WWW-Database Integration via Mobile Agents

By

Quang M. Trinh

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Computer Science
University of Manitoba
Winnipeg, Manitoba

0-612-80064-4

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES
*****
COPYRIGHT PERMISSION PAGE

WWW-DATABASE INTEGRATION VIA MOBILE AGENTS

BY

Quang M. Trinh

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree

of

MASTER OF SCIENCE

QUANG M. TRINH ©2001

For my parents Minh Trinh and Kim Do, and my brother, Hai Trinh.

Acknowledgments

This thesis would have not been completed without the guidance, assistance, and support of two people: Dr. Randal Peters and my advisor, Dr. Ken Barker. I would like to thank Dr. Peters for the encouragement and financial assistance he provided. I would like to thank Dr. Barker for his constant inspiration and support. Lastly, and most importantly, I would like to thank my family, my parents and my brother, for their support and encouragement throughout the years. Without them, this work would not have been possible.

Abstract

This thesis studies the integration of two current topics in the Database Research and Development; namely, WWW-Databases and Mobile Agents. This thesis uses the Web as a medium and agents to locate and deliver information by exploring how to exploit multi-agent system in a heterogeneous database system. By using metadata, Mobile Agents can effectively provide users access to multiple data sources on the Web. The goal is develop techniques so component databases can be added or removed dynamically thereby creating a flexible system model. Another contribution of this research is that the data transported by the agents are presented in the eXtensible Markup Language (XML) format, a markup language that has been widely used as metadata standard. XML is a good choice because it not only support enriched document structures but it also formats data results to support Business To Business (B2B) data exchange. By combining mature existing technologies with new technologies this architecture can support legacy systems while deploying the next generation of software. The key question to be addressed by this thesis is:

- How can mobile agents facilitate data exchange between various data sources while protecting the sources autonomy and providing the user with a globally consistent date view?

This is done while ensuring that:

- The addition and removal of component databases is done dynamically and
- Data exchanged on the WWW occurs while supporting existing applications.

Table of Contents

List of Figures

List of Tables

List of Terms

ACE	Agent-based Computational Economics
ASDK	Aglets Software Development Kit
API	Application Programming Interface
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
B2B	Business To Business
DDBMS	Distributed Database Management System
DTD	Document Type Declaration
EARTH	Efficient Architecture for Running THreads
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IIS	Internet Information Server
JDBC	Java Database Connectivity
JDK	Java Development Kit
MGA	Multi-population Genetic Algorithms
OO	Object Oriented
RDMBS	Relational Database Management System
RMI	Remote Method Invocation
SGA	Single-population Genetic Algorithms
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
W3C	World Wide Web Consortium
WWW	World Wide Web
WWW-DIMA	World Wide Web Database Integration via Mobile Agents
XML	eXtensible Markup Language

Chapter 1
Introduction

1.1 Introduction

The idea of using the World Wide Web (WWW) to provide access to the global information resources has been an attractive research area. However, with the advances in technologies and the exponential growth of the number of resources on the WWW, not only providing access but also managing multiple information sources is necessary. Today, combining the WWW and Relational Database Management Systems (RDBMS) [5] is a dominant technology for storing and processing data on the WWW, but there are drawbacks of using such a centralized RDBMS model. The two critical and most important drawbacks are:

- Performance: degradation in performance as the number of users increases and
- Reliability: failure of the centralized DBMS is the important (in the system) means no information is available

Distributed Database Management Systems (DDBMS) were designed to overcome the drawbacks of RDBMS albeit with some additional complexity (e.g. consistency and availability issues). A DDBMS is logically a single database system that is fragmented into several fragments where each fragment is stored in different machines on the network. The focus of a DDBMS is on the distribution and location of both data and processing so better performance can be achieved and to maximize accessibility. DDBMS can also be used for data integration of heterogeneous databases. Database integration encourages the use of emerging technologies (for example, relational databases and mobile agents [35]) and, at the same time, enables users to correlate information from multiple sources. Database integration sometime requires interoperability between database systems. Support for interoperability requires many additional requirements such as: platform and DBMS independence, schema semantic, data exchanging language, *etc.*

In the past 20 years, many client/server-based data access and exchange techniques for DDBMS have been developed. Such techniques include: Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA) and XML-RPC. Although, these techniques are well understood, however, their performance is inadequate.

1.2 Problem Statement

Current network technologies and the WWW permit easy sharing of information; though some techniques may be better than others. Presently, access to the vast amount of information on the WWW requires multiple steps for searching and navigating. This process can be time consuming. This is not to mention whether or not the information presented to the users is consistent. For example, what if the price of a particular round trip is lower in one travel agency than in another, how can this information be presented to the users so that the price is guaranteed to be the lowest?

The key issue to be considered by this thesis is how mobile agents can facilitate data exchange between various data sources on the WWW while protecting the data sources' autonomy and providing the user with a global consistent data view? This should be done while ensuring that

- The addition and removal of component databases dynamically is possible, and
- Data exchange on the WWW occurs while supporting existing applications

1.3 Research Goals

This thesis studies the integration of two cutting edge technologies: WWW-Databases and Mobile Agents [25]. This thesis proposes to use the Web as a medium and agents to locate and deliver information. The design of a multi-agent system in a heterogeneous database environment is explored. The goals of this thesis are:

- To show that through the use of metadata mobile agents can easily and effectively provide users access to multiple data sources on the Web.
- To develop techniques so that component databases can be added or removed dynamically thereby creating a flexible system architecture.
- To make use of the eXtensible Markup Language (XML) [38]. Data transported by the agents are presented using XML, a markup language that has been widely used as an ad-hoc metadata standard. XML is a good choice because not only does XML support enriched document structures but it also formats data results to support Business To Business (B2B) data exchange.
- Integration of existing and emerging technologies. By combining mature existing technologies with new technologies, this architecture can support legacy systems while deploying the next generation of software.

1.4 Organization of the thesis

The remainder of this thesis is divided into seven chapters that provide further details on the problem area, the approach and the related work, evaluation of the prototype and the conclusions. Chapter 2 surveys the background information and the related work. In particular, details of existing related technologies and emerging technologies that are used in the new prototype architecture are discussed in Chapter 2. Also, discussed in Chapter 2 are the benefits and drawbacks of software agents' technology, as well as a comparison between software agents and Object Oriented (OO) programming model.

Chapter 3 demonstrates the use of agents by discussing their use in the solution of computationally intensive problems in the area of computational finance. This chapter begins with a basic introduction to financial markets and then discusses the algorithms for generating a binomial, lattice tree-like structure for pricing security options. The chapter ends with a discussion of the agent-based implementation's overhead and the general communication timing between the agents.

Chapter 4 introduces the key concepts of agent systems. The architecture and implementation of the Aglets Software Development Kit (ASDK), a mobile agent API that was designed and developed by IBM are discussed in detail. Included in this discussion are aglets context, aglets communication, aglets migration, aglets event handlers and current security features supported by the ASDK. Chapter 4 ends with a general summary of the advantages of agent systems.

In Chapter 5, the proposed agent-based architecture for database integration is introduced and described in detail. Definitions of, and interactions between major, components in the architecture are examined individually. Supporting tools and software components are identified and the details of the implementation of the architecture are described. In particular, the details of the implementation and the interactions of the components are described at a low level abstraction. The chapter then provides detailed descriptions of the operations supported by the architecture and concludes with a discussion of optimization issues related to the architecture.

Chapter 6 provides an evaluation of the prototype described in Chapter 5. The evaluation is performed mainly with respect to Applets, which use a traditional client/server model. To understand the behavior of the prototype under heavy and/or light network traffic, the timing observations of the prototype were done at peak hours and off-peak hours. Chapter 6 ends with some timing comparisons and explanations of the results by comparing the prototype and the client/server Applet model.

Chapter 7 summarizes the recent publications related to the work presented in this thesis. These recent publications provide frame-works for comparisons with the work presented here. Such comparisons are also made in this chapter.

Conclusions and future research directions for the thesis are presented in Chapter 8. This chapter summarizes the work done, lists the contributions made and provides some suggestions for future research directions.

Chapter 2

Background Material

2.1 Introduction

In the early days of the WWW, accessing a single database on the WWW was a big accomplishment that brought the WWW "alive" by supporting dynamic content. The Common Gateway Interface (CGI) was one of the first approaches used to provide users access to dynamic (i.e. non pre-specified) data (e.g. from a database) on the WWW. Used in conjunction with the HyperText Markup Language (HTML), CGI is still widely used, but demands for better accessing techniques increase as the web becomes larger and more complex. With the vast amount of information available on the WWW, the bottleneck on the network increases as more users' request for information increases. Currently, distribution of data, processing and access to multiple data sources on the WWW are essential for building the next generation of web databases technology. This chapter introduces technologies that will be used in the research presented in this thesis to develop a new technique for accessing multiple data sources on the WWW.

2.2 A Distributed DataBase Management System (DDBMS) Model

A DDBMS is logically a single DBMS that is partitioned into several fragments where each fragment is stored on different machines in the network. The data model for a DDBMS is designed so that each data fragment is stored in a machine on the network where data in the fragment will be frequently accessed. Since data in a distributed environment are stored in multiple locations, improvements on reliability and performance can be achieved.

- Reliability: since data are distributed on multiple locations, failure of any location does not affect the availability of data from the other locations. This is possible because, in a distributed environment, all data and services at each location are independent of each other.
- Performance: storing data where it is accessed the most leads to faster response time and therefore increases throughputs and improves the overall performance of the system

A current motivation for the use of DDBMSs is the competition in global business operations among corporations. With the WWW as a medium, corporations are competing against each other to deliver information to their customers in the most efficient manner. While the concentration of DDBMSs is on the delivery of information, DDBMS can also be used for integrating legacy database technologies. Not only does the integration process encourage the use of emerging technologies by combining resources, it also enables users to correlate information from multiple sources. In general, DDBMSs can be classified into two categories: *homogeneous distributed* and *heterogeneous distributed* database management system.

2.2.1 Homogeneous Distributed Databases

A homogenous distributed database system is a distributed database system that integrates DBMS of only one-type. Homogeneous distributed database can be further divided into two groups: *autonomous* and *non-autonomous* homogeneous distributed databases.

- An autonomous homogeneous distributed database system is a distributed database system where both data and services in all nodes are independent of each other. This type of DDBMS is most appropriate for supporting independent data requests from the individual component databases.
- A non-autonomous homogeneous distributed database system is a distributed database system where all data and services are managed by one DDBMS through a global schema. This global schema is defined by combining the local schemas of all the component databases in the system. In particular, when a query is posed to the global schema, the query translator, a component of the DDBMS, translates the particular query to sub-queries for each of the component databases involved. Each sub-query is then executed locally using the local schema of the component database. The result of the original query is the combined query results of the sub-queries from the component databases presented in a format consistent with the global schema. Potentially, the global schema introduces two problems. First, any definition discrepancies between the global schema and the component schema must be resolved immediately. Second, to preserve data consistency, coordination between component DBMS must be enforced.

2.2.2 Heterogeneous Distributed Databases

A heterogeneous distributed database system is a distributed database system that integrates various types of DBMSs. In fact, a heterogeneous distributed database system is very similar to non-autonomous homogeneous distributed database system, except that a heterogeneous distributed database system also integrates multiple types of DBMSs. All data requests submitted locally to the component databases are processed using the local schema, and likewise, all data requests submitted from a remote location are processed using the global schema.

2.2.3 Distributed Query Processing

When constructing a query execution plan, and in addition to all the functions of a centralized query processor, a distributed query processor must take the following two factors into consideration: the network bandwidth and the amount of data to be transferred. In general, when a given query is submitted to a DDBMS, the following steps are taken [9]:

- The raw query is parsed and checked for valid syntax (to ensure that the query is valid for the particular DBMS).
- The DDBMS uses the global schema to construct functionally equivalent query-processing plans.
- The query optimizer compares the equivalent query plans, estimates the cost for each plan, and returns the optimal query plan.
- The optimal plan is executed and the result is returned to the user.

2.3 WWW-Databases

Traditional HTTP applications supported the distribution of only static hypertext-based information. Web-databases (and related tools) support the distribution of dynamic hypertext-based information. Although, there are many ways to access the contents of databases via the web, the two most common approaches are: the CGI and the Java Database Connectivity (JDBC) interface.

In general, there are three main components of any Web-database applications: the database itself, the information server (or the web-server) and a database interface (for example CGI or JDBC). Depending on the complexity of the applications, most Web-database applications involve either just server-side processing or both server-side and client-side processing. Server-side processing performs tasks on the server-side such as accessing the database and business-logic rules. The client-side processing, on the other hand, performs client-oriented tasks such as input validation. However, combining both client-side and server-side processing is essential to improve the overall performance of the Web-database application. That is, client-side processing can be used to validate or to preprocess users input(s) before the request is submitted to the server for processing.

Using the WWW as a medium and databases to store information, corporations create more opportunities to make profits by extending their services to the global market. In general, there are two categories of services on the WWW: secure and non-secure services. Secure services are secure electronic transactions such as online banking, airline reservations, *etc.* Non-secure services include all services that do not require protection such as providing general information about a particular corporation or about a particular product, *etc.* Another important use of the WWW is for Business To Business (B2B) information sharing between corporations. Besides protecting the information, another major problem with information sharing is that a common data exchange format must be defined or agreed to between corporations ahead of time.

