A FRAMEWORK FOR BDI AGENT-BASED SOFTWARE ENGINEERING

CHANG-HYUN JO, GUOBIN CHEN AND JAMES CHOI

Abstract. Agent-based software engineering has been regarded as a new notion to build complex software systems. A seamless development, from modeling to implementation of agent-based software, is the new revolution in software engineering. However, there has not been a simple and efficient means to facilitate agent-based software development.

In this paper, we propose an agent-based software development process based on the Belief-Desire-Intention (BDI) agent model as a new software development process. The Belief-Desire-Intention (BDI) model has been used a fundamental ingredient to the new agent-based modeling method. In our agent system proposed here, each agent is made flesh by assigning its own belief, desire and intention. Here we have shown a seamless software development modeling technique consistently based on the BDI model. Even though there are many valuable arguments to define agents beside the BDI model, we currently adopt the BDI model to our agent modeling because we try to develop a simple and realistic agent-based modeling technique. We will extend this work to support other key concepts in agent computing.

Here we propose a new approach comparing with our previous approach to support the BDI agent-based modeling techniques. The previous approach finds intention, desire and belief from two different kinds of use cases – external use cases and internal use cases – and supporting tools. The new approach finds in sequence desire, intention and belief by using not only different kinds of use cases, sequence diagram, activity diagrams, and dataflow diagrams. To prove the usability of our software development process, we also provide a case study of agent-based software development for the California Super Lotto System (CSLS) application. This paper also introduces
a brief structure of a CASE tool, which we have currently been developing to support the BDI agent-based software development process.

**Keywords:** Agent-based modeling, agent-oriented software engineering, CASE tool

---

1. Introduction

Following the design method, *top-down structured design*, also known as *composite design*, in 1960s and 1970s, *object-oriented analysis and design* has popularized since the 1990s [1] [16]. The *agent programming* has been recognized as a new paradigm of programming in the near future by simplifying the complexities of distributed computing and by overcoming the limitations of current user interface approaches [2] [7] [12] [14] [20] [21] [22]. Therefore, agent-based computing has been emerged as a next significant breakthrough in software development and a new revolution in software. Like a human agent, a software agent can carry out a task. A software agent has its own characteristics such as autonomous, social ability, reactivity, pro-activeness, cooperative, learnable and adaptable [17] [22]. Agent is a natural abstraction of the real world. It can model the real world, with its own goals, communicating and often working together with other agents to achieve mutual benefit. In this way agent-orientation can be considered a natural extension to object-orientation. Since an agent has in-built domain expertise, it can learn, reason logically, adapt to new circumstances, and work as a team to achieve more than they could individually. Agent-based software is more powerful than that of object-oriented.

In this paper, we adopt the belief-desire-intention (BDI) agent model to define the system. The principles of the architecture of the BDI agent were established in the mid-1980s [3]. This agent model has come to be possibly the best-known and best-studied model of practical reasoning agents [9] [14] by combining a reputable philosophical model of human practical reasoning and agent software implementations.

In BDI terms, *Beliefs* represent knowledge of the world. However, in computational terms, beliefs represent the state of the world, such as the value of a variable, a relational
database, or symbolic expressions in predicate calculus. Desires (or goals) form another essential component of system state. In computational terms, a goal may simply be the value of a variable, a record structure, or a symbolic expression in some logic. The important point is that a goal represents some desired end state. The committed plans or procedures are called Intentions, which represent the third necessary component of system state. Computationally, intentions may simply be a set of executing threads in a process that achieve the goals (desires) of the system.

