Web Based Agent System using Java

By

Qi Liu

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of

Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

WEB BASED AGENT SYSTEM USING JAVA

BY

QI LIU

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

Abstract

The development of the Internet has been generating more and more demands for online distributed systems. There are many technologies and tools developed that are suitable for building such systems, such as CORBA (Common Object Request Broker Architecture) and Java. Agent technology is another ongoing effort for developing scalable distributed systems. It focuses more on the cooperation and the knowledge sharing among programs. This thesis reviews the existing agent theory and one of the ACLs (Agent Communication Language) - KQML (Knowledge Query and Manipulation Language). It also reviews a Java package called JATLite, which is designed as a common framework for building agent-based systems that speak KQML. A web based agent system is designed and implemented with JATLite using Java. This system accommodates the common needs for communication and interaction in an education system, and provides users with personalized environment and tools which help them with their daily functions.

Acknowledgments

I would like to especially thank my supervisor Dr. R. McLeod, for his wisdom, guidance and encouragement throughout my thesis research. I would also like to give thanks to my examining committee members, Dr. McNeill, Dr. Graham and Dr. Hussain, for their interest in this research and the time and efforts they made to review my work.

I would like to extend my sincere thanks to all others who made the completion of this thesis possible. I want to thank IIC (Internet Innovation Centre) for funding my research. Special thanks to our network administrator Guy Jonatschick for his continuous support which made the process a much smoother sailing.

To my wife Li, I thank you for believing in me and making many things possible.

Last but not least, I want to thank my parents and grandparents, for their unconditional love and support through the years.

Table of Content

List of Figures

List of Tables

CHAPTER 1 *Introduction*

Since the introduction of the World Wide Web and the Internet, the world has witnessed an incredible boost in the use of distributed systems. Many technologies and tools have been developed to support this trend. Among them, agent technology is an important one. This thesis will demonstrate the application of agent technology by building a web based education system called AEA (Adaptive Education Agent) system.

1.1 Internet Explosion

The Internet came into existence in late 1970's as an outgrowth of the ARPAnet. Since then, the Internet has revolutionized the computer and communications world. The invention of the telegraph, telephone, radio, and computer set the stage for this unprecedented integration of capabilities. The Internet is at once a world-wide broadcasting capability, a mechanism for information dissemination, and a medium for collaboration and interaction between individuals and their computers without regard for geographic location.

Users of the Internet and especially of the World Wide Web (WWW) are growing exponentially. For the second half of 1993, the Web had a doubling period of under 3 months, and even today the doubling period is still nearly every 5 months. The great popularity of the Internet makes it a universal platform for thousands of interactive software systems.

1.2 Software Agent

The software world is one of great richness and diversity. Thousands of software products are available to users today, providing a wide variety of information and services in a range of domains. While most of these programs provide their users with significant value when used in isolation, there is increasing demand for programs that can cooperate to exchange information and services with other programs and thereby solve problems that cannot be solved alone.

Agent-based software engineering was invented to facilitate the creation of software able to cooperate. In this approach, application programs are written as *software agents,* i.e. "software 'components' that communicate with their peers by exchanging messages in an expressive agent communication language" [Ketchpel 94].

More about agents and agent communication languages will be presented later in chapter 2.

1.3 Java

The Java programming language is a general-purpose, object-oriented programming language. It has many advantages which make it easy to develop multi-agent software based on the Internet, e.g. platform independence, multi-threading, database access capability, convenient network programming and web browser integration. Java Develop Kit 1.2 (JDK1.2) is deployed in this development, which uses the main Java packages such as java.io, java.lang, etc. and other additional packages, such as javax.servlet and javax.swing. More detail will be presented later in chapter 7 and chapter 8.

1.4 Design Goal

The goal of this thesis is to demonstrate the application of agent technology and tools by building an online multi-agent system. This system is called Adaptive Education Agent (AEA) system. The agent system is designed to enhance the interactions among members of a traditional education system using instant messaging and course-specific news groups. The agent system provides an assignment and marking system to eliminate the paper work and decrease the circulation time. By collecting and classifying the information from all units of this education system, the agent system provides users with personalized environment which makes information accessing and processing more efficient. The structure and methodology used in this project could be suitable for building similar agent systems for large institutions to enhance their efficiencies and performance by distributing role-oriented agents within the institution. The role-oriented agent acts as a personal assistant by doing jobs that could be pre-defined and time consuming for the role, such as searching for a document among internal documents of the institute, or responding to requests from other roles according to the knowledge base the agent has without this role's effort.

1.5 Overview

The remainder of the thesis is organized as follows. Chapter 2 is an introduction on agent technology and KQML (Knowledge Query and Manipulation Language). Chapter 3 introduces a program package called JATLite. Chapter 4 introduces JDBC and mSQL. Chapter 5 introduces server side scripting and CGI. Chapter 6 deals with the design of the protocol for the system. Chapter 7 provides the design of the client agent. Chapter 8 covers the design of the service agent. Chapter 9 is the conclusion and future work.

CHAPTER 2 *Agents and KQML*

2.1 What is an Software Agent?

There are many definitions of software agent by agent researchers trying to divide today's software world into agents and non-agents. As described in the previous chapter, software agents are "software 'components' that communicate with their peers by exchanging messages in an expressive agent communication language" [Ketchpel 94]. Another popular definition is "An agent is a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and process [Shoham 97]". In Shoham's definition, the requirements for continuity and autonomy derives from the expectation that an agent be able to carry out the user's activities in a flexible and intelligent manner that is responsive to changes in the environment without requiring constant human guidance or intervention. Ideally, an agent that functions continuously over a long period of time would be able to learn from its experience, and cooperate with other agents.

To further classify agents usefully, the following listed properties are usually used by agent researchers,

Table 1: Possible Agent Properties

Property	Meaning
reactive	responds in a timely fashion to changes in the environment

Table 1: Possible Agent Properties

Property	Meaning
autonomous	exercises control over its own actions
goal-oriented	does not simply act in response to the environment
temporally continuous	is a continuously running process
communicative	communicates with other agents, perhaps including people
learning	changes its behavior based on its previous experience
mobile	able to transport itself from one machine to another
flexible	actions are not scripted
character	believable "personality" and emotional state

Agents may be classified according to a subset of these properties. The first four properties apply to all agents; the addition of other properties produces useful classes of agents such as mobile agents and learning agents. There are also other possible classifying schemes, such as according to the tasks agents perform.

All software agents are programs, but not all programs are agents. For example, web-based searching and filtering "agents" deploy a one-time query answering mechanisms so they don't fit the property that agents are continuously running processes. As another example, typed-message agents are distinguished from other types of software by their ability to communicate using a shared message protocol such as KQML. In this shared message protocol, at least some of the message semantics are typed and independent of the applications. In other words, this protocol is peer-to-peer protocol. This differentiates typed-message agents from client/server type of software in which a client/server protocol is used to communicate among modules.

2.2 KQML - Knowledge Query and Manipulation Language

2.2.1 Agent Communication Language (ACL)

The main advantages of agent technology is that agents communicate with others using an expressive communication language and thereby work together cooperatively to accomplish complex goals. The researchers in the ARPA Knowledge Sharing Effort have defined the components of the agent communication language (ACL) which consists of three parts - vocabulary, an inner language called KIF (short for Knowledge Interchange Format), and an outer language called KQML (short for Knowledge Query and Manipulation Language). An ACL message is a KQML expression in which the arguments are terms in KIF formed from words in the ACL vocabulary.

The vocabulary of ACL is a large dictionary of words appropriate to common application areas. Each word has an English description for a human to understand its meaning; and each word has formal annotations written in KIF for programs to use.

KIF is a prefix version of first order predicate calculus, with various extensions to enhance its expressiveness. It provides both a specification for the syntax of a language, and a specification for its semantics. KIF can be used to support translation from one content language to another, or as a common content language between two agents that use different native representation languages. Because of the independence of KIF's semantics, each message transmitted among agents has to include any implicit information about the sender, the receiver, the time of the message, and so on. The efficiency of communication can be enhanced by providing a linguistic layer in which context is taken into account. This is the function of KQML.

2.2.2 KQML

Knowledge Query and Manipulation Language (KQML) is a language that is designed to support interaction among intelligent software agents.

Communication takes place on several levels. Being able to locate and engage the attention of another agent with which an agent wishes to communicate is one part of the process, and packaging a message in a way that makes it sharable among agents is another.

When using KQML, a software agent transmits content messages, composed in a language of its own choice, wrapped inside a KQML message. The content message can be expressed in any representation language and written in either ASCII strings or one of many binary notations. KQML implementations ignore the content portion of a message except to recognize where it begins and ends.

KQML is expected to be supported by a software substrate that makes it possible for agents to locate one another in a distributed environment. Most current implementations come with custom environments of this type; these are commonly based on helper programs called routers or facilitators. These environments are not a part of the KQML specification. A facilitator is an agent that performs various useful communication services, e.g. maintaining a registry of service names, forwarding messages to named services or routing messages based on content.

The KQML language allows KQML messages to carry useful information, such as the names and addresses of the sending and receiving agents, a unique message identifier, and notations by any intermediate agents. There are also optional features of the KQML language which contain descriptions of the content: its language, the ontology, and some type of more general description. These optional features make it possible to support environments to analyze, route and deliver messages based on their content, even though the content itself is inaccessible.

The forms of these parts of the KQML message may vary, depending on the transport mechanism used to carry the KQML messages. In implementations which use TCP streams as the transport mechanism, they appear as fields in the body of the message. In other transport mechanisms the syntax and content of these message may differ. For example, in the E-mail implementation of KQML, these fields are embedded in KQML mail headers.

The set of performatives forms the core of the language. It determines the kinds of interactions one can have with a KQML speaking agent. The primary function of the performatives is to identify the protocol to be used to deliver the message and to supply a speech act that the sender attaches to the content. The performative signifies that the content is an assertion, a query, a command, or any other mutually agreed speech act. It also describes how the sender would like any reply to be delivered.

2.2.3 KQML String Syntax

As mentioned, a KQML message is also called a *performative*. A performative is expressed as an ASCII string using certain syntax. In performatives, parameters are identified by keywords, which are called parameter names. The parameter names must begin with a colon (:) and precede the corresponding parameter value.

The syntax of KQML is based on a balanced parenthesis list. The initial element of the list is the performative; the remaining elements are the performative's arguments as keyword/value pairs.

Following is KQML string syntax,

<performative> ::= (<word> {<whitespace> :<word><whitespace> <expression>}*)

<expression ::= <word> | <quotation> <string> | (<word> {<whitespace> <expression>}*)

<word> ::= <character><character>*

<character> ::= <alphabetic> | <numeric> | <special>

<special> ::= < > | = | + | - | * | / | & | ^ | ~ | _ | @ | $ | % | : | . | ! | ?

<quotation> ::= '<expression> | '<comma-expression>

<comma-expression> ::= "<stringchar>*" | #<digit><digit>*"<ascii>*

<stringchar> ::= \<ascii> | <ascii>-\-<double-quote>

Note: "*" means any number of occurrences, "-" indicates set difference, and "|" means a choice "OR".

2.2.4 KQML Semantics

The semantic model of KQML is a simple, uniform context for agents to view each others' capabilities. Each agent manages a KB (Knowledge Base). Communication with the agent is with regard to its KB, e.g., questions about what a KB contains, requests to add or delete a statement from the KB, or requests to use knowledge in the KB to route messages to appropriate agents.

The implementation of an agent may use a database system or a special data structure as a VKB (Virtual Knowledge Base), as long as wrapper code translates that representation into a knowledge base abstraction for other agents to understand.

The statements in a VKB can be classified into two categories: beliefs and goals. An agent's beliefs contain information about itself and its environment, including the VKBs of other agents. An agent's goals contain states of its environment that the agent will act to achieve. Performative definitions make reference to either or both of an agent's goals and beliefs. Agents exchange their content of VKB using KQML, but statements in VKBs can be represented by languages other than KQML.

2.2.5 Reserved Performative Parameters

Table 2: Reserved Performative Parameters [KQML Specification]

Keyword	*Meaning*
:content	the information about which the performative expresses an attitude
:force	whether the sender will ever deny the meaning of the performative
:in-reply-to	the expected label in a reply
:language	the name of representation language of the :content parameter
:ontology	the name of the ontology (e.g., set of term definitions) used in the :content parameter
:receiver	the actual receiver of the performative
:reply-with	whether the sender expects a reply, and if so, a label for the reply
:sender	the actual sender of the performative

2.2.6 Reserved Performative Names

Table 3: Reserved Performative Names for sender S and recipient R [KQML Specification]

Name	*Meaning*
achieve	S wants R to make something true of their environment
advertise	S is particular-suited to processing a performative
ask-about	S wants all relevant sentences in R's VKB
ask-all	S wants all of R's answers to a question
ask-if	S wants to know if the sentence is in R's VKB
ask-one	S wants one of R's answers to a question
break	S wants R to break an established pipe
broadcast	S wants R to send a performative over all connections
broker-all	S wants R to collect all responses to a performative
broker-one	S wants R to get help in responding to a performative
deny	the embedded performative does not apply to S (anymore)

Table 3: Reserved Performative Names for sender S and recipient R [KQML Specification]

Name	*Meaning*
delete	S wants R to remove a ground sentence from its VKB
delete-all	S wants R to remove all matching sentences from its VKB
delete-one	S wants R to remove one matching sentence from its VKB
discard	S will not want R's remaining responses to a previous performative
eos	end of a stream of responses to an earlier query
error	S considers R's earlier message to be mal-formed
evaluate	S wants R to simplify the sentence
forward	S wants R to route a performative
generator	same as a standby of a stream-all
insert	S asks R to add content to its VKB
monitor	S wants updates to R's response to a stream-all
next	S wants R's next response to a previously-mentioned performative
pipe	S wants R to route all further performatives to another agent
ready	S is ready to respond to R's previously-mentioned performative
recommend-all	S wants all names of agents who can respond to a performative
recommend-one	S wants the name of an agent who can respond to a performative
recruit-all	S wants R to get all suitable agents to respond to a performative
recruit-one	S wants R to get another agent to respond to a performative
register	S can deliver performatives to some named agent
reply	communicates an expected reply
rest	S wants R's remaining responses to a previously-mentioned performative
sorry	S cannot provide a more informative reply
standby	S wants R to be ready to respond to a performative
stream-about	multiple-response version of ask-about
stream-all	multiple-response version of ask-all
subscribe	S wants updates to R's response to a performative

Table 3: Reserved Performative Names for sender S and recipient R [KQML Specification]

Name	*Meaning*
tell	the sentence in S's VKB
transport-address	S associates symbolic name with transport address
unregister	a deny of a register
untell	the sentence is not in S's VKB

2.2.7 Performative Classification

A KQML performative consists of a performative name, a set of parameters and their associated arguments. According to the different purposes of performatives, they can be classified into 11 types, which are basic informative performative, database performative, basic response performative, basic query performative, multi-response query performative, basic effector performative, generator performative, capability-definition performative, notification performative, networking performative, facilitation performative. For instance, performatives such as "insert" and "delete" provide a capability to allow one agent to tell another to add or delete sentences in its VKB, therefore they are database performatives.

CHAPTER 3 *Introduction of JATLite*

JAT*Lite* stands for "Java Agent Template Lite" [JATLite]. It is a program package written in Java which has the capability of creating software agents. JAT*Lite* provides basic communication tools and templates based on TCP/IP and especially, facilitates the development of agents that exchange KQML messages. It is used in the AEA system as the communication layer to provide communicating capability.

The Figure 3.1 [JATLite] is the basic JAT*Lite* infrastructure. Agents register with an Agent Message Router using a user name and password, and then exchange information with other agents through the router.

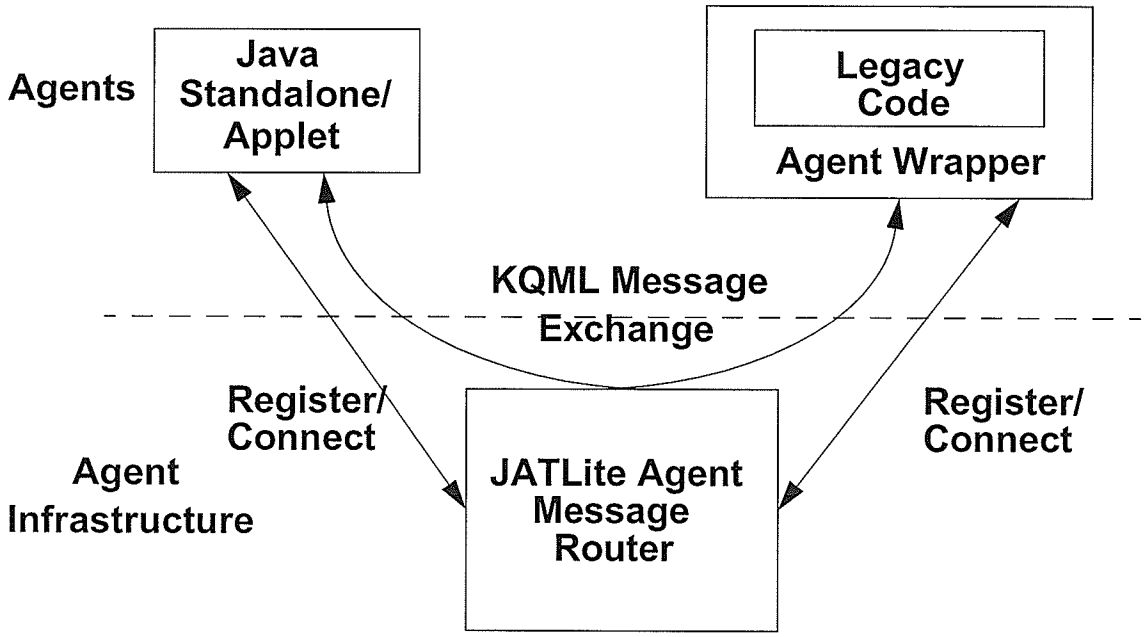

Figure 3.1 JATLite Agent Message Router

3.1 JAT*Lite* Architecture

The JATLite architecture has a hierarchy of specialized layers which are: Abstract Layer, Base Layer, KQML Layer, Router Layer and Protocol Layer. Developers can select the proper layer from which to build their systems.

- **Abstract Layer**

The Abstract Layer provides abstract classes for the implementations of higher layers. It includes AgentAction, ServerThread, ReceiverThread, MessageBuffer, ConnectionTable, Address, AddressTable and Security.

- **Base Layer**

The Base Layer provides TCP/IP communication tools without specifying any higher level protocols for connections. A user can build the higher level protocols as they wish and this makes Base Layer very flexible.

- **KQML Layer**

The KQML Layer is on top of Base Layer. It provides classes for storage and parsing KQML messages and restricts that only KQML will be used as communication language among agents.

- **Router Layer**

The Router Layer is on top of the KQML Layer. It provides registration and message routing and querying for agents.

- **Protocol Layer**

The Protocol Layer is on top of Router Layer. It supports various standard Internet services such as FTP. The Internet services can be used by both applications and applets. Standalone application agents can use built-in services without going through the router. Due to the fact that applet agents are only allowed to connect to the server spawned itself, the applet agents need router proxy services to implement Internet services.

3.2 JAT*Lite* Router

The JAT*Lite* router is a software application which receives messages from registered agents and routes them to the destination agents according to the receiver address in the messages.

Usually when an agent wants to send a message to another agent, it will try to build up a direct connection to the receiver agent. If the connection is created successfully, the sender agent will send the message. This mechanism is called message polling. If the direct connection couldn't be built for reasons like the receiver agent is an applet or the receiver agent is off-line at the moment, there would be a problem for the sender agent to send the message. Therefore, it is necessary for the router to have a message queue and routing mechanism to store the messages and send them as soon as the receiver agents are ready. In this way, agents do not need to be concerned about whether the receiver agent is available for the messages at the moment.

3.2.1 Router Features

- **Message Queuing**

All the messages that the router received will be queued to the file system and retrieved or deleted according to the requests from receiver agents.

- **Message Routing**

Based on the receiver information in messages, router can route them to the right agents whenever they are ready to receive them.

- **Agent Name Service**

Each agent does not need to maintain all the possible receivers' addresses. The agent can ask the Agent Name Service about the receiver address whenever it wants to send a message but does not know the receiver's address. The router is a convenient Agent Name Server which maintains the agent addresses.

- **Register**

Any agent which wants to use the router to communicate with another agent needs to register with the router first.

- **Security**

Every agent registered with the router has a name and password which will be used by the router to identify the agent before accepting connection from it. The router uses name and password pairs to provide basic security.

- **List-Agents**

The router is able to answer the agent with the list of registered agents when requested.

- **Disconnect**

The agent can request to disconnect with the router by sending a 'disconnect-agent' message to the router.

3.2.2 Router Components

- **The Router Server**

The router server is a Java application program with all the features described above. If one wants to use applet agents with the router, the router should be running on the HTTP server machine. If the agents are all stand-alone applications, the router could be running on any machine.

