Application of AI to the Design and Implementation of Airline Reservation System

Archana Mangal\textsuperscript{a}, Anoop Kumar Srivastava\textsuperscript{b}
\textsuperscript{a}Computer Science Department, Banasthali University, Rajasthan
Email: manarchana@gmail.com
\textsuperscript{b}Computer Science & Engineering Department, Institute of Engineering & Technology
M.I.A., North Extension, Alwar - 301030 Rajasthan, INDIA
Email: anoop.kumar.srivastava@yahoo.com

Abstract—In this paper, we present an application of Artificial Intelligence (AI) to the design and implementation of Airline Reservation System (ARS). We provide a multi autonomous agent based framework. Autonomous agents are intelligent enough and provide autonomy, simplicity of communication, computation, and a well developed semantics. The steps of design and implementation are discussed in depth, in particular the agent model, interaction pattern between agents, and ontology are given. We have developed mechanisms for coordination between agents using a language designed by us, which we call Virtual Enterprise Modeling Language (VEML). VEML is a dialect of Java and includes Knowledge Query and Manipulation Language (KQML) primitives. We have implemented a multi autonomous agent based system, called Airline Reservation System. We demonstrate efficacy of our system by discussing its salient features.

Index Terms—Multi Agent System(MAS), Knowledge Query and Manipulation language(KQML), Virtual Enterprise Modeling Language(VEML)

I. INTRODUCTION

Multi Agent Systems (MAS) are a growing field that has established itself after more than a decade of intensive research as a major field of computer science and artificial intelligence in particular. MAS can be best characterized as a software technology that is able to model and implement individual and social behaviour in distributed systems. It examines aspects of cooperation, coordination, negotiation, coalition formation, role assignment and self organization. A MAS can be defined as a loosely coupled network of problem solvers and interact to solve problems which are beyond the individual capabilities or knowledge of each problem solver [1].

MAS provides higher level of abstraction than traditional distributed programming; these abstractions are closer to user expectations that allow the designer more flexibility in determining system behaviour. MAS covers a variety of domain like distributed vehicle monitoring, manufacturing enterprise, air traffic control, space craft control, diagnosing faults in nuclear power plants, financial portfolio management, telecommunication, network control, transmission switching, service management, network management, supply chain man- agement, scheduling and control, material handling, holonic manufacturing system,

© Elsevier, 2013
concurrent engineering and collaborative engineering design [2]. In this paper, we have given the design and implementation of Airline Reservation System. In order to implement the coordination scheme among agents, a programming language is designed and developed, which is called Virtual Enterprise Modeling Language (VEML) [3]. Knowledge Query & Manipulation Language (KQML) which is based on speech act categories for describing protocols and agent communication strategies is used [4, 6]. In particular we have been able to identify a number of speech acts (and appropriate semantics) that cover a broad range of information services. In addition, we have used speech acts to cover negotiation: the process by which a set of agents come to terms on provision of information services and allocation of resources to the various service activities. The autonomous agents have been implemented in VEML [5]. This paper is organized as follows. Section 2 deals with motivation. Section 3 discusses related work. Section 4 describes the design. Section 5 discusses implementation. Section 6 presents salient features. Section 7 discusses conclusion.

II. MOTIVATION

As for intelligent agents in the travel industry, it is hoped that travel agencies will use the agents to obtain optimum packages for its customers, in lieu of the private user having to own the agent. This is of importance as large amount of travel data are presently set up for use by travel agents which the general public does not have access to. A very early version of an intelligent travel agent can be found at travelocity.com, but this is more of a large search engine with extensive filters that return the best possible solutions based on extensive inputs by the user, and most importantly, travelocity does not learn the habits of the user unless a registration form is filled out and updated on a regular basis.

The advantage of going to a system based on intelligent agent technology is its speed in problem solving. Basic features of the application will include the ability for a booking agent to perform the following:

- Book passengers on flights.
- Give passengers special needs or requests.
- Display information about flight and passengers.
- Display a seat layout map at appropriate times.
- Issuing boarding passes.

