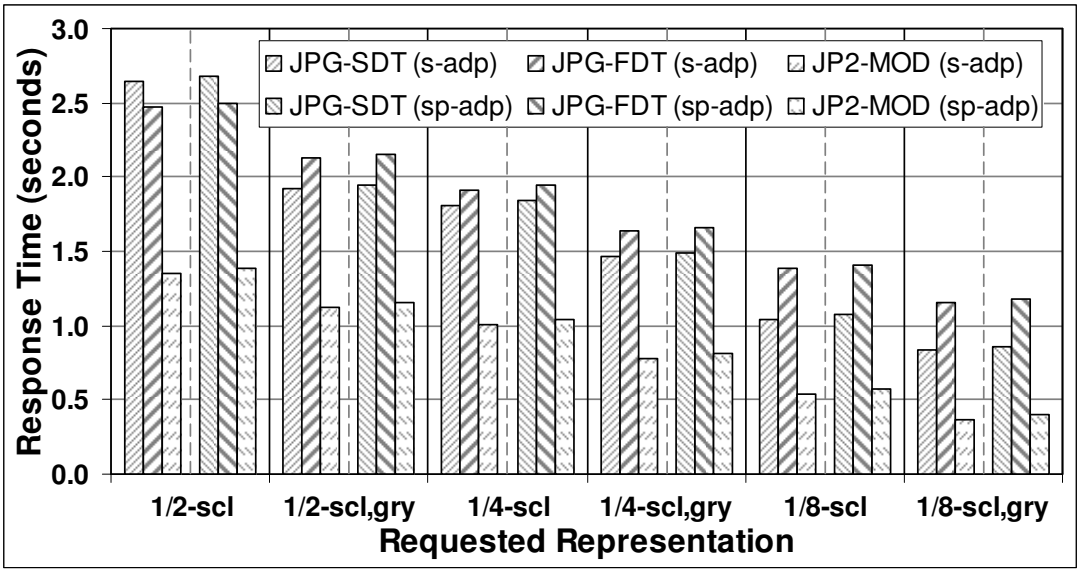


Figure 8.8 Response times of requesting image representations (all schemes) in first-time delivery

As seen in the figure, the response times of requesting the representations for the first time in *s-adp* and *sp-adp* do not differ so much. The response times in *sp-adp* are slightly higher than those in *s-adp*, owing to the decision-making process involved. The excess delays caused by the decision-making process in *sp-adp* are considered costs. The average costs in the respective JPG-SDT, JPG-FDT, and JP2-MOD schemes are about 28, 28, and 35 milliseconds. The JP2-MOD scheme has a higher cost than the other schemes probably because image `boat.jp2` had more representations in its ADP document than the specified numbers of representations for images `boat.jpeg` and `boat.jpg`; the ADP documents defined twenty-one representations for `boat.jp2` compared to fifteen representations each for `boat.jpeg` and `boat.jpg`. Conceivably, the greater the

number of representations, the longer is the time taken by the decision-making process to accomplish.

B. Subsequent delivery

The proxy may satisfy a subsequent request by reusing the representation that has been cached in the first-time delivery or, conversely, fetching the requested representation from the origin server. The former is deemed a cache *hit* and the latter a cache *miss*. A cache hit usually can be served faster than a cache miss because the image data is delivered directly from the proxy, which is closer to the client. Consequently, a cache hit has a lower cost than a cache miss.

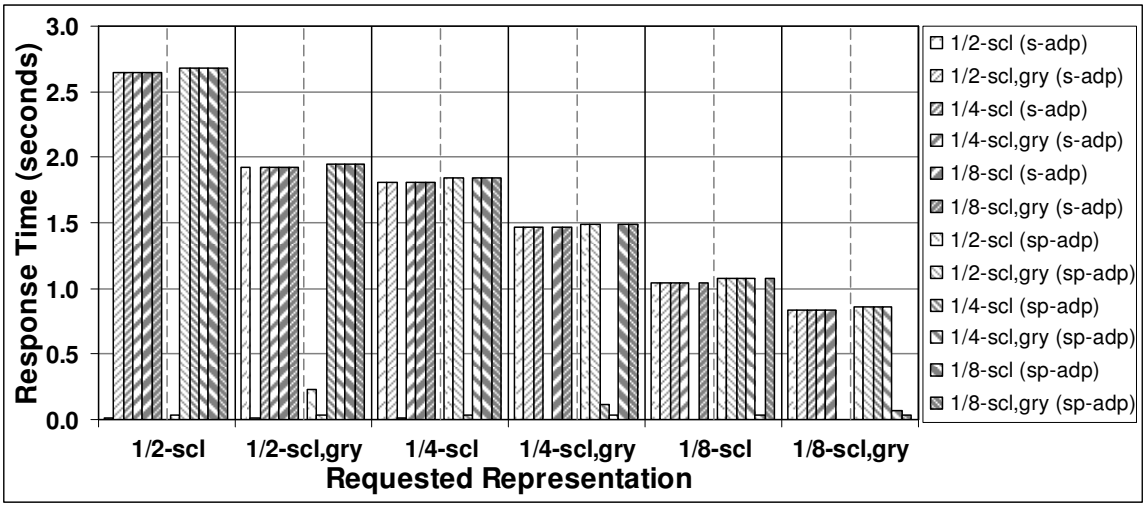


Figure 8.9 Response times of requesting image representations (JPG-SDT) in subsequent delivery

Figure 8.9 compares the response times of requesting `boat.jpeg`'s representations (adapted by the JPG-SDT scheme) in the subsequent delivery between the server-based (*s-adp*) and server/proxy-based (*sp-adp*) adaptation. The results are also classified according to the requested representations. Since it is a subsequent request, a representation from the previous request is supposed to exist in the cache. The bar chart in

the figure takes all possible cached representations into consideration. Each bar signifies the response time if a particular representation (indicated in the legend) is in the cache.

In *s-adp*, a cache hit occurs when the requested representation precisely matches the cached representation; in that case, the response time was very short, just about 5–6 milliseconds in our experiments. Otherwise, it is a cache miss and the requested representation has to be satisfied from the server; therefore, the resulting response time was not different from that in the first-time delivery.

In *sp-adp*, besides the typical cache hit above, the proxy may adapt the cached representation to generate the requested representation. As prearranged (refer to Figure 8.7(a)) and expected, there are three occasions in Figure 8.9 where the cached representation can be reused to generate the requested representation. The first occasion is when the 1/2-scaled, gray representation is requested and the 1/2-scaled one in the cache. The second is when the 1/4-scaled, gray representation is requested and the 1/4-scaled one in the cache. And the third is when the 1/8-scaled, gray representation is requested and the 1/8-scaled one in the cache. For the typical cache hit, *sp-adp* took about 33–35 milliseconds; the excess delay due to the decision-making process was also observed here. For the cache hit with adaptation, *sp-adp* took between 67 and 233 milliseconds (or, about 8–12% of the response times in the associated cache misses).

In general, *sp-adp* lost to *s-adp* if the requested representation was one of the 1/2-scaled, 1/4-scaled, and 1/8-scaled representations. In the remaining cases, *sp-adp* might win. Considering the average response time, for the subsequent delivery using the JPG-SDT scheme, the server/proxy-based adaptation (*sp-adp*) proves to be more beneficial, albeit marginally. Later on, detailed calculations of the benefits will be presented.

Similarly, the response times of requesting `boat.jpg`'s representations (adapted by the JPG-FDT scheme) in the subsequent delivery between *s-adp* and *sp-adp* are compared in Figure 8.10. A cache hit in *s-adp* corresponded to a 5–6 millisecond delay, whereas that in *sp-adp* was translated to a 33–35 millisecond delay; notice the similarity of both delays as compared to the previous results. Since the JPG-FDT scheme in *sp-adp* also exploits more data reuse, a cache hit with adaptation may occur, as well. Depending on the cached representation, the cache hit with adaptation took between 57 and 163 milliseconds (or, roughly 5–8% of the response times in the associated cache misses).

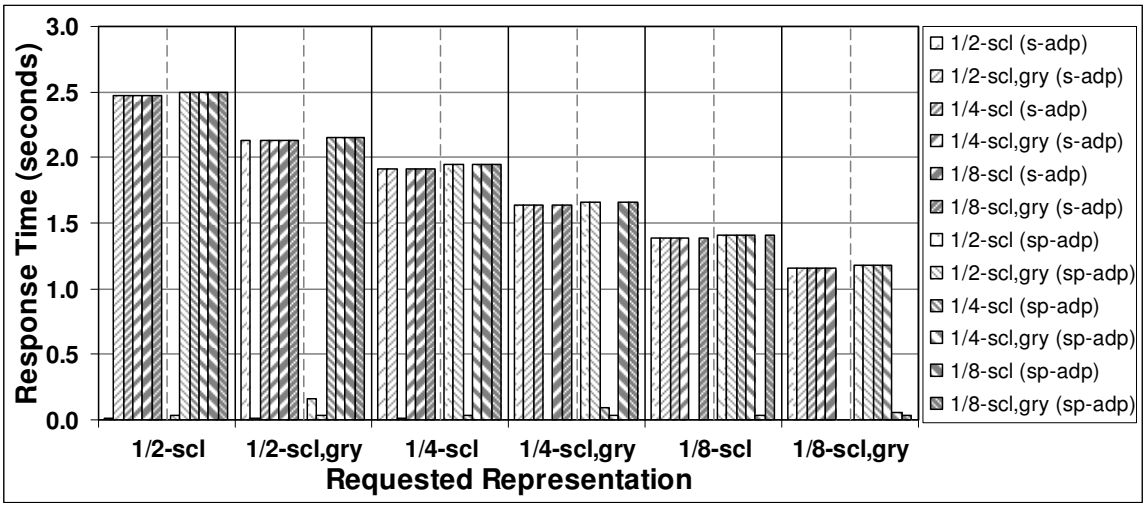


Figure 8.10 Response times of requesting image representations (JPG-FDT) in subsequent delivery

Like the previous results, *sp-adp* also won some cases and lost some to *s-adp*. Still, for the subsequent delivery using the JPG-FDT scheme, the overall results show that *sp-adp* manages to better *s-adp*.

Figure 8.11 compares the response times of requesting `boat.jp2`'s representations (adapted by the JP2-MOD scheme) in the subsequent delivery between *s-adp* and *sp-adp*. In *s-adp*, benefits were obtained in cases of cache hits, which took only 5–6 milliseconds to serve the requests. In *sp-adp*, those cache hits required about 39–41

milliseconds to complete; it means that there was an excess delay of 34–35 milliseconds for the decision-making process. In addition, more benefits can be obtained in *sp-adp* due to modulation's support for high data reuse. Any cached representation can be reused – either partly or fully – to generate the requested representation (see also Figure 8.7(b)). As observed in the bar chart, the response times in *sp-adp* are less than those in *s-adp*, except for the cases of cache hits in *s-adp*. Clearly, for the subsequent delivery using the JP2-MOD scheme, *sp-adp* significantly outperforms *s-adp*.

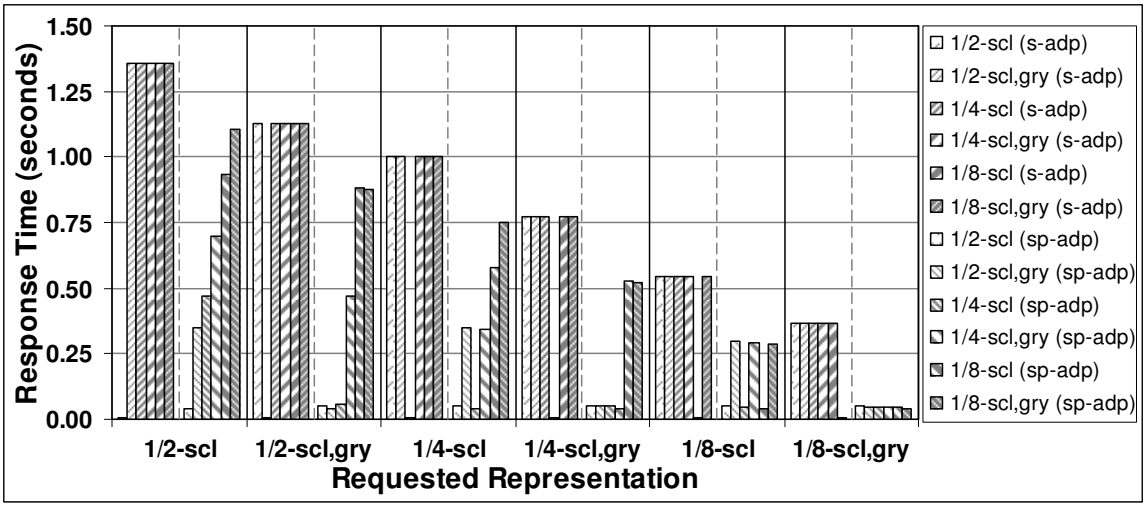


Figure 8.11 Response times of requesting image representations (JP2-MOD) in subsequent delivery

Having assessed the response times in the first-time and subsequent deliveries, we can now analyze the total benefit/cost of the server/proxy-based adaptation (*sp-adp*) as compared to the server-based adaptation (*s-adp*). For each requested representation, the benefit/cost is determined by the difference between the response time in *sp-adp* and that in *s-adp*. It is regarded a gain (benefit) if the response time in *sp-adp* is less than that in *s-adp*; otherwise, it is a loss (cost). Thus, the formula to determine the benefit/cost is

$$\delta = \textit{s-adp}\_\text{response\_time} - \textit{sp-adp}\_\text{response\_time}$$

The server/proxy-based adaptation is more beneficial (detrimental) than the server-based adaptation if the value of δ is positive (negative). Using the formula, we calculate the total benefit/cost of *sp-adp* as compared to *s-adp*, with respect to the employed schemes – namely JPG-SDT, JPG-FDT, and JP2-MOD. The calculations are detailed in Table 8.6.

Table 8.6 Total benefit/cost of server/proxy-based adaptation as compared to server-based adaptation

Requested Representation	JPG-SDT			JPG-FDT			JP2-MOD		
	1st	Ssq	Total	1st	Ssq	Total	1st	Ssq	Total
1/2-scaled	(-28.3)	(-28.4)	(-28.3)	(-28.4)	(-28.4)	(-28.4)	(-34.2)	530.6	248.2
1/2-scaled, gray	(-28.0)	258.6	115.3	(-28.5)	304.2	137.8	(-35.0)	543.2	254.1
1/4-scaled	(-27.4)	(-27.5)	(-27.4)	(-27.8)	(-27.8)	(-27.8)	(-34.0)	484.7	225.4
1/4-scaled, gray	(-28.3)	200.9	86.3	(-27.8)	235.0	103.6	(-35.3)	440.0	202.4
1/8-scaled	(-27.3)	(-27.4)	(-27.4)	(-27.8)	(-27.8)	(-27.8)	(-34.3)	284.3	125.0
1/8-scaled, gray	(-28.0)	104.0	38.0	(-27.4)	160.5	66.5	(-35.2)	261.0	112.9
Total B(C)			**26.1**			**37.3**			**194.6**

All values in Table 8.6 are in milliseconds. The calculations are done for each requested representation. As noticed, there are three columns for each employed scheme. The first column (with header "1st") is the calculated benefit/cost in the first-time delivery. The second column (with header "Ssq") is the calculated benefit/cost in the subsequent delivery; it is the average response times' difference of the six cases of cached representations. The third column (with header "Total") is the total benefit/cost of requesting the respective representation; it is the average of the previous two columns. At the bottom of the table (the last row), the total benefit/cost of server/proxy-based adaptation with respect to that particular scheme is determined; it is the average of the values in column "Total". Let us now examine the results for each scheme.

