# Agent-Based Stock Trader

Xin Feng
Department of Computer Science
University of North Dakota
Grand Forks, ND 58202-9015, USA

Chang-Hyun Jo
Department of Computer Science
California State University, Fullerton
Fullerton, CA 92834-6870, USA
jo@ecs.fullerton.edu

## Contact Point

Chang-Hyun Jo
Associate Professor
Department of Computer Science
California State University, Fullerton
Fullerton, CA 92834-6870, USA
(714) 278-7255
jo@ecs.fullerton.edu

# Agent-Based Stock Trader

Xin Feng
Department of Computer Science
University of North Dakota
Grand Forks, ND 58202-9015, USA

Chang-Hyun Jo
Department of Computer Science
California State University, Fullerton
Fullerton, CA 92834-6870, USA
jo@ecs.fullerton.edu

## ABSTRACT

In this paper, we introduce a unique implementation scheme of the Belief-Desire-Intention (BDI) model to be used in an agent-based application using Java. The example prototype system is the Agent-based Stock Trader (AST) that is a stock-trading expert based on intelligent agents. Agents in AST are based on the Belief-Desire-Intention (BDI) model in artificial intelligence.

This paper proposes how to program the BDI-based agents using the Java programming language, and how to make an agent-based application more intelligent and flexible. This paper contributes new implementation scheme of the BDI agents in the Java programming language useful on many applications. This work also shows how nicely implement the BDI agents with Java while manipulating BDIs intelligently and dynamically at runtime. Using our concepts and implementation scheme, the internet-based application like stock trading can be more intelligent and flexible.

## Keywords

Agent-Based Programming, Intelligent Agents, BDI model, Stock Trading Application

## 1. INTRODUCTION

An agent is a software entity that has some degree of intelligence and autonomy. It is a high-level system component, which is capable of having goals that it needs to be accomplished. From the traditional definitions, agents have the following properties: autonomous, perceptive, pro-active and cooperative [DeLoach 99]. Agents may have their autonomy and are not controlled directly by the others. The perception of agents allows the communication between the agents and their environments. Agents can cooperate with other agents to achieve the same goal. Intelligent agents have learning ability, so that those agents can learn and adapt to new environment to achieve their goals in the better way while learning. Agent computing is a new and active research area today [Franklin & Graesser 96] [Petrie 96].

To realize the intelligent agent computing effectively, a model based on Belief, Desire, and Intention (BDI) has been proved as a powerful technique [Bratman 87]. The "Belief" in BDI of an agent represents the knowledge about itself and the world (outside environment) of the BDI agent. The "Desire" in BDI represents a goal that the agent likes to achieve. The "Intention" describes a set of plans to achieve the predefined goal or to react to a specific situation.

There are many possible ways to support agents based on the BDI model. Multi-paradigm mixed with object-oriented programming, BDI concepts, and logic programming can be a way to support it. Another possibility we found was knowledge-base manipulation which database supports specific knowledge domain BDI. An embedded language such as a query language embedded in the existing languages is another possible solution to support the BDI-agent model. Other solution may include distributed computing language models similarly found in distributed computing systems such as JavaSpaces and JINI.

In this work, we have tried to program BDI-based agents for an example stocking trading application using the Java programming language. Since Java does not support agent programming, there was no proper language constructs to program agents. To make it worse, Java does not have any constructs that support the BDI concept at all, there is no way to program it. Therefore we have tried to program BDI-based agents using Java classes. We show here how to program agents, belief, desire and intention by using class constructs. Stock trading agents create objects for belief, desire and intention from the corresponding classes implemented already. The desire object finds appropriate an intention object to achieve its goal based on the information of the belief object. One of the merits of this paper is to show how to use the Java programming language nicely to build the BDI-agent application.

This paper shows not only how to build agents using Java, but also how to manipulate runtime knowledge dynamically to make agents intelligent.

There have been several research and experimental works based on the BDI model extending the Java programming language.

BDIM Agent Toolkit is implemented as a Java package to provide a prototype of runtime architecture [Busetta & R. 97, 98]. Developing a BDIM agent needs to derive classes for each of agent's plan and belief from the relevant BDIM base classes.

JAM Agents are composed of five primary components such as a world model, a plan library, an interpreter, an intention structure, and an observer [Huber 99]. JAM uses text-based files to specify the agent's beliefs, goals and plans. To implement a JAM agent, a user has to write appropriate primitive functions in Java and specify the agent's BDI in JAM files.

JACK offers Class, Interface, Method, Syntactic and Semantic extensions of Java implemented as Java plug-ins to support an agent-oriented development environment [JACK 99]. Especially, JACK uses the Database class as its data storage device to describe an agent's beliefs. Then the Database class is fully integrated with other JACK classes. However, JACK has its own extended syntax different from Java. To build a JACK agent, a user has to understand the JACK structure very well, but it is not easy.