Beliefs, Desires, Intentions are the basic components of an agent system designed for a dynamic, uncertain world. So, as BDI agents, they must have explicit goals to achieve or events to handle (desires); a set of plans (intentions) is used to describe how agents achieve their goals; a set of data called belief describes the state of the environment [10]. In this paper, we adopt some methods used in functional and OO methods to propose a simple and systematic approach for analyzing and designing BDI agents. There already have a few methodologies for the “agent computing”, such as Deloach’ MaSE [6], Frank’s methodology [4], and Wooldridge’s Gaia [21], but none of them is a simple and thereby an efficient method to analyze and design agent-based software. In our approach, we innovate the BDI Agent Software Development Process (BDI-ASDP) with the aid of the Use Case, the method of OO, to find the Desires and Intentions, and the Data Flow Diagram, the method of functionality, to find the Beliefs. Following the BDI Agent Software Development Process approach, developers can decompose systems into Belief, Desire, and Intention (BDI) agent models. The idea of our approach is partially from Einhorn’s methodology [7] that discovers the intentions, desires, and beliefs orderly. In their approach, they first use the External Use Case to analyze the requirement. Then, they apply the Internal Use Case for the design of their system. They also create the Agent Interaction Diagram, which is similar to the OO’s Sequence Diagram, to illustrate the interaction of agents. In our approach, we extract the Goals from the requirements first, and then the Intentions and Beliefs. We also add the Agent Activity Diagram to discover the intentions and Data Flow Diagram to find the beliefs in our approach.

In the following sections, we will first discuss what an agent is and what kind of agent we use in this paper. Second, we will illustrate how to analyze a system from its given requirements. That means how to find some agents from the system requirements. After that, we will discuss the way to design a complete system based on the analysis. Next we will discuss about a CASE Tool that we currently design and build. Finally, we will give
out the stereotype of the agents in Agent Programming Language. In order to illustrate our approach better, a case study will be given step by step. Because of the limitation of the paper length, a detailed case study is provided at http://www.ecs.fullerton.edu/~jo/research/GuibinChen2003/GuobinChen2003-
CaseStudy.pdf.

2. A New BDI Agent Modeling

2.1 BDI Agent Software Development Process

Here we only focus on the analysis and design phases. Our approach is designed to identify the Beliefs, Desires, and Intentions for agents during the software analysis and design phases. To make our approach more systematic, we will also refer to the requirement and implementation. The testing and maintenance phases are not discussed here.

In our previous approach, we suggested a software development process to capture system requirements in the sequence of Intentions, Desires and Belief [7] [12]. In the beginning of our development process we use external use cases, which are general plans indicating how a specific service can be provided from an external point of view. We then refine these plans into goals using internal use cases. The internal use cases decompose a service into one or more goals. In addition the internal use cases also provide a more precise description of each goal and its corresponding plan. After we have discovered a goal and described a plan for each goal we need to discover the beliefs that will be necessary for each goal to be completed. The beliefs are determined for each goal by analyzing each goal’s plans and determining what beliefs will be necessary for its completion. Now that we have described a complete BDI we can assign it to an agent.

Figure 1 shows our new approach to discover BDIs from system requirements. We find desire first from the system requirements and then find its intention and corresponding belief. This idea comes from the natural approach we usually do in the real world. In the
real world, people usually establish the goals in the beginning. Under these goals, they work out correspondent plans in order to fulfill these goals. For the sake of accomplishing the plans, people have to perform some actions. These need people to communicate with their environment, or in other worlds, people need to know the data to describe the world.

Following the natural style of human thought: goal-plan-data, our approach will first extract the desires from the requirement, and then create the proper intentions. At last we will find the beliefs. More specifically, we are going to use the Agent Model (namely Role Model such as External Use Cases) to extract goals (belief) from the requirements, the Dynamic Model (namely Coordination Model - Communication and Negotiation Model such as Internal Use Cases, Sequence Diagrams, and Activity Diagrams) to capture the plans (intention) in the following, and the Data Model (such as Data Flow Diagrams) to obtain the data (belief) moving in the environment. Our software modeling is processed and modified iteratively and evolutionally. Therefore, software engineers may return to the beginning if it is necessary. The high level view of developing BDI agents is depicted in Figure 2.
Our approach emphasizes on the analysis and design phases. In the analysis phase, we apply External Use Case to extract the desires from the external point of view as well as to fully understand the requirement and prepare for the design phase. In the next phase, the design phase, we utilize the Internal Use Case to capture the intentions and Data Flow Diagram to find the beliefs. After we discover the beliefs, desires, and intentions, we collect them together to form the BDI Agent Cards.

In the detail, the BDI Agent Software Development Process (BDI-ASDP) embraces ten steps: Initial Problem Statement, Enterprise Software Assessment, Brief External Use Case, Detailed External Use Case, Structuring Goals, Internal Use Case, Sequence Diagram, Agent Activity Diagram, Data Flow Diagram, and BDI Agent Cards. The BDI-ASPD is a specialization of traditional software engineering methodologies. The general phases and steps are shown on the Figure 2.