In the JPG-SDT scheme, any representation requested for the first time is always a loss, whereas the subsequent delivery may be a gain or a loss, depending on the requested representation. Combining the first-time and subsequent deliveries, requesting the grayscale representations may be beneficial whereas requesting the full color representations is perhaps detrimental. But overall, employing the JPG-SDT scheme, the server/proxy-based adaptation is slightly more beneficial than the server-based adaptation since the average response time is faster by 26.1 milliseconds.

Employing the JPG-FDT scheme gives similar characteristics to employing the JPG-SDT scheme. So, requesting the grayscale representations may offer more benefits than requesting the full color representations. In this scheme, the server/proxy-based adaptation is also more beneficial than the server-based adaptation; the average response time in the former is less than that in the latter by 37.3 milliseconds.

Requesting any representation for the first time in the JP2-MOD scheme is also a loss. But the subsequent request has a very high chance to be beneficial. All combined results of the first-time and subsequent deliveries are beneficial, too. In general, employing the JP2-MOD scheme, the server/proxy-based adaptation may gain 194.6 milliseconds in the average response time over the server-based adaptation.

It can be inferred from the above results that the server/proxy-based adaptation is more advantageous than the server-based adaptation in any scheme employed. Moreover, the JP2-MOD scheme gives the highest total benefit as compared to the total benefits of the JPG-SDT and JPG-FDT schemes. The gain from the average response time in the JP2-MOD scheme is greater than those in the JPG-SDT and JPG-FDT counterparts by factors of 7.5 and 5.2, respectively.

8.3.3 Stress Test

Like the previous evaluation of adaptation at the server, the evaluation of adaptation at the server/proxy is not complete without running a stress test on the system. With the help of a caching proxy (either original or enhanced), the server's load is expected to decrease. Since cases of cache hits can shorten the response times, the average response time is also expected to be lower than the average response time of the adaptation system without a caching proxy.

In accordance with the previous response time analyses, two distinct systems had been stress-tested in the experiments. Both systems consisted of a server and a caching proxy. The first system, comprising our enhanced server and a normal (original) proxy, employed server-based adaptation. The second system, comprising our enhanced server and proxy, employed server/proxy-based adaptation.

The same benchmarking application – i.e., `ab` of the ApacheBench package – was employed to stress-test both systems. It was carried out at the proxy; thus, a dedicated client's machine was unnecessary. In addition, the same three adaptation schemes – namely JPG-SDT, JPG-FDT, and JP2-MOD – plus the no-adaptation scheme were also compared and contrasted in the experiments. Six hundred requests were sent out during the stress test; it means 100 requests for each image's representation. Similarly, the number of concurrent connections was monitored periodically, but this time there were two links – i.e., the proxy-to-server and client-to-proxy links – to monitor. The maximum number of concurrent connections was set to thirty (30). In the adapting proxy, ten (10)

adaptors – depending on the scheme – were run simultaneously to serve requests for adaptation. The bandwidth of each connection to the server was also limited to 1 Mbps.

A. System employing server-based adaptation

Different from the previous assumption, in the system, there may be more than one image's representation stored in the proxy's cache. This is actually the typical behavior of the original proxy. Since objects in the cache are identified by the requests and since every request for an image's representation is unique (due to the attached query string at the end of the request's URI), all representations of the image may be stored as distinct objects in the cache. The benefit of storing multiple representations of an image in the proxy's cache is the avoidance of repeated adaptation processes. Furthermore, if there are many representations stored in the cache, chance is higher that a cache hit may happen for the next request for the image. However, the downside is that adaptation at the proxy may not be feasible. For the adaptation at the proxy to work effectively, the cached representation should be able to be identified so that a decision can be made whether the requested representation can be generated from the cached one. If multiple representations are allowed to be stored in the cache, the identification and decision-making processes will be difficult, if possible, to do. Some reasons may be suggested. First of all, it is hard to tell which cached objects are indeed the representations of the requested image; this is because they are considered distinct objects. Secondly, since direct identification of the existing representations in the cache is difficult to do, another alternative is to check exhaustively whether certain representations are in the cache, but this may be inefficient. Lastly, even if the cached representations can be identified,

selection has to be made with regard to the one used for adaptation; this makes the overall adaptation procedure more complex. Besides that, other drawbacks of caching multiple representations include more disk space's consumption and difficulty in maintenance.

Nevertheless, the use of a proxy in this system was merely for caching the representations and not for adaptation. The adaptation was done only at the server; hence, it is termed server-based adaptation. The test's results are presented in Table 8.7.

Table 8.7 Results of stressing the server/proxy system employing server-based adaptation

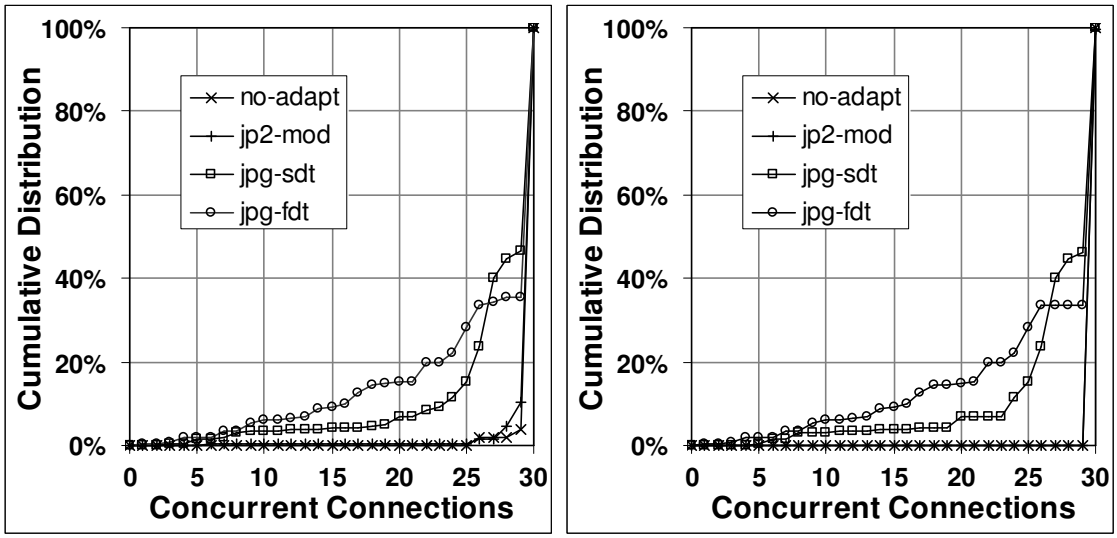
Scheme	No. of Reqs*	Attained Req Rate (req/sec)	Attained Xfer Rate (KB/sec)	Conn. Time+ (milliseconds)	Resp. Time+ (seconds)
No-adapt	600 (600)	213.45	20,436.71	1.03 (0)	0.12 (0.06)
JP2-MOD	600 (600)	153.77	14,722.56	1.01 (0)	0.17 (0.06)
JPG-SDT	600 (600)	21.69	2,070.81	0.91 (0)	1.17 (<0.01)
JPG-FDT	600 (600)	1.26	120.81	0.98 (0)	20.93 (<0.01)

Note: * values are *sent* (*completed*); + values are *mean* (*median*)

Compared to the system comprising just the server (called the server system, for simplicity), the server/proxy system's performance is greatly improved. This can be verified by contrasting the results presented in Table 8.7 with those presented in Table 8.5. The attained request rates increase by factors of 9.4, 14.2, 19.0, and 40.6 in the respective no-adaptation, JP2-MOD, JPG-SDT, and JPG-FDT schemes. Another key performance indicator is the average response time, which the current system manages to speed up by 7 to 45 times as compared to the server system. Obviously, the server/proxy system outperforms the server system.

Figure 8.12 depicts the cumulative distributions of the periodic numbers of concurrent connections on the proxy-to-server and client-to-proxy links. As seen in both

graphs of the figure, the characteristics of the proxy-to-server concurrent connections are very similar to those of the client-to-proxy counterparts. Examining the data traces indeed reveals that the periodic numbers of concurrent connections in both links were almost the same most of the time. It seems that one request on the proxy-to-server link determined several similar requests on the client-to-proxy link. Once a representation had been retrieved from the server and stored in the proxy's cache, the following requests for that representation could be served very briskly. That is why the number of client-to-proxy's concurrent connections resembled that of proxy-to-server's concurrent connections during the stress test.



(a) Proxy-to-Server link **(b) Client-to-Proxy link**

Figure 8.12 CDFs of periodic numbers of concurrent connections while stressing the server/proxy system employing server-based adaptation

The number of concurrent connections in the no-adaptation and JP2-MOD schemes was almost at the maximum (30) most of the time. On the proxy-to-server link, only 5% and 10% of the periodic numbers of concurrent connections for the two schemes have values in the range of 25 to 29; the remainders, which are 95% and 90%

respectively, are 30. On the client-to-proxy link, thirty concurrent connections were occupied all the time in both schemes. The similarity between the no-adaptation and JP2-MOD schemes means that adaptation by modulation is very fast.

In the graphs, curve *jpg-sdt* is steeper than curve *jpg-fdt*, particularly in the range of 20 to 29; it means that the periodic numbers of concurrent connections are more distributed if the JPG-FDT scheme, rather than the JPG-SDT scheme, is employed. Nevertheless, the graphs also demonstrate that the JPG-SDT scheme performs better than the JPG-FDT scheme in the tested system. Employing the JPG-SDT scheme, the system reached the maximum number of concurrent connections about half of the time. By contrast, employing the JPG-FDT scheme, the system occupied 30 concurrent connections almost 70% of the time.

B. System employing server/proxy-based adaptation

In the second system, which employed our enhanced server and proxy, adaptation might be carried out at the server or at the proxy, depending on the circumstances. Thereby, it is termed the server/proxy-based adaptation. The proxy may adapt the cached representation if the conditions allow it (refer to the diagrams of data reuses in Figure 8.7). If the proxy cannot reuse the cached representation, adaptation to the original image is then carried out at the server. Contrary to the condition in the previous server/proxy system employing server-based adaptation, here the proxy may keep only one representation of an image in its cache at any given time since the objective is to exploit data reuse by means of adaptation. Consequently, only the three adaptation schemes could be tested; the no-adaptation scheme was not involved in the stress test. Running the stress

test on this system produced rather diverse results. This is understandable since adaptation at the proxy depends on what representation available in the cache, and this may affect the overall system's performance. However, some forms of consistency can still be found in the results. Table 8.8 shows the test's results.

Table 8.8 Results of stressing the server/proxy system employing server/proxy-based adaptation

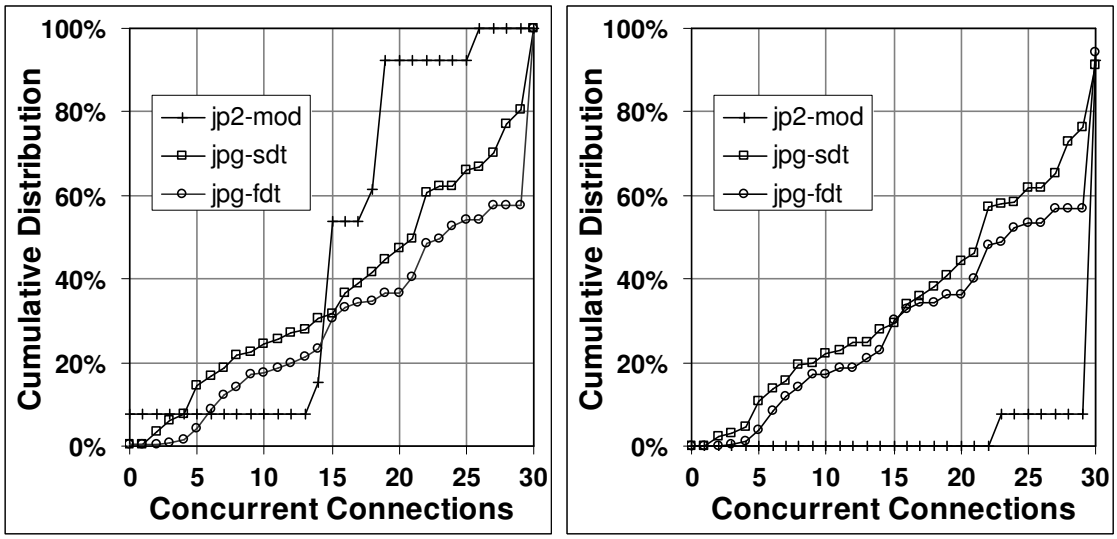
Scheme	No. of Reqs*	Attained Req Rate (req/sec)	Attained Xfer Rate (KB/sec)	Conn. Time+ (milliseconds)	Resp. Time+ (seconds)
JP2-MOD	600 (600)	19.05	1,978.63	31.91 (0)	1.35 (0.94)
JPG-SDT	600 (600)	4.08	390.34	0.80 (0)	4.57 (0.06)
JPG-FDT	600 (600)	0.80	76.82	0.51 (0)	26.91 (0.06)

Note: * values are *sent* (*completed*); + values are *mean* (*median*)

The test's results of the current server/proxy system also show some improvements over those of the server system, although the improvements are not as great as those achieved by the server/proxy system employing server-based adaptation. Compared to the server system, the attained request rates in this server/proxy system are improved by 1.8, 3.6, and 25.9 times in the JP2-MOD, JPG-SDT, and JPG-FDT schemes, respectively. In addition, the average response times are reduced by 32.2%, 79.6%, and 97.1% in the respective schemes. We found that, among the results' metrics, the average connection time might be the most inconsistent; they varied greatly from one test to another. Nonetheless, they did not affect much the system's performance.

Compared to the server/proxy system employing server-based adaptation, the current server/proxy system employing server/proxy-based adaptation is rather inferior. The latter's attained request rates in the JP2-MOD and JPG-SDT schemes are less than one-eighth and one-fifth of the former's attained request rates. But in the JPG-FDT

scheme, the latter's attained request rate can achieve better, about 64% of the former's attained request rate. Similarly, the average response times of the server-based adaptation are faster by factors of 7.8, 3.9, and 1.3 than those of the server/proxy-based adaptation in the respective JP2-MOD, JPG-SDT, and JPG-FDT schemes. This inferiority is quite expected since the server/proxy-based adaptation allows only one representation in the proxy's cache and exploits adaptation greatly to reuse the cached representation. These conditions have contributed to higher response times, and subsequently, lower request rates. The server-based adaptation, on the other hand, can store multiple representations in its proxy's cache and attain more cache hits without the need for adapting the cached representations.



(a) Proxy-to-Server link **(b) Client-to-Proxy link**

Figure 8.13 CDFs of periodic numbers of concurrent connections while stressing the server/proxy system employing server/proxy-based adaptation

The cumulative distributions of periodic numbers of concurrent connections while stressing this system are shown in Figure 8.13. Notice that on the client-to-proxy link the number of concurrent connections can reach beyond 30 (the maximum); the cumulative

distributions are not 100% at point 30 (see Figure 8.13(b)). This is because the proxy's CC/PP and ADP parsers may retrieve meta-data documents via the proxy, and these actions may increase the number of concurrent connections to the proxy. But these extra connections are temporary and insignificant.

As depicted by curves *jpg-sdt* and *jpg-fdt*, the periodic numbers of concurrent connections are very much dispersed when the system employed the JPG-SDT or JPG-FDT scheme. The graphs also demonstrate that the JPG-SDT scheme is still better than the JPG-FDT scheme. Employing the JPG-SDT scheme, the system reached the maximum number of concurrent connections 20% of the time. The amount of time of reaching the maximum number was doubled (becoming 40% of the time) if the JPG-FDT scheme was employed.