All of these works are based on Java. Jack uses DB nicely for belief, however, it does not support dynamic manipulation of desire and intention, while our work does. Even though JAM provides a way to manipulate BDIs, neither JAM or BDIM supports well dynamic manipulation of BDIs not like ours. Once above agents are built by users, then they are fixed during runtime. In other words, users cannot modify them dynamically at runtime. Users have to redefine and recompile them if a modification is needed. In our work, we can build a BDI agent directly in Java with the help of database that a user can handle the BDI agent dynamically by manipulating the relations among the agent's belief, desire, and intention defined in the BDI knowledge-base at runtime.

This section has described what we would like to talk in this paper, and how we can compare our work with the related works. This introduction also explains some basic concepts used in our work.

In the Section 2, we describe the architecture of our prototype application, Agent-based Stock Trader (AST), constructed based on the BDI-agent concepts. In the Section 3, we show our implementation scheme of AST with the real Java codes. The AST is a just small example program by which we describe our nice implementation scheme of the BDI-agent concept using Java with the help of database. Therefore, we describe how we can nicely implement the BDI-agents while showing how we can implement a BDI-agent-based application like AST.

The Section 4 shows an example session of AST. With a nice user interface, each session just shows how much the AST system is applicable while using the BDI agents described technically in the previous sections. Finally we conclude with future work and references.

## 2. OVERVIEW OF AST

### 2.1 The BDI Concepts in AST

Agent-based Stock Trader (AST) is a stock-trading expert based on intelligent agents, which uses BDI model in artificial intelligence. Beliefs in AST specify all kind of stock information that agents know. Agents have explicit goals to achieve or events (desires) to handle. The stock names to get recommendation from expert agents in AST can be goals. A set of plans (Intentions) is applied to describe how agents achieve their goals based on certain beliefs. Each plan elucidates how to achieve a goal under varying environments. In this paper, we define a belief as a set of states representing environments, a desire as a set of goals, and an intention as a set of plans.

The examples shown in the next sections describe how BDI-agents work within AST. According to the technical analysis of the history and current information of some stocks, one plan will give its suggestion to help people to invest. However, another plan may be used if the history, information, or stock is different. Agent itself can autonomously decide which plan will be executed according its current situation.

### 2.2 The Architecture of AST

The following figure shows the conceptual overview of the AST system. Three-tiered design is considered to make the AST system components portable and flexible [Figure 1].

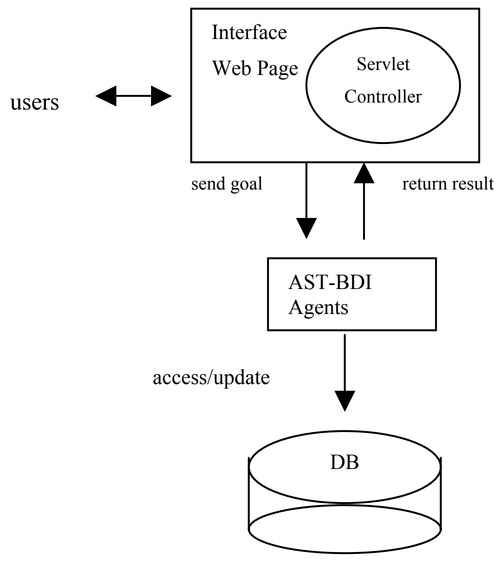


Figure 1. The Architecture of AST

The whole system consists three parts. The first level is the Servlet Controller, which is a link between users and BDI agents. It provides user interface and dynamically generates corresponding web pages. The second level consists of the BDI agents, which are the core part of AST. The third level is a data access layer where all information about the AST system such as the BDI agent's beliefs and the knowledge base are stored. In AST, we use relational database to represent the beliefs and the knowledge base.

When a user asks to find the recommendation of a specific stock, the interface agent will send this goal to the agent through the servlet program. Then the corresponding agent will check whether it can fulfill this goal. If so, it will choose proper plans to achieve the goal through its control structure and return the result. If not, it may directly ask other agents for help, or send a fail message back. Finally the interface agent generates the web page from which the user can get the result from the agent.

## 3. HOW TO IMPLEMENT BDI-AGENTS?

In this section we describe how to implement agents based on the BDI-agent concept using Java.

## 3.1 Agent-Based Programming Language

Agent-based programming is a new programming paradigm that has been arisen from research in distributed artificial intelligence. Unfortunately we have no proper agent-based programming language to implement the BDI agent concept well. We have decided to use the Java programming language to implement encapsulation of agents. However, Java does not support the BDI agent concept. Therefore we have to devise some mechanism to support the BDI concept on Java. We discuss here how to implement the BDI agents using Java. This is an important merit of this paper.

Even though we are designing a new agent-based programming language, Agent-Based Programming Language (APL), to support the BDI-based agents naturally, it is beyond the scope of this paper, and we discuss this in separate papers [Jo 2002].

## 3.2 How To Implement BDI-Agents

Java is a fully object-oriented platform-independent language and has a sufficient standard class library which including network programming facilities. It is a natural way to develop internet-based or web-based applications. Agent-based applications can be also developed in Java. Maybe it is not natural, but it seems to be suitable.

In AST, we use a relational database to represent an agent's belief, including the agent's knowledge base and the environment states.

