A Multi-Agent Based Tourism Kiosk on Internet

Chris Yeung, Pang-Fei Tung,
Department of Electrical & Electronic Engineering
University of Hong Kong
Jerome Yen
Department of Computer Science
University of Hong Kong

Abstract

*In this paper we discuss the implementation of a multi-agent based Tourism Kiosk for Hong Kong tourism industry on the Internet. This system allows the users to retrieve the most updated information about Hong Kong through any Java-enabled web browser. The complete system consists of a set of software agents who handle various information categories, such as, hotels, shopping centres, and cinemas, etc. The Knowledge Query and Manipulation Language (**KQML**) was selected as the agent communication language to develop our system, which takes care the exchange of information among agents. The users are able to send their queries to the system, then the system will dispatch one or more agents, depend on the analysis of the query, to search and retrieve information for the users.*

1. Introduction

With the explosive growth of the Internet and Internet applications, the World-wide Web (WWW) has captured the position to be the major enabler of building global information center. For example, digital library has become an important research area because of the recent development of the Internet and WWW technologies. In 1993, six research projects developing new methodologies or technologies to support digital library have been funded through a joint initiative of the National Science Foundation (NSF), the Department of Defense Advanced Research Projects Agency (ARPA), and the National Aeronautics and Space Administration (NASA). Since then, digital libraries research has been recognized as a national challenge in the U.S. High Performance Computing and Communications Program (HPCC). National challenges are the research on applications that may create nation-wide impact on the nation's competitiveness and the well-being of citizens.

Source information targeted by these digital library projects ranges from text, numerical data, visual images and symbols, sounds and spoken words to video clips. They may reside physically at thousands of geographically dispersed locations. When stored digitally, organized and connected through networks, the information resources become the central components of a digital library and available to authorized users from around the world.

For the tourism industry, there is no surprise that Internet has become a new media to deliver first hand information about services or products to the customers from around the world [7]. According to a survey based on the responses from sixty-one European city tourism offices in 1995, 53% of interviewed tourism managers already had Internet access, whereas 40% were planning to install Internet in the near future [10]. We believe such ratios can be higher in the tourism related industries, such as, major hotel chains, airlines and car rental companies, in the US.

Unfortunately, information gathering on Internet about tourist destinations is less effective as it supposed to be. The reason is that there is no organized URL directory (yellow book) for the users to know how and where to access to such information. Users might have to browse several sites before reaching the desired information. Moreover, most of today's WWW-based tourism information systems provide *static* web-pages. The term *static* refers to the pre-designed HTML-pages which seldomly been updated [1]. The major drawback of such systems is that it can hardly provide any useful information if the owners cannot constantly update the HTML-pages.

In order to solve such problem, we propose a multi-agent based tourism kiosk which runs on the Internet and aims at enhancing the effectiveness and efficiency of information retrieval on tourism information. Moreover, the front-end, which is a kiosk – a special interactive graphical user interface (GUI), has been developed to allow the users to enter their queries and to trace the searching and retrieval of the information. The information and data about city, hotels, and other items which are important to the tourists, are stored on several distributed databases.

In this system, a group of software agents have been developed with KQML and Java to work cooperatively with each other similar to that of the information desks in various tourism related industries or services, such as, hotel and Mass Transit Railway (MTR). Although the system has been implemented only to support a specific geographic area, i.e. Hong Kong, it is truly portable and scalable. The system was developed with the objected-oriented approach and that provides a great flexibility to the future expansion or migration. Only few modifications are required to migrate the kiosk to other regions.

In section 2, we will discuss briefly about the multi-agent systems. Technologies that used to develop the multi-agent system will be discussed in section 3. Architecture of the agent-based tourism society will be discussed in section 4. Implementation will be discussed in section 5. This paper is concluded with some discussions about the limitations of the multi-agent system and recommendations for future research.

2. Multi-Agent System

Software agents, especially multi-agent systems, have become an important research topic recently. The term *agent* has been used widely in many different areas, from personal assistants [6] and smart interfaces [3], to intelligent information retrieval systems [9] and intelligent searching spiders. Although a firm definition on agent has not been agreed upon, most works accept that an agent should have the following properties:

- Autonomy: agents should be able to operate without direct human's intervention, and have certain control over their actions and how to reach their goals [2].
- Social ability: agents should be able to communicate with other agents via agent-communication language in order to share information or exchange messages [5, 8].
- Reactivity: agents should be able to perceive changes occurred in the environment, and respond to such changes [11].

In our multi-agent system, the agents are basically independent software processes, which are run fully distributed and autonomously. Each agent has a special task to perform and special goal to achieve. Agents are able to communicate with each other and respond to the change of the environment. The agents in our system use Knowledge Query and Manipulation Language (KQML) [4] as the communication language to communicate, that is, send or receive messages, with other agents. Through the development and test of these coordinated agents, it is possible to explore the feasibility of building a totally autonomous agent-based society to support tourism industry.