The most related work to this is Einhorn’s research [7] [12]. Our approach is different from the research of Einhorn [7] [12] in two aspects. First, our approach discovers the BDI agents following the natural style of human thought: Desire-Intention-Belief, rather than Intention-Desire-Belief. Second, constructing the Conceptual Agent List is not necessary in our approach. We do not anticipate the agents. We form the agents on agent cards that we obtain from the process of the analysis and design. With analyzing the system requirements, we extract the goals. Following the Desires, we capture and create the Intentions. Based on the Desires and Intentions, we find and create the Beliefs for the Intentions to fulfill the Desires. The strategy of finding BDI is depicted in Figure 3.
Although we have drawn it as a single flow from left to right, in practice the methodology is iterative. The analysts or designers may move between steps and phases freely and each successive pass will produce additional detail providing a complete, yet consistent system design.

We have taken a brief tour of the complete BDI agent software development process. We will now systematically describe our process in much greater detail. The following section will describe each artifact in the order that is presented in Figure 2.

We will present a detailed discussion of each artifact that can be created in our process. We should provide a case study providing a practical view of an actual artifact once it is created. Because of the limitation of the paper length we will provide a detailed case study on our web page.

3. **Requirements Analysis**

The requirement is the initial part of our BDI Agent Software Development Process. It will assist us to understand the system and have a clear thought on how to construct it.
3.1 Step1: Initial Problem Statement

The purpose of the Initial Problem Statement is to describe the problem that a customer expects the system to solve. The Initial Problem Statement is the starting point of the system analysis. It is the input to the Capturing Goals step. Normally, it is the conceptualization of the system from the user’s point of view, and describes the general functions or services that the system will provide. The Initial Problem Statement may alternately be referred to in a more descriptive manner such as “The System Requirements” or “The User Specification”. Since the Initial Problem Statement is a result in which both the customer and software developers participate, it is a two party agreement on a high level view of the system. A full understanding of the Initial Problem Statement is the solid start of the whole process.

Case Study

1. Introduction

1.1 Purpose of this document
This is the Initial Problem Statement (IPS) for a California Super Lotto Finding System (CSLFS) using BDI Agent Software Development Process (BDI-ASDP) Methodology. This document describes the purpose, function and design of the product desired by California Super Lotto Club.

1.2 Scope of this document
The project is to produce California Super Lotto Finding System (CSLFS). CSLFS is an entertaining application to be implemented using Agent based approach. As for the design, it needs to be a multi-agent system design.

The data to be worked on will comes from two sources: from users and from other systems. User will provide an input data to be processed. The system will then process it in conjunction with information provided by other systems such as http://www.calottery.com/.

The aim of this system is to assist a person to select California Super Lotto numbers. This is broken down into two categories of sub-services: getting hot numbers and searching winning numbers.
1.3 Overview of Document

1.4 Abbreviations

1.5 References

2. General Description

2.1 Product Perspective
- This product is developed using Agent-Oriented system approach.
- This product will be designed using BDI-ASDP methodology.
- This product will be implemented in APL Agent Programming Language.
- This system is intended to be stand-alone, but it will work in conjunction with other system such as http://www.calottery.com/.

2.2 Product Function
The review of the functionality to be performed by the system are:
Finding hot numbers
The system will find the hot numbers for a player whenever the player requests this service.
Searching the favorite numbers
A player will input a draw of numbers into the system and it will search if his favorite numbers have ever won.
Storing and updating Interests, Preferences and History
The system will store a draw of new numbers that it obtains from the web site http://www.calottery.com/ into its database.
Display output
System will display output after query is processed successfully.

2.3 General Constraints
The initial design is for a non-interactive system and is a subject of changes in the future into a more interactive one. The initial design will focus on finding hot numbers and searching the favorite numbers. Other services will be added later.

2.4 Assumptions and Dependencies

The server, which provides data and storage facility, is assumed to run continuously.

3. Specific Requirements

3.1 Functional Requirement

3.1.1 Finding Hot Numbers

3.1.1.1 Purpose

The purpose is to enable the system to provide the hot numbers to a player.

3.1.1.2 Input

User needs to send a request to the system.

