Case Study
Build an Application Using BDI Agent Software Development Process

Guobin Chen
Department of Computer Science
California State University, Fullerton
Fullerton, CA 92834-6870
714-278-7255
jo@ecs.fullerton.edu

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

ABSTRACT

This project is to present a practical software development process, which is useful to agent-based software engineering. Since an agent is a new concept in software engineering area, there is not much matured approach to analyze and design agents. Therefore, analysts currently adopt both extended versions of OO and functional techniques in most agent-based software engineering technique.

We propose here an agent-based software development process based on the BDI agent concepts. While another approach of our work [Einhorn 2002] extracts intention, desire, and belief by introducing two valuable extended use cases, external and internal use cases. In this approach, we extract goals from the external use cases, and then find corresponding intentions and belief by using internal use cases and other supportive tools such as sequence diagrams, activity diagrams and dataflow diagrams. Following this approach, software engineers can systematically extract desire, intention and belief, and then assign the BDIs to agents to make them be flesh in our agent-based system.

The BDI Agent

In this project, analysts adopt the belief-desire-intention (BDI) rational agent model to define our system. An agent is concurrent, autonomous, intelligent and self-contained object [Jo 2002]. The BDI model gets its name from the fact that it recognizes the primacy of beliefs, desires, and intentions in rational action. Jo explained very well in his research paper:

> "Agents have explicit goals to achieve or events to handle (desires). A set of plans (intentions) is used to describe how agents achieve their goals. Each plan describes how to achieve a goal under varying environments (beliefs). A set of data called belief describes the state of the environment." [Jo 2001]

Notice, analysts use Agent-based approach in our work, not the Agent-oriented approach.

> "Agent-based programming is programming based on agents. A program consists of a set of agents and their collaboration. Agent-oriented approach involves agents that learn themselves by experience and adapt themselves based on the previous learning to the current environment."[Jo 2001]

Thus, our BDI Agent model does not support learning and adaptation.

Process Description

As analysts know, the simplest and easiest software engineering life cycle model is the Waterfall model. Almost all of the software engineering methodologies are generated from it. Similarly, our approach is also depended on it, but only includes the requirement, analysis, and design. This approach is designed to identify the Beliefs, Desires, and Intentions for agents during the software analysis and design phases. Analysts do not involve the implementation, testing, and maintenance phases.

In the real world, people usually establish the goals in the beginning. Under these goals, they work out correspondent plans in order to fulfilling 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 words, people need to know the data to describe the world.

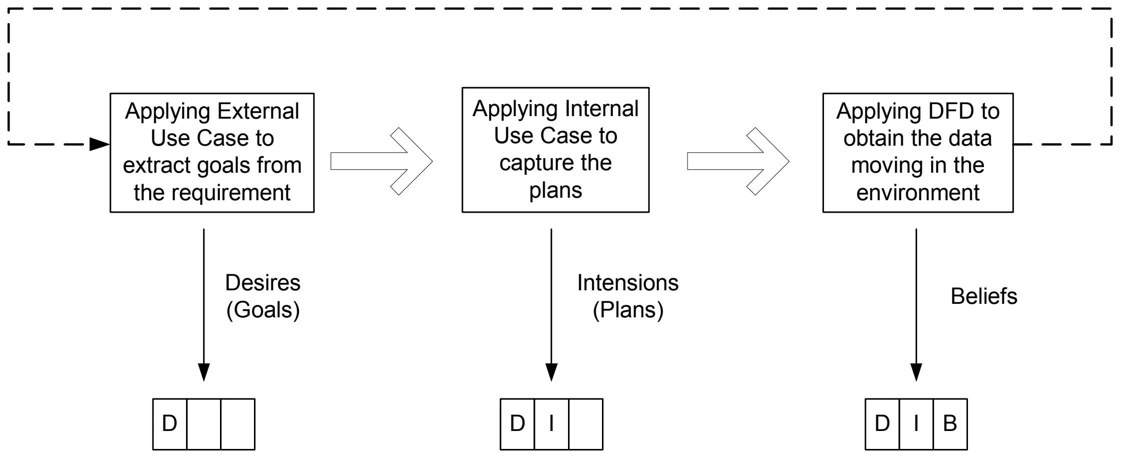
Applying External Use Case to extract goals from the requirement
Applying Internal Use Case to capture the plans
Applying DFD to obtain the data moving in the environment
Desires (Goals)
Intensions (Plans)
Beliefs
D
D I
D I B