2.4 XML

The eXtensible Markup Language (XML) [38] is a recent standard markup language developed by the World Wide Web Consortium (W3C) [36] in 1996. XML is a subset of the Standard Generalized Markup Language (SGML) [31], which was also developed by the W3C. The primary purpose of XML is to provide a user-defined and application-independent data format that supports data exchange between applications and/or data sources on the WWW. Unlike the relational data model, XML uses *elements, tags* and *attributes* to describe the data structure and semantics of the data. In general, all XML documents must be *well-formed.* A well-formed XML document is a document that satisfies the following two criteria:

- All elements and tags in the document must match exactly (case sensitive) and
- All elements and tags in the document are nested properly

```
<EMPLOYEE>
        <ID>12345</ID>
        <FNAME>Jennie</FNAME>
        <LNAME>La</LNAME>
        <ADDRESS>411 Cumberland Ave</ADDRESS>
        <PHONE>1-204-123-4567</PHONE>
        ...
</EMPLOYEE>
<EMPLOYEE>
        <ID>12346</ID>
        <FNAME>Tammy</FNAME>
        <LNAME>Pearson</LNAME>
        <ADDRESS>256 Morley Ave</ADDRESS>
        <PHONE/>
        ...
</EMPLOYEE>
```

Figure 2.1: A Well-formed XML document

Figure 2.1 shows an example of a well-formed XML document. In particular, <EMPLOYEE> is the beginning element and </EMPLOYEE> is the ending element of an employee record. <ID>, <FNAME>, <LNAME>, *etc.* are the beginning elements and </ID>, </FNAME>, </LNAME>, *etc.* are the ending elements that describe employee records. In the case of the second employee record, the element <PHONE/> indicates that the phone element is empty and does not have any value associated with it.

In addition to being well-formed, an XML document may be *validated* if the particular document has a *Document Type Declaration* (DTD) associated with it. The first purpose of the DTD is to allow a user to define the document's structure, legal elements and attributes. It also specifies the order of the elements and attributes that will appear throughout the document. With the use of a pre-agreed-upon DTD, independent organizations can share their business data without any modifications to their current infrastructure system, even with legacy or incompatible systems. While DTDs are not required for XML documents, they can be defined externally or internally to the document.

```
<?xml version="1.0"?>
        <!DOCTYPE EMPLOYEELIST [
                <!ELEMENT EMPLOYEE (ID, FNAME, LNAME, ...)>
                <!ELEMENT ID            (#PCDATA)>
                <!ELEMENT FNAME         (#PCDATA)>
                <!ELEMENT LNAME         (#PCDATA)>
                ...
        ]>
< EMPLOYEELIST >
        <EMPLOYEE>
                <ID>12345</ID>
                <FNAME>Jennie</FNAME>
                <LNAME>La</LNAME>
                ...
        </EMPLOYEE>
        ...
</ EMPLOYEELIST >
```

Figure 2.2: A Well-formed XML document with an internal DTD

The first line in Figure 2.2 is the header of the XML document. This indicates to the XML parser that the document is an XML document. The second line in Figure 2.2 defines the document type. That is, EMPLOYEELIST is the document type. The third line defines that there exists element(s) EMPLOYEE in the EMPLOYEELIST document. The same line also defines ID, FNAME, LNAME, etc. as elements within each EMPLOYEE element. The fourth line in Figure 2.2 indicates that the element ID is of type PCDATA (Parse Character DATA). Collectively, line 2 through to the line defining an <EMPLOYEELIST> constitute an internal DTD.

```
<?xml version="1.0"?>
<!DOCTYPE EMPLOYEELIST [
        <!ELEMENT EMPLOYEELIST (EMPLOYEE)+>
                <!ELEMENT EMPLOYEE (ID, FNAME, LNAME, ...)>
                <!ELEMENT ID            (#PCDATA)>
                <!ELEMENT FNAME         (#PCDATA)>
                <!ELEMENT LNAME         (#PCDATA)>
                ...
]>
```

Figure 2.3: An external employee list DTD file

Figure 2.3 shows an external DTD of the EMPLOYEE element within the EMPLOYEELIST document. Figure 2.4 below shows an employee list XML document with a reference to an external DTD file. The second line in Figure 2.4 indicates the external DTD reference to the file `eList.dtd` (shown in Figure 2.3).

```
<?xml version="1.0"?>
<!DOCTYPE EMPLOYEELIST SYSTEM "eList.dtd">
< EMPLOYEELIST >
        <EMPLOYEE>
                <ID>12345</ID>
                <FNAME>Jennie</FNAME>
                <LNAME>La</LNAME>
                ...
        </EMPLOYEE>
        ...
</ EMPLOYEELIST >
```

Figure 2.4: A Well-formed XML document with an external DTD reference

The second purpose of a DTD is that one can also use a DTD to validate an XML document. The validation process is done through an XML parser. There are free XML parsers available in just about every programming language [40, 41, 37, 21, 32, 39].

Figure 2.5 below shows the validation process for an XML document. In general, XML was chosen for, and is significant in this thesis because it presents a format that both users and machines can understand.

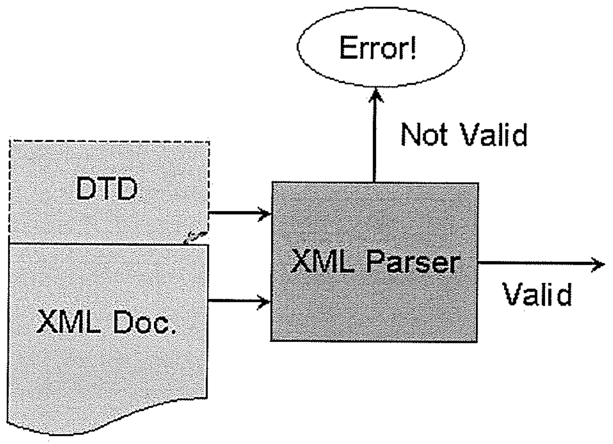

Figure 2.5: XML document validation process

2.5 JDBC

Java Database Connectivity (JDBC) [18] is an Application Programming Interface (API) developed by Sun. The purpose of the JDBC API is to provide a standard for access to relational databases from Java programs. Since the JDBC API is written purely in Java [20], it supports platform-independent applications with virtually any DBMS as the back-end. JDBC technology has been widely accepted so many database vendors have endorsed it and provide their own drivers to their backend DBMSs. A list of vendors who have endorsed the JDBC technology and their supporting drivers can be found in [19].

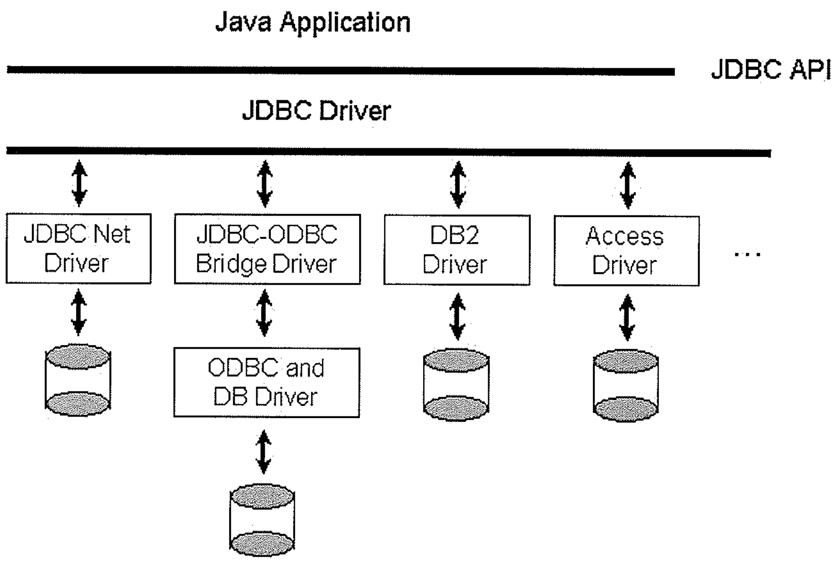

Figure 2.6: Overview of JDBC architecture

The JDBC API supports both two-tier and three-tier models. In both models, an appropriate JDBC driver for the particular DBMS product must be loaded before a database connection is established between the application and the DBMS itself. After a database connection is established, the application can use the provided API to send

Structured Query Language (SQL) statements to manipulate the data in the database. Figure 2.6 provides an overview of the JDBC/ODBC architecture.

2.6 Software Agents

A software agent system is a software engineering paradigm that supports the concepts of being autonomous, learning and supporting collaboration, and communication between objects. Typically, an agent is an object or a single component in an application domain that was designed for some specific task. For example, an agent can be as simple as a sub-routine or can be as complicated as a daemon manager. An agent can also be *intelligent* if the agent can handle tasks with minimal intervention from users. Agents can either learn from past experiences or from the training of some knowledge-based system and they use the knowledge for references when making decisions about what to do next.

In general, software agents are categorized into two types: *stationary agents* and *mobile agents*. Stationary agents operate only in the environment where they are created and are appropriate for solving simple and isolated problems. All stationary agents support the autonomous, learning and collaboration, and communication properties. Mobile agents are like stationary agents, but in addition, they have the ability to move between hosts under their own control to obtain better performance. For more complicated problems, more than one agent may be needed to solve the problems effectively.

The following properties are defined for all software agent systems.

- Autonomous

  In addition to inheriting all the Object Oriented (OO) properties, agents are independent of each other and are typically designed with some specific tasks in mind.

- Communications and Collaborations

  Agents can communicate and collaborate with one another. Communication between agents is done primarily through message passing. Collaborations, on the other hand, are more general having agents (from the same domain or different domains) working together by exchanging data. This results in a multi-agent system.

- Mobility

  Agents with mobility properties can migrate between machines on the network. When transporting between servers, all agent states are transported with the agents.

Intelligence and mobility is a good combination for many applications, especially for networks and distributed applications. For example, in distributed query processing, it is important to reduce the amount of data communication between the client and the server. One way of doing this is to migrate the agent with the query to where the data is located. Upon arrival, the agent can extract the data and bring back only the required data. Working where the data is located minimizes the amount of data that must be transferred on the network and therefore it uses less of the available bandwidth. An alternative, more traditional approach is a permanent connection available from the client to the server. This connection is maintained while the processing is carried out on the server. The connection is terminated only after the data result is delivered to the client. Extensive recent studies [[35], [24], [27], [16], [30]] have shown that in the traditional approach, the performance of the application is directly proportional to the number of clients (or users) and the size of the databases. In particular, the network throughput decreases as the number of users increases.

Currently, there are many available implementations of mobile agent software packages. A listing of documents and source code for these implementations can be found in [28].

2.6.1 Benefits of Software Agents

Over the past 4 years, there have been many significant performance evaluations of the software agent paradigm. Recent publications have shown that software agent paradigm is suitable for distributed processing [14] and software agents have better performance than existing client/server models such as RMI [16] and Applets [30]. Other significant benefits of software agents are:

- Software agents can be used to support real-time application. An example of this is provided by Kotz et al. [23] who uses agents to support a data filtering application on real-time wireless networks.
- Though, software agents increase the load on the server, they can be used to reduce the required network bandwidth. This is especially appealing for wireless environments or low bandwidth networks and or when the server is a cluster of processors and is capable of handling the load. Mobile agents can also be used to support disconnected operations. Mobile agents can be created and sent to the server to do some particular tasks while the original host or owner disconnected from the network. Once the processing is completed, and only when the original host is re-connected, the agents return back to the original host.

2.6.2 Drawbacks of Software Agents

Although, software agents offer certain advantages over conventional OO programming model, there are also difficulties with their use that have not yet been thoroughly investigated:

- To support the agent's mobility, a network of agent servers must be built. This requires an agent server to run on each of the hosts to be used by the agents.

- Security is a major issue in all mobile agent systems. Identification, authentication, and authorization of all agents must be strictly enforced. Other security issues that have not been addressed adequately are how to deny entrance of agents from another or untrusted domain, or how to prevent malicious acts of agents that arrived from another or untrusted domain?
- A centralized authority is necessary for authorization. Since agents are independent objects, there is no central control over the agents. If an agent poses malicious acts, then how can another agent terminate the agent with the malicious acts?

2.6.3 Software Agents vs. OO Models

Since software agents are designed and implemented as objects, both software agents and OO models have very similar properties. Both objects and agents, in their own rights, are individual components and can be added or removed from the system dynamically. Interactions in both models are done primarily by exchanging messages between the components in the system. Both models support data encapsulation and polymorphism. One major difference between the two models is that software agents are objects that have the ability to move between machines on the networks. When moving between machines on the networks, agents carry all their state and code with them. Agents can also be intelligent. That is, agents can be trained to adapt themselves to surrounding environmental changes and make decisions based on past experiences.

2.6.4 Aglets Software Development Kit (ASDK)

The Aglets Software Development Kit (ASDK) [1] is a Java based mobile agent software package that was designed and developed by IBM. ASDK supports the concepts of autonomous, independent and dynamic execution of agents. ASDK is an extension of the Java Development Kit (JDK) [20] and is based on the client/server model. The unique combination of the Java language (platform independent) and dynamic execution makes ASDK appropriate for distributed and parallel applications in heterogeneous environments. ASDK is platform-independent and also supports mobility services and dynamic workload adjustment. Thus, with the ASDK, agents can roam the networks until the task is completed and they can be added or removed dynamically from the application. Other factors supporting for the choice of ASDK are:

- Security

  Current ASDK, Aglets1.2.0, supports a fine grained security model.
- Communication

  Both current and previous versions of ASDK support message passing protocol as a communication mechanism.
- Hyper Text Transfer Protocol (HTTP)

  Both current and previous versions of ASDK support submit and response http requests.
- Agent Transfer Protocol (ATP) [2]

  Both current and previous versions of ASDK support the ATP. ATP is a protocol for transferring agents between networked computers.