3. Technology Platform

The most important design criterion of our system is platform independence. All core system components, which include: behaviour of agents and protocols of communication, searching and retrieval of information on the distributed databases, and interface for end-users which can be accessed through Internet, are all developed using the Java or other technologies which are platform independent. The linkages among the components or various agents are provided by the KQML, which encapsulates all the necessary message passing and communication capabilities which are needed by our system.

3.1. Agent Messaging – KQML

In order to provide an effective, constructive and intelligent interaction among software agents, a common understanding of the terms used in the given context is required. This implies that there is a need for three fundamental and distinct characteristics [4]: a common language for all the agents; a common understanding of how the knowledge or messages are exchanged over Internet; and the mechanism to support the exchange of the first two items.

The Knowledge Query and Manipulation Language (KQML) is a language for exchanging information and knowledge among software agents over the network. It was developed by the ARPA as part of the ARPA Knowledge Sharing Effort, which aimed at developing techniques and methodology for building large-scale knowledge bases with sharable and reusable properties. It should be noticed that KQML is not a transport protocol like http or ftp. It is a message format as well as a message-handling protocol to support run-time knowledge sharing among agents. The language allows an agent to interact with other agents in a confined environment or to share knowledge or information to support complex problem solving in an agent society.

For example, a KQML message generated by an agent to look up room prices of a hotel may be encoded as following:

```
(ask-price:sender agent
          :receiver hotel-server
          :ontology TOURISM
          :content (room-price
                    :name Hotel_A
                    :room twin
          )
)
```

The above KQML message can be divided into three layers:
1. The content layer, designated by the :content keyword to describe the message data itself;
2. The communication layer which describes lower level communication parameters through the :sender and :receiver pairs; and
3. The message layer, denoted by the value of :ontology, determines the kind of interactions one can have with a KQML-conformance agent.

As illustrated in the above example, KQML is highly generic and neutral. KQML separates the contents of the messages and the contexts about how the messages should be interpreted or executed. KQML does not attempt to understand the contents of the messages, but it detects how the messages should be delivered and what to do when a message reaches the other end. It is up to the corresponding agent to interpret the message content based on the information about the context.

3.2. Agent Behavior – JATLite/JAT_0.3

JATLite/JAT_0.3, developed by the Stanford University, is the core of the multi-agent development tools that used in our multi-agent system. It provides a set of fully functional templates. It is written entirely in the Java language which supports the construction of software agents that able to communicate peer-to-peer with other agents through the Internet. Such environment that consists of software agents is called agent community. All agent messages use KQML as a top-level protocol or message wrapper. The architecture of the JAT was specifically designed to allow the replacement and specialisation of major functional components, which include the GUI, low-level messaging, message interpretation, and resource handling. Consequently, it is desirable to use JAT as a platform to develop a wide range of agents to support different application domains.

3.3. Agent Information Management – JDBC

Every agent in the society must be able to access real-time information in various levels of details subject to its behaviour and security constraints. The information is best captured in database systems which support the distributed processing and data security across private networks and Internet. In order to guarantee that any agent be able to be connected to a specific database engine, the Java DataBase Connectivity (JDBC) programming interface has been selected to meet the requirement of vendor independence. In the actual implementation, all information are managed by the mSQL[1]. Which is a lightweight database engine designed to provide fast access to stored data with low memory requirements. It also has been designed to work in a client/server environment over a TCP/IP network.

4. Architecture of Agent-based Society

In this section we will provide an overview about the system architecture of our multi-agent system. At the present stage, a prototype of an agent society has been developed as shown in Figure 1. There are four fundamental agents in this agent society, namely Information Agent (IA); Agent Name Server (ANS); Category Group Agent (CGA); and Client Agent (CA). Most applications can be developed by defining agents on the top level through the inheritance property enforced by the object-oriented methodologies. The framework of this agent society more or less can be specified by the coordinating manner of the agents. The behavior of each agent is described below:

Information Agent (IA)