FIGURE 2. OVERVIEW OF THE DEVELOPMENT PROCESS

Following the natural style of human thought: goal-plan-data, our approach will extract the desires from the requirement first, and then create the proper intentions. At last analysts will find the beliefs. More specifically, analysts are going to apply the External Use Case to extract goals from the requirement primarily, the Internal Use Case to capture the plans in the following, and the Data Flow Diagram to obtain the data moving in the environment at the end. Please notice that the software analysis and design is not one time event. It should be processed and modified iteratively. Therefore, analysts may return to the beginning, as analysts need. The high level view of developing BDI agents is depicted Figure 2.

In the detail, the BDI Agent Software Development Process (BDI-ASDP) embraces eight sub-phases: Initial Problem Statement, Enterprise Software Assessment, Brief External Use Case, Detailed External Use Case, Internal Use Case, Sequence Diagram, Agent Activity Diagram, Agent Belief List, and BDI Agent Cards.

Our approach emphasizes on the analysis and design phase. In the requirement analysis phase, analysts 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. Next is the design phase. In this phase, analysts utilize the internal use case to capture the intentions and data flow diagram to find the beliefs. Finally, analysts discovery the BDI and therefore, can form the agents.

The BDI-ASPD is a specialization of traditional software engineering methodologies. The general phases and steps are shown on the Figure 3.

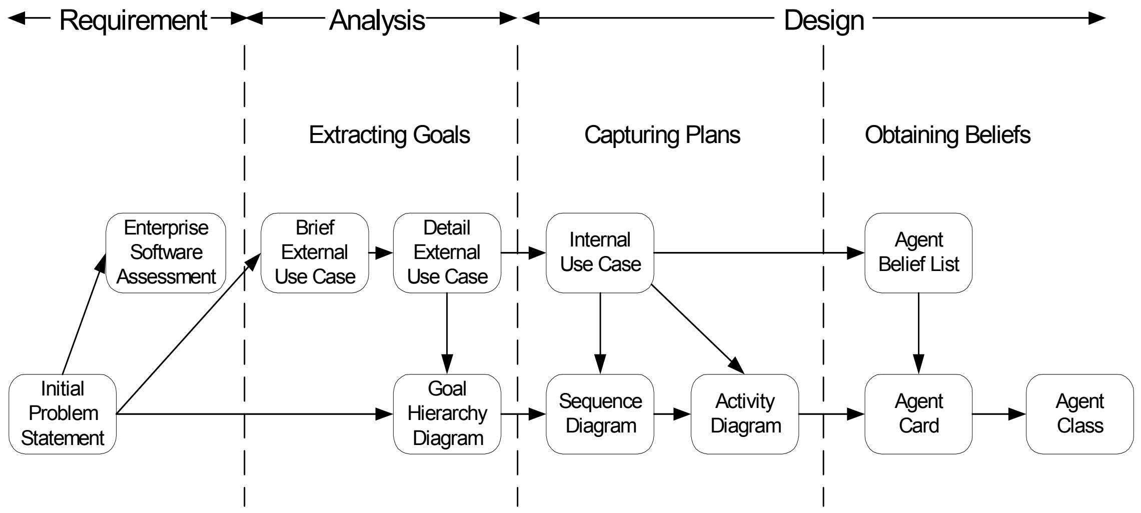

FIGURE 3. BDI AGENT DEVELOPMENT PROCESS

While analysts have drawn it as a single flow from left to right, in practice the methodology is iterative. The intention is that the analyst or designer may move between steps and phases freely and that each successive pass will produce additional detail providing a complete, yet consistent system design.

A major strength of BDI-ASPD is the ability to track changes throughout the process. Every object created during the analysis and design phases can be traced forward or backward through the different steps to other related objects. For instance, a goal derived in the Brief External Use Case step can be traced to a specific role or agent card. Likewise, an agent card can be traced back through roles to the system level goal it was designed to satisfy.