Chapter 3:

Architecture of the Aglets Software Development Kit (ASDK)

3.1 Introduction

Aglets (***Ag****ents and App****lets***) [1] is a software agent development kit that adopts the concepts based on everyday human activities [25]. Some agents created with the Aglets Software Development Kit (ASDK) will do what they are told while others are programmed with heuristics to decide on what to do next. The latter requires training from either a knowledge-based system or from past encountered experiences. Unlike other paradigms, aglets can move from one host to another on the network. When moving between hosts, aglets carry their code and state with them. Upon arrival at the destination host, aglets resume their execution from where they left off. Though, there are difficulties yet to be dealt with, it is predicted that software agents will be the next generation of software engineering for application developments.

3.2 Aglets Context (Aglets Server)

Because of their unique ability to transport themselves from one host to another, aglets (or software agents or just agents[1]) require a service provider on every host on the network where they may execute. This service provider is called an *aglet context* or an *aglet server.* Each host on the network can have more than one context, and contexts on the same host are differentiated from each other by unique port numbers. Each aglet context in a host can have a number of aglets and aglets created in one context can be dispatched to another context on the same host or on a different host. Aglets in the same context share the same resources provided by that context.

Figure 3.1 shows two aglet servers in the host named quillaja. The first aglet server is atp://quillaja:4000, which has two aglets residing in it and the second aglet server is atp://quillaja:5000 has one aglet residing in it. The values 4000 and 5000 indicate the port numbers of the two aglet servers.

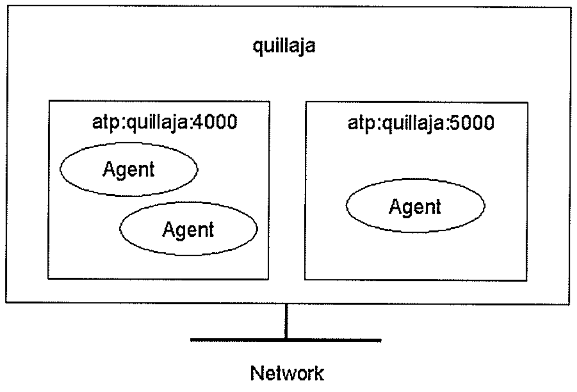

Figure 3.1: Relationship between a host, an aglet, and an aglet server

[1] The term software agents and agents are used interchangeably from this point forward.

3.3 Aglets Proxy

Communication between aglets in the same or in different contexts is done through each aglet's proxies. A proxy is a public interface between the aglet and the rest of the world. A proxy is also a shield that protects aglets from each other. For an aglet to send a message to another aglet, the sender aglet must first obtain the proxy of the recipient aglet. For all stationary aglets, the values of their proxies remain the same for the aglets' entire lives. However, for mobile aglets, the values of their proxies change when the aglets arrived at new aglet servers. To maintain the communication channel, the values of the new proxies must be updated immediately after the aglets' arrive at their new aglet servers. Aglet proxies can be obtained either when aglets are created or by using the aglet name and the aglets' unique identification in a call to get the aglet's proxy. The following shows how aglet proxies can be obtained.

```
AgletProxy ap =
   getAgletContext.createAglet(getCodeBase(), name, parameters);
```

where:

`getCodeBase():`

a function that gets the codebase address of where the aglet code resides

`name:`

the path and name of the server where the aglet's class object is located

`parameters:`

an array of object parameters to be used to initialize the specific aglet to be created

The method `getAgletContext.createAglet(...)` is used to create aglets and to obtain the proxies of the newly created aglets. The method `getProxy(...)`, on the other hand, is used to get an aglet's proxy after the aglet arrives at a new aglet server.

```
AgletProxy ap = getProxy(id, hostAddress);
```

where:

`id:`

the aglet's unique identification

`hostAddress:`

the address of the host where the codebase of the aglet is located. The port number of the aglet server is also included in this `hostAddress`

3.4 Aglets Communication

Aglets communicate with each other by message passing. In particular, messages can be sent using two protocols: a one-to-one communication protocol or a one-to-all communication protocol (also known as broadcast). One-to-one communication options include:

- One way (non-blocking, asynchronous)

  The sender aglet sends a message and continues execution regardless of whether or not the intended recipient aglet gets the message
- Wait-for-reply (blocking, synchronous)

  The sender aglet sends a message and waits for a reply from the intended recipient aglet before continuing execution
- Future reply (non-blocking, synchronous)

  The sender aglet sends a message, continues executing but expects a reply back from the intended recipient aglet at sometime in the future

3.5 Aglet Migration

When aglets move between hosts on the network, they carry all their current state and code with them. This means, upon the arrival at the destination host, aglets can request resources, if needed, and continue on from where they left off. Furthermore, aglets can be programmed to detect the status of hosts on the network and pick the destination host that has the most available resources before moving to the destination host. Once the new host is selected, agents can release resources currently held and prepare for dispatch to the new destination. Figure 3.2 shows the internal process of migrating aglets from one host to another. The following steps take place when an aglet is preparing for migrating to a new destination host.

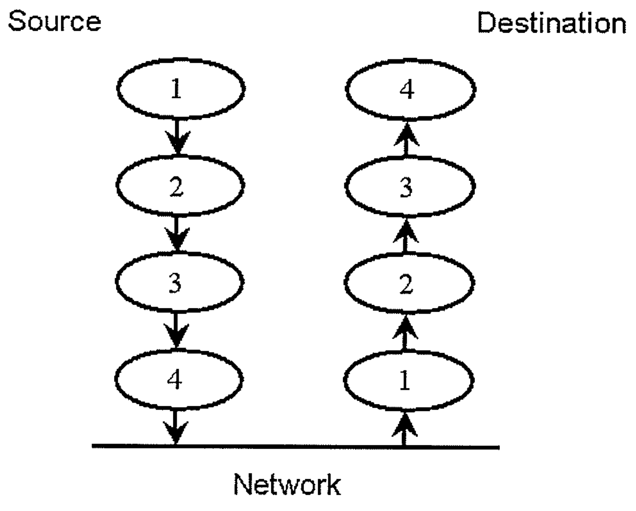

Figure 3.2: The internal migrating process of aglets from one host to another host

At the source (or the current) host:

1. Stop or wait for the current instruction to complete.
2. Serialize all state (including data and code) into "packages".
3. Encode the serialized packages for the particular network transport protocol to be used.

4. Transfer the encoded packages to the destination host.

At the destination host:

1. Receive the packages from the network transport protocol.
2. Decode the received packages.
3. De-serialize the packages into state, and code of the agent.
4. Continue execution where left off.

3.6 Other Aglet Behaviors

Other aglet behaviors are based on the message handling routine and the following three event handlers: the mobility event handler, the cloning event handler and the persistence event handler. Each handler has a event listener. In all cases, these three listeners are added at the beginning of the aglets' execution and removed when the aglets are terminated. Mobility and cloning events are used in distributed and/or parallel computing applications. The mobility handler dispatches or retracts aglets from one host to another host while cloning duplicates aglets resulting in multiple aglets with the same functionalities but different states. The persistence handler stores the serialized aglet (consisting of the aglet's code and state) into secondary storage. Table 3.1 below summarizes the API of the mobility event handler, Table 3.2 summarizes the API of the cloning event handler and Table 3.3 summarizes the API of the persistence event handler.

Table 3.1: Mobility event handler

Mobility Event Handler	
`onDispatching()`	is called when an aglet attempts to dispatch to a remote host.
`onArrival()`	is called when the aglet arrives at a destination host.
`dispatch()`	is called when the aglet dispatches itself to a remote host. This method automatically invokes the `onDispatching()` method
`OnReverting()`	is called when a parent aglet attempts to retract a child aglet from a remote host.

Table 3.2: Cloning event handler

Cloning Event Handler	
`onCloning()`	is called when an aglet attempts to clone itself.
`onClone()`	is called to initialize the state of the cloned aglet. (This method acts as the constructor of the cloned aglet).
`onCloned()`	is called on the cloner after the cloning process is successful. From this point on, both the cloner and the clonee are independent of each other.

Table 3.3: Persistence event handler

Persistence Event Handler	
`Deactivate()`	is called when an aglet temporary halts its current execution for a known period of time (specified in milliseconds). The aglet server will re-activate the deactivated aglet after the duration of the time period.
`Activate()`	is called when an aglet attempts to activate another aglet that has been deactivated earlier.

3.7 Aglet Construction

Aglet construction is very similar to object creation in the OO model. Instead of *constructors* and *destructors*, however, aglets have `onCreation(...)` and `onDisposing()` methods. The `onCreation(...)` method takes an array of objects as arguments. These arguments are used to initialize the aglet. The argument array can either be empty (`null`) or contains a number of arguments. The `onDisposing(..)` method, on the other hand, is automatically invoked when the aglet terminates its execution. The purpose of `onDisposing()` method is to free up allocated resources. After `onCreation(...)` is executed, the method `run()` is automatically invoked on the new aglet and the aglets' execution begins. Aglets can also create child-aglets. This is known as the *parent-and-child* or *master-and-slave* model. The child-aglets, in turn, can also create child-aglets of their own and so on. Aglets and their ancestors or descendants can communicate with each other by using either broadcast or the one-to-one communication protocols described previously.

3.8 Aglet Security

Security is a major concern for all mobile agent systems. All untrusted aglets are capable of committing malicious acts. An example of this is an untrusted agent from one host can dispatch itself to another host and take many of the available resources and/or commit other malicious acts. Theoretically, all agent systems must include all of the following security services, as suggested by Harrison, *et al.* [14]:

- Domain authorization and authentication

  The network domain authorities are responsible for protecting local network resources, allocating and reallocating resources in the domain to local agents and visitor agents. Only those trusted agents (from trusted domains) will get resources. Untrusted agents and/or agents from untrusted domains will be denied both access to the domain and to its resources.

- Host authorization and authentication

  Host authorization and authentication are used to protect the underlying operating system. This includes allocating and reallocating the resources of the host. Resource authorization and authentication are given to agents based on their rights. For example, an agent from a neighbor domain may have more rights than an agent from a domain elsewhere.

- Secure communication

  All communications between agents must be secured. Validating the identity of the sender aglet is required prevent message forgery and certain other malicious acts.

Currently the ASDK does not fully support all these recommended security services. Security features supported by the ASDK are: domain authorization and authentication, host authorization and authentication, and resource authorization and authentication. Resource permissions are set in the aglet's policy file (located in the user's `.aglet/security` folder). This policy file is very similar to that of the Java security policy file. Aglet security defined in the Aglet's policy file supports the following three categories of permissions:

- Code base ATP (Agent Transfer Protocol) permissions
- Code base HTTP (HyperText Transfer Protocol) permissions
- Agent code-based permissions

Depending on where the code base of the aglet is loaded, appropriate security enforcement will be applied to the aglet accordingly. That is, initially, when the code base of an aglet is loaded from a remote context then the code base ATP permissions are applied to the aglet. When the code base of an aglet is loaded from an archive (that is a JAR file) then the code base HTTP permissions are applied to the aglet. When the code base of an aglet is loaded from a local host the agent code base permissions are applied to the aglet. Although, there are three different categories of security permissions, they all include the same set of permissions (e.g.: all three categories have read and write access permissions). Other security features included in the current implementation of ASDK are: file access permissions, network access control permissions, run-time permissions for libraries, aglet run-time permissions, message permissions.

Aside from the difficulties mentioned in Section 2.6.2, software agent's paradigm forces developers to look at problems from a wide perspective rather than just focusing on solving the problems themselves. Some benefits of using the mobile agent paradigm include:

- Reduced network load
- Ease of distributing workload
- Scalability

Chapter 4

Agents in Economics

4.1 Introduction

Agents can be applied to solve problems in many different areas. Well-known areas such as parallel and distributed processing, e-commerce, scientific computing, information retrieval in the WWW as well as new emerging areas such as Agent-Based Computational Economics (ACE). ACE is a relatively new concept, which has recently received the attention of many researchers. Tesfatsion [33] presents a general overview of, and discusses current research directions, for ACE. LeBaron [26] presents a rule-based approach using past information (including instrument prices) to forecast future outcomes while Chen and Yeh [4] use Single-population Genetic Algorithms (SGAs) and Multi-population Genetic Algorithms (MGAs) with "schools" to imitate strategies from other agents (a school is a collection of up-to-the-minute updated market information and market participants). Thulasiram, *et al.* [34] present multi-threaded algorithms using the Efficient Architecture for Running THreads (EARTH) [12] for pricing American *call* and *put* options. This chapter will discuss the construction of a dynamic multi-agent architecture for forecasting the pricing of an American option using a non-combining binomial lattice approach described later in the section.

In the world of financial markets, any delay in information processing can potentially lead to financial losses [34]. Computational Finance (CF) problems are typically non-linear, computationally intensive and dynamic so they are difficult to solve "on-demand". Prices and market information change dynamically and collaborations between sellers and buyers are expected. With this in mind, and because of the complexities of the problems, solving any of these problems sequentially would be very slow and inefficient. A better approach for solving these problems is to apply parallel algorithms so the overall workload can be distributed among several processors and therefore better performance can be achieved. Designing effective parallel algorithms for these problems can be very complex and time consuming. Multi-agent systems may offer an environment where some CF algorithms can be easily converted into parallel algorithms and where the load is dynamically and evenly distributed on each processor.