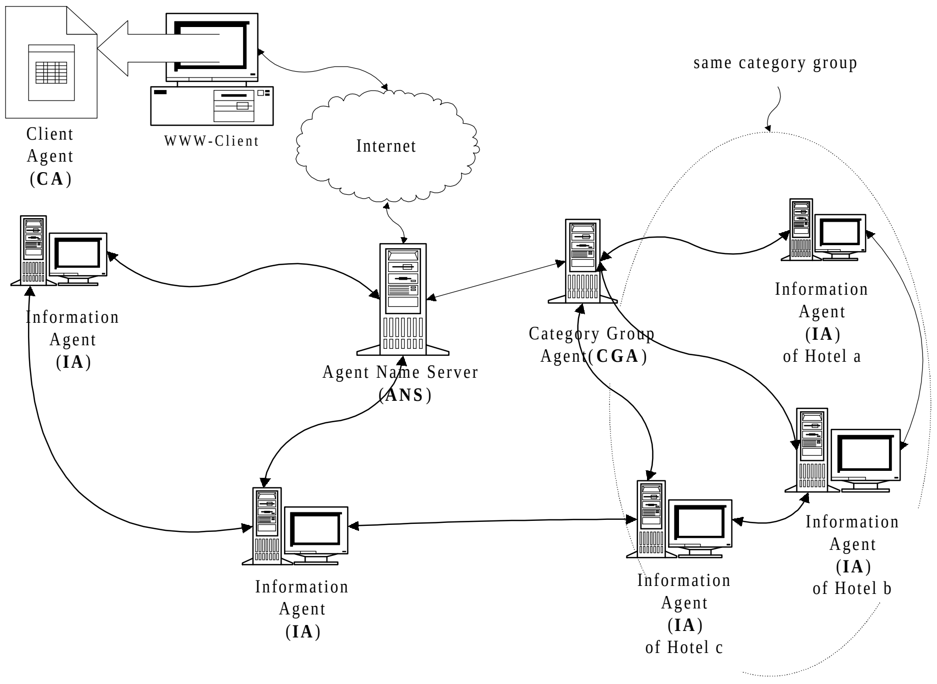
- It is the basic entity of the agent society, which is resided in any site that interested in offering services, such as, tourism information of hotels.
- It must be able to access data that stored locally in order to process the queries that come from other agents.
- A site may optionally limits the agents to access to non-critical data only (address, phone number, pricing or types of room services in case of a hotel) which are also available to other IAs.
- Once invoked, an IA must register itself to the ANS (will be discussed below) and be ready to

[1] mSQL is a public domain software from Hughes Technologies

offer services to other agents.

Agent Name Server (ANS)

- ♦ It is the core agent to maintain and coordinate a list of information agents, which are categorized into groups according to the types of information these agents provide.
- ♦ It spawns a CGA (to be discussed below) to handle those IAs in the same category.
- ♦ It acts as a communication channel for a set of cooperating agents under the condition that there may be site-specific security restrictions, which forbid direct path of agent dialogue . For example, the Java applet embedded in the HTML page is allowed to make connection to the single site from which it was downloaded only in most Java-enabled browser.



same category group
Client Agent (CA)
WWW-Client
Internet
Information Agent (IA)
Agent Name Server (ANS)
Category Group Agent (CGA)
Information Agent (IA) of Hotel a
Information Agent (IA) of Hotel b
Information Agent (IA) of Hotel c
Information Agent (IA)

Figure 1. Architecture of the agent society

Category Group Agent (CGA)

- ♦ It is an agent that performs likes a proxy server to reduce the workload of ANS and to provide data cache for the IAs in the same category. This agent follows an object-oriented design template, which encourages the creation of a group object to assure that the same behavior can be created in its subordinated objects.
- ♦ Once the CGA is created, the ANS communicates with the CGA directly instead of the IA for data or information. When ANS submits a request to a CGA, the CGA will check the data in the cache first before asking the IAs to search for the information required. Therefore it helps reducing network traffics and improves the overall system performance. The improvement is more significant if a slow connection between the ANS and an IA does exist.

Client Agent (CA)

- ♦ A CA handles all end-user queries and the interface with the ANS, which in turns dispatches agents to serve the queries.
- ♦ We can classify the CA as the format of a Java *applet* that embedded in the HTML page and be able to be downloaded on the user's machine for execution. The applet provides the GUI and assists the user in the browsing of the information that returned by the IAs. We have implemented such applets with digital map technology, which speeds up the display and manipulation of geographic information by an order of magnitude. In section 6, we will describe this CA applet with emphasis on user-friendliness.

Figure 2 illustrates the two most active communication phases among various agents. The ANS remains passive most of the time until either a new IA requests for registration or a CA intends to forward a query. The registration activity is called the society binding phase and the query activity is performed during the information retrieving phase. Once a CA connection is granted, the ANS will inform the CA about all initially available services through the IAs. All subsequent queries are either going through the ANS or the cached CGA before reaching those IAs concerned.