We have thought about the following BDI mapping table [Table 1] that includes current states of belief, desires to achieve, and its corresponding intentions. By using this BDI mapping table, we can manipulate dynamically belief, desire, and intention at runtime. We can also show mapping among them, and we can change their mapping dynamically at runtime.

| Belief | Desire | Intention |
|---|---|---|
| Bi | Di | Ii |
| … | … | … |
| Bk | Dk | Ik |

Table 1. BDI Mapping Table

A proactive BDI agent acts based on a goal, scans belief, finds an intention based on them, and achieve the goal by executing intention [Figure 2a]. A reactive BDI agent reacts based on belief, finds a goal and its intention based on it, can also affect its belief [Figure 2b]. DeLoach also mentioned about a goal-directed behavior [DeLoach 99] for proactive agents.

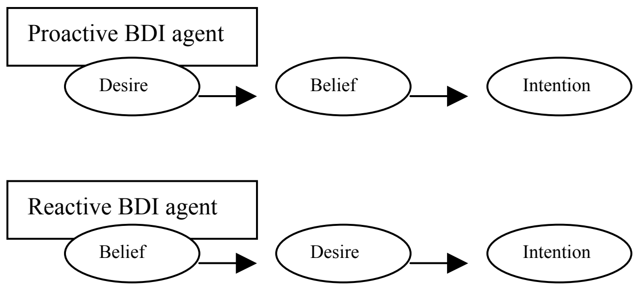


Figure 2 (2a & 2b). Proactive/Reactive BDI Agents

The AST application starts by initiating a certain goal defined in the desire definition. Based on specific belief an intention is chosen to achieve the goal based on current states of belief. There may be several sophisticated plans for each intention. Each plan triggers the event handler to achieve the goal based on behavioral description in each plan. An intention consists of a combination of one or more plans. In our current implementation, we use only one plan for each intention. To implement the concept of the BDI mapping table, we use several database tables with Java classes like the following. The related Java codes are shown in the next section.

The following table [Table 2] shows the AST database contents.

| AST Database | |
|---|---|
| **Table Name** | **Content** |
| Client | Clients' personal information and account balance. |
| Holding | What kinds and how many stocks clients own. |
| Orders | All orders clients posted and the agent processed. |
| Knowledge | Agent's knowledge. Desire, belief values and a synthesis value of belief values. |
| BPmap | Shows what plan will be chosen according the certain condition. |

Table 2. AST Database Contents

It is good to be able to manipulate BDIs dynamically for the BDI-based agent system. Belief can be dynamically changed to represent current environment. Desires can be set and changed to new ones at run time based on current situations such as belief. Intentions can be changed, updated, or newly added to achieve a current desire. However, it is hard to manipulate BDIs directly and dynamically well using Java. To solve this, the belief values can be stored in a file or database table, and can be manipulated dynamically at run time. For this AST application, belief was stored in a database table with which stock agents can consult and update dynamically. In the AST, the desire is also stored and manipulated dynamically in a database table. Both desire and belief are stored at the "Knowledge" table in the AST database. To map the corresponding plans in the intention dynamically based on both the current goal defined in the desire and the current environment based on the belief, another table "BPmap" has been used in the AST system. The table "Client" holds clients' account information. The table "Holding" keeps the information of stocks owned by clients. The table "Orders" keeps the information of orders which clients post and the AST agent should process. Among contents in the AST database, the table "Knowledge" is the most important table to manipulate the BDI information dynamically.

The following table "Knowledge" shows a knowledge base for desire and belief [Table 3].

| AST BDI Knowledge | | |
|---|---|---|
| **Field Name** | **Data Type** | **Description** |
| Symbol | Text | Stock name |
| A | Number | World economy |
| B | Number | US economy |
| C | Number | Financial markets and institutions |
| E | Number | Others |
| F | Number | Buy or sell market |
| Belief | Number | The result of analysis of A~F |

Table3. AST Database Knowledge Table Definition

The stock name "Symbol" is a kind of desire to get any recommendation on its stocks through the stock market for stock exchange (either selling or buying). For example, the "Symbol" may hold desire that represents a certain stock of company such as ORCL (Oracle), YHOO (Yahoo), IBM (IBM), MS (Microsoft), and etc. The knowledge A through F represents its belief that represents the current environment surrounding its stock. Its belief holds the values that represent the current status of environment such as world economy, US economy, financial markets and institutions, other factors, and buy/sell market. The final field "Belief" holds the value that we can get from the analysis of the belief factors, A~F, by applying certain mathematical and stochastic equations. In summary, the "Symbol" keeps the value of current goal/desire, the values from A to F keep the current environment/belief, and the value of "Belief" is synthesized from the values of A through F. The following table shows possible values for the table "Knowledge" [Table 4].

| Values in Knowledge | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Symbol** | **A** | **B** | **C** | **D** | **E** | **F** | **Belief** |
| YHOO | -1 | 0 | 1 | 1 | 1 | 1 | 3 |
| … | … | … | … | … | … | … | … |