Of interest to note are characteristics of the concurrent connections when the JP2-MOD was employed (curve *jp2-mod*). On the proxy-to-server link, almost 10% of the periodic numbers of concurrent connections are zero. This happened when the cached image was the 1/2-scaled representation, which could be reused to satisfy further requests without the need for retrieving more image data from the server. Also, the periodic numbers of concurrent connections to the server are concentrated between 15 and 25; some tests gave a higher range and some gave a lower range. But more importantly, the system employing the JP2-MOD scheme hardly ever reached 30 (the maximum) concurrent connections. By contrast, on the client-to-proxy link, most (at least, 90%) of the time requests to the system employing the JP2-MOD scheme used 30 concurrent connections or even more; thus, the requests were still served at the full speed. All of these mean that employing the JP2-MOD scheme can reduce the server's load and reuse

more the cached data. This feature is unique to the server/proxy system employing server/proxy-based adaptation and cannot be found in the previously tested systems.

8.3.4 Exploration of Data Reuse

To comprehend the benefits of data reuse in our proposed framework, further exploration is required. Some adaptation schemes were again compared in the following experiments to differentiate one scheme from another in reusing cached data. One scheme is modulation in the JPEG 2000 image standard (JP2-MOD), which has demonstrated its support for high data reuse. The diagram of Figure 8.7(b) depicts the possible data reuses among `boat.jp2`'s representations. Another scheme is transcoding in the JPEG image standard, which data reuses among `boat.jpg`'s representations are depicted in the diagram of Figure 8.7(a). Since there are two types of JPEG transcoders – SDT and FDT – and both behave similarly in data reuse, we used just one of them in the experiments. The JPG-SDT scheme was selected over the JPG-FDT scheme because the previous experimental results have shown that the former generally betters the latter. The last scheme is a relaxed version of the JPG-SDT scheme. In Subsection 8.3.2, three conditions have been suggested to guide data reuse in the JPG-SDT scheme. Basically, the cached representation can only be reused to generate the requested representation if the former has higher resolution, color feature, and quality value than or equal to the latter. In the relaxed JPG-SDT scheme, the last condition regarding the quality value is omitted. By doing this, the data reuse in the relaxed JPG-SDT scheme is higher than that in the original JPG-SDT scheme, as depicted in Figure 8.14. Employing the relaxed JPG-SDT

scheme may deteriorate the average quality of the resulting representations, but this may be compensated by the benefits reaped from data reuse. An associated adapting interface was developed to accomplish the relaxed JPG-SDT scheme.

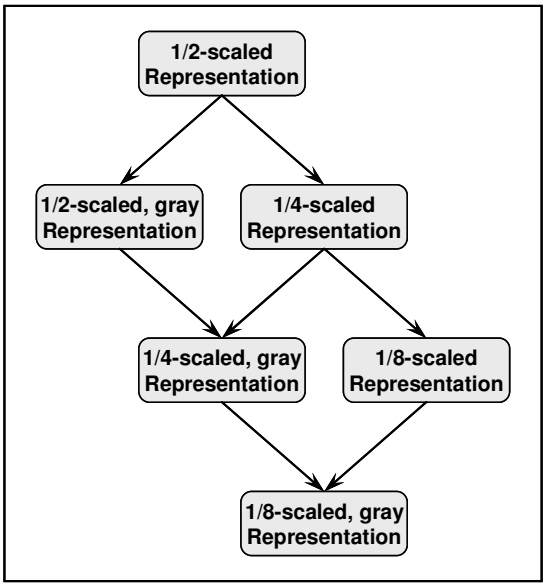


Figure 8.14 Data reuses among image representations in JPG-SDT with relaxed policy

In the following experiments, the system comprising our enhanced server and proxy was used to explore data reuse among the three adaptation schemes. Images `boat.jp2` and `boat.jpg` were used as the test data. Each request for the images also contained a reference (URI) to one out of six client profiles, each of which corresponded with a particular representation. The contained reference was generated randomly, and as a result, the images' representations were requested randomly, as well. The requests were sent out in sequence, meaning that each request must be completed before the next request can be sent out. The server also limited each connection's bandwidth to 1 Mbps. The response time of completing each request was noted, and the accumulated results are plotted in Figure 8.15.

In the JPG-SDT scheme, the response times above 0.5 seconds were cache misses and those below 0.5 seconds were cache hits, either with or without adaptation. When a cache miss happens, the cached representation is replaced by the new representation retrieved from the server. Hence, the JPG-SDT scheme still requires data retrieval from the server from time to time. As observed in the figure, the average response time of the JPG-SDT scheme is likely greater than 1.5 seconds.

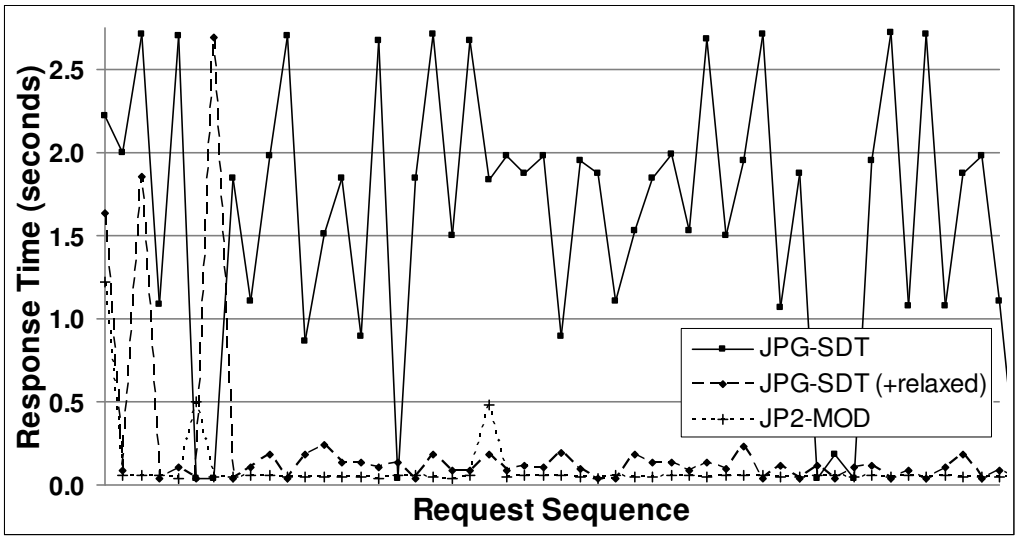


Figure 8.15 Response times of serving a sequence of requests in server/proxy-based adaptation

Similarly, the response times in the relaxed JPG-SDT scheme that are above 0.5 seconds were cache misses and those below 0.5 seconds were cache hits. Different from the original version, the relaxed version has more cache hits. As shown in the figure, for this scheme, there are only two peaks denoting cache misses. The last peak was actually the response time of requesting the 1/2-scaled representation of image `boat.jpg`. Once this particular representation was obtained, the subsequent requests could be served locally by the proxy, and therefore, there was no further cache miss. In the relaxed JPG-

SDT scheme, a cache hit without adaptation took less than 40 milliseconds whereas a cache hit with adaptation took between 90 and 240 milliseconds.

The JP2-MOD scheme emerged the winner in data reuse. By employing this scheme, whatever representation available in the cache, the proxy can always reuse it – either partly or fully – to satisfy the upcoming requests. When the cached representation can only partly satisfy a request, the proxy needs to retrieve more image data from the server to fully satisfy the request; the cached representation is not replaced but instead enhanced. The two peaks for the JP2-MOD scheme in the figure signify the times when the proxy needed to retrieve more image data from the server. In both cases, the response times were only about 0.5 seconds. The last peak, too, was the response time of requesting the 1/2-scaled representation of image `boat.jp2`. After retrieving that representation, the remaining requests resulted in cache hits. In the JP2-MOD scheme, a cache hit without adaptation took about 45 milliseconds whilst a cache hit with adaptation took between 50 and 60 milliseconds.

The results evidently corroborate our initial proposition that data reuse can reduce the average response time of serving client requests. Among the three schemes tested, modulation in the JPEG 2000 standard – which offers high data reuse – gives the lowest average response time. The relaxed JPG-SDT scheme, which is a modified version of the JPG-SDT scheme, gives higher data reuse and produces lower average response time than the original JPG-SDT scheme.

The second stage of our evaluation has analyzed the costs and benefits between the server-based adaptation and the server/proxy-based adaptation in a system comprising

a server and a caching proxy. Response time analyses have shown that the server/proxy-based adaptation is more beneficial than the server-based adaptation. The server/proxy-based adaptation gains 26.1, 37.3, and 194.6 milliseconds in the average response time over the server-based adaptation when the JPG-SDT, JPG-FDT, and JP2-MOD schemes, respectively, are employed. However, to make the analyses possible, it was assumed that only one representation of an object could stay in the proxy's cache all the time. When that assumption was removed in the server-based adaptation, we saw a great improvement in the system's performance. Compared to the system without a caching proxy, the server/proxy system employing server-based adaptation gains an increase of 9–41 times in the attained request rate and a decrease of 86–98% in the average response time. Alas, at the moment we could not repeat the same success to the server/proxy-based adaptation since the condition of one representation for each object was still upheld. Compared to the system without a caching proxy, the server/proxy system employing server/proxy-based adaptation gains an increase of 2–26 times in the attained request rate and a decrease of 32–97% in the average response time.

Further, we have explored data reuse in different adaptation schemes. In the JPG-SDT scheme, data reuse is limited to certain cases, and therefore, cache hits rarely occur. For this scheme, the attained average response time in the experiments is higher than 1.5 seconds. In the relaxed JPG-SDT scheme, the cached representation can be reused most of the time, particularly after the representation with the highest fidelity is cached. The average response time in the relaxed JPG-SDT scheme is slightly over 200 milliseconds. The JP2-MOD scheme is the winner, among the three schemes, in data reuse. In addition to the features in the relaxed JPG-SDT scheme, the JP2-MOD scheme does not replace

the cached representation but top it up with additional image data to form a representation with a higher fidelity; thereby, the cached representation can always be reused albeit, perhaps, partly. Consequently, the average response time in the JP2-MOD scheme is below 100 milliseconds, which is 18–23 times better than that in the JPG-SDT scheme and 2.4–2.5 times better than that in the relaxed JPG-SDT scheme.

8.4 Concluding Remarks

The first evaluation has shown the benefits and costs of adaptation at the server. The server-based adaptation can reduce the response time of serving a client's request, particularly if the bandwidth on the server-proxy link is limited. Instead of sending the original image to the proxy for adaptation, the server may adapt the image and deliver the adapted result. However, there are some costs of executing adaptation at the server, and most are related to the server's performance. Due to its need for memory and computation cycles, adaptation may escalate the server's load. Consequently, the attained request rate is declined and the average response time is prolonged, as compared to the original server's performance. That is why fast and efficient adaptation processes are needed. Fast alone is not enough. The frequency-domain JPEG transcoder is supposed to process faster than the spatial-domain JPEG transcoder, but due to its high demand for memory space, it is hardly usable for serving simultaneous requests. Modulation has proven to fit the requirements. It is significantly faster and more efficient than transcoding.

Another way to reduce the server's load above is to utilize a caching proxy. This has been demonstrated in the second evaluation. The caching proxy can store an image's

representation adapted and retrieved from the server and use it for serving the subsequent requests. By doing this, the need for involving the server in subsequent requests is reduced. Eventually, the average response time of serving the requests is cut down. In addition to returning exactly the requested image's representation, our enhanced proxy can adapt the cached representation and produce another (usually lower in fidelity) representation of the image. Thus, the enhanced proxy can highly reuse the cached representation and further reduce the server's involvement. Moreover, it is very efficient in utilizing the cache space. Nevertheless, there is a trade-off in accomplishing the high data reuse. In our enhanced proxy, only one representation of each image is allowed to exist in the cache at any given time. The restriction is necessary so that the adaptation can be carried out effectively in a straightforward fashion. But the loss due to the restriction is quite substantial, too. The evaluation shows that employing the normal proxy – which regards each image's representation as a distinct object – can achieve better system's performance than employing our enhanced proxy if simultaneous requests are considered. To improve the performance of our model prototype, future work is required to balance the need for high data reuse (efficiency) and the benefit of caching multiple representations (data replication).

Chapter 9

Conclusions and Future Work

The first section concludes the entire discussion – proposal, implementation, evaluation, and others alike – in the thesis. The second section suggests some future work to improve our proposed framework for pervasive Web content delivery.

9.1 Conclusions

Web content delivery is facing challenges from many fronts: users' demands, providers' needs, and technologies' advancement. Some of the challenges are clients' heterogeneity, increased multimedia data traffic, demand for customized content, and demand for easy and fast deployment of Web services. This thesis tried to address the challenges. In doing so, two objectives have been set out. The first objective is to devise a fine-grained, scalable Web data model, and the second is to design a framework for pervasive Web content delivery. Our contributions, in the efforts to achieve those objectives, are concluded in the following subsections.

9.1.1 Fine-Grained, Scalable Web Data Model

We devised a fine-grained, scalable data model, which was inspired by the object decomposition and construction found in the latest multimedia standards, in particular progressive and hierarchical JPEG, JPEG 2000, MPEG-4, and H.264. By employing the proposed data model, various representations of a multimedia object can be generated in a fast and simple manner. The rationale behind the fast and simple processes is the absence of complex computations, which are usually involved in the multimedia adaptation.

Based on the scalable data model, a novel content adaptation was specified. Modulation, the new content adaptation, is the process to obtain an object's representation by means of adjusting (dropping and/or adding) the building blocks of the object. Modulation is characterized as a fast, reversible, but exclusive adaptation process. As mentioned earlier, the fast process is attributed to the absence of complex computations.

Modulation is reversible because, from an object's representation, not only can a lower fidelity representation be constructed, but a higher fidelity representation can be, as well. Finally, modulation is exclusive because it always results in the same data-format as the original object.

9.1.2 Modulation in JPEG 2000

Modulation was further implemented using the JPEG 2000 standard as an illustration. Modulation in JPEG 2000 was done in three types of scalability: quality, resolution, and component scalability. Three JPEG 2000 modulators were developed. The first modulator, `JP2Selector`, is used to generate representations and supplements of a JPEG 2000 image. `JP2Joiner`, the second JPEG 2000 modulator, combines a representation and a suitable supplement to produce an enhanced representation. The last modulator, `JP2Converter`, is used to change the progression order of a JPEG 2000 image (or its representation); this modulator cannot be applied to a supplement.

Further, we compared and contrasted modulation (the new content adaptation) and transcoding (the oft-cited content adaptation) through experiments. While JPEG 2000 was selected to illustrate modulation, JPEG was employed for transcoding. Two types of JPEG transcoders were experimented; those are spatial-domain (SDT) and frequency-domain (FDT) transcoders. Results show that modulation outperforms transcoding in all three adaptation aspects – namely quality, resolution, and component. Depending on the adaptation aspects, the processing times of modulation are 7–33 times faster than those of transcoding. More importantly, the processing times of modulation are linear and

consistent in those three adaptation aspects; that corroborates the simplicity of modulating processes. In addition, the experiments have demonstrated that modulation is reversible.

9.1.3 Framework for Pervasive Web Content Delivery

We evaluated two adapting approaches, i.e., server-based and proxy-based adaptation, using an analytical model. Proxy-based adaptation offers technical and economical benefits whereas server-based adaptation gives more control over the Web contents and is better in preserving their end-to-end semantics. The analytical model shows that, in the first-time delivery, the server-based adaptation is more beneficial than the proxy-based one. By contrast, in the subsequent delivery, the proxy-based adaptation is more beneficial than the server-based one.