REQUIREMENT

The requirement is not the emphasis part of our BDI Agent Software Development Process. Analysts include it in our approach, as it is the fundamental part of our system. It will assist us to understand the system and have a clear thought on how to construct it.

Step 1 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.

Software developers usually construct the Initial Problem Statement based on interviewing and questioning customers, and/or studying business documents. Because our approach focuses on the analysis and design phase, analysts will not describe how to take requirement here.

<u>Case Study</u>

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 a type of stand-alone entertainment software to be implemented using Agent Oriented 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 another 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 assistant 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**

  Introduction (Section 1) provides an overview of the entire Initial Problem Statement document.

  General Description (Section 2) describes the product in overview, addressing product functions, characteristic of user, general constraints, and assumptions and dependencies.

  Functional Requirement (Section 3) describes the various functions of the software

- **1.4 Abbreviations**

  IPS: This is the Initial Problem Statement

  CSLFS: California Super Lotto Finding System

  BDI-ASDP: BDI Agent Software Development Process

APL: Agent Programming Language

- **1.5 References**

  ANSI/IEEE Std 830-1984, IEEE Standard for Software Requirement Specification.

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 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.

3.1.2.2 Input

A draw of numbers including 5 winning numbers and one mega number.

3.1.2.3 Process

The system will match the draw of numbers that the player provided, with those numbers storing in the database.

3.1.2.4 Output

The combination of numbers and times will be displayed to the player.

- **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.

  - **3.2.2 Hardware Interface**

    The system will run on any standard pc hardware.

  - **3.2.3 Software Interface**

    This project will be build using APL Agent Programming Language with a Graphical User Interface (GUI). Hence, the operating system should have the capability to support this application program.

- **3.2.4 Communication Interface**

  The communication interface should be versatile enough in order to support pluggable interface feature.

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. He does not totally trust the computer. So, he selects his own two numbers combining with the machine selecting numbers. 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. Then, he makes his final decision and leaves.

Step 2 Enterprise Software Assessment

The Enterprise Software Assessment describes how the new system would impact the current one. This process basically illustrates on what environment the current system runs. Is the old system run on Windows or Macintosh? What kind of language, compiler or debug tool does it use? Can analysts still use the old database? The bottom line is: Is there any problem when the customer applies the new system on its ancestor, and how to solve it if there is any? By looking at the previous issues analysts probably can get a better idea of how the system might be implemented for the customer.

Case Study

There is not any requirement or constrain to the CSLFS, because the constructing system is the first version. The new software will not impact any ancestor.

SYSTEM ANALYSIS (Capturing Goals)

The next two sections, the Brief External Use Case and the Detailed Use Case, belong to the analysis phase. Analysis phase emphasizes an *investigation* of the problem and requirements, rather than a solution [Larman 2002]. During analysis phase, analysts emphasize on finding the first element of agents—Desire, in the problem domain.

Use Case is a functional description of the entire system. The reason analysts adopt it as the tool for the analysis phase because of the following features:

- Use Cases are the main tasks performed by the users of the system.
- Use Cases describe the behavioral aspects of the system.
- Use Cases are used to identify how the system will be used.
- Use Cases are a convenient way to document the functions that the system must support.
- Use Cases are used to identify the components (agents) of the system.

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 analysts 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.

A goal is an abstraction of functional requirements (without losing the essence of the requirement) and is kept in the system-level context. It is always defined as a system-level objective; lower-level constructs may inherit or be responsible for goals. In consequence, every action within a system must support a particular goal. Usually, “Goals can be extracted from *what* the system is trying to achieve and generally remain constantly throughout the entire analysis and design process” [DeLoach and Wood 2000].

The primary task of the Brief External Use Case is to utilize *Brief Use Case* technique (Black-Box Use Case [Larman 2002]) to capture the system goals. The Capturing Goals starts from the initial problem statements. Analysts break them down into a set of brief use cases, and then, extract one system goal from each brief use case.

In order to finding the system goals, analysts adopt the *Brief Use Case* technique in the analysis phase, because not only it 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 [Fantechi et al 2003]. If there is a use case captured from the system requirements, analysts 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, usually person or other systems, which trigger the system behavior to achieve a certain goals.