3.1.1.3 Process

System will gather the newest result from the official site of the California state lottery at http://www.calottery.com and the history numbers from the database and apply some algorithms to generate some hot numbers and display to the player.

3.1.1.4 Output

A group of hot numbers will be displayed in front of a player.

3.1.2 Searching Favorite Numbers

3.1.2.1 Purpose

The system will provide a service to a player if his favorite numbers have ever won.

3.2 External Interface Requirement

3.2.1 User Interface

The interface needs to be extremely simple and obvious. Any user should immediately know how to get their desired response from the system without any confusion.
Example Scenario
A player is willing to play the California Super Lotto with the assistance of the CSLFS. First, he clicks the Hot Number button. The computer responds him with some numbers displaying on the screen. Before he fills out his tickets, he wants to know if his combinations have occurred before. He clicks the Search Number button. The computer pops up a form to ask him input a draw of numbers. After he submits his numbers, the computer displays the search result with the combination of winning numbers and the winning times with each combination. At last, he makes his final decision.

3.2 Step 2: Enterprise Software Assessment
The Enterprise Software Assessment describes how the new system would impact the current enterprise software. This process basically illustrates on what environment the current system runs. It is to answer the following question: Is there any problem when the new system runs with the existing enterprise software, and how to solve it if there is any?

4. System Analysis: Capturing Goals
Analysis phase emphasizes an investigation of the problem and requirements, rather than a solution. During analysis phase, we emphasize on finding the first element of agents—Desire, in the problem domain.

4.1 Step 3: Brief External Use Case
The Brief External Use Case treats the System as a black box, and shows how the entities outside of the system interact with the system. The reason we call such Use Cases external Use Cases, as they can show how external entities in the environment interact with the system and how the external entities use the system to get something done.
Usually, “Goals can be extracted from what the system is trying to achieve and generally remain constantly throughout the entire analysis and design process” [6]. They can be unfolded into finer- and finer-grained goals. Cockburn creates three level goal theories in his book [5]. He argued that, in certain circumstances, we have summary-level, user-level, and subfunction-level goals even though we can always find upper level goals over the summary-level and lower level goals under the subfunction-level. Jo also discusses the concept of different level goals in his paper [10]. He states that we can find a global goal for a system and a set of intentions to achieve the global goal. We implement these intentions that in turn become the sub-goals of other intentions, and so on and on. That gives us an idea that we can decompose goals from top to bottom, step by step.

The primary task of the Brief External Use Case is to utilize Brief Use Case technique [15] to capture the system goal (global goal). We start from the initial problem statements and will break it down in the next step, the Detailed External Use Case, to extract sub-goals of it. The Brief Use Case technique in the analysis phase, not only can help us understand the system better but also find a goal for an agent. Fantechi’s research states that a use case defines a goal-oriented set of interactions between external actors and the system [8]. If there is a use case captured from the system requirements, we can extract a goal from this use case. Besides capturing the system goals, the Brief Use Case technique also describes the interaction between a system and its environment, and reflects external actors. Actors usually are persons or other systems, who trigger the system behavior to achieve a certain goal. Goals should be specified as abstractly as possible without losing the essence of the requirements. Analysts can perform this abstraction by removing detailed information when specifying goals [6]. For instance, the overall goal of a system might be to “Determine if an invalid user tries to login.” Thus, one of the sub-goals is to detect login violations. How to detect violations may be changed at different time or in various operating systems and thereby should not be a goal or sub-goal.

During the Brief External Use Case process, we identify the services of the system being developed from an external point of view; we do not describe the internal workings, components, or design of the system. Through this task, we determine what services our system should provide. The Brief External Use Case captures who (actor) does what
(interaction) with the system, for what purpose (goal), without dealing with the system internals.