Drawbacks of existing system can be categorized into two forms like manual or automated system.

Manual System: The system is very time consuming and lazy. This system is more prone to error and sometimes the approach to various problems is unstructured.

Automated System: This system is used for automatically updates. But it needs an employee to do it.

Some features that multi agent system will help the customers:

- Interaction: ability to communicate with environment and other agent.
- Cooperation: ability to interact with other agent to achieve a common purpose
- Competition: ability to interact with other agent that the success of one agent implies failure of other agent.
- Organization: ability to relationship between agent and the product.
- Sociality: ability to participate in multiple relationships.

III. RELATED WORK

“Sardine” is an agent based system for Airline ticket bidding system. In this system there are two agents, one is buyer agent and the other is airline agent. These agents coordinate properly to handle buyers request properly. This system is developed in Java Servelets [15]. Neural Network can also be implemented to act like an intelligent agent system for Airline reservation system. It must be trained and tested properly to recognize customer request and categorize them efficiently. They present a research that suggests that the most significant barriers to online booking can be eliminated or reduced with a software agent approach. They present the structure, behaviour and the operating environment of a multi channel software agent application, FareTracer, which has been designed to assist consumers in making their own travel reservation on the web. Based on a dynamic modular design, the FareTracer can scan any number of predefined and ill-structured data sources on the web to pick out and retrieve only the essential information in a matter of seconds/minutes.

A quick search on the Internet will reveal a host of Web sites where people can find, buy, and sell goods [12]. None of these sites use autonomous agents to negotiate on behalf of their owners. However
there is an agent mediated system developed by MIT Media Laboratory which helps users to automate the process of buying and selling [11, 14]. This has certain limitations such as: (a) it has only synchronous communication, (b) it provides only local agents, (c) third party creation of agents is not possible, and (d) it is based on request-response protocol. These limitations have been tried to satisfy in our implementation using VEML. The closest system to this is MAGMA although there are significant differences. For example, MAGMA attempts to completely model the real world, representing things such as banks, account, currency, rainchecks etc. The approach is to add only as much infrastructure as necessary for agents to negotiate and complete transactions. This allows the system to be integrated into already existing transaction-based systems such as real vendors and financial institutions. There has also been substantial research and analysis of market-based systems. The typical approach is to model an optimization problem (e.g., resource allocation) as a marketplace consisting of multiple agents each trying to maximize its own “utility”. This differs from the traditional AI approach, in that agents are now competing marketplace systems has been shown to be an effective technique. Examples of such systems include Challenger, a system for allocating processing resources within a network of workstations, and Michael Wellman’s WALRAS, an algorithm used to calculate competitive equilibria where agents submit single good demand functions to market clearing auctions. Multi-agent electronic marketplace is similar to these market-based systems in that it attempts to efficiently and optimally match buyers and sellers [10, 12, 13].

IV. DESIGN OF AIRLINE RESERVATION SYSTEM (ARS)

The design of ARS is done and agent model, interaction pattern between agents, ontology, and VEML design is pro-vided.

A. Agent Model and Interaction Pattern

This example involves three agents: a passenger agent P, an airline clerk C, and C’s supervisor S. P, S, and C are all computer programs [9]. Note that by confirming a reservation, the airline enters into a commitment to issue a boarding pass to the passenger at the appropriate time. Again, the following exchange involves requesting (a query is a request to inform), informing, permitting, and denying as well as mental components such as knowledge and commitment. March,
P to C: Please inform me what flights you have from San Francisco to New York on April 18.
C to P: Flight 354 departs at 8.30, flight 293 departs at 10.00, flight 441 departs at noon.
P to C: Please book me on 354. C to P: That is sold out.
P to C: Please book me on 293.
C to P: That is confirmed; your reservation number is 112358. P to C: Please book me also on 441.
C to P: That conflicts with 293; I am not allowed to double book a passenger.
P to C: Please get permission to do so.
C to S: I request permission for the following double booking. S to C: Permission denied.
C to P: Sorry, I can not get approval. April 18, at the airport
P to C: My name is P; I have reservation for flight 293. C to P: Here is your boarding pass.