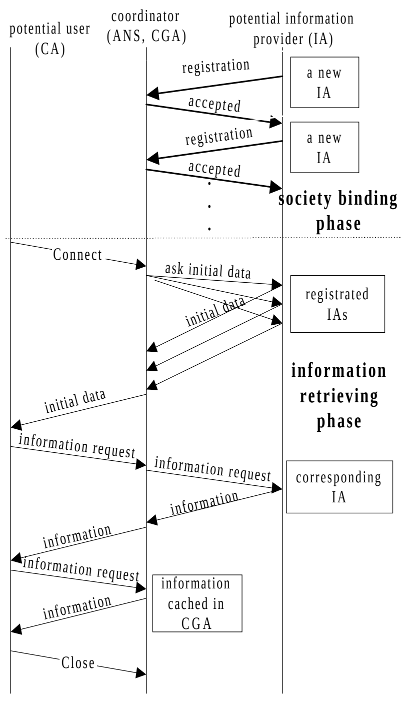


Figure 2: Agent communication phases

4.1. Tourist Society – A Scenario

To better explain how the above architecture serves as an agent society for tourism, we will walk through the process that involved a retrieval of the detail information about the *Park Lane Hotel* for a user.

Figure 3 depicts the four process flows among the agents as follows:

(1) A Client Agent sends a KQML message to the ANS according to the request from the end-user:

```
(evaluate :sender Client-Agent908
          :receiver ANS
          :language KQML
          :ontology HKMap
          :content (ask-detail
                     :type hotel
                     :name (Park Lane Hotel)
                     :maparea (Causeway Bay)
          )
)
```

(2) The ANS first searches through the resource table to find out which IA has the information on *Park Lane Hotel* , then ANS will send a KQML message to that IA for information;

```
(evaluate :sender ANS
          :receiver Agent-A
          :language KQML
          :ontology HKMap
          :content (ask-detail
                     :from Client-Agent908
                     :type hotel
                     :data (Park Lane Hotel)
                     :maparea (Causeway Bay)
          )
)
```

If a CGA of hotel exists , ANS will send this KQML message to the CGA directly without searching through the table.

(3) The IA (Agent-A) sends the result from the database back to the ANS;

```
(evaluate :sender Agent-A
          :receiver ANS
          :language KQML
          :ontology HKMap
          :content (tell-detail
                     :from Client-Agent908
                     :type hotel
                     :data (Park Lane Hotel)
                     :maparea (Causeway Bay)
                     :result ([4][address : 129 , Liu To Road
                          , Causeway Bay][tel :
                          (852)2745617][Price of room A:
                          HK$ 700/ per day][Price of
                          room B: HK$1000/ per day]
                          )
          )
)
```

(4) ANS sends the received result back to CA , and CA displays the result to the user.

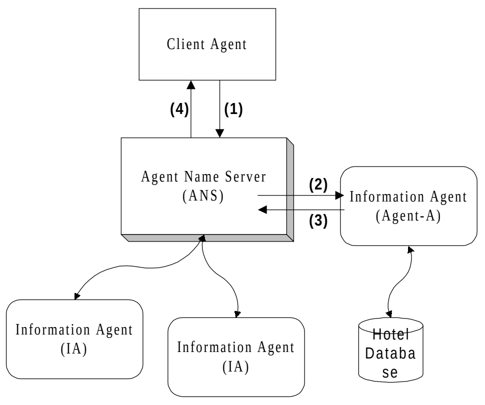
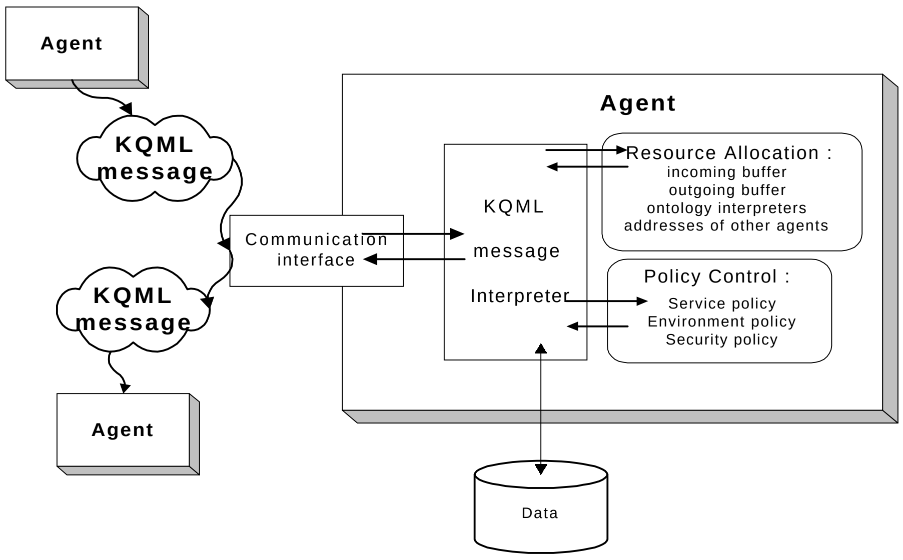


Figure 3.Communication processes for a Hotel query



Agent
KQML message
Communication interface
KQML message
Agent
Agent
KQML message Interpreter
Resource Allocation : incoming buffer outgoing buffer ontology interpreters addresses of other agents
Policy Control : Service policy Environment policy Security policy
Data

Figure.4. Agent object model

5. Agent modelling and Implementation

The four agent entities described in the previous section all inherit from an abstract agent object model (Figure 4). This agent model consists of four major components: a KQML message interpreter, a set resource allocation methods, a set of control policies, and a communication interface.