Goals should be specified as abstractly as possible without losing the essence of the requirement. An analyst can perform this abstraction by removing detailed information when specifying goals [DeLoach 2000]. 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 is a requirement that may change with time or between various operating systems and should not be included as a goal or sub-goal.

Our approach for the development of these BDI agent models begins from the services provided by the system. Following the threads of services, analysts can define their purposes, and determine the top-level goals that the agents must be able to achieve.

During the Brief External Use Case process, analysts identify the services of the system being developed from an external point of view; analysts do not describe the internal workings of the system, its components, or design. In this section analysts determine what services our system should provide. As analysis is developing, analysts can treat the whole system as a black box, and all outside entities as external agents who interact with the system. Then, from the view of external agents, analysts decompose the system into use cases, one for each service. Thus, the Brief External Use Case captures *who* (actor) does *what* (interaction) with the system, for what *purpose* (goal), without dealing with system internals. Analysts describe each brief external use case with Functionality and a one-paragraph Scenario.

<u>Case Study</u>

By studying the Initial Problem Statement in the previous case study, analysts find that the overall of the California Super Lotto Finding System (CSLFS) is to assistant a player to play the California Super Lotto. It has two services or functionalities to meet this goal. One is "find hot numbers", and the other is "search favorite numbers". If the Initial Problem Statement is too general to find the existed scenarios for these goals, analysts must ask the user to provide scenarios for them. In our case, analysts construct the scenarios according to the Initial Problem Statement. Analysts get the artifact of the External Use Case in following:

Assistant a player to play the California Super Lotto
1. A player asks the CSLFS for the Hot Number service. 2. The CSLFS displays a group of numbers on the screen. 3. The player clicks the Search Number button. 4. The CSLFS pops up a form. 5. The player inputs a draw of numbers and submits them. 6. The CSLFS displays the search result on the screen.

Find hot numbers

1. A player clicks the Hot Number button. 2. The CSLFS displays a group of numbers on the screen.

Search if the player's favorite numbers have ever won
1. The player clicks the Search Number button. 2. The CSLFS pops up a form. 3. The player inputs a draw of numbers and submits them. 4. The CSLFS displays the search result on the screen.

After this step, analysts abstract three goals:

- Assistant a player to play
- Find hot numbers
- Search the player's favorite numbers

Step 4 Detailed External Use Cases

The purpose of the Detailed External Use Case is utilizing the *Fully Dressed Use Case* technique [Larman 2002] to help us better understand structure of the system. It decomposes the Brief External Use Case analysts have got from the previous process into precise details. The Detailed External Use Case gives out the specific description of the system action, but still from the outside of view. The principles of the use case analysts adopt in this project are based on Larman's theory [Larman 2002]. In this step, analysts take one service at a time, and explore it into detail description, in that analysts can make a plan for each service. The following is the Detailed External Use Case format, which can systematically thrill down and assistant software developers to generate detailed descriptions:

Use Case	
Goal	
Primary Actor	
Stakeholders	
Preconditions	
Postconditions	
Main Success Scenario	
Extensions	

Figure 4. Detail External Use Case Form

Let us explain the sections in the Detail Use Case Format first. The External Use Case gives the use case name. Usually, analysts can transform the short phrase in the Functionality of the Brief External Use Case into a noun term.

In the Primary Actor section, analysts write down the name of initial actor who triggers the system behavior.

Stakeholders are those who have an interest in a particular decision, either as individuals or representatives of a group). This includes people who influence a decision, or *can* influence it, as well as those affected by it. The stakeholders suggest and bound what the system must do. Cockburn states in his book:

> The [system] operates a contract between stakeholders, with the use cases detailing the behavioral parts of that contract… The use case, as the contract for behavior, captures *all and only* the behaviors related to satisfying the stakeholders' interests [Cockbun 01].

When analysts find any action that satisfies the stakeholders' interests, analysts should record it in the use case.

In the preconditions, analysts can state what must always be true before beginning a scenario in the use case. Analysts do not need to test these conditions that they are true; rather, analysts just assume that they are true. Usually, a precondition is the successful result of the previous use case.

The postconditions state what must be true after successful completion of the use case, which are for either the main scenario or some alternate path. The postconditions should meet the needs of all stakeholders.