Based on the findings, we concluded that the best way to deploy the adaptation service is to blend those two approaches and get most of the benefits. To put it simply, collaboration between the origin server and the proxy may yield the best adaptation approach. Accordingly, the system architecture of pervasive Web content delivery was founded on that collaboration. At a given time and condition, adaptation may be carried out at the server or at the proxy, whichever is more beneficial.

To perform content adaptation, the system architecture involves some adapting modules and supporting meta-data. The adapting modules comprise downscaling modules, upscaling modules, and transcoding modules (used only if necessary). The downscaling modules, used to construct an object's representation by reducing the building blocks of the object, are installed at the server and the proxy. The upscaling

modules, used to construct an enhanced object's representation, are installed at the proxy. And finally, the transcoding modules are employed by the proxy to convert the data-format, in case the client device cannot render the scalable data-format. The supporting meta-data may come from the server (e.g., directives regarding the content adaptation), the client (e.g., information about the client's constraints and preferences), or other intermediaries on the Internet (e.g., the network traffic's condition).

9.1.4 Model Prototype of Pervasive Web Content Delivery

We developed a model prototype based on the proposed framework for pervasive Web content delivery. For the client meta-data, CC/PP (Composite Capability/Preference Profiles) – a profile developed by W3C – is employed to describe device capabilities and user preferences. A client's CC/PP document can be stored in any Website (perhaps, of the client device's vendor) and should be referred to by every request belonging to the client in its `CCPP` header. For the server meta-data, we devised ADP (Adaptation Profiles) using RDF/XML to list representations of a Web object, including their characteristics and generation. An ADP document, which describes a particular Web object, has a similar resource name to the object but with the extra '.adp' extension. By separating the client and server meta-data from the HTTP messages, the proxy may cache them as different entities and reuse them for subsequent requests.

The server application (i.e., Apache) was enhanced to accommodate the image adaptors (JPEG transcoders and JPEG 2000 modulators). Some interfaces, developed using PHP scripting language, are employed to link the server application to the adaptors.

Some improvements were applied to the proxy application (i.e., Squid) so that adaptation can be performed at the proxy. There are four notable improvements on the proxy application: changes in the workflow, incorporation of external modules (such as CC/PP parser, ADP parser, and several adapting interfaces), addition of a rule-based decision maker, and adoption of API-liked adapting commands (*apcoms*).

The model prototype was then evaluated and compared with the existing Web content delivery. The first evaluation analyzed the costs and benefits of adaptation at the server. Adaptation at the server, as compared to that at the proxy alone, can reduce the client response time, in particular if the bandwidth of the server-proxy link is limited. Compared to the Web service without adaptation, adaptation at the server incurs some losses owing to the adaptation process. The cost of adaptation latency varies in different network environments and adaptation schemes. Overall, JPEG 2000 modulation (JP2-MOD) scheme has the lowest cost, followed by JPEG spatial-domain transcoding (JPG-SDT) and JPEG frequency-domain transcoding (JPG-FDT) schemes, in that order. Another cost of adaptation at the server is performance decrement signified by decreased request rate, prolonged average response time, and increased server's load. The request rate drops by 52.1% in JP2-MOD, 95.0% in JPG-SDT, and 99.9% in JPG-FDT. The average response time escalates by factors of 2.4, 26.6, and 1114.8 in JP2-MOD, JPG-SDT, and JPG-FDT, respectively. The server's load is determined by the percentage of time that the system spends in holding the maximum number of concurrent connections; the percentages in the respective JP2-MOD, JPG-SDT, and JPG-FDT are about 2, 2.4, and 3.6 times higher than that in the system without adaptation. It can be concluded that JP2-MOD is pretty fast, JPG-SDT barely acceptable, and JPG-FDT unacceptable.

The second evaluation analyzed the costs and benefits of an adaptation system comprising a server and a proxy. Cost-benefit analyses between a server/proxy system employing server-based adaptation and that employing server/proxy-based adaptation show that the latter is in general more beneficial than the former. However, it was assumed in the analyses that only one representation of an image could be cached at any given time. This condition is necessary to make the analyses possible and comparable. When the condition was lifted up, the performance of the system employing server-based adaptation proves to be better than that of the system employing server/proxy-based adaptation. Compared to the server system without a caching proxy, the server/proxy system employing server-based adaptation gains an increase of 9–41 times in the attained request rate and a decrease of 86–98% in the average response time. By contrast, the server/proxy system employing server/proxy-based adaptation only gains an increase of 2–26 times in the attained request rate and a decrease of 32–97% in the average response time. Nevertheless, the system employing server/proxy-based adaptation (particularly, in the JP2-MOD scheme) shows a marked reduction in the server's load. The reduction is attributed to the high data reuse offered by modulation.

We may conclude that the fine-grained, scalable data model indeed changes the paradigm of content adaptation in the Web. Adaptation can be accomplished in a fast and efficient manner, with minimal computations. Modulation, the novel content adaptation based on the data model, has proven to work well on a scalable data-format. The data model requires a new framework for Web content delivery. The framework, which is based on the collaboration between the origin server and the proxy, can effectively reduce the client response time and reduce the server's load by means of high data reuse.

9.2 Future Work

Even though this thesis has thoroughly demonstrated the benefits of our proposed data model and framework for pervasive Web content delivery, much work is still needed to improve them and to see them materializing into real practice. Some of the efforts that we may pursue in the future are elaborated in the following subsections.

9.2.1 Wide Implementation of Modulation

As noted in the thesis, modulation is an exclusive adaptation, which can be applied to a scalable data-format. Although this condition may limit the proliferation of modulation, we observe that the trend in the multimedia standards goes toward scalable presentation. Besides JPEG 2000, the latest multimedia standards such as MPEG-4 and H.264 also support scalable presentation. We believe that they can be exploited by modulation, as well. If Web objects of those multimedia standards are served to heterogeneous clients, modulation may be the fastest and most efficient way to accomplish it. Hence, implementation of modulation should be extended to them, too.

It is equally important to explore the possibilities to implement modulation on a textual document, particularly on a Web container. The challenge is to define a textual document's building blocks which can be dropped and added without sacrificing its semantics. A Web container – i.e., an HTML (or XML) document – usually contains several embedded objects. Each object's presentation may depend on the others', and together they should show an integral presentation. Modulation on the Web container will focus on the supposed dimensions of an embedded object relative to the container, the

priority of each object, the relation among objects, and others alike. Since special tags may be required, modulation on an HTML document may not be suitable, but that on an XML-based document is recommended. Modulation in other structured text formats – namely PDF, PS, RTF, etc. – may deal with removal of some sentences, phrases, or even images without changing the overall semantics of the contents.

9.2.2 Enhanced Adapting Proxy

As stated before, our current research does not put emphasis on completeness of the adapting proxy's features. Thus, there is still room for improvement in our work. One of them is manipulation of the cached objects which was done through the adoption of a set of adapting commands (*apcoms*), similar to an API library in other applications. We do not close the possibility to extend the set of adapting commands with more functions in future work. This is especially important if the enhanced proxy is to be used in real practice.

Another candidate for improvement is the proxy's rule-based decision maker. For now, only limited operations are supported in the matching rules. More operations, perhaps complex ones, may be added in the future.

9.2.3 Resource-Friendly Adaptor

In the experiments we found that our JPEG frequency-domain transcoder did not perform as expected. In fact, its performance was very disappointing, in particular when it was employed at the server to serve simultaneous requests. The frequency-domain

transcoding may require a less number of complex computations than what the spatial-domain transcoding needs, but it demands more memory space. The lesson makes us realize that a resource-friendly adaptor is necessary to support pervasive Web content delivery. More research is required to find out how a resource-friendly adaptor employing a particular adaptation technique (i.e., transcoding, modulation, and so forth) may be developed. The research may range from finding better frequency-domain processing to practicing data (and code) optimization.

9.2.4 High Data Reuse vs. Data Replication

The experimental results also highlight the need to balance high data reuse with data replication, in regard to caching an object. The former tries to greatly reuse the cached object by means of adaptation, and the latter stores all adapted representations of the object in the cache. As mentioned, to carry out adaptation at the proxy effectively and efficiently, there should not be many representations of the object in the proxy's cache; otherwise, there would be additional delay in finding and fetching the proper object's representation to be adapted. However, by restricting the number of object's representations in the cache, the system may lose some benefits resulting from pure cache hits (i.e., cache hits without adaptation). Future research should consider this trade-off to improve the system's performance.

Equally important is to see the benefits modulation can give to the cache replacement policies. Instead of evicting a cached object entirely, using modulation, we may reduce the object's fidelity and still keep it in the cache. The benefits will be great.

Bibliography

[AbSA95] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching Proxies: Limitations and Potentials. In *Proc. of 4th International World Wide Web (WWW) Conference*, Boston (MA), December 1995.

[AlLK02] M. Altinel, Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, B. G. Lindsay, H. Woo, and L. Brown. DBCache: Database Caching for Web Application Servers. In *Proc. of 2002 ACM SIGMOD International Conference on Management of Data*, Madison (WI), June 2002.

[AmPT02] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan. A Self-Managing Data Cache for Edge-of-Network Web Applications. In *Proc. of 11th International Conference on Information and Knowledge Management (CIKM)*, McLean (VA), November 2002.

[AmPT03] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan. DBProxy: A Dynamic Data Cache for Web Applications. In *Proc. of 19th International Conference on Data Engineering (ICDE)*, Bangalore (India), March 2003.

[BaBC04] A. Barbir, E. Burger, R. Chen, S. McHenry, H. Orman, and R. Penno. *Open Pluggable Edge Services (OPES) Use Cases and Deployment Scenarios, RFC 3752*, IETF, April 2004.

[BaPC04] A. Barbir, R. Penno, R. Chen, M. Hofmann, and H. Orman. *An Architecture for Open Pluggable Edge Services (OPES), RFC 3835*, IETF, August 2004.

[BeFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0, RFC 1945*, IETF, May 1996.

[BeHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web, *Scientific American*, May 2001.

[Ber91] T. Berners-Lee. *The HTTP Protocol As Implemented in W3*. 1991. URL: http://www.w3.org/Protocols/HTTP/AsImplemented.html.

[Ber92] T. Berners-Lee. *HTTP: A Protocol for Networked Information*. 1992. URL: http://www.w3.org/Protocols/HTTP/HTTP2.html.

[BeRV04] L. Bent, M. Rabinovich, G. M. Voelker, and Z. Xiao. Characterization of a Large Web Site Population with Implications for Content Delivery. In *Proc. of 13th International World Wide Web (WWW) Conference*, New York (NY), May 2004.

[BhJA98] H. Bharadvaj, A. Joshi, and S. Auephanwiriyakul. An Active Transcoding Proxy to Support Mobile Web Access. In *Proc. of 17th IEEE Symposium on Reliable Distributed Systems*, West Lafayette (IN), October 1998.

[BoAM04] C. Bornhovd, M. Altinel, C. Mohan, H. Pirahesh, and B. Reinwald. Adaptive Database Caching with DBCache. *Data Engineering, vol. 27, no. 2*, pp. 11–18, June 2004.

[CaC96] R. L. Carter and M. E. Crovella. Measuring Bottleneck Link Speed in Packet-Switched Networks. *Technical Report BU-CS-96-006, Boston University*, Boston (MA), 1996.

[CaCC03] C. Canali, V. Cardellini, M. Colajanni, R. Lancellotti, and P. S. Yu. Cooperative Architectures and Algorithms for Discovery and Transcoding

of Multi-Version Content. In *Proc. of 8th International Workshop on Web Content Caching and Distribution (WCW)*, Hawthorne (NY), September 2003.

[CaCC04] C. Canali, V. Cardellini, M. Colajanni, and R. Lancellotti. Evaluating User-Perceived Benefits of Content Distribution Networks. In *Proc. of 2004 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, San Jose (CA), July 2004.

[CaCL05] C. Canali, S. Casolari, and R. Lancellotti. Architectures for Scalable and Flexible Web Personalization Services. In *Proc. of 1st International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA)*, Orlando (FL), June 2005.

[CaCL06a] C. Canali, V. Cardellini, and R. Lancellotti. Content Adaptation Architectures Based on Squid Proxy Server. *World Wide Web, vol. 9, no. 1*, pp. 63–92, March 2006.

[CaCL06b] C. Canali, M. Colajanni, and R. Lancellotti. Distribution of Adaptation Services for Ubiquitous Web Access Driven by User Profiles. In *Proc. of 11th IEEE Symposium on Computers and Communications (ISCC)*, Cagliari – Sardinia (Italy), June 2006.

[CaYH00] V. Cardellini, P. S. Yu, Y. W. Huang. Collaborative Proxy System for Distributed Web Content Transcoding. In *Proc. of 9th International Conference on Information and Knowledge Management (CIKM)*, McLean (VA), November 2000.

[ChC02] C. H. Chi and Y. Cao. Pervasive Web Content Delivery with Efficient Data Reuse. In *Proc. of 7th International Workshop on Web Content Caching and Distribution (WCW)*, Boulder (CO), August 2002.

[ChC03] C. Y. Chang and M. S. Chen. On Exploring Aggregate Effect for Efficient Cache Replacement in Transcoding Proxies. *IEEE Transactions on Parallel and Distributed Systems, vol. 14, no. 6*, pp. 611–624, June 2003.

[ChE99] S. Chandra and C. S. Ellis. JPEG Compression Metric as a Quality Aware Image Transcoding. In *Proc. of 2nd USENIX Symposium on Internet Technologies and Systems (USITS)*, Boulder (CO), October 1999.

[ChEV00] S. Chandra, C. S. Ellis, and A. Vahdat. Differentiated Multimedia Web Services Using Quality Aware Transcoding. In *Proc. of IEEE INFOCOM 2000 – 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, Tel Aviv (Israel), March 2000.

[ChSE00] C. Christopoulos, A. Skodras, and T. Ebrahimi. The JPEG2000 Still Image Coding System: An Overview. *IEEE Transactions on Consumer Electronics, vol. 46, no. 4*, pp. 1103–1127, November 2000.

[CoL04] M. Colajanni and R. Lancellotti. System Architectures for Web Content Adaptation Services. *IEEE Distributed Systems Online, Web Systems Invited Article*, June 2004.

[CoLY05] M. Colajanni, R. Lancellotti, and P. S. Yu. Distributed Architectures for Web Content Adaptation and Delivery. *Web Content Delivery* (X. Tang, J. Xu, and S. T. Chanson – eds.), Chapter 13, pp. 285–304, Springer, 2005.

[CoMT01] I. Cooper, I. Melve, and G. Tomlinson. *Internet Web Replication and Caching Taxonomy, RFC 3040*, IETF, January 2001.

[DaCK04] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proc. of ACM SIGCOMM 2004 Conference*, Portland (OR), August 2004.

[DaCT03] M. Day, B. Cain, G. Tomlinson, and P. Rzewski. *A Model for Content Internetworking (CDI), RFC 3466*, IETF, February 2003.

[DaPW04] A. Davis, J. Parikh, and W. E. Weihl. EdgeComputing: Extending Enterprise Applications to the Edge of the Internet. In *Proc. of 13th International World Wide Web (WWW) Conference*, New York (NY), May 2004.

[Den05] P. J. Denning. The Locality Principle. *Communications of the ACM, vol. 48, no. 7*, pp. 19–24, July 2005.

[DeZ01] S. Deshpande and W. Zeng. Scalable Streaming of JPEG2000 Images using Hypertext Transfer Protocol. In *Proc. of 9th ACM Multimedia Conference*, Ottawa (Canada), October 2001.