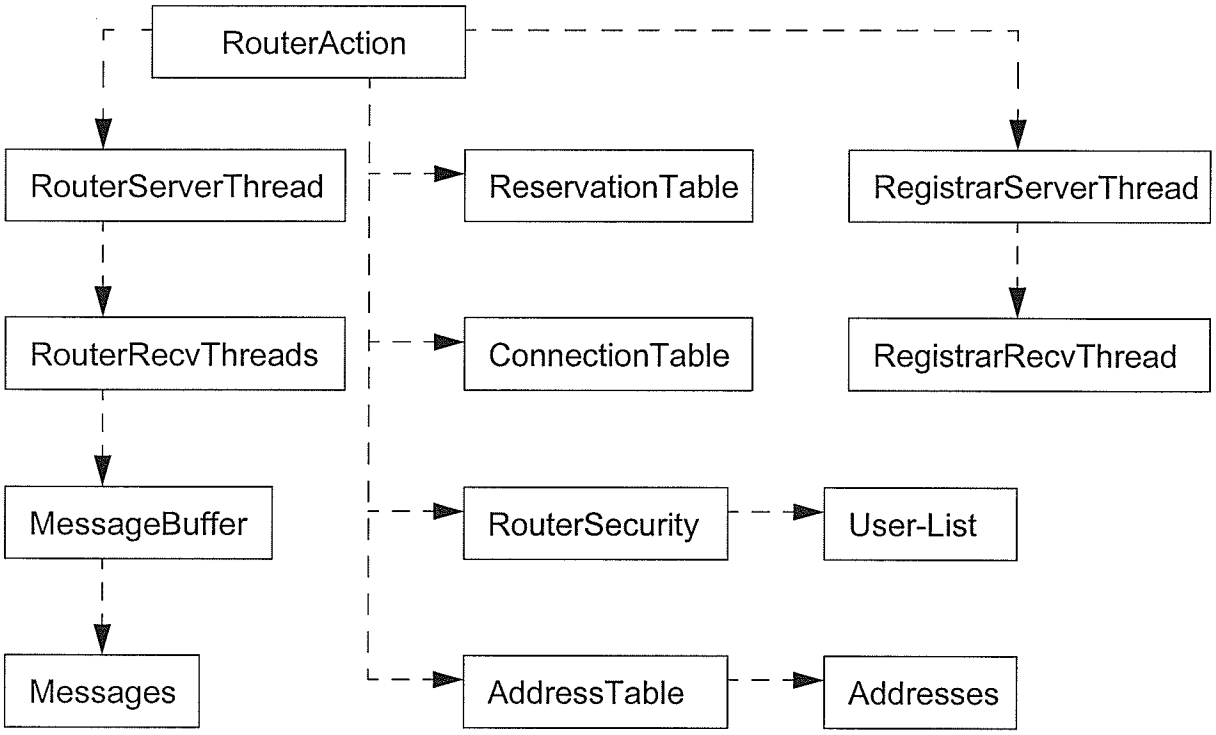

Figure 3.2 The RouterServer Component Diagram

Figure 3.2 [JATLite] illustrates the components and their relationships within a router server.

1. RouterAction initializes all the data members and manages reserved messages

2. RegistrarAction runs as a thread to receive an agent registration. It has one RegistrarServer-Thread which accepts the connections from agents which want to register with the router. RegistrarRecvThread is then created to receive information for registration.
3. ReservationTable contains the reserved message information.
4. ConnectionTable contains the current on-line receiver threads. Each receiver thread has the same name as the agent name.
5. RouterSecurity secures the incoming connections by checking whether the agent satisfies the connection protocol.
6. AddressTable contains the address information of all the registered agents.
7. RouterServerThread accepts the connections from registered agents and creates Router-RecvThread.
8. RouterRecvThread receives the message from the connection and routes the message to the receiver. It has message queue to buffer the received messages.

- **The Router Client**

The router client can be a Java application agent or a Java applet agent. RouterAction is the main class in it. Figure 3.3 [JATLie] shows the minimum components for the router clients. They can be described as follows:

1. ConnectionTable contains connection information; Connection can be with the router or other agents.
2. RouterSecurity sends connection protocol message and receives disconnection protocol message.
3. AddressTable contains address information for the router and other agents.

4. MessageQueue stores the retrieved messages from the MessageBuffer.
5. RouterRecvThread is connected to the router to send messages to and receive messages from the router.
6. MessageBuffer buffers the received messages from the router.

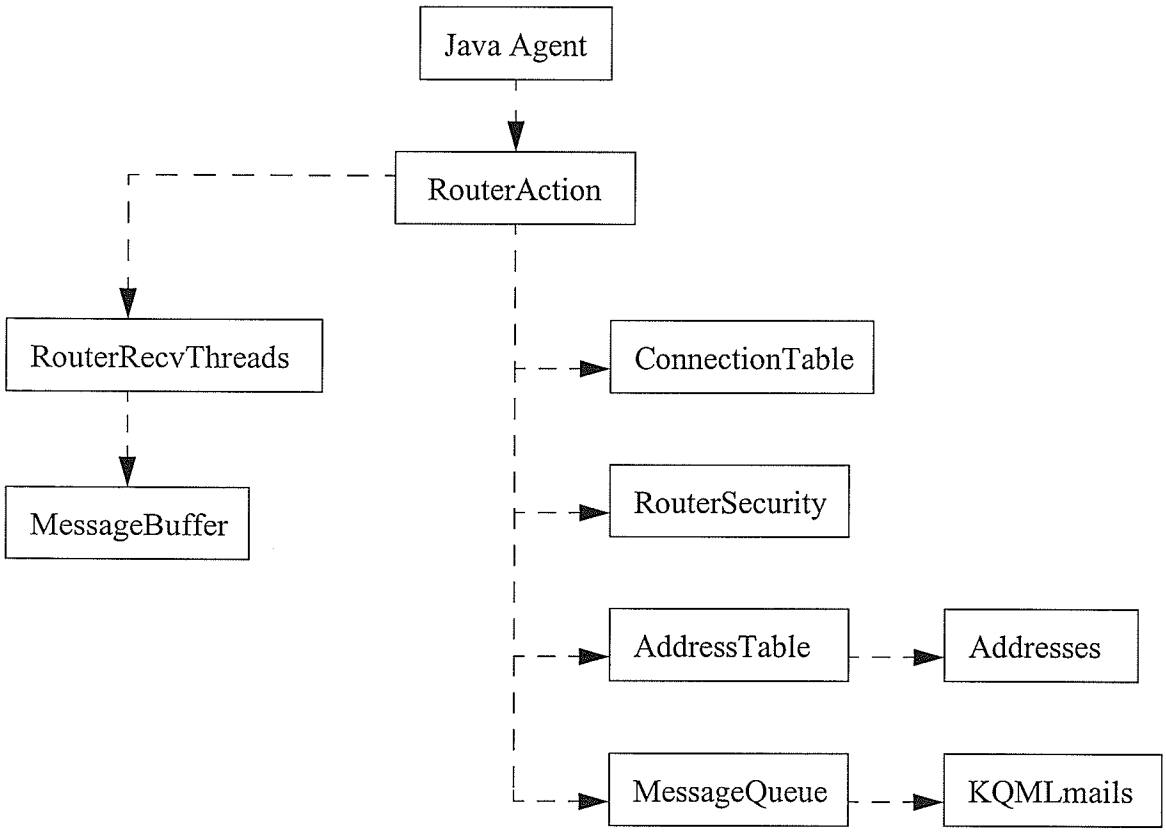

Figure 3.3 The RouterClient Component Diagram

CHAPTER 4 JDBC and mSQL

Like other applications in the area of information technology, information storage is a big concern and a variety of database products are often used to address it. A powerful database means fast access to the information and enhanced overall performance, but it also means that it needs a large disk space for installation and running. Since the Adaptive Education Agent system is designed and developed under a personal account in the ECE department, the resources are insufficient to install and run big and powerful database product such as Oracle. Therefore, a small and more portable mSQL (miniSQL) was chosen instead. This chapter will cover briefly the technologies used to develop the information storage part for AEA.

4.1 JDBC

4.1.1 Introduction

JDBC stands for "Java Database Connectivity" [Java]. It is a Java package designed to allow Java programs to connect to a database and query or update it using the industry standard query language. One advantage of using JDBC is that the Java database program can run on different platforms as long as it has the Java Runtime Environment (JRE) installed; another advantage is that the data can be transferred from one type of database (such as Microsoft SQL server) to another (such as Oracle), and the Java program can still read and write the data.

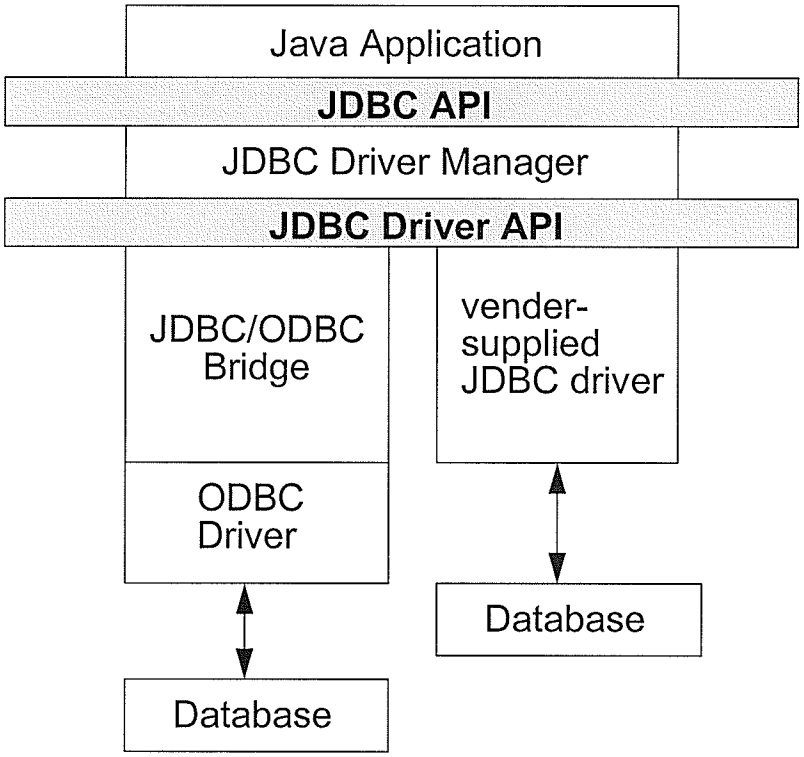

Figure 4.1 JDBC Working Structure

Figure 4.1 [Cornell 96] is the JDBC working structure. The JDBC consists of two layers. The top layer is the JDBC API (application programming interface). Java applications use this API to communicate with the JDBC driver manager, sending SQL statements. The second layer is the JDBC driver API. It defines the interface to the JDBC driver manager, and existing database venders follow this API to write their device drivers if they want their database products to work with JDBC. Another way for JDBC to communicate with various database products is through an ODBC driver. Open DataBase Connectivity (ODBC) [Cornell 96] is an API developed by Microsoft to allow users to abstract a program from a database. It provides the programmer with one interface to work with different databases. Because many ODBC drivers were already devel-

oped prior to JDBC being introduced, a JDBC/ODBC bridge was built between a JDBC driver and a ODBC driver in order to use the current ODBC drivers.

4.1.2 Basic Concepts

- **Database URL**

When using JDBC, the database is located by a URL (Uniform Resource Locator). Just like people need a WWW URL (web address) to go to the web site. A database URL is the database address for people to connect to a specific database. The format of the URL is **jdbc:driver_name:source_name**. The driver_name will be the driver used between JDBC and the database. For example, if ODBC data is accessed, the URL will be **jdbc:odbc:source_name** which means JDBC/ODBC bridge is used to access the data source under the source_name in the database. The format of the source_name follows the standard URL naming convention of **//host-name:port/other**. The "hostname" is used to locate the machine on which the database engine is running. The "port" is used by the database engine to receive the queries. The "other" contains the name of the data source, and may contain the user name and password if they are required to access the database.

- **Connecting to a Database**

The requirements to connect to a database are the correct URL and the account information (username and password). Given the database URL, the JDBC manager will try to find a driver specified in the URL. If it succeeds, a connection will be created to the database for SQL queries and results. Here is an example how it is done using JDBC,

Connection con = DriverManager.getConnection("jdbc:odbc:mydb", "user", "passwd");

The **DriverManager** is the class for loading database drivers and creating database connections. The **Connection** class describes the database connections and is a result of a successful loading of the database driver.

- **Executing Queries**

A **Statement** object is used to execute queries in JDBC. It is obtained from the Connection object which is returned when making the connection to the database.

```
Statement stmt = con.createStatement();
```

Queries then can be sent via the Statement object to the database and a **ResultSet** object will be returned holding the results of executing the queries by the database.

```
ResultSet rs = stmt.executeQuery(query);
```

The results are accessible by using the accessor methods of the ResultSet object. There are accessor methods for every Java data type, such as getString and getInt. They can take numeric argument as the column number or string argument as the name of the column to get the value of the specific column.

4.2 mSQL

mSQL (mini Structured Query Language) is a lightweight database engine designed by Hughes technologies to provide fast access to stored data with low memory requirements. The version used is mSQL 2.0 which is a second generation of mSQL. It has many improvements over mSQL 1.0 such as enhanced indexing, new data types and redesigned server architecture. These improvements provide mSQL with better handling of complex queries and large data sets.

4.2.1 mSQL 2.0 Language Specification

The industry standard query language currently is ANSI(American National Standards Institute) SQL. As the name indicates, mSQL covers a subset of features provided by ANSI SQL. It allows programs to create, manipulate and retrieve data while some other relational capabilities are not supported, such as views and nested queries.

- **The Create Statement**

The create statement is used to create tables, indices, and sequences and there are three corresponding constructs,

```
CREATE TABLE table_name (
        col_name col_type [ not null ]  [ , col_name col_type [ not null ] ] ...
)


CREATE [ UNIQUE ] INDEX index_name ON table_name (
        field_name,  [ , field_name ] ...
)

CREATE SEQUENCE ON table_name [ STEP step_val ] [ VALUE initial_val ]
```

The CREATE TABLE statement creates a new table definition with specified column names and column types. There are four column types: char(length), text(length), int and real. "Not null" is optional and with it, all the fields in that column must have a value of the type specified. CREATE INDEX creates indices on certain table fields and these indices normally increase performance when a query contains these certain fields in its WHERE statement. CREATE SEQUENCE creates a sequence for a table with a start value and an increasing or decreasing value. A table can only contain one sequence, and it is accessed by selecting the _seq system variable from the table.

- **The Drop Statement**

As the opposite of the create statement, the drop statement is to remove a structure from the database. The structure can be a table, an index or a sequence. The syntax of the drop statement is given below,

```
DROP TABLE table_name

DROP INDEX index_name FROM table_name

DROP SEQUENCE FROM table_name
```

- **The Insert Statement**

The insert statement adds new data entries into a table in the database. The valid syntax is below,

```
INSERT INTO table_name [ ( column [ , column ] ... ) ]
VALUES (value [, value] ... )
```

The number and order of the values must match the number and order of column names. If the column names in the statement are omitted, the values must be in the order the columns were defined.

- **The Select Statement**

The select statement allows users to query the database with certain condition to get data entries that fit the criteria. The syntax of select statement is,

```
SELECT [table.]column [ , [table.]column ] ...
        FROM table [ = alias] [ , table [ = alias] ] ...
        [ WHERE [table.] column OPERATOR VALUE
          [ AND | OR [table.]column OPERATOR VALUE] ... ]
        [ ORDER BY [table.]column [DESC] [, [table.]column [DESC] ]
```

The OPERATOR can be <,>,=,<=,>=,<>,LIKE,RLIKE or CLIKE. VALUE can be a literal value or a column name. The ORDER keyword is to sort the returned data in either ascending or descending order.

- **The Delete Statement**

The delete statement allows users to remove one or more data entries from a table as long as the condition is defined in the same way as it is in the select statement. The syntax of the delete statement is,

```
DELETE FROM table_name
        WHERE column OPERATOR value
        [ AND | OR column OPERATOR value ] ...
```

- **The Update Statement**

The update statement allows users to modify the data entries in the table. This can be done to one or more data entries which fit the condition specified by the WHERE construct. The syntax of the update statement is,

```
UPDATE table_name SET column=value [ , column=value ] ...
        WHERE column OPERATOR value
        [ AND | OR column OPERATOR value ] ...
```

4.2.2 mSQL 2.0 System Variables

System variables are variables built to the database engine. They are used to provide access to server maintained information or meta data relating to the database. The system variable's name contains a leading underscore as an identifier which is not valid for table names or field names.

- **_rowid**

The _rowid system variable provides a row identifier for any row in a table. When it is queried and identified, it holds the internal record number used by the database engine to access that row. The following is an example of how to query the _rowid and use it,

```
SELECT _rowid, content FROM mail_order

WHERE number='1001'

UPDATE mail_order SET price='50'
WHERE _rowid=44
```

The first query got the _rowid value and the "content" value of the data entry which has a "number" value of 1001. Suppose the _rowid value is 44, it can then be used to identify that specific row by the database engine to do other queries, such as changing the price for that order. This mechanism provides the fastest access by the engine.

- **_timestamp**

The _timestamp system variable provides the system time at which a row in the table was modified last time. It is not designed for applications to use. Rather, it is a reference that the database engine can use to determine if one row was created before or after another. It can also be used to synchronize a remote database for database replication.

- **_seq**

The _seq system variable provides access to the current sequence value of the table.After the current sequence is returned, the sequence is updated to the next value in the sequence.

- **_sysdate**

The _sysdate system variable provides the current date and time on the server machine in standard UNIX time format. This variable can be selected from any table.

- **_user**

  The _user system variable returns the user which submitted the query.

4.3 JDBC & mSQL

As described earlier, JDBC has two ways of working with various database products. One is through vendor-supplied JDBC drivers which are made specifically for different database products. Another way is using a more generic JDBC/ODBC bridge to work with already existing ODBC drivers for different database products. In this thesis we are using the first approach, i.e., vendor-supplied JDBC driver. The product used here is mSQL-JDBC 1.1b1 developed by Imaginary. The following codes shows an example of creating a connection with the mSQL database using msql-jdbc driver.

```
String msqlDriver = "com.imaginary.sql.msql.MsqlDriver";
String url = "jdbc:msql://mcleod2.ee.umanitoba.ca:1114/AEA";
Connection con = null;
Statement stmt = null;
try {
    Class.forName(msqlDriver);  // force loading of driver
    con = DriverManager.getConnection(url, s_user, s_password);
    stmt = con.createStatement();
}
catch(Exception e) {
    System.out.println(e);
 }
```

CHAPTER 5 *The Scripting World*

Today's on-line world is powered by numerous programs running on web servers. They provide dynamic content generated by users' inputs and inquiries, such as searching results on the keyword a user types or the flight information the flight the user selects. The AEA (Adaptive Education Agent) system relies on information provided by such facilities, and in this case, they are server side programs running on the website of University of Manitoba. It is an important function for an agent system to be capable of communicating with other programs or agents. Much of the information an agent needs may already exist or may be impossible for it to generate by itself. Also, there are existing programs designed to do different jobs, and it makes more sense for an agent system to utilize them to perform the same functions. We call those server side programs external programs here because they are outside the AEA system. Those external programs can be classified into two categories, CGI based scripting and server-side scripting (SSI). CGI based scripting includes CGI programs, and server-side scripting includes Netscape's server-side JavaScript technology, Microsoft's Active Server Pages technology, Java Servlets and JavaServer Pages, etc. Even though they have pros and cons in terms of implementation and performance, they all share a common feature: they receive requests from clients, process requests and send results back. The CGI scripting is portable across all the server platforms, whereas SSI is not due to its proprietary nature of technology, while SSI is more computationally efficient. The following briefly introduces CGI scripting and server side scripting.

5.1 CGI

- **Introduction**

The Common Gateway Interface (CGI) acts as middleware among web servers, external databases, and information sources. CGI scripts or programs are extensions to the core functionality of a web server. They perform specific information-processing, retrieval, and formatting tasks on behalf of their web servers. CGI defines a method for the web server to accommodate additional programs that are used to access external applications from within the context of web documents. The following diagram describes the processing done with CGI.

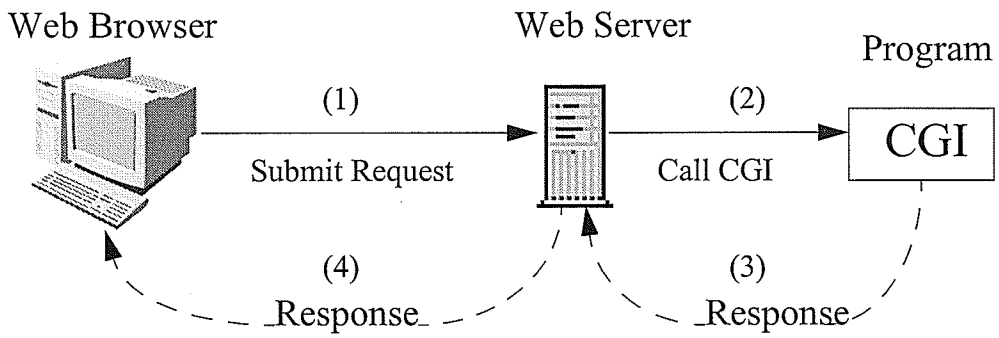

Figure 5.1 CGI Processing

- **CGI Data Input**

HTML forms are usually used to transmit user customized information to a web server, which invokes the proper CGI application to process the information and send back the results. In order to provide the web server with the information, there are some necessary components in a HTML form: HREF or ACTION, METHOD, and ENCTYPE.

HREF or ACTION specifies the URL of the CGI application the form content will be submitted to. It tells the web server which CGI program this request needs to be sent to.