5.1. Resource Allocation

Every agent requires different resources. Some require incoming and outgoing system buffers and some need accessing and managing ontology interpreters. These resources have been organised as a hierarchy of hash tables for efficient access. It is important to understand that this hierarchy dose provide information about where and how an agent receives information in order to exhibit autonomous behaviour. For example, before an agent is able to start a communication to another agent, it needs to know the actual IP address of the receiver by searching through the agent table in the resource allocation.

Table 1. Resource tables

Resource Allocation HashTable

Keyword (String)	Value (Java Object)
"message buffer"	KQMLmessage
"interpreter"	Ontology Interpreter HashTable
"agent table"	Cooperating Agent HashTable

Ontology Interpreter HashTable

Keyword (String)	Value (Interpreter object)
"database"	DatabaseInterpreter.class
"HKMap"	HKMapInterpreter.class
"agent"	AgentInterpreter.class

Table 1 shows a simplified two-level structure for resource allocation. The top table incorporates two other hash tables, one for managing ontology interpreters and the other for maintaining links with other cooperative agents. The bottom table instructs an agent to locate the necessary interpreter for every ontology in the KQML message. For example, if a message has an :ontology value of "database", then the Java interpreter DatabaseInterpreter.class will be loaded to decrypt and process the :content part of the message. It should be noted that an interpreter can be located at any internet site other than the place where the agent resides.

5.2. KQML Message Interpreter

Once an incoming KQML message is detected, it will be passed to the KQML message interpreter through the communication interface. The interpreter first determines whether the sender of the message is an already known agent or a new agent. If it is a new agent the interpreter will update the resource table by including the new agent. Then the interpreter dissembles the KQML message into tuples of keywords and value pairs. After that, the interpreter will try to interpret the message according to the specified ontology (the message layer of KQML) by loading the matched ontology interpreter, which is the element in the agent resource field. The following is the sample Java code used in the interpreter:

```
String Ontology =  KQMLmessage.getValue("ontology");
Interpreter  =  (Interpreter)(agent.getResource("interpreter"
                    )).getElement(Ontology);
```

For each ontology interpreter, the message is interpreted according to the content layer contained. For example, consider the following KQML message:

```
(evaluate :sender Agent-A
          :receiver Agent-B
          :ontology database
          :content (update-data
                   :name Hotel-DB
                   :item 302
                   :value 500
          )
)
```

The KQML message interpreter of Agent-B will load the ontology interpreter of database to interpret the content layer of the above message. As a result, the item 302 in the database Hotel-DB will be updated by Agent-B.

Therefore, an agent can have different ontology interpreters to handle different task or request. Each of these ontology interpreters actually implements the knowledge that agent is supposed to have on a particular subject. Using the Java technology, an interpreter can be downloaded during runtime to guarantee the access of most updated information any time.

5.3. Policy Control

At current stage, there are three aspects of the control policies on the behaviour of an agent.

1) **Service policy** controls the availability of the services provided by a certain agent. For example, some services from an agent may be available only during office hours, otherwise the agent simply rejects connection requests from other agents for these services.
2) **Environment policy** controls the behaviour of the agent on varying environmental parameters. For example, the IA in our multi-agent system will try to detect the connection state to ANS periodically. In case that the ANS is down, the IA will try to register itself again automatically when the ANS is rebooted.

3) **Security policy** has always been a serious issue in any networked computer system. In a multi-agent system, there are security requirements on the services that provided by the system. This implies that a service from a particular agent will be made available if and only if certain security constraints have been met. For example, consider the following situation as shown in Figure 5.

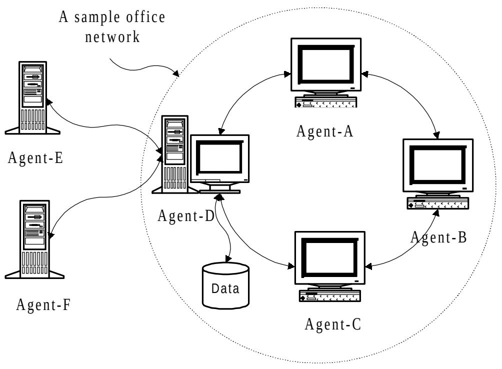


Figure 5 : a sample multi-agent system

Agent-D as shown above is the agent able to handle the request from other agents on database. The services provided by Agent-D and the corresponding security level are listed in following table.