[DoHR97] F. Douglis, A. Haro, and M. Rabinovich. HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching. In *Proc. of 1st USENIX Symposium on Internet Technologies and Systems (USITS)*, Monterey (CA), December 1997.

[DoRM01] C. Dovrolis, P. Ramanathan, and D. Moore. What do Packet Dispersion Techniques Measure? In *Proc. of IEEE INFOCOM 2001 – 20th Annual*

Joint Conference of the IEEE Computer and Communications Societies, Anchorage (AK), April 2001.

[Dow99] A. B. Downey. Using Pathchar to Estimate Internet Link Characteristics. In *Proc. of ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'99)*, Cambridge (MA), August 1999.

[ElC03] J. Elson and A. Cerpa. *Internet Content Adaptation Protocol (ICAP), RFC 3507*, IETF, April 2003.

[ESI01] *ESI – Accelerating E-Business Applications: Overview*, 2001. URL: http://www.esi.org/overview.html.

[FeCD99] A. Feldmann, R. Cáceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments. In *Proc. of IEEE INFOCOM 1999 Conference*, New York (NY), March 1999.

[FiGM99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1, RFC 2616*, IETF, June 1999.

[FoB96] A. Fox and E. A. Brewer. Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation. In *Proc. of 5th International World Wide Web (WWW) Conference*, Paris (France), May 1996.

[FoGB96] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proc. of 7th International Conference on Architectural Support for Programming*

Languages and Operationg Systems (ASPLOS), Cambridge (MA), October 1996.

[FoGC97] A. Fox, S. D. Gribble. Y. Chawathe, A. S. Polito, A. Huang, B. Ling, and E. A. Brewer. Orthogonal Extensions to the WWW User Interface Using Client-Side Technologies. In *Proc. of 10th Annual ACM Symposium on User Interface Software and Technology (UIST)*, Banff – Alberta (Canada), October 1997.

[FoGC98] A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer. Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives. *IEEE Personal Communications, vol. 5, no. 4*, pp. 10–19, August 1998.

[FrJJ01] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A Global Internet Host Distance Estimation Service. *IEEE/ACM Transactions on Networking, vol. 9, no. 5*, pp. 525–540, October 2001.

[GuSG02] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *Proc. of ACM SIGCOMM Internet Measurement Workshop (IMW) 2002*, Marseille (France), November 2002.

[HaBL98] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas. Dynamic Adaptation in an Image Transcoding Proxy for Mobile Web Browsing, *IEEE Personal Communications, vol. 5, no. 6*, pp. 8–17, December 1998.

[ICAP01] *ICAP White Paper v1.01*, July 2001.

URL: http://www.i-cap.org/docs/icap_whitepaper_v1-01.pdf.

[IhMM01] S. C. Ihde, P. P. Maglio, J. Meyer, and R. Barrett. Intermedia-based Transcoding Framework. *IBM Systems Journal, vol. 40, no. 1*, 2001.

[ISO15444-1] International Organization for Standardization. *Information Technology – JPEG 2000 Image Coding System – Part 1: Core Coding System, ISO/IEC 15444-1:2004 (E).*

[ISO15444-5] International Organization for Standardization. *Information Technology – JPEG 2000 Image Coding System – Part 5: Reference Software, ISO/IEC 15444-5:2003 (E).*

[ITU92] International Telecommunication Union. *Information Technology – Digital Compression and Coding of Continuous-Tone Still Images – Requirements and Guidelines (ITU-T T.81)*, September 1992.

[Jac97] V. Jacobson. Pathchar – A Tool to Infer Characteristics of Internet Paths. Presented at the Mathematical Sciences Research Institute (MSRI); slides available from ftp://ftp.ee.lbl.gov/pathchar/msri-talk.pdf, Berkeley (CA), April 1997.

[JaD02] M. Jain and C. Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *Proc. of ACM SIGCOMM 2002 Conference*, Pittsburgh (PA), August 2002.

[JoWM96] A. Joshi, R. Weerasinghe, S. P. McDermott, B. K. Tan, G. Bernhardt, and S. Weerawarana. Mowser: Mobile Platforms and Web Browsers. *Bulletin of the IEEE Computer Society Technical Committee on Operating Systems*

and Application Environments (TCOS), vol. 8, no. 1, pp. 13–16, Spring 1996.

[KaKO98] J. Kangasharju, Y. G. Kwon, and A. Ortega. Design and Implementation of a Soft Caching Proxy. In *Proc. of 3rd International WWW Caching Workshop*, Machester (England), June 1998.

[Kly99a] G. Klyne. *A Syntax for Describing Media Feature Sets, RFC 2533*, IETF, March 1999.

[Kly99b] G. Klyne. *Corrections to "A Syntax for Describing Media Feature Sets", RFC 2738*, IETF, December 1999.

[KnLM03] B. Knutsson, H. Lu, J. Mogul, and B. Hopkins. Architecture and Performance of Server-Directed Transcoding. *ACM Transactions on Internet Technology (TOIT), vol. 3, no. 4*, pp. 392–424, November 2003.

[KrLM97] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. In *Proc. of 1st USENIX Symposium on Internet Tecnologies and Systems (USITS)*, Monterey (CA), December 1997.

[KrMK99] B. Krishnamurthy, J. C. Mogul, and D. M. Kristol. Key Differences between HTTP/1.0 and HTTP/1.1. In *Proc. of 8th International World Wide Web (WWW) Conference*, Toronto (Canada), May 1999.

[KrR01] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison-Wesley, 2001.

[KrWZ01] B. Krishnamurthy, C. Willis, and Y. Zhang. On the Use and Performance of Content Distribution Networks. In *Proc. of ACM SIGCOMM Internet Measurement Workshop (IMW) 2001*, San Francisco (CA), November 2001.

[LaB01] K. Lai and M. Baker. Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. In *Proc. of 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, San Francisco (CA), March 2001.

[Li01] W. Li. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. *IEEE Transactions on Circuits and Systems for Video Technology, vol. 11, no. 3*, pp. 301–317, March 2001.

[Lia99] J. Liang. New Trends in Multimedia Standards: MPEG4 and JPEG2000. *Informing Science: Special Issue on Multimedia Informing Technologies – Part I, vol. 2, no. 4*, pp. 101–106, 1999.

[LiS03] J. Li and H. H. Sun. On Interactive Browsing of Large Images. *IEEE Transactions on Multimedia, vol. 5, no. 4*, pp. 581–590, December 2003.

[LuA94] A. Luotonen and K. Altis. World-Wide Web Proxies. In *Proc. of 1st International World Wide Web (WWW) Conference*, Geneva (Switzerland), May 1994.

[Mah99] B. A. Mah. Pchar: Child of Pathchar. Presented at the Department of Energy, Next Generation Internet Program (DOE NGI) Testbed Workshop; slides available from http://www.kitchenlab.org/www/bmah/Talks/pchar-NGI-99-Slides.pdf, Berkeley (CA), July 1999.

[MaSR02] A. Maheshwari, A. Sharma, K. Ramamritham, and P. Shenoy. TranSquid: Transcoding and Caching Proxy for Heterogeneous E-Commerce Environments. In *Proc. of 12th International Workshop on Research Issues in Data Engineering (RIDE): Engineering e-Commerce / e-Business Systems*, San Jose (CA), February 2002.

[McC00] S. McCreary and K. C. Claffy. Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange. In *Proc. of 13th ITC Specialist Seminar on Measurement and Modeling of IP Traffic*, Monterey (CA), September 2000.

[MoCC06] I. Mohomed, J. C. Cai, S. Chavoshi, and E de Lara. Context-Aware Interactive Content Adaptation. In *Proc. of 4th International Conference on Mobile Systems, Applications, and Services (Mobisys)*, Uppsala (Sweden), June 2006.

[MoDF97] J. C. Mogul, F. Douglis, A. Feldmann, and B. Kryshnamurthy. Potential Benefits of Delta Encoding and Data Compression for HTTP. In *Proc. of ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'97)*, Cannes (France), September 1997.

[Mog01] J. C. Mogul. Server-Directed Transcoding. *Computer Communications, vol. 24, no. 2*, pp. 155–162, February 2001.

[MoKD02] J. Mogul, B. Krishnamurthy, F. Douglis, A. Feldmann, Y. Goland, A. van Hoff, and D. Hellerstein. *Delta Encoding in HTTP, RFC 3229*, IETF, January 2002.

[MoSL99] R. Mohan, J. R. Smith, and C. S. Li. Adapting Multimedia Internet Content for Universal Access. *IEEE Transactions on Multimedia, vol. 1, no. 1*, pp. 104–114, March 1999.

[NaV95] B. K. Natarajan and B. Vasudev. A Fast Approximate Algorithm for Scaling Down Digital Images in the DCT Domain. In *Proc. of IEEE International Conference on Image Processing (ICIP) 1995, vol. 2*, pp. 241–243, Washington D.C., October 1995.

[NgZ01] T. S. E. Ng and H. Zhang. Towards Global Network Positioning. In *Proc. of ACM SIGCOMM Internet Measurement Workshop (IMW) 2001*, San Fransisco (CA), November 2001.

[NgZ04] T. S. E. Ng and H. Zhang. A Network Positioning System for the Internet. In *Proc. of USENIX 2004 Annual Technical Conference*, Boston (MA), June 2004.

[NiGB97] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley. Network Performance Effects of HTTP/1.1, CSS1, and PNG. In *Proc. of ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'97)*, Cannes (France), September 1997.

[OlMG05] C. Olston, A. Manjhi, C. Garrod, A. Ailamaki, B. M. Maggs, and T. C. Mowry. A Scalability Service for Dynamic Web Applications. In *Proc. of 2nd Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar (CA), January 2005.

[OrCA97] A. Ortega, F. Carignano, S. Ayer, and M. Vetterli. Soft Caching: Web Cache Management Technigues for Images. In *Proc. of 1st IEEE Signal Processing Society Workshop on Multimedia Signal Processing (MMSP)*, Princeton (NJ), June 1997.

[PeA97] F. Pereira and T. Alpert. MPEG-4 Video Subjective Test Procedures and Results. *IEEE Transactions on Circuits and Systems for Video Technology, vol. 7, no. 1*, pp. 32–51, February 1997.

[PeE02] F. Pereira and T. Ebrahimi (eds.). *The MPEG-4 Book*. Prentice Hall PTR, 2002.

[PlA04] C. Plattner and G. Alonso. Ganymed: Scalable Replication for Transactional Web Applications. In *Proc. of 5th International Middleware Conference*, Toronto – Ontario (Canada), October 2004.

[RaS02] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison-Wesley, 2002.

[RaXA03] M. Rabinovich, Z. Xiao, and A. Aggarwal. Computing on the Edge: A Platform for Replicating Internet Applications. In *Proc. of 8th International Workshop on Web Content Caching and Distribution (WCW)*, Hawthorne (NY), September 2003.

[RaXD03] M. Rabinovich, Z. Xiao, F. Douglis, and C. Kalmanek. Moving Edge-Side Includes to the Real Edge – the Clients. In *Proc. of 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, Seattle (WA), March 2003.

[ReYH00] R. Rejaie, H. Yu, M. Handley, and D. Estrin. Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications in the Internet. In *Proc. of IEEE INFOCOM 2000 – 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, Tel Aviv (Israel), March 2000.

[SaGG02] S. Saroiu, P. K. Gummadi, and S. D. Gribble. SProbe: A Fast Technique for Measuring Bottleneck Bandwidth in Uncooperative Environments. URL: http://sprobe.cs.washington.edu/sprobe.ps.

[ShLB04] B. Shen, S. J. Lee, and S. Basu. Caching Strategies in Transcoding-Enabled Proxy Systems for Streaming Media Distribution Networks. *IEEE Transactions on Multimedia, vol. 6, no. 2*, pp. 375–386, April 2004.

[Sik97] T. Sikora. The MPEG-4 Video Standard Verification Model. *IEEE Transactions on Circuits and Systems for Video Technology, vol. 7, no. 1*, pp. 19–31, February 1997.

[SiPS06] S. Sivasubramanian, G. Pierre, M. v. Steen, and G. Alonso. GlobeCBC: Content-Blind Result Caching for Dynamic Web Applications. *Technical Report IR-CS-022, Vrije Universiteit*, Amsterdam (The Netherlands), June 2006.

[SiTR04] A. Singh, A. Trivedi, K. Ramamritham, and P. Shenoy. PTC: Proxies that Transcode and Cache in Heterogeneous Web Client Environments. *World Wide Web, vol. 7, no. 1*, pp. 7–28, March 2004.

[SmCL99] J. R. Smith, V. Castelli, and C. S. Li. Adaptive Storage and Retrieval of Large Compressed Images. In *Proc. of IS&T/SPIE Symposium on*

Electronic Imaging: Science and Technology – Storage & Retrieval for Image and Video Databases VII, San Jose (CA), January 1999.

[SmML98] J. R. Smith, R. Mohan, and C. S. Li. Transcoding Internet Content for Heterogeneous Client Devices. In *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)*, Monterey (CA), May/June 1998.

[TaM02] D. S. Taubman and M. W. Marcellin. *JPEG2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publisher, 2002.

[TaP03] D. Taubman and R. Prandolini. Architecture, Philosophy and Performance of JPIP: Internet Protocol Standard for JPEG2000. *Visual Communications and Image Processing 2003 (VCIP2003), Proc. of SPIE vol. 5150*, pp. 791–805, Lugano (Switzerland), June 2003.

[TeSS97] D. L. Tennenhouse, J. M. Smith, W. D. Siscoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine, vol. 35, no. 1*, pp. 80–86, January 1997.

[ThMW97] K. Thompson, G. J. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. *IEEE Network, vol. 11, no. 6*, pp. 10–23, November 1997.

[Tho01] S. A. Thomas. *HTTP Essentials: Protocols for Secure, Scaleable Web Sites*. John Wiley & Sons, 2001.

[Ver02] D. C. Verma. *Content Distribution Networks: An Engineering Approach*. John Wiley & Sons, 2002.

[WiMQ05] M. Wijnants, P. Monsieurs, P. Quax, and W. Lamotte. Exploiting Proxy-Based Transcoding to Increase the User Quality of Experience in Networked Applications. In *Proc. of 1st International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA)*, Orlando (FL), June 2005.

[WoSH99] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems, vol. 15, no. 5–6*, pp. 757–768, October 1999.

[WuLZ01] F. Wu, S. Li, and Y. Q. Zhang. A Framework for Efficient Progressive Fine Granularity Scalable Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology, vol. 11, no. 3*, pp. 332–344, March 2001.

