Multi-resolution Browsing of Web Documents in Distributed Agent Environment

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING
OF THE HONG KONG POLYTECHNIC UNIVERSITY
IN PARTIALLY FULFILLMENT OF THE REQUIREMENT
FOR THE DEGREE OF
MASTER OF PHILOSOPHY

By

Stanley M. T. Yau

The Hong Kong Polytechnic University

2001

Abstract

A mobile environment is weakly connected, characterized by low communication bandwidth and poor network connectivity. Users retrieving a web document in the mobile environment may have to wait for a long period before actually reading the document, possibly wasting time to obtain documents that they are not interested in. Therefore, we present a multi-resolution transmission and browsing mechanism for web documents. A web document is partitioned into multiple organizational units based on the significance of the information content in the document. A unit with a higher information content, indicating a higher amount of information captured by the unit, will be transmitted earlier. The mobile users will be able to explore the more content-bearing portion of the document earlier and terminate the transmission of an irrelevant document sooner.

The higher content-bearing portion of the document should be transmitted successfully with a high probability. This ensures that the mobile users could at least obtain a high level content of the document and determine whether the corrupted portions need to be retransmitted. We present the fault-tolerant multi-resolution transmission mechanism using the Vandermonde-Matrix based Forward Error Correction (VMFEC) coding method. An organizational unit can be divided into M raw packets, and then an encoder transforms the M raw packets to produce $N \geq M$ cooked packets. The mobile users can use the decoder to retrieve any combination of M cooked packets for reconstructing the original organizational unit.

We employ a mobile agent model to combat the problem of poor network connectivity. The mobile agent can migrate to a base station that virtually establishes a connection to the mobile user. Therefore the mobile user could assume that the connection seldom gets broken and does not need to be concerned with the connectivity any more. Our research efforts are integrated in the Distributed Agent Environment (DAE), which supports the mobile agent in the mobile environment. The DAE provides the mobile agent to perform the mobile user's requests, store the result in the user's profile, and retrieve the result from the profile when connectivity from the mobile user is available.

Finally, we design and implement the mobile web browser based on multi-resolution transmission and the mobile agent model and integrate into DAE to demonstrate the look and feel of the multi-resolution web browsing paradigm. We make the performance comparison between the mobile agent models against the conventional client/server models. We have illustrated with a number of experiments that returning results actively by mobile agent is an effective solution.

Acknowledgement

Many thanks to my supervisor Dr. Hong-Va Leong, and my co-supervisor Dr. Antonio Si for their guidance, constructive criticism and gentle natures.

Many thanks to all my research fellows, Boris, Ken and Jimmy, who made the journey far easier and share their experiences.

I also thank all research students and technical team in the department of computing, who provide a warm and comfortable environment.

Last but not least, I have to thank my family and girl friend for their endless love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we focus on a mobile environment in which *mobile users* are often allowed to navigate web documents via web browsers with mobile devices, which could be laptop computers with reasonable computing power and storage. We term such an environment, a *mobile web environment.* Since communication between the mobile client and the base station in the mobile web environment is via low communication bandwidth channels with poor network connectivity, it is important that traffic generated due to web access consumes as little bandwidth as possible. Therefore, the conventional approach to web navigation suffers from serious limitations.

One common conventional approach navigates web documents by submitting *querying keywords* to a search engine for the retrieval of some relevant documents. If the mobile user needs to refine those results by more specific criteria, it could generate more request and response messages in a network. If the *HyperText Transfer Protocol* (HTTP) is adopted, each pair of messages requires the creation of a new session in a web server [46]. It is obvious that bi-directional communication is too expensive in the mobile web environment. The more communication traffic is generated, the more power and bandwidth are consumed. To optimize and utilize the limited bandwidth and save the power, we propose a *mobile agent* model as a representative of the mobile client to perform the

tasks in the base station. Furthermore, we propose a *multi-resolution transmission and browsing* mechanism which allows the mobile client to navigate a web document at various resolution by controlling the *Level of Detail* (LOD). In addition, the multi-resolution transmission and browsing mechanism allows a document to be transmitted and browsed at coarser resolution and fills in details progressively in the order of importance or other sequence at will. It is similar to the concept of progressive JPEG, where the more significant coefficients are transmitted earlier and insignificant ones are transmitted later. The mechanism allows the mobile user to focus on the most interesting portion of the document and decide early to stop transmission when the mobile user shifts his/her interest to other documents. This technique could minimize the use of scarce wireless bandwidth with early termination of transmission of irrelevant documents.

One limitation of the multi-resolution transmission paradigm is its lack of resilience to faulty transmission. An organizational unit could get corrupted while being transmitted via a faulty wireless channel [33]. A mobile client is not able to determine if the corrupted units are content-bearing and thus the whole document will need to be retransmitted. Since retransmitting the whole document is expensive, it is important that high content-bearing units should have a high probability of being transmitted successfully so that a mobile client could at least obtain a high level content of the document and determine if the remaining units need to be transmitted.

We extend the multi-resolution transmission paradigm with a fault-tolerant transmission capability, which enhances the transmission of important organizational units so that a mobile client could recover the units sent over the unreliable network due to data corruption. We call this scheme *fault-tolerant multi-resolution transmission.* The mobile client is able to obtain an overall content of a web document and either terminate the transmission of the remaining document or decide if the corrupted portions of the document need to be transmitted.

In conventional mobile client/server model, the server, which is often represented by

the base station, provides a fixed set of operations that mobile client can invoke from a remote machine. If the base station does not provide an operation that matches the mobile client's task exactly, either the mobile client must make a series of cross network calls to lower-level operations, or the base station developer must add a new operation to the base station. The first option brings much intermediate data across the network, potentially wasting the scarce bandwidth, especially if the intermediate data are not useful beyond the end of the mobile client task. The second option is an intractable programming task as the number of distinct mobile clients increases. In addition, it discourages modern software engineering since the server becomes complex, providing many specialized routines rather than providing simple, general primitives.

Mobile agents, on the other hand, do not waste extra bandwidth even if the base station provides only low-level operations, simply because an agent can migrate to the base station where it performs any desired processing before returning just the final result to the mobile client. It is important that the mobile agent avoids more intermediate messages and conserves more bandwidth. Despite these advantages, there are two tradeoffs that must be considered. First, although a mobile agent eliminates the transmission of intermediate data, the agent itself must be sent to the base station. If the size of the agent is larger than that of the intermediate data, it will obviously consume more bandwidth than a conventional mobile client/server model implementation. Second, the mobile agent uses less processing time at the mobile client but more time at the server, since the agent performs all intermediate computations at the base station and is typically written in a cross-platform programming language, e.g. Java. If the base station is heavily loaded, a CPU-intensive agent can easily lead to a longer response time than the corresponding cross network calls. Together, these tradeoffs mean that the mobile agent is a suitable model in an environment with low network bandwidth, high network load, high connection fee, high amount of intermediate data, and high server power.

In fact, mobile agent model can minimize network traffic and provide a non-stop pro-

cessing environment even under frequent disconnection conditions. The mobile client and the base station are de-coupled in the sense that instead of getting intermediate results many times, the mobile client interacts with the network only when it is submitting the mobile agent and when the mobile agent returns with result data. For example, a laptop computer or other mobile devices can send an agent into the network; the agent will continue with its work on behalf of the client even if the mobile device disconnects and will be ready with the result when the mobile device reconnects. This asynchronous interaction between the mobile client and the base station provides robustness against frequent disconnection suffered by the mobile environment. In addition, since the mobile agent can incorporate a mobile client's preference, less data need to be transmitted over the network. To this end, we propose a *Distributed Agent Environment* (DAE) that is designed to provide a robust runtime environment for mobile agents. In other words, mobile agents have the ability to orchestrate the tasks and store the result data in the base station whenever disconnection occurs.

The rest of this thesis is organized as follows. The background for mobile computing model and markup language are described in Chapter 2. In Chapter 3, we study and discuss relevant related research work. We elaborate in details the multi-resolution transmission and browsing mechanism in Section 4.1. The Vandermonde-Matrix based Forward Error Correction (VMFEC) coding method is discussed in Section 4.2. The Distributed Agent Environment (DAE) is proposed in Section 4.3. The implementation and prototype of mobile web browser and DAE are described in Chapter 5. We also conduct a performance study of Wisconsin benchmark using the mobile agent-based computing and compare the performance with the fault-tolerant transmission scheme in Chapter 6. Chapter 7 presents a conclusion of this research work. Finally, Chapter 8 highlights our future work.

Chapter 2

Background

In this chapter, we describe the background knowledge on mobile computing models and markup languages. In Section 2.1, we elaborate the strengths and weaknesses on various mobile agent models. In Section 2.2, we give an introdution to the evolution of markup langauges.

2.1 Mobile Computing Models

Devising appropriate models for structuring applications that involve wireless elements is an important issue to developing software for mobile computing. In this section, we consider extensions to the popular client/server model, which can be classified into mobile client/server model [26], mobile client/agent/server model [31] and mobile client/intercept/server model [24], and finally mobile agent model [29].

2.1.1 Mobile Client/Server Model

A client is an application component executing in a computing system to request the service. A server is an application component executing in another computing system to provide the service. In a mobile environment (see Figure 2.1 (a)), an important parame-

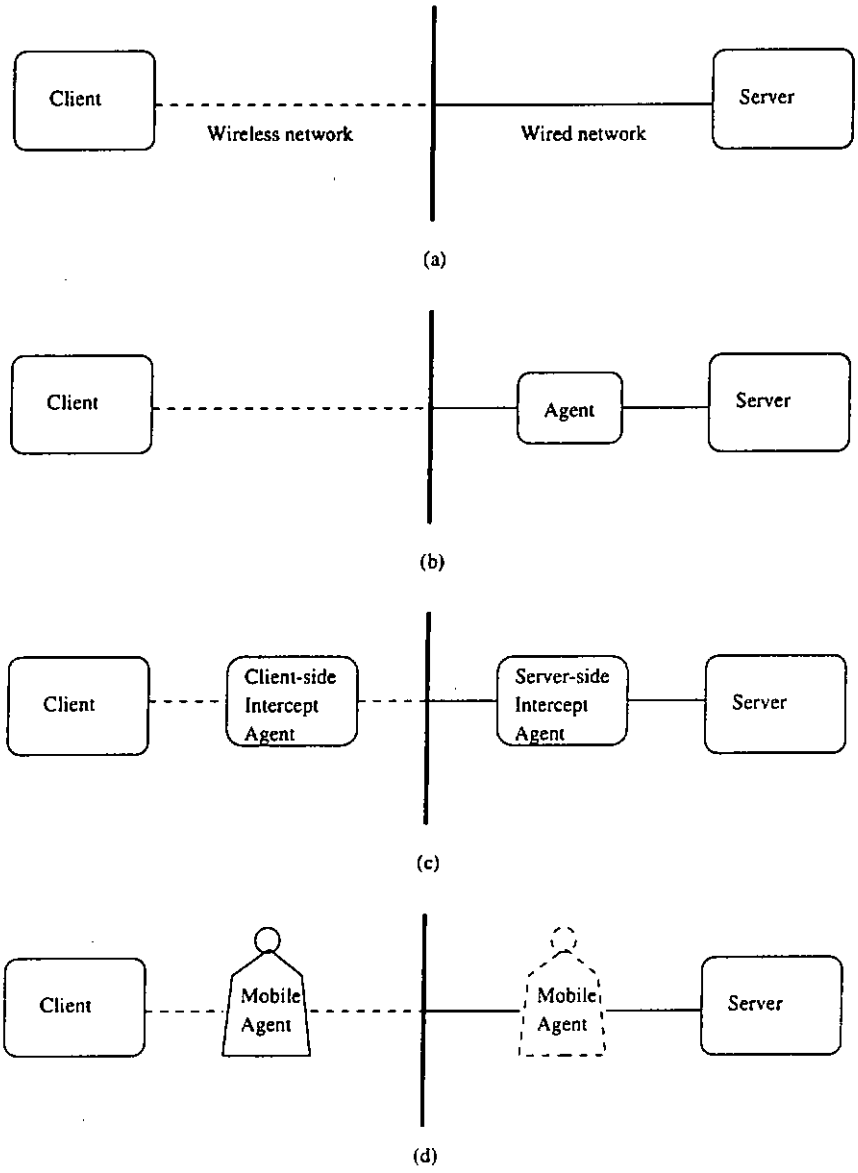

Figure 2.1: Mobile Computing Models (a) Mobile Client/Server Model, (b) Mobile Client/Agent/Server Model, (c) Mobile Client/Intercept/Server Model, (d) Mobile Agent Model.

ter in the design of a client/server architecture is the type of communication mechanism used to exchange information between mobile client and server. One way is the direct exchange of messages between the mobile client and the server. This model is not adequate for slow and unreliable networks as in the mobile environment. In this environment, syn-

chronous *Remote Procedure Call* (RPC) is inadequate since the mobile client is blocked in case of disconnection. In addition, conventional RPC leaves small chance of reducing communication cost [56]. To address these problems, there have been various proposals for asynchronous RPC. In asynchronous RPC, when an application issues an RPC, the RPC is stored in a local stable log and control is immediately returned to the application. When the client is connected, the log is drained in background and any queued RPC is forwarded to the server. Queuing RPC leaves space for performing various optimizations on the log. For example, the Rover toolkit [26] reorders logged requests based on consistency requirements and application-specified operation priorities. Specifically, if a mobile client is disconnected between issuing the request and receiving the reply, the server may periodically attempt to contact the mobile client and deliver the reply. This extended RPC mechanism enables applications to use different communication channels for the request and response.

Extensions to the conventional client/server model, such as queuing RPC, are necessary to sustain disconnected operation and weak connectivity. Further optimizations, such as filtering or compression, are also important. Therefore, the conventional client/server model must be extended to provide components responsible for implementing such appropriate optimizations and imposing minimum changes in clients and servers. In addition, for lightweight and unreliable mobile clients, it is critical to be able to move part of their operation to the wired network. Such considerations lead to the extensions of the client/server model presented in the following subsections.

2.1.2 The Mobile Client/Agent/Server Model

The most popular computational model in a mobile environment is a three-tier or mobile client/agent/server (c/a/s) model. The mobile client sends the request message to the agent and the agent forwards to the server (see Figure 2.1(b)). In general, an agent is just a surrogate of the mobile client on the wired network. This architecture reduces

the impact of the limited bandwidth and the poor reliability of the wireless network by continuously maintaining the mobile client's presence on the wired network via the agent. Agents split the interaction between mobile client and server in two parts, one between the mobile client and the agent, and one between the agent and the server. Different protocols can be used for each part of the interaction and each part of the interaction may be executed independently of the other.

Minimal functionality at the agent includes support for messaging and queuing for communication between the mobile client and the agent. The agent can also assume a more active role [8]. When application-specific predefined events occur, it notifies the client appropriately. To reduce computations on the mobile unit, the agent might be responsible for starting or stopping specific functions at the mobile unit or executing client specific services. The c/a/s model is the most appropriate for lightweight mobile clients of limited computational power and resources. It moves responsibilities from the mobile client to the agent. For example, the agent can manage a high computational request of a mobile client with only final result transmitted back to the mobile client. Located on the wired network, the agent has access to high bandwidth links and large computational resources, that it can use to benefit its mobile client. Similarly, performing caching at the agent can minimize long round trip delay on the network and reduce application response time.

To deal with disconnection, a mobile client can submit its requests to the agent and wait to retrieve the result data when connection is re-established. In the meantime, any request to the disconnected mobile client can be queued at the agent to be transferred upon reconnection. The agent can be used in a similar way to preserve battery life. The mobile client submits its request to the agent, enters the doze mode, and waits for the agent to wake it up when the response becomes available. The agent can effectively handle weak connectivity in a variety of ways. The agent can employ a number of optimization techniques to minimize the size of data to be transmitted to the mobile client depending

on the type of data and on the specific application [3, 40]. The agent can manipulate the data before their transmission to the mobile client, by changing their transmission order so that the most important information is transferred first [31], by performing data specific lossy compression that customizes content to the specific constraints of the client, or by batching together multiple replies [61].

While the mobile client/agent/server model offers a number of advantages, it fails to maintain the current computation at the mobile client during periods of disconnection. At the event of a disconnection, the mobile client cannot continue to operate uninterrupted. Finally, the agent can directly optimize only data transmission over the wireless network from the wired network to the mobile client but not vice versa.

2.1.3 The Mobile Client/Intercept/Server Model

To address the shortcomings of the c/a/s model, Housel et al. [24] proposed the deployment of a client-side agent that will run at the end-user mobile device along with the agent of the c/a/s model that runs within the wired network (see Figure 2.1 (c)). The client-side agent intercepts client's request, and together with the server-side agent, performs optimizations to reduce data transmission over the wireless network, improve data availability and sustain uninterrupted mobile computation. From the mobile client's point of view, the client-side agent appears as the local server proxy that is co-resident with the client. Similarly, the server-side agent appears as the local client proxy that resides on the wired network. This model can be viewed as the adaptation of the mobile client/server model for a mobile environment with the pair of agents responsible for minimizing the effect of the mobile environment. Since the pair of agents is virtually inserted in the data path between the mobile client and the server, the intercept model is transparent to both the mobile client and the server. Therefore, the agent pair can be employed with any mobile client application.

The communication protocol between the two agents can facilitate highly effective data

reduction and protocol optimization without limiting the functionality or interoperability of the client. The cooperation of the two agents allows for more efficient optimizations of the wireless network from the benefit of different applications. In addition, the agent pair can realize application specific optimizations more effectively.

This model offers flexibility in handling disconnection. For instance, a local cache may be maintained at client-side agent. The cache can be used to satisfy the mobile client's requirements for data during disconnection. Cache misses may be queued by the client-side agent to be served upon reconnection. Similarly, requests to the mobile client can be queued at the server-side agent and transferred to the mobile client upon reconnection. Weak connectivity can also be handled in a variety of ways. For example, relocating computation from the client-side agent to the server-side agent or vice versa can minimize the effect of weak connectivity.

The main weakness of the model is that it is more appropriate for heavy-weight mobile clients with enough computational power and secondary storage. Furthermore, every application requires development work both at the server and at the mobile client. However, there is no need to develop a pair of agents for every instance of an application. Since the functionality and optimizations performed by the agent pair is generic enough, it is only required to develop a different pair of agents per application.

2.1.4 Mobile Agent Model

Mobile agent is a process dispatched from a source computer to accomplish a specified task [12, 29]. Each mobile agent is a computation along with its own data and execution state. In Figure 2.1 (d), the mobile agent paradigm extends the RPC communication mechanism according to which a message sent by a mobile client is just a procedure call. After its submission, the mobile agent performs processing autonomously and independently of the sending mobile client. When the agent reaches a server, it is delivered to an *agent execution environment.* Then if the agent possesses necessary authentication

credentials, its executable parts are started. To accomplish its task, the mobile agent can transport it to another server, spawn new agents, or interact with other agents. Upon completion, the mobile agent delivers the results to the sending mobile client or to another server. The mobile agent computational paradigm is not orthogonal to the mobile client/server model and its extensions. For example, the agent of the c/a/s model may be seen as a form of static agent. It is because the static agent lacks the ability to migrate to other servers.

One of the main obstacles to the acceptance of mobile agent in commercial applications is security. Protection against viruses, preventing mobile agents from entering endless loops and consuming all available resources, authentication, and privacy are just a few of the implementation challenges to be addressed [16].

The driving force motivating mobile agent-based computation is threefold. First, mobile agent provides an efficient, asynchronous method for searching for information or services in rapidly evolving networks, i.e. mobile client is launched into the unstructured network and roams around to gather information. Second, mobile agent can tolerate intermittent connectivity and slow networks. This second property makes the use of mobile agent in mobile computing very attractive [20, 21]. Third, the agent computational model supports disconnected operation. During a brief connection service, a mobile client submits an agent and then disconnects. The agent proceeds independently to accomplish the delegated task. When the task is completed, the agent waits till reconnection is established to return the result to the mobile client. Weak connectivity is also supported by this model since the overall communication traffic through the wireless link is reduced from a possibly large number of messages to the submission of a single agent and then of its result. In addition, by letting mobile clients submit agents, the burden of computation is shifted from the resource-poor mobile clients to the more powerful server over the fixed network. Mobility is inherent in the model. Mobile agent migrates not only to find the required resources but also to follow mobile clients. Finally, mobile agent provides the

flexibility to load functionality to and from a mobile client depending on bandwidth and other available resources.

2.2 Markup Languages

Most documents on the web are stored and transmitted in *HyperText Markup Language* (HTML) nowadays. HTML is a simple language well suited for hypertext, multimedia and the display of small and reasonably simple documents. Actually, HTML is based on *Standard Generalized Markup Language* (SGML) [25], which is a language for logical document structure and is an international standard for publishing. SGML is based on the principles of the generic encoding of documents. SGML allows documents to describe its logical structure and not its physical presentation.