METHOD is the HTTP method used to submit the fill-out form to the web server. There are two types of request methods, GET and POST. GET is the default method, which appends form content to the URL as with a normal request. An example of a request using the GET method is, <A HREF="http://www.home.com/cgi-bin/test.cgi?question+one">. The GET method limits the number of characters it can transfer between the web browser and the web server to a maximum of 255 characters. The POST method causes the HTML form contents to be sent to the web server in a data block rather than as part of URL, and it does not have the length limitation of the request. Here is an example of a POST request using HTML form,

```
<FORM METHOD="POST" ACTION="http://www.home.com/cgi-bin/test.cgi>
<INPUT NAME="question1">
<INPUT NAME="question2">
<INPUT TYPE="submit" VALUE="Submit Query">
</FORM>
```

ENCTYPE specifies the encoding for the form contents. It applies only if the method is POST and it has two possible values: the default and application/x-www-form-urlencoded.

There are also interface elements in the form itself to let users fill in information for submitting to CGI applications, such as INPUT, TEXTAREA, etc. After HTML form data is submitted to the web server, values of a set of variables called environment variables are set according to the request by the web server.

Environment variables are entities that exist within a particular computer environment. They contain the information about the web server, external programs and client requests. Therefore,

environment variables known to the web server and CGI are used to pass data about the HTTP request from a server to the CGI application. These variables are accessible to both the server and any CGI application that the web server invokes.

If the request method is GET, the CGI program reads the request string from QUERY_STRING. If the request method is POST, the CGI program will determine the size of the request using CONTENT_LENGTH and read that amount of data from standard input.

The web server may also use command-line arguments to pass data about an information request from a client. This technique is used only in the case of an HTML <ISINDEX> request.

- **CGI Data Output**

After the CGI program processes the request from the client, it sends the generated HTML document back to the client web browser in the standard way.

- **Sample CGI Code**

Usually, the clients for CGI programs are web browsers such as Internet Explorer and Netscape. The clients can also be Java programs, either stand-alone or Java applet. They send the same requests as web browsers send to the web server. The mechanism stays the same. The client program needs to build a socket connection with the web server on the host machine where the CGI program resides and send a request in the exact format that a web browser would send. A stand-alone Java programs can access CGI program on any host while a Java applet can only access the ones on the host it is loaded from. The following is an example,

```
try
{
   //build raw socket to web server
   Socket socket = new Socket("www.home.com", 80);
```

```
    //initialize input and output streams
    outs = sock.getOutputStream();
    ins = sock.getInputStream();

    dataout = new DataOutputStream(outp);
    datain = new DataInputStream(inp);

    if (method == POST)
    {
       //send the HTTP HEADER to web server
       dataout.writeBytes("POST test.cgi HTTP/1.0\n" +
                          "Access: text/plain\n" +
                          "User-Agent: CGIAccess/1.0\n" +
                          "Content-type: application/x-www-form-urlencoded\n" +
                          "Content-length: " + CommArea.length() + "\r\n\r\n");
       //send the request content
       dataout.writeBytes(CommArea);
    }
    else if (method == GET)
    {
       //send the HTTP HEADER & request content
       dataout.writeBytes("GET test.cgi?" + CommArea + " HTTP/1.0\r\n" +
                          "User-Agent: CGIAccess/1.02\r\n\r\n");
    }
 }
 catch(Exception e) {}
```

5.2 Netscape Server Side JavaScript

Unlike a CGI program, server-side JavaScript (SSJ) can only be implemented in a Netscape web server. Figure 5.21 shows how SSJ fits into a Netscape web server. Inside the web server, the SSJ runtime environment is built from three main components, the JavaScript runtime library, the database access server and the Java virtual machine. The JavaScript application manager runs on top of server-side JavaScript, as do any applications the user created.

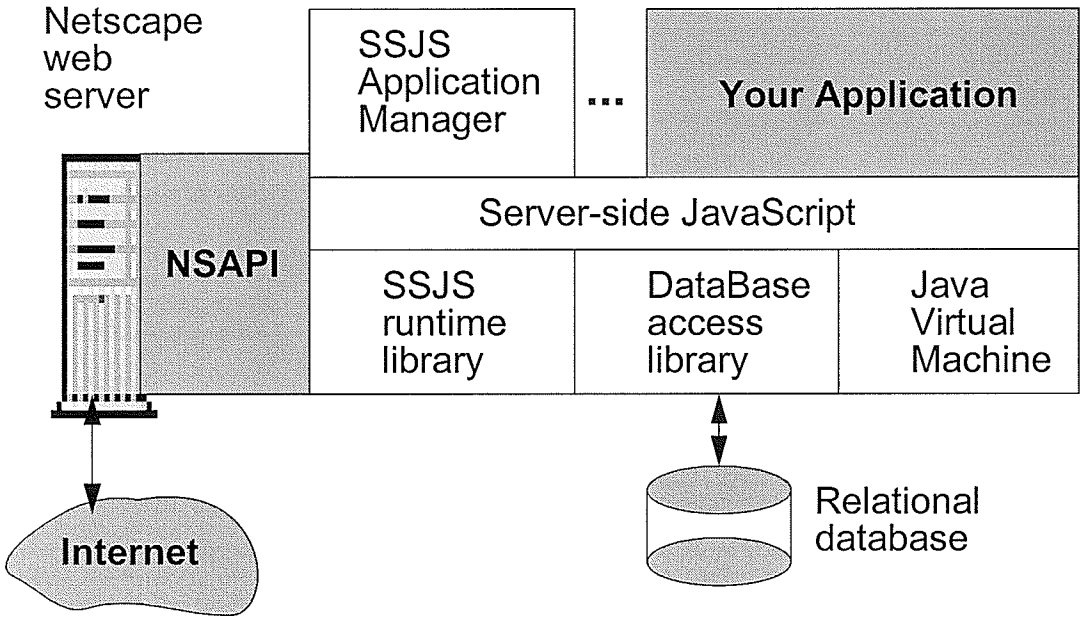

Figure 5.2 SSJ in Netscape

The JavaScript runtime library provides basic JavaScript functionality. The database access library extends the base server-side JavaScript functionality with classes and objects that provide seamless access to the external database server; The Java virtual machine is not only for used with JavaScript, but any Java application running on the server.

The basic procedure for a user to access a JavaScript application that is installed on a web server is,

1. A user accesses the application URL with a web browser, and the web browser sends the request to the server for a page in the application.

2. If the request is to a page under the application URL, the JavaScript runtime engine finds the file corresponding to that URL.

3. The runtime engine constructs an HTML page to send to the client in response. It runs the byte codes associated with SERVER tags from the original source code HTML, creating an

HTML page based on those byte codes and any other HTML found in the original. This process may involve accessing databases and complex computations.

4. The runtime engine sends the new HTML page to the client (web browser).

5. The JavaScript runtime engine inside the web browser interprets any client-side JavaScript statements, formats HTML output, and displays results to the user.

For example, assume the client requests a page with this source:

```
<html>
<head> <title> Add New Customer </title> </head>

<body>
  ...
<server>
if ( project.lock() ) {
  project.lastID = 1 + project.lastID;
  client.customerID = project.lastID;
  project.unlock();
}
</server>

<h1>Add a New Customer </h1>
<form method=POST action="add.htm">
<p>ID:
<br>
<server>write("<B>" + project.lastID + "</B>");
</server>
  ...
</form>
</body>
</html>
```

When this page is accessed, the runtime engine on the server executes the code associated with the SERVER tags. If the new customer ID is 38, the server sends the following HTML page to the client to be displayed,

```
<html>
<head> <title> Add New Customer </title> </head>

<body>
```

```
...
<h1>Add a New Customer </h1>
<form method="post" action="add.htm">
<p>ID:
<br><B>38</B>
...
</form>
</body>
</html>
```

Like client-side JavaScript, server-side JavaScript implements the JavaScript 1.2 language completely.

5.3 Microsoft Active Server Pages (ASP)

Microsoft Active Server Pages (ASP) is another server-side scripting technology that works with Microsoft's Internet Information Services (web server). An ASP page is an HTML page that contains server-side scripts and has a ".asp" file extension. The runtime processing of ASP is similar to that of Netscape server-side JavaScript. The web server calls the ASP engine to process the request whenever the web browser requests an .asp file. After ASP executes all the script commands contained in that ".asp" file, it will return a standard web page to the web browser. For ASP, the default scripting language is VBScript but many other languages may also be used. Below is an example of an ASP page,

```
<%@ Language=VBScript %>
<html>
<head>
<title>Example</title>
</head>
<body>
<%
FirstVar = "Hello world!"
%>
<%FOR i=1 TO 3%>
<%=FirstVar%>
```

```
<%NEXT%>
</body>
</html>
```

Those lines in bold are VBScript commands, and the percent sign with brackets are used as delimiters around the tags. After the ASP engine processes these commands, it will send the following result back to web browser,

```
<html>
<head>
<title>Example</title>
</head>
<body>
Hello World!Hello World!Hello World!
</body>
</html>
```

5.4 Java Servlets

- **Introduction**

Java servlets [Java] are generic server extension. Servlets are based on a Java class that can be loaded dynamically to expand the functionality of a web server. It runs inside a Java Virtual Machine (JVM) on the server.

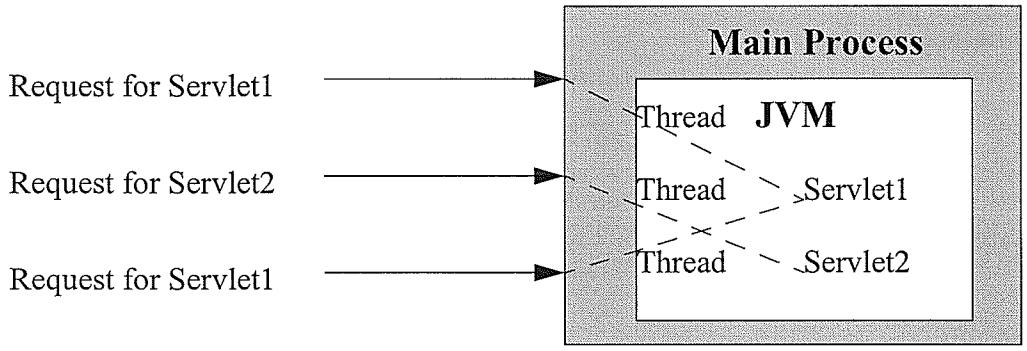

Figure 5.3 Servlet Processing

Servlets have advantages over traditional CGI programs. Servlets use separate threads within the web server process while CGI programs use multiple processes to handle separate requests. The way how servlets handle requests saves time by not starting and terminating processing for each request. Servlets are also portable across operating system and web servers due to the fact that they run inside a JVM. Also, servlets can make use of all the core Java APIs and third party code to handle various kinds of tasks.

- **Communicating With Java Servlets**

As discussed, web clients (browsers or programs) use HTTP methods, such as GET and POST to send queries to CGI programs. While CGI programs use standard input and environment variables to get the query data, servlets use classes and interfaces from the Servlet API to achieve the same goal. The Servlet API has two classes and every servlet needs to extend one of them: javax.servlet.GenericServlet and javax.servlet.http.HttpServlet. The HTTP servlet is the most used Java Servlet that handles HTTP requests as a web server's extension. It is the only servlet type with which programs written here will communicate. For simplicity of presentation, the HTTP servlets are simply called servlets from this point forward. Every servlet has a doGet() method and a doPost() method which will be invoked by the web server each time the servlet is requested. These two methods accept two parameters: a request object and a response object. The request object contains the request data sent from the web client and the servlet writes the response to the client within the response object. As the name of the method suggests, the doGet() method handles the GET queries and doPost() method handles POST queries.

For a Java programs to communicate with a servlet, there are a number of options. Text-based HTTP communication is the most simple and straightforward approach, in which the client program sends requests in the same format discussed in the CGI section to the servlet using an HTTP

connection. The servlet then processes the request and sends back the result to the client program. The second option takes advantage of the fact that the client programs and servlets are both written in Java. Object-based HTTP communication allows the Java programs and servlets to send Java objects for handling request and response using HTTP connections. There are also two other options: RMI (Remote Method Invocation) and CORBA (Common Object Request Broker Architecture). RMI allows one Java object to invoke methods on another Java object running on a different machine. CORBA has the same capability as RMI does without the limitation that programs must be written in Java. A C++ object can therefore communicate with a Java object using CORBA and its IIOP (Internet Inter-ORB Protocol).

5.5 JavaServer Pages

JavaServer Pages (JSP) [Java] are an extension of the Java Servlets. Usually, the content that a servlet returns to a web client is prepared inside the servlet program with HTML tags and content embedded. As such every time something is changed in the content, the servlet source code needs to be changed, compiled and restarted. This is not an efficient way of doing things. To solve this, JSP uses tags and scriptlets to encapsulate the logic that generates the content. JSP allows the separation of the dynamic parts of web pages from the static HTML by adding JSP tags and scriptlets into normal HTML files to make a JSP file. Usually a JavaBean is used to provide the dynamic content specified by those special tags. The scriptlets let Java code be inserted directly into a JSP file. When this JSP is requested for the first time, the JSP engine will generate a dynamic servlet program based on the JSP file and the JavaBean program, and the servlet returns the result back to client. If the JSP file is modified to change the presentation of the result, the JSP engine will

regenerate the servlet program to reflect the changes. After the servlet is built, all requests can be sent to it directly.

Since the servlet ultimately does the processing, the means of communicating with JSPs are the same for the servlet.

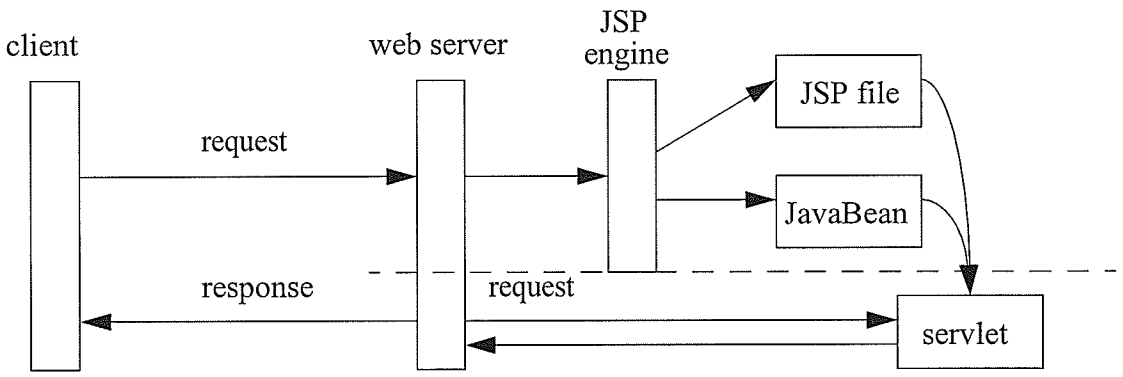

Figure 5.4 JSP Work Flow

CHAPTER 6 The Structure of AEA

6.1 The Structure of AEA

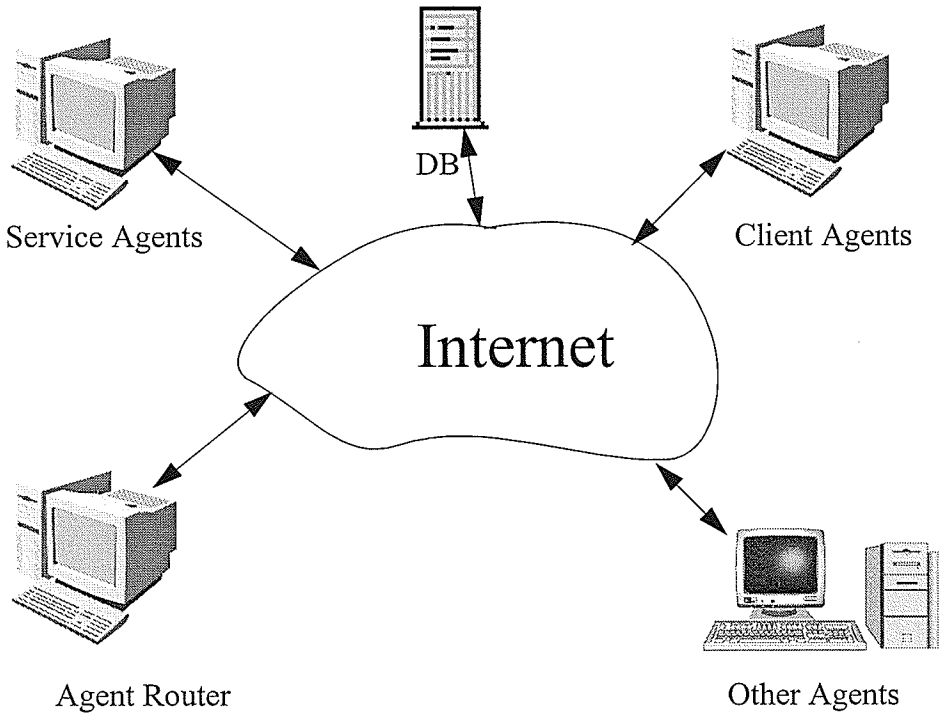

Figure 6.1 The Structure of AEA

Figure 6.1 is the basic structure of AEA. The AMR (Agent Message Router) in JATLite package runs on the router machine. It relays messages among agents. The database machine has an mSQL database engine running on it, which is the data storage for the system. The client agent is one type of agent of the system, providing users with a graphical user interface, while the service agent handles user's requests. Details of these agents will be discussed later in chapter 7. The cli-

ent agent machine is any machine the user loads the client agent on. There are several types of service agent, and they can run on different machines. Other agents are programs outside of the system, and are typically CGI programs. The direct connections among components in the diagram are between service agents and the router, client agents and the router, service agents and database machine, service agents and other agents. There is no limitation about which machine the database, and service agents reside on, but the router must be running on the same machine that all the client agents are loaded from.

6.2 Communication Between Agents

The client agent sends a user's request to the router which reroutes them to the proper service agents according to the types of the requests. After finishing executing the requests (this may require querying the database and external programs), the service agents send the results back to the router which reroutes them back to the original client agent. In some cases, one type of service agent may need information from another type of service agent. Their communication can follow the same approach by going through the router, but they also have the choice to deploy in a direct network connection, because they are free to build network connections among themselves. All the messages, including the requests and results, are coded in KQML.

Figure 6.2 illustrates how a client agent communicates with a service agent,

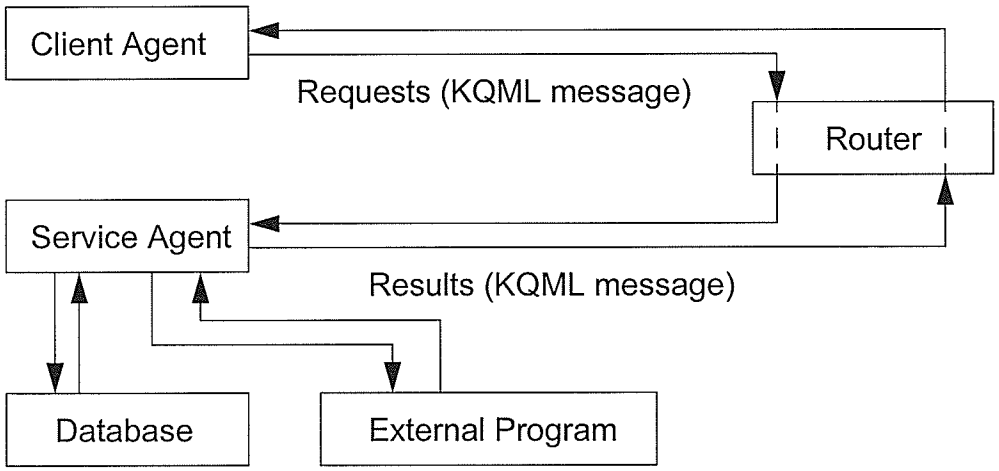

Figure 6.2 Work Flow Diagram for Client and Service Agent Communication

6.3 Agent Communication Protocol

In every client/server application, data transmission and interpretation are important parts. A protocol is usually designed for each application based on different purposes of the applications. A protocol is the language clients and servers speak to each other in order to perform certain tasks. For instance, a web server and a web client (browser) use HyperText Transfer Protocol (HTTP) to send information. The browser sends the HTTP request to the web server, requesting a certain web page or dynamic web content. The web server either returns a HTTP response with the requested web page in it or activates the CGI program or server side scripts to generate dynamic content and return it in the HTTP response. The web browser then displays the HTTP response.

The AEA system is built on top of the JATLite package which was introduced earlier. JATLite uses KQML messages among the agents, so the protocol used by the AEA system needs to be embedded into KQML messages. The KQML message contains address information which the router uses to send the message to the proper recipient. It also contains certain performative(s) specifying the type of action that the sender agent wants the recipient agent to do. However, usually the performatives are not specific enough for each application that is build over JATLite. New protocol is needed for each application and is transmitted within KQML messages. The following diagram (Figure 6.3) describes this approach.

[1] Graphical User Interface detects the user's action.

[2] Action is interpreted into a command from the protocol; the command is then embedded into a KQML message.

[3] The router client transmits the KQML message to the router.

[4] Based on the recipient name in the KQML message, the router redirects the message to a service agent.

[5] The service agent extracts the command from the KQML message and executes it according to the protocol.