Appendix A

ADP Schema
[http://rdfs.example.org/adp-schema]

```
<?xml version='1.0'?>

<!DOCTYPE rdf:RDF [
  <!ENTITY ns-rdf  'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <!ENTITY ns-rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
  <!ENTITY ns-xsd  'http://www.w3.org/2001/XMLSchema#'>
  <!ENTITY ns-adp  'http://rdfs.example.org/adp-schema#'>
]>

<rdf:RDF
  xmlns:rdf  =' &ns-rdf;'
  xmlns:rdfs =' &ns-rdfs;'
  xmlns:adp  =' &ns-adp;'>

<!-- ADP (Adaptation Profiles) class definitions -->
  <rdfs:Class rdf:about='&ns-adp;Profile'>
    <rdfs:label xml:lang="en">ADP Profile</rdfs:label>
    <rdfs:subClassOf rdf:resource='&ns-rdfs;Resource'/>
    <rdfs:comment xml:lang="en">
      This class is the super-class of all object profiles.
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about='&ns-adp;ImageProfile'>
    <rdfs:label xml:lang="en">ADP Image Profile</rdfs:label>
    <rdfs:subClassOf rdf:resource='&ns-adp;Profile'/>
    <rdfs:comment xml:lang="en">
      This class is to specify the profile of an image object.
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about='&ns-adp;VideoProfile'>
    <rdfs:label xml:lang="en">ADP Video Profile</rdfs:label>
    <rdfs:subClassOf rdf:resource='&ns-adp;Profile'/>
    <rdfs:comment xml:lang="en">
      This class is to specify the profile of a video object.
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about='&ns-adp;AudioProfile'>
    <rdfs:label xml:lang="en">ADP Audio Profile</rdfs:label>
```

```
    <rdfs:subClassOf rdf:resource='&ns-adp;Profile'/>
    <rdfs:comment xml:lang="en">
      This class is to specify the profile of an audio object.
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about='&ns-adp;TextProfile'>
    <rdfs:label xml:lang="en">ADP Text Profile</rdfs:label>
    <rdfs:subClassOf rdf:resource='&ns-adp;Profile'/>
    <rdfs:comment xml:lang="en">
      This class is to specify the profile of a textual document.
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about='&ns-adp;Adaptor'>
    <rdfs:label xml:lang="en">ADP Adapting Module</rdfs:label>
    <rdfs:subClassOf rdf:resource='&ns-adp;Resource'/>
    <rdfs:comment xml:lang="en">
      This class is to specify the adapting module.
    </rdfs:comment>
  </rdfs:Class>

<!-- ADP Structure and Attribute class definitions -->

  <rdfs:Class rdf:about='&ns-adp;Structure'>
    <rdfs:label xml:lang="en">ADP Structural Property</rdfs:label>
    <rdfs:subClassOf rdf:resource='&ns-rdf;Property'/>
    <rdfs:comment xml:lang="en">
      All properties that are structural elements of an ADP profile
      are defined as instances of adp:Structure.
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about='&ns-adp;Attribute'>
    <rdfs:label xml:lang="en">ADP Attribute Property</rdfs:label>
    <rdfs:subClassOf rdf:resource='&ns-rdf;Property'/>
    <rdfs:comment xml:lang="en">
      All object properties should be defined as instances of
      adp:Attribute.
    </rdfs:comment>
  </rdfs:Class>

<!-- ADP structural property definitions -->

  <adp:Structure rdf:about='&ns-adp;representation'>
    <rdfs:label xml:lang="en">ADP representation property</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;Profile'/>
    <rdfs:range  rdf:resource='&ns-rdfs;List'/>
    <rdfs:comment xml:lang="en">
      A property listing the representations of an object.
    </rdfs:comment>
  </adp:Structure>
```

```
  <adp:Structure rdf:about='&ns-adp;adaptedBy'>
    <rdfs:label xml:lang="en">ADP adaptor property</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;Profile'/>
    <rdfs:range  rdf:resource='&ns-adp;Adaptor'/>
    <rdfs:comment xml:lang="en">
      A property indicating the module used for the adaptation.
    </rdfs:comment>
  </adp:Structure>

<!-- ADP attribute property definitions -->

  <!-- Profile Attributes -->

  <adp:Attribute rdf:about='&ns-adp;contentType'>
    <rdfs:label xml:lang="en">Content-type</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;Profile'/>
    <rdfs:range  rdf:resource='&ns-xsd;string'/>
    <rdfs:comment xml:lang="en">
      A string describing the content-type of an object.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-adp;dataSize'>
    <rdfs:label xml:lang="en">Data-size</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;Profile'/>
    <rdfs:range  rdf:resource='&ns-xsd;integer'/>
    <rdfs:comment xml:lang="en">
      An integer describing the data-size of an object.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-adp;presentationWidth'>
    <rdfs:label xml:lang="en">Presentation's Width</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;Profile'/>
    <rdfs:range  rdf:resource='&ns-xsd;integer'/>
    <rdfs:comment xml:lang="en">
      An integer describing the minimum width (in pixels) required
      to display an object.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-adp;presentationHeight'>
    <rdfs:label xml:lang="en">Presentation's Height</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;Profile'/>
    <rdfs:range  rdf:resource='&ns-xsd;integer'/>
    <rdfs:comment xml:lang="en">
      An integer describing the minimum height (in pixels) required
      to display an object.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-adp;adaptingParams'>
    <rdfs:label xml:lang="en">Adapting Parameters</rdfs:label>
```

```
    <rdfs:domain rdf:resource='&ns-adp;Profile'/>
    <rdfs:range  rdf:resource='&ns-rdf;Bag'/>
    <rdfs:comment xml:lang="en">
      A list of parameters used to adapt an object.
    </rdfs:comment>
  </adp:Attribute>

  <!-- Adaptor Attributes -->

  <adp:Attribute rdf:about='&ns-adp;runEnvironment'>
    <rdfs:label xml:lang="en">Running Environment</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;Adaptor'/>
    <rdfs:range  rdf:resource='&ns-xsd;string'/>
    <rdfs:comment xml:lang="en">
      A string describing the running environment of an adaptor,
      e.g. linux-executable, windows-executable, java, javascript,
      VBScript, etc.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-adp;runEnvVersion'>
    <rdfs:label xml:lang="en">Running Env.'s Version</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;Adaptor'/>
    <rdfs:range  rdf:resource='&ns-xsd;string'/>
    <rdfs:comment xml:lang="en">
      A string describing the version of the running environment
      mentioned in attribute adp:runEnvironment, e.g. if
      adp:runEnvironment='java', perhaps adp:runEnvVersion='1.4.2'.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-adp;additionalLibs'>
    <rdfs:label xml:lang="en">Additional Libraries</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;Adaptor'/>
    <rdfs:range  rdf:resource='&ns-rdf;Bag'/>
    <rdfs:comment xml:lang="en">
      A list of libraries that may be needed by an adaptor.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-adp;reversible'>
    <rdfs:label xml:lang="en">Reversible Adaptation</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;Adaptor'/>
    <rdfs:range  rdf:resource='&ns-xsd;boolean'/>
    <rdfs:comment xml:lang="en">
      A boolean value denoting whether an adaptation is reversible.
    </rdfs:comment>
  </adp:Attribute>

</rdf:RDF>
```

Appendix B

Experimental Server Meta-Data

ADP Schema Extension [http://svr.my-dom.org/ADP/adp-svr-profile]

```
<?xml version='1.0'?>

<!DOCTYPE rdf:RDF [
  <!ENTITY ns-rdf   'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <!ENTITY ns-rdfs  'http://www.w3.org/2000/01/rdf-schema#'>
  <!ENTITY ns-xsd   'http://www.w3.org/2001/XMLSchema#'>
  <!ENTITY ns-adp   'http://rdfs.example.org/adp-schema#'>
  <!ENTITY ns-sprof 'http://svr.my-dom.org/ADP/adp-svr-profile#'>
]>

<rdf:RDF
  xmlns:rdf  = '&ns-rdf;'
  xmlns:rdfs = '&ns-rdfs;'
  xmlns:adp  = '&ns-adp;'>

<!-- ADP Image Profile Attributes -->

  <adp:Attribute rdf:about='&ns-sprof;grayscale'>
    <rdfs:label xml:lang="en">Grayscale</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;ImageProfile'/>
    <rdfs:range  rdf:resource='&ns-xsd;boolean'/>
    <rdfs:comment xml:lang="en">
      A boolean describing whether the image is grayscale.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-sprof;jpgScaledownFactor'>
    <rdfs:label xml:lang="en">SD Factor of JPEG image</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;ImageProfile'/>
    <rdfs:range  rdf:resource='&ns-xsd;integer'/>
    <rdfs:comment xml:lang="en">
      An integer describing the scaled-down (SD) factor used to
      retrieve the JPEG image presentation from the original.
      Note: the factor is in 1/N
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-sprof;jpgQuality'>
    <rdfs:label xml:lang="en">Quality of JPEG image</rdfs:label>
```

```
    <rdfs:domain rdf:resource='&ns-adp;ImageProfile'/>
    <rdfs:range  rdf:resource='&ns-xsd;integer'/>
    <rdfs:comment xml:lang="en">
      An integer describing the JPEG image's quality.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-sprof;jp2ProgressionOrder'>
    <rdfs:label xml:lang="en">PO of J2K image</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;ImageProfile'/>
    <rdfs:range  rdf:resource='&ns-xsd;string'/>
    <rdfs:comment xml:lang="en">
      A string describing the progression order (PO) in a JPEG 2000
      image, e.g. lrcp, rlcp, rpcl, pcrl, cprl.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-sprof;jp2NoComponents'>
    <rdfs:label xml:lang="en">No of Cmps of J2K image</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;ImageProfile'/>
    <rdfs:range  rdf:resource='&ns-xsd;integer'/>
    <rdfs:comment xml:lang="en">
      An integer describing the number of color components in
      a JPEG 2000 image.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-sprof;jp2NoResolutions'>
    <rdfs:label xml:lang="en">No of Lvls of J2K image</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;ImageProfile'/>
    <rdfs:range  rdf:resource='&ns-xsd;integer'/>
    <rdfs:comment xml:lang="en">
      An integer describing the number of decomposition levels
      (resolutions) in a JPEG 2000 image.
    </rdfs:comment>
  </adp:Attribute>

  <adp:Attribute rdf:about='&ns-sprof;jp2NoLayers'>
    <rdfs:label xml:lang="en">No of Lyrs of J2K image</rdfs:label>
    <rdfs:domain rdf:resource='&ns-adp;ImageProfile'/>
    <rdfs:range  rdf:resource='&ns-xsd;integer'/>
    <rdfs:comment xml:lang="en">
      An integer describing the number of quality layers in
      a JPEG 2000 image.
    </rdfs:comment>
  </adp:Attribute>

</rdf:RDF>
```

Profile of boat.jpeg (for SDT) [http://svr.my-dom.org/images/boat.jpeg.adp]

```
<?xml version='1.0'?>

<rdf:RDF
  xmlns:rdf  = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:rdfs = 'http://www.w3.org/2000/01/rdf-schema#'
  xmlns:adp  = 'http://rdfs.example.org/adp-schema#'
  xmlns      = 'http://svr.my-dom.org/ADP/adp-svr-profile#'
  xml:base   = 'http://svr.my-dom.org/images/boat.jpeg'>

  <!-- Quality Indicator in rdf:ID
    0 - original
    1 - very high
    2 - high
    3 - average
    4 - low
    5 - very low
  -->

  <adp:ImageProfile
   rdf:about='http://svr.my-dom.org/images/boat.jpeg'>
    <adp:contentType>image/jpeg</adp:contentType>
    <adp:dataSize>5065493</adp:dataSize>
    <adp:presentationWidth>1976</adp:presentationWidth>
    <adp:presentationHeight>2960</adp:presentationHeight>
    <grayscale>false</grayscale>
    <jpgScaledownFactor>1</jpgScaledownFactor>
    <jpgQuality>100</jpgQuality>
    <adp:adaptedBy
     rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
    <adp:representation rdf:parseType='Collection'>
    <!-- Reduced-Resolution Representation(s) -->
      <adp:ImageProfile rdf:ID='p03-half_resl'>
        <adp:dataSize>1340018</adp:dataSize>
        <adp:presentationWidth>988</adp:presentationWidth>
        <adp:presentationHeight>1480</adp:presentationHeight>
        <jpgScaledownFactor>2</jpgScaledownFactor>
        <adp:adaptedBy
         rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
        <adp:representation rdf:parseType='Collection'>
        <!-- Reduced-Quality Representation(s) -->
          <adp:ImageProfile rdf:ID='p00-def-half_resl-3'>
            <adp:dataSize>157673</adp:dataSize>
            <jpgQuality>47</jpgQuality>
```

```
     <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
    </adp:ImageProfile>
  <!-- Reduced-Component Representation(s) -->
    <adp:ImageProfile rdf:ID='p00-half_resl-gray-3'>
      <adp:dataSize>131034</adp:dataSize>
      <grayscale>true</grayscale>
      <jpgQuality>41</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
    </adp:ImageProfile>
  </adp:representation>
</adp:ImageProfile> <!-- END OF #p03-half_resl -->

<adp:ImageProfile rdf:ID='p01-quarter_resl'>
  <adp:dataSize>362128</adp:dataSize>
  <adp:presentationWidth>494</adp:presentationWidth>
  <adp:presentationHeight>740</adp:presentationHeight>
  <jpgScaledownFactor>4</jpgScaledownFactor>
  <adp:adaptedBy
   rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>

  <adp:representation rdf:parseType='Collection'>

  <!-- Reduced-Quality Representation(s) -->
    <adp:ImageProfile rdf:ID='p01-quarter_resl-2'>
      <adp:dataSize>205983</adp:dataSize>
      <jpgQuality>96</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p01-def-quarter_resl-3'>
      <adp:dataSize>114579</adp:dataSize>
      <jpgQuality>88</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p01-quarter_resl-4'>
      <adp:dataSize>64401</adp:dataSize>
      <jpgQuality>67</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p01-quarter_resl-5'>
      <adp:dataSize>38096</adp:dataSize>
      <jpgQuality>30</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
    </adp:ImageProfile>
  <!-- Reduced-Component Representation(s) -->
    <adp:ImageProfile rdf:ID='p01-quarter_resl-gray'>
```

```
          <adp:dataSize>268245</adp:dataSize>
          <grayscale>true</grayscale>
          <adp:adaptedBy
           rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
        </adp:ImageProfile>
        <adp:ImageProfile rdf:ID='p01-def-quarter_resl-gray-3'>
          <adp:dataSize>86810</adp:dataSize>
          <grayscale>true</grayscale>
          <jpgQuality>84</jpgQuality>
          <adp:adaptedBy
           rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
        </adp:ImageProfile>
      </adp:representation>
    </adp:ImageProfile> <!-- END OF #p01-quarter_resl -->

    <adp:ImageProfile rdf:ID='p02-eighth_resl'>
      <adp:dataSize>98052</adp:dataSize>
      <adp:presentationWidth>247</adp:presentationWidth>
      <adp:presentationHeight>370</adp:presentationHeight>
      <jpgScaledownFactor>8</jpgScaledownFactor>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>

      <adp:representation rdf:parseType='Collection'>

      <!-- Reduced-Quality Representation(s) -->
        <adp:ImageProfile rdf:ID='p02-def-eighth_resl-3'>
          <adp:dataSize>57329</adp:dataSize>
          <jpgQuality>96</jpgQuality>
          <adp:adaptedBy
           rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
        </adp:ImageProfile>
        <adp:ImageProfile rdf:ID='p02-eighth_resl-4'>
          <adp:dataSize>23568</adp:dataSize>
          <jpgQuality>77</jpgQuality>
          <adp:adaptedBy
           rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
        </adp:ImageProfile>
      <!-- Reduced-Component Representation(s) -->
        <adp:ImageProfile rdf:ID='p02-eighth_resl-gray-3'>
          <adp:dataSize>37473</adp:dataSize>
          <grayscale>true</grayscale>
          <jpgQuality>93</jpgQuality>
          <adp:adaptedBy
           rdf:resource='http://svr.my-dom.org/ADP/jpgxcd-sdt'/>
        </adp:ImageProfile>
      </adp:representation>
    </adp:ImageProfile> <!-- END OF #p02-eighth_resl -->

  </adp:representation>
</adp:ImageProfile>
<!-- END OF http://svr.my-dom.org/images/boat.jpeg -->
```

```
  <adp:Adaptor rdf:about='http://svr.my-dom.org/ADP/jpgxcd-sdt'>
    <adp:runEnvironment>Linux executable</adp:runEnvironment>
    <adp:reversible>false</adp:reversible>
  </adp:Adaptor>

</rdf:RDF>
```