Figure 1 represents agent model of airline reservation system. Table 1 shows interaction summary among agents.

B. Ontology

AIR RESERVATION: is a set of inter-related UNITS which are grouped together to ACHIEVE some common PURPOSE.

![Figure 1. An agent model of airline reservation system](image-url)
<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Message #</th>
<th>Message Content</th>
<th>Performativ</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>C</td>
<td>1</td>
<td>Flight from San Francisco to New York.</td>
<td>ask one reply</td>
</tr>
<tr>
<td>P</td>
<td>C</td>
<td>3</td>
<td>Book me on 354</td>
<td>request</td>
</tr>
<tr>
<td>C</td>
<td>P</td>
<td>4</td>
<td>sold out</td>
<td>reply</td>
</tr>
<tr>
<td>P</td>
<td>C</td>
<td>5</td>
<td>Book me on 293</td>
<td>request</td>
</tr>
<tr>
<td>C</td>
<td>P</td>
<td>6</td>
<td>confirmed</td>
<td>reply</td>
</tr>
<tr>
<td>P</td>
<td>C</td>
<td>7</td>
<td>Book me also on 441</td>
<td>request</td>
</tr>
<tr>
<td>C</td>
<td>P</td>
<td>8</td>
<td>conflict with 291, not allowed</td>
<td>reply</td>
</tr>
<tr>
<td>P</td>
<td>C</td>
<td>9</td>
<td>get permission</td>
<td>request</td>
</tr>
<tr>
<td>C</td>
<td>S</td>
<td>10</td>
<td>permission for double booking</td>
<td>request</td>
</tr>
<tr>
<td>S</td>
<td>C</td>
<td>11</td>
<td>permission denied</td>
<td>reply</td>
</tr>
<tr>
<td>C</td>
<td>P</td>
<td>12</td>
<td>sorry</td>
<td>reply</td>
</tr>
<tr>
<td>P</td>
<td>C</td>
<td>13</td>
<td>confirmation from passenger</td>
<td>request</td>
</tr>
<tr>
<td>C</td>
<td>P</td>
<td>14</td>
<td>boarding pass</td>
<td>reply</td>
</tr>
</tbody>
</table>

UNIT: is an entity for managing the activities to Achieve one or more PURPOSE. A UNIT may be booking request process, booking approval process and a clerk or facilitator. PASSENGER AGENT: is an entity for the PURPOSE of getting the information about flights on particular date and subsequently request for booking.

SUPERVISOR AGENT: is an entity for the PURPOSE of providing approval for booking a particular flight.

CLERK AGENT OR FACILITATOR: is an entity for the PURPOSE of providing facilities to both passenger agent and supervisor agent.

BOARDING PASS: is an item which is given to the passenger to travel via flight.

TIKTET REQUEST: is a statement defining a passenger’s requirement in terms of detail of flight like date and time of arrival and departure, boarding pass.

AGENT: is an entity for performing the activities to achieve the PURPOSE of buying, selling and facilitating.

REGISTRATION: is a process by which passenger agent express his/her needs for particular flight.

DEREGISTRATION: is a process by which passenger agent cancels his/her reservation for particular flight.

TIME LIMIT: is a period during which tickets are due to be purchased or cancelled.

NEGOTIATION: is a method by which passenger agent and clerk agent will communicate with each other.