[6] If a result needs to be sent, the service agent will compose it using commands in the protocol, and then embed it into a KQML message and transmit it to the router. The result has two possible type of recipients. One is the original client agent, and the other is another client agent(s). This depends on the purpose of the original message from the original client agent.

[7] The router sends the KQML message to client agent(s).

[8] The client agent extracts the command from the KQML message and interprets it with the protocol.

[9] The display on the client agent is updated according to the command.

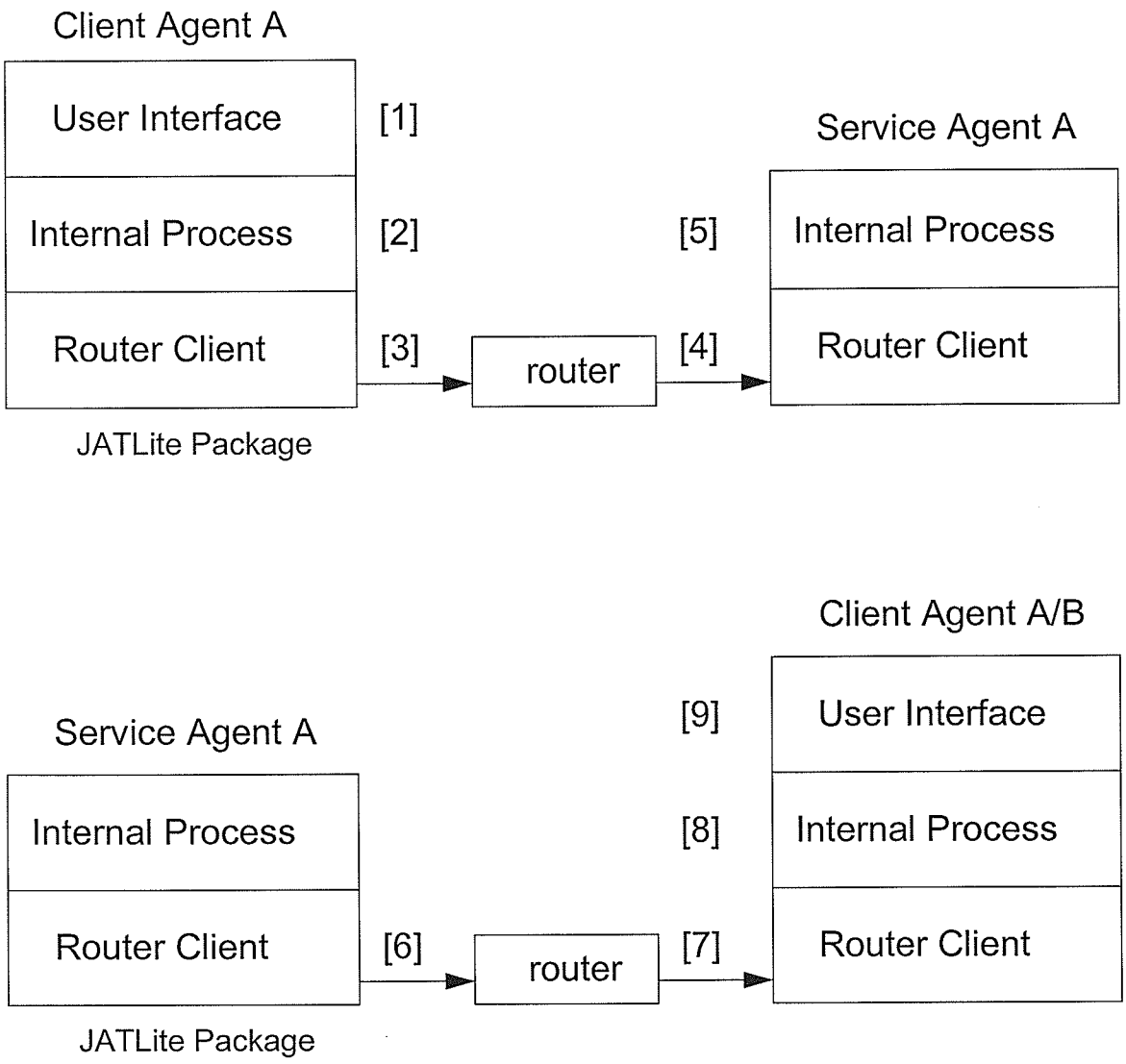

Figure 6.3 Data Transmission Sequence

The transmission mechanism of JATLite and KQML were discussed in previous chapters. The protocol used by client agents and service agents is named the Adaptive Education Agent

Protocol (AEAP). The AEAP is embedded into the content portion of the KQML string and is partly associated with the KQML performatives.

There are two types of commands in AEAP: commands used from service agents to client agents (type s-c) and commands used from client agent to service agent (type c-s). There are name conventions for those commands, &&command&& for type s-c, and ##command## for type c-s. The following table is a summary of the commands in the protocol,

Table 4: AEAP Command Type C - S

Command	KQML performative	Meaning
anynewstuff	ask-one	C asks S if there is any new message, submit, or mark
loadcourse		C asks S to load the course information from its KB (Knowledge Base)
loadmessage		C asks S to load messages from its KB
loadwebsite		C asked S to load the website information from its KB
querycourse		C asks S to start a query about information on a course; following the command is the course number
querymark		C asks S to load mark information from its KB
querystudents		C asks S to start a query about information on students that are taking a course; following the command is the course number
queryta		C asks S about the TA information on the certain course; following the command is the course number
addcourse	tell	C tells S to add a course to its KB; following the command is the course number
addweb		C tells S to add a website to its KB; following the command is the address of the website
markchecked		C tells S that new marks for C have been checked
messagechecked		C tells S that new messages for C have been checked
newuser		C tells S to verify the new user's information in order to add it into S's KB; following the command is the user information

Table 4: AEAP Command Type C - S

Command	KQML performative	Meaning
queryinfo		C tells S to start a query about a website's modification information; following the command is the index number of the website in its KB
removecourse		C tells S to remove a course from its KB; following the command is the course number
removedept		C tells S to remove all courses under one department from its KB; following the command is the department number
removeweb		C tells S to remove a website from its KB; following the command is the index number for the website in its KB
submitwork		C tells S to prepare for submitting its work
submitchecked		C tells S that new submissions for C have been checked
updateta		C tells S to update TA information for a course in its KB; following the command are the course number and the TA's name(s)

CHAPTER 7 Designing The Client Agent

The client agent provides users with an interface to use all the features AEA supports. Based on the number of roles in a traditional education system, the client agent currently consists of four categories of agent types: student agent, professor agent, TA (teaching assistant) agent and technician agent. Some administrative roles in an education system are not involved in the teaching and learning process, so no agents have been designed for them at this time. Besides the features common to all client agents, each category has its own unique features.

7.1 Java Applet

As described before, a client agent is a Java applet program. An applet is a program written in the Java programming language that can be included in an HTML page. When a user uses a Java technology-enabled browser to view a page that contains an applet, the applet's code is transferred to the user's system and executed by the browser's Java Virtual Machine (JVM).

To include an applet in a web page, the HTML file needs to include an <APPLET> tag. The complete syntax for the APPLET tag is as follows. Required elements are in **bold.** Optional elements are in regular typeface. Elements that the user specifies are in *italics.*

<APPLET
CODEBASE = *codebaseURL*
ARCHIVE = *archiveList*
CODE = *appletFile* or **OBJECT = *serializedApplet***
ALT = *alternateText*
NAME = *appletInstanceName*
WIDTH = *pixels* HEIGHT = *pixels*

```
    ALIGN = alignment
    VSPACE = pixels  HSPACE = pixels
>
<PARAM NAME = appletAttribute1 VALUE = value>
<PARAM NAME = appletAttribute2 VALUE = value>
. . .
 </APPLET>
```

CODE, CODEBASE, etc. are attributes of the applet tag; they give the browser information about the applet. The only mandatory attributes are CODE, WIDTH, and HEIGHT. Each attribute is described below.

CODEBASE specifies the base URL of the applet--the directory that contains the applet's code. The code can be either a single Java class or contained in a Java archive file (JAR). ARCHIVE describes one or more archives containing classes and other resources that will be loaded onto the user's machine. CODE gives the name of the file that contains the applet's compiled Applet subclass. OBJECT gives the name of the file that contains a serialized representation of an Applet. ALT specifies any text that should be displayed if the browser understands the APPLET tag but can't run Java technology-based applets. NAME specifies a name for the applet instance, which makes it possible for applets on the same page to find (and communicate with) each other. WIDTH and HEIGHT give the initial width and height (in pixels) of the applet display area, not counting any windows or dialogs that the applet brings up. ALIGN specifies the alignment of the applet. VSPACE and HSPACE specify the number of pixels above and below the applet (VSPACE) and on each side of the applet (HSPACE). <PARAM NAME = appletAttribute VALUE = value> group specifies parameters that need to be passed onto the applet class when the applet is loaded.

The main advantages of an applet are that it can be distributed easily to any machine on the Internet, it can be used to construct a sophisticated graphical user interface and it supports net-

work connections. These attributes are essential to the design and implementation of the AEA system. Applets also have security limitations, such as they cannot make network connections except to the host that they came from and this affects the system's structural flexibility. Components that communicate with applets need to stay on the same host machine. Also, applets cannot ordinarily read or write files or system properties on the host that they are executing on.

7.2 Designing Concepts

- **Authentication**

Like other end user applications in a client/server systems, the client agent uses an authentication system in which users have user ids and passwords to log in and out of the AEA system. Besides the user id and password combination, there is one more factor identifying the users' client agent type. Because there are four client agent types, a user must try to invoke the right type of client agent. Even with right user id and password, a student has no way to enter a professor agent.

To automate the procedures for new user's registration, and ensure that only registered university students can register in the AEA system, the AEA system needs to take advantage of the current authentication system of the university. Every university student has a student number and PIN number to access their student record, and they use this information to access the official university database. Based on the fact that there are programs running on the university's websites which use the authentication system to provide students with their records, these programs can be treated and used as external agents to the AEA system. By querying these external agents, the AEA system is able to guarantee the uniqueness and validity of every user that registers. After registering, each user can have their own user id and password for the AEA. For professor agent and technician agent, since there is a small number of people in these roles, their user name and

password can be picked and entered into the system directly and manually. For the TA agent, based on the fact that a TA is also a registered student, the same mechanism for a student to register to the AEA system is applied here. The only difference is a TA needs to be assigned by the professor of that course in order to add that course to his or her list.

- **Course Based Focus**

Since the four types of client agents are related through courses, and each course has a corresponding course number, the course numbers are what tie all types of agents together. Professors teach one or more courses; Students take courses; TA's mark assignments and assist students with labs for certain courses; Technicians prepare instruments and other necessities for the labs of certain courses. Most of the functions that client agents support are based on the course they are associated with. For example, every client agent has a message bulletin boards for every course that is on their list.

- **Automation**

The main goal of the AEA system is to integrate most of the daily actions that are essential to the teaching and learning process, and increase the efficiency and quality of education by decreasing the time and effort taken for those daily actions. To achieve that, automation must be applied to the functions that the client agent performs. For example, a student does not need to know the professor's contact information to contact him or her. When a student registers in a course, he or she will be able to contact the professor who is teaching that course without knowing the name or email address of the professor. Similarly, a student does not need to know who the TA is for that course. He or she should be able to submit a finished assignment or lab report and the TA for that course will receive them automatically.

7.3 Client Agent Classification

- **Student Agent**

The student agent is designed to meet the common needs of the students during the course of study, which may include communicating with the instructor, discussing questions with classmates and submitting assignments, etc.. The functions or features that a student agent has are follows,

- Message Tool. Students can use this tool to compose and send text messages to other agents involved in certain courses, e.g., professor, TA, or technician. Students can also retrieve, read and delete messages that other agents send to them.
- Bulletin Board. Students can use this tool to compose and post comments or questions about specific courses. They can also read and reply to comments or questions posted by other people. There is one bulletin board for every course.
- Submit Tool. Students can use this tool to submit files in many formats. These files could be assignments, lab reports or other documents, and will be sent to the assigned person who is responsible to mark them. In the current system setup, the TA is the assigned person responsible to mark the assignments.
- View-Mark Tool. Students can use the view-mark tool to check the marks of assignments or lab reports. They can also view the marker's comments on each of their assignments or reports.
- Web Monitor. Students can use this tool to monitor websites of their choice. The tool monitors the last modification dates of the websites so that users will be able to determine which ones have been updated and whether or not a visit to that website is necessary.

- **Professor Agent**

The term "Professor Agent" may not be accurate, because the instructor of the class does not need to be a professor to do the job. So the name here does not have a literate meaning, and it should be understood as "Instructor Agent". Like the student agent, the professor agent also has a set of functions that will be used to deal with the teaching job. Some of those functions are identical to those in the student agent and they will not be described in detail.

- Message Tool.
- Bulletin Board.
- Web Monitor.
- Student-Query Tool. A professor can use this tool to get the information about all the students that are taking the course, including their student numbers and phone numbers.
- TA-Assignment Tool. This tool is part of the authentication system for the TAs. Because there is no administrative agent in the AEA system, the professor agent is designed to be responsible for assigning TA(s) to the course he or she is teaching. That means a TA can add a course number to the course list in his or her agent after the professor of that course assigns him or her as a TA. Every student could register as a TA, but only those with courses on the course list have the capability to work as a TA.

- **TA Agent**

The special function that a TA agent has is the Mark Tool. All of the functions are listed below,

- Message Tool.
- Bulletin Board.

- Web Monitor.
- Mark Tool. TAs can use this tool to view students' submitted assignments and lab reports, make comments on them and give the final mark for each of them. The scores and comments then will be sent back to the corresponding student.

- **Technician Agent**

The current version of the AEA system does not specify any special function for technician agents. The functions that a technician agent has are a subset of student agent. They are listed below,

- Message Tool.
- Bulletin Board.
- Web Monitor.

7.4 Class Diagram

Figures 7.1(1) and 7.1(2) are the class diagrams of the client agent. There are too many classes to be drawn on one page and combination of those two figures dictates the logic and relationship among those classes that construct the client agent. Brief descriptions about each involved class are the following:

RouterClientAction is an abstract class in the RouterLayer that contains methods of sending and receiving messages. Any class that needs to have this capability must inherit from this class and implement its Act() method.

AgentRouterClientAction is the implementation of class *RouterClientAction*. It provides basic communication functions to all the other classes in the client agent.

CGIAccess is a class that provides methods to communicate with server side programs such as CGI programs and Java servlets. In this case, the class *JBoard* creates class *CGIAccess*'s instances and uses its method to talk to CGI programs.

ClassListMain is a subclass of *JFrame* class that constructs the main window for the client agent. It is composed of other function classes and is invoked by class *ClientAgent*.

ClassListSub is a subclass of *JInternalFrame* class that constructs a window containing the course list. It handles managing the course list and since all other functions are based on the course selection, it also provides other classes with information.

ClientAgent is a subclass of *JApplet* class which gets loaded by the web browser's JVM after a user logs in. It constructs the startup window of the agent and starts an object of the class *Class-ListMain*.

DynamicTree is a subclass of *JPanel* class that constructs the course list in a tree style. It handles the actual addition and removal of the tree nodes as class ClassListSub gives it order to update the course list.

DynamicTreeBoard is a subclass of *JPanel* class that constructs the tree view of the message list for the bulletin board application. It handles the addition and removal of the tree nodes as class *JBoard* passes the update information.

JBoard is a subclass of *JFrame* class that constructs the bulletin board window. It handles all the message for posting and viewing functions by using the *CGIAccess* class' methods to talk to a bulletin board CGI program running at the web server.

LogWindow is a subclass of *Applet* class that constructs the log in window of the client agent. It is the first class loaded by the web browser's JVM. It starts an instance of either the class Register or the class *ClassListMain* based on the type of user's login.

MessageWindow is a subclass of *JInternalFrame* class that constructs various types of message windows. It displays different interfaces based on the call it gets from the class ClassListSub, such as composing a message or viewing a message.

Register is a subclass of *JApplet* class that constructs the registration window for new users. It starts the *ClassListMain* class in the event that a new registration is granted and the user logs in.

StudentListWindow is a subclass of *JInternalFrame* class that constructs the window displaying student information which is a special function of a professor agent.

StudentMarkWindow is a subclass of *JInternalFrame* class that constructs the window managing the marks for students.

URLInfo is a class that dictates the properties of a URL, and provides methods for accessing and setting those properties. It is a supporting class for the class *WebMonitor*.

ViewSubmit is a subclass of *JInternalFrame* class that constructs the window allowing users to view all the submitted files, such as assignments and reports, and assign marks to them.

WebMonitor is a subclass of *JFrame* class that constructs the window displaying information about the monitored websites.