We all know that HTML is lack of extensibility, structure, and validation in contrast to SGML. HTML does not allow users to specify their own tags or attributes in order to parameterize or semantically qualify their data. HTML does not support the specification of deep structures needed to represent database schemas or object-oriented hierarchies. HTML does not support the kind of language specification that allows applications to check data for structural validity on importation. On the other hand, a generic SGML is one that supports the arbitrary complexity and makes possible the qualities of extensibility, structure, and validation missing from HTML. However, full SGML contains many optional features that are not needed for web applications.

In this case, the World Wide Web Consortium (W3C) has created an SGML Working Group to build a set of specifications to make it easy and straightforward to use the beneficial features of SGML on the web [36, 59]. The goal of the W3C SGML activity is to enable the delivery of self-describing data structures and arbitrary depth and complexity to applications that require such structures. Therefore, a simplified subset of SGML specially designed for web applications, *eXtensible Markup Language* (XML), is

introduced in 1997.

HTML	*XML*
Known tag set	No fixed tag set
Tags can be omitted	Tags can't be omitted
Element and attribute names are case-insensitive	Element and attribute names are case-sensitive
Many documents only use the Western European character set	Unicode is mandated
Some features of SGML are not generally supported	Many features of SGML are supported (parsed and unparsed entities, CDATA section , and so on)

Table 2.1: Some differences between HTML and XML

Although HTML and XML are similar in many ways, due to their common SGML heritage, they are also different. Table 2.1 shows some of the differences. HTML has a known tag set, so HTML author knows that <IMG> always indicates an image. XML is a *meta-language* that can be used to create many tag sets; thus <IMG> could mean anything, and an image could be represented by any element. On the other hand, XML has features that HTML documents typically don't have. For example, XML documents often contain parsed entities, a way of including text or markup that is repeated. Parsed entities are often used for XML data that can be validated by XML parser. Unparsed entities must specify its file type by means of notation, since those files cannot be represented by simple text formats, such as image, sound and video file.

There are several standards being developed within the XML framework including the description of data, the validation of data, the linkage of data, the formatting of data, the transformation of data, and the presentation of data. In the description of data, XML is a meta-markup language that allows user to create their own set of tags for documents. Each set of tags describes the semantics of the data that they are enclosed. In the validation of data, *Document Type Definition* (DTD) is an optional, but powerful feature of XML that provides a formal set of rules to define a document structure. In the linkage of data, XPointer [37] and Xlink [38] extend HTML linking mechanism by providing external specifications of locations, multiple links, external links. In the formatting of data, *eXtensible Stylesheet Language* (XSL) [1] is a language for expressing a stylesheet.

The stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary. In the transformation of data, *XSL Transformations* (XSLT) [11], which is a language for transforming XML documents into other XML documents, is designed for use as part of XSL. In the presentation of data, *eXtensible HyperText Markup Language* (XHTML) provides the basis for a family of document types that will extend HTML, in order to support a wide range of new devices and applications, by defining modules and specifying a mechanism for combining these modules. This mechanism will enable the extension of XHTML in a uniform way through the definition of new modules. As the use of XHTML moves from the traditional desktop user agents to other platforms, it is clear that not all of the XHTML elements will be required on all platforms. For example a hand-held device or a cell-phone may only support a subset of XHTML elements. In addition, XHTML is compatible with the conventional HTML browsers.

Chapter 3

Related Work

3.1 Mobile Web Browsing

A typical web page today contains a *HyperText Mark-up Language* (HTML) [48] document, and many embedded images. Twenty or more embedded images are quite common. Each of these images is an independent object on the Internet, retrieved or validated for change separately [2]. The common behavior for the web browsing mechanism is to fetch the base HTML document, and then immediately fetch the embedded objects that are typically located on the same web server. It is an inefficient design of the HTTP/1.0 which creates a separate TCP connection for each transfer and increases the overhead and the latency [46, 57]. Actually, the newly released HTTP/1.1 [17] standard is designed to address this problem by encouraging multiple transfers of objects over one connection. The web server will also compress HTML files[1] for transporting across the Internet, providing a substantial aggregate savings in the amount of data that has to be transmitted. Therefore the persistent connection and the compressed HTML can avoid these inefficiencies and improve the network performance [34].

In the Client-Proxy-Server model, Mowser [27], TeleWeb [54], and WebExpress [3] have performed active transcoding of HTTP request from the client while sending it to the server, according to the preference set by the mobile host, so that the document in

[1] Image files are already in a compressed format so this improvement applies only to text.

the most suitable format is retrieved. It also processes the received HTTP data before sending the document to the mobile host if necessary.

Although transformed data broadcasting mechanism [33] and FEC-based reliable multicast protocol [51] have addressed the utilization and the fault tolerant protocol of broadcast channel in the mobile environment, it seems that it is less beneficial when applied on web surfing. It is because the greatest benefit of using broadcasting channel comes only when there is enough shared data with many mobile clients. However, mobile clients seldom browse the same web document at the same time. Therefore, we need to find out a protocol that is suitable for web browsing and utilizing the bandwidth effectively in the mobile environment.

Actually, there is a protocol applied on web browsing in the mobile environment. *Wireless Application Protocol* (WAP) [61] is focused on enabling the interconnection of wireless devices, such as cellular telephones and radio transceivers, that can be used for web access. The *Wireless Markup Language* (WML) and WMLScript do not assume that a keyboard or a mouse is available for user input, and are designed for small screen displays [61]. Unlike the flat structure of HTML documents, WML documents are divided into a set of well-defined units of user interactions. Each unit of interaction is called a card that are created by letting the user navigate back and forth between cards from one or several WML documents. WML provides a smaller, telephony aware, set of markup tags that makes it more appropriate than HTML to implement within wireless devices.

3.2 eXtensible Mark-up Language

Although HTML begins in simple format, it has grown more and more complicated in the newest version. Developing browser has been made more complicated by the forgiving the nature of existing browser, which has allowed users to omit end tags, include incredibly malformed code, and nested tags in strange combinations. It is restricted to further development on extensibility, structure and validation [4]. Most of the HTML files are

stored in a flat file, they are hard to separate into different sections dynamically except those are generated by *Common Gateway Interface* (CGI) [44].

To overcome the limitations in the HTML, *eXtensible Mark-up Language* (XML) [6] a subset of the ISO standard: *Standard Generalized Mark-up Language* (SGML) [36, 59] introduces a more flexible mechanism on processing web documents without SGML's complexity. XML is up to the task of being the universal data interchange format because it is simple to read and self-describing. XML is fully portable because it is ASCII and can be transferred using standard protocols such as HTTP. Any application can interpret XML by using a simple parser and derive information by navigating the document tree. Each field and attribute is delimited with an identifying tag, so every data element comes with identifying meta-data. The nesting of the fields impose structure on the elements, so the relationship between data elements can be inferred.

In addition, the XML data can further describe itself and define a validation scheme by including a *Document Type Definition* (DTD). The DTD defines the structure of the data object up front so that applications can ascertain what attributes are included and what fields are optional. This enables application programs to interpret data on the fly and respond dynamically.

In [5], it describes a lightweight, flexible and reusable utility for transferring data between XML documents and relational database. By viewing and XML document as a tree of objects, we are able to exploit and expand object-relational mappings and schema generation to build a utility that not only transfers data between documents and databases of know schema, but also generates both XLM DTDs and relational schemas at runtime for on the fly loading and extraction of data. Several middle-ware products are available for transferring data between relational databases and XML documents. These fall into two broad categories: template driven and model-driven. In template-driven products, like XSQL Servlet [43] and XML Servlet [7], commands are embedded in a template that is processed by the middle-ware. For example, a SELECT statement might be replaced

by its results, formatted as XML. Template-driven middle-ware is extremely flexible and often includes programming structures such as loops and if statements. Model-driven products, such as XML SQL Utility [42], define a data model for the XML document and then explicitly or implicitly map this to the database.

In the presentation model for the new generation of web technologies, the formatting of a document is conducted through the use of style sheet [10]. The style sheet is a separate document, which allows authors and users to attach style (i.e. fonts, size and spacing) to structure document. For example, if a client requests a structured document from a server, the server can interpret which browser the client adopts by the HTTP's header of the request. Then the server can generate an XHTML document based on the corresponding XML and XSL documents.

3.3 Mobile Agent Technology

In a conventional approach, the *Remote Procedure Call (RPC)* mechanism is often used to communicate between client and server. A client calls a procedure, which is implemented on the server side and the server responds to the client with the result. Actually, Java RMI [58], DCOM [9], and CORBA [45] adopt this mechanism on the Internet. However, the client has to know what procedures that are provided by the server in advance. The server has the duty to accomplish the work with a series of remote procedure calls. Since the communication cost is expensive and the mobile device suffers from limited power budget, it seems that it is not a good communication mechanism applying on the mobile web environment.

Therefore, [30] introduces the mobile agent model which enables a client not only to call procedure in the server, but also to supply the procedure to execute. Harrison [23] has stated three aspects, efficiency, persistence, and fault-tolerant, in the mobile agents.

- *Efficiency.* If the mobile agent can move across networks to location where resources reside, the network traffic can be reduced since the mobile agent can pre-process

data and decide which is the most important information to transfer. This is an important aspect when considering users who connect through the low bandwidth channel, e.g. wireless network.

- *Persistence.* Once the mobile agent is launched, it should not be responsible for the system that launches it and should not be affected if that node fails. The concept of the mobile agent migrating between network nodes gives it the ability to survive and to reach out for as many resources as possible. This is useful for the mobile client due to the fact that it can log on, launch the mobile agent, log off and check the progress or receive the result later on.

- *Fault-tolerant.* In a client-server relationship, the state of the process is generally spread over the client and the server. In the event of network or server failure during a request, it is difficult for the mobile client to resume the process and retransmit the request to the server because the network connection had been broken. However, since the mobile agent does not need to maintain permanent connection and its state is centralized within itself, the mobile agent can send the results when the mobile client reconnects to the server.

In general, there are two types of agents, *static agent* and *mobile agent.* A static agent belongs to a *residential agent.* All the residential agents process and stay in a *Domain* which is the execution environment. A mobile agent belongs to a *visiting agent.* All the visiting agents execute in the *Place* where is their execution environment [41].

Many research projects deal with the implementation of a general purpose mobile agent system, e.g. IBM Aglets [29], and ObjectSpace Voyager [60] whose framework are programmed with Java language. However, ObjectSpace Voyager also supports CORBA architecture. In principle, Java RMI, DCOM and CORBA have similar capabilities. However, Java RMI is restricted to inter-communication with Java RMI objects. It is not designed to inter-operate with other ORBs or languages. CORBA and DCOM

support the construction and the integration of client-server applications in heterogeneous distributed environment. CORBA is more flexible and more interoperable than DCOM, since DCOM is designed to support procedural programming and homogeneous platform, while CORBA is designed to support object-oriented programming and heterogeneous platforms [21]. In [28], there is a discussion about system-level issues and language-level requirements that arise in the design of mobile agent systems. In [22], performance on the mobile agent paradigm is evaluated in comparison to the client/server paradigm. This evaluation has been conducted on top of the Java environment, using respectively RMI, the Aglets mobile agents platform.

On the other hand, Fűnfrocken [15] has proposed a method to integrate a mobile agent into a web server. This is not a good idea because of overloading on web server that processes security checking and redirects tasks for each entry of mobile agents. In our point of view, we maintain web server to provide only web information rather than supporting additional functions. Therefore, we propose a DAE which is dedicated for mobile agent execution [62]. It is described in details in Section 4.3.

3.4 Fault-tolerant Transmission Scheme

Although the mobile agent model can tolerate disconnection, once the mobile agent is successfully migrated to perform the tasks in the base station, the results of the task will not be able to be recovered from noisy transmission channels. Both transformed data broadcasting mechanism [33] and FEC-based reliable multicast protocol [51] have been proposed to transmit data to a collection of clients, while addressing the utilization and fault-tolerant issues. In addition, both schemes seem to be less oriented towards wireless web surfing. This is because the greatest benefit from utilizing broadcast channels pays off only when there is sufficient amount of information to be shared among a large number of mobile clients. However, mobile clients seldom browse the same web document at the same time. We thus need to consider the fault-tolerant issue in a point-to-point context.

We proposed the *fault-tolerant multi-resolution transmission* [32] scheme which allows information units with high information content in a web document to be recovered from transmission error. A mobile client can then obtain an overview content of a web document in a short moment, and can terminate the transmission of the remaining document. If the mobile user is not interested in the web document, it should not wait until the document is reconstructed from redundant packets or upon the retransmission of corrupted portions. In this scheme, it can practically remove the need for acknowledgments while still allowing for reliable communications.

In addition, several software packages have been made publicly available which provide packet-level FEC encoding/decoding functionality [49, 52]. In [52], the researcher have implemented and increased the interface flexibility of an existing *Vandermonde-Matrix based Forward Error Correcting* (FEC) coding package [50] in C-language to Java version.

Chapter 4

Multi-resolution Transmission and Browsing Mechanism

Multi-resolution transmission and browsing mechanism allows a document to be transmitted and browsed at a coarse resolution, with the details to be filled in progressively, in the order of importance. This has a similar flavor as JPEG compression for images, where significant coefficients are transmitted earlier and insignificant ones are transmitted later, or are even dropped without much impact to the quality of image. The multi-resolution transmission of a document allows a user to decide early if the document currently being explored is of any interest and this could minimize the usage of scarce wireless bandwidth with early termination of the transmission of irrelevant documents. In Section 4.1, we describe the details of the multi-resolution transmission and browsing mechanism.

One limitation of the multi-resolution transmission paradigm is its lack of resilience to faulty transmission. To enhance the reliability of delivering the documents over the mobile environment, we extend the multi-resolution transmission paradigm with a fault-tolerant transmission capability so that a mobile client could recover the packets sent over the unreliable network due to data corruption. We term this mechanism the fault-tolerant multi-resolution transmission mechanism in Section 4.2.

Finally, we propose a *Distributed Agent Environment* (DAE) in Section 4.3 that is designed to provide a robust runtime environment for mobile agents. In other words,

mobile agents have the ability to orchestrate tasks and store results in the base station whenever disconnection occurs.

4.1 Multi-resolution Transmission and Browsing Mechanism

We propose a multi-resolution transmission and browsing mechanism by including *Level of Detail* (LOD) in the request message to achieve multi-resolution transmission in the mobile environment. We observe that many structured documents, e.g. research papers, technical reports, specifications, and news articles, are published on Internet. When we browse those documents, we are usually waiting for a long downloading time because of the large document size and sequential transmission in traditional web browsing. In multi-resolution web browsing, we assume that the structured document can be organized as five levels of LOD which are **Document**, **Section**, **Subsection**, **Subsubsection** and **Paragraph**. These five levels form a total order: **Paragraph** $\lhd$ **Subsubsection** $\lhd$ **Subsection** $\lhd$ **Section** $\lhd$ **Document**. In other words, the **Document** level is composed with elements at the **Section** level, **Section** level element is composed with **Subsection** level elements and soon. The highest level of detail is **Document** level and the lowest level of detail is **Paragraph** level. For example, the mobile client defines the LOD in **Document** level, the web document download sequence is no different from that in the conventional web browsing mechanism, since there is only one unit for transmission. However, if the mobile client defines the LOD in other levels, the transmission sequence of sections in the document could be ordered by the different types of information content of each section [31].

The semantic description for each document is called the *structural characteristic* (SC). It illustrates the structural organization of a document and defines the order of document transmission at the required LOD. For example, the SC of paper [56] is shown in Figure 4.1, **Document** node contains seven children nodes that are from Section 0

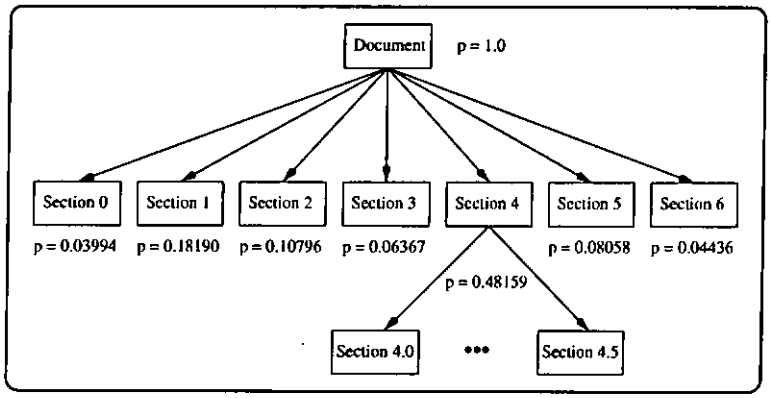

Figure 4.1: A structural characteristic of a document

to Section 6. Furthermore, Section 4 contains six subsection nodes from Section 4.0 to Section 4.5. Each node, n, is associated with a content level, p, which indicates the information content captured within the organizational unit. The content level associated with the **Document** node is 1, since the document contains whole information of the document. The content level associated with Section 0 is 0.03994, Section 1 is 0.18190, Section 2 is 0.10796, Section 3 is 0.06367, Section 4 is 0.48159, Section 5 is 0.08058 and Section 6 is 0.04436.

Therefore, it indicates that Section 4 has captured more information content compared with other sections. If the mobile client defines LOD in **Section** level, the transmission sequence is ⟨ **Section 4, Section 1, Section 2, Section 5, Section 3, Section 6, Section 0** ⟩. A desirable property of information content is that the information contents of the organizational units follow the additive rule, i.e., information contents of sub-units of an organizational unit add up to that of the unit. In other words, for a given node, n_i, with children nodes, $\{n_{i,1}, ..., n_{i,m}\}$, $p_i = \sum_{j=1}^{m} p_{i,j}$. Note that the content levels of all leaf nodes, i.e., those modeling paragraphs of a document, add up to 1.

In Figure 4.2, to generate the SC for a XML document, the document is pre-processed by the Document Analyzer. The pre-processing procedure creates and stores a keyword-based document tree that describes the organization and information content of the document in the Database Server. The transmission sequence of the document is also stored

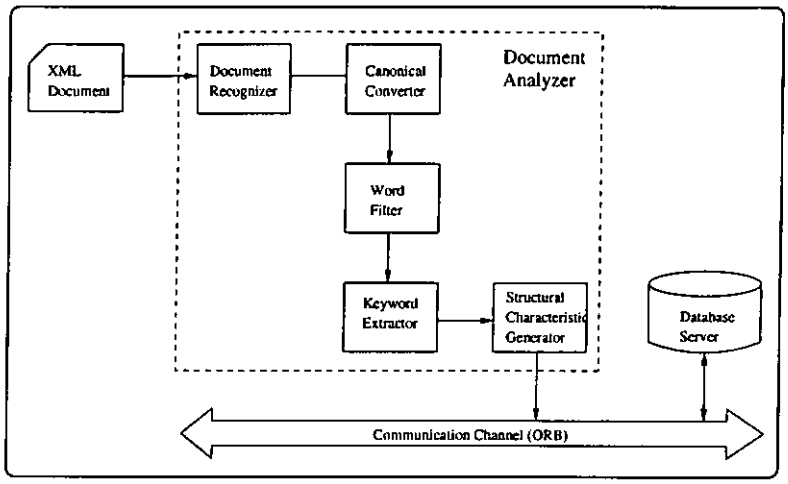

Figure 4.2: The steps for document analyzer

into the *Transmission Ordering Table* (TOT). We employ ORB for communicating with the database server, providing interoperability with other databases. In addition, the SC of the document can be generated by Document Analyzer that derives the content level of each organizational unit from the corresponding document's keyword-based document tree. Figure 4.2 depicts the pre-processing procedure. The pre-processing procedure is composed of five modules: *document recognizer*, *canonical converter*, *word filter*, *keyword extractor*, and *structural characteristic generator*, operating in a pipelined fashion.

4.1.1 Document Recognizer

The document recognizer is a lexical analyzer, which converts an input stream of characters into a stream of words. We adopt an XML processor that parses the mark-up and validates XML document as the document recognizer. In Figure 4.4, the DTD defines the elements that may be used, and dictates where they may be applied in relation to each other. Therefore, the document structure rules, document hierarchy and granularity can be established easily.