C. Virtual Enterprise Modeling Language (VEML)

VEML is a dialect of the Java programming language directly enabling software agent oriented programming and development of ARS [3]. VEML includes KQML primitives. KQML is a language and an associated protocol to support the high level communication between autonomous agents [8]. KQML is an abstraction, a collection of primitives plus the assumption of a simple model for inter-agent communication. There is no such thing as an implementation of KQML in the sense that KQML is not a compiled language. VEML is a compiled language. The VEML grammar includes Java grammar with additional keywords and statements. VEML is a tool to build software applications that dynamically interact and communicate with their immediate environment (user, local resources and computer system) and/or the world, in an autonomous (or semi-autonomous), task oriented fashion. VEML simplifies programming on the Internet by providing synchronous and asynchronous communication among agents with the help of message passing. Universal naming is possible in VEML. VEML supports communication among multiple inter-generating computations/nodes. It is a language for programs to use to communicate attitudes about information, such as querying, stating, achieving, believing, requiring, subscribing and offering.
D. VEML as a Framework of MAS

There are some abstract requirements that are part of the VEML model.

- An agent sends and receives messages, called perfor- matives, that access the knowledge (knowledge base, knowledge store or knowledge content) of the agent.
- Agents may interact asynchronously with more than one other agent at the same time.
- The particular network protocol used (SMTP, TCP/IP, HTTP etc.), i.e. the transport mechanism for the perfor- matives, is not part of the requirements. We particularly encourage implementations that allow for multiple trans- port mechanisms.
- Agents are known to one another by their symbolic names, rather than their IP addresses.
- For agents that reside in the same network domain there exists at least one agent with a special status, a facilitator, that is particularly suited to process the so-called networking and facilitation performatives.

VEML supports networking and facilitation performatives to have the implementation of all the above mentioned approaches.

V. IMPLEMENTATION

To test the validity of what has been described, a Airline Reservation System is implemented, which demonstrates air-line reservation.

A. Agent Programming

A source file containing agent program is written in VEML which must have a “.veml” extension. This program in VEML is preprocessed into Java source code, using a VEML compiler written by us [7]. The generated Java source code uses a library for agents developed in Java. After preprocessing VEML code, any Java compiler can be used to produce the bytecodes for the program, which can then be executed on top of any Java Virtual Machine (JVM) implementation.

B. Syntax of VEML Program

Following is the sample program in VEML. VEML programs are grouped into related agent programs, which are called modules. A module can contain several agent interfaces and programs. A program may extend another program (single inheritance). A VEML program can implement zero or more interfaces (multiple interface inheritance). The inheritance hierarchy is similar to Java, where every class extends a top-level object class.

module Veml;
program Agent; begin // Body of the program Agent void main() begin // Body of the main method end end

C. Methods of Coordination between Agents

Since all agent communication is via synchronous and asynchronous message passing, there is a need to have the methods for coordination purpose.

msg.send()

The send method is used by the sender agent to send message to the receiver agent. The address of the receiver agent is specified in the message. This method sends a message to the receiver agent and does not expect a reply.

msg.setReplyWith()

The setReplyWith method is used by the sender agent to send a message to the receiver agent and indicates to the receiver that the sending agent expects a reply. The receiver agent need not to give the reply. (Note: to receive the expected reply, the receiving agent must register a handler. This mode of operation is termed as “asynchronous” send). msg.sendAndAwaitReply(String performative, String contents, String receiver)

The sendAndAwaitReply method is used by the sender agent to send the message to the receiver agent and wait for reply. This method performs a “synchronous” send as opposed to an “asynchronous” send.

sendMsgAndAwaitReply()

The sendMsgAndAwaitReply method creates, sends and re-ceives a message in one call.

getSender()

The getSender method is used to get the sender agent’s name.

getPerformative()

The getPerformative method is used to get the performative used for sending message.
getContents()
The getContents method is used to get the contents of the message sent.
getReceiver()
The getReceiver method is used to get the receiver agent’s name.
registerHandler(String performative, Handler handler) The registerHandler method is used to register the handler (e.g. ask-one, reply etc.).
getMessage()
The getMessage method is used to get the message.
agent.shutdown()
The shutdown method is called by the agent to shutdown itself and the agent will be deregistered from the Agent Name Server (ANS).
doClient()
doClient method is used by the client agent to send the content of the message to the server agent and wait for the reply.
doServer()
doServer method is used by the server agent to start the server process and receive the content of the message from the client agent.