4.2 The Basics of Financial Markets

There are always risks involved with investments. As a rule of thumb in all investments, the higher the risk, the higher the return. In the early years of investments, a small number of investors instantly became millionaires' when their stock prices skyrocketed while others lost all or a large part of their investments when their stock prices tumbled drastically. To help protect and attract more investors, *futures* and *options* were introduced in the trading markets. A *Future* is the right or contract to buy or sell stocks at some future time with obligations (e.g. a future is a contract, which grants the holder/buyer the right to buy or sell stocks within a specific period). An O*ption*, on the other hand, is the right or contract to buy or sell stocks at a future time without any obligations (e.g. an option is a contract, which grants the holder/buyer the right to buy or sell stocks at anytime within a specific period). There are two types of options: *call-options* and *put-options*. A *call-option* is the right to buy and a *put-option* is the right to sell without obligations. Even with futures and options, there are still risks involved with some investments.

In 1973, however, Merton, Scholes and Black developed a mathematical risk free management model (the *Black-Scholes* Model [3]) that guaranteed profitable investments using an option pricing technique. This complex option-pricing model led to explosive growth in stock options and other financial derivatives worldwide and earned Merton and Scholes a Nobel Price in Economics.

One useful and very popular technique for pricing options is to use a binomial lattice model to represent different possible paths that might reflect the security price over the life of the option. The binomial model is particularly useful in setting up option portfolios with reduced risk factors according to the *risk-neutral valuation* principal in pricing options [15]. Using such a scheme, the option pricing is always equal to its expected payoff in a risk-neutral world, discounted at the risk-free interest rates. Using this technique, it is possible to calculate the future price of an underlying asset and its option value over any given period. The input parameters required for creating a binomial lattice are:

- S: the current asset price
- T: the time period (from the current date) to the expiration date
- r: the risk-free interest rate
- σ: the volatility of the asset
- n: the number of intervals to be calculated
- K: the strike price

The height of the lattice is the number of intervals to be calculated (*n*). In general, the higher the value of *n*, the better the accuracy of the result. Figure 4.1 shows a general, one-step binomial lattice model ($n = 1$) with an initial asset price (*S*) of \$40 and a strike price (*K*) of \$50.

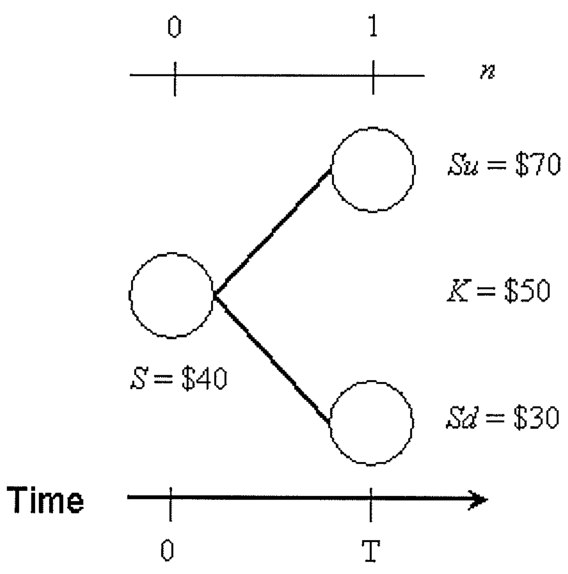

Figure 4.1: A general one-step binomial model (n = 1)

For simplicity, we generalized the problem to calculating the price of an option with the underlying initial asset price *S* over the period of $[0,T]$. During this period, the asset price can either move up (e.g. *Su* in Figure 4.1) or down (e.g. *Sd* in Figure 4.1). The new prices, *Su* and *Sd*, are calculated by the factors at which the asset price moves. Let *u* and *d* be the factors for which the asset price moves up and down at each time interval. At time $t = T$, we have two new possible prices: *Su* and *Sd,* where $u = e^{\sigma\sqrt{\Delta t}}$ and $d = e^{-\sigma\sqrt{\Delta t}}$ and $u > 1.0$ and $d < 1.0$. A more detailed description of the binomial lattice model can be found in [34] and [3]. Figure 4.2 shows a general two-step binomial lattice model (n = 2).

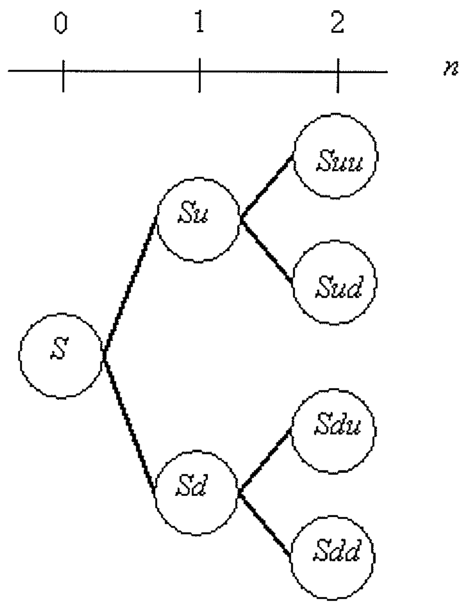

Figure 4.2: A general two-step binomial model (n = 2)

4.3 Basic Algorithms

In general, there are two approaches that can be used to generate the binomial lattice structure: top-down and bottom-up. Each approach solves the same option-pricing problem in a manner consistent with the input parameters mentioned in the previous section. While there are advantages and disadvantages in both approaches, the top-down approach generates the actual tree structure while the bottom-up approach uses formulas. Since the focus of this chapter is on load distribution, we will not address the bottom-up approach because it is more natural to generate the actual tree structure in load distribution.

In the top-down approach, the binomial lattice is generated in a recursive top-down manner starting from t = 0. The generating process stops when the required number of intervals (n) in the lattice have been created. There are two phases during the generating process: the generating phase and the propagating phase. During the generating phase, the tree-like structure is generated and the asset price for each node in the lattice is calculated down the tree-like structure. When the bottom of the lattice is reached, both the asset price and the option price are calculated and propagated up the tree. For all other levels *l*, $0 \leq l \leq n$, the option price of the asset is calculated based on the asset price and option price from the previous level of the lattice. The following pseudo-code shows how the binomial lattice structure is created.

```
GenerateBinomialTopDown(IN inputs, OUT outputs)
Begin
    IF (current level = n)                              (1)
        calculate the asset price for this level        (2)
        calculate the option price for this level       (3)
        return the asset price and the option price     (4)
    ELSE
        current_level = current_level + 1               (5)
        Generate the left sub tree (lattice)            (6)
        GenerateBinomialTopDown(inputs, outputs)        (7)
        Generate the right sub tree (lattice)           (8)
        GenerateBinomialTopDown(inputs, outputs)        (9)
        calculate the asset price for this level        (10)
        calculate the option price for this level       (11)
        return (the asset price and the option price)(12)
    END IF
End // end of GenerateBinomialTopDown
```

4.4 The Design of An Agent-Based Binomial Lattice Option Pricing System

The design presented in this section is focused on dynamic-load distribution among the available processors. The load distribution is calculated based on the number of available[2] agent servers so that the workload is evenly distributed among these agent servers. Each agent in the system is responsible for a portion of the overall lattice. In particular, the workload for each agent is the computational portion of that particular sub-lattice. Figure 4.3 shows this process for a system with two processors (processor 0 is excluded since it does not do any of the actual computation of the lattice) and where $n = 3$.

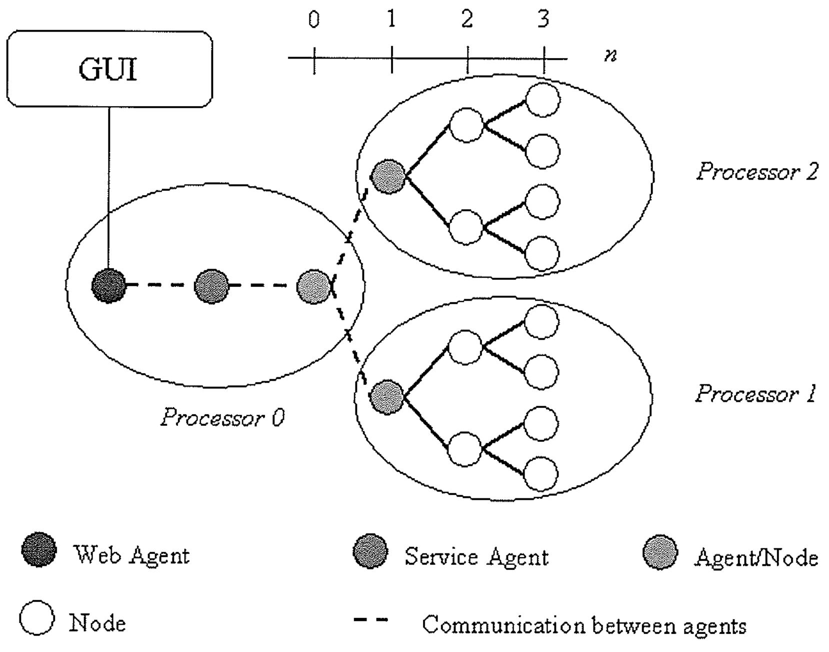

Figure 4.3: Architecture design with two processors and $n = 3$

[2] For simplicity, the number of agent servers available is always assumed to be a power of 2. Handling different numbers of servers is not difficult but would complicate the presentation of the design.

In the design, the web agent, located on processor 0, is responsible for handling the interaction between the users via a GUI (Graphical User Interface) and the service agent. This interaction is based on event triggers. The service agent, also located on processor 0, is responsible for forwarding requests/results from the web agent to the agents (in the same processor) and vice versa. Interaction between all the agents is based on the message passing protocol. All agents in the system, except the web agent and the service agent, are dynamically created in a recursive top-down manner with respect to the top of the lattice. The number of agents created depends on the number of processors used. For example, in Figure 4.3 above, we have an agent in processor 0 and an agent in each one of the two processors used. The agent/node in processor 0 acts as the root of the lattice and is responsible for collecting results from the other two agents in the two processors used. In general, in addition to the web agent and the service agent, the number of agents/nodes created is $2^{P-1} - 1$ where P is the number of processors used (P does not include processor 0). Figure 4.4 shows the same design with four processors ($P = 4$) and $n = 3$.

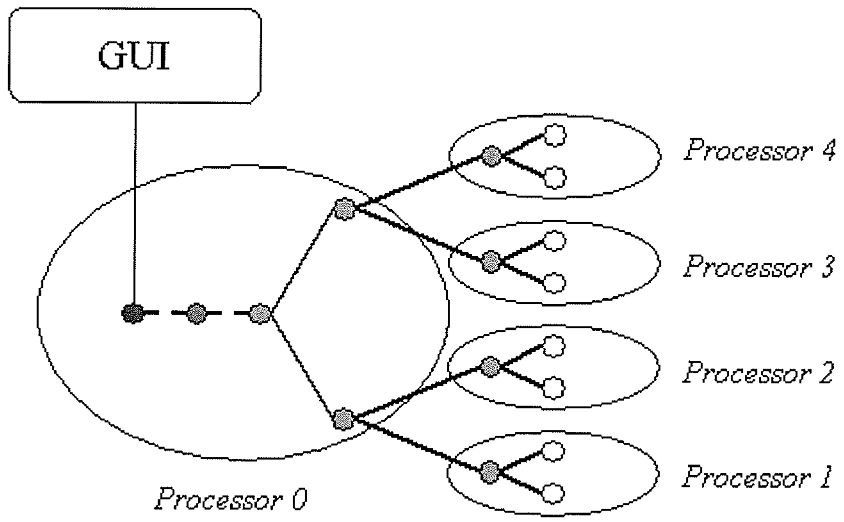

Figure 4.4: Architecture design with four processors and $n = 3$

The purpose of this chapter is to use mobile agents to develop a technique that allows for dynamic workload adjustment. The design ensures that the overall lattice computation is always evenly distributed among the available processors. In general, given *P* and *n* intervals, *n* must be greater than or equal to $log_2 (P)$ for all $P = 2^i$ where $i \geq 1$.

Communication between the agents is a major overhead determining the overall performance. The overhead here is the communication time between the agent(s) in processor 0 and the other *P* agents in the other *P* processors. This communication time is largely dependent on the network traffic (i.e. that number of bytes send by the agents and the current available bandwidth). This particular design works best when n is large and $\log_2$ (P) is relatively large. That is each processor should handle a reasonably large sized sub-lattice. When n is closer to the value of $\log_2$ (P), the time required for communication may be larger than the actual computational time of the sub-lattice at a particular processor P_i. In this case, because the network is much slower compares to the processing time of the sub-lattice at a particular processor.

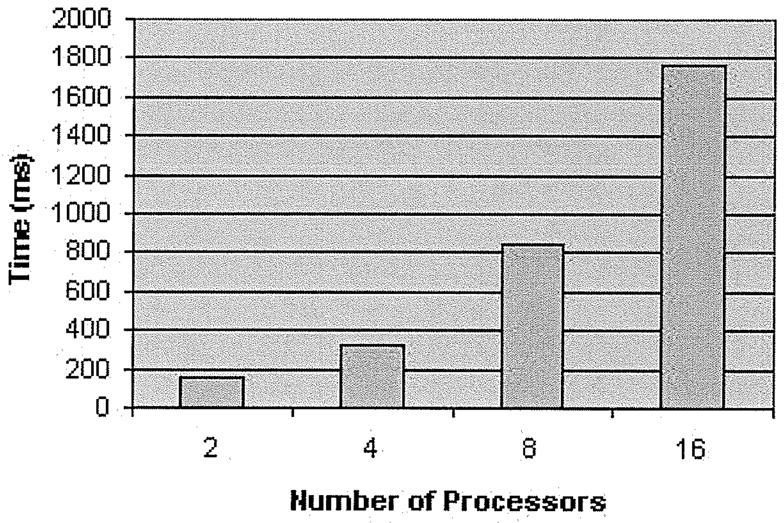

Figure 4.5: Communication timing vs. number of processors

Figure 4.5 shows the communication time between processors. As the number of processors increases, the communication time between the processes grows exponentially.