The main success scenario describes the typical success path that satisfies the interests of the stakeholders. The step one always indicates the trigger event that starts the scenario. The scenario records the interaction between actors, the validation, and the state changed by the system.

Extensions state all the alternative scenarios, either success or failure. It is common that the extension section has longer and more complex context than the main success scenario.

<u>Case Study</u>

At the beginning, analysts need to name each of the external use cases. By looking back the artifacts analysts have got in the last step, analysts have already had the functionalities of "Find hot numbers" and "Search if the player's favorite numbers have ever won". Hence, analysts name our external use case as "HotNumber" and "SearchNumber". Because the "player" is the initial trigger of the system behavior, obviously it is the Primary Actor for both External use cases. At this point, the player is the only person who is interested in the hot numbers and searching numbers. So, it should be the only stakeholder in these scenarios.

Before the HotNumber scenario happens, the system should have been run and the player should launches his requirement. These minimum requirements are the preconditions.

After the system process the player's query, all hot numbers should been displayed on the screen. That could be the postcondition. Since the external use case decrypts system behaviors only from the view of outside of system, the story is quite simple: the player request the HotNumber Service, and then the HotNumber Service displays all hot numbers on the screen. Every time if the player issues the HotNumber Service, the system will display all hot numbers. There will not have any exception.

The differences between the HotNumber and the SearchNumber are the Preconditions, Scenario, and Extensions. Before the SearchNumber processes, it must obtain five winning numbers and one mega number. So, inputting five winning numbers and one mega number is the precondition. Because analysts need to input numbers, the system has to pop up a form for input after the player request SearchNumber service. When the player enters numbers, the system processes those numbers and displays the results. Analysts get two more actions in the scenario part of the SearchNumber. If the player inputs numbers incompletely or out of range, the system will request re-enter numbers.

Use Case	HotNumber
Goal	Find hot numbers
Primary Actor	The player
Stakeholders	The player
Preconditions	The system has been run The player launches his requirement
Postconditions	All hot numbers are displayed on screen

Main Success Scenario	1. The CSLFS launches the Hot Number Service 2. The Hot Number Service displays all hot numbers on the screen
Extensions	None

Use Case	SearchNumber
Goal	Search favorite numbers
Primary Actor	The player
Stakeholders	The player
Preconditions	The results in the search format are displayed on screen.
Postconditions	The results in the search format are displayed on screen
Main Success Scenario	1. The CSLFS launches the Search Number Service. 2. The Search Number Service provides the input form. 3. The player inputs expected five winning numbers and one mega number. 4. The Search Number Service processes the query. 5. The Search Number Service displays the search results on the screen.
Extensions	2a. Submitted data is incomplete: 2a1. The system requires missing numbers 2a2. The player provides the missing numbers

Step 5 Structuring Goals

This step is to map the goals into a Goal Hierarchy Diagram. Since each goal is different in size, importance, the level of complication, the analysts should carefully analyze such different 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 accomplish the Goal Hierarchy Diagram, the following three steps should be processed. The first step is to recognize the overall system goal and treat it as the highest priority. Such task will be more complex in the occurrence of more than one main goal without a single system goal. In the case like that, an overarching system goal should be structured to cover these goals. Furthermore, each sub goal should possess similar functions with their parents', and just in the lower level of hierarchy.

Goals need breaking down into smaller sub-goals to the reasonable point. The judgment of reasonable point is very subject. There are no specific rules to guide how far the analyst should decompose goals. Analyst always faces the dilemma of too far or not far enough. The general rule to handle such dilemma is to break down the goal too far rather than not far enough. The logic behind this rule is that there is impossible to decompose goals later in the further process if the analyst does not decompose goals enough before. However, if the analyst decomposes goals further than enough, such sub-goals will be automatically put back in the later step.

Some goals are important parts of the system even if they are not in direct support of the primary system goal. For instance, expecting the assistance of dynamic resources, a system may require a goal to find dynamic resources. While this goal does not support the main functional goal of the system, it facilitates the system to meet its other requirements. It is necessary to create and place another "branch" of the Goal Hierarchy Diagram under the primary system goal.