Profile of boat.jpg (for FDT) [http://svr.my-dom.org/images/boat.jpg.adp]

```
<?xml version='1.0'?>

<rdf:RDF
  xmlns:rdf  = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:rdfs = 'http://www.w3.org/2000/01/rdf-schema#'
  xmlns:adp  = 'http://rdfs.example.org/adp-schema#'
  xmlns      = 'http://svr.my-dom.org/ADP/adp-svr-profile#'
  xml:base   = 'http://svr.my-dom.org/images/boat.jpg'>

  <!-- Quality Indicator in rdf:ID
    0 - original
    1 - very high
    2 - high
    3 - average
    4 - low
    5 - very low
  -->

  <adp:ImageProfile
   rdf:about='http://svr.my-dom.org/images/boat.jpg'>
    <adp:contentType>image/jpeg</adp:contentType>
    <adp:dataSize>5065493</adp:dataSize>
    <adp:presentationWidth>1976</adp:presentationWidth>
    <adp:presentationHeight>2960</adp:presentationHeight>
    <grayscale>false</grayscale>
    <jpgScaledownFactor>1</jpgScaledownFactor>
    <jpgQuality>100</jpgQuality>
    <adp:adaptedBy
     rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>

    <adp:representation rdf:parseType='Collection'>

    <!-- Reduced-Resolution Representation(s) -->
      <adp:ImageProfile rdf:ID='p03-half_resl'>
        <adp:dataSize>1557164</adp:dataSize>
        <adp:presentationWidth>988</adp:presentationWidth>
        <adp:presentationHeight>1480</adp:presentationHeight>
        <jpgScaledownFactor>2</jpgScaledownFactor>
```

```
  <adp:adaptedBy
   rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>

  <adp:representation rdf:parseType='Collection'>

  <!-- Reduced-Quality Representation(s) -->
    <adp:ImageProfile rdf:ID='p00-def-half_resl-3'>
      <adp:dataSize>157538</adp:dataSize>
      <jpgQuality>57</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>
    </adp:ImageProfile>
  <!-- Reduced-Component Representation(s) -->
    <adp:ImageProfile rdf:ID='p00-half_resl-gray-3'>
      <adp:dataSize>130310</adp:dataSize>
      <grayscale>true</grayscale>
      <jpgQuality>50</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>
    </adp:ImageProfile>
  </adp:representation>
</adp:ImageProfile> <!-- END OF #p03-half_resl -->

<adp:ImageProfile rdf:ID='p01-quarter_resl'>
  <adp:dataSize>403494</adp:dataSize>
  <adp:presentationWidth>494</adp:presentationWidth>
  <adp:presentationHeight>740</adp:presentationHeight>
  <jpgScaledownFactor>4</jpgScaledownFactor>
  <adp:adaptedBy
   rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>

  <adp:representation rdf:parseType='Collection'>

  <!-- Reduced-Quality Representation(s) -->
    <adp:ImageProfile rdf:ID='p01-quarter_resl-2'>
      <adp:dataSize>203633</adp:dataSize>
      <jpgQuality>96</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p01-def-quarter_resl-3'>
      <adp:dataSize>115150</adp:dataSize>
      <jpgQuality>90</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p01-quarter_resl-4'>
      <adp:dataSize>63838</adp:dataSize>
      <jpgQuality>75</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>
    </adp:ImageProfile>
```

```
    <adp:ImageProfile rdf:ID='p01-quarter_resl-5'>
      <adp:dataSize>37742</adp:dataSize>
      <jpgQuality>43</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>
    </adp:ImageProfile>
  <!-- Reduced-Component Representation(s) -->
    <adp:ImageProfile rdf:ID='p01-quarter_resl-gray'>
      <adp:dataSize>293718</adp:dataSize>
      <grayscale>true</grayscale>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p01-def-quarter_resl-gray-3'>
      <adp:dataSize>87382</adp:dataSize>
      <grayscale>true</grayscale>
      <jpgQuality>87</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>
    </adp:ImageProfile>
  </adp:representation>
</adp:ImageProfile> <!-- END OF #p01-quarter_resl -->

<adp:ImageProfile rdf:ID='p02-eighth_resl'>
  <adp:dataSize>92727</adp:dataSize>
  <adp:presentationWidth>247</adp:presentationWidth>
  <adp:presentationHeight>370</adp:presentationHeight>
  <jpgScaledownFactor>8</jpgScaledownFactor>
  <adp:adaptedBy
   rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>

  <adp:representation rdf:parseType='Collection'>

  <!-- Reduced-Quality Representation(s) -->
    <adp:ImageProfile rdf:ID='p02-def-eighth_resl-3'>
      <adp:dataSize>62054</adp:dataSize>
      <jpgQuality>98</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p02-eighth_resl-4'>
      <adp:dataSize>23985</adp:dataSize>
      <jpgQuality>87</jpgQuality>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>
    </adp:ImageProfile>
  <!-- Reduced-Component Representations -->
    <adp:ImageProfile rdf:ID='p02-eighth_resl-gray-3'>
      <adp:dataSize>37114</adp:dataSize>
      <grayscale>true</grayscale>
      <jpgQuality>95</jpgQuality>
```

```
            <adp:adaptedBy
             rdf:resource='http://svr.my-dom.org/ADP/jpgxcd'/>
          </adp:ImageProfile>
        </adp:representation>
      </adp:ImageProfile> <!-- END OF #p02-eighth_res1 -->

    </adp:representation>
  </adp:ImageProfile>
  <!-- END OF http://svr.my-dom.org/images/boat.jpg -->

  <adp:Adaptor rdf:about='http://svr.my-dom.org/ADP/jpgxcd'>
    <adp:runEnvironment>Linux executable</adp:runEnvironment>
    <adp:reversible>false</adp:reversible>
  </adp:Adaptor>

</rdf:RDF>
```

Profile of boat.jp2 (for Modulation) [http://svr.my-dom.org/images/boat.jp2.adp]

```
<?xml version='1.0'?>

<rdf:RDF
  xmlns:rdf  = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:rdfs = 'http://www.w3.org/2000/01/rdf-schema#'
  xmlns:adp  = 'http://rdfs.example.org/adp-schema#'
  xmlns      = 'http://svr.my-dom.org/ADP/adp-svr-profile#'
  xml:base   = 'http://svr.my-dom.org/images/boat.jp2'>

  <!-- Quality Indicator in rdf:ID
    0 - original
    1 - very high
    2 - high
    3 - average
    4 - low
    5 - very low
  -->

  <adp:ImageProfile
   rdf:about='http://svr.my-dom.org/images/boat.jp2'>
    <adp:contentType>image/jp2</adp:contentType>
    <adp:dataSize>5065561</adp:dataSize>
    <adp:presentationWidth>1976</adp:presentationWidth>
    <adp:presentationHeight>2960</adp:presentationHeight>
    <grayscale>false</grayscale>
    <jp2ProgressionOrder>cprl</jp2ProgressionOrder>
    <jp2NoComponents>3</jp2NoComponents>
    <jp2NoResolutions>5</jp2NoResolutions>
```

```
<jp2NoLayers>10</jp2NoLayers>
<adp:adaptedBy
 rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>

<adp:representation rdf:parseType='Collection'>

<!-- Reduced-Resolution Representation(s) -->
  <adp:ImageProfile rdf:ID='p05-half_resl'>
    <adp:dataSize>2403166</adp:dataSize>
    <adp:presentationWidth>988</adp:presentationWidth>
    <adp:presentationHeight>1480</adp:presentationHeight>
    <jp2NoResolutions>4</jp2NoResolutions>
    <adp:adaptedBy
     rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>

    <adp:representation rdf:parseType='Collection'>

    <!-- Reduced-Quality Representation(s) -->
      <adp:ImageProfile rdf:ID='p00-def-half_resl-3'>
        <adp:dataSize>158733</adp:dataSize>
        <jp2NoLayers>5</jp2NoLayers>
        <adp:adaptedBy
         rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
      </adp:ImageProfile>
    <!-- Reduced-Component Representation(s) -->
      <adp:ImageProfile rdf:ID='p00-half_resl-gray-3'>
        <adp:dataSize>130454</adp:dataSize>
        <grayscale>true</grayscale>
        <jp2NoComponents>1</jp2NoComponents>
        <jp2NoLayers>5</jp2NoLayers>
        <adp:adaptedBy
         rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
      </adp:ImageProfile>
    </adp:representation>
  </adp:ImageProfile> <!-- END OF #p05-half_resl -->

  <adp:ImageProfile rdf:ID='p01-quarter_resl'>
    <adp:dataSize>790186</adp:dataSize>
    <adp:presentationWidth>494</adp:presentationWidth>
    <adp:presentationHeight>740</adp:presentationHeight>
    <jp2NoResolutions>3</jp2NoResolutions>
    <adp:adaptedBy
     rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>

    <adp:representation rdf:parseType='Collection'>

    <!-- Reduced-Quality Representation(s) -->
      <adp:ImageProfile rdf:ID='p01-quarter_resl-1'>
        <adp:dataSize>494679</adp:dataSize>
        <jp2NoLayers>8</jp2NoLayers>
        <adp:adaptedBy
         rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
      </adp:ImageProfile>
```

```
    <adp:ImageProfile rdf:ID='p01-quarter_resl-2'>
      <adp:dataSize>200964</adp:dataSize>
      <jp2NoLayers>6</jp2NoLayers>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p01-def-quarter_resl-3'>
      <adp:dataSize>115217</adp:dataSize>
      <jp2NoLayers>5</jp2NoLayers>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p01-quarter_resl-4'>
      <adp:dataSize>64299</adp:dataSize>
      <jp2NoLayers>4</jp2NoLayers>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p01-quarter_resl-5'>
      <adp:dataSize>37937</adp:dataSize>
      <jp2NoLayers>3</jp2NoLayers>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
    </adp:ImageProfile>
  <!-- Reduced-Component Representation(s) -->
    <adp:ImageProfile rdf:ID='p01-quarter_resl-gray'>
      <adp:dataSize>316270</adp:dataSize>
      <grayscale>true</grayscale>
      <jp2NoComponents>1</jp2NoComponents>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p01-def-quarter_resl-gray-3'>
      <adp:dataSize>86958</adp:dataSize>
      <grayscale>true</grayscale>
      <jp2NoComponents>1</jp2NoComponents>
      <jp2NoLayers>5</jp2NoLayers>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
    </adp:ImageProfile>
  </adp:representation>
</adp:ImageProfile> <!-- END OF #p01-quarter_resl -->

<adp:ImageProfile rdf:ID='p02-eighth_resl'>
  <adp:dataSize>236283</adp:dataSize>
  <adp:presentationWidth>247</adp:presentationWidth>
  <adp:presentationHeight>370</adp:presentationHeight>
  <jp2NoResolutions>2</jp2NoResolutions>
  <adp:adaptedBy
   rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
```

```
  <adp:representation rdf:parseType='Collection'>

  <!-- Reduced-Quality Representation(s) -->
    <adp:ImageProfile rdf:ID='p02-eighth_resl-2'>
      <adp:dataSize>127253</adp:dataSize>
      <jp2NoLayers>7</jp2NoLayers>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p02-def-eighth_resl-3'>
      <adp:dataSize>58439</adp:dataSize>
      <jp2NoLayers>5</jp2NoLayers>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
    </adp:ImageProfile>
    <adp:ImageProfile rdf:ID='p02-eighth_resl-4'>
      <adp:dataSize>23723</adp:dataSize>
      <jp2NoLayers>3</jp2NoLayers>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
    </adp:ImageProfile>
  <!-- Reduced-Component Representation(s) -->
    <adp:ImageProfile rdf:ID='p02-eighth_resl-gray-3'>
      <adp:dataSize>36860</adp:dataSize>
      <grayscale>true</grayscale>
      <jp2NoComponents>1</jp2NoComponents>
      <jp2NoLayers>5</jp2NoLayers>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
    </adp:ImageProfile>
  </adp:representation>
</adp:ImageProfile> <!-- END OF #p02-eighth_resl -->

<adp:ImageProfile rdf:ID='p03-sixteenth_resl'>
  <adp:dataSize>70624</adp:dataSize>
  <adp:presentationWidth>124</adp:presentationWidth>
  <adp:presentationHeight>185</adp:presentationHeight>
  <jp2NoResolutions>1</jp2NoResolutions>
  <adp:adaptedBy
   rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>

  <adp:representation rdf:parseType='Collection'>

  <!-- Reduced-Quality Representation(s) -->
    <adp:ImageProfile rdf:ID='p03-def-sixteenth_resl-3'>
      <adp:dataSize>25438</adp:dataSize>
      <jp2NoLayers>5</jp2NoLayers>
      <adp:adaptedBy
       rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
    </adp:ImageProfile>
  <!-- Reduced-Component Representation(s) -->
    <adp:ImageProfile rdf:ID='p03-sixteenth_resl-gray-3'>
```

```
            <adp:dataSize>13356</adp:dataSize>
            <grayscale>true</grayscale>
            <jp2NoComponents>1</jp2NoComponents>
            <jp2NoLayers>5</jp2NoLayers>
            <adp:adaptedBy
             rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
          </adp:ImageProfile>
        </adp:representation>
      </adp:ImageProfile> <!-- END OF #p03-sixteenth_resl -->

      <adp:ImageProfile rdf:ID='p04-thirtysecond_resl-3'>
        <adp:dataSize>10672</adp:dataSize>
        <adp:presentationWidth>62</adp:presentationWidth>
        <adp:presentationHeight>93</adp:presentationHeight>
        <jp2NoResolutions>0</jp2NoResolutions>
        <jp2NoLayers>5</jp2NoLayers>
        <adp:adaptedBy
         rdf:resource='http://svr.my-dom.org/ADP/jp2mod'/>
      </adp:ImageProfile>

    </adp:representation>
  </adp:ImageProfile>
  <!-- END OF http://svr.my-dom.org/images/boat.jp2 -->

  <adp:Adaptor rdf:about='http://svr.my-dom.org/ADP/jp2mod'>
    <adp:runEnvironment>Linux executable</adp:runEnvironment>
    <adp:reversible>true</adp:reversible>
  </adp:Adaptor>

</rdf:RDF>
```