Document recognizer performs a single pass on each document. During the pass, document recognizer creates a tree-like structure, called *document tree*. Figure 4.3 and Figure 4.4 depict a sample XML document and the corresponding DTD respectively.

```
1  <?XML version="1.0" encoding="UTF-8" ?>
2  <document id="sac98.xml" title="Maintaining Page Coherence for Dynamic HTML Pages" LoD="0">
3     <section id="0" title="Abstract" LoD="1">
4        <subsection id="0.0" title="" LoD="2">
5           <subsubsection id="0.0.0" title="" LoD="3 ">
6              <para id="0.0.0.0" LoD="4">
7              Fueled largely by the gaining popularity of World Wide Web (web) servers and browsers, ...
8              </para>
9           </subsubsection">
10          .
11          .
12       </subsection">
13       .
14       .
15    </section>
16    .
17    .
18    <section id="4" title="The Design and Implementation of Page Coherence Mechanism" LoD="1">
19       <subsection id="4.0" title="" LoD="2">
20          <subsubsection id="4.0.0" title="" LoD="3 ">
21             <para id="4.0.0.0" LoD="4">
22             We borrow the idea proposed in the Leases file caching mechanism ...
23             </para>
24             <para id="4.0.0.1" LoD="4">
25             When a query is initiated to generate a dynamic page, ...
26             </para>
27             .
28             .
29          </subsubsection">
30       <subsection id="4.1" title="" LoD="2">
31          <subsubsection id="4.1.0" title="" LoD="3 ">
32             <para id="4.1.0.0" LoD="4">
33             .
34             .
35             </para>
36          </subsubsection">
37          .
38          .
39       </subsection">
40    </section>
41    .
42    .
43 </document>
```

Figure 4.3: The sac98.xml

In Figure 4.3, there is the excerpt from the paper [56] which is written in XML, the document begins with a processing instruction: <?XML ...?>. This is the XML declaration whose presence explicitly identifies the document as an XML document and indicates the version of XML. According to the XML specification, the name appearing in the end-tag must also match exactly with the name that appears in the start-tag. Each element must be completely enclosed by another element, except for ancestor of all

```
1 <!- XML document DTD
2 DTD for short SAC98.xml
3 AUTHOR: Stanley yau (csmtyau@comp.polyu.edu.hk)
4 VERSION: 1.0 (1 May, 2000) ->
5
6 <!- define internal entities ->
7 <!ENTITY % Block "(para | graphic | list | table)*" >
8 <!ENTITY % Hilite "markup | emph-strong | emph-italic" >
9
10 <!- MAIN STRUCTURE ->
11 <!ELEMENT document      (section*) >
12 <!ATTLIST document      id CDATA #REQUIRED>
13 <!ATTLIST document      title CDATA #REQUIRED>
14 <!ATTLIST document      LoD CDATA #REQUIRED>
15
16 <!ELEMENT section       (subsection*) >
17 <!ATTLIST section       id CDATA #REQUIRED>
18 <!ATTLIST section       title CDATA #REQUIRED>
19 <!ATTLIST section       LoD CDATA #REQUIRED>
20
21 <!ELEMENT subsection    (subsubsection*) >
22 <!ATTLIST subsection    id CDATA #REQUIRED>
23 <!ATTLIST subsection    title CDATA #REQUIRED>
24 <!ATTLIST subsection    LoD CDATA #REQUIRED>
25
26 <!ELEMENT subsubsection (para*) >
27 <!ATTLIST subsubsection id CDATA #REQUIRED>
28 <!ATTLIST subsubsection title CDATA #REQUIRED>
29 <!ATTLIST subsubsection LoD CDATA #REQUIRED>
30
31 <!- BLOCK STRUCTURES ->
32 <!ELEMENT para          (#PCDATA | %Hilite; — x-ref)*>
33 <!ATTLIST para          id CDATA #REQUIRED>
34 <!ATTLIST para          LoD CDATA #REQUIRED>
35
36 <!ELEMENT graphic       EMPTY>
38 <!ATTLIST graphic       id ID #IMPLIED
38                         id ENTITY #REQUIRED>
39
40 <!ELEMENT list          (item+)>
41 <!ELEMENT item          (#PCDATA | %Hilite; | x-ref)*>
42 <!ATTLIST item          type (number | random) "random">
43
44 <!- cross-references to other text in the document ->
45 <!ELEMENT x-ref          (#PCDATA)>
46 <!ATTLIST x-ref          source IDREF #REQUIRED>
```

Figure 4.4: The sac98.dtd

other elements, i.e. the root of document which is the "document" element.

In the validating process, the validation engine would compare the DTD against a complete document instance, and produce a report containing errors or warning, if any.

4.1.2 Canonical Converter

After parsing the XML document into keyword-based document tree, we need to select and calculate the weight of keywords in the document. We start by treating all words in the keyword-based document tree as keyword candidates and put them into candidate list for qualification. We know that not every keyword candidate in the candidate list can be qualified to a keyword finally. We use canonical converter to reduce the variant of keyword candidates in candidate list by stemming all words from any tense to their present infinitive form, and all suffixes such as adjectives and adverbs. The first step is to stem all keyword candidates into their general term. Our canonical converter is implemented by Porter's algorithm [18] which is categorized by affix removal stemmers. It is an iterative longest match stemmer, which removes the longest possible string of characters from a word according to a set of rules. For example, "engineering", "engineered", and "engineer" can be stemmed into "engineer" for keyword candidates.

4.1.3 Word Filter

It has been recognized since the earliest days of information retrieval that many of the most frequently occurring words in English like "the", "of", "and", "to", etc..., are worthless as keyword candidates when calculating information content of the organizational units. They are referred to as "close words". All pronouns, prepositions, conjunctions and interjections are classified into *close words*. The other words including nouns, verbs, adjectives, and adverbs are classified as *open words* which may contain useful information.

Word filter use a *stoplist* dictionary, which included 425 words, derived from the Brown corpus of 1,014,000 words drawn from a broad range of literature in English [18]. When stemmed keyword candidates are passed into word filter, all keyword candidates belonging to close words are removed from candidate list. To speed up the filtering process, we hash all words in the stoplist dictionary into main memory as a close-words cache to minimize the lookup overhead from the storage.

4.1.4 Keyword Extractor

Before this stage, all keyword candidates are conflated to present infinitive form and all close words have been removed. We then pass all keyword candidates into keyword extractor for selecting keywords finally. We know that not all keyword candidates are qualified as keywords for a document because different words might bear different degree of information content with respect to the document. For each keyword candidate, a, we count its number of occurrences within the document. The mean occurrences, μ, of all keyword candidates and the corresponding standard deviation, σ, are computed. Each keyword candidate, a, will be qualified as a descriptive keyword for the document if its number of occurrences falls within the threshold interval $[\mu - \alpha\sigma, \mu + \beta\sigma]$ where α and β are two adjustable user-defined parameters. By adjusting these two parameters, the keyword extraction process is flexible enough to adapt to different collections of documents. For example, keyword candidates only occurring once in the document will not be considered as keyword in a document. This could be accomplished by using a moderate value of α to increase the lower bound of the threshold.

In addition to the frequency distribution analysis, we also employ some heuristics in keyword extractor process. We observe that some emphasized words bear some special meanings within a document, such as newly defined terms. The keyword extractor explicitly looks at these kinds of tags with keyword attribute defined as true; for instance the title of the document for each section, subsection, and subsubsection are selected as keywords. This is based on the observation that the title of a document is usually a representative description of the document. Therefore, those keywords would be regarded as keywords even if their number of occurrences falls beyond the threshold interval.

4.1.5 Structural Characteristic Generator

Our multi-resolution browsing mechanism is based on the observation that the significance of every organizational unit does not follow an even distribution. There should be

different degrees of contribution by different organizational units in a document. The design of a preliminary browsing system based on the concept of multiple granularities has been described as a simple notion of information content and relative information content. We know that, SC can be refined based on the user query, so we derive the query-based information content for better SC generation process to define a more semantics-oriented ranking in the document.

Information Content

The set of keywords in a document, D, will be used to define the significance of an organizational unit. Since each keyword may appear in different organizational units with different frequencies, it is unfair to treat them alike. We observe that the keywords which occur more frequently than others should carry more information in the whole document. However, each occurrence of those high frequency keywords must be less significant. We expect this decay in importance to be exponential rather than linear. A weight is associated with each keyword that determines its relative importance in a document, as a logarithmic function of the other keyword occurrence.

For notational convenience, we denote the number of occurrences of a keyword, a, in a document, D, by $|a_D|$ and the number of occurrences of a in an organizational unit, n_i, by $|a_{n_i}|$. The occurrence vector, V_D, of the set of keywords in D, $A_D = \{\ a \mid a$ is a keyword in $D\ \}$, can be represented as $V_D = \{\ |a_D| \mid a \in A_D\ \} = \{\ v_1, v_2, ..., v_{|A_D|}\ \}$. The weight of each occurrence of a keyword, a, for a document, D is denoted as ω_a. It is defined as $\omega_a = 1 - \log_2(|a_D|/||V_D||)$ where $||V_D||$ is the norm of the occurrence vector V_D. We choose the infinity norm $||V_D||_\infty = max(v_i)$. Using infinity norm, we observe that in a special case where all keywords occur only once, the norm of the occurrence vector will be 1. The logarithm will be zero and thus ω_a for any keyword a, will be 1. The information content, p_i, of an organizational unit, n_i, is now defined to be the weighted sum of the keywords in the organizational unit, normalized with respect to the

document, D:

$$p_i = \frac{\sum_{\forall \text{keyword } a \in n_i} |a_{n_i}| \omega_a}{\sum_{\forall \text{keyword } d \in D} |d_D| \omega_d}$$

We observe that the additive rule for information contents of sub-units will hold so that the total information content for the document D adds up to unity. This rule holds because for an organizational unit, n_j, with m sub-units, $n_{j,1}, n_{j,2}, ..., n_{j,m}$, its information content, p_j, will be:

$$p_j = \frac{\sum_{\forall \text{keyword } a \in n_j} |a_{n_j}| \omega_a}{\sum_{\forall \text{keyword } d \in D} |d_D| \omega_d}$$

$$= \frac{\sum_{k=1}^{m} \sum_{\forall \text{keyword } a \in n_{j,k}} |a_{n_{j,k}}| \omega_a}{\sum_{\forall \text{keyword } d \in D} |d_D| \omega_d}$$

$$= \sum_{k=1}^{m} \frac{\sum_{\forall \text{keyword } a \in n_{j,k}} |a_{n_{j,k}}| \omega_a}{\sum_{\forall \text{keyword } d \in D} |d_D| \omega_d}$$

$$= \sum_{k=1}^{m} p_{j,k}$$

Relative Information Content

In order to better quantify the amount of information transmitted over the scarce wireless network, we defined the notion of *Relative Information Content* (RIC), γ_i, that enables us to determine the optimal transmission order for organizational units at the required LOD such that a client could perceive more information with the shortest possible duration [63]. The RIC of organizational unit n_i, is defined as the ratio of its information content to its content length: $\gamma_i = p_i / l_i$. Unfortunately, the additive rule for relative information content does not hold.

Query-based Information Content

In web document browsing, usually the set of documents is returned as the result of a search process from a search engine, using a kind of *vector space model* for identifying relevant documents, perhaps filtered and adjusted with user profiles and/or relevance feedback. We now consider the definition of significance of an organizational unit which is returned as the result of a search process. The significance of an organizational unit changes dynamically due to the presence of the extra information in the querying words.

We now denote the query Q by a vector, analogous to the document D. The query Q contains a set of keywords, which we call the querying words, $A_Q = \{\ a \mid a$ is a keyword in $Q\ \}$. Normally, all words in the query Q which are not close words should be considered keywords and the number of occurrences of a in Q is $|a_Q| = 1$. This forms an occurrence vector V_Q. Sometimes, a user might want to emphasize a particular keyword by repeating the keyword several times during a search process. We should thus take into account of multiple occurrences of a querying word in defining its weight, so as to be symmetrical to the processing of the document. The weight of a querying word a is denoted as $w_a^Q = 1 - \log_2(|a_Q|/||V_Q||)$. Using an infinity norm will yield the nice property that all weights will be 1, since this is the common case when a query Q is issued by a user to search for web documents using a search engine. The *Query-based Information Content* (QIC), q_i^Q, of an organizational unit, n_i, with respect to query Q is now defined to be the weighted sum of the keywords in the organizational unit, and including those of the querying words, normalized with respect to the document, D, and query, Q:

$$q_i^Q = \frac{\sum_{\forall \text{keyword } a \in n_i} |a_{n_i}| \omega_a . \omega_a^Q}{\sum_{\forall \text{keyword } d \in D} |d_D| \omega_d . \omega_d^Q}$$

Note that the *additive rule* still holds for QIC

$$q_j^Q = \frac{\sum_{\forall \text{keyword } a \in n_j} |a_{n_j}| (\omega_a \omega_a^Q)}{\sum_{\forall \text{keyword } d \in D} |d_D| (\omega_d \omega_d^Q)}$$

$$= \frac{\sum_{\forall \text{keyword } a \in n_j \cap Q} |a_{n_j}| (\omega_a \omega_a^Q)}{\sum_{\forall \text{keyword } d \in D \cap Q} |d_D| (\omega_d \omega_d^Q)}$$

$$= \frac{\sum_{k=1}^{m} \sum_{\forall \text{keyword } a \in n_{j,k} \cap Q} |a_{n_{j,k}}|(\omega_a \omega_a^Q)}{\sum_{\forall \text{keyword } d \in D \cap Q} |d_D|(\omega_d \omega_d^Q)}$$

$$= \sum_{k=1}^{m} \frac{\sum_{\forall \text{keyword } a \in n_{j,k} \cap Q} |a_{n_{j,k}}|(\omega_a \omega_a^Q)}{\sum_{\forall \text{keyword } d \in D \cap Q} |d_D|(\omega_d \omega_d^Q)}$$

$$= \sum_{k=1}^{m} q_{j,k}^Q$$

Modified Query-based Information Content

Notice that the QIC of each organizational unit is determined every time the search engine receives a query from a user. Since the weights of keywords, ω_a, of a document, D, remain unchanged across queries, only the contribution by querying words is small, the number of terms involved in the computation is small and thus, the computational overhead of QIC is quite low. One might object against QIC based on the fact that QIC of some organizational units may become zero due to the absence of a querying word. We could choose to trade slight computational efficiency for a more general definition of QIC by replacing the product querying word with their sum. However, to ensure that individual weights are in comparable scale, we associate a scaling factor, ρ, with ω_a^Q. This *Modified Query-based Information Content* (MQIC), $\tilde{q}_i^Q$, is defined as:

$$\tilde{q}_i^Q = \frac{\sum_{\forall \text{keyword } a \in n_i} |a_{n_i}|(\omega_a + \rho\omega_a^Q)}{\sum_{\forall \text{keyword } d \in D} |d_D|(\omega_d + \rho\omega_d^Q)}$$

where ρ indicates the degree of deviation a contribution of user's querying words. Thus, we calculate the ρ as the following equation. The higher the value of ρ means that the more keywords in the document, and the more importance of the user's querying keywords.

$$\rho = \frac{\sum_{\forall \text{keyword } n_i \in D} |a_{n_i}|}{\sum_{\forall \text{keyword } a \in Q} |a_Q|}$$

Note that the *additive rule* for modified query-based information contents of sub-units still hold under MQIC:

$$\tilde{q}_j^Q = \frac{\sum_{\forall \text{keyword } a \in n_j} |a_{n_j}|(\omega_a + \rho\omega_a^Q)}{\sum_{\forall \text{keyword } d \in D} |d_D|(\omega_d + \rho\omega_d^Q)}$$

$$= \frac{\sum_{\forall \text{keyword } a \in n_j \cap Q} |a_{n_j}|(\omega_a + \rho\omega_a^Q)}{\sum_{\forall \text{keyword } d \in D \cap Q} |d_D|(\omega_d + \rho\omega_d^Q)}$$

$$= \frac{\sum_{k=1}^{m} \sum_{\forall \text{keyword } a \in n_{j,k} \cap Q} |a_{n_{j,k}}|(\omega_a + \rho\omega_a^Q)}{\sum_{\forall \text{keyword } d \in D \cap Q} |d_D|(\omega_d + \rho\omega_d^Q)}$$

$$= \sum_{k=1}^{m} \frac{\sum_{\forall \text{keyword } a \in n_{j,k} \cap Q} |a_{n_{j,k}}|(\omega_a + \rho\omega_a^Q)}{\sum_{\forall \text{keyword } d \in D \cap Q} |d_D|(\omega_d + \rho\omega_d^Q)}$$

$$= \sum_{k=1}^{m} \tilde{q}_{j,k}^Q$$

4.1.6 An Example of Structural Characteristic Generation

We demonstrate the SC using paper [56] as an example. For simplicity reason, we ignore all equations, figures, and tables in the document. In Table 4.1, there are three columns of SC which are generated by, from left to right, information content generator (Info. Cont. p), relative information content generator(RIC γ), query-based information content generator (QIC q^Q) and modified query-based information content generator (MQIC $\tilde{q}^Q$). Moreover, we specify "Dynamic HTML, Caching" as querying keywords for the two query-based information content generators. We have many sections, subsections, subsubsections and paragraphs in the table. To cater for the fact that some paragraphs do not belong to any subsection, we classify them as a *virtual subsection*. For example, all paragraphs belong to **Section 1, Introduction**, are grouped under the virtual subsection **Section 1.0**. Furthermore, our numbering scheme starts from zero in each content level,

Sect. / Subsect. / Subsubsect. / Para.	Info. Cont. p	RIC γ	QIC q^Q	MQIC $\tilde{q}^Q$
0	0.03994	0.29306	0.09796	0.12669
0.0	0.03994	0.29787	0.09796	0.12669
0.0.0	0.03994	0.30284	0.09796	0.12669
0.0.0.0	0.03994	0.21347	0.04116	0.04658
0.0.0.1	0.03045	0.36201	0.05680	0.08011
1	0.18190	0.26547	0.29449	0.36325
1.0	0.18190	0.26632	0.29449	0.36325
1.0.0	0.18190	0.26718	0.29449	0.36325
...	...	...	...	...
2	0.10796	0.29642	0.12549	0.10696
2.0	0.10796	0.29822	0.12549	0.10696
2.0.0	0.10796	0.30004	0.12549	0.10696
...	...	...	...	...
3	0.06367	0.26398	0.06241	0.06686
3.0	0.06367	0.26641	0.06241	0.06686
3.0.0	0.06367	0.26888	0.06241	0.06686
...	...	...	...	...
4	0.48159	0.29108	0.33360	0.20117
4.0	0.10211	0.29893	0.14422	0.10104
4.0.0	0.10211	0.30087	0.14422	0.10104
4.0.0.0	0.02234	0.24680	0.04617	0.04691
4.0.0.1	0.02604	0.31990	0.05930	0.03663
4.0.0.2	0.01676	0.31805	0.01564	0.00334
4.0.0.3	0.03698	0.33831	0.02311	0.01416
4.1	0.17121	0.28930	0.07629	0.02345
4.1.0	0.17121	0.29039	0.07629	0.02345
4.1.0.0	0.03976	0.29408	0.01248	0.00910
4.1.0.1	0.01206	0.21807	0.00000	0.07733
4.1.0.2	0.05594	0.33658	0.03253	0.01038
4.1.0.3	0.03259	0.25925	0.00939	0.00175
4.1.0.4	0.03086	0.31019	0.02190	0.00140
4.2	0.07573	0.30795	0.01252	0.00385
4.2.0	0.07573	0.31073	0.01252	0.00385
4.2.0.0	0.03157	0.31537	0.01252	0.00168
4.2.0.1	0.01526	0.36586	0.00000	0.00007
4.2.0.2	0.02890	0.29980	0.00000	0.00150
4.3	0.09200	0.32271	0.06065	0.01971
4.3.0	0.09200	0.32522	0.06065	0.01971
4.3.0.0	0.01750	0.29166	0.02190	0.00008
4.3.0.1	0.02753	0.31680	0.00939	0.00140
4.3.0.2	0.04697	0.35995	0.02936	0.01748
4.4	0.02181	0.24049	0.03679	0.04692
4.4.0	0.02181	0.24647	0.03679	0.04692
4.4.0.0	0.02181	0.25275	0.03679	0.04692
4.5	0.01872	0.20419	0.00313	0.00621
4.5.0	0.01872	0.20921	0.00313	0.00621
4.5.0.0	0.01872	0.21448	0.00313	0.00621
5	0.08058	0.19297	0.00622	0.03365
5.0	0.08058	0.19399	0.00622	0.03365
5.0.0	0.08058	0.19502	0.00622	0.03365
...	...	...	...	...
6	0.04436	0.26999	0.07983	0.10143
6.0	0.04436	0.27365	0.07983	0.10143
6.0.0	0.04436	0.27741	0.07983	0.10143
...	...	...	...	...

Table 4.1: The structural characteristic of information content, relative information content, query-based information content and modified query-based information content

such that the content level of **Abstract** in a document belongs to **Section 0**, and its lower level is naming as **Section 0.0**. Moreover, we denote the two paragraphs under **Section 0.0.0** as **Section 0.0.0.0** and **Section 0.0.0.1**.

For instance, at the **Section** LOD, the transmission order based on information content would be ⟨ **Section 4, Section 1, Section 2, Section 5, Section 3, Section 6, Section 0** ⟩. but using the relative information content, the order would be ⟨ **Section 2, Section 0, Section 4, Section 6, Section 1, Section 3, Section 5** ⟩. Thus **Section 0** is detected to be more important and transmitted earlier by using relative information content to determine the transmission order.

