A RESTAURANT FINDER USING BELIEF-DESIRE-INTENTION AGENT MODEL AND JAVA TECHNOLOGY

Dongqing Lin
Department of Computer Science
University of North Dakota
Grand Forks, ND 58202
(701) 777-3477
dlin@cs.und.edu

Thomas P. Wiggen
Department of Computer Science
University of North Dakota
Grand Forks, ND 58202
(701) 777-3477
wiggen@cs.und.edu

Chang-Hyun Jo
Department of Computer Science
California State University, Fullerton
Fullerton, CA 92834
(714) 278-7255
jo@ecs.fullerton.edu

ABSTRACT
It is becoming more important to design systems capable of performing high-level management and control tasks in interactive dynamic environments. At the same time, it is difficult to develop and maintain such complex systems with our traditional software techniques. The agent-oriented/based systems, rooted in a different view of computational entities, offer prospects for a qualitative change in this aspect. In this work, we adopt the basic architecture of a Belief-Desire-Intention (BDI) agent model and develop a more intelligent and dynamic system by using agent programming. Applying BDI concepts, we will implement an experimental framework in our Restaurant Finder system. The agent will possess learning behavior based on the user's feedback and the principle of inferring preferences. In addition, the agent will also update its knowledge dynamically based on analysis of user's interaction with the system. Our Restaurant Finder system is designed for use on mobile systems. This system has been constructed and implemented based on a three-tier architecture. The Restaurant Finder system will demonstrate an example of BDI agent programming and the J2ME Mobile Information Device Profile (MIDP) client application design.

Keywords
Belief-Desire-Intention (BDI) Agent-Based Programming

1. INTRODUCTION
In recent years, it's often required that new software systems be capable of working with a complex and uncertain world. These systems need to process imperfect and limited information from a substantially changing environment. Efficient and flexible computer architectures and languages that reduce the complexity and time for system specification and modification have been developed to meet these demands [7]. Among them, the agent-based modeling techniques provide an effective architecture to modularize the sophisticated systems and solutions to handle the interaction between the systems and the environment dynamically.

--
ISCA CATA-2003
March 26-28, 2003, Honolulu, Hawaii

Agent computing is based on agents. An agent is a concurrent, autonomous, intelligent, and self-contained object [8]. The agent model based on Belief-Desire-Intention (BDI) has been proved as a powerful computing technique [2]. The Belief, Desire, and Intention represent the essential parts of the state, which best describes the systems and the environment. A Belief represents knowledge of the world, i.e. the agent itself and its varying local environment. A Desire is the goal to achieve or event to handle, which can be the value of a variable or a symbolic expression. Applying this Desire concept leads to desire-oriented computing instead of conventional task-oriented computing. A desire includes relevant tasks to be completed. Desire-oriented computing focuses on what to do instead of how to do as defined in task-oriented computing. An Intention is a set of plans to realize the predefined goals or react to a specific situation.

One interesting BDI agent system called JACK has extended the Java programming language in agent modeling [1]. JACK has Class, Interface, Method, Syntactic and Semantic extensions of Java implemented as Java plug-ins to support an agent-oriented development environment. JACK also uses the Database class to define and manipulate its beliefs. Another Mobile BDI Agent Toolkit applies Java packages to provide a runtime environment for BDI model programming [3]. An application of BDI agent modeling, Agent-based Stock Trader shows how to design and implement the BDI concepts using Java classes and Microsoft Access database system [6]. Meanwhile a new agent-based programming language, APL, has recently been proposed and developed. An APL prototype is to translate the APL source into the Java source codes executable on the Java Virtual Machine [9].

Applying BDI concepts, we have implemented an experimental framework for our Restaurant Finder system. However, there have been no proper programming languages to well support agent programming naturally. So in this work, we try to enhance the capability of object-oriented languages to build this agent system. The belief, desire and intention are initially set up as separate classes, which are integrated as basic units for a BDI agent. The dynamic mapping from the specific Belief to an Intention plan is realized by maintaining a mapping table. In addition, the agent possesses learning behavior based on the user's feedback and the principle of inferring preferences. The agent also updates its knowledge dynamically to make it more intelligent by analyzing the user's interaction with the system. The dramatic increase in the use and availability of mobile devices has resulted in the ability to access information anytime and anywhere. Our Restaurant Finder system is built for use on mobile phones.