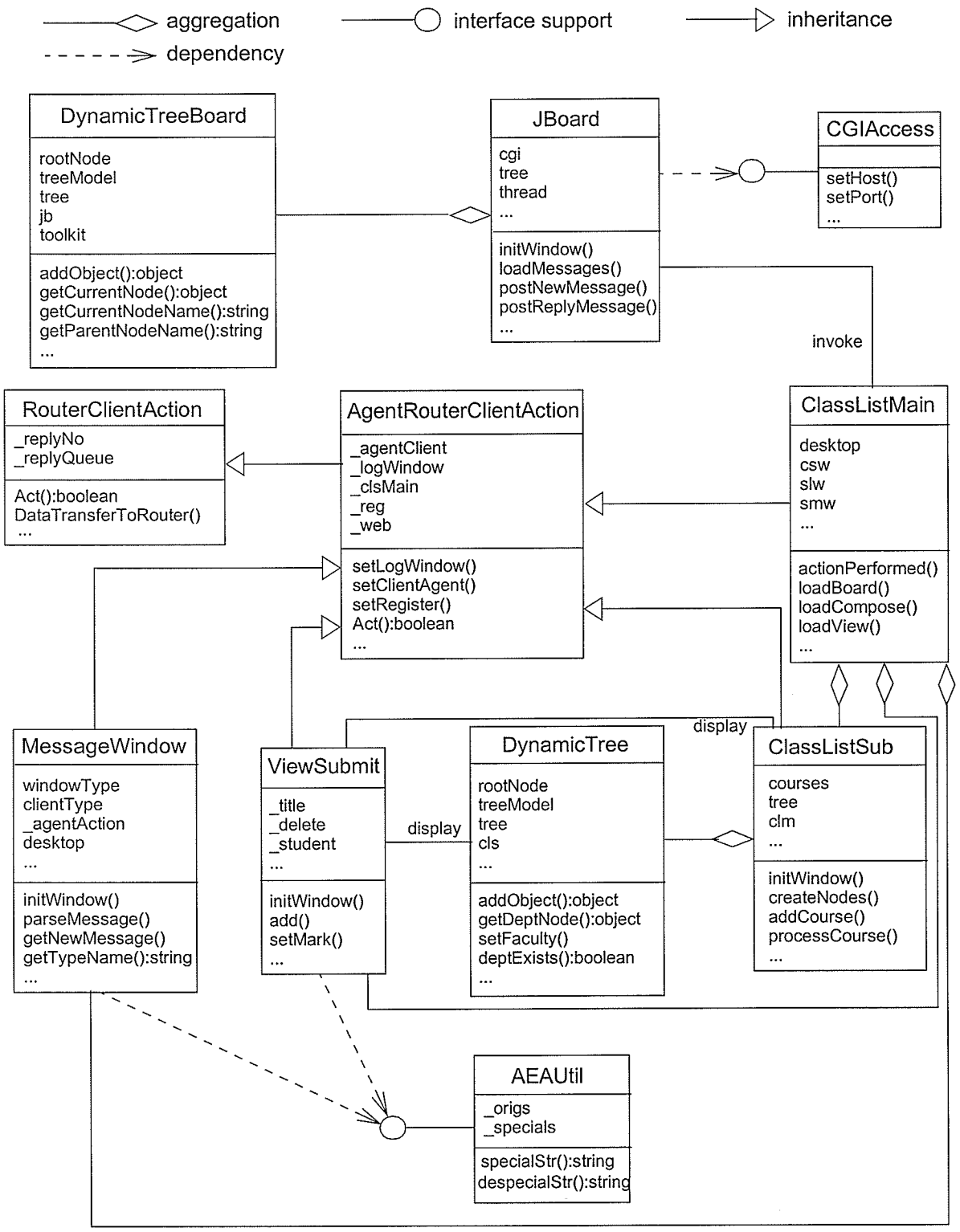

Figure 7.1(1) Client Agent Class Diagram

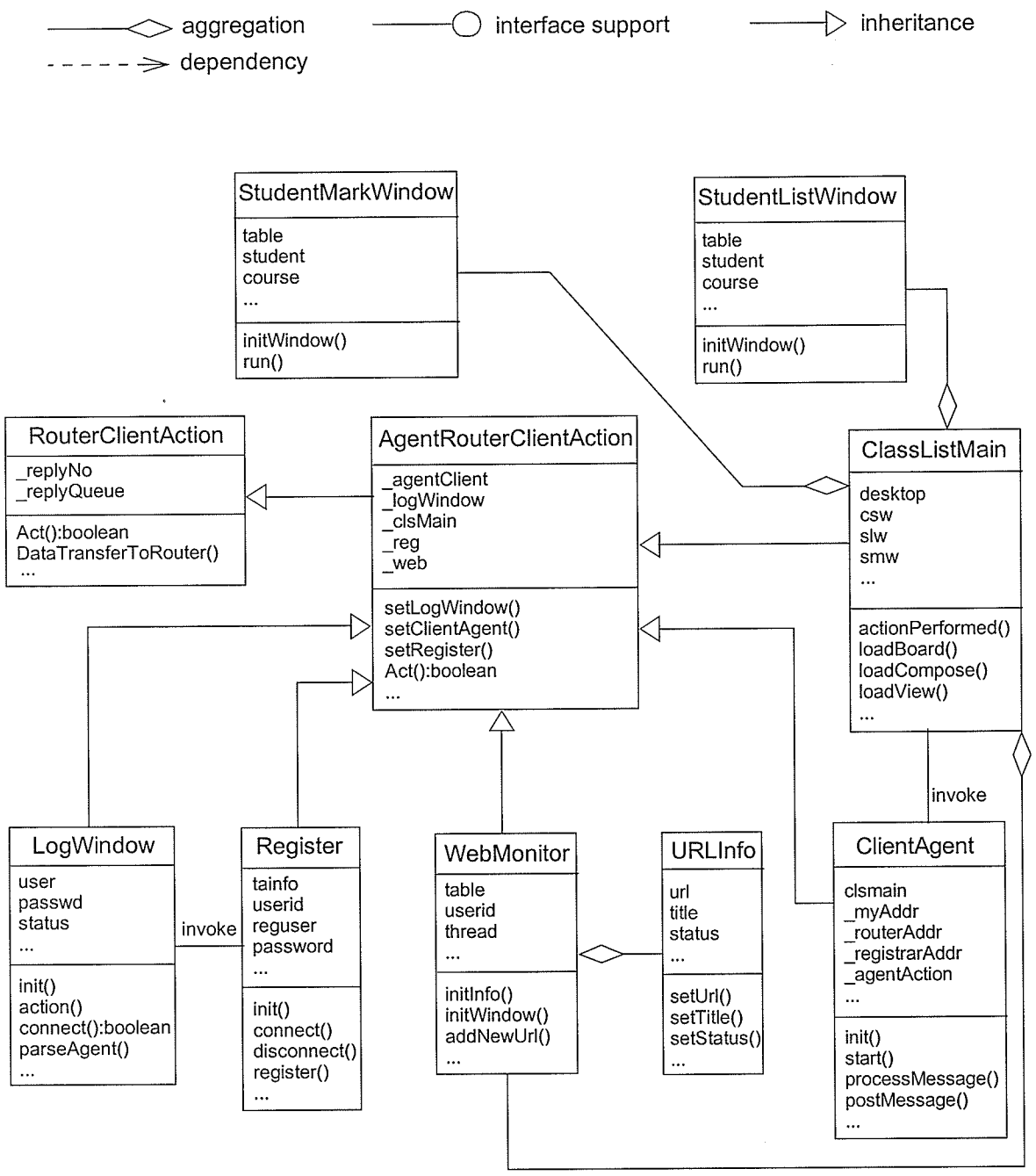

Figure 7.1(2) Client Agent Class Diagram

7.5 Setup Client Agent

Users access the client agents through their web browsers, and as described before, an <APPLET> tag in the HTML file specifies the applet's attributes from the directory where the applet is to parameters that initiate the applet. The tag tells the web browser to create an instance of the Java class in the Java Runtime Environment (JRE) built in the web browser. The current web browsers' JRE supports Java class written for JDK 1.1. The client agent classes use Java's advanced graphical package called Swing instead of the conventional AWT package. The Swing package is not included in JDK1.1 and comes as a patch package, thus the JRE in some web browsers can't execute classes that use Swing. As such, here are two options to address this problem. One is including the Swing package (a JAR file) as one of archives that the <APPLET> tag specifies. When a user tries to start the client agent, the user's browser will load the JAR file which contains all the Swing classes that the client agent's classes use, thus the client agent can run normally. Another option is using a Java Plug-in. A Java Plug-in is an extension of JRE that can be downloaded from Sun's website and installed on the user's browser. There are several versions of Java Plug-ins supporting Java applets written for different version of JDK. The latest Java Plug-in will support all versions. After the Plug-in is installed, the web browser loads the new JRE instead of the old one every time there is an applet embedded in an HTML page, and because the Swing is part of JDK since version 1.2 and is supported by new JRE, the client agent can find the necessary Swing classes and run normally.

There is an HTML file named "logon.html" on the server host. Users will be given a link to this file as the location of the client agent. As option number one, the Swing package is distributed to the user's machine along with the applet classes. The following code block is the portion in the HTML file that defines the applet - the client agent. The LogWindow.class is the initial class to be

loaded and displays the login window. There are three JAR files used. The Java Archive (JAR) file format enables the developer to bundle multiple files into a single archive file. Typically a JAR file will contain the class files and auxiliary resources associated with applets and applications. The jat.jar file is the JATLite package containing the classes for the transmission level. Class *IPRouterClientAction* in the class diagram and its supporting classes are contained in that JAR file. The agent.jar file contains all the classes of the client agent and resources such as images used in the applet. The swing.jar file contains the Swing package used by the client agent.

```
<APPLET CODEBASE = ".." CODE = "LogWindow.class" ARCHIVE =
"jat.jar,agent.jar,swing"
WIDTH = 300 HEIGHT = 250>
</APPLET>
```

For option number two using a Java Plug-in, the HTML file needs to be modified to specify the use of Sun's JRE via the Java Plug-in Software. There is a specification written by Sun on how to make the necessary changes, and also there is a free Java Plug-in Software HTML Converter that will automatically make the changes to the HTML. The following code is the converted result obtained by using the HTML converter on the original HTML. It will instruct the user to download and install the Java Plug-in version 1.4 in order to run the embedded applet. As you can notice, the archive list does not include the swing.jar any more.

```
...
<!-- HTML CONVERTER -->
<OBJECT
  classid="clsid:CAFEEFAC-0014-0000-0000-ABCDEFFEDCBA"
  WIDTH = 300 HEIGHT = 250
        codebase="http://java.sun.com/products/plugin/autodl/jinstall-1_4_0-win.cab#Ver-
sion=1,4,0,0">
  <PARAM NAME = CODE VALUE = LogWindow.class >
  <PARAM NAME = CODEBASE VALUE = "..">
  <PARAM NAME = ARCHIVE VALUE = "jat.jar, agent.jar">
```

```
    <PARAM NAME="type" VALUE="application/x-java-applet;jpi-version=1.4">
    <PARAM NAME="scriptable" VALUE="false">
  <COMMENT>
  <EMBED
        type="application/x-java-applet;jpi-version=1.4"
        CODE = LogWindow.class
  CODEBASE = ".."
  ARCHIVE = "jat.jar,agent.jar"
        WIDTH = 300
        HEIGHT = 250
        scriptable=false
        pluginspage="http://java.sun.com/products/plugin/index.html#download">
      <NOEMBED>
  alt="Your browser understands the &lt;APPLET&gt; tag but isn't running the applet, for some
reason."
  Your browser is completely ignoring the &lt;APPLET&gt; tag!
  </NOEMBED>
  </EMBED>
  </COMMENT>
  </OBJECT>

  <!--
  <APPLET CODEBASE=".." CODE = LogWindow.class ARCHIVE="jat.jar,agent.jar"
WIDTH = 300 HEIGHT = 250>
  alt="Your browser understands the &lt;APPLET&gt; tag but isn't running the applet, for some
reason."
  Your browser is completely ignoring the &lt;APPLET&gt; tag!
  </APPLET>
  -->

  <!--"END_CONVERTED_APPLET"-->
  ...
```

CHAPTER 8 Designing The Service Agent

Service agents are stand-alone Java applications. They all provide client agents with queried results in certain aspects and if necessary, save users' information into a database. By the types of services they provide, the service agents are classified into five types: Central Agent, Web Monitor Agent, Message Handle Agent, CGI Query Agent, and Submit Agent. The service agents are registered with the router using their service names and passwords prior to any actual communication and the router maintains a registry containing the name password pairs. As long as the service agents know where the router is located, they can log on and build connections to the router and the router will forward all the queries from client agents to the right service agents.

The service agents depend on their knowledge bases to answer client agents' queries. The knowledge base could be filled out with information in many ways, such as entering them manually and updating information from the later interactions with other agents. In the AEA system, all the service agents' knowledge bases are initially empty, they need to obtain information from other sources in order to answer the queries from client agents. Sources include useful programs accessible from the Internet, such as CGI programs and Java Servlets. There are also other reasons for service agents in the AEA system to communicate with other existing programs,

- Some information is already out there. It is easier to just "talk" to programs which have access to that information than to build the same information. For example, if a student wants information about a course which is already saved in a certain database, accessible by some ASP programs on the University of Manitoba website, the service agent can just ask that ASP program about the course information the student wants, and send the infor-

mation back to the student. Meanwhile, service agents save that information to its knowledge base for use in the same query next time.

- Some information is confidential. Service agents need external programs that can access such confidential information source to process some client agent queries. For example, when a student wants to register in the AEA, the system needs to validate if he or she is currently registered at the University of Manitoba, otherwise, the system won't let the student register. The student number and PIN number are needed to do such a validation check, and by asking the student to enter the information and forwarding them to the program running on the University of Manitoba website to validate them, the service agent can achieve a high level of security without keeping any of students' confidential records in its knowledge base.

8.1 Class Diagram

Figure 8.1 shows the class diagram for service agents. The following is a brief description of the figure.

RouterClientAction is an abstract class in the RouterLayer that contains methods for sending and receiving messages. Any class that needs to have this capability must inherit from this class and implement its Act() method.

CentralAgent mainly handles administrative functions and manages users' profiles.

CGIAccess is a class that provides methods to communicate with server side programs such as CGI programs and Java servlets. Several service agents use its support to query programs outside the system.

MessageHandleAgent is a class that controls the system's message flow. Its major task is to redirect messages to appropriate receivers based on the course the messages are related to.

QueryCourse is a class that gets course information from external programs.

QueryStudents is a class that gets information about students who register in a course from external progams.

RandomGenerator is a class that generates random multi-digit characters. It has control over the types of character to use and number of digits to generate. It is used by the *CentalAgent* class to generate validation codes for the client agent's submit function.

SingleQuery is a class that represents one query instance. It is composed of *QueryCourse*, *QueryStudents* and *ValidationQuery* classes. Class *CGIQueryAgent* can invoke multiple instances of this class to handle multiple requests.

CGIQueryAgent is a class that processes requests involving external programs.

SubmitAgent is a class that manages the work of the submitting and marking functions.

URLInfo is a class that dictates the properties of a URL, and provides methods accessing and setting those properties.

ValidationQuery is a class that validates students' and professor's information through external programs.

WebMonitorAgent is a class that processes requests regarding monitoring websites.

WebAgent is a class that does the actual monitoring of all requested websites.

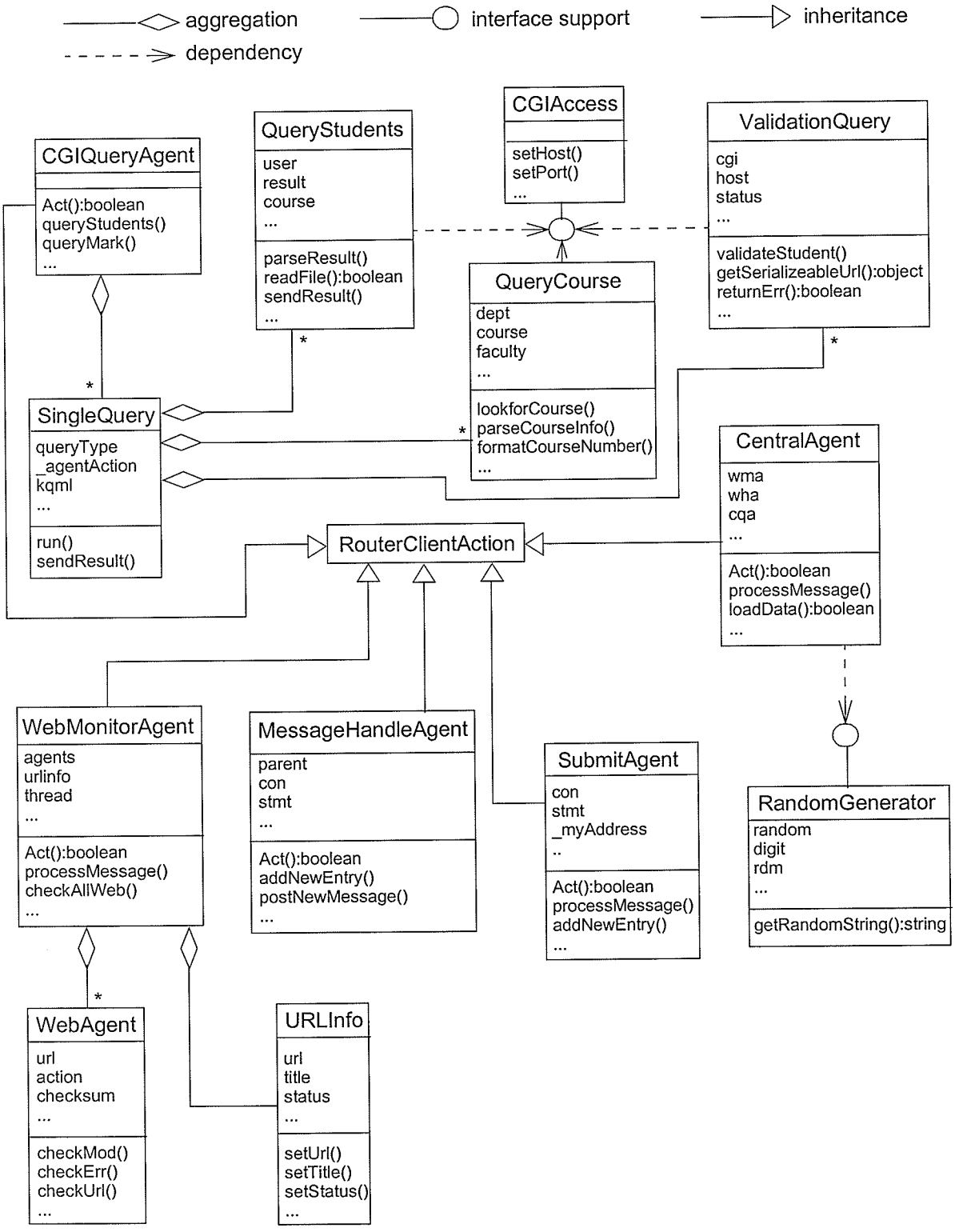

Figure 8.1 Service Agents Class Diagram

8.2 The Central Agent

The Central Agent (CA) is the core service agent. It handles most of the requests sent by client agents by either forwarding them to other service agents or forking proper procedures to process the requests.

8.2.1 Authentication Function

One important function for the Central Agent (CA) is the authentication process. It is the first process the CA will perform for client agents. As with any other user specific system, authentication is crucial to both the system's security and user identification.

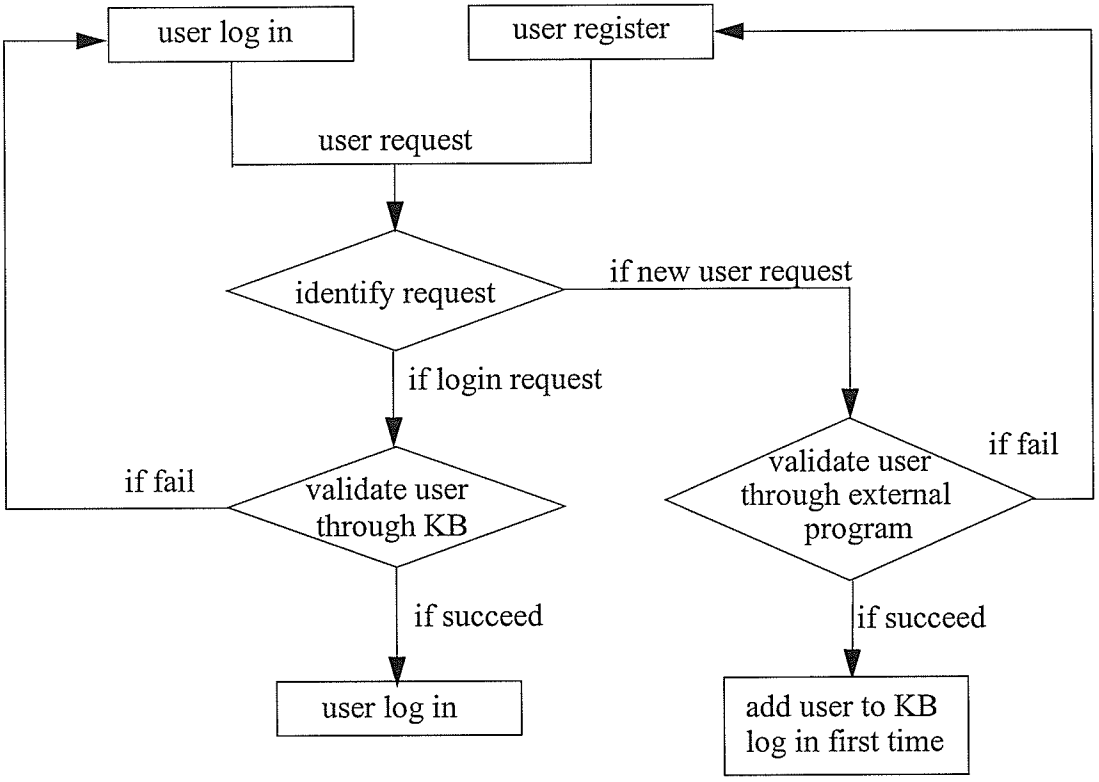

Figure 8.2 Authentication Procedure

The diagram shown in Figure 8.2 illustrates the flow of the authentication process. Basically it includes two type of processes, one for user login requests and the other for user registration requests. The details of those processes are listed below.

1. If a user logs in with user name "newuser" and password "newuser", the registration window will prompt and ask for identification details in order for the CA to process the registration. These details include the user's student number and PIN number which are used to register at the University of Manitoba. If a user logs in with a user name other than "newuser", the user name and password are sent to the CA for validation.
2. For a new registration request, the CA forks a process which will send a query containing the user's identification to an external program. In this case, the external program is an ASP program from the University of Manitoba website handling students requests for their academic histories.
3. The external program processes the request by validating the information with the official database and returns the results back to the CA. If the student number and PIN number combination doesn't pass the validation, one of two reasons will be given to the CA; either the student number does not exist or the PIN number is wrong. Otherwise, the student's academic information will be returned. In these cases, all the information is transmitted in an HTTP formatted data stream from the external program to the CA, and the CA parses the data stream and determines the content of the response by looking for certain keywords.
4. If the student passes the validation, the CA will ask him to enter a user name and a password in order to create a new account in the AEA system. After the CA makes sure the student's inputs are valid, i.e., the user name is unique and the password is between 4 and 8 characters long, it will create a new account and send an approval message to the client agent which will

log the student into the AEA system for the first time with the new user name. If the student failed the validation, the CA will send a failure message to the client agent which will go back to the registration window and wait for new input.

5. For a user's log on request, the CA will fork a process to check its own knowledge base (KB) which in this case is an mSQL database. The user provides a user id and a password and the CA validates them. If the user's information is right, the CA will send the client agent an approval message and the user will be logged on to the AEA system. If the user's information is wrong, the CA will return one of two error messages to the client agent, either the user does not exist or the password is wrong, and the client agent will return to the login window again for the user to correct his information.

8.2.2 Profile Function

Each user's profile includes mainly their course list and associated web URL list. The course list is a list of courses under a specific user's account. For students, it is a list of courses they are taking or took. For instructors, it is a list of courses they are teaching. For technicians, it is a list of courses they need to maintain the labs for. For TA's, it is a list of courses they are teaching assistants for. Almost all of the communication among client agents and between client agents and service agents are based on the course list. A user's URL list contains a list of website addresses that the user wants to monitor. Every time a user logs on his or her client agent, the CA will get the profile for this user from its knowledge base and send it back to the client agent. This is the first step before all other profile related actions occur and can be called profile initialization. After the client agent obtains the initial profile for the user and displays it to the user, it can receive further action taken by the user and send corresponding requests to server agents.

There are four types of client agents, and there are, correspondingly four different ways to update the course list. Figure 8.3 illustrates the general procedures the CA will go through to add a course, and they are different based on the client agent type. Following the diagram are the details of those procedures.