If the mobile user changes SC generator to query-based information content, the transmission sequence would be ⟨ **Section 4, Section 1, Section 2, Section 0, Section 6, Section 3, Section 5** ⟩. Using the same querying keywords for modified query-based information content, the transmission sequence would be ⟨ **Section 1, Section 4, Section 0, Section 2, Section 6, Section 3, Section 5** ⟩. As mentioned in Section 4.1.5, using QIC produces some zero value of information content, i.e. **Subsection 4.2.0.1** and **Subsection 4.2.0.2** of QIC column of Table 4.1. Therefore, we have only implemented IC and MQIC generators for our prototype in Chapter 5.

4.2 Fault-tolerant Multi-resolution Transmission Mechanism

The Internet is quite unstable in terms of connectivity. Occasional disconnection during transmission of web information is common. This situation will get worse in the context of a mobile environment, since the wireless communication channel is suffering from a larger degree of signal distortion, interference, and attenuation. Data received will get corrupted very easily and frequently. We would like to enhance the reliability of delivering organizational units by introducing redundancy so that web documents can be received successfully with a much higher probability and without the need to request the base station to retransmit corrupted document portions. Our approach is based on the cryptographic encoding of information [47]. In particalar, we are adopting the *Vandermonde-Matrix based Forward Error Correction* (VMFEC) approach. We assume that the document can be divided into M pieces, each of which is a fundamental unit of transmission over the wireless network. These pieces are called *data packets.* Data packets are received either *intact* (without error) or *corrupted* (with detectable error). A missing packet can be detected when the next packet is received, since the wireless channel is FIFO but unreliable. Simple sequence number as used in the datalink layer transmission protocol suffices in this context. We propose to adopt the cyclic redundancy code (CRC) for the detection of packet corruption, since it has a low computational cost and high error coverage.

A property of the cryptographic transformation technique proposed in [47] is that a file can be divided into M "raw" packets. Via a matrix multiplication procedure, these M packets can be transformed into $N \geq M$ "cooked" packets at the sender side such that if "any" M out of the N cooked packets can be collected at the receiver side, the original file (all the M raw packets) can be reconstructed via another matrix operation based on the polynomial code. Figure 4.5 gives a graphical representation of the encoding and decoding process. The transformation adopted in [47] is cryptographic in nature,

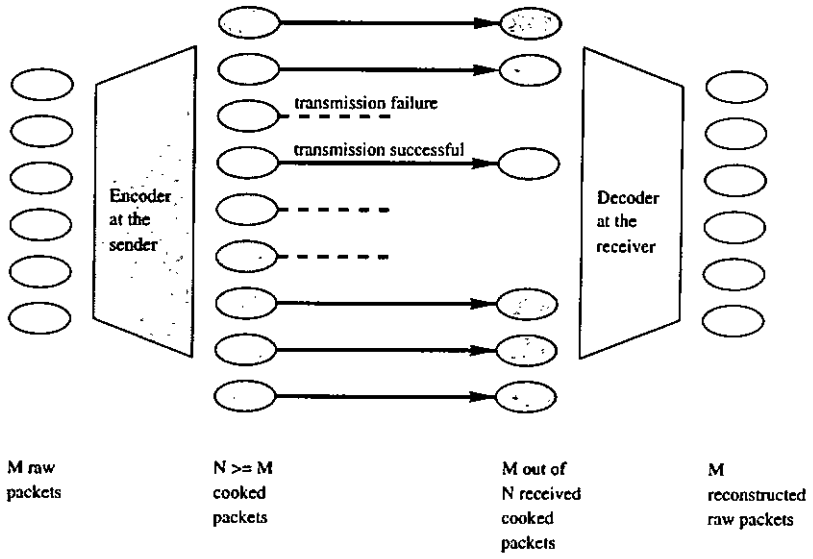

Figure 4.5: A property of the cryptographic transformation technique

such that collecting any M - 1 cooked packets is completely useless in recovering the raw packets. A slight modification is to adopt the Vandermonde polynomial in the transformation stage, followed by making the upper portion of the multiplying Vandermonde matrix into an identity matrix via elementary matrix transformation. This will ensure that first M cooked packets appear in exactly the same form as the raw packets, thus saving recovering effort. Furthermore, it allows a portion of the original information to be used once they are available, without the need to wait until M different cooked packets are collected.

In [32], we have adopted VMFEC encoding/decoding scheme and conducted some simulated experiments on the performance on the fault-tolerant multi-resolution transmission mechanism. We have evaluated the overhead using VMFEC for encoding/decoding and compared with different mobile agent models. The performance study is presented in Section 6.4.1.

4.3 Distributed Agent Environment

Mobile agent is an effective paradigm for distributed applications, and are particularly attractive for mobile computing. Mobile computing devices include physically mobile computers such as laptops and personal digital assistant. All these devices are frequently disconnected from the network for long periods, often have low bandwidth, unreliable connections into the network, and often change their IP address with each reconnection. We propose a *Distributed Agent Environment* (DAE) for the mobile agents travelling anywhere to perform the tasks over the network without being greatly affected by the limitation of the mobile environment.

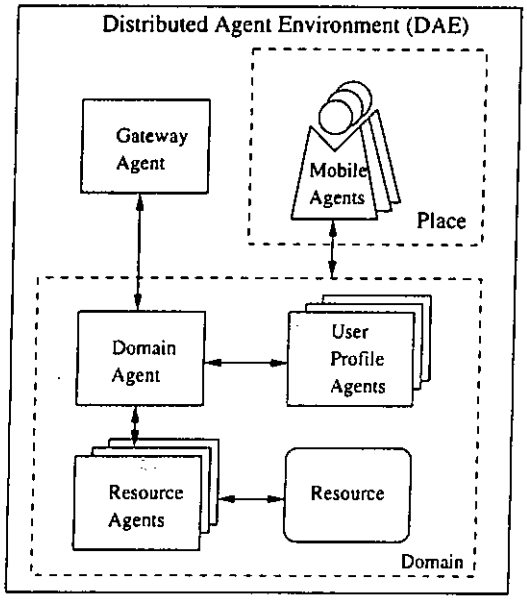

Figure 4.6: A distributed agent environment

DAE is a logical boundary used to divide different agents and resources into manageable and distinct entities [21, 35, 41]. Our DAE architecture is illustrated in Figure 4.6; we introduce two types of agents, static and mobile agents, in DAE. Static agents provide resources and facilities to mobile agents, and mobile agents migrate among DAE by taking advantage of the locality and resources to satisfy their goals. In addition, all mobile agents, which belong to visiting agent, are executed in the place where the execution

environment is located. All static agents, which belong to residential agent, are executed in the domain where is the execution environment.

Gateway Agent, a static agent, provides a firewall function to protect unauthorized mobile agents that fail the security check against accessing the DAE. Domain Agent, a static agent, coordinates the activities occurring in the domain [13]. Domain agent provides a migration service to the mobile agents. If the migration of the mobile agent is rejected, it will be a duty of the domain agent to restart the mobile agent and allow the mobile agent to choose a new destination afterward. Moreover, it acts as a central manager, which registers all information resources provided by the resource agents in the domain. Therefore, it advertises public information resources and provides an inquiry service to the mobile agents from both inside and outside of the DAE to ensure that it contains information resources that will help the mobile agents to achieve their goals. It is a mechanism for the mobile agents to interrogate the contents of a particular DAE before migration. Resource Agents are static agents which exist in the domain to provide and access particular information resources, e.g. databases. In other words, only resource agents understand how to access the resources and permission structures associated with the resources. User Profile Agents provide the mobile agents with a group of user preferences and customize the results returned to the mobile agents. Finally, Mobile Agents can determine where is the best DAE to migrate by enquiring the local domain agent for a list of domains of which it is aware. The mobile agent can use this information to contact each domain agent and determine which one offers a set of information resources which are compatible with its own goal set.

Chapter 5

Implementation and Prototype

In this chapter, we describe the design and implementation of the prototype of *Mobile Web Browser* (MWB) and *Distributed Agent Environment* (DAE). The design described in this chapter aims at modularity and reconfigurability. Hence, we specified the design of the prototype using the object-oriented notation that made it easier to implement in Java. In Section 5.1, we present the architecture of the prototype and describe the design of MWB and DAE with the class diagrams. We give a brief description of the API using in the prototype in Section 5.2. In Section 5.3, we illustrate the system look and feel with a sequence of screen shots about multi-resolution transmission and browsing with the prototype of MWB.

5.1 Design of Prototype

We have implemented a prototype of the MWB and DAE using Java 1.3 and ObjectSpace Voyager Pro 3.3 following the architecture depicted in Figure 5.1. The main feature of the mobile web browser is able to render progressively the organizational units of XHTML document at the appropriate position according to the transmission sequence from DAE. The entire set of web documents are stored in XML file format including DTD and XSL in the Database Server. We employ Oracle database system to maintain the SC of the web documents. The communication between Oracle and the Document Transmitter

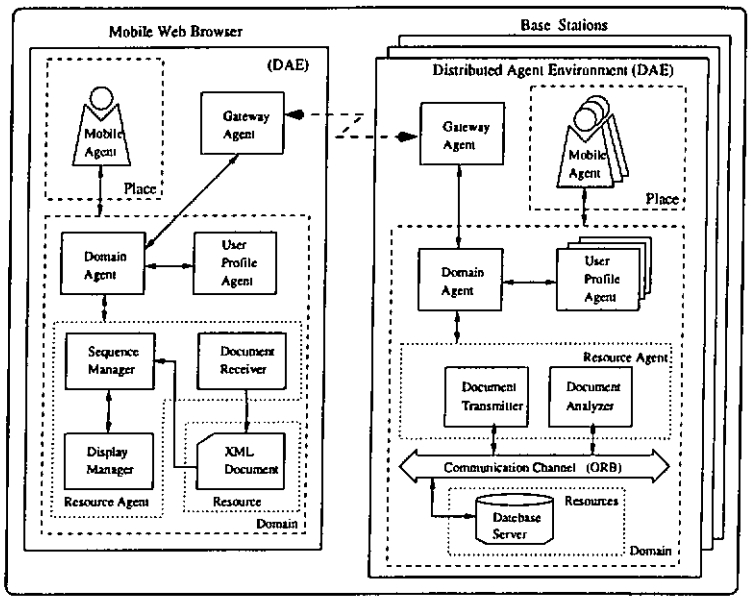

Figure 5.1: The overview of MWB and DAE

and Document Analyzer is via *Object Request Broker* (ORB) and JDBC protocol. It is important that we employ ORB and JDBC for communicating with the Database Server thus providing interoperability with other databases. We adopt LotusXSL 1.0.1 to XML transformations. In our prototype, the SC of XML documents are generated by Document Analyzer that derives the content level of each organizational unit from the corresponding document's keyword-based document tree. The Document Transmitter in the DAE is responsible for transforming XML with XSL into XHTML on the base station and transmitting XHTML documents to the mobile web browser.

5.1.1 Mobile Web Browser

The *Mobile Web Browser* (MWB) is composed of six classes and implemented the class of ActionListener, as depicted in Figure 5.2.

- SplashScreen provides a startup image to illustrate the version of this browser.
- ServerThread is responsible for monitoring and dispatching the BufferThread. Each instance of BufferThread contains a separate thread of control that listens on a

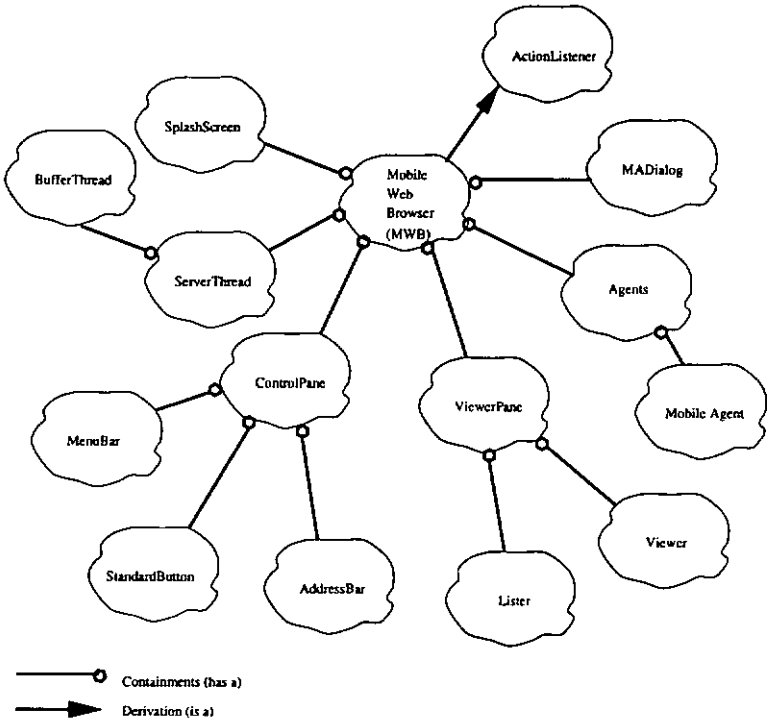

Figure 5.2: A class diagram of MWB

socket port used to collect the data stream from the base station.

- ControlPane is the control component that governs the behavior of all graphical user interfaces including the following:
  - MenuBar is for the menu of the browser.
  - StandardsButton is for the necessary buttons of the browser.
  - AddressBar is for showing the URL of the web document.
- ViewerPane displays information available in the browser. The class of ViewerPane contains two classes Lister and Viewer.
  - Lister is a Sequence Manager to organize and indicate which organizational units of the web document are received in the browser.
  - Viewer is a display manager to render and display the corresponding organizational units of the web document in the browser.

- Agents is devoted to monitoring the available Mobile Agent which is responsible for collecting user's preferences and migrating to the base station to perform the tasks.
- MADialog is a class of dialog box, which accepts the request, LOD and query words, from the mobile user.

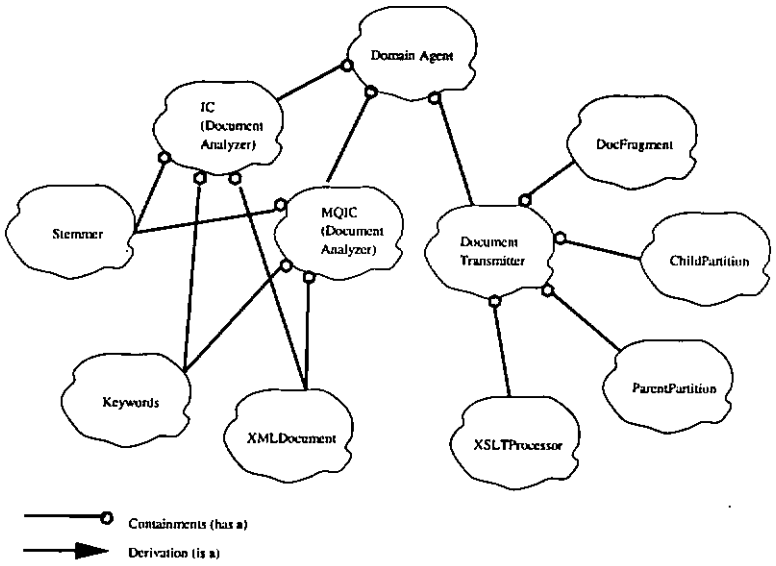

Figure 5.3: A class diagram of DAE

5.1.2 Distributed Agent Environment

The Domain Agent is the central manager in *Distributed Agent Environment* (DAE). The Domain Agent organizes three classes including IC, MQIC and Document Transmitter, as depicted in Figure 5.3.

We classify IC and MQIC as a Document Analyzer. The Document Analyzer is extensible by calling classes for different types of IC.

- IC is responsible for generating the information content of the web document.
- MQIC is responsible for generating the modified query-based information content of the web document.

- Both IC and MQIC contain the classes of Stemmer, Keywords and XMLDocument.
  - Stemmer is a class to stem out all words to their infinitive form.
  - Keywords is a class to filter out all the close words, and preserve the open words.
  - XMLDocument is devoted to parsing, validating and formulating the *Document Object Model* (DOM) of the web documents.
- Document Transmitter contains four classes, which include XSLTProcessor, ParentPartition, ChildPartition and DocFragment.
  - XSLTProcessor transforms a web document to another web document according to the given XSL and XML files.
  - ParentPartition is responsible for partitioning the document hierarchy of the web document.
  - ChildPartition is responsible for partitioning the organizational units of the web document.
  - DocFragment is a utility class to partition the organizational units with certain parameters.

5.2 The Description of the API

In this section, we give a brief description of the API for the classes used in the prototype.

5.2.1 The Agents class

As we discussed in the previous section, the Agents class contains a MobileAgent class. The methods of the Agents are depicted in Figure 5.4. This class collects all the methods that enable a mobile agent to migrate to the base station. The Agents has defined the resources port 7000, the outgoing port 8000, and the incoming port 9000. The *Agents()*

method, as a constructor of Agents, starts a new instance with specified local host IP address. First, the constructor calls the *initialize()* method for creating a repository of the resource classes needed in the local DAE. Second, Agents calls *startUP()* for searching the connectable remote DAEs. Third, Agents calls *MASetup()* for creating the instance of MoblileAgent, selecting a suitable DAE for migration and setting the necessary arguments for the mobile agent performing the task in the remote DAE. Basically, a new instance of MobileAgent is created for every new task requested by the mobile client. When the mobile client wants to shut down the mobile web browser, *shutDown()* is responsible for releasing all the resources agents and the ports.

```
public class Agents
{
   final private String resPort = "7000";
   final private String outPort = "8000";
   final private String inPort = "9000";
   private MobileAgent mobileAgent;

   public void Agents(String localHostIP)
   public void initialize()
   public void startUP()
   public void MASetup(String docName, String LoD, String query, String userID)
   public void shutDown()
   public String setResourceURL(String resURL)
   public String setLocalHost(String localHostIP)
   public String setRemoteHost(String remoteHostIP)
}
```

Figure 5.4: The Agents class

5.2.2 The MobileAgent class

The constructor method, *MobileAgent()* (see Figure 5.5), is activated by Agents calling the *MASetup()* method. Agents provides the necessary arguments to *MASetting()* and calls *Migrate()* for migrating to the desirable remote DAE. After the mobile agent migration process, Agents requests the ServerThread to create a new thread and listen to the incoming port for receiving the results.

```
public class MobileAgent implements IMobileAgent, Serializable
{
  public void MobileAgent()
  public void MASetting(String resourceURL, IDomainAgent place,
        String localHostIP, String remoteHostIP, String docName,
        String LoD, String query, String userID)
  public void Migrate(IDomainAgent place)
}
```

Figure 5.5: The MobileAgent class

5.2.3 The ServerThread class

The ServerThread (see Figure 5.6) periodically listens to the incoming port. Once the results are returned from the remote DAE, the ServerThread spawns a new thread BufferThread for receiving data.

```
public class ServerThread extends Thread
{
  final private String incomingPort = "10000";
  private ServerSocket socket;
  private Socket s;
  private DataInputStream in;

  public void ServerThread() throws IOException
  public void run()
}
```

Figure 5.6: The ServerThread class

5.2.4 The DomainAgent class

The DomainAgent class (see Figure 5.7) is a core class in the DAE, which is composed of two classes including DocumentAnalyzer and DocumentTransmitter. When the mobile agent migrates successfully, the mobile agent starts to perform the task by calling and passing the necessary arguments into *requestDoc()*. The DomainAgent calls the DocumentAnalyzer to generate the SC of the requested document. Actually, DocumentAnalyzer calls either IC (see Figure 5.8) or MQIC (see Figure 5.9), depending on the query words are provided by the mobile agent or not. When the mobile agent notifies through the *sendDoc()* method of DomainAgent, the DocumentTransmitter sends the document progressively to the mobile client.

```
public class DomainAgent
{
   final private String resPort = "7000";
   final private String outPort = "8000";
   final private String inPort = "9000";
   private IC ic;
   private MQIC mqic;
   private DocumentTransmitter docTransmitter;

   public void DomainAgent()
   public void requestDoc(String LoD, String userID)
   public void requestDoc(String LoD, String query, String userID)
   public void sendDoc(String userID)
}
```

Figure 5.7: The DomainAgent class

5.2.5 The DocumentAnalyzer class

There are two type of DocumentAnalyzer. IC is a class to generate the information content. MQIC is a class to generate the modified query-based information content. The DocumentAnalyzer follows a series of procedures which has been mentioned in Section 4.1. When *Analysis()* of DocumentAnalyzer is called by the DomainAgent, *DocumentRecognizer()* requests the Database Server to retrieve the requested document. Then *CononicalConverter()* starts by treading all words of the document as keyword candidates into the CANDIDATE and calls Stemmer to stem the keyword candidates into their general term. *WordFilter()* removes the *close words*, with the close words dictionary, STOPLIST, and *KeywordExtractor()* selects the qualified keywords from the CANDIDATE. *StructuralCharacteristicGenerator()* is responsible for generating the information content of each organizational unit in various level of detail. In MQIC, the query words are also applied to *QueryRecognizer()* to generate the weighting of each keywords.