Table 4. Sample Values in Knowledge Table

The following table shows the table "BPmap" that describe the mapping to the corresponding plans defined in the intention, from the information based on the current environment defined in the belief [Table 5]. Therefore, it is a kind of mapping table from belief to intention.

| BPmap | | |
|---|---|---|
| **Field Name** | **Data Type** | **Description** |
| LowLimit | Number | Lower limit of belief |
| UpLimit | Number | Upper limit of belief |
| Plan | Text | Corresponding plan |
| PlanChoice | Text | Choice of a plan |

Table 5. BPmap Table Definition

The condition fields, LowLimt and UpLimit, represent the lower limit and upper limit of the value of belief, from which the corresponding plans are selected. Both "Plan" and "PlanChoice" are to choose plans defined in the AST intention, which can be selected by a certain rules defined in the AST system based on the current goal/desire and the current situation/belief.

The following table shows some example values for each column in "BPmap" table [Table 6].

| Values in BPmap | | | |
|---|---|---|---|
| **LowLimit** | **UpLimit** | **Plan** | **PlanChoice** |
| -3 | 3 | Hold | 4 |
| … | … | … | … |

Table 6. Sample Values in BPmap Table

The "Plan" may have recommendation values for stock exchange such as "Strongly Sell", "Moderate Sell", "Hold", "Moderate Buy", or "Strongly Buy". The "PlanChoice" holds the identification number for each plan defined in the AST intention definition. Theoretically, we may have one or a combination of plans. However, to make this example simple, we use only one plan as a recommendation value.

## 3.3 How To Program BDI-Agents

The stock export agent can be created by defining its own Belief, Desire, and Intention classes. The agent has its own main controller whose control structure is shown in Figure 3.

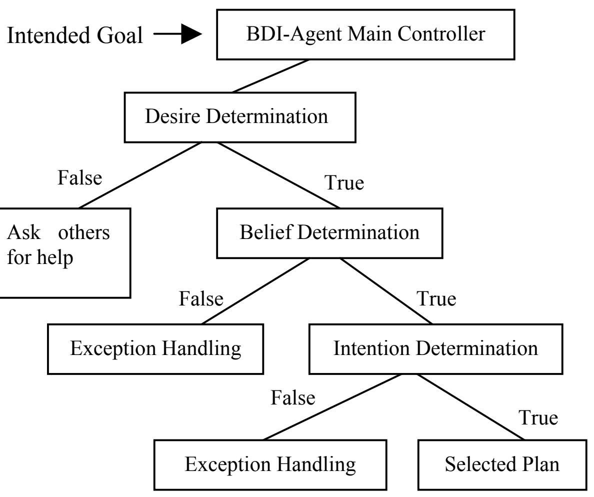


Figure 3: BDI-Agent Runtime Control

The following code is the Belief class to access and update the knowledge base for its belief. The "getBelief(String goal)" function accesses the belief knowledge base based on the current goal ("getBelief(String goal)"), and performs some mathematical and stochastic equations to calculate the exact current environment surrounding the goal. (Even though in this example application we use a just simple summation formula, in the real application, any heuristic formula can be used to diagnose the exact case and stock market.).

```
// Belief.java
package agent;
import java.util.*;
import java.sql.*;

public class Belief {
  private Connection connection;

  public Belief (Connection x)
  {
      this.connection = x;
  }

  public Connection getConnect() {return connection;}

  //Get believes according to the desire
  public int getBelief (String goal)
  {
    try {
      Statement st = connection.createStatement();
      String sql = "SELECT * FROM knowledge WHERE
Symbol='" + goal + "'";
      ResultSet rs = st.executeQuery(sql);
      boolean more = rs.next();
      if (more) {
        int sum = 0;
        ResultSetMetaData rsmd = rs.getMetaData();
        for (int i=2; i<rsmd.getColumnCount(); ++i)
              //1st element: Symbol
              sum += rs.getInt(i);
          System.out.println("sum="+sum);
          return sum;    // A value of belief synthesis
        }
        else
              return 11111; // 11111 is  the false flag
      }
    catch (SQLException sqlex) {
        sqlex.printStackTrace();
        System.out.println("Desire SQL error\n");
        return 11111;  // A false flag
      }
   }

  // update believes of the desire
  // … …
}
```

The Desire class accesses to its knowledge base and finds whether a certain goal can be achieved or not (by "achievable(String goal)"). If the goal is achievable the main agent program will select proper plans under proper situation based on belief. The following program shows the Desire class for the AST application.

```
// Desire.java
package agent;
import java.util.*;
import java.sql.*;

public class Desire {
    private Connection connection;

    public Desire (Connection x)  // constructor
    {
              this.connection = x;
    }

    public boolean achievable(String goal)
    {
        try {
              Statement st = connection.createStatement();
            String sql = "SELECT Symbol FROM knowledge
WHERE Symbol='" + goal + "'";
              ResultSet rs = st.executeQuery(sql);
              if (rs.next()){
                  System.out.println(rs.getString(1));
               return true;
              }
              else
                  return false;
        }
        catch (SQLException sqlex) {
              sqlex.printStackTrace();
              System.out.println("Desire SQL error\n");
              return false;
        }
  }
  // add a new goal
  // … …
  // delete a goal
  // … …
}
```