Appendix C

Experimental Client Meta-Data

CC/PP Schema Extension [http://cli.my-dom.org/CCPP/gen-cli-profile]

```
<?xml version='1.0'?>

<!DOCTYPE rdf:RDF [
  <!ENTITY ns-rdf   'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <!ENTITY ns-ccpp  'http://www.w3.org/2002/11/08-ccpp-schema#'>
  <!ENTITY ns-cprof 'http://cli.my-dom.org/CCPP/gen-cli-profile#'>
]>

<rdf:RDF
  xmlns:rdf  = '&ns-rdf;'
  xmlns:rdfs = 'http://www.w3.org/2000/01/rdf-schema#'
  xmlns:ccpp = '&ns-ccpp;'>

<!-- CC/PP Component Definitions -->
  <rdfs:Class rdf:about='&ns-cprof;HardwarePlatform'>
    <rdfs:label xml:lang="en">CC/PP Hardware Platform Component
      </rdfs:label>
    <rdfs:subClassOf rdf:resource='&ns-ccpp;Component'/>
    <rdfs:comment xml:lang="en">
      This class is to specify hardware component.
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about='&ns-cprof;SoftwarePlatform'>
    <rdfs:label xml:lang="en">CC/PP Software Platform Component
      </rdfs:label>
    <rdfs:subClassOf rdf:resource='&ns-ccpp;Component'/>
    <rdfs:comment xml:lang="en">
      This class is to specify software component.
    </rdfs:comment>
  </rdfs:Class>

  <rdfs:Class rdf:about='&ns-cprof;BrowserUA'>
    <rdfs:label xml:lang="en">CC/PP Browser User Agent Component
      </rdfs:label>
    <rdfs:subClassOf rdf:resource='&ns-ccpp;Component'/>
    <rdfs:comment xml:lang="en">
      This class is to specify browser component.
```

```
    </rdfs:comment>
  </rdfs:Class>

<!-- CC/PP Attribute Definitions -->

  <ccpp:Attribute rdf:about='&ns-cprof;name'>
    <rdfs:label xml:lang="en">Item's Name</rdfs:label>
    <rdfs:domain rdf:resource='&ns-ccpp;Component'/>
    <rdfs:range  rdf:resource='&ns-ccpp;string'/>
    <rdfs:comment xml:lang="en">
      A string describing an item's name.
    </rdfs:comment>
  </ccpp:Attribute>

  <ccpp:Attribute rdf:about='&ns-cprof;vendor'>
    <rdfs:label xml:lang="en">Item's Vendor</rdfs:label>
    <rdfs:domain rdf:resource='&ns-ccpp;Component'/>
    <rdfs:range  rdf:resource='&ns-ccpp;string'/>
    <rdfs:comment xml:lang="en">
      A string describing the vendor of the item.
    </rdfs:comment>
  </ccpp:Attribute>

  <ccpp:Attribute rdf:about='&ns-cprof;version'>
    <rdfs:label xml:lang="en">Item's Version</rdfs:label>
    <rdfs:domain rdf:resource='&ns-ccpp;Component'/>
    <rdfs:range  rdf:resource='&ns-ccpp;string'/>
    <rdfs:comment xml:lang="en">
      A string describing the version of the item.
    </rdfs:comment>
  </ccpp:Attribute>

  <ccpp:Attribute rdf:about='&ns-cprof;cpu'>
    <rdfs:label xml:lang="en">CPU Type</rdfs:label>
    <rdfs:domain rdf:resource='&ns-cprof;HardwarePlatform'/>
    <rdfs:range  rdf:resource='&ns-ccpp;string'/>
    <rdfs:comment xml:lang="en">
      A string describing the CPU type.
    </rdfs:comment>
  </ccpp:Attribute>

  <ccpp:Attribute rdf:about='&ns-cprof;scrollbars'>
    <rdfs:label xml:lang="en">Supported Scrollbars</rdfs:label>
    <rdfs:domain rdf:resource='&ns-cprof;BrowserUA'/>
    <rdfs:range  rdf:resource='&ns-rdf;Bag'/>
    <rdfs:comment xml:lang="en">
      A list of supported scrollbars in the browser's window
      (i.e.: horizontal, vertical).
    </rdfs:comment>
  </ccpp:Attribute>

  <ccpp:Attribute rdf:about='&ns-cprof;htmlVersionsSupported'>
    <rdfs:label xml:lang="en">Supported HTML Versions</rdfs:label>
```

```
    <rdfs:domain rdf:resource='&ns-cprof;BrowserUA'/>
    <rdfs:range  rdf:resource='&ns-rdf;Bag'/>
    <rdfs:comment xml:lang="en">
      A list of supported HTML versions.
    </rdfs:comment>
  </ccpp:Attribute>

  <ccpp:Attribute rdf:about='&ns-cprof;maxObjectSize'>
    <rdfs:label xml:lang="en">Max Size of a Web Object</rdfs:label>
    <rdfs:domain rdf:resource='&ns-cprof;BrowserUA'/>
    <rdfs:range  rdf:resource='&ns-ccpp;integer'/>
    <rdfs:comment xml:lang="en">
      An integer describing the maximum size of a web object to be
      presented.
    </rdfs:comment>
  </ccpp:Attribute>

</rdf:RDF>
```

Profile of Client1 [http://cli.my-dom.org/CCPP/Client1]

```
<?xml version='1.0'?>

<!DOCTYPE rdf:RDF [
  <!ENTITY ns-ccpp  'http://www.w3.org/2002/11/08-ccpp-schema#'>
  <!ENTITY ns-cprof 'http://cli.my-dom.org/CCPP/gen-cli-profile#'>
]>

<rdf:RDF
  xmlns:rdf      = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:ccpp     = '&ns-ccpp;'
  xmlns:ccpp-cli = 'http://www.w3.org/2002/11/08-ccpp-client#'
  xmlns:prf      = '&ns-cprof;'>

  <rdf:Description rdf:about='http://cli.my-dom.org/CCPP/Client1'>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Hardware_Color'>
        <rdf:type rdf:resource='&ns-cprof;HardwarePlatform'/>
        <prf:name>Desky</prf:name>
        <prf:vendor>Dell</prf:vendor>
        <prf:version>Optiplex GX280</prf:version>
        <prf:cpu>Intel Pentium 4 -- 3.0 GHz</prf:cpu>
        <ccpp-cli:pix-x>1024</ccpp-cli:pix-x>
        <ccpp-cli:pix-y>768</ccpp-cli:pix-y>
        <ccpp-cli:color>full</ccpp-cli:color>
      </rdf:Description>
    </ccpp:component>
```

```
    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Software'>
        <rdf:type rdf:resource='&ns-cprof;SoftwarePlatform'/>
        <prf:name>GNU/Linux</prf:name>
        <prf:vendor>Fedora Core</prf:vendor>
        <prf:version>3.0</prf:version>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Browser_Scroll'>
        <rdf:type rdf:resource='&ns-cprof;BrowserUA'/>
        <prf:name>Firefox</prf:name>
        <prf:vendor>Mozilla</prf:vendor>
        <prf:version>1.0.1</prf:version>
        <prf:scrollbars>
          <rdf:Bag>
            <rdf:li>horizontal</rdf:li>
            <rdf:li>vertical</rdf:li>
          </rdf:Bag>
        </prf:scrollbars>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>

</rdf:RDF>
```

Profile of Client2 [http://cli.my-dom.org/CCPP/Client2]

```
<?xml version='1.0'?>

<!DOCTYPE rdf:RDF [
  <!ENTITY ns-ccpp  'http://www.w3.org/2002/11/08-ccpp-schema#'>
  <!ENTITY ns-cprof 'http://cli.my-dom.org/CCPP/gen-cli-profile#'>
]>

<rdf:RDF
  xmlns:rdf      = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:ccpp     = '&ns-ccpp;'
  xmlns:ccpp-cli = 'http://www.w3.org/2002/11/08-ccpp-client#'
  xmlns:prf      = '&ns-cprof;'>
  <rdf:Description rdf:about='http://cli.my-dom.org/CCPP/Client2'>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Hardware_Grey'>
```

```
        <rdf:type rdf:resource='&ns-cprof;HardwarePlatform'/>
        <prf:name>Desky</prf:name>
        <prf:vendor>Dell</prf:vendor>
        <prf:version>Optiplex GX280</prf:version>
        <prf:cpu>Intel Pentium 4 -- 3.0 GHz</prf:cpu>
        <ccpp-cli:pix-x>1024</ccpp-cli:pix-x>
        <ccpp-cli:pix-y>768</ccpp-cli:pix-y>
        <ccpp-cli:color>grey</ccpp-cli:color>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Software'>
        <rdf:type rdf:resource='&ns-cprof;SoftwarePlatform'/>
        <prf:name>GNU/Linux</prf:name>
        <prf:vendor>Fedora Core</prf:vendor>
        <prf:version>3.0</prf:version>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Browser_Scroll'>
        <rdf:type rdf:resource='&ns-cprof;BrowserUA'/>
        <prf:name>Firefox</prf:name>
        <prf:vendor>Mozilla</prf:vendor>
        <prf:version>1.0.1</prf:version>
        <prf:scrollbars>
          <rdf:Bag>
            <rdf:li>horizontal</rdf:li>
            <rdf:li>vertical</rdf:li>
          </rdf:Bag>
        </prf:scrollbars>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>

</rdf:RDF>
```

Profile of Client3 [http://cli.my-dom.org/CCPP/Client3]

```
<?xml version='1.0'?>

<!DOCTYPE rdf:RDF [
  <!ENTITY ns-ccpp  'http://www.w3.org/2002/11/08-ccpp-schema#'>
  <!ENTITY ns-cprof 'http://cli.my-dom.org/CCPP/gen-cli-profile#'>
]>
```

```
<rdf:RDF
  xmlns:rdf      = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:ccpp     = '&ns-ccpp;'
  xmlns:ccpp-cli = 'http://www.w3.org/2002/11/08-ccpp-client#'
  xmlns:prf      = '&ns-cprof;'>

  <rdf:Description rdf:about='http://cli.my-dom.org/CCPP/Client3'>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Hardware_Color'>
        <rdf:type rdf:resource='&ns-cprof;HardwarePlatform'/>
        <prf:name>Desky</prf:name>
        <prf:vendor>Dell</prf:vendor>
        <prf:version>Optiplex GX280</prf:version>
        <prf:cpu>Intel Pentium 4 -- 3.0 GHz</prf:cpu>
        <ccpp-cli:pix-x>1024</ccpp-cli:pix-x>
        <ccpp-cli:pix-y>768</ccpp-cli:pix-y>
        <ccpp-cli:color>full</ccpp-cli:color>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Software'>
        <rdf:type rdf:resource='&ns-cprof;SoftwarePlatform'/>
        <prf:name>GNU/Linux</prf:name>
        <prf:vendor>Fedora Core</prf:vendor>
        <prf:version>3.0</prf:version>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Browser_NoScroll'>
        <rdf:type rdf:resource='&ns-cprof;BrowserUA'/>
        <prf:name>Firefox</prf:name>
        <prf:vendor>Mozilla</prf:vendor>
        <prf:version>1.0.1</prf:version>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>

</rdf:RDF>
```

Profile of Client4 [http://cli.my-dom.org/CCPP/Client4]

```
<?xml version='1.0'?>
```

```
<!DOCTYPE rdf:RDF [
  <!ENTITY ns-ccpp  'http://www.w3.org/2002/11/08-ccpp-schema#'>
  <!ENTITY ns-cprof 'http://cli.my-dom.org/CCPP/gen-cli-profile#'>
]>

<rdf:RDF
  xmlns:rdf      = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:ccpp     = '&ns-ccpp;'
  xmlns:ccpp-cli = 'http://www.w3.org/2002/11/08-ccpp-client#'
  xmlns:prf      = '&ns-cprof;'>

  <rdf:Description rdf:about='http://cli.my-dom.org/CCPP/Client4'>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Hardware_Grey'>
        <rdf:type rdf:resource='&ns-cprof;HardwarePlatform'/>
        <prf:name>Desky</prf:name>
        <prf:vendor>Dell</prf:vendor>
        <prf:version>Optiplex GX280</prf:version>
        <prf:cpu>Intel Pentium 4 -- 3.0 GHz</prf:cpu>
        <ccpp-cli:pix-x>1024</ccpp-cli:pix-x>
        <ccpp-cli:pix-y>768</ccpp-cli:pix-y>
        <ccpp-cli:color>grey</ccpp-cli:color>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Software'>
        <rdf:type rdf:resource='&ns-cprof;SoftwarePlatform'/>
        <prf:name>GNU/Linux</prf:name>
        <prf:vendor>Fedora Core</prf:vendor>
        <prf:version>3.0</prf:version>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PC_Browser_NoScroll'>
        <rdf:type rdf:resource='&ns-cprof;BrowserUA'/>
        <prf:name>Firefox</prf:name>
        <prf:vendor>Mozilla</prf:vendor>
        <prf:version>1.0.1</prf:version>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>

</rdf:RDF>
```

Profile of Client5 [http://cli.my-dom.org/CCPP/Client5]

```
<?xml version='1.0'?>

<!DOCTYPE rdf:RDF [
  <!ENTITY ns-ccpp  'http://www.w3.org/2002/11/08-ccpp-schema#'>
  <!ENTITY ns-cprof 'http://cli.my-dom.org/CCPP/gen-cli-profile#'>
]>

<rdf:RDF
  xmlns:rdf      = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:ccpp     = '&ns-ccpp;'
  xmlns:ccpp-cli = 'http://www.w3.org/2002/11/08-ccpp-client#'
  xmlns:prf      = '&ns-cprof;'>

  <rdf:Description rdf:about='http://cli.my-dom.org/CCPP/Client5'>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PDA_Hardware_Color'>
        <rdf:type rdf:resource='&ns-cprof;HardwarePlatform'/>
        <prf:name>Palmo</prf:name>
        <prf:vendor>PalmOne</prf:vendor>
        <prf:version>Tungsten</prf:version>
        <prf:cpu>Intel XScale</prf:cpu>
        <ccpp-cli:pix-x>320</ccpp-cli:pix-x>
        <ccpp-cli:pix-y>480</ccpp-cli:pix-y>
        <ccpp-cli:color>full</ccpp-cli:color>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PDA_Software'>
        <rdf:type rdf:resource='&ns-cprof;SoftwarePlatform'/>
        <prf:name>Palm OS</prf:name>
        <prf:vendor>PalmSource</prf:vendor>
        <prf:version>5.4</prf:version>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PDA_Browser'>
        <rdf:type rdf:resource='&ns-cprof;BrowserUA'/>
        <prf:name>Blazer</prf:name>
        <prf:vendor>Handspring</prf:vendor>
        <prf:version>4.0</prf:version>
        <prf:scrollbars>
          <rdf:Bag>
```

```
            <rdf:li>horizontal</rdf:li>
            <rdf:li>vertical</rdf:li>
          </rdf:Bag>
        </prf:scrollbars>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>

</rdf:RDF>
```

Profile of Client6 [http://cli.my-dom.org/CCPP/Client6]

```
<?xml version='1.0'?>

<!DOCTYPE rdf:RDF [
  <!ENTITY ns-ccpp  'http://www.w3.org/2002/11/08-ccpp-schema#'>
  <!ENTITY ns-cprof 'http://cli.my-dom.org/CCPP/gen-cli-profile#'>
]>

<rdf:RDF
  xmlns:rdf      = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:ccpp     = '&ns-ccpp;'
  xmlns:ccpp-cli = 'http://www.w3.org/2002/11/08-ccpp-client#'
  xmlns:prf      = '&ns-cprof;'>

  <rdf:Description rdf:about='http://cli.my-dom.org/CCPP/Client6'>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PDA_Hardware_Grey'>
        <rdf:type rdf:resource='&ns-cprof;HardwarePlatform'/>
        <prf:name>Palmo</prf:name>
        <prf:vendor>PalmOne</prf:vendor>
        <prf:version>Tungsten</prf:version>
        <prf:cpu>Intel XScale</prf:cpu>
        <ccpp-cli:pix-x>320</ccpp-cli:pix-x>
        <ccpp-cli:pix-y>480</ccpp-cli:pix-y>
        <ccpp-cli:color>grey</ccpp-cli:color>
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PDA_Software'>
        <rdf:type rdf:resource='&ns-cprof;SoftwarePlatform'/>
        <prf:name>Palm OS</prf:name>
        <prf:vendor>PalmSource</prf:vendor>
        <prf:version>5.4</prf:version>
```

```
      </rdf:Description>
    </ccpp:component>

    <ccpp:component>
      <rdf:Description
       rdf:about='http://cli.my-dom.org/CCPP/PDA_Browser'>
        <rdf:type rdf:resource='&ns-cprof;BrowserUA'/>
        <prf:name>Blazer</prf:name>
        <prf:vendor>Handspring</prf:vendor>
        <prf:version>4.0</prf:version>
        <prf:scrollbars>
          <rdf:Bag>
            <rdf:li>horizontal</rdf:li>
            <rdf:li>vertical</rdf:li>
          </rdf:Bag>
        </prf:scrollbars>
      </rdf:Description>
    </ccpp:component>

  </rdf:Description>

</rdf:RDF>
```