Case Study

By studying the Initial Problem Statement in the Step 1, we find that the California Super Lotto Finding System (CSLFS) is to assist a player playing the California Super Lotto. It has two services or functionalities to meet this goal. One is to find hot numbers, and the other is to search favorite numbers. If the Initial Problem Statement is too general to find the global goal from the existed scenarios, we must ask the customers to provide other scenarios. In our case, we construct the scenarios according to the Initial Problem Statement. We get the External Use Case in following:

<table>
<thead>
<tr>
<th>Use Case</th>
<th>(The use case name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Actor</td>
<td>(The initiator)</td>
</tr>
</tbody>
</table>

4.2 Step 4: Detailed External Use Case

The purpose of the Detailed External Use Case is to describe the system services more precisely from the external view by decomposing the Brief External Use Case we have got from the previous process. Eventually, writing the Detailed External Use Cases helps us to find goals. The Detailed External Use Case gives out the specific description of the system action, but still from the outside of view. We take one action at a time from the Brief External Use Case, and explore it into detail descriptions, such as decomposing the global goal into sub-goals. The following is the Detailed External Use Case format, which can systematically thrill down and assistant software developers to generate the detailed descriptions:
<table>
<thead>
<tr>
<th>Stakeholders</th>
<th>(All entities related to the system)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>(The always true conditions before processing this use case)</td>
</tr>
<tr>
<td>Postconditions</td>
<td>(The always true conditions after successful completion of this use case)</td>
</tr>
<tr>
<td>Main Success</td>
<td>(The typical successful actions)</td>
</tr>
<tr>
<td>Scenario</td>
<td></td>
</tr>
<tr>
<td>Extensions</td>
<td>(The alternate actions)</td>
</tr>
</tbody>
</table>

**Case Study**

<table>
<thead>
<tr>
<th>Use Case</th>
<th>HotNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Actor</td>
<td>The player</td>
</tr>
<tr>
<td>Stakeholders</td>
<td>The player</td>
</tr>
<tr>
<td></td>
<td>The CSLFS</td>
</tr>
<tr>
<td>Preconditions</td>
<td>1. The CSLFS has been connected to the Internet.</td>
</tr>
<tr>
<td></td>
<td>2. The CSLFS has been connected to the database.</td>
</tr>
<tr>
<td>Postconditions</td>
<td>All hot numbers are displayed on screen</td>
</tr>
<tr>
<td>Main Success</td>
<td>1. The player requests the CSLFS for the Hot Number Service.</td>
</tr>
<tr>
<td>Scenario</td>
<td>2. The Hot Number Service displays all hot numbers on the screen.</td>
</tr>
<tr>
<td>Extensions</td>
<td>None</td>
</tr>
</tbody>
</table>

### 4.3 Step 5: Structuring Goals

This step is to map the goals, derived from Step 3 and Step 4, into the Goal Hierarchy Diagram. Since each goal is different in size, importance, and the level of complication,
the analysts should carefully analyze such difference and pay more attention to their importance and inter-relationships. Then, the Goal Hierarchy Diagram should present these relationships and classify goals into different catalogs based on the level of complication and importance.

In order to draw the Goal Hierarchy Diagram, the following two steps should be processed. The first step is to recognize the overall system goal and treat it as the highest priority. To accomplish this task, we simply go back to the Step 3, and retrieve the overall system goal from the Brief External Use Case. Such task will be more complex when a system has more than one main goal. In the case like that, an overarching system goal should be structured to cover those main goals.

Goals need breaking down into smaller sub-goals to the reasonable points. There are no specific rules to guide how far the analyst should decompose goals. The general rule to handle such dilemma is to break down the goal too far rather than not far enough. To get the sub-goals for the main goal, what we do is to look back the Step 4 and retrieve the sub-goals from the Detailed Brief External Use Case. Now, we can form the Goal Hierarchy Diagram.

**Case Study**

![Goal Hierarchy Diagram]

5. **System Design**

From this point, we shift from the analysis to the design phase. During agent-based design, there is an emphasis on defining software agents and how they collaborate to fulfill the requirements.
As we know, the design phase is the most difficult part in the software engineering. It involves elements of both science and art. It is particularly challenging to the software engineers when the task is to design an entirely new system. Unfortunately, this is the common circumstance in software engineering. To solve this difficulty, engineers need a synthesis of experience and knowledge, in other words, the pattern. The patterns facilitate human comprehension of the observations and the subsequent development of a scientific theory. Not surprisingly, in our approach, we will utilize some agent patterns suggested by Einhorn and Jo [7].