D. Illustrations of Message Sending and Receiving in ARS

1) Passenger agent P sends query to airline clerk agent C.
   println("agent" + " " + msg.getSender() + " " + "sending" + msg.getPerformative() + " " + "with query" + msg.getContents() + "to agent" + msg.getReceiver());
   **Output:** agent passenger P sending ask-one with query what flights you have from San Francisco to New York on April 18 to agent clerk C.

2) Agent clerk C received query from agent passenger P.
   println(msg.getPerformative() + "handler of agent" + msg.getReceiver() + "received query from agent" + msg.getSender() + "=>")
   + msg.getContents());
   **Output:** ask-one handler of agent clerk C received query from agent passenger P => what flights you have from San Francisco to New York on April 18.

3) Agent clerk C sends reply to agent passenger P.
   println("received reply from agent" + inMsg.getSender() + "by agent" + inMsg.getReceiver() + "=> Performative = " + inMsg.getPerformative() + "the reply received from agent" + inMsg.getSender() + "is as given below ...");
   **Output:** received reply from agent clerk C by agent passenger P =>
   Performative = reply the reply received from agent clerk C is as given below ...
   list of flights with their schedule

4) Passenger agent P received reply from agent clerk C.
   println(msg.getPerformative() + "handler of agent" + msg.getReceiver() + "received reply from agent" + msg.getSender() + "="
   + msg.getContents());
   **Output:** reply handler of agent passenger P received reply from agent clerk C => list of flights with schedule

5) Passenger agent P sending query to agent clerk C.
   println("agent" + " " +
   msg.getSender() + " " + "sending" +
   msg.getPerformative() + " " + "with query" + msg.getContents() + "to agent" + msg.getReceiver());
   **Output:** agent passenger P sending ask-one with query
   Please book me on 354 to agent clerk C.

6) Agent clerk C receives query from agent passenger P.
   println(msg.getPerformative())
8) Agent clerk C gives reply to agent passenger P.
   println("received reply from agent" + inMsg.getPerformative() + "by agent" + inMsg.getReceiver() + ", " + inMsg.getContents());
   Output: received reply from agent clerk C by agent aclient $
   Performative = reply
   the reply received from agent clerk C is as given below
   ...
   That is sold out

E. Architecture

Figure 2. Shows the architecture of Airline Reservation System, which comprises the following components: (a) Front-end: a GUI handles all of the user interactions. The front end is implemented in VEML. Front end is a screen shot of ARS showing the form, which user has to fill for different purposes. The user has to place the order in the format provided in the form. (b) Back-end: an ARS engine where the agents actually “live” and interact with one another. The back end is implemented in VEML and KQML. The back-end implements a request response service: a client sends a request to the back-end, the back-end services that request. The front-end and back-end communicates with one other via TCP/IP sockets.

Figure 3 and 4 shows screen shots of Airline Reservation System.

![Architecture of Airline Reservation System](image1)

![Screen shot of Main Frame in Airline Reservation System](image2)
V I. Salient Features

The salient features of Airline Reservation System are as follows:

A. Naming, Security, and Direct Connection

Agents can have any number of logical names that don’t contain the hostname. There is no way for an agent to “overhear” a conversation between two other agents. This provides security. Agents can establish direct connections with each other for bulk data transfer.

B. Agent Creation and Run

Each agent runs within an individual Java thread. Each agent gets registered with the ARS facilitator. The facilitator maintains registry of all agents. Since agents are Java threads, which are light weight processes, a large number of threads can be created. There is no limitation on the number of agents.

C. Third Party Creation of Agents

We have used protocols and semantics for defining more extensible agent communications. “This protocol is open to allow third-party agents (with their own unique strategies) to participate in the agent coordination”. We envision developers creating sophisticated commerce agents that require a non-trivial amount of resources to complete it’s market analysis. Such agents could potentially be run more efficiently on a user’s local machine and communicate/negotiate with other agents via open protocol. Agents can be developed in any language e.g. C, Lisp, Prolog, Java and VEML. This system supports protocol like TCP/IP, SMTP, and FTP.