5.2.6 The DocumentTransmitter class

When the mobile agent decides to deliver the requested document, *Transmits()* of DocumentTransmitter (see Figure 5.10) is notified by the DomainAgent. *Transmits()* calls *getDocHeader()* for getting the document hierarchy, *getDocToT()* to get the transmission ordering table and *getDocBody()* to retrieve the organizational units of the document. All

the XML documents are transformed by *Xml2Xhtml()* before sending the corresponding XHTML documents to the mobile client.

```
public class IC
{
   private Stemmer stemmer;
   private Hashtable STOPLIST;
   private Hashtable CANDIDATE;

   public void IC()
   public String DocumentRecognizer(String LoD)
   public void CononicalConverter()
   public void WordFilter()
   public void KeywordExtractor()
   public void StructuralCharacteristicGenerator()
}
```

Figure 5.8: The IC class

```
public class MQIC
{
   private Stemmer stemmer;
   private Hashtable STOPLIST;
   private Hashtable CANDIDATE;

   public void IC()
   public String QueryRecognizer(String LoD)
   public String DocumentRecognizer(String LoD)
   public void CononicalConverter()
   public void WordFilter()
   public void KeywordExtractor()
   public void StructuralCharacteristicGenerator()
}
```

Figure 5.9: The MQIC class

```
public class DocumentTransmitter
{
  final private String incomingPort = "10000";
  final private int MAXBUFFER = 65506;
  private Socket s;
  private DataOutputStream out;

  public void DocumentTransmitter()
  public String getDocHeader(String docName, String userID)
  public String getDocToT(String docName, String userID)
  public String getDocBody(String docName, String userID, String sid)
  public String Xml2Xhtml(String xml, String xsl)
  public void Transmits(String data, int docType, String docID)
}
```

Figure 5.10: The DocumentTransmitter class

5.3 The Prototype of MWB and DAE

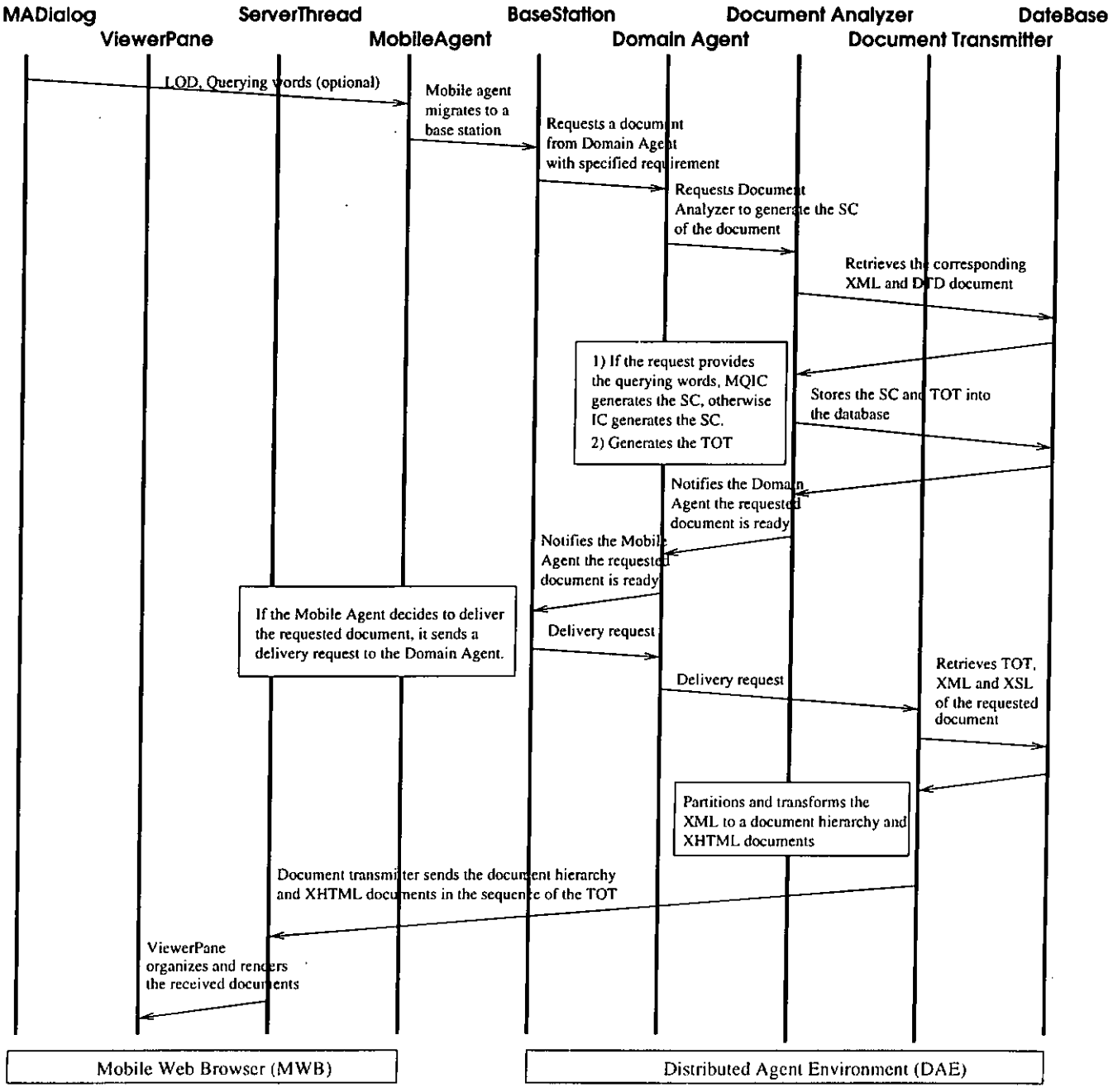

Figure 5.11: Process flow of a mobile agent retrieving document in DAE

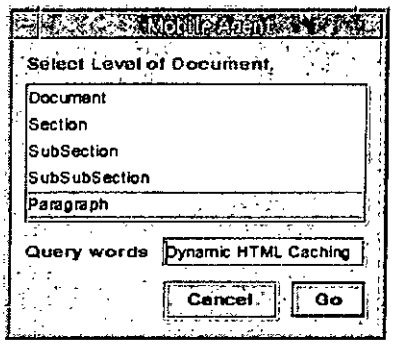

Figure 5.12: A dialog box for mobile agent in the MWB

Figure 5.11 illustrates the process flow of the mobile agent retrieving document in DAE. When a mobile user requests a web document, the user should provide the fields, LOD and querying words (optional), in the dialog box of the mobile web browser (see Figure 5.12). The browser activates the Mobile Agent migrating to the base station and spawns a thread and listen to an incoming socket port and receive the requesting document.

When the Mobile Agent has successfully migrated to the base station from the mobile web browser, the agent starts to request a document and provide the necessary parameters to the Domain Agent. When the Domain Agent receives the request of the Mobile Agent, the Domain Agent calls the Document Analyzer to generate the corresponding SC of the requesting XML document by the IC or MQIC, depending respectively on whether the querying words are provided or not. Then the Document Analyzer generates the TOT, which contains the transmission sequence based on the information content of the requesting document. At this point, both SC and TOT are stored into the Database Sever. In the mean time, the Domain Agent notifies the agent that the requested document is ready. If the agent decides to deliver the requested document, the agent notifies the Domain Agent to deliver the requested document. The Domain Agent, then, sends a delivery request to the Document Transmitter to retrieve the corresponding TOT, XML and XSL of the requested document from the Database Server. In addition, the Document Transmitter partitions and transforms the XML into a document hierarchy and XHTML

documents based on LOD defined by the mobile user. Then the Document Transmitter establishes an outgoing TCP or UDP socket connection and sends the document hierarchy and the rest of the XHTML documents, in the sequence of the TOT to the mobile web browser.

Figure 5.13 shows a series of snapshots of the browser when a document is transmitted in **Paragraph** LOD. The content displayed at three different stages after **Paragraph 1.0.0.1**, **Paragraph 1.0.0.3** and **Paragraph 1.0.0.0** of **Section 1** have been received and rendered as depicted in Figure 5.13(a), 5.13(b) and 5.13(c) respectively. To give users a better idea of what has been received in the browser, our prototype also provides a road map, maintained by a Sequence Manager on the left-hand side of the browser. The Sequence Manager shows the corresponding organizational units that have been received and rendered by the browser at the moment. Figure 5.13(d) illustrates the final state when all paragraphs have been received and rendered by the browser completely.

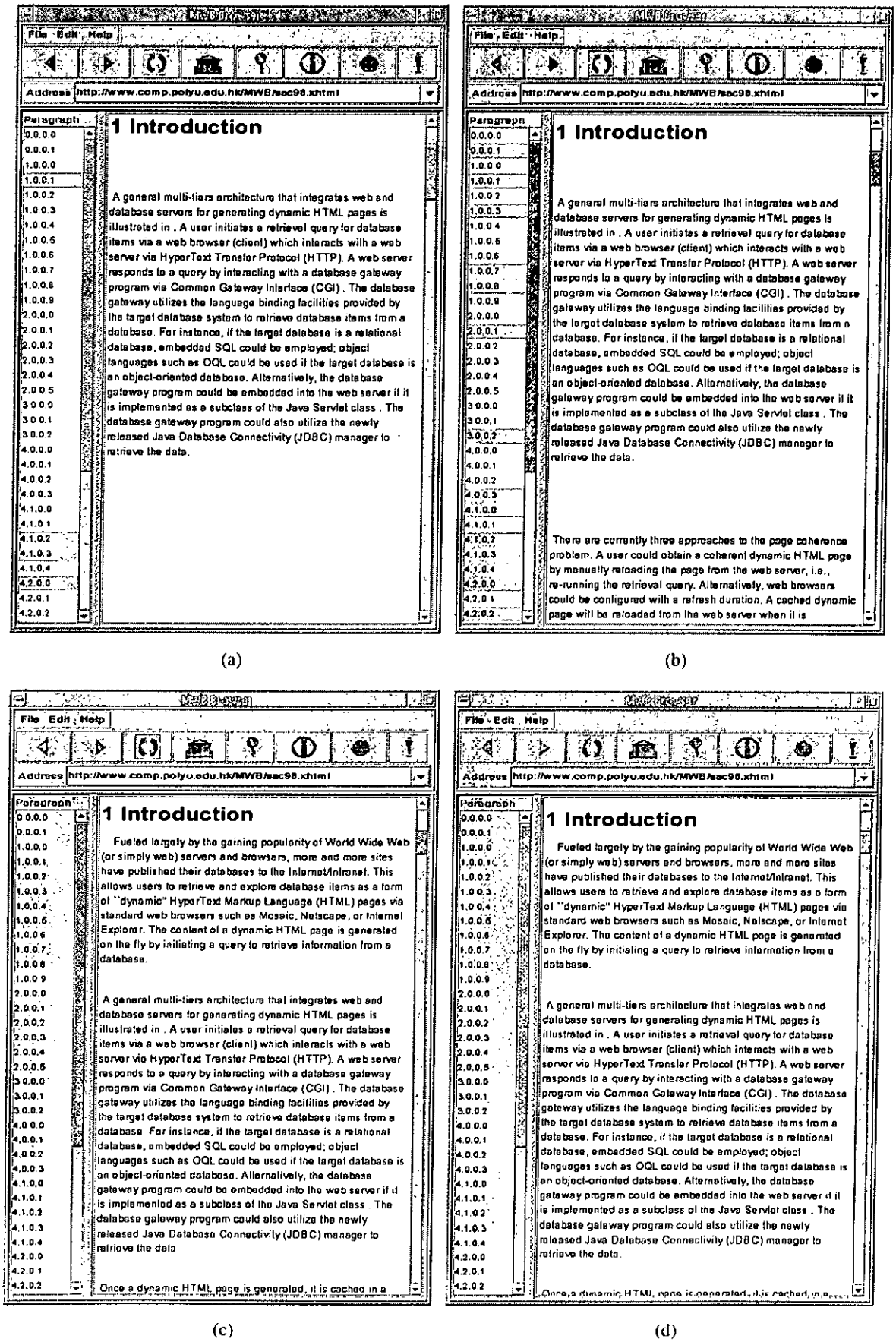

Figure 5.13: Snapshots of MWB in paragraph LOD

Chapter 6

Performance Study

In our experimental model, we assume that web documents have already been partitioned and stored in a backend database according to the structural characteristic for each document. A web client submits requests via dial-up, wireless or wired network to a base station and retrieves the web documents based on the web client's desirable querying words and LOD. To approximate the possibly large set of different queries to retrieve the web documents in the backend database [55], we adopt the queries of Wisconsin benchmark [14] to evaluate the performance on the primitive database operations in an Oracle database. This will help to provide a guideline on the impact of the response time versus the size of the query result set. Therefore, we can focus on the study of the performance on mobile agent-based computing in DAE and to compare the performance with client/server models.

In conventional mobile agent model, a mobile agent migrates to the base station, performs the desired processing and returns the results through a retract operation of the mobile client. We refer to this model as *Mobile Agent* (MA) or pure mobile agent computing. Our studies indicate that the overhead of carrying the results with the mobile agent is quite high.

To remedy this problem, we propose two variants of pure mobile agent computing, namely, *Mobile Agent with TCP* (MA_TCP) and *Mobile Agent with UDP* (MA_UDP).

In MA_TCP, mobile agent is dispatched to the base station for processing, as before. However, the results are not carried back to the client together with retraction of the mobile client. Rather, they are explicitly transmitted back to the client by means of TCP. Similarly, UDP is used to transmit results back to the client in MA_UDP. As a base case for performance comparison, we adopt the standard *Common Gateway Interface* (CGI) model for client/server interaction. The performance difference between these four communication structures to access the backend database [64] is presented in Section 6.2. Comparison among the dial-up, the wireless, and the wired network is presented in Section 6.3. In appendix A, we measure the performance on various client/server models which including CGI, *Servlet*, *Common Object Request Broker Architecture* (CORBA), and *Remote Method Invocation* (RMI) in accessing a back-end database.

The reliability of transmission in a mobile environment is also an important issue, since the wireless communication channel suffers from a larger degree of signal distortion, interference, and attenuation. We proposed a fault-tolerant transmission mechanism for mobile agent model which is called *Mobile Agent with Forward Error Correction* (MA_FEC). MA_FEC is based on the VMFEC coding method, which is discussed in Section 4.2 and it adoptes UDP as a transmission protocol. The performance on VMFEC encoding and decoding is presented in Section 6.4.1. In Section 6.4.2, we present the performance for mobile agents models with varying error rates when downloading a web document. Finally, in Section 6.4.3, we show the performance for all mobile agent models across different networks, in parallel to Section 6.2 and Section 6.3.

6.1 Experimental Environment

In our experimental framework, the client, the web server, and the database server all reside in different machines (see Figure 6.1). We have installed three different network environments including the wired network with 100Mbps bandwidth, wireless network with 2Mbps, and dial-up network with 28.8kbps. The detail configuration is presented

in the following:

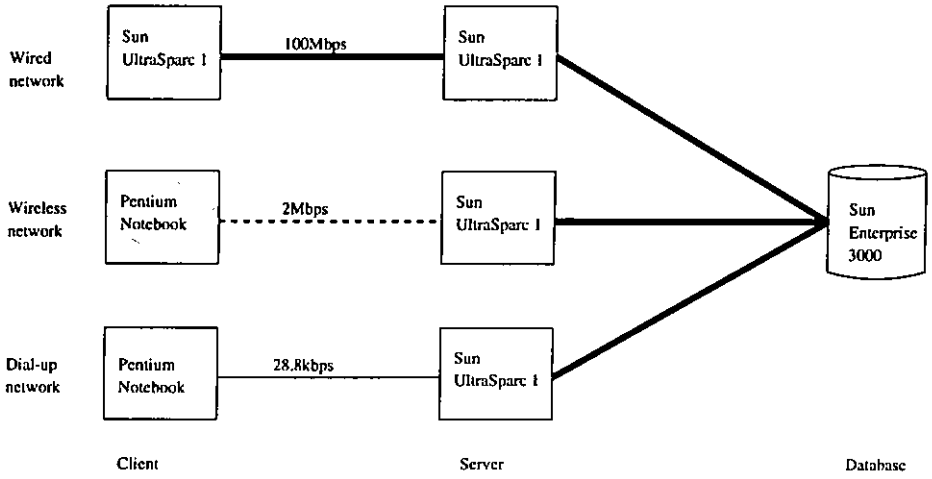

Figure 6.1: The framework of the experiment across different networks

- In the wired network, both the client and the web server run on Sun UltraSparc 1 platform, equipped with 64MB of memory. All the machines are connected via campus-wide LAN with 100Mbps bandwidth.

- In wireless network setting, the client runs on a notebook with Pentium 100MHz processor and 48MB of memory, under Microsoft Windows NT 4.0 operating system. The client and web server are connected via wireless LAN with 2Mbps bandwidth.

- In the dial-up network, the client runs on a machine with the same configuration as the notebook. The bandwidth of the dial-up is 28.8kbps.

In the database configuration, all queries are directed to an Oracle 7 database server, residing on a Sun Enterprise 3000 server machine, which is configured with two 168MHz UltraSparc processors and 1GB of memory. We have implemented four different communication structures, namely, traditional client/server web-based computing CGI, straightforward mobile agent computing MA, mobile agent with TCP result collection MA_TCP and mobile agent with UDP result collection MA_UDP. The CGI program is written

in Perl 5. The mobile agents are developed with ObjectSpace Voyager ORB Pro 3.3. Finally, accesses to Oracle are via JDBC driver 7.3.4.

We adopted the Wisconsin benchmark [14] in configuring our Oracle database with three relations, **WIS1**, **WIS2**, **WIS3**. In **WIS1** and **WIS2**, each contains a total of **MAXTUPLES** tuples. The number of tuples in **WIS3** is 10% of **MAXTUPLES**. The schema is depicted in Table 6.1, following closely the Wisconsin Benchmark.

Attribute Name	*Range of Values*	*Order*	*Comment*
unique1	0-(MAXTUPLES-1)	random	unique, random order
unique2	0-(MAXTUPLES-1)	sequential	unique, sequential
two	0-1	random	(unique1 mod 2)
four	0-3	random	(unique1 mod 4)
ten	0-9	random	(unique1 mod 10)
twenty	0-19	random	(unique1 mod 20)
onePercent	0-99	random	(unique1 mod 100)
tenPercent	0-9	random	(unique1 mod 10)
twentyPercent	0-4	random	(unique1 mod 5)
fiftyPercent	0-1	random	(unique1 mod 2)
unique3	0-(MAXTUPLES-1)	random	(unique1)
evenOnePercent	0,2,4,...,198	random	(onePercent * 2)
oddOnePercent	1,3,5,...,199	random	(onePercent * 2)+1
stringu1	-	random	candidate key
stringu2	-	sequential	candidate key
string4	-	cyclic	

Table 6.1: Attribute specification of scaleable Wisconsin benchmark relations

In our experiments, we vary **MAXTUPLES** from 100, 1000 to 10000. The number of tuples in **WIS3** is 10% of **MAXTUPLES**. Each experiment is repeated 10 times and the average over the 10 trials is measured. In each experiment, we measure the response time, in seconds, from the moment that a query is initiated at a client to the moment the results of the query are returned to the client. We also measure the breakdown of time for the whole querying cycle. The time spent from client to web server and the time from web server back to client is termed the client/server turnaround, denoted as **C-S**. The time taken for web server to database and then back is termed the server/database turnaround denoted as **S-D**. The total response time is thus denoted as **C-D**. To conduct a comparative study among the various communication structures, it would be more appropriate to compare the ratio of time spent by the various communication structures

against the one with the worst performance, thus contrasting their relative performance. This is characterized by the speedup metric, since it is independent of the size of database and that of the result sets across the queries.

Table 6.2 lists the set of Wisconsin benchmark template queries we used in our experiment for **MAXTUPLES** = 10000. There are totally 20 queries, which we have classified into two groups: read-only queries and update queries. The read-only query group contains a total of 16 queries and the update query group contains a total of 4 queries. The parameters to the queries are proportional to the actual size of database. The size of result sets for each query is presented in Table 6.3.