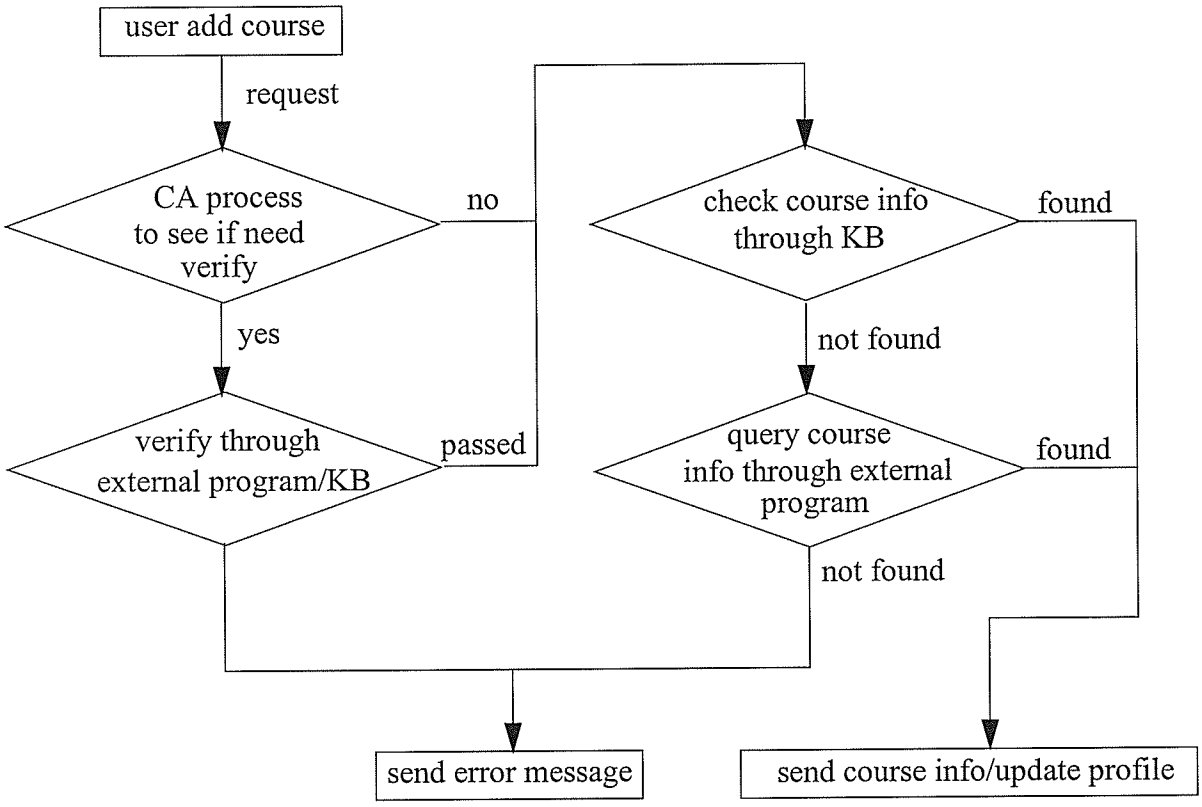

Figure 8.3 Adding A Course

For a student agent, courses that the agent can have under its course list are every course being held in the university. Whether the student is taking that specific course or not is not under the system's consideration. Based on the student agent's functions described earlier, there is no such function that will affect a course in a harmful way. Also, a student should be able to get informa-

tion and give feedback as to when he or she is auditing a course, so the CA will not validate the real registration when a student agent submits a request to add a course. The following describes the procedures to add a course to a student agent.

1. A student asks to add a course to the course list. The student agent sends a request with a course number the student provides to the CA.
2. The CA will first check its knowledge base to see if that course's information is already there. If the check is positive, the CA will send back the information and update the profile for that student in its knowledge base. If the check is negative, the CA will fork a process to send a request to an external program which will validate the course. In this case, the external program is a ASP program on the University of Manitoba website providing course information including the course name, number, department and faculty name, and the time slots it is held in.
3. The external program performs its search and sends back the results to the CA. If it finds a match to the course number, it will send all the related information to the CA and the CA will parse the results and record the necessary information into its knowledge base. The CA will also update the student's profile and send a success response along with the course information to the student agent. If the external program does not find a match to the course number, it will send a response to the CA indicating that, and the CA will send an error response to the student agent.
4. If the student agent receives an error response from the CA, it will tell the student that no such course exists under the given course number by raising a warning window. If the student agent receives a success response with course information, it will add the new course to the course list categorized by faculties and departments.

For professor agents, adding a course means the user (i.e. professor) is teaching the course. The CA must check with an authorized source to make sure the professor is in fact teaching the course. This is one more step different than the procedures with the student agent. The following describes the procedures for the CA to add a course to a professor agent.

1. A professor makes a request to add a course to the course list. The professor agent sends a request with the course number and professor's employee number to the CA. The employee number is also saved as part of the professor's profile and the purpose is to provide information for necessary validations.
2. Before the CA searches for the course information, it forks a process to send a request to an external program along with the course number and the employee number. The external program is a ASP program providing instructors with information about courses they are teaching including a complete registered student list with contact information. If the course number and employee number combination passes the check point, the program will send back to the CA the student list, or otherwise an error message indicating that the professor is not teaching the course.
3. If the professor is eligible to add the course (i.e., he or she is teaching the course), the CA will go through the same procedures described previously for the student agent in order to add the course. If not, the CA will send an error response to the professor agent.
4. Upon receiving the response from the CA, the professor agent will either add the course or display an error message the same way that a student agent does.

For a TA agent, the courses on the course list are that the TA is assisting. The professor teaching one course will assign the TA or TAs for the course using his or her professor agent. A TA can only add a course to his or her TA agent after the professor assigns his or her user id as the TA of

the course. This is the basic authentication rule for a TA to add a course. The details of how a TA is assigned will be discussed later in this chapter as one of the service agent's functions. The following describes the procedures for the CA to add a course to a TA agent.

1. A TA asks to add a course to the course list. The TA agent sends a request with the course number and the TA's user id to the CA.
2. The CA checks its knowledge base to see if the TA's user id was assigned as a TA of the course indicated by the course number.
3. If the check is positive, the CA will simply send a response with the course information to the TA agent and update the TA's profile in its knowledge base. If the check is negative, the CA will send an error response to the TA agent.
4. The TA agent will analyze the response and either add the course to the list or display an error message.

For a technician agent, there is no checking rule when a technician wants to add courses on to the technician agent. The CA will take the same procedures as it does for a student agent to add the course and will update the technician's profile.

The CA also manages the user's URL list as part of the profile. Figure 8.4 shows the control flow of the procedures. These procedures are the same for all types of client agents. When a user wants to add a website URL to his or her watch list, the client agent sends a request to the CA containing the URL list. Upon receiving the request, the CA will first check the user's profile to see if the URL is already listed there. If so, the CA will simply send a message back to the client agent indicating that, otherwise, it will forward this request to the Web Agent (WA). The details of how the WA processes the request and obtains information about the URL will be described later in this chapter. After the CA gets the result back from the WA, it will examine the result to

see if the URL has been resolved. If the result is positive, meaning that the website that the URL pointing to is reachable and the WA has successfully got the site information, the CA will record the URL into the user's profile and send the URL's information to the client agent. If the result is negative, meaning the URL is not valid or the website is not accessible, the CA will send an error message back to the client agent.

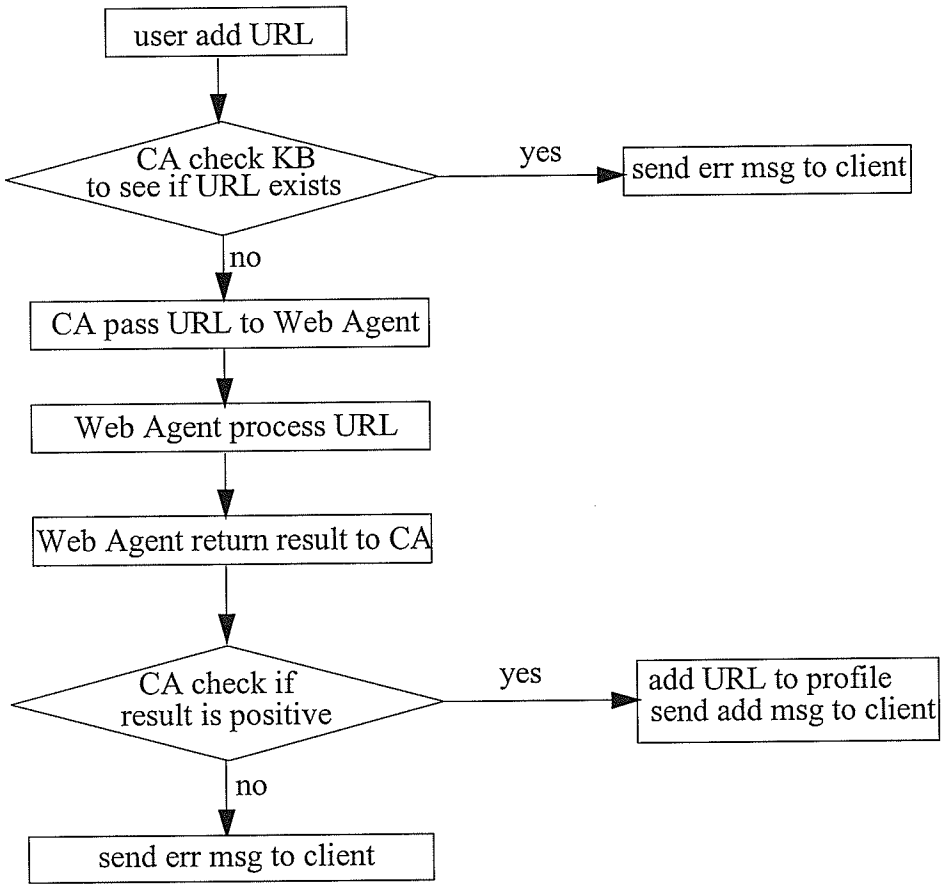

Figure 8.4 Adding URL Procedure

As part of the profile function, the CA also handles requests for deleting courses or web URLs from a user's profile. What the CA does is simply removes the deleted items from the user's profile in its knowledge base. All the information about the courses stay and is ready to be sent to the

client agent if it request it later. The URLs information is not saved because every time a URL is requested, all the information about it must be obtained in real time so that it reflects the URL's present status.

8.2.3 Other Functions

There are also other functions that the CA performs to operate the system. They include:

- Assigning a TA. When a professor agent wants to assign a TA(s) to the course(s) he or she is teaching, the request goes to the CA along with the course number and the TA's user id for the AEA system. The CA validates all the information including the correctness of the TA's id, and updates its knowledge base with the information by setting the TA for the course. If there was a TA assigned to the course before, that information will be overwritten by the new one. If there are more than one TA for a course, the professor will send all of their user ids separated by commas. The CA then records multiple TAs for that course.
- Submitting work. As mentioned before, the student agent has the ability to submit the student's work, such as assignments, to the appropriate agents. When a student tries to use this function, the system generates a temporary web page for the user to select the desired file from the local file system and the work type, and then submits it. The CA is responsible for generating this page. First, it creates a random 20 digit code and saves it with the student's id into its knowledge base. It then generates an HTML file named with the code and sends the URL of the file back to the student agent. After the student submits his or her work, the student agent sends the CA a notification message and the CA removes the temporary HTML file from its file system. The random number is also a security check point for use when another service agent handles the student's actual submitting, and will be explained in a later sections.

8.3 The CGI Query Agent

The CGI Query Agent (CQA) is responsible for dealing with all the external programs that are used in the system. As mentioned before, the Central Agent must get certain information from external programs and it actually does this through the CQA. Therefore, the main function that the CQA has is to provide the CA with services that involve external programs. It includes obtaining the course information for the CA when a client agent asks to add a course. It also includes validating the student's and professor's identification when they register with the AEA system, and the professor's identification when he or she adds a course to the client agent. The CQA usually forks multiple threads for these tasks, because the processing time relies heavily on the external programs and the network connection. Thus the CQA is open for new requests while other threads wait for their results.

The CQA also serves the client agents directly. The client agent sends requests to the CQA and the CQA does its query with external programs and returns the results back to the client agent. Usually the result in this situation is not required to be recorded into the knowledge base. One such function is to get student lists for professor agents. A professor agent has the ability to list all the registered students' information including their names, student numbers and contact phone number. The professor agent sends this request directly to the CQA and the result is not saved anywhere.

8.4 The Message Handle Agent

One major function the AEA system is to provide client agents with is the messaging system. It enables users to send and receive instant messages to and from people related to any particular course. Users don't need to know who those people are or what their user names are, they only

need to specify which type of people they want to send the message to. Handling this messaging function is the Message Handle Agent (MHA). The MHA receives messages associated with course numbers. The final recipients of those messages are specified in four categories which are the same as the four client agent types. The MHA looks up its knowledge base and find the corresponding users' user names based on the course number and the types of receivers, and then relays those messages to each of the users.

8.5 The Submit Agent

The Submit Agent (SA) handles requests regarding the submitting of work and the marking processes. The SA has a supporting Java servlet running as the work horse. The submitted files are sent from each student agent's machine directly to the servlet. Figure 5.26 illustrates the work flow of submitting a file and marking a file.

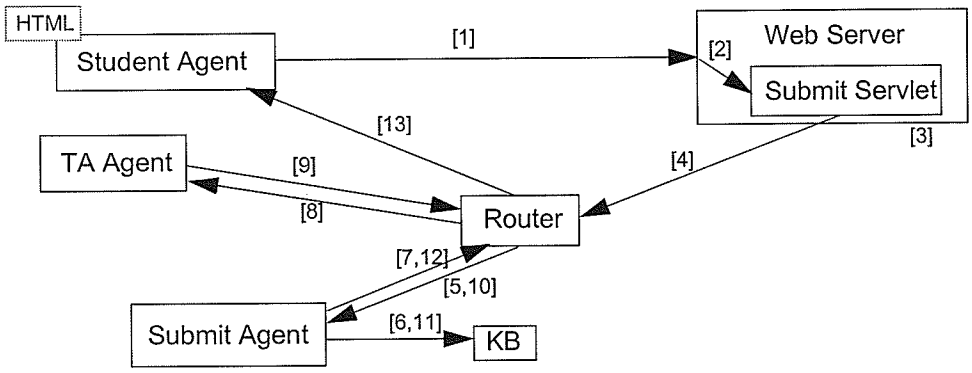

Figure 5.26 Submit Agent Work Flow

[1] The Central Agent creates a temporary HTML file for student agents to submit work. The HTML file specifies the servlet's path as the recipient of the inputs.

[2] After the web server gets the HTTP request, it forwards it to the waiting submit servlet.

[3] The submit servlet receives all the data as an HTTP data stream. It extracts the file name, user name and the file itself, then renames the file under a predefined naming convention and saves it to a directory in the local file system.

[4,5] The submit servlet composes a KQML message containing information about the new file name and sends the user name to the SA through the router.

[6] The SA updates the knowledge base with information coded in the message.

[7,8] The SA sends a reminder message through the router to the TA agent responsible for marking the submitted file.

[9,10] When a TA marks a file, the TA agent sends a message containing the mark, comments, and the student's user name to the SA through the router.

[11] The SA updates the knowledge base with the information.

[12,13] The SA sends a reminder message through the router to the student agent who submitted the file.

8.6 The Web Monitor Agent

The Web Monitor Agent (WMA) handles requests for adding new websites from client agents. The WMA maintains a list of URLs that it is currently monitoring. The list includes every distinctive website that users have requested to monitor in the past. The WMA regularly starts queries to get the latest status of every URL on the list and saves the information into an array of URL objects. Whenever there is a request coming in for a certain website's status, the WMA will pull that information from the object array and send it back in a KQML message. There is a WebAgent class that does the actual querying to each website. It can be forked into many threads and run individually. The following explains how it queries each website for status information,

[1] The web agent sends a message through a socket that is connected to the remote web server requesting the header information for one specific web page that it serves.

[2] If the remote web server returns the header information and provides the "Last-Modification" field in it, the web agent will parse the results and look for the value for the "Last-Modification" field.

[3] If web agent can find the last modification date, it will pass it to the WMA which in turn records it into its URL object array. The web agent then goes to sleep for a certain amount of time before it wakes up and does the next round of queries. The last modification time will be used by the web monitor tool in client agents.

[4] If the remote web server does not provide "Last-Modification" field within the header information, the web agent needs to examine the checksum of the HTML document itself. If the checksum is identical to the previous one, it then marks the web page unchanged, otherwise, it marks the examining time as the last modification time and pass it to the WMA.

8.7 Setup of Service Agents

The procedures to setup service agents are as follows,

[1] Start the JATLite router on machine *A* which is the same machine the web server is running on. The web server serves the HTML file containing the client agent applet. To start the router, in a console on machine *A*, go to the JATLite directory, and type

```
> java ProtocolLayer.IPRouterAction RouterLayer/Resource/routerscript &
```

This starts the router and it waits for messages.

[2] Start the mSQL database engine on machine *B*. The database's location is pre-defined in order for service agents to query it. To start the mSQL engine, in a console on machine *B*, type,

```
> msql2d &
```

This starts the database and all the tables are loaded.

[3] Start service agents by executing their classes. For example, to start the Central Agent, type the following in a console,

```
> java CentralAgent
```

Those service agents can run on any machines available on the network as long as they have access to the router machine and the database machine.

[4] Put the Submit servlet on machine *A*. It is supported and started by JSWDK (JavaServer Web Development Kit) 1.0.1.

CHAPTER 9 The AEA GUI

The previous chapters covered the design of the system. This chapter will demonstrate how the system works by showing different use case scenarios. Also, it will go through the main functions featured in the client agent and their corresponding interfaces.

- **Login, registration windows**

This window is a user's entry point to the client agent. As described before, there are four types of client agents that a user can log into: student agent, professor agent, TA agent and technician agent. Correspondingly, there are four image buttons a user can click on to login with their user name and password. If the user is new, he or she can use the user name and password combination of "newuser" and "newuser" to enter the registration session. Any of the four image buttons can be used for registration. The specific agent type can be decided in a later process.

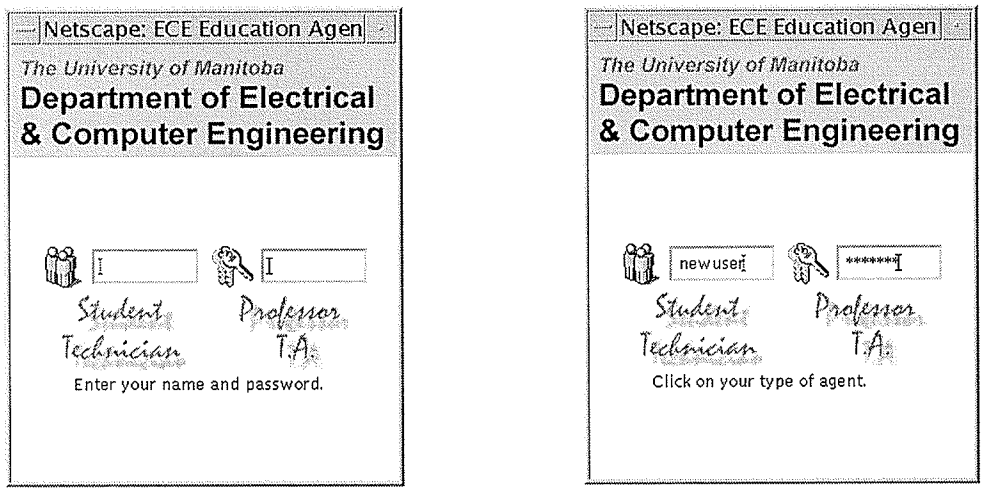

Figure 9.1 Login Window

Figure 9.2 shows the registration window. In the case of a new registration, after the user enters into the register session, identification information will be asked in order for the server to validate him or her to see if the user is eligible to register onto the AEA system. The information for students is the combination of their student number and PIN number with which they could use to register in a regular university sessions. This method applies to TAs too, because being a registered student usually is the first requirement to apply for a TA. The information needed for a professor or technician to register with AEA is a little bit different. Since they don't have the same identification (student number and PIN number) as students do, and their population is far smaller than the students, the registration information could be put into the system prior to their registrations by an administrator. Another piece of information is the user name which is used to log onto the AEA system. After the user is identified successfully and the user name provided is unique from existing users in AEA system, he or she will be led to set up the password. Otherwise, the user will be led back to the previous window to enter the identification information again.

After entering and confirming the password, the user submits the information and will log into the AEA system for the first time and enter the main window.

In the case of an existing user login, the client agent will send his or her user name and password to the service agent for validation. If they pass, the user will enter the main window.

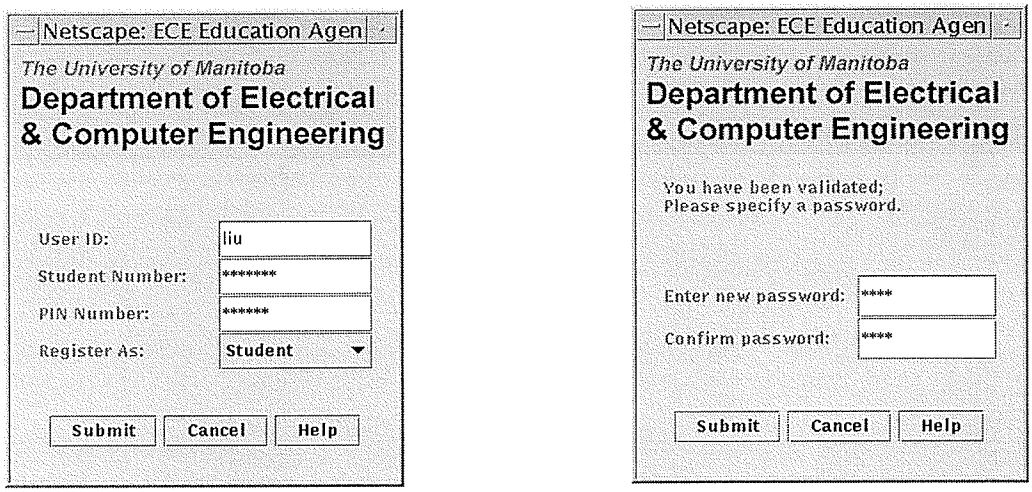

Figure 9.2 Registration Window

- **The Main window**

The main window (shown in Figure 9.3) is the front entrance for all the features that the client agent provides. It contains three image buttons: the "Course List" button, the "View Message" button and the "Help" button. As described before, most of the features that a client agent has are based on the course list (a list of courses that a student can enroll in or a professor teaches, etc.). The "Course List" button loads up the "Course List" window which contains all the tools providing available functions. The "View Message" button loads up the function "view message" in the message tool. This provides users with a convenient and speedy way to check the messages in his or her agent in one step. The "Help" button loads up a window containing concise information on how to begin using the client agent. The blank area in the middle is an information board for major functions that a client agent provides. For instance, it will display a message after the user logs on if there are new messages or new assignments for this user.