This chapter demonstrates the effectiveness of mobile agents in a distributed environment. Beside from the use of multi-agents to construct the well-known binomial tree-like structure, another focus of this chapter is the use agents for load distribution among processors. That is, each agent in the system is responsible for a sub-lattice on a particular processor so that the total computational of the lattice is evenly distributed among the processors. The use of mobile agents and their effectiveness presented in this chapter motivated the use mobile agents in this thesis.

Chapter 5:

The WWW-DIMA Prototype: Architecture and Implementation

5.1 Introduction

This chapter introduces a scalable architecture capable of delivering information from multiple data sources on the WWW. This new architecture uses the dynamic deployment of mobile agents in a distributed environment. The design goals of the architecture are as follows. First, the architecture must allow the dynamic addition and removal of component data sources while providing users with a single-step access to multiple data sources on the WWW. Second, the architecture must also allow support Business To Business (B2B) data exchange. These two requirements are particularly important as the former allows scalability of the architecture and the later allows integration of data sources on the WWW including legacy data sources. Section 5.2 describes the architecture together with the supporting software and tools used in the implementation. Section 5.3 summarizes the function of the components while Section 5.4 describes the interactions between the components of the architecture. Section 5.5 presents the implementation of the architecture in some detail. Section 5.6 describes the operations supported by the architecture, and Section 5.7 discusses certain optimization issues related to the implementation details of the architecture.

5.2 The WWW-DIMA Prototype Architecture

Recall that the purpose of this research is to explore the use of a mobile-agents system to integrate homogeneous/heterogeneous databases on the WWW. Figure 5.1 below shows the overall architecture of the WWW-DIMA (WWW-Database Integration via Mobile Agents) prototype.

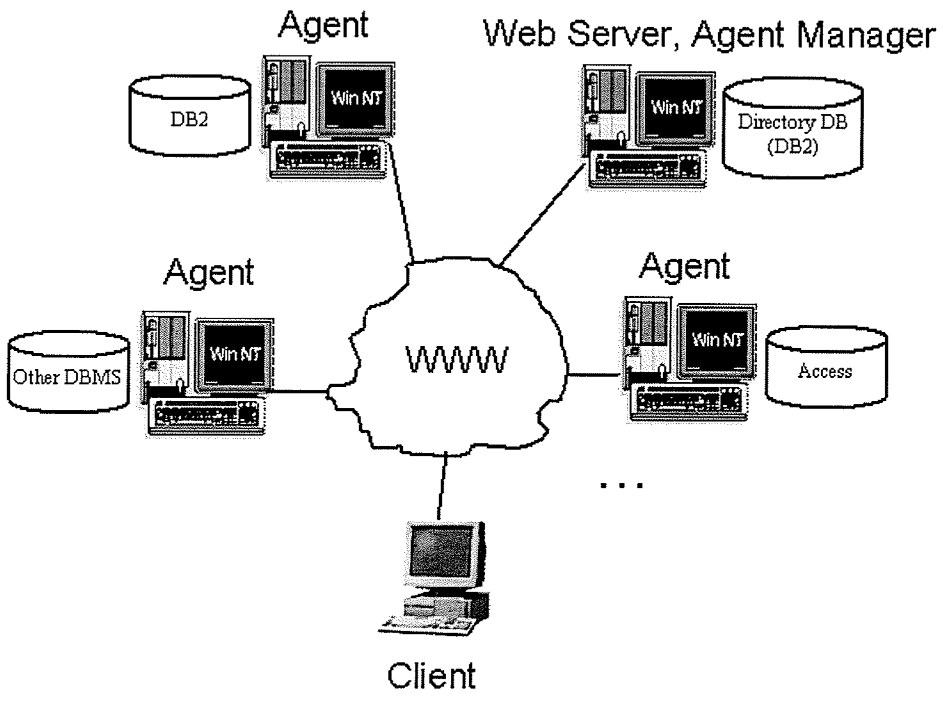

Figure 5.1: The proposed architecture of the WWW-DIMA prototype

Figure 5.1 shows the overall architecture of the WWW-DIMA prototype. The WWW-DIMA prototype uses mobile agents to integrate component database schemas into a global directory (the Directory Database) and to deliver data to the users from one or more relational data sources. The Directory Database is created with the assumption that all the component database schemas are semantically correct.

The implementation of WWW-DIMA prototype is on the Windows Families (Windows 2000 and Windows NT) using Aglets [1] and Java [20]. Other tools used in the development are:

- Microsoft Internet Information Services (IIS) [17]
- IBM Universal Database DB2 [8]
- Microsoft Access Database
- Java Database Connectivity/Open Database Connectivity (JDBC/ODCB) [18]

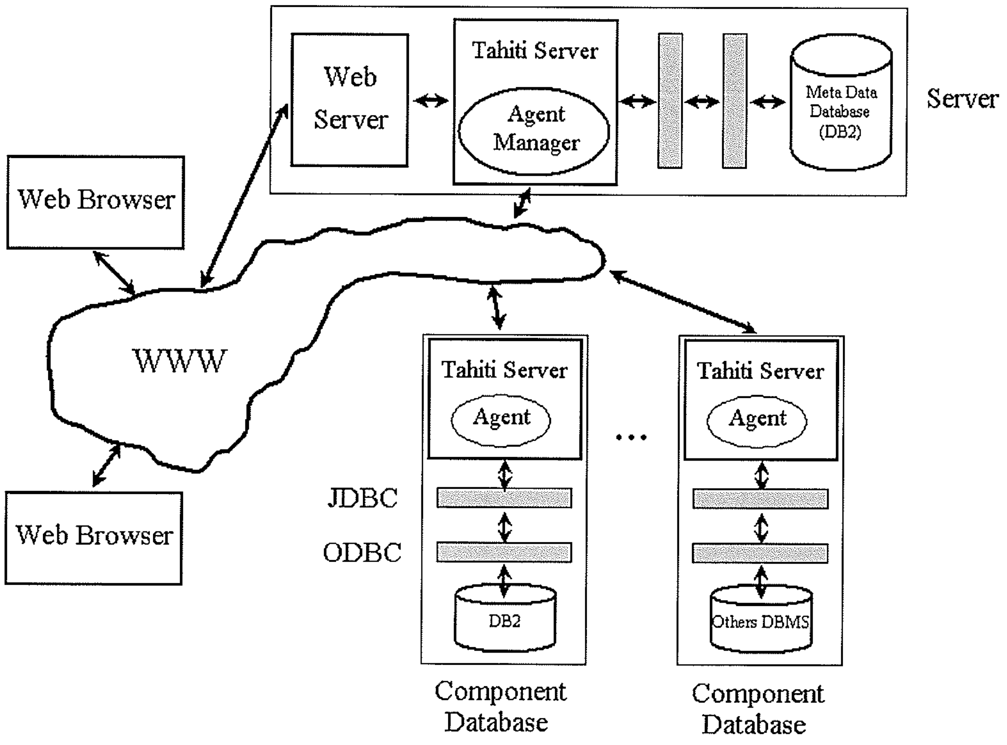

Figure 5.2: Details of the proposed WWW-DIMA prototype architecture

5.3 Summary of Components of the WWW-DIMA

5.3.1 Directory Database

The directory database contains a catalog of all the other component databases on the network. Only the Agent Manager and the Service Agent Manager (both defined later) use the information in this database. Information included in this database is:

- Host names of all the component databases.
- Host status of all the component databases.
- Database instance names of all the component databases.
- Data source names (DSNs) for all the component databases.
- User ids and passwords to connect to the data sources for all components hosts.

5.3.2 Tahiti Servers

Tahiti servers provide the contexts or aglet servers that provide the resources and environments for aglets that are to be created and executed. An instance of the Tahiti server is required to run on each of the component database hosts.

5.3.3 Service Agent Manager[†]

The Service Agent Manager coordinates and manages all the Service Agents in the system. Task assignments to each individual service agent are based on the information from the Directory Database.

[†] Neither the Service Agent Manager nor the Service Agents are shown in Figure 5.2 because they are only periodically created and activated.

5.3.4 Service Agents†

Periodically, these aglets, as directed by the Server Agent Manager, roam the network and collect the information about the component databases and their corresponding host statuses. Any changes in the local component schemas will be detected and updated in the Directory Database by these Service Agents.

5.3.5 Agent Manager

The Agent Manager is a component of the WWW-DIMA prototype, which responsible for handling requests from the Web Server and delivering of data from the Agents to the Web Server. When a request query is received, the Agent Manager decodes the request and re-directs the request to those Agents that are responsible for handling the request (i.e. those Agents running on sites with data needed to evaluate the query). Decoding is the process of extracting the location(s) of data and the catalog information from the Directory Database. After the decoding process is done, the Agent Manager sends messages and waits for replies from the necessary agent(s). When all the replies are received and combined, the result of the request is forwarded to the web server (IIS), which then presents it to the user. By using the information from the Directory Database, the Agent Manager can effectively direct agents to appropriate data sources. In the case where a data source is unavailable, the Agent Manager can direct agents to alternative data source(s). The current implementation of the WWW-DIMA prototype, however does not support this functionality.

5.3.6 Agents

Agents are created and sent to remote locations by the Agent Manager. The number of agents created depends on the number of online-available databases in the Directory Database. That is, an Agent for each online-available database. Upon arrival at the destination hosts, these agents wait for messages from the Agent Manager. When a message is received, the agent will carry out the corresponding task and respond back to the Agent Manager with the query result.

5.4 Component Interactions of the WWW-DIMA Prototype

Figure 5.3 illustrates the interactions between the components in the WWW-DIMA prototype. These interactions are:

1. Client submits a request to the Web Server.
2. The Agent Manager obtains the request from the Web Server.
3. The Agent Manager decodes the user request and formats the user request into an internal format using information from the Directory Database.
4. The Agent Manager sends the internal request to the particular Agents (at remote hosts) that are responsible for handling the request.
5. Agents, at the remote location, receive the request, extract the requested data from their local databases and format the data result as XML.
6. The requested data, in XML format, is sent back to the Agent Manager.
7. The Agent Manager parses the XML data and forwards the raw data result to the Web Server.
8. The raw data result is sent to the original request through a web page.

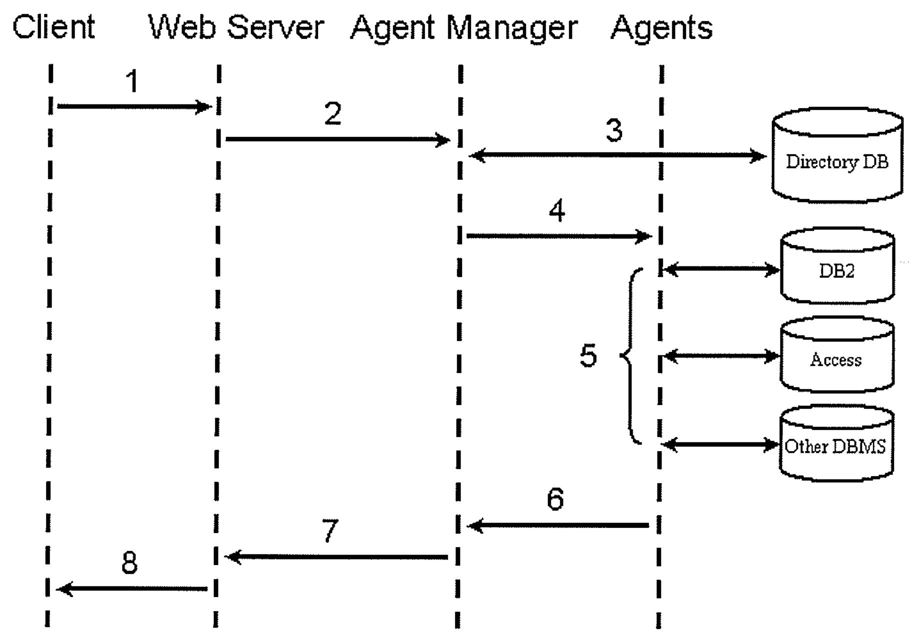

Figure 5.3: Component interactions of the WWW-DIMA prototype

5.5 Implementation Details of the WWW-DIMA Prototype

Figure 5.4 highlights some implementation details of the proposed architecture. Since the WWW-DIMA prototype uses mobile agents to locate and deliver information from multiple component databases, an agent context or an agent server must run on each of the component database hosts, including the server. (The server here refers to the host that has the web server running on it). As described in Section 5.3.2, an agent server (in this case, the agent server for Aglets, called Tahiti) is an application program that provides the services and environment permitting agents to execute. Using a Tahiti context, the system is launched on the server and a WWW-DIMAAgent is created. This agent is the main thread of execution of the WWW-DIMA system. Once created, the WWW-DIMAAgent creates an agent called ServiceAgentManager. At this point, the WWW-DIMAAgent halts its execution and waits for a "complete" signal from the

ServiceAgentManager. The main purpose of the ServiceAgentManager is to periodically create and coordinate the ServiceAgents which check and update the status of the component database hosts in the Directory database. Once created, the ServiceAgentManager establishes a local database connection to the Directory database and extracts the 'available' component hosts and database information (the 'available' component hosts here refer to the component hosts that were last known to be 'available' for connecting to via the network). The information extracted includes the component host names, the component host IP (Internet Protocol) addresses, the agent server port numbers, the ODBC names, and user ids and passwords for connecting to each component database. For each record extracted, the ServiceAgentManager creates a ServiceAgent and it with the record information. The ServiceAgentManager also tracks the number of ServiceAgents created so that it knows when the ServiceAgents have completed their work (checking each component database). Once a ServiceAgent is created, the ServiceAgent tries to dispatch itself to the component host. At this point, there are two possible scenarios:

- Scenario 1: the component host is unreachable. That is, the component host became unavailable sometime after the last 'checkpoint'. This means the ServiceAgent will not be able to dispatch to the component host. The ServiceAgent then marks this component host as 'unavailable' in the Directory database. The ServiceAgent then terminates its own execution.
- Scenario 2: the component host is reachable. The ServiceAgent dispatches itself to the component host, connects to the component database locally and extracts information about the component database. The information extracted includes the number of relations, number of attributes for each relation and the data type for each attribute. This information is then saved as state of the agent and brought back to the server. The information brought back is checked against the information in the Directory database and updates are made accordingly. In the case of component schema changes, additional schemas, attributes and data types are added or marked as

'removed' in the Directory database. The ServiceAgent then terminates its own execution.