Table 2: The service and security requirement

Service	Security requirement
retrieve data	trustful, locally
update data	locally

The classification of agents in Figure 5 is listed in Table 3.

Table 3: classification on sample agents

Agent Name	Security level
Agent-A	locally
Agent-B	locally
Agent-C	locally
Agent-E	trustful
Agent-F	untrustworthy

Therefore, neither of the services would be available to Agent-F; Agent-E would only be able to retrieve data from Agent-D; all the agents in the local network as shown in Figure 5 are able to retrieve data as well as update the data in database.

5.4. Communication Interface

The communication interface as shown in Figure 4 provides as a common channel for the agent to deliver and receive the KQML message from other agents through the TCP/IP network. Besides, it isolates the agent from actual network protocol by providing an abstract and encapsulated mechanism.

As mentioned in section 3, Java is the programming language that was designed to work over the Internet. In fact, it is far easier to implement a network system in Java than most of other languages. For example, the following figure shows a multithreading TCP/IP socket connection between a server (agent) and a group of clients.

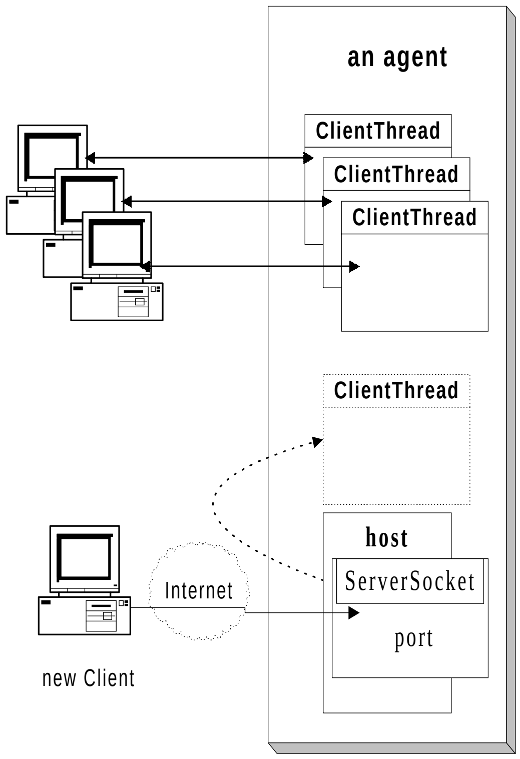


Figure 6. The multithreading socket connection

<u>*ServerSocket*</u> as shown above is a class which enables a Java system to accept connections from clients(agents) across network by opening a <u>*ServerSocket*</u> on a particular local port number.

The agent(as an ANS) will hand the new connection from a client to a new instance of the <u>*ClinetThread*</u> class that will receive the message from that client or agent. Hence, the agents in our system are able to handle multi-connection from other agents.

6. Front-end Applet as Client Agent

To demonstrate the functionality of the multi-agent kiosk system, we have implemented an Internet site to offer tourists the information about Hong Kong. Once a user visits our web page, it will undergo the information retrieving phase as described in section 4 (Fig. 2). Figure 7 below depicts the initial layout of the page loaded through a web browser.

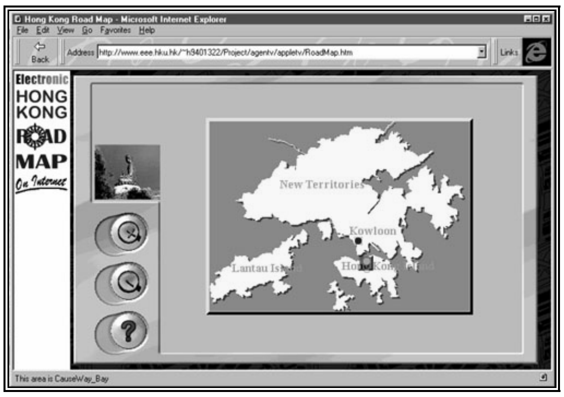

Hong Kong Road Map - Microsoft Internet Explorer
http://www.eee.hku.hk/~h9401322/Project/agentv/appletv/RoadMap.htm
Electronic HONG KONG ROAD MAP On Internet
New Territories
Kowloon
Lantau Island
Hong Kong Island
This area is CauseWay_Bay

Figure 7. Initial layout of the Tourism Kiosk

The Java *applet* first displays an outlined map of Hong Kong, then wait for user to select any of the predefined regions on the map. The figure shows all the regions that available on the system. After a particular region has been selected, for example, *Causeway Bay*, which is one of the busiest business districts in Hong Kong, the client agent (the *applet*) starts to communicate with the ANS to search for the information agents which can provide the tourism information on that particular region. Then, the corresponding information agents send the initial data back to the client agent through the ANS. Besides, a more detail map of *Causeway Bay* will be loaded and displayed to the user as shown in Figure 8.