Theoretically, we can build an intention class to list a set of plans the agent prepares for possible goals. Each plan is represented as a method in the intention class, which is a behavioral representation to describe how to achieve a goal under a particular situation represented by a set of states in the belief representation. However, Java does not support dynamic configuration (mapping) of a combination of plans, which are suitable for the BDI agent concept. Therefore, in our current implementation, we use an intention-mapping table ("BPmap") in the database to store a set of rules that explain how to dynamically select a suitable plan under a particular situation (performed by the "selectPlan" function). Then the agent can dynamically access and monitor its belief through JDBC and SQL statements. The following program shows its Intention class.

```
// Intention.java
package agent;
import java.util.*;
import java.sql.*;

public class Intention {
    private Connection connection;

    public Intention(Connection x)
    {
        this.connection = x;
    }

 public String selectPlan(int bresult) {
      try {
            Statement st = connection.createStatement();
            String sql = "SELECT Plan FROM BPmap WHERE
UpLimit>" + bresult
              + " and LowLimit<=" + bresult;
            ResultSet rs = st.executeQuery(sql);

         if (rs.next()){
              String s = rs.getString(1);
              System.out.println("Intention plan = "+s);
            return s;
            }
            else
              return "fail";
      }
      catch (SQLException sqlex) {
          sqlex.printStackTrace();
             return "fail";
      }
   }

  // update the mapping table
  // … …
}
```

We may also have methods to update plans by which we can update the mapping values in the mapping table (BPmap table). It is a good idea to have a proper programming language to update plans dynamically and more naturally.

Finally, we can create an agent class by declaring its own belief, desire, and intention from their definitions. When a goal is sent to an agent through the desire, the agent will check whether it can handle. If so, its corresponding desire will be achieved by executing proper plans defined in its intention. If not, it may delegate it to other agents for help. It can be implemented by message exchange among multi-cooperative agents, even though our current system is not implemented in this way. The following code shows the Agent class for the AST application.

```
// Agent.java
package agent;
import java.util.*;
import java.sql.*;

public class Agent {
   public Belief B;
   public Desire D;
   public Intention I;

   public Agent (Connection x) {  //constructor
       D = new Desire(x);
       B = new Belief(x);
       I = new Intention(x);
   }

   // agent performance controller
   public String perform(String goal)
   {
       if ( D.achievable(goal) ) {
          int b = B.getBelief(goal);
          if (b==11111)  // 1111 means False flag
             return "Belief sensor error";
          else {
             String p = I.selectPlan(b);
             return p;
          }
       }
       else
          return "Unachievable goal";
    }

}
```

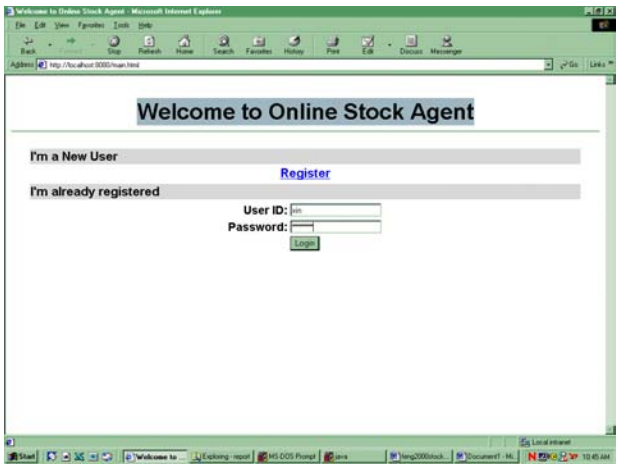
An agent always starts from the behavioral description named "perform" in this application ("perform(String gaol)"). A desire is passed to the application through the string "goal" ("String goal"). The desire class checks whether the current goal is achievable or not (by "D.achievable(goal)"). If it is achievable, the system gets the current environment through the belief (by "B.getBelief(goal)"). Unless the current belief is not available, its plan is chosen to achieve the given goal using the information of belief ("I.selectPlan(b)").

## 4. EXAMPLE SESSIONS

Here we show some example sessions using AST.  The AST system is working based on the principles described in the previous sections. Here we just list example sessions via snap shots of the application.

### 4.1 Login Entry

The AST system starts with the entry form of user login. The registered users may login with their user Ids and passwords. The new user may register into the system. This session is related to the table "Client" in the database [Table 2]. Here is a snapshot of the entry login form.