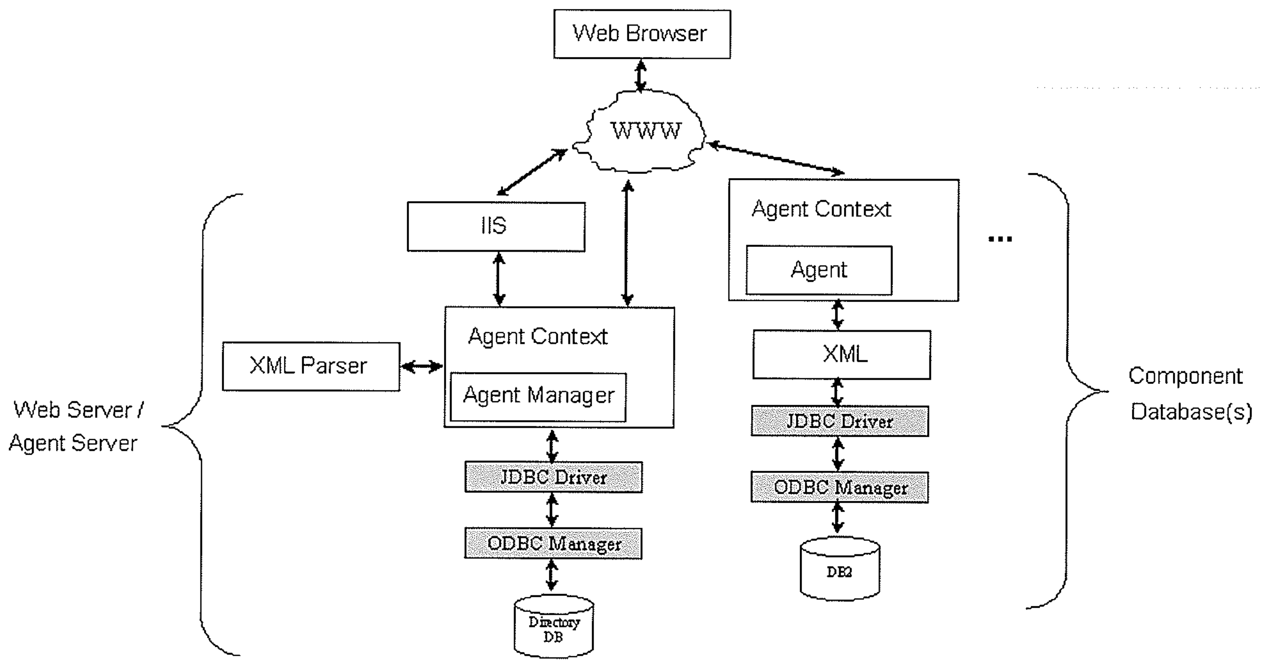

Figure 5.4: Details implementation of the WWW-DIMA prototype

When all ServiceAgents are terminated, the ServiceAgentManager sends a message to the WWW-DIMAAgent indicating the checking process is now completed. After the message is sent, the ServiceAgentManager then deactivates itself for a period of 5 minutes (this 5 minutes time frame was chosen because 5 minutes is a reasonable amount of time for checking the status of the component hosts. As shorter time frame would create unnecessary overhead). While deactivated, all the state and code of the ServiceAgentManager are written to the local disk. After the deactivated time period, the ServiceAgentManager is activated automatically by the agent context and the checking process is repeated again.

When the WWW-DIMAAgent receives the message indicating the checking status is done from the ServiceAgentManager, it creates another agent called the

AgentManager. Once created, the AgentManager establishes a local database connection to the Directory database and extracts the available component host and database information. For each record extracted, the AgentManager creates an Agent and initializes it with the record information. The number of Agents created depends on the number of available component hosts (i.e. an Agent for each component host) in the Directory database and is tracked by the AgentManager. Once created, Agents dispatch themselves to their assigned component hosts and wait for requests from the AgentManager. The operations supported by the WWW-DIMA architecture are described in Section 5.6.

As stated in Section 1.3, one of the research goals is to be able to add or remove component databases dynamically. Adding a new component database is done manually by adding the following information to the Directory database: the component host name, the component host IP address, the port number, the ODBC name, the user id and password for accessing the component database. Removing component databases can be done in two ways:

- Removing the entry completely from the Directory database. This scenario is appropriate when the data source will no longer be used.
- Marking the entry as 'removed' in the Directory database. This scenario is appropriate when the data source is temporary unavailable.

Terminating the WWW-DIMA system can be done in two ways: by terminate the agent context or by terminating the WWW-DIMAAgent. In either case, the method `dispose()` of the WWW-DIMAAgent is called on terminating. Before the WWW-DIMAAgent is terminated, a terminating message is composed and sent to the AgentManager and the ServiceAgentManager. The WWW-DIMAAgent is then terminated. Upon receiving the terminating message from the WWW-DIMAAgent, again, before terminating itself, the AgentManager forward the terminating message to all Agents in remote component hosts. The AgentManager then terminates itself. In the component hosts, upon receiving the terminating message, Agents de-allocate the resources held and then terminate themselves. When agents are terminated, their threads are permanently removed from the agent context. Similarly, the terminating process is the same for the ServiceAgentManager and all ServiceAgents.

A major drawback of the WWW-DIMA prototype is that failure of either the server or the AgentManager means no information is available. The former is obvious. The later is because the AgentManager is the mediator between web server and the Agents in the component hosts.

5.6 Operations Supported by the WWW-DIMA Prototype

Using a web browser (e.g. Netscape or Internet Explorer), the users can connect to the AgentManager and query the contents of any component databases in the system. Users submit their requests and receive results through the web server (IIS in the prototype) in the form of HTTP POST response and HTTP response messages. For all requests submitted by the users, the AgentManager (see Figure 5.4) decodes the requests and forwards the requests to the appropriate (at remote locations) that are responsible for satisfying the requests. At this point, the AgentManager blocks and waits for a reply back from the Agents in the remote locations.

At the remote location, once a request from the AgentManager is received, the Agent establishes a local database connection to the component database via ODBC. The Agent then extracts only the required data and formats the query result into XML format. Since Agents use JDBC/ODBC to connect to the database locally, the agents can connect to virtually any DBMS back-end that supports the JDBC/ODBC technologies on the local host. Once the data formatting is done, the XML-query result is encoded as a message and sent back to the AgentManager as a message though the Agent's proxy.

Back on the web server, when the XML-query result is received, the Agent Manager uses JDOM [21] to parse the XML-query result and format the contents of the query result into an HTML page. This includes generating the heading and appropriate HTML tags for the HTML result page. Once this assembly process is done, the Agent Manager forwards the HTML page as a response to the Web Server. The web server then presents this page to the users as the result of the original query.

5.7 Optimization of the WWW-DIMA Prototype

In the first implementation of the WWW-DIMA prototype, a number of Agents are created when the prototype is launched. The number of Agents created is dependent on the number of available component databases in the Directory Database (i.e. an Agent per component host). Each Agent is then responsible for handling data requests from a component database in the system. For every query submitted by the users, the particular Agent responsible for the query is dispatched to a remote host where the requested data is located. Once the data extraction is complete, the agent then dispatches itself back to the original location with the requested data. The requested data is then presented to the user by the web server via the AgentManager.

The overhead in the first implementation arises because every query submitted by the users requires an Agent to be dispatched to the data source, and then back to the server with the query result. As mentioned previously, every time an Agent transports itself from one host to another, all state, and code are transported with the Agent. This is costly because for every query, in addition to the data result, all agent state, and code must be transported. The second efficiency issue is the network bandwidth. That is, as the number of requests submitted by the users increases, the traffic between the host where the AgentManager resides and the hosts of the component databases increases. This is because large amount of data (query results, states and code(s) of Agents) being transferred between these hosts.

In the second implementation, Agents are created and dispatched to the hosts of the component databases when the system is launched. Upon arrival at the destination, Agents await messages from the AgentManager. When a user request is received, the AgentManager formats the user request into an internal format and then composes the internal format into a message. The message, and only the message, is then sent over the network to the Agent who is responsible for the request. When a query is received from the AgentManager, the Agent will carry out the corresponding task and reply back with only the query result. In this implementation, only the requested queries and the queries' results are sent over the network, instead of dispatching Agents for every single request. As estimated, the speed up of the second implementation over the first implementation is about 25%.

Chapter 6: Performance Evaluation of the WWW-DIMA Prototype

Two databases are used in evaluating the performance of the WWW-DIMA prototype. To simulate a heterogeneous environment, two different DBMSs are used: Microsoft Access [42] and IBM DB2 [8]. The Microsoft Access database is called U_RECORDS. U_RECORDS is a typical university student records database. This database contains information about students, staff, courses, *etc.* The second is an IBM DB2 database called MANUFAC. MANUFAC is a typical production manufacturer database. Information included in this database are products, suppliers, product category, *etc.* To obtain consistent timing results, each query in the WWW-DIMA prototype was submitted 100 times and the average time was obtained. The same queries were also evaluated using the Client/Server model using Applets. Applets were chosen because, like the WWW-DIMA prototype, Applets use the web server to connect to the database thus they represent a fairer type of comparison.

Evaluations of the WWW-DIMA prototype were performed on a network of workstations running Windows 2000 (Pentium 4, 1.4GHz processor speed with 512 MB of RAM) on 100Mbps switched Ethernet. The following are used in evaluating both the WWW-DIMA prototype and the client/server implementations.

- *Total query time (T)*: the time between when the request is submitted on the client and when the query result is received. This does not include the time taken to display the query result.
- *Agent processing time (T_a)* on component hosts. We do not include constant time factors such as time to decode the user requests, time to compose the query results and present the results to the user. T_a includes only the following:
    - $T_{db\text{-}xml}$: time taken to extract the data from the database and format the query result(s) in XML
    - $T_{en\text{-}msg}$: time taken to encode the XML-query results into messages (at the sender end)
    - $T_{de\text{-}msg}$: time taken to decode the messages back to the XML-query results.
- *Communication time (T_c)*: the time between when the request is sent from the Agent Manager and the time when the reply is received. T_c is largely dependent on the load of the network at the time when the message is sent and, of course, on the size of the data being sent.

Furthermore, to minimize the runtime differences, the same runtime environment (Java) and the same database definitions were used in both approaches. In general,

$$T = T_c + T_{db\text{-}xml} + T_{en\text{-}msg} + T_{de\text{-}msg} \qquad (1)$$

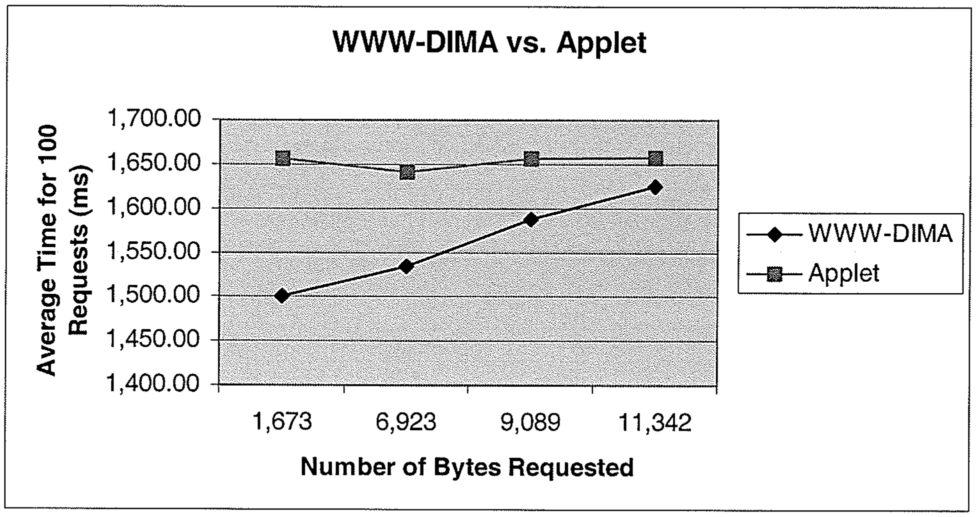

Figure 6.1: Comparison of the WWW-DIMA prototype and Applet at off-peak hours

The timing observations of the prototype were done at two different times on a closed subnet: at peak hours and during off-peak hours (early in the morning when the network traffic is low). The timing was repeatedly observed over a period of four days and very similar patterns were obtained. To ensure variation, the size of the data results requested were purposely set to vary from 1.6 KB to 11 KB. Figure 6.1 shows a comparison of the WWW-DIMA implementation and the traditional client/server implementation (Applet) at off-peak hours. The results from Figure 6.1 show that the time required for the Applet implementation changes only slightly as their size of the query results increases. This is because all Applets are downloaded before they are executed in the web browser environments and while Applets are running, permanent connections to the server are established; note that, as the number of Applets increases so does the traffic on the server's end. The time required for the WWW-DIMA prototype, on the other hand, increases as the query result size increases but remains lower than the time required for the Applet-based implementation.