Query code	*Read-only Queries:*
Q1	select * into from WIS1 where unique2 >= 0 and unique2 <= 99
Q2	select * into from WIS1 where unique2 >= 792 and unique2 <= 1791
Q3	select * into from WIS1 where unique1 >= 0 and unique1 <= 99
Q4	select * into from WIS1 where unique1 >= 792 and unique1 <= 1791
Q5	select * from WIS1 where unique2 = 2001
Q6	select * into from WIS1, WIS2 where WIS1.unique2 = WIS2.unique2c and WIS2.unique2c < 1000
Q7	select * into from WIS1, bprime where WIS1.unique2 = bprime.unique2c
Q8	select * into from WIS1, WIS2, WIS3 where WIS3.unique2b = WIS1.unique2 and WIS1.unique2 = WIS2.unique2c and WIS1.unique2 < 1000
Q9	select * into from WIS1, WIS2 where WIS1.unique1 = WIS2.unique1c and WIS1.unique1 < 1000
Q10	select * into from WIS1, BPRIME where WIS1.unique1 = BPRIME.unique1c
Q11	select * into from WIS1, WIS2, WIS3 where WIS3.unique1b = WIS1.unique1 and WIS1.unique1 = WIS2.unique1c and WIS1.unique1 < 1000
Q12	select distinct two, four, ten, twenty, onePercent, stringu4 into from WIS1
Q13	select distinct two, four, ten, twenty, onePercent, tenPercent, twentyPercent, fiftyPercent, unique3, evenOnePercent, oddOnePercent, stringu1, stringu2, stringu4 into from WIS1
Q14	select min(WIS1.unique2) from WIS1
Q15	select min(WIS1.unique3) from WIS1 group by WIS1.onePercent
Q16	select sum(WIS1.unique3) from WIS1 group by WIS1.onePercent
Update code	*Update Queries:*
U1	insert into WIS1 values (10001, 74000, 0, 2, 0, 10, 50, 688, 1950, 4950, 99500, 1, 100, 'MxxxxxxxxxxxxxxxxxxxxxxxxxxGxxxxxxxxxxxxxxxxxxxxxxxxC', 'GxxxxxxxxxxxxxxxxxxxxxxxxxxCxxxxxxxxxxxxxxxxxxxxxxxxA', 'OxxxxxxxxxxxxxxxxxxxxxxxxxxOxxxxxxxxxxxxxxxxxxxxxxxxO')
U2	delete from WIS1 where unique1=10001
U3	update WIS1 set unique2 = 10001 where unique2 = 1491
U4	update WIS1 set unique1 = 10001 where unique1 = 1491

Table 6.2: Queries of Wisconsin benchmark

Query / Update Code	*100 tuples*	*1000 tuples*	*10000 tuples*
Q1	758	3852	39156
Q2	4248	39368	395652
Q3	744	3758	38804
Q4	4198	39362	395568
Q5	388	388	388
Q6	7668	77698	787946
Q7	7668	77698	787946
Q8	11388	116098	1177886
Q9	7476	77210	783834
Q10	7668	77698	787946
Q11	11196	115610	1173874
Q12	12880	48436	51522
Q13	37240	374382	3763414
Q14	6	6	6
Q15	582	582	582
Q16	580	1002	1402
U1	4	4	4
U2	4	4	4
U3	4	4	4
U4	4	4	4

Table 6.3: The size of result set (in bytes) of Wisconsin benchmark

6.2 Performance on Different Models

In our first experiment, we look at the performance on the Wisconsin queries using different communication structures, namely, CGI, MA, MA_TCP, and MA_UDP. The measured performances from the four communication structures, expressed in terms of speedup against the slowest one, are depicted in Figure 6.2. The first column depicts the speedup of **C-S** turnaround against MA and the second column the speedup of **S-D** turnaround against CGI.

From Figure 6.2, we observe that MA_UDP has the best performance, followed by MA_TCP. Both of them perform better than the pure mobile agent solution. This is expected since the results are transmitted back to the client via the more efficient TCP and UDP mechanisms, whereas in MA, the results are retracted together with the agent from the web server or base station back to the client. Furthermore, UDP is more effective in this case, due to minimal network connection establishment overhead.

We observe that in general, MA_TCP and MA_UDP perform much better than CGI, illustrating the effectiveness of the mobile agent based model when TCP or UDP are

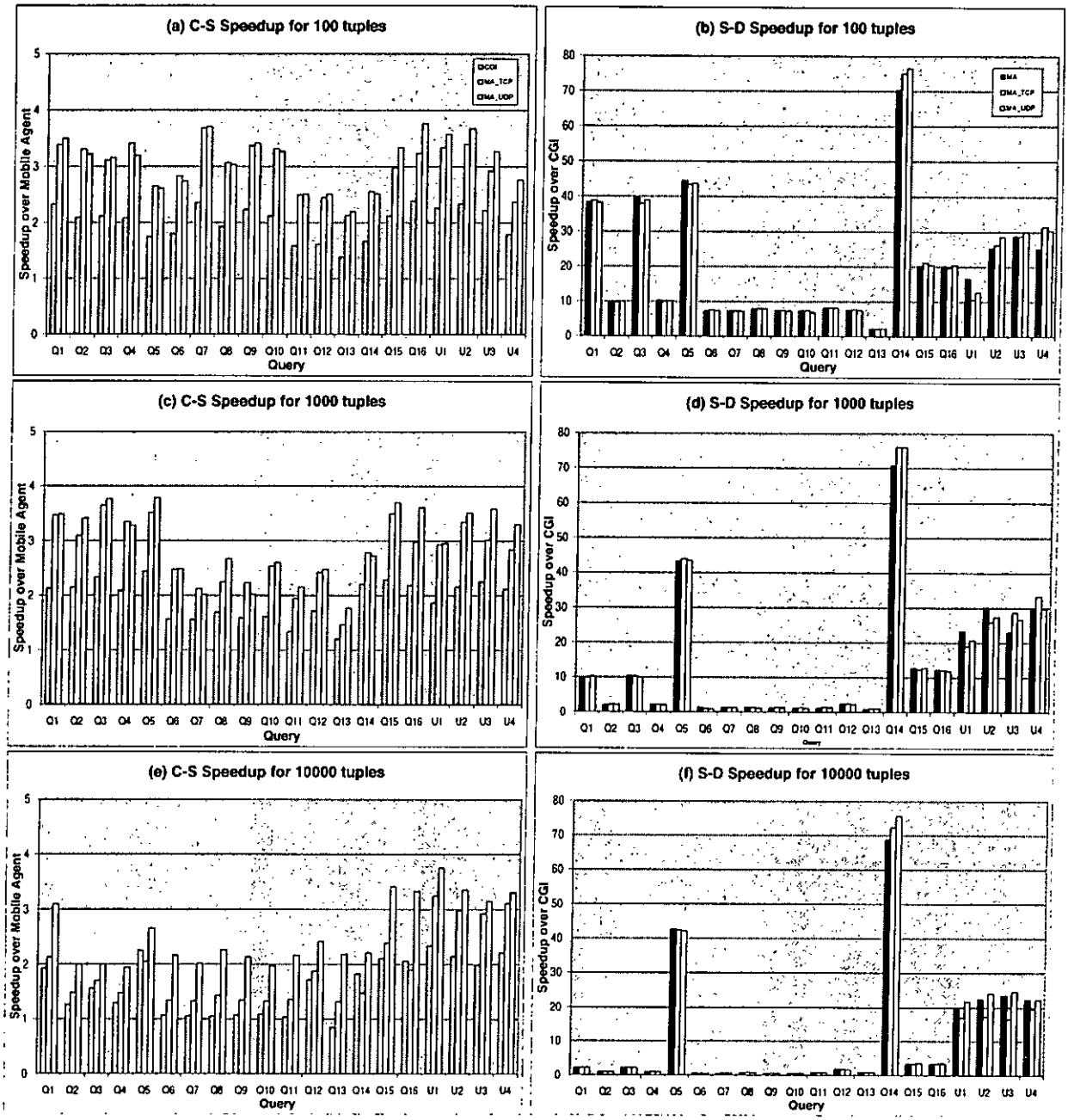

Figure 6.2: Performance on Wisconsin queries with varying communication structures

used in conjunction. The more interesting cases arise when MA is compared with CGI. When client/server turnaround is concerned, CGI performs better than MA in general, with a speedup around 1 to 2. This is primarily because in MA, there are significant overheads in launching a mobile agent from the client DAE to the web server DAE

at the base station. Furthermore, the overhead of bringing back the results with the agent at the end of its mission is even higher. Such two types of overheads are higher than the invocation and process creation overhead of a CGI program in most cases, possibly except for Q13 with a database of 10000 tuples. However, purely agent migration overhead is lower than the invocation overhead of CGI, leading to better performance on MA_TCP and MA_UDP. On the other hand, when server/database turnaround is concerned, MA is performing better than CGI in most cases. This is understandable, since the overhead of CGI interaction with database is quite significant. Furthermore, there is virtually no difference among the performance on the three mobile agent models as far as server/database turnaround is concerned, as the three only differ in the way results are conveyed from the web server back to the client, but not in the way how the database is accessed. The seemingly high speedup to some of the queries is due to the fact that the fixed CGI overhead is large compared with actual time spent for the database query. This occurs in Q5 and Q14, both of which have only a small result set (see Table 6.3). The same occurs for Q1 and Q3 in a small database, both also return small result sets. Similarly, update queries also return small result sets.

When the database size increases, there seems to be no major change in the relative performance. The only observable impact is with the speedup of **S-D** for some read-only queries. This is probably due to the fact that the fixed overhead for CGI execution towards database does not increase much. However, those read-only queries return a result set with size proportional to the size of database, thereby increasing the variable overhead for the execution, thus bring down the speedup as measured against CGI.

We rescale the second column of Figure 6.2 to highlight and study the speedup of those queries with a moderate speedup in Figure 6.3. It is interesting to observe that the speedup of some queries for a large database is close to and may be slightly lower than 1, indicating that CGI is performing as well as the rest. A closer investigation reveals that these queries result in large server to database response time in all models. The

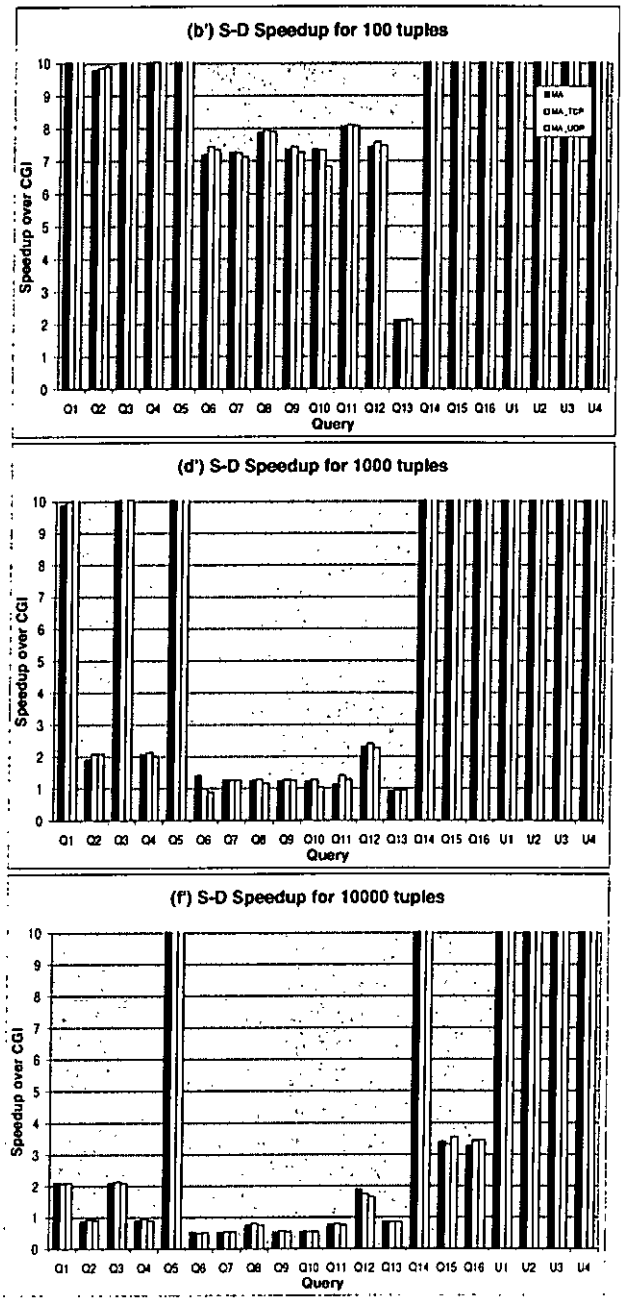

Figure 6.3: Performance on different queries in wired network

overhead of CGI invocation vanishes when compared to the actual query execution time by the database server, thus bringing all speedups to around 1.

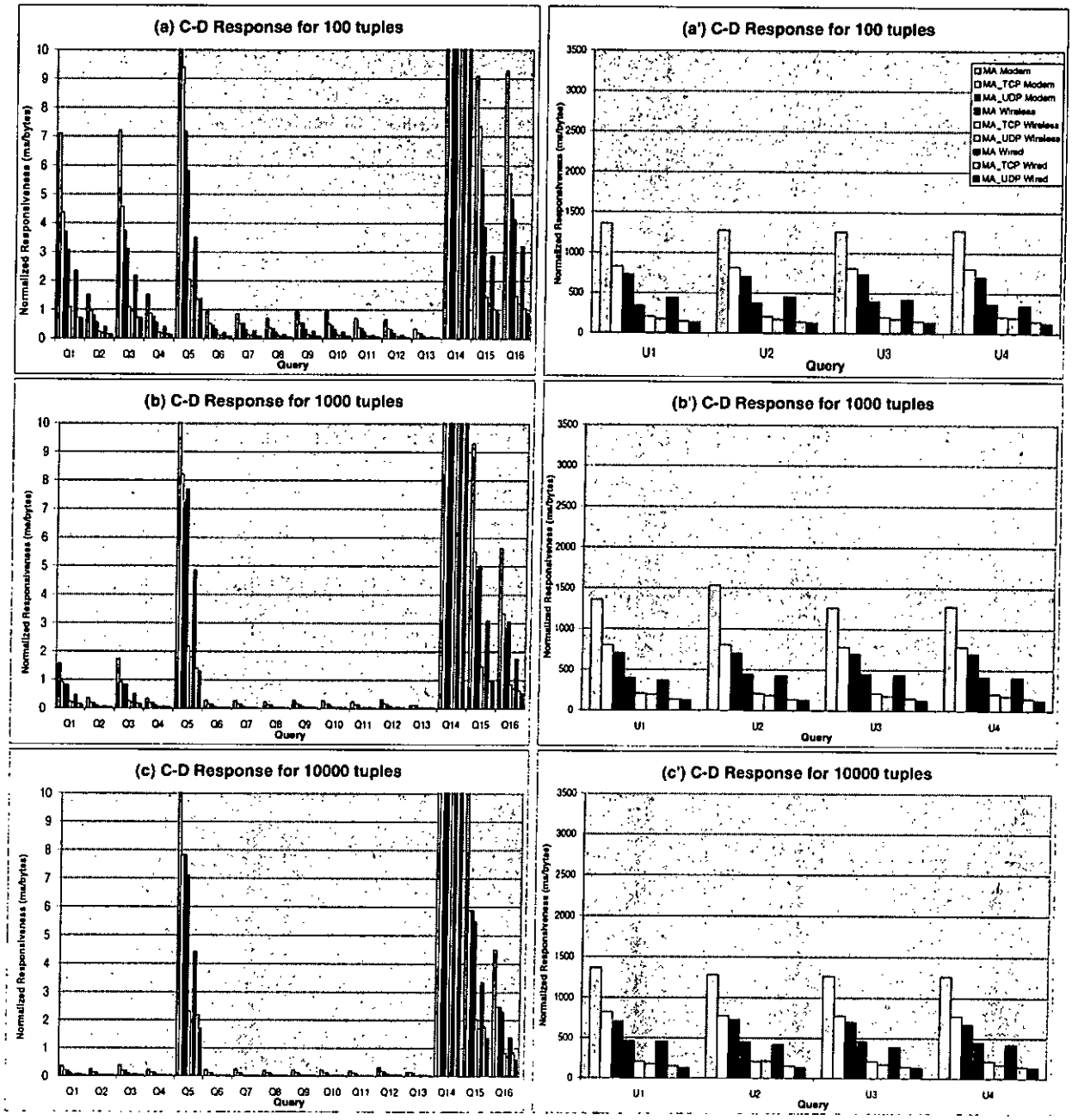

Figure 6.4: Performance across different networks

6.3 Performance across Networks

As a mobile user, a very important factor is the responsiveness of queries, besides connectivity and mobility. In this experiment, we therefore measure the overall response time using the mobile agent models, under the dial-up, wireless and wired network. Since the

different queries in Wisconsin benchmark vary a lot in nature, we would like to make a fairer comparison, by measuring the *normalized responsiveness* of the queries using the different communication structures. Normalized responsiveness is measured by the time (in ms) spent to retrieve one byte of result. The results in terms of responsiveness (for client/database or C-D turnaround) are depicted in Figure 6.4. In the figure, we present the results according to read-only queries and update queries, in two columns. This is because read-only queries normally have large result sets and update queries usually involve very small result sets. It is clear that the performance trends in the three mobile agent models are similar, for the dial-up network, the wireless network and the wired network, namely, performance on MA_UDP is slightly better than that of MA_TCP, while both of them are better than MA. The explanation is the same as in previous experiment. The change of network platform has no impact on the relative performance on the different mobile agent models. Larger databases normally result in better normalized responsiveness, due to the increase of result set size with respect to the relatively stable processing overhead, leading to a slower time per result byte. The poor normalized responsiveness for update queries as well as those for Q5, Q14, Q15, and Q16 is due to the small result sets of the associated queries, compared with the relatively stable overhead.

6.4 Performance on Fault-tolerant Multi-resolution Transmission Mechanism

To complement the multi-resolution transmission and browsing mechanism with support for reliable mobile web browsing under mobile environment, we have evaluated the performance on the VMFEC encoder and decoder under different platforms in Section 6.4.1. We compare the performance on MA_FEC to the other mobile agent models which include MA, MA_TCP, and MA_UDP in different networks in Section 6.4.3.

In the experimental framework, we adopt the same experimental environment as in Section 6.1. Instead of adopting the queries in the Wisconsin benchmark returning the

variable size of results with different queries, we fixed the amount of the results returning from the database in 10240 bytes for simplicity. Each experiment is repeated 10 times and the average time spent over the 10 trials is measured.

6.4.1 Performance on VMFEC Encoder and Decoder

In this experiment, we study the run-time performance on the VMFEC encoder and decoder in Java version under Sun UltraSparc 1 and Pentium platforms. On each platform, we assume that a certain amount of data is encoded and decoded by the VMFEC encoder and decoder. We term the amount of data as a block. The block contains a number of packets. A packet contains a number of bytes. For simplicity, we fixed the size of a block to 10240 bytes in our experiments. We vary the packet lengths, p, in 256 bytes, 512 bytes, and 1024 bytes. Therefore, when a longer length of packet is selected, the number of raw packets in a block is fewer. We vary the number of redundant packets generated from the raw packets, as a redundancy ratio, to perform the VMFEC encoding and decoding (see Table 6.4). We obtain the time spent, in terms of seconds, to generate certain number of redundancy packets from the VMFEC encoder and decoder.