5.1 Step 6: Internal Use Cases

The Internal Use Case concerns interactions among elements inside the system. The reason we call such Use Cases internal Use Cases is because they can show how entities interact to the system internally and how the entities use each other to get things done. The purpose of this step is to identify the intentions (plans). Based on the external use cases, we decompose the scenario of each use case into details. That will help us find intentions for the goals and discover more roles that are not easy to be seen from the external point of view. In order to create the brief internal use cases, we read the external use cases described in Step 4 and decompose their scenarios while keeping the rest of the parts unchanged. In our research, a role includes a particular goal or set of goals and a set of intentions. It will be mapped to an agent who is responsible for satisfying those goals.

Case Study

<table>
<thead>
<tr>
<th>Use Case</th>
<th>HotNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Finding hot numbers</td>
</tr>
<tr>
<td>Primary Actor</td>
<td>The player</td>
</tr>
<tr>
<td>Stakeholders</td>
<td>The player</td>
</tr>
<tr>
<td></td>
<td>CSLFS (wants to display hot numbers to the player)</td>
</tr>
<tr>
<td>Framework For BDI Agent-Based Software Engineering</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HotNumberManager</th>
<th>PastWinningNumberManager</th>
</tr>
</thead>
</table>

**Preconditions**
The CSLFS has been connected to the Internet.

**Postconditions**
All hot numbers are displayed on screen.

**Main Success Scenario**
1. The player requests the CSLFS for the Hot Number Service.
2. The CSLFS launches the Hot Number service request.
3. The HotNumberManager asks for the current numbers from the CurrentNumberManager.
4. The CurrentNumberManager returns the current numbers to the Hot Number Manager.
5. The HotNumberManager asks for all past winning numbers from the PastWinningNumberManager.
6. The PastWinningNumberManager returns all past winning numbers to the HotNumberManager.
7. The HotNumberManager applies some algorithms to all the numbers.
8. The HotNumberManager returns hot numbers to the CSLFS.
9. The CSLFS displays all hot numbers on the screen.

**Extensions**
5a. The current numbers are the same as the latest result.
5a.1. The CurrentNumberManager will supply nothing to the HotNumberManager.
5.2 Step 7: Sequence Diagram

Based upon the Use Case scenarios, the Sequence Diagram illustrates the sequence of events that are transmitted and the relationship between roles (a role includes a particular goal or set of goals and a set of intentions). It shows how the roles communicate with one another over time. The role that initiates the communication becomes the initiator, and the receiver becomes the responder. These roles are candidate agents. They will become agents when we discover beliefs and assign the beliefs to them.

The use cases can be transformed systematically to the Sequence Diagram. Every entity decrypted in the use cases can be mapped into a role while any communication or information passing between use case entities becomes an event. Every participant in a Sequence Diagram becomes a role. The sequence of the events is based on the use case description. In our agent-based modeling, the diagram helps us to specify the sets of actions that will make up our plans. Normally, we create one Sequence Diagram for each use case. However, if a use case has several alternate use cases, then multiple Sequence Diagrams may be created. After identifying the participating roles, we should draw the arrow lines for all events in the use cases.

Case Study
5.3 Step 8: Agent Activity Diagram

An Agent Activity Diagram, or Agent Event Diagram, expresses operations and the events that triggered agents [18]. Operations are processes that can be requested as a unit and can carry out the state changes. Events define state changes that result from operations and invoke other operations via triggers. The Agent Activity Diagram is much like the flowcharts of old. It shows steps (called activities) as well as decision points and branches [19]. In UML, an Activity Diagram talks more about these transitions and events causing the changes. It is an extension of the State Diagram. State Diagrams highlight states and represent events as arrows between states. Activity Diagrams put the spotlight on the events.

Our Agent Activity Diagram is derived from the UML Activity Diagram. Similar to the UML Activity Diagram, the Agent Activity Diagram depicts all events that trigger those roles, and operations that are processed by those roles. We call those potential agents as roles at this moment because they have only been assigned goals. Since an agent plan is a set of activities, we can collect the events and operations from the Agent Activity Diagram and save them as intentions for future use.

To construct the Agent Activity Diagram is quite systematic. What we do is to retrieve each specification of action from the scenarios of the External Use Cases generated in
both step 3 and step 4 and the Internal Use Cases generated in step 6 as well as the roles of the Sequence Diagram in step 7. If we find some activities that take place at the same time, analysts just use the fork arrow line to connect them. Please do not forget to add some descriptions into the Internal Use Case when you discover the necessary activities, or delete some descriptions when you think they are unnecessary.