Figure 6.2 shows the comparison of the WWW-DIMA implementation and the traditional client/server (Applet) implementation during peak hours. The time required for the Applet implementation (shown in Figure 6.2) is similar to that in Figure 6.1, except with some additional overhead due to the higher network traffic. The time required for the WWW-DIMA implementation shows a similar trend to that of Figure 6.1. During testing it was noted that the timing of the WWW-DIMA prototype was occasionally higher than that of the Applet implementation. These anomalies were the results of network congested at the time when the experiment was done. Experiments with the WWW-DIMA prototype have shown that even with the number of bytes requested as large as 45K, the response time of the WWW-DIMA prototype is still better than that response time of the Applet implementation.

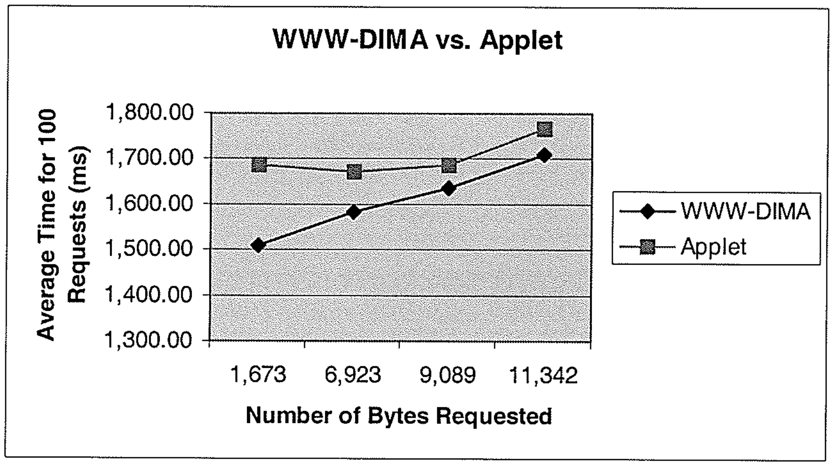

Figure 6.2: Comparison of the WWW-DIMA prototype and Applet at peak hours

The response times obtained by the WWW-DIMA prototype in both Figure 6.1 and 6.2 are also consistent with those obtained by Ismail, et al [16] and Papastavrou, et al [30]. Though, the response times of the WWW-DIMA prototype are better than those of the Applet implementation, even better response times can be obtained by reducing the complexity of the internal implementation of the Aglets frame-work. Recent studies of Aglets [16, 30] have suggested that the main overhead in Aglets is the ATP (Agent Transport Protocol). Since the WWW-DIMA prototype uses the aglet's ATP, its performance is affected because of the additional overhead. A future implementation might want to look at replacing the ATP with a simpler ATP implementation.

Figure 6.3 below compares the response time of the WWW-DIMA prototype at peak and off-peak times. As the number of bytes requested increases, the response time of the WWW-DIMA prototype at peak times grows faster than at off-peak time. This is because at peak times the network traffic is higher than that at off-peak times. By observation and under the assumption that the network traffic is constant, the response time of the WWW-DIMA prototype is directly proportional to the number of bytes requested.

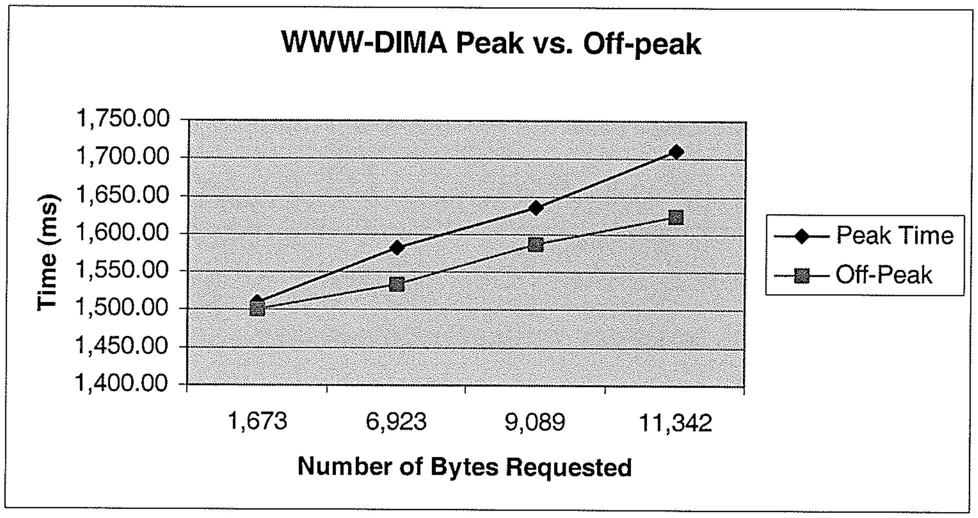

Figure 6.3: Comparison of the WWW-DIMA prototype at peak and off-peak times

Unlike other recently published work, the agents in the WWW-DIMA prototype use messages to communicate with each other. As mentioned previously, communication time (T_c) for the WWW-DIMA prototype is largely dependent on two factors: the network traffic and the amount of data being transferred. Based on the evaluation of the WWW-DIMA prototype, the average data exchange time between the agents is about 816 ms. That is, the WWW-DIMA prototype performs best when the size of the requested data is small.

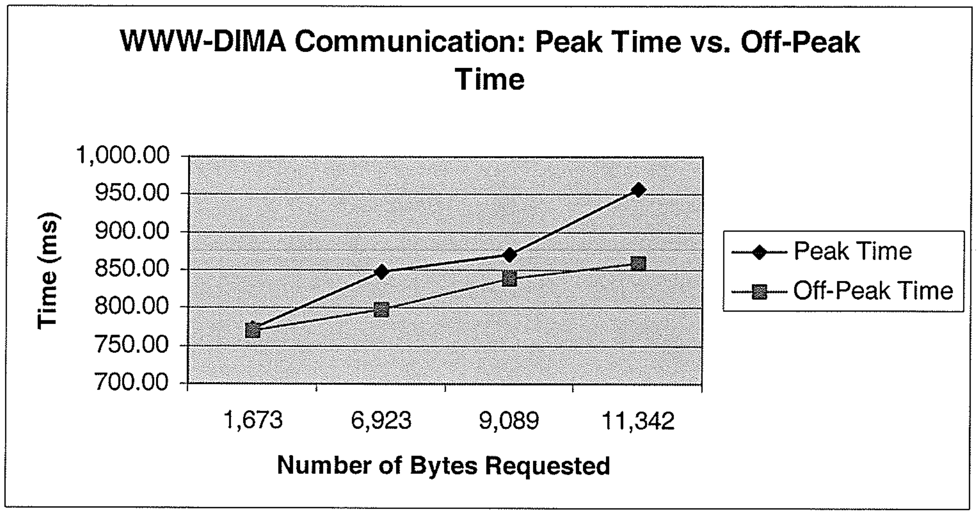

Figure 6.4: The WWW-DIMA prototype communication time at peak and off-peak times

Figure 6.4 compares the communication time of the WWW-DIMA prototype at peak time and off peak time. Communication time of the WWW-DIMA prototype is largely dependent on the network traffic at the time of the evaluation and the number of bytes been transferred.

Chapter 7: Related Work

Many working prototypes of systems that support distributed information retrieval using mobile agents have been implemented and evaluated recently. Although, all the prototypes make use of the same software engineering paradigm, mobile agents, their underlying implementations and approaches are very different. Though, different agent frame-works were used, the following work is related to that presented in this thesis.

7.1 Performance of Mobile Agents

Kotz, *et al.* [23] has analyzed and studied the performance of mobile agents for filtering data streams in a wireless network environment. They assumed that the agent server received steady raw input data streams broadcast from the Internet. They also assumed that each wireless client is interested in only a subset of the raw streams. In this particular application, the following two observations were made. First, to minimize the workload on the wireless network, transmitting only the relevant data should be transmitted. Second, to minimize the power consumption on the wireless clients, filtering of the raw input data streams should be done on the server before sending the data to the wireless clients. With a set of predefined parameters and assumptions, the authors developed an analytical model and a set of equations for filtering the input data streams on both the server and the clients. These equations are based on the number of agents used and the available bandwidth between the agent server and the agents on the wireless clients. They had demonstrated the benefit of using mobile agents in a wireless network. They presented only an analytical model of their experiments

Kotz, *et al.* [24] compare the performances of four different agent-based frameworks (D'Agents [6, 7], EMAA [10, 11, 13], KAoS [22], NOMADS [29]) with traditional client/server techniques for distributed information retrieval [24]. In the agent-based approach, agents are dispatched from the client hosts to a server where a collection of documents is stored. On the server, the agents search for the requested document(s) and bring back only the requested document(s). In the client/server approach, clients connect to the server, download all available documents, and filter for the requested document(s) on the client-side. Though, the four frameworks share a common implementation language, Java, their performance are much slower. These differences are due to the internal implementation of each framework. (For example, both EMAA and KAoS cache the open socket on the agent server and re-use the socket for migrating the agents between the hosts. Caching and re-using these sockets save overhead in performance. D'Agents and NOMADS, on the other hand, reopen a new socket for each migration)

These performance differences are insignificant when compared to the cost of migrating the agents between hosts. Their experiment was done with different numbers of clients (1 to 20) on a 1, 10 and 100 mbps network. Kotz, *et al.* conclude that both the agent-based model and the client/server model are dependent on the bandwidth of the network. They also conclude that the client/server application's performance is best on high bandwidth network. Agent-based application performance, on the other hand, depends on two factors: the network bandwidth and the overhead of the agent's framework. The model presented by Kotz, *et al.* dispatches an agent for every user request. This is inefficient compared to the WWW-DIMA prototype, which uses a message for every user request. That is, with messages, only relevant data are transferred (instead of relevant data, agent's state and code).

Ismail, *et al.* [16] compare the performance of RMI (Remote Method Invocation), Aglets and a minimal self-built mobile agent prototype. All three models were built using the same language, Java. Their comparisons were done based on the following assumptions: in the client/server model, the client uses two RPCs (Remote Procedure Calls) to connect to two different servers to obtain the requested information. In the mobile agent model, an agent is created and sent sequentially to two servers to collect information. The collected information from the second server is based on the information collected from the first server. Once the information is collected from the two servers, the agent dispatches itself back to the client. Their results confirm that both mobile agent models (Aglets and the minimal self-built mobile agents prototype) provide better performance than the RMI client/server model. Also, the Aglets timing they report is consistent with that of the WWW-DIMA prototype.

7.2 Design Comparisons

Papastavrou, *et al.* [30] has developed a very similar model to that of the WWW-DIMA prototype. Unlike the WWW-DIMA prototype, however, Papastavrou *et al.* is a web-based Aglet framework that uses Applets as the client interface and to control the agents behind the scene. An advantage of using Applets as an interface is that other standard Java GUI components can easily be integrated. However, the more components that are added to the Applet, the slower the download of the Applet will be. Other major differences in WWW-DIMA prototype are that: the WWW-DIMA prototype uses HTML as the user interface (usable even with minimal bandwidth) and additional component databases can be added or removed from the system dynamically. Overall, the network performance of the WWW-DIMA prototype is also better because it avoids the downloading overhead of the Applet.

Vlach [30], is another prototype multidatabase management system based on mobile agents that is similar to the WWW-DIMA prototype. Vlach supports all database operations (search, insert, update and delete) as well as database stored procedures. As with the WWW-DIMA prototype, Vlach confirmed that migrating agent(s) for each query is slow and that using message passing will result in better performance. Vlach also employs migration of agents to support both global distributed query execution and procedures.

Magnanelli, *et al.* [27] use a database to store “knowledge data” and “statistical information” to assist agents with their tasks on the WWW. That is, Magnanelli, *et al.*’s agents use information stored in user’s database as the starting point of the search. Knowledge data includes the searching starting point(s), user preferences, agent preferences, *etc.* Statistical information, on the other hand, includes the location of data, past frequent data access patterns, the results, *etc.* Magnanelli, *et al.’s* agents use both keyword search and pattern-based search. In both cases, agents follow the ‘interesting’ links to find the information requested. Search results are saved, filtered and presented to the users at a later time. Over time, preferences and past reference information are updated in a database for future uses. Future work by Magnanelli, *et al.* includes web-access and XML documents, features that are included in the WWW-DIMA prototype.

Chapter 8:

Conclusions and Future Research Directions

8.1 Conclusions

Past research has shown that using the traditional client/server model for accessing multiple data sources on the web is sometimes slow, and can be complicated to implement. In addition, the client/server model changes the performance of the applications as the number of users increases. The new architecture described in this thesis is both flexible and scalable. The new architecture is flexible because deployments of Agents are done dynamically. Also, the new architecture uses ServiceAgents to check and update the status of component data sources before Agents are sent to the data sources. The new architecture is scalable because additional data sources can be added to the architecture by adding the data source information and the component host information in the Directory Database. With the use of mobile agents, the new architecture allows users to effectively access multiple data sources on the web and yet manages the complexity of the system.