Redundancy Ratio	*1.0*	*1.1*	*1.2*	*1.3*	*1.4*	*1.5*	*1.6*	*1.7*	*1.8*	*1.9*	*2.0*
p=256 bytes	0	4	8	12	16	20	24	28	32	36	40
p=512 bytes	0	2	4	6	8	10	12	14	16	18	20
p=1024 bytes	0	1	2	3	4	5	6	7	8	9	10

Table 6.4: Number of redundant packets needed under different packet lengths

In Figure 6.5 and Figure 6.6, we observe that the VMFEC encoding and decoding time under the Pentium platform is less than the Sun UltraSparc 1 platform. Encoding takes roughly the same time as that of decoding. The encoding and decoding times are roughly linear with the redundancy ratio. In Table 6.4, when the packet length is 1024 and the redundancy ratio is 2.0, the number of redundant packets needed is 10. The encoding and decoding time is 1.5 seconds each for Sun platform. To the extreme, when the packet length is 256 and the redundancy ratio is also 2.0, the number of redundant

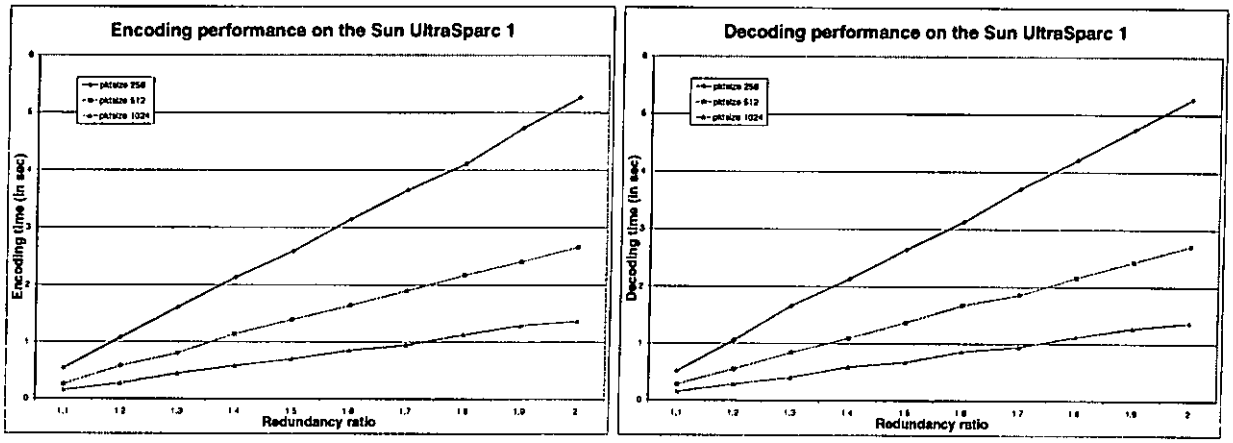

Figure 6.5: Performance on VMFEC encoder and decoder on Sun UltraSparc 1

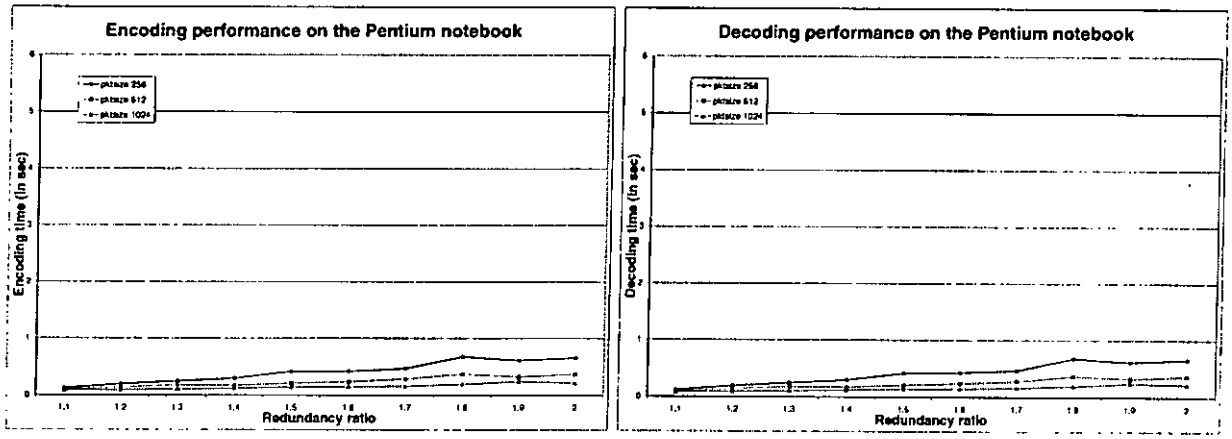

Figure 6.6: Performance on VMFEC encoder and decoder on Pentium notebook

packets needed is 40. The encoding and decoding time is around 5.5 seconds each.

In [52], the comparison on the performance of Java-language Vandermonde coder and the C-language Vandermonde coder has been reported. It is found that the Java-language Vandermonde coder is hundred times slower than the C-language Vandermonde coder, but the Java-language Vandermonde coder using just-in-time (JIT) compilation can reduce the difference to only 25 to 30 times slower. In this thesis, we use the Java-language Vandermonde coder for portability and for test of concept and feasibility. We can use a C-language Vandermonde coder if performance becomes an issue.

We observe that the longer the packet length composing a block and the lower redundancy ratio is selected, the lesser redundant packets are needed and the shorter

encoding/decoding time is performed either on Sun UltraSparc 1 platform or Pentium platform. In the mobile environment, the mobile devices have a low processing power CPU than Pentium and Sun UltraSparc 1 machines and the packet length is usually fixed and short, it is important that we should choose the packet length as large as possible and the redundancy ratio as small as possible.

6.4.2 Performance for Mobile Agent Models with Varying Error Rates

In this experiment, the mobile agent retrieves a web document of size 10240 bytes from database. We vary the packet lengths in 256 bytes, 512 bytes and 1024 bytes to transmit the whole web document to the mobile client. We evaluate the performance on MA_UDP, MA_TCP, and MA_FEC in the presence of transmission errors across different networks. For the sake of simplicity, we assume that transmission errors occur according to a Poisson arrival distribution and the redundancy ratio is 1.5. The distribution is defined as $E(l_i) = 1 - e^{-\lambda l_i}$ where λ is the error rate from 0.1 to 0.5, and l_i is a normalized packet length which can be 1, 2, or 4.

In Figure 6.7, the first, the second and the third columns show the performance on MA_UDP, MA_TCP and MA_FEC across wired, wireless, and dial-up networks respectively. We observe that when the error rate increases, the response time increases for all packet lengths in MA_UDP, MA_TCP and MA_FEC models across all networks. It is because the higher the error rate is, the more packets will be transmitted with failure, therefore the more packets will have to be re-transmitted and the longer the response time are. On both wired and wireless networks, the response time of MA_UDP is faster than MA_TCP, and then MA_FEC in all error rates. It is shown that the overhead of computation on encoding and decoding in MA_FEC is quite high. However, in dial-up network, when the error rates are from 0.3 to 0.5 with 1024 bytes packet length, the response time on MA_FEC is faster than MA_UDP and MA_TCP. When the error rates

are from 0.1 to 0.5 with 1024 bytes packet length, the response time on MA_FEC is faster than MA_TCP. It is because the contribution of redundant packets has reduced the overhead of re-transmission of failed packets. In addition, we observe that the performance on MA_FEC with 512 bytes packet length with error rates from 0.2 to 0.5 is very close to the performance on MA_TCP with same packet length accordingly. In other words, if MA_FEC adopts a C-language Vandermonde coder instead of a Java-language Vandermonde coder, the overhead of computation on encoding and decoding can decrease substantially.

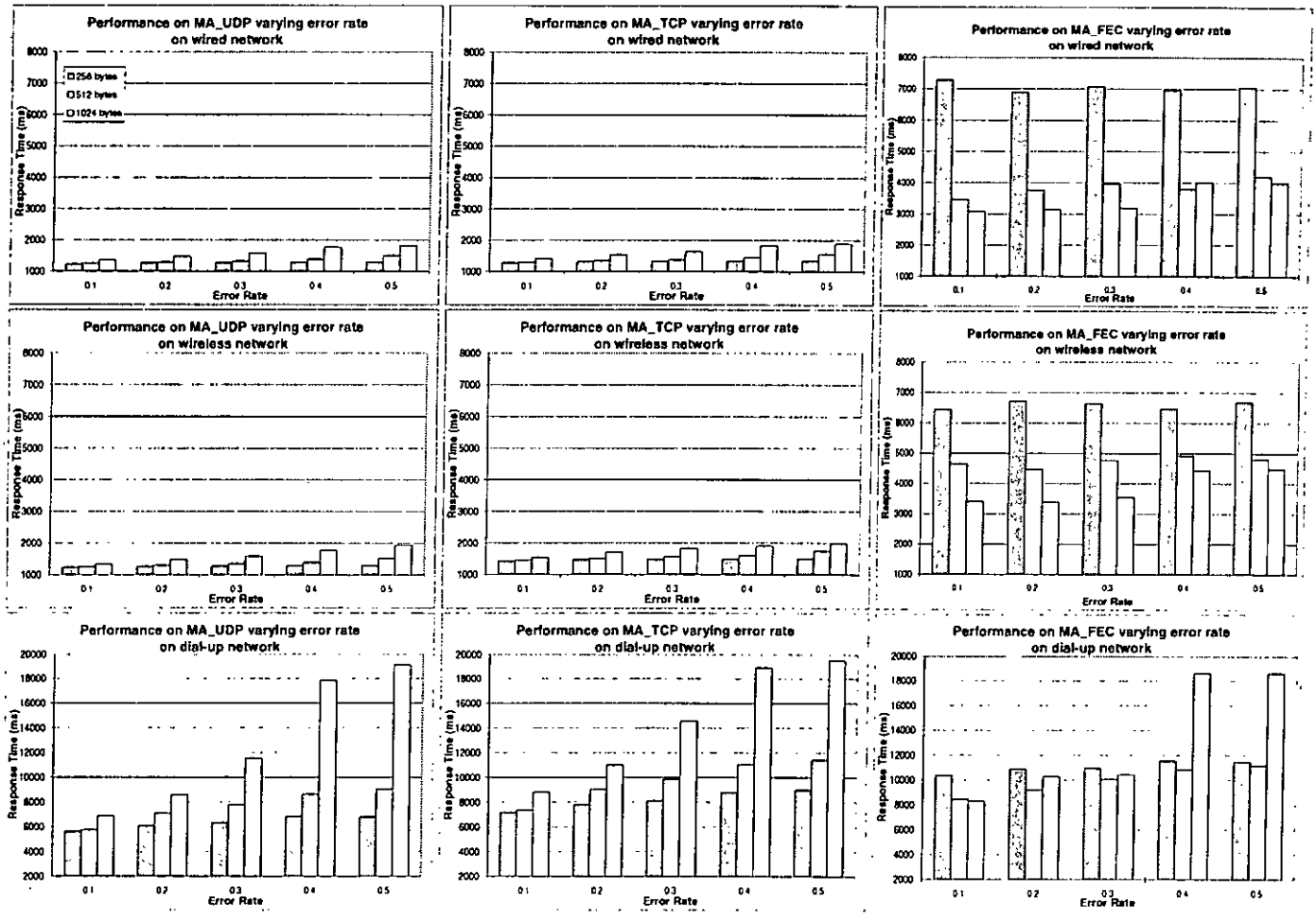

Figure 6.7: Performance for Mobile Agent Models with varying error rates

6.4.3 Performance for Mobile Agent Models across Different Networks

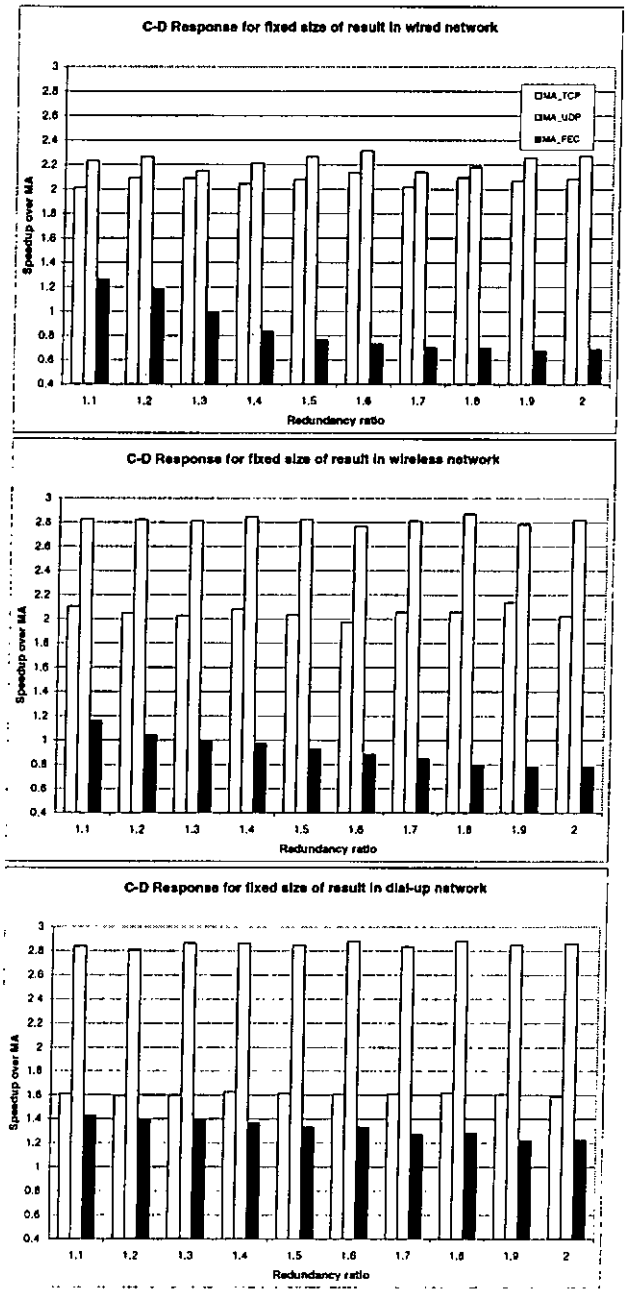

Figure 6.8: Performance for mobile agent models across different networks

This experiment aims at comparing the performance on MA_FEC with the other mobile agent models to retrieve a web document of size 10240 bytes, from database. For the sake

of simplicity, we assume that the maximum packet length is 1024 bytes in the mobile environment. We choose a block size of 10 and packet length of 1024 for the parameters on VMFEC encoder and decoder. We vary the redundancy ratio from 1.1 to 2.0. The measured performances from all mobile agent models, expressed in terms of speedup against the slowest model, MA, are depicted in Figure 6.8. The first rows, the second rows and the third rows represent the speedup of **C-D** response for fixed size of result on the dial-up network, the wireless network and the wired network respectively.

From Figure 6.8, the performance on MA_UDP, in terms of speedup, is better than MA_TCP, while both of them are better than MA_FEC and MA in all networks.

We observe that the MA_FEC performs better than MA. When redundancy ratio is 1.1, the MA_FEC performs better than MA, which has a speedup around 1.4 in the dial-up network and around 1.2 in the wireless network and the wired network. When redundancy ratio is 1.3, the MA_FEC still performs better than MA in the dial-up network, which has a speedup around 1.4 but only performs equal to MA in both wireless network and wired network. When the redundancy ratio is increased from 1.3 to 2.0, the speedup of MA_FEC in the dial-up network also keeps in 1.2, but the speedup of MA_FEC in the wireless and the wired network are decreased to 0.8 and 0.7 respectively.

We observe that when the communication bandwidth is low, i.e. 28.8kbps in the dial-up network, the overhead for MA migration is relatively high and the overhead of encoding/decoding is relatively low. In other words, when the communication bandwidth is high, i.e. 100Mbps in the wired network, the overhead for MA migration is relatively low and the overhead of encoding/decoding is relatively high. We conclude that the MA_FEC not only performs better than MA, but also can tolerate the transmission fault in the low communication bandwidth network.

Chapter 7

Conclusions

A mobile environment is weakly connected, characterized by low communication bandwidth and poor connectivity. Users retrieving a web document in the mobile environment may have to wait for a long period of time before actually reading the document, possibly wasting the time to obtain documents that they are not interested in. To shorten the response time and utilize the scarce communication bandwidth, we have presented the multi-resolution transmission and browsing mechanism for transmitting and browsing web documents at various levels of detail. We have explored issues related to the development and deployment of generating the *structural characteristic* (SC) of the XML documents. We have implemented the Document Analyzer that derives the content level of each organizational unit from the corresponding document's keyword-based document tree and generates the information content of SC which is stored into the Database Server. Based on the notion of information content, the Document Transmitter progressively transmits the main document content before transmitting supplementary information in the sequence of *Transmission Ordering Table* (TOT). We have proposed the *Query-based Information Content* (QIC) and *Modified Query-based Information Content* (MQIC) to generate different transmission sequences of organizational units based on different users interest on the same document. We have enhanced the multi-resolution transmission and browsing paradigm support for mobile web browser; we have developed the redun-

dant transmission scheme, namely VMFEC, to increase the recoverability of corrupted document fragments due to unreliable wireless transmission.

We have designed *Distributed Agent Environment* (DAE) to highlight on the usefulness of the mobile agent model, which provides a platform for the multi-resolution transmission and browsing mechanism, dedicated to mobile web users. We have implemented the prototype of mobile web browser to browse the XML documents progressively from the remote web server and database server. We have demonstrated the look and feel of the mobile web browser using the mobile agent model to request and retrieve a XML document with the multi-resolution transmission and browsing mechanism implemented in the DAE.

Upon implementation of a prototype, we attempt to locate for the performance breakeven point between the mobile agent models against the conventional client/server models. A major performance penalty to the mobile agent solution is due to the return of results to the mobile client at the end of its itinerary upon retraction. Alternative mechanisms of returning results are explored, namely, via TCP and via UDP. The protocols are implemented and the performance is studied under different communication bandwidth, under environments over the dial-up network, the wireless network and the wired network respectively. We have concluded with a number of experiments that returning results actively by mobile agent is perhaps the most effective solution among others. We have measured the performance on VMFEC encoding/decoding and incorporated it into mobile agent model, MA_FEC. We have concluded with an experiment that the performance on MA_FEC is also better than pure mobile agent model in a low communication bandwidth network.

Some issues still remain to be addressed and some existing solution to be improved. Though our multi-resolution transmission and browsing paradigm support mobile web browser on our own-defined XML documents, it is not compatible with a well-defined XML document. The heavy-weight mobile agent model is not suitable to integrate to

the low-end machines, i.e. hand-held devices and personal digital assistant. It is also important that the persistent caching and the prefetching mechanism are efficient approach to utilize the low communication bandwidth in the mobile web environment. We hope and believe that the work described in this thesis could contribute to create a mature and comprehensive background for the evolution of the research area concerned with multi-resolution web browsing in the mobile web environment.

Chapter 8

Future Work

In our prototype of mobile web browser, we currently only support the browsing of XML documents that are defined by the DTD (see Figure 4.4). In the future, we believe that many newly published research papers will be generated in XML, e.g. an XML version of ACM SIGMOD Record [39]. We should enhance the mobile web browser that can adopt a well-defined DTD and have the XML documents source for further research and exploration in mobile web browsing.

Presenting a mobile user with a summary of each organizational unit in a document can help the user identify quickly which organizational units are most relevant to the user's needs. This can be either a generic summary which gives an overall sense of the content of each organizational unit, or a query-based summary which presents the content of each organizational unit that the most closely related to the user's querying words [19]. In other words, integration of document summarization techniques into our multi-resolution transmission and browsing mechanism can shorten the document transmission time from the base station to the mobile client. Therefore the mobile user can receive more summarized information of the document as soon as possible and to better filter out unwanted information earlier.

In mobile environment, caching is an effective technique to improve interactive performance, since the connection often has low and unreliable bandwidth. It is unclear

where to place a proxy server for a mobile user who may roam among several base stations. Thus, the cache in a mobile environment can be better placed at the mobile device itself. We are investigating cache persistent across sessions in order to benefit from locality between different sessions. Additionally, a persistent cache may be pre-loaded for disconnected operation [53], which is transparent to a user unless a cache miss occurs. Returning to normal operation is also transparent, unless a conflict is detected.

If we knew exactly which pages a user needed next, we would retrieve only those pages in advance utilizing the idle communication channel. If it is assumed that the prefetching operation always finishes before the user requests the next page, then we would enjoy zero latency with no extra bandwidth consumption. With respect to a collection of related pages in the form of a cluster, we are investigating intelligent prefetching based on information content and user profiling.

When the mobile agent migrates frequently across different distributed agent environments, the mobile agent should be as lightweight as possible. The mobile agent should also utilize and retrieve the user preferences from the user profile agent in the distributed agent environment. We will explore how to reduce the overhead of the migration process in the mobile agent model. We propose that the mobile agent should reside in a base station instead of the low-end mobile device. In this case, the low-end mobile device only sends the initial setup parameters and requests to the mobile agent in the base station, and the migration process can execute in the high-end machines. To this end, the mobile user only submits the specification of the requested mobile agent to the base station, then the base station can customize the functionality of the mobile agents to perform the mobile user's tasks remotely.

Finally, we would like to obtain more users experiences in browsing web documents using our system by doing user satisfaction survey. The survey should encompass with data availability on the transmission sequence of organizational units in a web document, accessibility across different networks and comments on multi-resolution transmission and

web browsing mechanism. In addition, we can adopt online usage profiling tool to request users ranking the importance on each organizational unit in a web document and feedback into our system after finishing the transmission of the web document. Combining the contribution on users ranking and information content in the web document, we can improve the performance on multi-resolution transmission and browsing mechanism and then provide more users interesting and content-bearing organizational units in the web document to users earlier.

Appendix A

Performance Study on Different Client/Server Models

We report to our experimental evaluations on the performance on various middle-tier infrastructures including *Common Gateway Interface* (CGI), *Servlet*, *Common Object Request Broker Architecture* (CORBA), and *Remote Method Invocation* (RMI) in accessing a back-end database. Our experimental framework is based on the Wisconsin Benchmark [14]. We classify the benchmark queries into two sets: read-only queries and updates queries and we compare the performances of each middle-tier infrastructure in evaluating different groups of queries.

In our experimental model, the client, the web server, and the database server all reside in different machines. The client and the web server run on Sun UltraSparc 1 platform, equipped with 64MB of memory. An Oracle database server, running Version 7, is used in our experiments, residing in a Sun Enterprise 3000 machine. The Enterprise 3000 machine is configured with two 168MHz UltraSparc processor and 1GB of memory. All the machines are connected via campus-wide Ethernet.