Case Study

Agent Activity Diagram for HotNumberManager
5.4 Step 9: Agent Belief List

An agent’s beliefs are a set of data describing the state of the environment. They are the knowledge that intentions use to fulfill their goals (desires). In order to find those data that agents need, we apply the Data Flow Diagram (DFD) in our approach. DFD is a functional methodology, which is also popularized in OO analysis and design. Data Flow Diagrams show the flow of data from external entities into the system, and how the data moved from one process to another, as well as its logical storage.

Simply based on the use cases in Step 3, Step 4 and Step 6, we generate the DFD and collect the data flows between those processes. These data are the Beliefs we need in our Agent Cards. This is the most important job in the entire BDI agent software development process. After we finish this step, we get all three necessary parts of agents: Desires, Intentions, and Beliefs. Then, we can form the completed BDI agents.

The purpose and value of the Data Flow Diagram is primarily data discovery, not process mapping. We utilize the Data Flow Diagram in our agent software development process because the DFD can help us find the data moving around in the environment.

We use the step-wise refinement approach, starting from the context diagram and then the next level. To create the DFD, we work through Use Case scenarios, described in Step 6. We follow the logic of the Use Cases, concentrating on how the data is processed by each step.

After we build up the DFD, we collect these data and insert them into the Belief Form, one Belief Form for each Use Case:

Case Study
5.5 Step 10: BDI Agent Cards

The BDI Agent Cards are the collecting formats to bring together all parts of agents into entities: agents, beliefs, desires and intentions. At previous steps, we have already figured out the materials we demanded. Here, we just collect them and form them into groups based on the goals we try to accomplish. The BDI Agent Cards are the valuable tools that can be used in the construction of agent-based software. After the construction of the BDI Agent Cards, assigning the goals to agents is completed. Also, we have to assign the agent name. Usually, we give the agent name as same as the role name. We can assign a different name if we think it is necessary. The following is the format of the BDI Agent Cards:

<table>
<thead>
<tr>
<th>Use Case</th>
<th>HotNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beliefs</td>
<td>HotNumber_Request</td>
</tr>
<tr>
<td></td>
<td>Current_Numbers</td>
</tr>
<tr>
<td></td>
<td>Past_Winning_Numbers</td>
</tr>
<tr>
<td></td>
<td>Hot_Numbers</td>
</tr>
</tbody>
</table>
The end product of this phase is an Agent Class Diagram. The Agent in the Agent Class Diagram was designed, as the following figure. It contains the agent name, desire, belief, and intention. The Agent Class Diagram reflects the relationship between agents.

**Case Study**

<table>
<thead>
<tr>
<th>Agent:</th>
<th>CSFLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDI list:</td>
<td>Belief:</td>
</tr>
<tr>
<td></td>
<td>Current_Numbers</td>
</tr>
<tr>
<td></td>
<td>HotNumber_Request</td>
</tr>
<tr>
<td></td>
<td>SearchNumber_Request</td>
</tr>
<tr>
<td></td>
<td>Expected_Numbers</td>
</tr>
<tr>
<td></td>
<td>Hot_Numbers</td>
</tr>
<tr>
<td></td>
<td>Search_Results</td>
</tr>
<tr>
<td></td>
<td>Desire: Assisting Player</td>
</tr>
<tr>
<td></td>
<td>Intention: RequestHotNumber( )</td>
</tr>
<tr>
<td></td>
<td>ReturnHotNumber( )</td>
</tr>
<tr>
<td></td>
<td>DisplayHotNumber( )</td>
</tr>
<tr>
<td></td>
<td>RequestSearchNumber( )</td>
</tr>
<tr>
<td></td>
<td>DisplayForm( )</td>
</tr>
<tr>
<td></td>
<td>InputNumber( )</td>
</tr>
<tr>
<td></td>
<td>ReturnSearchResult( )</td>
</tr>
<tr>
<td></td>
<td>DisplaySearchResult( )</td>
</tr>
</tbody>
</table>
5.6 Step 11: Implementation

Implementation is about to code the system based on the artifacts constructed in the previous steps. It is pretty straightforward, because we have already known the agent names, beliefs, desires and intentions. If we discover new agents in this phase, we have to go back to modify all previous artifacts and make them consistent.