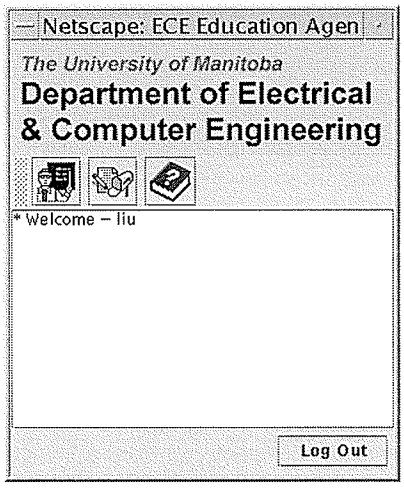

Figure 9.3 Main Window

- **The Course List Window**

The Course List Window (shown in Figure 9.4) contains all the functions that a client agent provides. The following sections will cover all the functions and their corresponding interfaces. The first thing a new user needs to do is to add course(es) to their course list. Without a course list containing courses, the client agent does not have much to do. There are some differences on guidelines about adding courses among different types of client agents. Student agents can add any course he or she wants to, and it doesn't need to be officially registered by the user. This is based on the assumption that a student doesn't want to add courses they are not taking and that student agents will not be able to affect other agents significantly. To a professor agent, adding a course means the user is teaching that course and has access to important information about students taking the course, thus the system needs verification to see if the user is designated to teach the course before that course can be added onto the user's course list. As described earlier, the professor agent has the responsibility to assign a TA for the course he or she is teaching. Only

assigned TAs will be able to add the corresponding courses onto their TA agent. A technician usually deals with issues for many courses at once, and the information about which courses a technician can add is pre-authorized by the administrator.

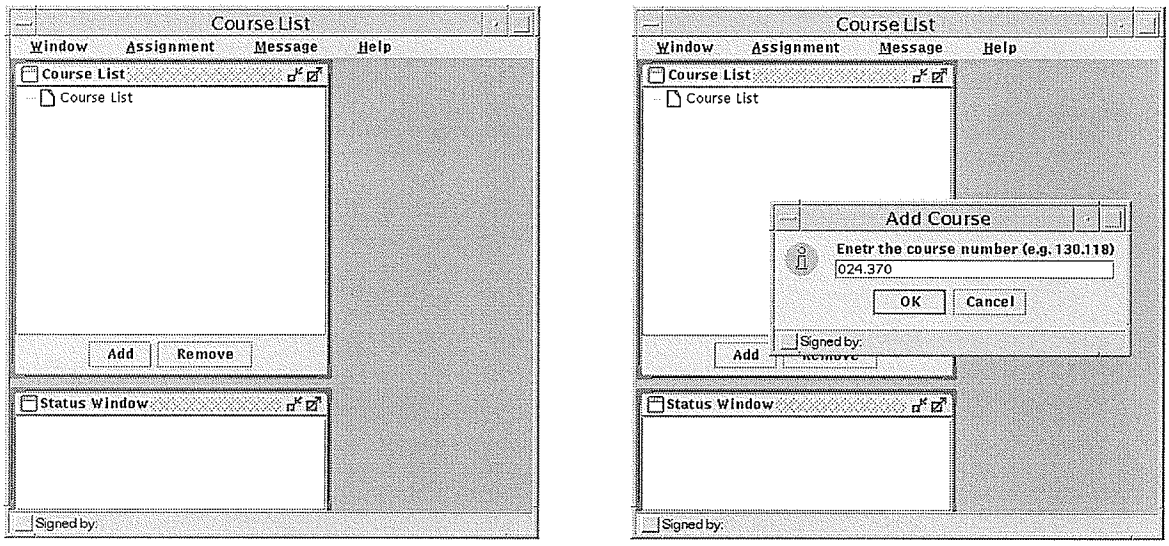

Figure 9.4 Course List Window

The general way of adding a course is using the six digit course number and then submitting it. The corresponding server will first verify if the user is eligible to add the course and then process the data and return either an error or the course information, such as the name of the course and the department and faculty that offers the course. Meanwhile, the status window displays the status of the process. Figure 9.5 shows the result of adding a course successfully for a student agent. Note that a popup menu is available containing all accessible tools.

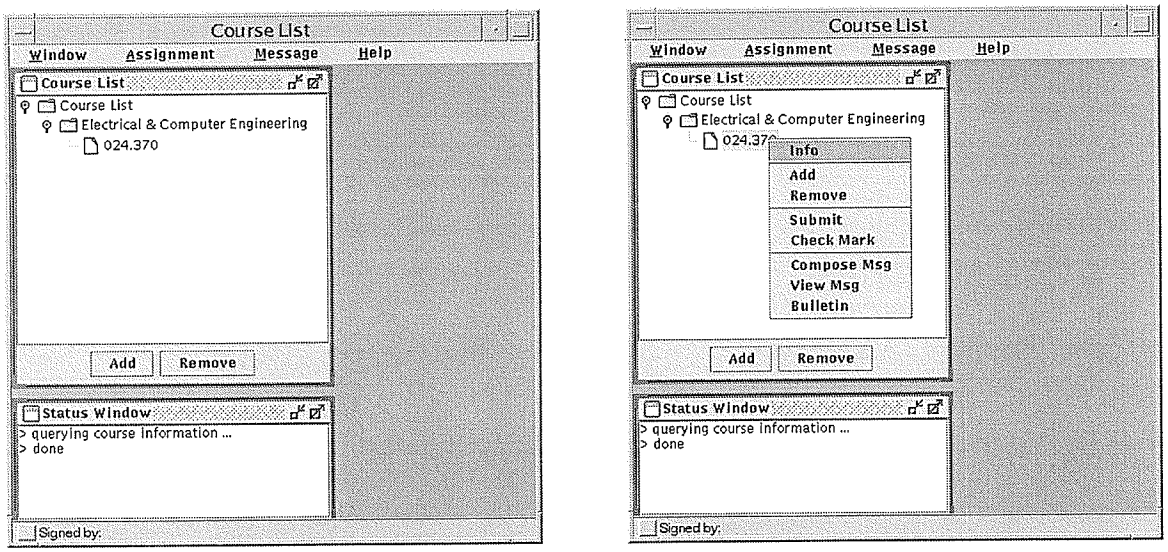

Figure 9.5 Add Course

- **The Message Tool & Bulletin Board**

One of the functions that a client agent provides is sending and retrieving messages to and from others, as well as posting and reading messages on and from a bulletin board. As described before, the message tool and bulletin board are features available to all four types of client agents. The message tools contains two components. One is the "compose component" which allows users to compose and send a message; the other is "view component" which allows a user to view received messages. There is also a bulletin board tool under the message category but with a different way of communicating. It allows a user to post and view messages in a news group fashion. The "compose component" is similar to the email tools that we use everyday which provide an editing and viewing interface for emails. The difference is that a regular email tool sends the emails based on the receivers' email addresses provided, while the "compose component" sends the messages based on the course selected. That is, the user needs to select a course in order to

send out messages to course related personnel, and no individual address is necessary for the message delivery. Figure 9.6 shows the menu used to launch these components,

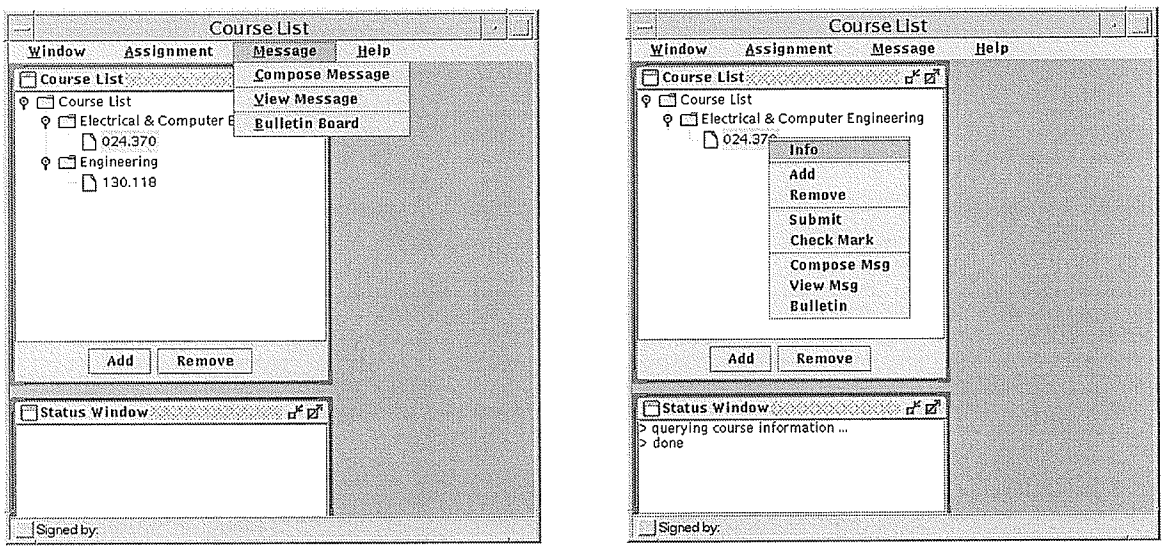

Figure 9.6 Menu

- **The Compose Message Window**

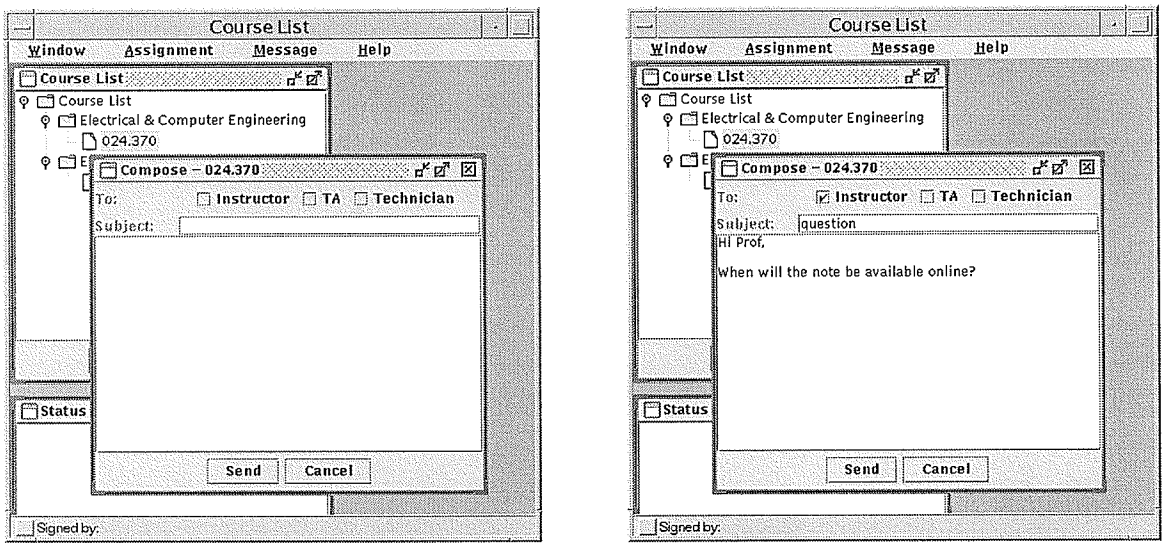

Figure 9.7 Compose Message

Figure 9.7 shows the interface of the compose message window in a student agent. The title bar displays the course number this message is related to. There is a recipient panel which lists agent types other than the student agent, which means by selecting one recipient or more from the recipient panel, this student can compose and send a message to any of these people related to the course. The subject text field holds the subject of the message, and the text area holds the message body. After finishing composing as the right part in Figure 9.7 shows, the user can click the "send" button send the message. In this example, student "liu" composed a message to the instructor of the course "024.370" asking a question.

- **The View Message Window**

If there are new messages waiting for the user, a reminder message will be displayed on the main window of the user's agent. For example, as the instructor of course "024.370", professor "meng" will have the main window like the one shown on the left part of Figure 9.8 when he logs on to his client agent. When he clicks the "View Message" image button, the "course list" window will be launched with the "view message window" open, as shown in the right figure in Figure 9.8. The upper part of the "view message" window holds a list of messages received by this user, and each entry in the list contains the course number that the message is regarding, the sender's id, the subject of the message and the date and time when the message was sent. The lower part of the "view message" window holds the message content. The "Delete" button deletes the message entries from the message list, and the "Reply" button will launch a compose window.

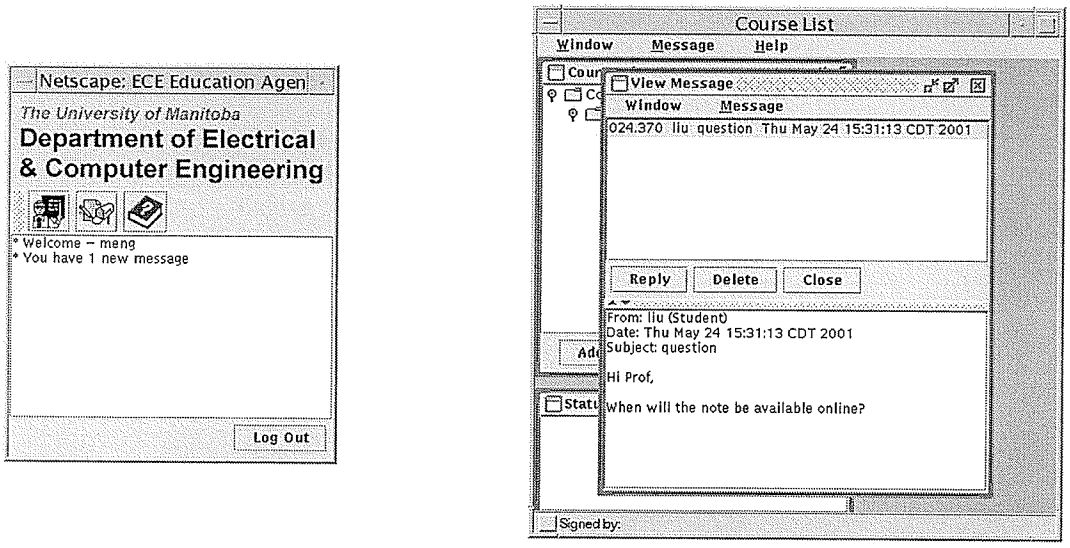

Figure 9.8 View Message

- **The Reply Message Window**

Figure 9.9 shows the reply message window. The part on the left is the "reply message" window. It is a "compose message" window except that the receiver of the message is not editable. If the receiver has logged on to his/her client agent when the message is received, there will be an information window pop up saying a new message has been received. If the receiver's "view message" window is opened when the message is received, it will be displayed on the "view message" window automatically (shown as the right part in Figure 9.9) accompanied by a beep sound.

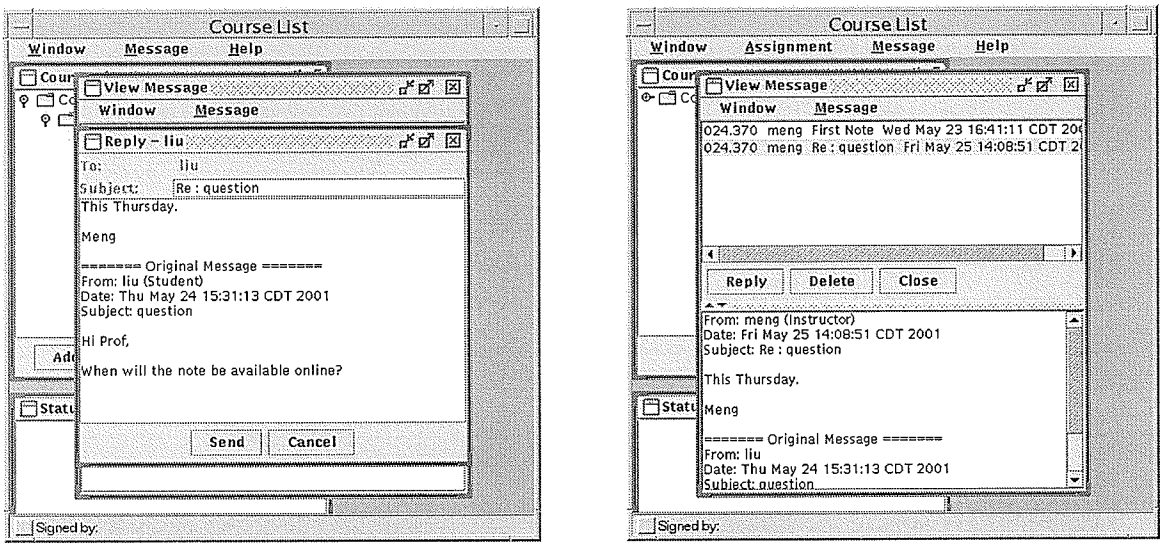

Figure 9.9 Reply Message

- **The Bulletin Board Window**

There is also a bulletin board function in the message tools. It provides a discussion and comment space for people related to any one course. People can post their comments and reply to the others, and discuss questions and problems they have during the study of the course. Figure 9.10 shows the bulletin board window. The left part contains all the current topic threads on the board. The ones with a file icon are without replies, and those with folder icon have replies to them. By selecting the thread on the left the content of the thread will be displayed on the right.

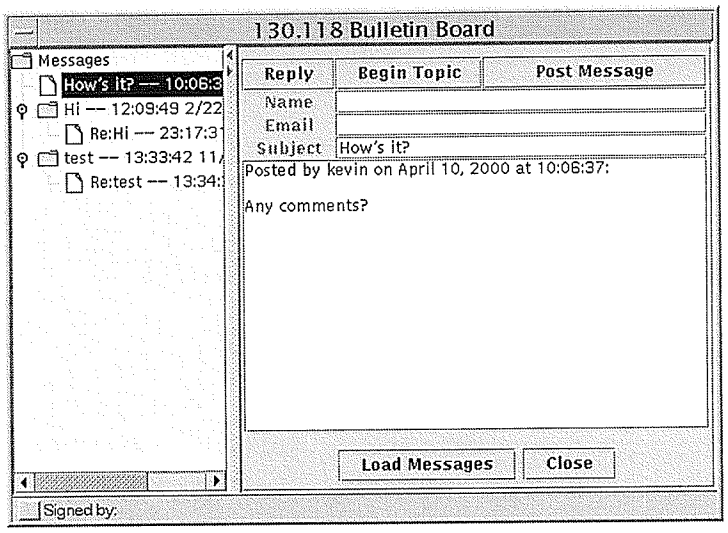

Figure 9.10 (1) Bulletin Board

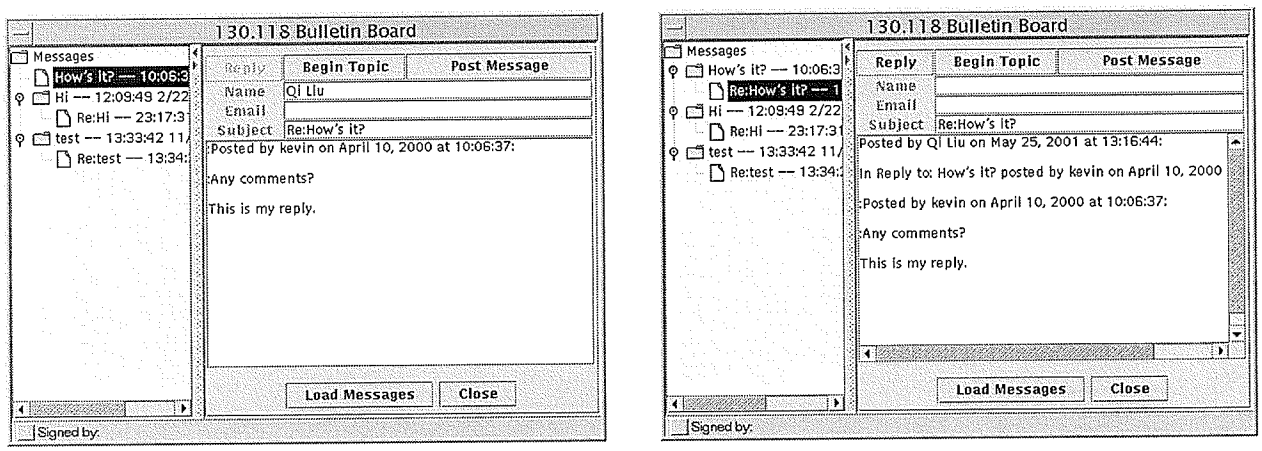

Figure 9.10 (2) Bulletin Board

The following describes the functions of the buttons on the bulletin board,

- The "Begin Topic" Button : Used to compose a message on a new topic. The field on the right on Figure 9.10 will be clear for inputting new contents.
- The "Reply" Button : Used to start composing a reply to an existing message on the board. Figure 9.10(2) illustrates this process.

- The "Post Message" Button : Used to post a message to the bulletin board. The message can either be a new message or a reply to a previous message, and it will be posted on the board almost instantly.
- The "Load Messages" Button : Used to reload the messages. If there are new messages not yet displayed to this board, it will load them.
- The "Close" Button : Close the bulletin board.