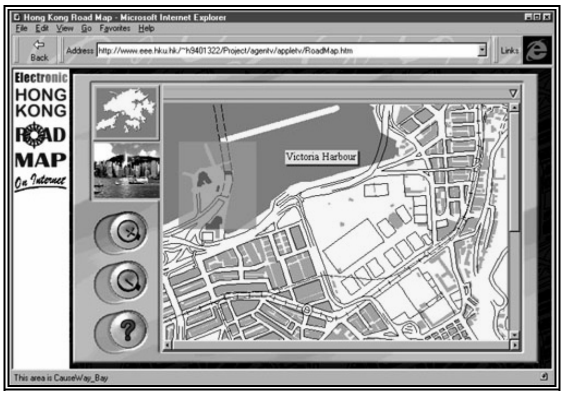

Hong Kong Road Map - Microsoft Internet Explorer
http://www.eee.hku.hk/~h9401322/Project/agentv/appletv/RoadMap.htm
Electronic HONG KONG ROAD MAP On Internet
Victoria Harbour
This area is CauseWay_Bay

Figure 8. Browsing selected region

The map as shown above is not a scanned image of a map in the format of gif or jpeg . It is a vectorized map which converted from a DXF file (an AutoCad file format). The resulting map file is small in size, for example, the map of *Causeway Bay* is about 300K bytes in size(the original DXF file is 2M bytes in size). A special Java graphics engine embedded in the *applet* is used to interpret the digital map. The map can be displayed in different resolution according the request from user.

Therefore, the user will be able to zoom in further on the map using either the mouse drag or clicking on the zoom-in button placed at the left-hand side of the applet frame as shown in Figure 9. In addition, as the user moves around the map, a small bubble text box will appear to display the name of the building or facility nears by the mouse cursor.

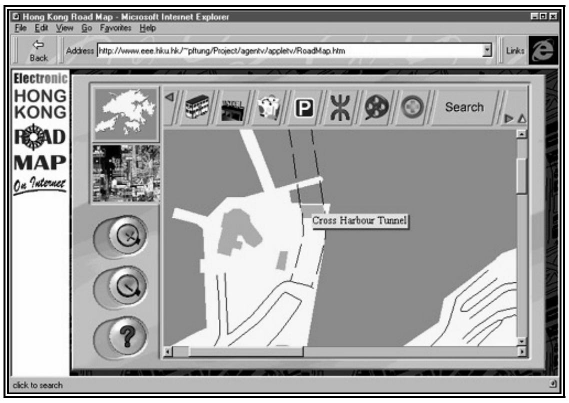
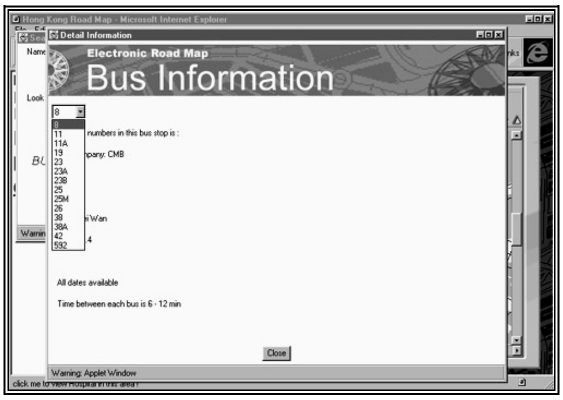

Hong Kong Road Map - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back
Address http://www.eee.hku.hk/~pftung/Project/agentv/appletv/RoadMap.htm
Links
Electronic HONG KONG ROAD MAP On Internet
Search
Cross Harbour Tunnel
click to search

Figure 9. Magnification on the digital map

A user is also able to retrieve further information about that building or facility by clicking on the area near the text box. A frame will be popped up which, for example, will show the full address, phone number and other information about that building. The actual information retrieval flow can refer to **4.1. Tourist Society - A Scenario** part of section 4 (Fig. 3). Moreover, a fast location searching interface is also provided. By clicking on any of the icons on the top panel of the applet (referring to Figure 9) information about a particular service will be retrieved and forwarded to the user. These icons represent the agents for buses, hotels, shopping centers, parking lots, mass transit railway (MTR), cinema theaters, and hospitals. After one service is selected, all the related spots managed by the agent will be highlighted on the current map.



numbers in this bus stop is :

pany: CMB

i Wan

All dates available

Time between each bus is 6 - 12 min

Figure 10. A sample bus time table

Besides, the user is also able to obtain detailed information in the usual way by clicking on the <u>Search</u> button located on the north panel of the applet. A frame will be popped up, and the user will be able to search through the IAs for information by specifying keywords or select from a default list supplied by the ANS. Figure 10 shows a sample information frame of a bus time table.