The programming language used in our approach is the Agent Programming Language (APL) [11] [23]. With this language, we can define the necessary agents and their BDIs, and then define the Belief, Desire, and Intention respectively. The structures of defining the agent, Belief, Desire, and Intention are given in the following figures.

```java
agent [Agent Name] extends BDI {
    belief [Belief Name] ;
    desire [Desire Name] ;
    intention [Intention Name] ;
    main( ) {...} //main body to execute on this agent
}
```
Belief [Belief Name] extends BDI {
    [Belief declaration]
    : //some other belief declarations
    :
    [name of accessor for belief];
    : //some other accessors for belief
    [name of modifier for belief];
    : //some other modifiers for belief
}

desire [Desire Name] extends BDI {
    [Desire Name]( ) { … }
    : //some other goal names
    :
}

intention [Intention Name] extends BDI {
    [Intention Name]( );
    : //some other intention names
    :
}

6. BDI Agent Modeling CASE TOOL

Based on the BDI Agent-based Modeling Technique (BDI AMT) introduced in the previous sections, we are developing a CASE tool [Figure 4]. The BDI agent-based
modeling technique CASE tool (BDI AMT CASE Tool) consists of a front-end which helps software engineers translates models in the textual/XML format into Java programs as output.

While the existing CASE tools cannot support BDI agent modeling concept, our BDI AMT CASE Tool is supposed to support BDI agent modeling technique for entire software life cycle.

The front-end of the BDI CASE tool consists of many modules to help software engineer construct many different kinds of artifacts such as textual/form editors for initial problem statements, enterprise software assessment, external use cases, internal use cases, agent lists, agent belief lists, and graphic editors for agent relationship diagrams, agent interaction diagrams, and BDI agent cards. The textual and graphical models are stored in the XML format based on the SVG format that is a widely used to describe graphical objects. Through the same format, models could be shared by different tools and from different sites via collaboration tools.

The back-end of the BDI CASE tool has been designed to read the models in the textual/XML format as input and translates into the Java programs as output.
One of other research related to the BDI agent computing is a development the BDI Agent-based Programming Language (BDI APL) [11] [23]. We plan to extend our back-end to support APL and Java in the future. We have implemented a prototype of the front-end, but the back-end is not completed.

Another supportive research of ours is a collaborative work tool for the BDI CASE tool. Such a collaborative work tool is to support multiple software engineers to co-work to design a BDI agent system while connecting each other from different sites. There are two different concepts involved in this tool such as ownership and authorship. The ownership declares who is an owner of a model collaboratively constructed. An authorship concept was introduced to allow an author to edit a model one at a time while other engineers can look at the same model from their own editor windows. The authorship is transferable and an author could be changed at run time. We have implemented a collaborative work tool. The BDI agent-based CASE tool will be available in public after its publication from our research web page at http://ecs.fullerton.edu/~jo/research.

7. Conclusions

Although the agent-based computing has been discussed for a few years, it is still in its babyhood. Remember, as an agent, it should be able to learn from its environment and manipulate itself to coordinate with others. However, our current BDI agent model does not support these kinds of complex tasks. Thus, how to let the BDI agent to learn and adapt could be the future research work. Support for mobile and multi-agents are good area to clarify.

Providing a detailed process to support a full life-cycle of software and a supporting tool is very important to realize the idea we suggest here. We will continuously refine our modeling techniques based on BDI agents, and implement supporting tools. We believe that agent technology, especially intelligent agents, is one of the most promising ideas for reducing the complexity of software engineering. It will change the way we develop software.
Acknowledgement

A condensed version of this paper [13] was presented at the ACM Symposium on Applied Computing (ACM SAC 2004), Software Engineering Track.

References

A Framework For BDI Agent-Based Software Engineering


Appendix

A case study is provided to show how to use our process suggested here to develop a BDI agent-based application such as the California Supper Lotto Finding System (CSLFS) using the BDI Agent Software Development Process. Because of the limitation of the paper length, we provide a detailed case study on our research web page at http://jo.ecs.fullerton.edu/research/GuibinChen2003/GuobinChen2003-CaseStudy.pdf.

Authors addresses:

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