- **The Submit Tool**

The submit tool simplifies the way that students hand in their reports, assignments, or lab reports. This tool can directly submit students' files to whoever is responsible for checking and marking them, rather than students printing them and bringing them to a collecting box in front of the TA's office. Of course in some cases, the reports or assignments are more easily done by hand. For example, for assignments with mathematical equations and signs are not suitable to this tool. It is designed for submitting of computer files on-line instead of files in paper off-line. The submit tool feature is available only to student agents.

Figure 9.11 displays the procedure for submitting a file. The "Submit" menu command can be found under the "Assignment" menu, or the popup menu on the course. Figure 5.13(a) is the submit form created for the specific course. A type of either "Assignment" and "Lab Report" can be selected in the "select a type" drop-down list, and a number indicating the order of the assignment or the lab report is listed in the "select a number" drop-down list. The "Browse" button is to bring up a file dialog on the local machine so the user can select a file to submit.

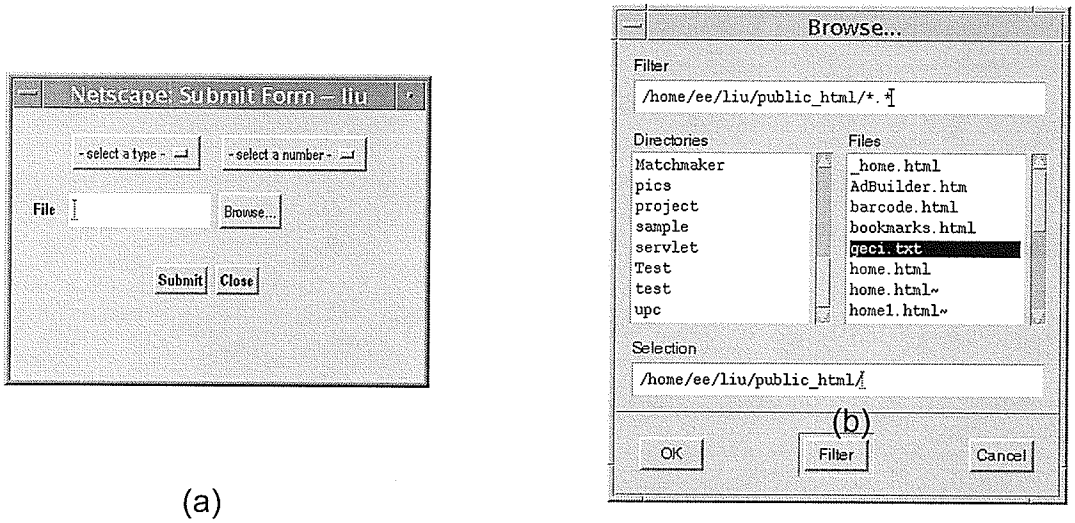

Figure 9.11 Submit Form

- **The Mark Tool**

The mark tool provides a TA (or Instructor) with a way of viewing and marking the submitted files from students related to assignments and lab reports. When the TA logs on to his/her client agent, there will be a message stating that a new assignment has been submitted. There is a "Mark assignment" menu under the "Window" menu which will bring up the mark tool. Figure 9.12(1) shows the mark tool interface and an example of a TA viewing and marking a submitted assignment.

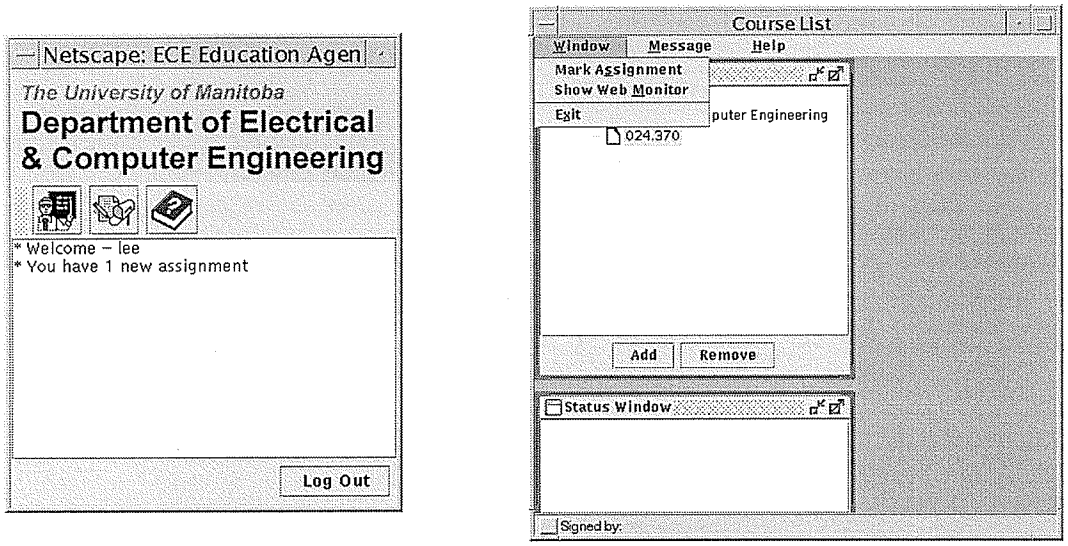

Figure 9.12 (1) Mark Tool

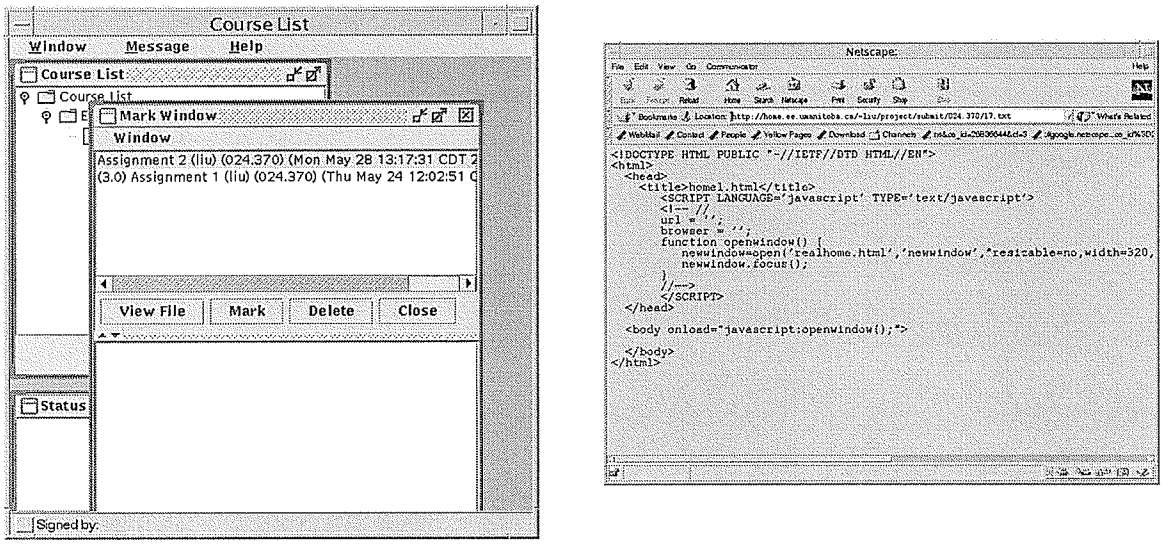

Figure 9.12 (2) Mark Tool

Figure 9.12 (2) shows the interface of the mark tool. The upper part of the tool contains a list of files submitted to the TA. Each list item consists of the file type (assignment or lab report) and

the sequence number, the student user name and the submission time. For a marked item, the mark will appear at the front of the item. To view certain items, the TA selects it and clicks on the "View File" button. A web browser containing the content of the submitted file will display, or a specific application will be loaded according to the file type of the submitted file. Files with certain extensions belong to certain MIME types, and the web browser's configuration file contains information on which application is for which MIME type. For example, if the file is a PDF file with the file extension of ".pdf", the web browser will load Adobe Acrobat Reader to display that file.

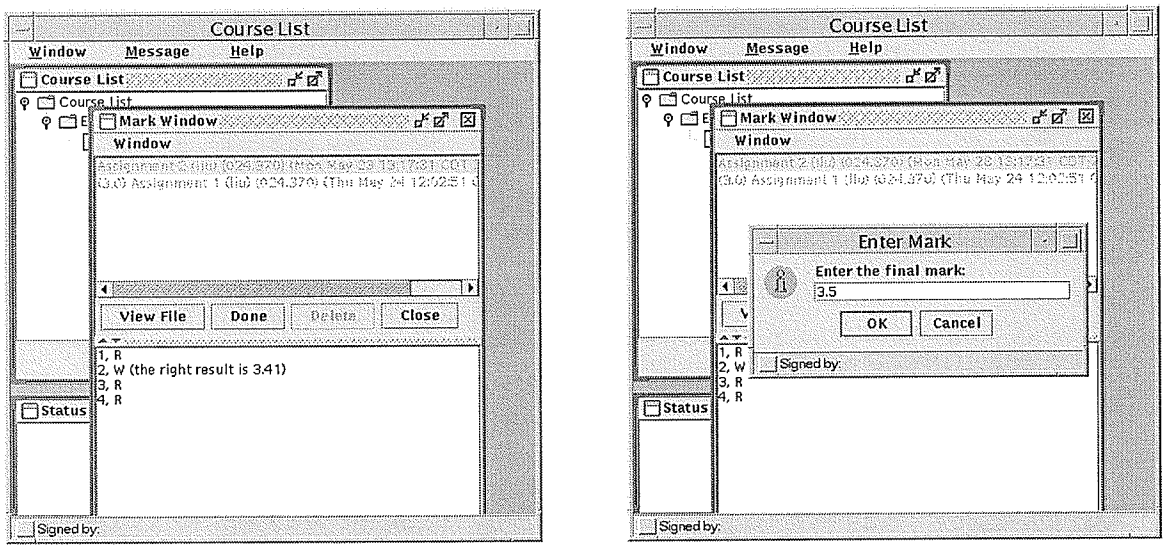

Figure 9.12 (3) Mark Tool

After the TA views and examines the submitted file, he or she will try to mark it just like marking assignments and lab reports on paper. The first step to mark a file is selecting the file, and then click on the "Mark" button. This will result in the upper portion of the "Mark Window" containing the submission list being disabled and the lower portion containing the text area being made editable. Also, the "Mark" button turns into a "Done" button to be used to finish marking process. The TA lists his/her comments for the file in the text area, as shown in Figure 9.12(3).

After the TA finishes his/her comments, he or she will then click on the "Done" button, and a dialog will pop up requesting the final mark for this submitted file. After the final mark is entered, this process of marking a submitted file comes to an end, and the item on the TA's submitted file list is updated with the mark added to the front of the item which indicates that the file is marked. The TA is also able to remove any of the listed items from the upper window.

- **The View-Mark Tool**

The student agent has a View-Mark Tool which is designed to view marks the student has got for assignments and other types of documents he or she has submitted. When the student logs on to his or her student agent, a notification message will remind him or her of new marks in his client agent. He can open the View-Mark tool from the "Assignment" menu or from the popup menu on the course. Figure 9.13 shows the reminder message and the tool's interface. There are two portions to the View-Mark tool. The upper portion is a list of all marks this student received, and the lower portion is a display area for comments provided for each specific assignment or report. Each of the entries in the list contains a description of the mark, including the final mark given, the name of the assignment, the course number which the assignment is for, and the date and time that the mark was given. Selecting an entry will display the corresponding comments in the lower window. Also, the student can manage his or her mark list by deleting old ones from the list.

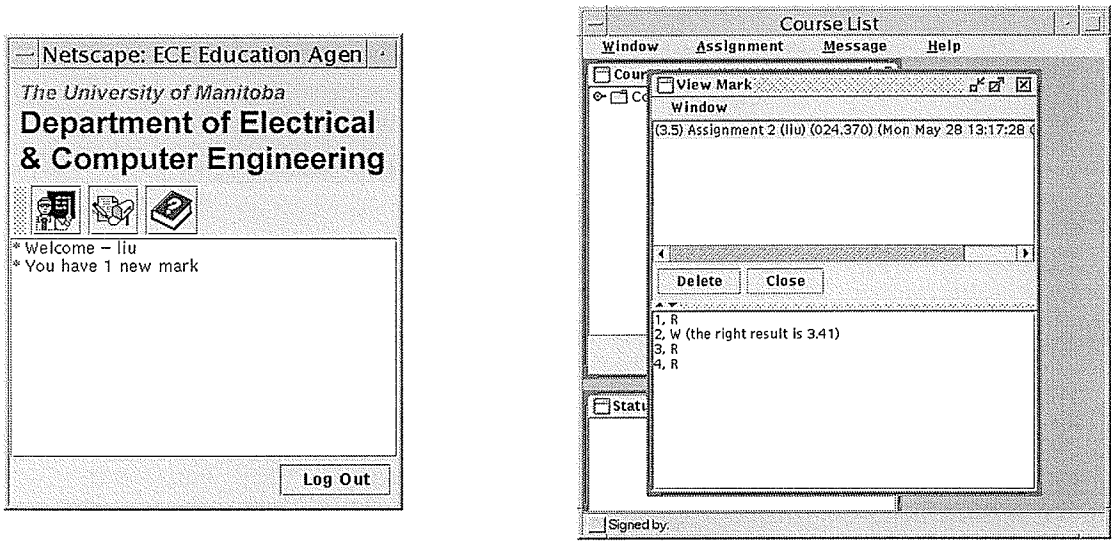

Figure 9.13 View Mark

- **The Web Monitor Tool**

Today's students are relying more and more on getting information from the Internet. They are using the Internet as an ultimate alternative to libraries to write their reports or do other home-work, and often the professors put class notes and assignments on their own website as well. The web monitor tool is designed to assist users to manage and browse websites more efficiently. It allows users to add websites of their own interest and monitor their updating information. The web monitor tool can be loaded from the "Window" menu. Figure 9.14 shows the tool's interface which consists of a table displaying websites' information. There are a group of function buttons as well.

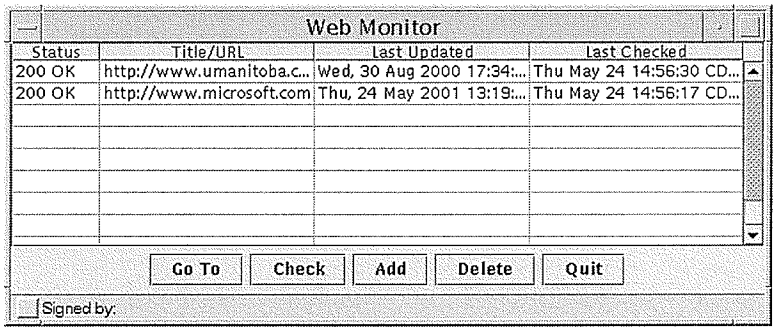

Figure 9.14 Web Monitor Tool

The information for each website includes the status of the site, the last modification date and the last checking date. The status and modification date are checked by the AEA service agents every 30 minutes and reported on the monitor. Thus, users can monitor whether those websites have been updated recently to determine if it is necessary to visit them, or just use the web monitor as a bookmark portal that can be accessed anywhere over the Internet. To add a new website to the web monitor, the user can simply click the "Add" button, and an input dialog will pop up to allow the user to enter the URL of the website that he or she wants to monitor. When the user clicks "OK" button, the new website will then be added onto the web monitor. Figure 9.15 shows this process. Users can also remove websites from the monitor by selecting the website and clicking the "Delete" button.

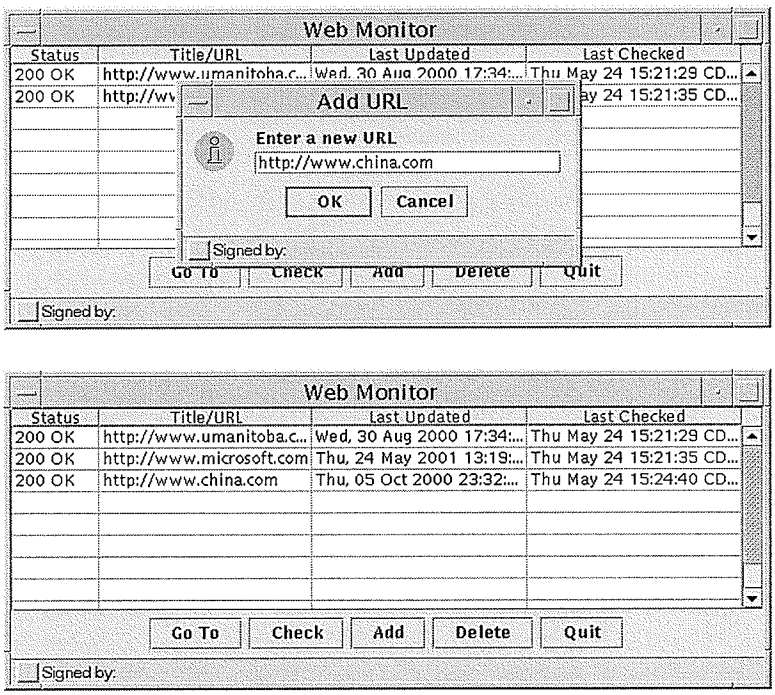

Figure 9.15 Add URL

CHAPTER 10 Conclusions & Future Work

10.1 Conclusions

This thesis reviewed agent technology and some available software packages suitable for developing a web based multi-agent system.

JAT*Lite* provides a robust agent infrastructure in which agents go through a message router to exchange buffered messages between other agents on the Internet. It works well serving the message based communicating needs for the AEA system.

The AEA system achieved the design goal which is to enhance the communication and interaction among people involved in an educational system. By providing users with a set of tools, the system is able to reduce the effort users have to make to perform certain functions such as submitting assignment. This will help students be able to focus more on studying. The system also demonstrates that the concept of a multi-agent online system can be beneficial to a large scale institute.

10.2 Future Work

- Security. Sensitive information is passed around in the AEA system, such as student numbers and PIN numbers. The security issue is important. JAT*Lite* only addresses the security issue at a high level, which uses a user name and password combination for each agent using the message router. Messages transmitted in data streams are not protected. If someone can intercept messages and spot what is inside, there is no point in deploying authenti-

cation at a user level. The JSSE (Java Secure Socket Extension) package [Java] could be used to increase the security of the system. It implements a Java version of SSL (Secure Sockets Layer) [Java] and TLS (Transport Layer Security) protocols [Java] and includes functionality for data encryption, server authentication, message integrity, and optional client authentication. Using JSSE, developers can provide for the secure passage of data between a client and a server. For the AEA system, the clients are both clients and service agents. The server is the router.

- Functionality. There were several functions developed for each type of client agent in this system, and they certainly cannot meet all the requirements. There is always room for new functions that are suitable for such system. For example, the professor can put links to resources related to the course on the Internet to his agent. Students who take the course can view those resources from their agents automatically.
- Roles. The system supports four types of client agents corresponding to four different roles. The administrator agent's interface was not developed, and its functions were fulfilled by working directly on the mSQL database to update the agent's knowledge base. There are also other types of client agents that may be necessary to work with the existing ones, e.g., administrative roles like the secretary of the department.
- Update. There are several external programs that agents in the system rely on for information. If the location, name, or requesting parameters of those programs are changed, the agents must update their internal processes in order to be able to continue to contact those programs. Other work is to have such updates organized and given a generic interface to make it part of the administrator's function.

References:

[Cornell 96] Cornell Gary, "Core Java", 1996

[Date 97] Date C. J., "A guide to the SQL standard: a user's guide to the standard database language SQL. 4th ed", 1997

[Finin 97] Tim Finin, Yannis Labrou, and James Mayfield, "KQML as an agent communication language", 1997

[Harmon 98] Harmon P., Waston M., "Understanding UML: The Developer's Guide - With a Web-Based application in Java", 1998

[Hughes] Hughes Technologies, "http://www.Hughes.com.au"

[Imaginary] Imagianry Productions, "http://www.imaginary.com"

[JATLite] JATLite, "http://java.stanford.edu"

[Java] "http://java.sun.com"

[Jeon] Jeon H., Petrie C., Cutkosky M.R., "JATLite: A Java Agent Infrastructue with Message Routing"

[Ketchpel 94] Ketchpel S. P., "Software agents. Communications of the ACM", 1994

[Norwell 98] Norwell MA, "Antonomous Agents and Multi-agent Systems", 1998

[Patchett 97] Patchett C., Wright M., "The CGI/Perl Cookbook", 1997

[Petrie] Petrie C.J., "Agent-Based Engineering, the Web, and Intelligence"

[Pooley 99] Pooley R.J., "Using UML: Software engineering with objects and components", 1999

[Reese 97] Reese, George Henkle, "Database programming with JDBC and Java. 1st ed.", 1997

[Stephens] Stephens L.M., Huhns M.N., "Database Connectivity Using an Agnet-Based Mediator System"

[Sun] The Java Tutorial, "http://java.sun.com/docs/books/tutorial/"

[Tittel 96] Tittel E., Gaither M., "Web Programming Secrets with HTML, CGI, and Perl", 1996

[Weinman 96] Weinman W., "The CGI Book", 1996

[Wooldridge 95] Mike Wooldridge and Nick Jennings, "Intelligent Agents: Theory and Practice", 1995

[Zukowski 99] John Zukowski, "Definitive guide to Swing for Java 2", 1999