2. CONCEPT AND DESIGN

2.1 BDI Agent Model

The BDI agent modeling mechanism has been argued as a flexible achitecture for building an intelligent goal-driven agent [10]. One purpose of this work is to show that the BDI agent model can be used to build a system that can learn and adapt its behavior according to accumulated experiences and changing environment. The user's choices, restaurant information, and local environments will be recorded as basic elements of Belief. Initially the criteria will be set up for choosing a most appropriate restaurant based on the user's preference and current situation. The plans, i.e., Intention, will be called to meet the criteria and achieve the goal. When a user searches the restaurants, various plans that reflect the current and local information will be utilized to prepare a list of restaurant candidates at run-time. Then the system can make comparisons and give the user a final recommendation based on this itemized group of restaurants. Here the agent model has the exact Desire to achieve. It will let users select the restaurant quickly and satisfactorily, and will re-build the information database for the agent model each time the user makes a choice. This Restaurant Finder application also demonstrates the computability of the BDI model in constructing a real-world system. The concepts of BDI can be effectively designed and implemented in a way similar to the normal practice applied in unified process of software engineering nowadays [5, 8].

2.2 Architecture Design

A three-tier design has been applied to the Restaurant Finder as in Figure 1.

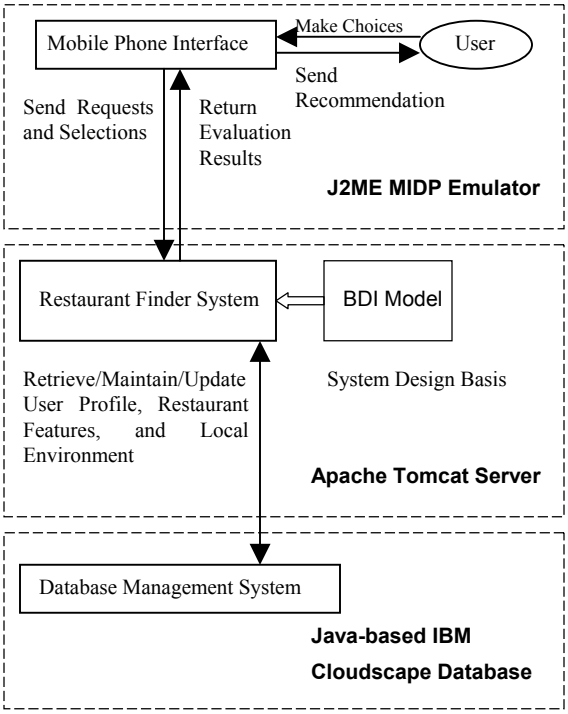

Figure 1. Three-tier Architecture of Restaurant Finder System.

In the Restaurant Finder system, the first layer is the client part (i.e., the user interface). The J2ME™ (Java™ 2 Micro Edition) MIDP (Mobile Information Device Profile) Emulator will be used to simulate the mobile phone screen that acts as the J2ME client. J2ME™ is Sun's newest Java platform for developing applications for various consumer devices, such as set-top boxes, embedded systems, mobile phones and cell pagers. MIDP is a set of APIs that allows developers to handle mobile-device-specific issues, such as creating user interfaces, storing information locally and networking. The second layer is the Web server part, the servlet engine for the Restaurant Finder system, implemented using Apache Tomcat Server, that processes the client's selections, updates the system database, and sends back the recommendation. This server is the Java servlet container to handle the user's Get/Post request and deliver a restaurant candidates screen or a restaurant recommendation screen to the user. The third layer is the information tier, a database system using the Cloudscape relational database, which is a pure-Java database management system using IBM Cloudscape [4]. This information storage is also a server for the relational database management system. The interface will direct the user request to be processed at the corresponding BDI agent built using Java Servlet technology. The response, i.e., the restaurant recommendation from the system, or the initial restaurant candidate list from the user profile, will be shown through the same user interface too. The BDI agent-based system will retrieve, maintain, and update the database system. The main contents of the database are user profile, restaurant features, and local environment.