Welcome to Online Stock Agent - Microsoft Internet Explorer

http://localhost:8080/main.html

Welcome to Online Stock Agent

I'm a New User

Register

I'm already registered

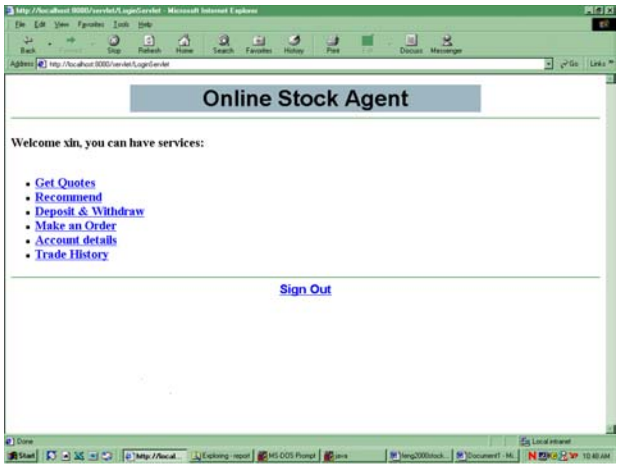
User ID: xin

Password:

Login

### 4.2 Basic Services

After successful login, a list of services regarding online stock trading that AST can provide appears. Currently the prototype system provides six services such as getting quotes, recommendation for stock trading, deposit/withdraw, order placement, account details, and trading history.



http://localhost:8080/servlet/LoginServlet - Microsoft Internet Explorer

Online Stock Agent

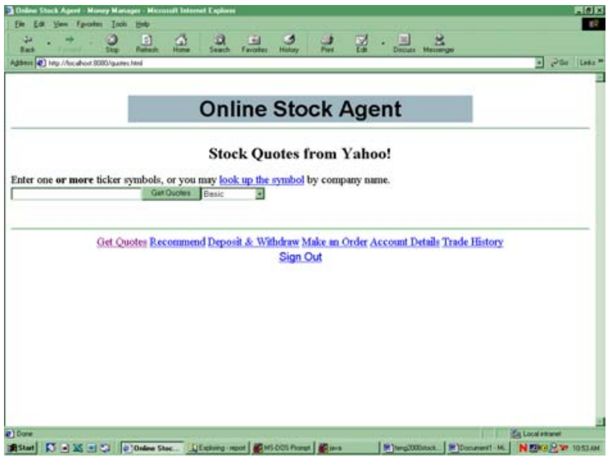
Welcome xin, you can have services:

- Get Quotes
- Recommend
- Deposit & Withdraw
- Make an Order
- Account details
- Trade History

Sign Out

### 4.3 Stock Quotation

The selection of the stock quote option brings the user to the stock quotation menu. We borrow the Stock Quotes from Yahoo [Yahoo 02], due to the limitation of accessing to a real stock market database.



Online Stock Agent - Money Manager - Microsoft Internet Explorer

http://localhost:8080/quotes.html

Online Stock Agent

Stock Quotes from Yahoo!

Enter one **or more** ticker symbols, or you may look up the symbol by company name.

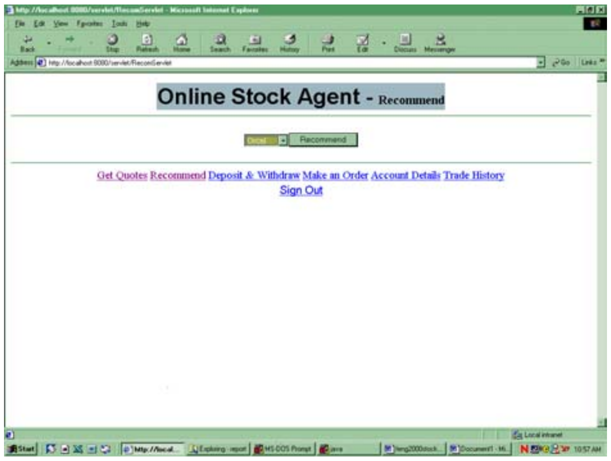
Get Quotes   Basic

Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History

Sign Out

### 4.4 Stock Trading Recommendation

The recommendation service provides a recommendation to buy, hold, or sell for a certain stock. Actually the expert agent does the back-ground work. Once a user selects a stock and clicks on the "Recommend" button, this stock name will be sent to the expert BDI-Agent as a goal through the servlet program. Then the expert BDI-Agent achieves the goal (by "perform(String goal)") just as we described in the previous sections. It checks its desire and current belief to select and execute a suitable plan. Finally the expert BDI-Agent sends back its achievement, the recommendation of the stock, to the interface through the servlet program. Both "Knowledge" and "BPmap" tables [Table 2] are related to this session.



http://localhost:8080/servlet/RecomServlet - Microsoft Internet Explorer

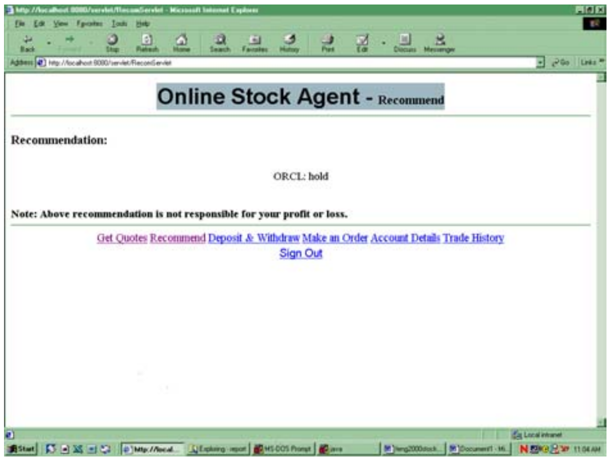
Online Stock Agent - Recommend

Orcl   Recommend

Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History

Sign Out

Once the agent performs a certain function needed to get an appropriate recommendation, the system shows the result. This is the corresponding answer from the export agent. With this particular example, the AST system recommends the user to hold the ORCL stock continuously.