The new architecture described here is also offers a performance advantage compared to other recent systems. Unlike other work, the WWW-DIMA prototype uses message passing instead of agent dispatching. Dispatching agents can be costly as the transfer time is directly related to the total number of bytes transferred across the network. Although, some publications have studied the concepts of integrating mobile agents and WWW-Databases, they require additional components or services to be installed on the clients. The implementation of this work is mainly relies on the Directory Database, AgentManager, Agents, ServiceAgentManger, ServiceAgent and other existing supporting tools and software used in the prototype. No changes are required to the existing tools and software. One disadvantage of the work presented here, and of all mobile agent systems in general, is that agent servers must be installed on all the component hosts.

Finally, mobile agents are appropriate for solving computationally intensive applications (discussed in Chapter 4) and are appropriate for wireless environment [16] (an environment where the server is a cluster of powerful processors capable of handling intensive computational processing and the network bandwidth between the server and the users is limited). In this environment, agents can be dispatched to the server, process the requests on the server and bring back to the clients only the relevant data results. This approach reduces the network loads but increases the server loads. One attractive feature of the WWW-DIMA is that it can integrate any DMBS that supports the JDBC technology. Other contributions of the WWW-DIMA prototype include:

- The WWW-DIMA prototype can integrate any DBMS that supports the JDBC technology.
- Addition and removal of component databases is done automatically.

- Data exchanged on the WWW occurs while supporting existing applications. That is, the data transported by the agents are presented in the eXtensible Markup Language (XML) format, a markup language that has been widely used as metadata standard. With rich document structures like XML, not only can data transported by the agents now can be used over the web, but the data can also be used for Business To Business (B2B) data exchange between existing and/or future applications.

8.2 Future Work

Since software agents are a relatively new area, much more work is still required. The drawbacks described in Section 2.6.2 must be address adequately before this software engineering paradigm can be used widely. Currently, the WWW-DIMA prototype does not support multiple concurrent requests. As mentioned previously, for every single request submitted by the user(s), the Agent Manager has to decode the request, forward the request to the Agent who is responsible for the request, and wait for the response to be returned from the Agent. As a result, when multiple concurrent requests are submitted, the Agent Manager's HTTP requests queue grows until the queue is full and subsequent requests are ignored. Aside from handling concurrent multiple requests, other future research areas include:

- Data caching

  Caching of frequently accessed data records to improve access time and performance of the system. However, some mechanism is required to guarantee data consistency. Data caching can be handled by the AgentManager.
- Distribution of Agent code

  Since the Agent Manager knows the locations of all agents at the component hosts, pre-distribution of the agent code to all the component hosts would save time when agents are created in the remote host. This does not apply to WWW-DIMA prototype since agents are created and dispatched to component hosts once at the beginning when the system is launched. No additional agents are created on the component hosts. Distribution of the agents' code can be used to improve performance in other applications.

- Support of other database operations

  The WWW-DIMA prototype can be extended further to support other concurrent database operations: update, insert and delete. Clearly some modifications of the WWW-DIMA prototype interface is required.
- Secure communication channel

  In any agent system, a secure communication medium between agents is required. This is critically important for applications with sensitive data such as online banking or e-commerce applications. Authentication of agents should also be included so interactions are properly validated.
- Support of disconnected operations

  Mobile agents are appealing to wireless clients because mobile agents can be disconnected from the client, move to the location of the data to be accessed, carry out the requested task and bring back only the relevant data once the client is reconnected.
- Save time by pre-processing

  Agents can also be parts of a real-time system. In particular, agents can be used for pre-processing by requesting and downloading information ahead of time. This requires the agents to 'memorize' user preferences. That is, 'what' information needs to be downloaded and 'when'. The downloaded information is then presented to the user when appropriate.
- Integration with other existing client applications

  With the use of XML, other existing applications and/or future applications can be easily integrated with the WWW-DIMA prototype. Figure 7.1 shows an example of the integration of the WWW-DIMA prototype with other desktop applications.

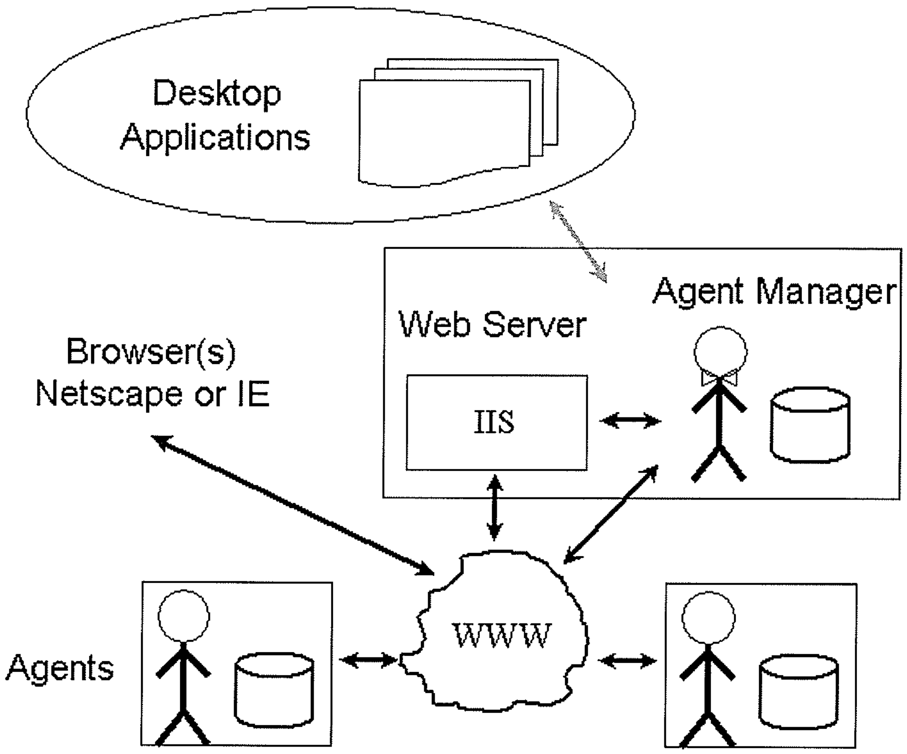

Figure 7.1: Integration of the WWW-DIMA prototype with other applications

- Support for alternate data sources

  As discussed at the end of Section 5.3.5 (and assuming that there are replicas of the data sources), in the case where a data source is unavailable, the AgentManager can assign agents with alternate data sources. So, if an agent encounters an unavailable data source, the agent can dispatch to the alternate data source and extract the data from there.

References

[1] IBM Aglets Software Development Kit (ASDK), available at http://www.trl.ibm.com/aglets/

[2] Agent Transfer Protocol, available at http://www.trl.ibm.com/aglets/atp/atp.htm

[3] Fisher Black and Myron Scholes. *The Pricing of Options and Corporate Liabilities*, Journal of Political Economy, Vol.81, Jan.-June 1973, pp.637-654

[4] Shu-Heng Chen and Chia-Hsuan Yeh. *Evolving Traders and the Business School with Genetic Programming: A New Architecture of the Agent-Based Artificial Stock Market*, Journal of Economic Dynamic & Controls, Vol. 25, Feb 2001, pp. 363-393

[5] Edgar F. Codd, *A Relational Model of Data for Large Shared Databanks*, Communications of the ACM, June 1970, pp. 337-387

[6] R. S. Gray. *Agent Tcl: A Flexible and Secure Mobile-Agent System.* PhD thesis, Department of Computer Science, Dartmouth College, June 1997. Available as Dartmouth Computer Science Technical Report RT98-327

[7] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus. *D'Agents: Security in a Multiple-Language, Mobile-Agent System.* In G. Vigna, editor, Mobile Agents and Security, Vol. 1419 of LNCS, p. 154-187. Springer-Verlag, 1998

[8] DB2 Universal Database, available at http://www-4.ibm.com/cgi-bin/db2www/

[9] Bipin Desai, *An Introduction to Database Systems*, Reading, Galgotia Publications (West Publishing) 1991

[10] S. McGrath, D. Chacón, and K. Whitebread. *Intelligent Mobile Agents in the Military Domain.* In *Proc. Of the Autonomous Agents 2000 Workshop on Agents in Industry*, Barcelona, Spain, 2000

[11] D. Chacón, J. McCormick, S. McGrath, and C. Stoneking. *Rapid Application Development Using Agent Itinerary Patterns.* Technical Report #01-01. Lockheed Martin Advanced Technology Laboratories, March 2000

[12] K. B. Theobald, J. N. Amaral, G. Heber, O. Maquelin, X. Tang, and G. R. Gao. *Overview of the Threaded-C language.* Technical Report 19, Computer Architecture and Parallel Systems Laboratory, University of Delaware, Mar. 1998

[13] Russell P. Lentini, Goutham P. Rao, Jon N. Thies, and Jennifer Kaye. *An Extendable Mobile Agent Architecture*. Lockheed Martin Advanced Technology Laboratories.

[14] Colin G. Harrison, David M. Chess, Aaron Kershenbaum. *Mobile Agents: Are they a good idea?*, Research Report, IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY, 10598

[15] John C. Hull. *Options, Futures and Other Derivatives*, Reading, Prentice Hall, Fourth Edition

[16] L. Ismail and D. Hagimont. *A performance Evaluation of the Mobile Agent Paradigm.* ACM SIGPLAN Notices, p. 306-313, 1999

[17] William R. Stanek, Microsoft Windows 2000 And IIS 5 0 Administrators Pocket Consultant, Reading, Microsoft Press, April 2001

[18] Java Database Connectivity (JDBC) ™: Application Programming Interface (API), available at http://java.sun.com/products/jdk/1.2/docs/guide/jdbc/index.html

[19] Java Database Connectivity (JDBC)-Industry Support, available at http://java.sun.com/products/jdbc/industry.html

[20] J. Gosling, B. Joy, and G. Steele, *The Java Language Specification*, Addison-Wesley, Reading, MA, 1996

[21] JDOM Documentation, available at http://www.jdom.org/downloads/docs.html

[22] J. M. Bradshaw, S. Dutfield, P. Benoit, and J. D. Woodley. *KAoS: Toward an Industrial-strength open agent architecture.* In J. Bradshaw, editor, *Software Agents*, p. 375-418. AAAI/MIT Press, 1997

[23] David Kotz, Guofei Jiang, Robert Gray, George Cybenko, Ronald A. Peterson. *Performance Analysis of Mobile Agents for Filtering Data Streams on Wireless Networks.* Mobile Networks and Applications, Computer Science Technical Report Dartmouth College TR2000-366, May, 2000

[24] Robert S. Gray, David Kotz, Ronald A. Peterson Jr., Joyce Barton, Daria Chacon, Peter Gerken, Martin Hofmann, Jeffrey Bradshaw, Maggie Breedy, Renia Jeffers, and Niranjan Suri. *Mobile-Agent versus Client/Server*

Performance: Scalability in an Information-Retrieval Task. Tech Report TR2001-386, Dartmouth College, Jan 2001

[25] Danny B. Lange and Mitsuru Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addition Wesley, Reading, MA, USA, 1998

[26] Blake LeBaron. *Building Financial Markets with Artificial Agents: Desired Goals, and Present Techniques,* Computational Markets, MIT Press, Feb. 1999

[27] Mario Magnanelli. and Moira Norrie. *Databases for Agents and Agents for Databases*. In *Proc. of 2nd International Bi-Conference Workshop on Agent-Oriented Information Systems*, June 2000.

[28] Mobile Code, Agents, and Java, available at http://www.infosys.tuwien.ac.at/Research/Agents/

[29] N. Suri, J. M. Bradshaw, M. R. Breedy, P. T. Groth, G. A. Hill, and R. Jeffers. *Strong Mobility and Fine-Grained Resource Control in NOMADS*. In *Proc. of the second Int'l Symp. on Agent Systems and Applications and Fourth Int'l Symp. On Mobile Agents (ASA/MA 2000)*, volume 1882 of LNCS, p. 2-15, Zurich, Switzerland, September 2000. Springer-Verlag.

[30] Stavros Papastavrou, George Samaras, Evaggelia Pitoura, Mobile Agents for WWW Distributed Database Access. In *15th Int'l Conference on Data Engineering* (ICDE99), Sydney, 1999

[31] Standard Generalized Markup Language (SGML), The World Wide Web, available at http://www.w3.org/MarkUp/SGML/Overview.html

[32] TclXML (TclXML), available at http://www.zveno.com/zm.cgi/in-tclxml/

[33] Leigh Tesfatsion, *Introduction to the Special Issue on Agent-Based Computational Economics,* Journal of Economic Dynamics & Controls Vol. 25 pp. 281-293, Feb. 2001

[34] R. K. Thulasiram, L. Litov, H. Nojumi, C. T. Downing and G. R. Gao. *Multithreaded Algorithms for Pricing a Class of Complex Options,* International Parallel and Distributed Processing Symposium, April 2001

[35] Richard Vlach. *Mobile Database Procedures in MDBAS*. In *Proceedings of Mobility in Databases and Distributed Systems, IEEE, 2001*

[36] The World Wide Web Consortium, available at http://www.w3c.com

[37] Xerces Java Parser (XERCES-J), available at http://xml.apache.org/xerces-j/

[38] Extensible Markup Language (XML), The World Wide Web Consortium, available at http://www.w3.org/TR/REC-xml

[39] The Perl Extension Module XML::Parser, available at http://wwwx.netheaven.com/~coopercc/xmlparser/intro.html

[40] XML Parser for Java (XML4J), available at http://www.alphaworks.ibm.com/tech/xml4j

[41] XML Parser for C++ (XML4C++), available at http://www.alphaworks.ibm.com/tech/xml4c

[42] J. Viescas, *Running Microsoft Access 2000,* Reading, Microsoft Press, 1999