Each time the user logs in, the information about the local environment and restaurants will be provided according to the location and the time that the user uses the mobile phone. The Restaurant Finder system will first propose to the user a predefined candidate list of restaurants based on current conditions and user preferences recorded in the database. The user then can either choose a restaurant from the list or make new selections from a menu to be shown on the mobile phone screen. The system will make an optimum choice to meet the user's goal by selecting the most appropriate restaurant from the database. Meanwhile the system will also update the database system to reflect the user's new preferences based on his/her current choices. This can also be used to update the criteria in the database system to be used as the plans for building future restaurant candidate lists. In this way, the system is able to remember user preferences and provide a more recent candidate list next time the user logs in. Thus the system is designed with a dynamic learning ability in the run-time environment by manipulating the database system. The basic reasoning mechanism is to assign a specific weight value to each element in the database and calculate the total amount for each possible selection of a restaurant. The weight values are also subject to change after each selection made by the user.

3. IMPLEMENTATION OF THE SYSTEM

3.1 Core of BDI Agent

Although Java does not support directly BDI agents in our case, Java has suitable class structures that can be used to build the basic units, like Belief, Desire, and Intention in an agent model. For the time being, a relational database system is chosen to support the Java BDI-agent programming. Although the class structures of BDI concepts cannot be changed at run-time, the information in the database can be updated dynamically according

to the changing environment. The information system will be re-built every time a new restaurant candidate list is generated for the user using Java Servlet technology. The learning and reasoning ability of the BDI agent will be realized dynamically with the aid of the database system. Using Structured Query Language (SQL), the Belief elements will be stored, retrieved, and updated in several tables of the database system. Similarly, criteria used to suggest restaurants will also be stored and updated in an Intention mapping table. The restaurant candidate list will be constructed by retrieving these restaurant items to meet user's requests for different environments.

For this application, the main items about the restaurant feature are type, price, location and parking which are established in the Belief table of the database. Each element for these items will be assigned a weight value from 1 to 10, representing the quality or grade for each element. The higher value means a restaurant closely satisfies user's preference according to this item. These values will be updated each time the user makes different choices and a final decision. Recommendations will be made by the system according to the comparison of total weight values of available restaurants in terms of these data items. The system will advise the user dynamically by changing the weight values each time user makes new choices. Even in the user interface part that is the mobile phone screen, the items shown on the menu each time will also be different based on the current situation and the user's former selections. The restaurant candidate list will be built and updated each time the user starts the system. Based on previous information, an initial candidate list will be shown on the mobile screen when the user starts the system. This will accelerate the process to let the user make a decision in an acceptable or short time. According to the current Belief items and the user's preference, this list will be different and customized each time the user logs in. After that, a dynamic menu will be shown on the screen for the user to make new choices if the user has a different or new idea than the recommended restaurants in the candidate list. By calculating the total weight values of all the concerned Belief elements, a new optimum restaurant candidate list will be provided to the user.

3.2 Structure of BDI Agent

For this Restaurant Finder system, the basic data structures for BDI have been developed based on the relational database table. The Desire is clear, which is to suggest an appropriate restaurant to satisfy user's requirement. The contents for the Belief are maintained in two tables such as Restaurant (Information about available restaurants around) and Profile (User's profile for selecting restaurants).

The Restaurant table represents the current local situation in which the user starts the Finder system. It will be generated dynamically and will simulate the real environment for the restaurants in the surrounding area. Each time the user logs in, the process of selecting restaurants will be performed according to varying conditions. The fields of the Restaurant table include ID, Name, Type, Price, Location, Parking, and Address.

The core part of the Belief is the user Profile table, which provides permanent storage for the user's history list of selected restaurants. Initially, an empty table only with meta-data schema is designed for the first time the user starts the Finder system. Thereafter, the information about the user's selections will be recorded in the table after the user successfully makes decisions. The schema of the Profile table is the same as that of the Restaurant table.