Online Stock Agent - Recommend
Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History
Sign Out

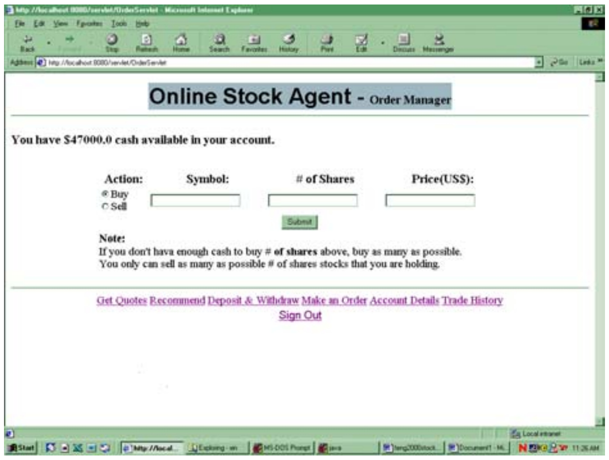
**Recommendation:**

ORCL: hold

**Note: Above recommendation is not responsible for your profit or loss.**

## 4.5 Posting An Order

Here the user can post his stock order, and the order management agent will process it for the user. The order will be either successful or fail based on the several reasons such as the availability of the stock in the market to buy, and the availability of the funds to buy the stock, and etc. The related table in the database is "Orders" [Table 2].


Online Stock Agent - Order Manager
Submit
Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History
Sign Out

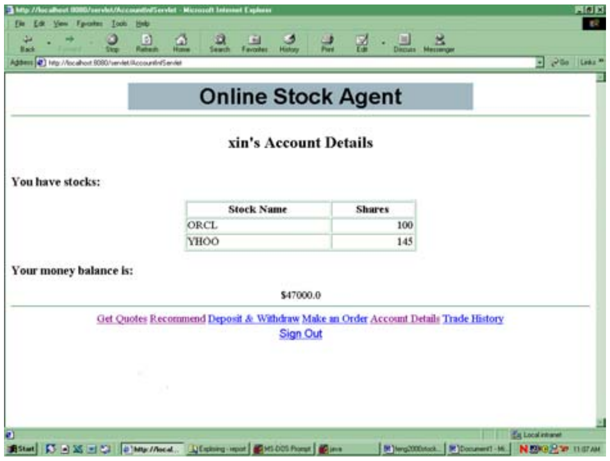
**You have $47000.0 cash available in your account.**

| Action: | Symbol: | # of Shares | Price(US$): |
|---|---|---|---|
| [✓ Buy] Sell | | | |

**Note:**
If you don't hava enough cash to buy # **of shares** above, buy as many as possible.
You only can sell as many as possible # of shares stocks that you are holding.

## 4.6 Account Details and Trading History

The account detail option in the main menu can be selected to see the user's account in detail. It shows what stocks and how much money the user owns. The tables "Client" and "Holding" [Table 2] relate to this session.


Online Stock Agent
Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History
Sign Out

**xin's Account Details**

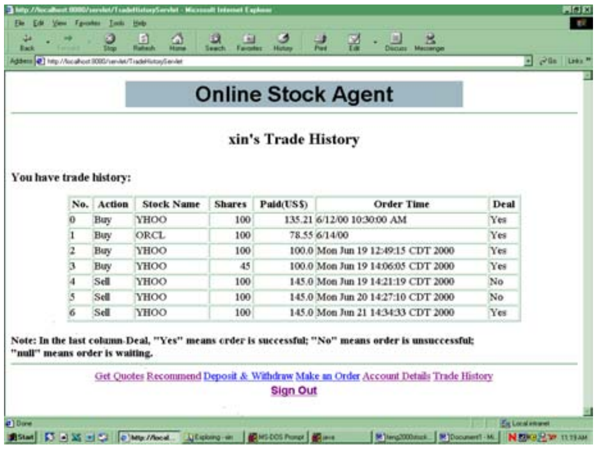
**You have stocks:**

| Stock Name | Shares |
|---|---|
| ORCL | 100 |
| YHOO | 145 |

**Your money balance is:**

$47000.0

The user may select the trading history option to see the user's all trade history in the current past.