D. Routers

The router in this system provides an easy-to-use link between application and network viz. (1) Routers are a content independent message routers, (2) All routers are identical, just an executing copy of the same program, (3) A router handles all messages going to and from it’s associated agent i.e. each agent has it’s own separate router process. Thus it is not necessary to make extensive changes in the program’s internal organisation to allow it to asynchronously receive messages from a variety of independent sources, (4) The router provides this service for the agent and provides the agent with a single contact point for the communication with the rest of the network, (5) It provides both client and server functions for the application and can manage multiple simultaneous connections with other agents, (6) Routers relies solely on its performatives and arguments, (7) A router directs a message to a particular Internet address as specified by the message, (8) When an application exits, the router sends another message to the facilitator, removing the application from the Facilitator’s database, and (9) Routers can be implemented with varying degree of sophistication although they can guarantee to deliver all messages.


**E. Facilitator**

The facilitator agent of this system performs following useful services: (1) Maintain a registry of service names, (2) Forward message to named services, (3) Routes messages based on the content, (4) Provides matchmaking between information providers and clients, (5) Provides mediation and translation services, (6) This is a simple software agent which maintains the database of active agents, (7) This provides the registry of agent names and addresses, (8) It is used by all the agents as they arise to register their names and addresses and to subsequently locate other agents to which messages are to be sent, and (9) It accepts register and unregister performatives to maintain it’s database and responds to ask-one and ask-all etc.

**F. Scalability**

When an agent is created, it acquires a Java thread. So, depending on the number of agents created, it acquires that many number of threads. These threads are software programs. Each thread defines a separate path for execution. Multi tasking threads require less overhead than multi tasking processes. Processes are heavyweight tasks that require their own separate address spaces. Interprocess communication is expensive and limited. Context switching from one process to another is also costly. Threads on the other hand are lightweight. They share the same address space and cooperatively they share the same heavyweight process. Interthread communication is inexpensive and context switching from one thread to the next is low cost.

Multi threading enables to write a very efficient programs that make maximum use of the CPU because idle time can be kept to a minimum. Java manages threads and makes multi threading especially more convenient, because many of the details are handled by the programmer. Hence the issue in this case is not the processes occupied by agents in the operating system rather it is the memory. Each agent or thread consumes memory from the operating system. So a large number of agents will have large memory requirement.

**G. Performance and Efficiency**

The system performance guarantees to agents so that the agents can meet real time constraints. The parameters of argument are time to complete the process, cost factor, band- width requirement and transfer of code. This system is a single unified framework with message passing in which wide range of distributed applications can be implemented efficiently and easily. This system uses a network service by remotely invoking its operations. The results are transmitted back over the network to the sending agent which processes them depending on the result. The service and client platforms have to provide an infrastructure to support agents. Messages can be transmitted from one agent of a machine to the other agent on the same machine or the other machine in a heterogeneous network. The station based agents system helps to reduce communication costs, bandwidth requirements, and improves performance by becoming more efficient. This consumes less memory. In a system like Airline Reservation System the impact is much more with respect to time.

**VII. Conclusion**

This paper presents an application of AI to the design and implementation of Airline Reservation System which is a multi autonomous agent based system. We have provided agent model, interaction pattern between agents and an ontology. We have developed an agent oriented modeling language and have provided the methods used for working of the agents. Finally, the architecture and salient features of Airline Reservation System are provided. Here our focus is in supporting the best communication, coordination and problem solving mechanism available with minimum programming effort on the developer’s side. Hence this architecture can be directly applied in practice. As all the facts surrounding software agents recommends, Airline Reservation System is just one application for a new generation of automated information services. AI has a large role to play in improving the gen- erality, robustness and overall competence of these services. We believe that the field of AI has an equally important role to play in architectural infrastructure (in concert with other technologies, of course).

**REFERENCES**


119