The difficult but interesting part is how to implement the Intention dynamically. The user Profile table is the basis for making reasonable recommendation to the user. Each time the user logs in, the system will first analyze the user's previous selections with available local restaurants. Then the system will provide the user a restaurant candidate list that reflects the user's previous selections. If the user has a new idea, the system will also be able to provide a customized shortcut menu for the user to choose. The analysis is based on the frequency of restaurants appearing in the Profile table. Here a Java two-dimensional array will be used to construct such a candidate list first. All the information about a specific selection can be included in this array easily. The array size will be determined dynamically based on the Profile table information when the user logs in. After the user selects a restaurant, all the pertinent information will be added to the user Profile table for future reference. Note that the role of the Intention is to provide alternative plan(s) to use in finding optimal solution for the user. The plans are chosen according to user's interests that may vary every time using the Finder system. These interests are itemized as different criteria in building plans. These plans can also be combined to make an optimal suggestion to the user. This process reflects a dynamic mapping from Belief to Intention with the help of database system. An Intention mapping table is built to simulate the dynamic mapping in this application.

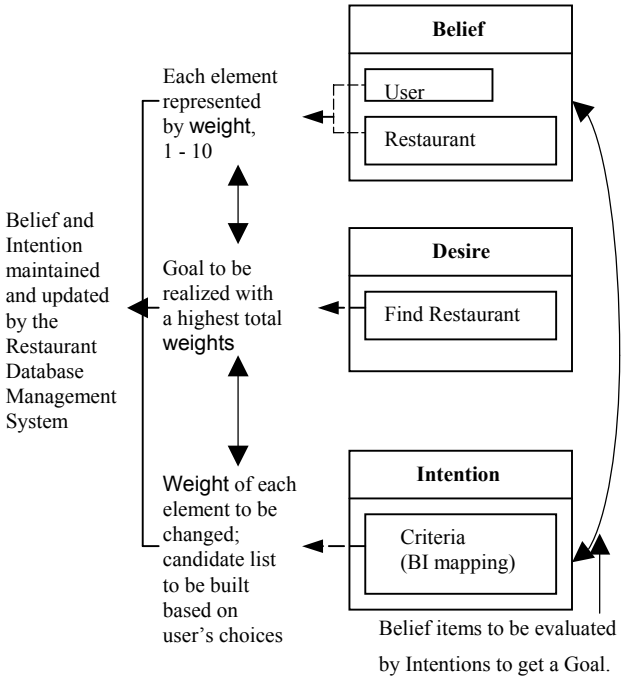

Figure 2. Basic Structure of BDI Agent-based Restaurant Finder

For one instance, the Intention class is implemented to provide information for use by the BDI agent in this application. Based on user's selection, the pertinent restaurant type, price, location and parking will be converted into corresponding integer numbers from 1 to 10. The agent will sum these numbers up for each restaurant in the local area. The restaurant with highest total

weight value will be shown in the mobile phone interface. The process implemented by the Intention will be controlled dynamically by using an Intention mapping table. In the IntentionMap table, iName, iType, iPrice, iLocation and iParking are the base values to be used by the Intention class. The iNum is a control number to define which set of data in the IntentionMap table will be applied this time. By giving different value of iNum, the Intention class will be able to determine which plan is to apply for the current case. In such a way, the BDI agent would probably provide different recommendations to the user even for the same local environment.

Modular design can be easily applied to the BDI agent-based system. First the Belief, Desire and Intention will be built based on Java classes. Then these pre-defined structures will be assembled to build an agent. The basic structure of BDI agent-based Restaurant Finder system is shown in Figure 2.

4. EXPERIMENTAL RESULTS

The Restaurant Finder system has been tested by applying a simulated local environment. Figure 3 shows only a few snapshots for a series of screens that are provided from the system.

Figure 3. Type Screen and Recommendation.