Online Stock Agent
Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History
Sign Out

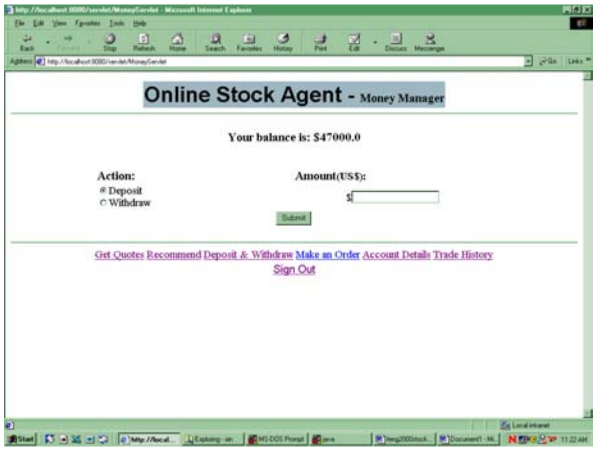
**xin's Trade History**

**You have trade history:**

| No. | Action | Stock Name | Shares | Paid(US$) | Order Time | Deal |
|---|---|---|---|---|---|---|
| 0 | Buy | YHOO | 100 | 135.21 | 6/12/00 10:30:00 AM | Yes |
| 1 | Buy | ORCL | 100 | 78.55 | 6/14/00 | Yes |
| 2 | Buy | YHOO | 100 | 100.0 | Mon Jun 19 12:49:15 CDT 2000 | Yes |
| 3 | Buy | YHOO | 45 | 100.0 | Mon Jun 19 14:06:05 CDT 2000 | Yes |
| 4 | Sell | YHOO | 100 | 145.0 | Mon Jun 19 14:21:19 CDT 2000 | No |
| 5 | Sell | YHOO | 100 | 145.0 | Mon Jun 20 14:27:10 CDT 2000 | No |
| 6 | Sell | YHOO | 100 | 145.0 | Mon Jun 21 14:34:33 CDT 2000 | Yes |

**Note: In the last column-Deal, "Yes" means order is successful; "No" means order is unsuccessful; "null" means order is waiting.**

The fund manager handles both deposit and withdraw for the user.


Online Stock Agent - Money Manager
Submit
Get Quotes Recommend Deposit & Withdraw Make an Order Account Details Trade History
Sign Out

**Your balance is: $47000.0**

| Action: | Amount(US$): |
|---|---|
| [✓ Deposit] Withdraw | $ |

## 5. CONCLUSIONS

Agent-based computing is emerged as a future-computing paradigm. The BDI model is one of the powerful techniques to describe autonomous intelligent agents. In this paper we have presented a stock trading application based on intelligent agents using the BDI model.

One of the merits of this work is to show how nicely to use the object-oriented language, Java, can be used to implement the BDI-agent-based application. The Java programming language does not support any construct for the BDI-agent concepts. However, in our work, the agent and its desire, and intention are programmed as Java classes. Its information of desire, belief, and intention are stored in a database and updated dynamically at runtime as the environment changes. Therefore, in our work, we show how nicely to use the Java programming language to program the BDI-agent-based application by using a database to implement the knowledge base for BDIs.

We also show how we can manipulate BDIs dynamically at runtime without having any trouble while Java does not support any runtime knowledge management. Our work also shows how we can implement a real world application like stock trading using the BDI-agent model to represent the real world problem more naturally in a better way.

One of the future works may include to implement this system in a real agent-based programming language like APL [Jo 2002] which we have developed as a concurrent research work. It may prove how BDI agent model well match the real world solution and solve the complex system more intelligently.

The Java/Servlet programming technique on the Internet has been used to implement our prototype system.

## 6. REFERENCES

[1] Agent Oriented Software Pty. Ltd., JACK Intelligent Agents User Guide, http://www.agent-software.com.au, 1999.

[2] Bratman, Michael E., Intention, Plans, and Practical Reason, Harvard Univ. Press, 1987 (also by CSLI Publication, 1999).

[3] Busetta, Paolo and Ramamohanarao, Kotagiri. The BDIM Agent Toolkit Design, Technical Report 97/15, Department of computer Science, The University of Melbourne, Australia, 1997. http://www.cs.mu.oz.au/publications/tr_db/TR.html.

[4] Busetta, Paolo and Ramamohanarao, Kotagiri. An architecture for Mobile BDI Agent, Mobile Computing Track, ACM SAC '98, 1998.

[5] DeLoach, Scott A. Multiagent Systems Engineering: A Methodology and Languages for Designing Agent Systems, http://en.afit.af.mil/ai/publications/Conference/aois-99/MaSE-AOIS99.htm, 1999.

[6] Franklin, Stan and Graesser, Art. Is it an Agent, or just a Program? : A Taxonomy for Autonomous Agents, http://www.msci.memphis.edu/~franklin/AgentProg.html, Also in the Proc. of the 3rd International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

[7] Huber, Marcus J. JAM: A BDI-theoretic Mobile Agent Architecture, Proc. Of the Autonomous Agents '99, Seattle, USA, 236-243, 1999.

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

[9] Petrie, Charles J. Agent-Based Engineering, the Web, and Intelligence, http://www-cdr.stanford.edu/NextLink/Expert.html, also appeared in the IEEE Expert, (December 1996).

[10] Yahoo Stock Quotes, http://finance.yahoo.com, 2002.