Table 6.2 of Chapter 6 lists the set of Wisconsin benchmark template queries that we have experimented in our evaluation. The set contains a total of 20 queries and we have classified the queries into two groups: read-only queries and update queries. The set of read-only queries contains a total of 16 queries while the set of update queries

contains a total of 4 queries. We measure the amount of time (in seconds) required to execute each query from the moment the query is initiated at the client to the moment the results of the query are returned to the client. We experiment the queries under four different infrastructures, including CGI, Servlet, CORBA, and RMI. The CGI programs are implemented using Perl 5, while both Servlet and RMI are implemented using Java 1.1.6. For CORBA, we employed the Visibroker 3.4. Database access is via Oracle JDBC driver 7.3.4.

Each experiment is repeated 30 times and the average time spent over the 30 trials is measured. To conduct a comparative study among the various infrastructures, it would be more appropriate to compare the ratio of time spent by the various models against the traditional CGI model, thus allowing an easier comparison of their "relative" performance. This is characterized by the *speedup* metric.

In this experiment, we look at the performance on the different queries under various infrastructures. In Figure A.1, we show the speedup from client to web server (denoted C-S) in first column and the speedup from web server to database server (denoted S-D in the second column. We observe that CORBA and RMI, in general produce the best performance. Servlet achieves a better performance than CGI, but performs slightly worse than CORBA or RMI. The superior performance on Servlet over CGI is commonly known since the web server will fork a new process for each CGI invocation, resulting in a large context-switching overhead.

We observe that the C-S overhead of Servlet is much higher than that of CORBA or RMI. In CORBA or RMI, a web client only needs to contact the web server once to download web documents. Once the web document is loaded, the web client only needs to communicate with the CORBA or RMI server to retrieve the database content. That explains why C-S overhead is much smaller in CORBA or RMI. By contrast, the S-D overhead of Servlet is much lower than that of CORBA or RMI. This is because every retrieval of database content will have to go through the CORBA or RMI server,

constituting extra overhead. To give a clearer picture of the performance in **S-D** response time, we re-plot some diagrams of Figure A.1 in Figure A.2 in a scale to highlight the speedup differences for the majority of queries, as CGI is exceedingly slow in a small number of the queries, driving up the speedup to a very large value. In Figure A.2, it is interesting to observe that the speedup of some of the queries for large database is close to and even slightly lower than 1, indicating that CGI is performing as well as the rest. A closer investigation reveals that these query results in very large server to database response time in all models. The overhead of CGI invocation vanishes compared to the actual query execution time by the database server.

One could also observe from Figure A.1 that the performance on CORBA and RMI is very close. CORBA seems to perform slightly better than RMI for more of the queries, but it loses out in others, including all update queries. This indicates that the performance difference between them is query-dependent and not quite conclusive. We believe this is probably due to the particular implementations of the CORBA and RMI packages that we used.

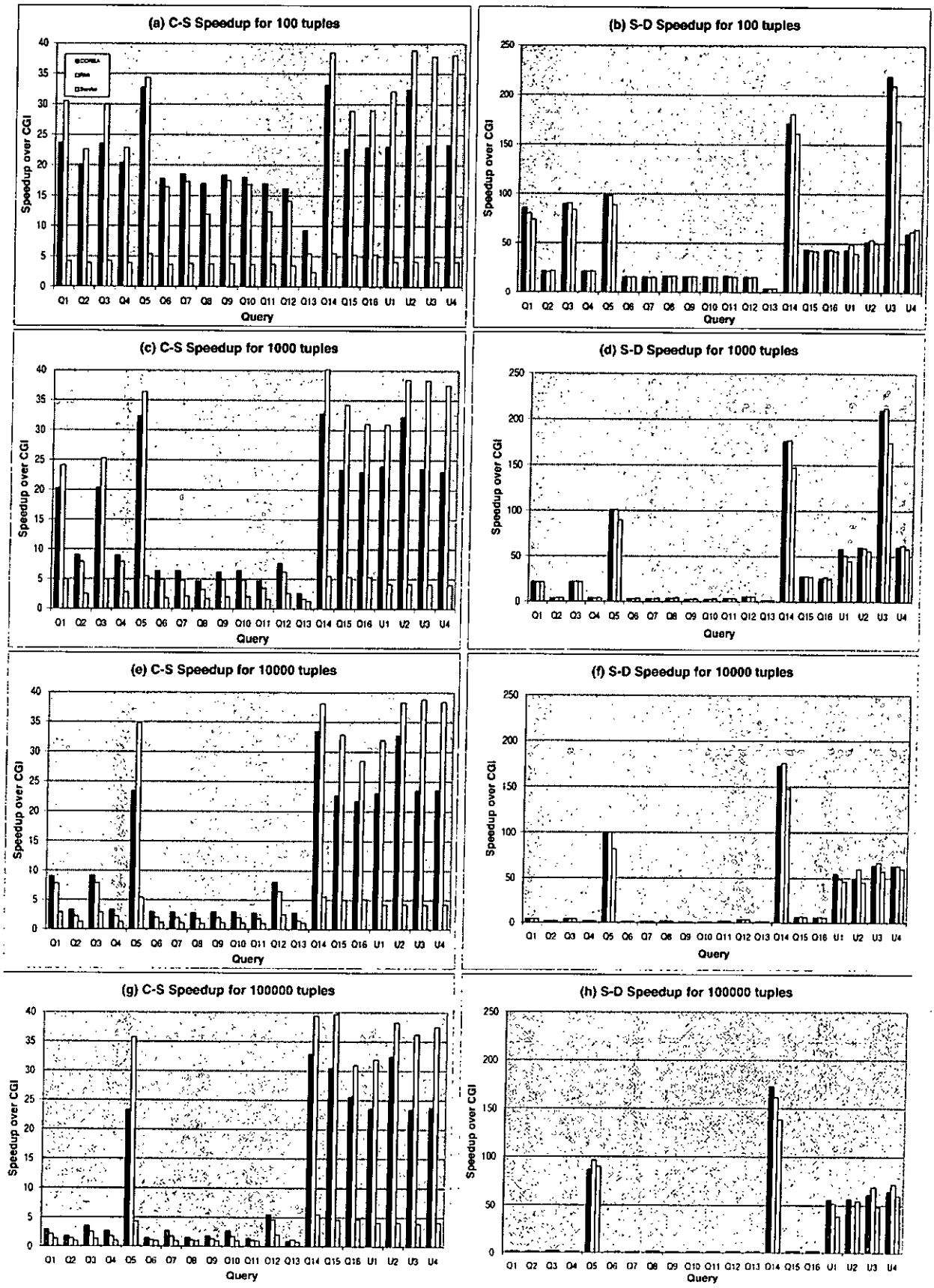

Figure A.1: Performance on Wisconsin queries with varying client/server models

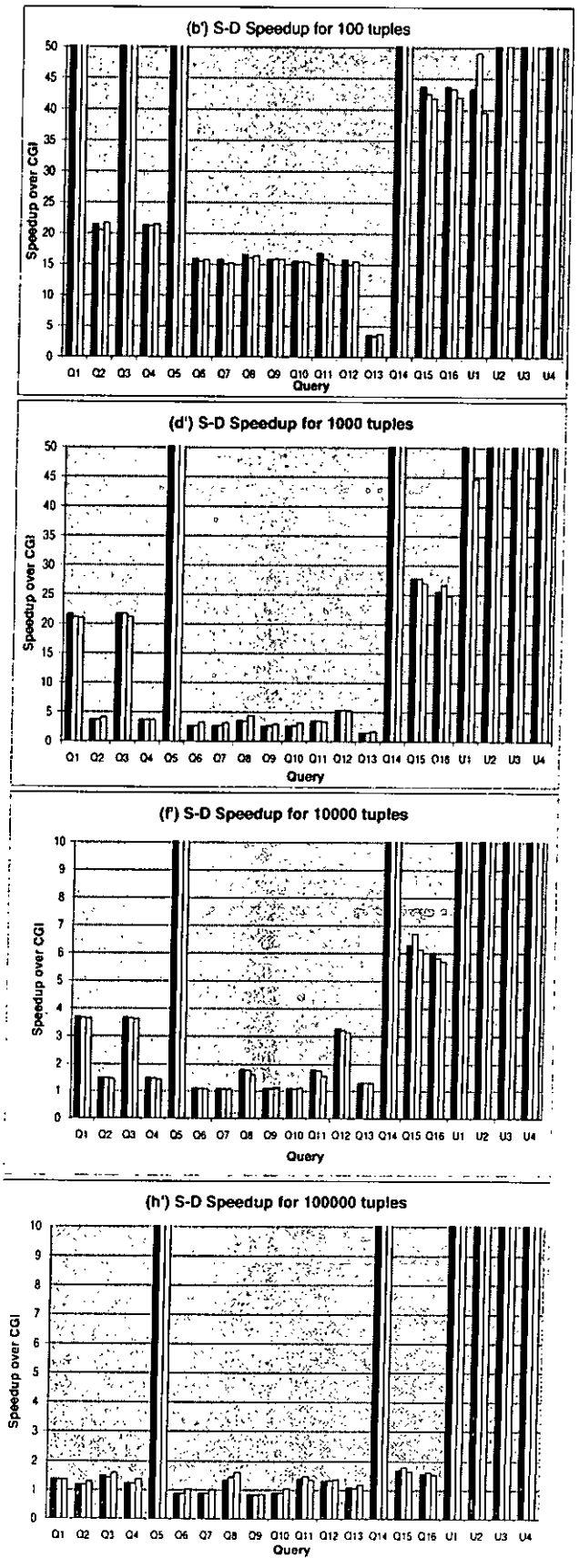

Figure A.2: Performance on Wisconsin queries with difference in S-D of client/server models

Bibliography

[1] Sharon Adler, Stephen Deach, and et al. Extensible Stylesheet Language (XSL) Version 1.0, March 2000. Available at http://www.w3.org/TR/xsl.

[2] Martin F. Arlitt and Carey L. Williamson. Internet Web Servers: workload characterization and performance implications. In *IEEE/ACM Transaction. Networking*, volume 5, pages 631–645, October 1997.

[3] Harini Bharadvaj, Anupam Joshi, and Sansanee Auephanwiriyakul. An Active Transcoding Proxy to Support Mobile Web Access. In *Proceedings of the Seventeenth IEEE Symposium on Reliable Distributed Systems*, pages 118–123, 1998.

[4] Jon Bosak. XML, Java, and the future of the web, November 1996.
Available at http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.html.

[5] Ron Bourret, Christof Bornhövd, and Alejandro P. Buchmann. A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Databases. *Advanced Issues of E-Commerce and Web-Based Information Systems, 2000. WECWIS 2000. Second International Workshop*, pages 134–143, 2000.

[6] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. eXtensible Markup Language (XML) 1.0, 10th February 1998. Available at http://www.w3.org/TR/REC-xml.

[7] Cerium Component Software: XML Servlet, 2000.
Available at http://technet.oracle.com/tech/xml/xsql_servlet/.

[8] David Chess, Benjamin Grosof, Colin Harrison, David Levine, and Colin Parris. Itinerant Agents for Mobile Computing. Technical report, IBM Research Report, IBM Research Division, 1995.

[9] Pi-Yu Emerald Chung, Yennun Huang, Shalini Yajnik, Deron Liang, Joanne C. Shih, Chung-Yih Wang, and Yi-Min Wang. DCOM and CORBA Side by Side, Step by Step, and Layer by Layer. *C++ Report*, 10(1):18–30, January 1998.

[10] Paolo Ciancarini, Fabio Vitali, and Cecilla Mascolo. Managing Complex Documents Over the WWW: A clase Study for XML. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):629–638, 1999.

[11] James Clark. XSL Transformations (XSLT) Version 1.0, November 1999. Available at http://www.w3.org/TR/xslt.

[12] William R. Cockayne and Michael Zyha. *Mobile Agents.* Manning Publications Co., 1997.

[13] Jonathan Dale and David C. DeRoure. A Mobile Agent Architecture for Distributed Information Management. In *Proceedings of the International Workshop on the Virtual Multicomputer*, March 1997.

[14] David J. DeWitt. *The Benchmark handbook : for database and transaction processing systems.* Morgan Kaufmann, 1991.

[15] Stefan Fűnfrocken. How to Integrate Mobile Agents into Web Servers. In *Proceedings of IEEE 6th Workshop on Enabling Technologies : Infrastructure for collaborative Enterprises*, pages 94–99, 1997.

[16] Stefan Fűnfrocken. Integrating Java-based Mobile Agents into Web Server under Security Concerns. In *Proceedings 31st Annual Hawaii International Conference on System Sciences*, volume 7, pages 34–43, 1998.

[17] Roy Fielding, Jim Gettys, and Jeff Mogul. Hypertext Transfer Protocol – HTTP1.1, January 1997. Available at http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[18] William B. Frakes and Ricardo Baeza-Yates. *Information Retrieval : Data Structures and Algorithms.* Prentice-Hall, 1992.

[19] Jade Goldstein, Mark Kantrowitz, Vibhu Mittal, and Jaime Carbonell. Summarizing text documents: sentence selection and evaluation metrics. In *Proceedings on the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 121–128, 1999.

[20] Robert Gray, David Kotz, Saurab Nog, Daniela Rus, and George Cybenko. Mobile Agents: The Next Generation in Distributed Computing. In *Proceedings of IEEE International Symposium on Parallel Algorithms Architecture Synthesis*, pages 8–24, 1997.

[21] Lars Hagen, Markus Breugst, and Thomas Magedanz. Impacts of Mobile Agent Technology on Mobile Communication System Evolution. *IEEE Personal Communications*, pages 56–69, August 1998.

[22] Daniel Hagimont and L. Ismail. A Performance Evaluation of the Mobile Agent Paradigm. In *ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications*, pages 306–313, November 1999.

[23] Colin Harrison, David Chess, and Aaron Kershenbaum. Mobile Agents: Are they a Good Idea? Technical report, IBM Research Report, IBM Research Division, 1995.

[24] Barron C. Housel, George Samaras, and David B. Lindquist. WebExpress : a client/intercept based system for optimizing web browsing in a wireless environment. In *ACM Mobile Networks and Applications*, volume 3, pages 419–431, 1998.

[25] ISO/IEC. *Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML).* Amendment 1., 1988. International Organization for Standardization.

[26] Anthony D. Joseph, Joshua A. Tauber, and M. Frans Kaashoek. Mobile computing with the Rover toolkit. *IEEE Transactions on Computers*, 46(3):337–352, March 1997.

[27] Anupam Joshi, Sanjiva Weerawarana, and Houstis E.N. On disconnected browsing of distributed information. In *Seventh International Workshop on Reseach issues in Data Engineering*, pages 101–107, 1997.

[28] Neeran M. Karnik and Anand R. Tripathi. Design Issues in Mobile-Agent Programming Systems. *IEEE Concurrency*, 6(6):52–61, July 1998.

[29] Danny B. Lange and Daniel T. Chang. IBM Aglets Workbench - Programming Mobile Agents in Java. In *White Paper, IBM Corporation, Japan*, August 1996.

[30] Danny B. Lange and Mitsuru Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, pages 88–89, March 1999.

[31] Hong Va Leong, Dennis McLeod, Antonio Si, and Stanley M. T. Yau. Multi-resolution Transmission and Browsing in Mobile Web. In *CIKM Workshop on Web Information and Data Management*, pages 13–16, 1998.

[32] Hong Va Leong, Dennis McLeod, Antonio Si, and Stanley M. T. Yau. On Supporting Weakly-Connected Browsing in a Mobile Web Environment. In *the 20th International Conference on Distributed Computing Systems*, pages 538–546, April 2000.

[33] Hong Va Leong and Antonio Si. Data Broadcasting Strategies over Multiple Unreliable Wireless Channels. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 96–104. ACM, 1995.

[34] Mika Liljeberg, Timo Alanko., Markku Kojo, Heimo Laamanen, and Kimmo Raatikainen. Optimizing world-wide web for weakly connected mobile workstations: An indirect approach. In *2nd International Workshop on Services in Distributed and Networked Environments*, 1995.

[35] Anselm Lingnau and Oswald Drobnik. An infrastructure for mobile agents: Requirements and architecture. In *Proceedings 13th DIS Workshop, Orlando, Florida*, September 1995.

[36] Eve Maler and Jeanne El Andaloussi. *Developing SGML DTDs From Text to Model to Markup.* Prentice Hall, 1996.

[37] Eve Maler, Steve DeRose, and et al. XML Pointer Language (Xpointer) Draft 6, December 1999. Available at http://www.w3.org/TR/xptr.

[38] Eve Maler, Steve DeRose, and et al. XML Linking Language (XLink) Draft 21, February 2000. Available at http://www.w3.org/TR/xlink.

[39] Paolo Merialdo. ACM SIGMOD Record: XML Version, 2000.
Available at http://www.acm.org/sigs/sigmod/record/xml/.

[40] Jeffrey C. Mogul, Fred Douglis, Anja Fledmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *Proceedings of the ACM SIGCOMM*, pages 181–194, 1997.

[41] Object Management Group (OMG). Mobile Agent System Interoperability Facilities Specification. Technical report, Object Management Group, Framingham, Mass, 1997. Available at ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf.

[42] Oracle Corporation. Oracle XML SQL Utility for Java, 2000.
Available at http://technet.oracle.com/tech/xml/oracle_xsu/.

[43] Oracle Corporation. Oracle XSQL Servlet, 2000.
Available at http://technet.oracle.com/tech/xml/xsql_servlet/.

[44] Andy Oram and Linda Mui. *CGI Programming on the World Wide Web*. O'Reilly, 1996.

[45] Robert Orfali and Dan Harkey. *Client/Server Programming with Java and CORBA*. Addison-Wesley, 1997.

[46] Venkata N. Padmanabhan and Jeffrey C. Mogul. Improving HTTP Latency. In *The Second International WWW Conference*, volume 28, pages 25–35, December 1994.

[47] Michale O. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. *Journal of the ACM*, 36(2):335–348, Apr. 1989.

[48] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.0 Specification, 18th December 1997. Available at http://www.w3.org/TR/REC-html40.

[49] Luigi Rizzo. Software FEC in computer communications.
Available at http://info.iet.unipi.it/~luigi/fec.html.

[50] Luigi Rizzo. Effective Erasure codes for Reliable Computer Communication Protocols. *ACM Computer Communications Review*, pages 24–36, April, 1997.

[51] Luigi Rizzo and Lorenzo Vicisano. RMDP: an FEC-based Reliable Multicast protocol for wireless environments. *Mobile Computing and Communications Review*, 2(2), 1998.

[52] Dan Rubenstein. Increasing the Functionality and Availability of Reed-Solomon FEC Codes: a Performance Study. Technical report, Department of Coumputer Science, University of Massachusetts, August, 1998.
Available at http://www-net.gaia.cs.umass.edu/~drubenst/software.html.

[53] Mahadev Satyanarayanan, James J. Kistler, P. Kumar, and M.E. Okasaki. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4):447–459, Apr 1990.

[54] Bill N. Schilit, Fred Douglis, David M. Kristol, Paul Krzyzanowski, James Sienicki, and John A. Trotter. TeleWeb: Loosely connected access to the world wid web. In *The Fifth International World Wide Web Conference*, 1996.

[55] Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David Dewitt, and Jeffrey F. Naughton. Relational Database for Querying XML Documents: Limitations and Opportunities. *Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, Edinburgh, Scotland, UK*, pages 302–314, 1999.

[56] Antonio Si, Hong Va Leong, and Stanley M.T. Yau. Maintaining Page Coherence for Dynamic HTML Pages. In *Proceedings of the 1998 ACM Symposium on Applied Computing*, pages 767–773, 1998.

[57] Simon E Spero. Analysis of HTTP Performance Problems, 1994.
Available at http://www.ibiblio.org/mdma-release/http-prob.html.

[58] Sun Microsystems. Java remote method invocation specification, October 1998. http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html.

[59] Brian E. Travis and Dale C. Waldt. *The SGML Implementation Guide : A buleprint for SGML migration.* Springer-Verlag, 1995.

[60] ObjectSpace Voyager. ObjectSpace Voyager Core Technology, 1997. Available at http://www.objectspace.com/developers/voyager/white/VoyagerTechOview.pdf.

[61] WAP Forum. Wireless Application Protocol Architecture Specification. Technical report, Wireless Application Protocol Forum Ltd., 1998. Available at http://www1.wapforum.org/tech/terms.asp?doc=SPEC-WAPArch-19980430.pdf.

[62] Stanley M. T. Yau. Multi-resolution Browsing of XML Documents in a Distributed Agent Environment. *The Second ACM Hong Kong Postgraduate Research Day*, 1999.

[63] Stanley M. T. Yau, Hong Va Leong, Dennis McLeod, and Antonio Si. On Multi-resolution Document Transmission in Mobile Web. *ACM SIGMOD Record*, 28(3):37–42, 1999.

[64] Stanley M. T. Yau, Hong Va Leong, and Antonio Si. Multi-resolution Web Document Browsing in a Distributed Agent Environment. *The Second International Conference on Mobile Data Management (MDM '2001)*, pages 279–281, 2001.