It is assumed that the user will start the system in an area with local restaurants. The pertinent information about these restaurants includes name, type, price, location and parking condition of the restaurants. The addresses of the restaurants are stored in the Restaurant table in the database. In processing the restaurant information, each data item will be assigned a weight value according to the Intention plan selected by the system and user's choices. Then the total weight values will be compared by the system and a most appropriate restaurant will be provided to the user. In the meantime, the system will record the user's choices and update the user's Profile table.

On the Welcome screen of the Restaurant Finder system, when the Select button is clicked, it will direct an initial message to the Candidate screen. If this is the first time to use the system, there is no record in the user Profile table. The screen shows a message "No restaurant candidates yet!". The user needs to input his/her preferences and let the system recommend a restaurant.

For example, a user may like a restaurant with Italian food and medium prices, but may not care about the location and parking. By comparing the available restaurants, for example, Pizza Hut may be selected by considering prices. If the user changes his/her idea, he/she can simply click the Select button to continue the same process again. This time another restaurant is recommended. In the meantime, the user's selection will be written to the Profile table, which reflects the user's most recent preference.

When the user starts the system with previous records already in the system, at the Candidate screen, the system will give the user a candidate list of at most three restaurants. These restaurants are those the user selected lately or the most frequently selected. The candidate list will be updated each time the user makes choices.

5. CONCLUSIONS AND FUTURE WORK

Agent-based computing has been emerging as a major programming paradigm for the future. The Belief-Desire-Intention (BDI) agent modeling provides an efficient technique to build complex and intelligent system with its ability to learn and adapt and its characteristic for modular design. However, there are not many practical applications based on the BDI-agent concept developed yet. In this work we present an experimental framework for a Restaurant Finder system using BDI model and Java technology. We demonstrate how to design and implement the agent-based system in Java with the help of database system. The system is able to learn from previous experiences and adapt to the changing environment. The merit of this work is to show how to design a real-world application based on the BDI-agent model. The noble idea of a belief-intention mapping table allows us to make the system reflective and intelligent by learning and adaptation.

There is a great risk that run-time dynamic mapping will affect the performance of the system. This is still an important issue for the BDI agent-based modeling. Java does not support directly run-time knowledge management or function implementation. It is a big challenge to handle how to update or add the Intention plans at run-time without affecting the system. If this can be done, the advantages of BDI agent-based modeling will be fully realized. This is part of the work for agent modeling to be completed in the future.

6. REFERENCES

[1] Agent Oriented Software Pty. Ltd., *JACK Intelligent Agents User Guide*, URL = www.agent-software.com.au, 1999.

[2] Bratman, Michael E., *Intention, Plans, and Practical Reason*, Harvard University Press, Cambridge, MA, 1987.

[3] Busetta, Paolo and Ramamohanarao, Kotagiri, *an Architecture for Mobil BDI Agent*, Mobile Computing Track, ACM SAC' 98, 1998.

[4] Deitel, Paul J., Harvey M., Deitel, Santry, Sean E., *Advanced JavaTM 2 Platform – How to Program*, Prentice Hall, Upper Saddle River, NJ, 463, 531-533, 543-549, 718-739, 755-784, 2002.

[5] Einhorn M. Jeffery and Jo, Chang-Hyun, A BDI Agent Software Development Process, University of North Dakota, 2002.

[6] Feng, Xin and Jo, Chang-Hyun, *Agent-based Stock Trader*, Dept. of Computer Science, University of North Dakota, 2002.

[7] Georgeff, Michael, Pell, Barney, Pollack, Martha, Tambe, Milind, and Wooldridge, Michael, *The Belief-Desire-Intention Model of Agency* (1999), Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98), URL =citeseer.nj.nec.com/georgeff99beliefdesireintention.html.

[8] Jo, Chang-Hyun, *A Seamless Approach to the Agent Development*, ACM SAC 2001, Las Vegas, NV, 641-647, 2001.

[9] Jo, Chang-Hyun and Arnold, Allen J., *the Agent-based Programming Language: APL*, ACM SAC 2002, Madrid, Spain, 27-31, 2002.

[10] Rao, Anand S. and Georgeff, Michael P., *BDI Agents: From Theory to Practice*, Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, USA, June 1995.