The power of the multi-agent based tourism kiosk lies on the fact that the users are able to interact with an autonomously agent society in a interactive and user-friendly way. The front-end interface is the only item that can be accessed by the users. The communications or exchange of messages among the agents are done at the back-end. Also, each agent has a simple knowledge base. If an agent cannot locate a piece of information, the agent can forward the request to the other agents. This approach provides a far better way on data update and integrity for real-time tourism information retrieval over Internet.

7. System Evaluation

The prototype of the multi-agent system has been implemented. We will conduct two experiments to test the usefulness and technical performance of the multi-agent system. The first experiment is to test the usefulness and users' satisfaction with the system. In the pilot study, we let two groups of students: graduate students and final-year undergraduate students to test the system and give us their comments and evaluation. We particularly ask the students to pay attention to the response time and other attributes which are important to users of the Internet applications. At the end of this year, we will formally introduce the system to the users of Internet to allow them from all over the world to access the system. By that time, we will conduct the experiment to collect data from users to compare the multi-agent based kiosk with other traditional tourist information centers or systems that developed with other information technologies.

The second experiment is a technical benchmarking. Besides the multi-agent approach, we will also develop a centralized Web database system, which is a pure client-server system. In such case, there will be no communication directly among the agents. There will be a set of functions, but not a set of agents. We would like to identify the differences between these two approaches in terms of, for example, response time and flexibility. We believe such technical comparison is important for us to understand the strength and limitations of the multi-agent approach.

There are significant overheads in the processing, interpretation and communication with the KQML language. Efficiency of such multi-agent system can also be limited by the traffics on the network and it is important to know how much such factors affect the performance and users' satisfaction with the multi-agent systems through experiments. We will also explore the other multi-agent languages, for example, IDEAS and Telescript.

8. Conclusion

In this paper, we proposed and have implemented a multi-agent based tourism kiosk for the users on the Internet to support their searching and retrieval of tourism information about Hong Kong. By using the multi-agent approach, most of the data maintenance is automatically done by the information agents at the local tourist offices. The users will always be able to access to the most update tourism information about Hong Kong. The kiosk would be especially useful for the users who are planning a trip to Hong Kong or who are visiting Hong Kong if they have access to the Internet.

References

[1] Burger F., Kroiß P., Pröll B., Richtsfeld R., Sighart H., Starck H., TIS@WEB-Database Supported Tourist Information on the Web. In Information and *Communication Technologies in Tourism 1997*, Springer, pp. 180-189, 1997.

[2] Castelfranchi C., Guarantees for autonomy in cognitive agent architecture . In *Intelligent Agents: Theories, Architectures, and Languages*, pp. 56-70.Springer-Verlag:Heidelberg, Germany.

[3] Etzioni O., Weld D., A softbot-based interface to the Internet. Communications of the ACM, Vol. 37, No. 7, pp. 72-79, July 1994

[4] Finin T., Labrou Y., Mayfield J., KQML as an agent communication language, University of Maryland Baltimore County , Baltimore MD USA.

[5] Harold R. E., *Java Network Programming*, O'Reilly & Associates, Inc., 1997

[6] Maes P., Agents that reduce Work and information overload. *Communications of the ACM*, Vol. 37, No. 7, pp. 30-40, July 1994

[7] Passmann C., Pipperger W., Prof. Dr. Schertler W., How to assess WWW-applications for tourism information systems form the end-user perspective. Methodical design and empirical evidence. In *Information and Communication Technologies in Tourism 1997*. Springer, pp. 208-220, 1997.

[8] Riecken, D., Intelligent Agents. *Communications of the ACM*, Vol. 37, No. 7, pp. 18-21, July 1994

[9] Rus D., Gray R., Kotz D., Autonomous and Adaptive agents that Gather Information .Dartmouth College, Hanover, NH 03755

[10] Wöber K. W., Services and Functions of European City Tourist Office, Report of the Research and Statistics Working Group of the Federation of European Cities' Tourist Offices (FECTO), Administration Board Meeting, Dublin, March 22nd 1996.

[11] Wooldridge M. J. and Jennings N., R., Agent Theories, Architectures, and Languages: A Survey. In M. J. Woodridge and N. R. Jennings, editors, *Intelligent Agents*. Springer-Verlag, Berlin, 1995.