<u>Case Study</u>

Here, analysts list all goals obtained from previous step:

1. Helping a player to play the California Super Lotto number

1.1 Finding the hot numbers

1.1.1 Getting the current result

1.1.1.1 Connecting to web sit http://www.calottery.com/

1.1.1.2 Determining if there is a newest result

1.1.1.3 Returning the newest result

1.1.2 Getting the past winning numbers

1.1.3 Displaying the hot numbers

1.2 Searching the player's favorite numbers

1.2.1 Getting the favorite numbers from the player

1.2.2 Getting the past winning numbers

1.2.3 Displaying the search result

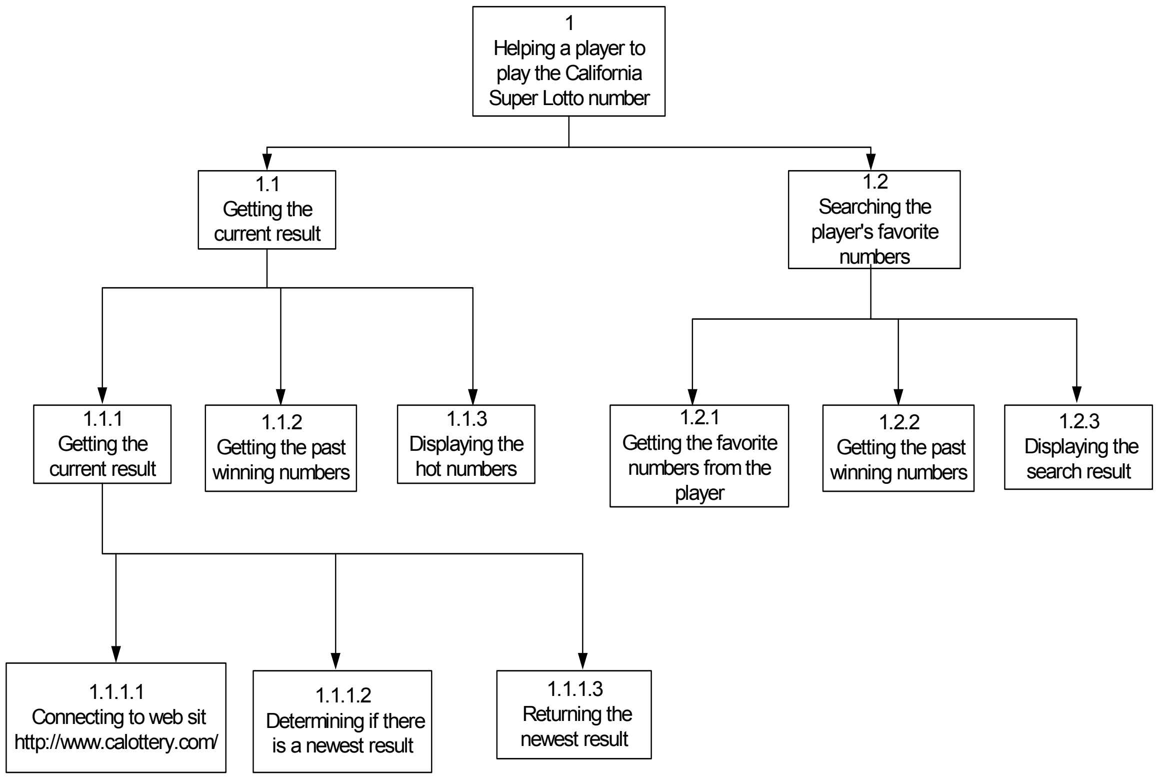
1
Helping a player to play the California Super Lotto number
1.1
Getting the current result
1.2
Searching the player's favorite numbers
1.1.1
Getting the current result
1.1.2
Getting the past winning numbers
1.1.3
Displaying the hot numbers
1.2.1
Getting the favorite numbers from the player
1.2.2
Getting the past winning numbers
1.2.3
Displaying the search result
1.1.1.1
Connecting to web sit http://www.calottery.com/
1.1.1.2
Determining if there is a newest result
1.1.1.3
Returning the newest result

Figure 5. Goal Hierarchy Diagram

SYSTEM DESIGN

From this point, analysts shift from analysis to design phase. Design phase emphasizes a *conceptual solution* that fulfills the requirements, rather than its implementation [Larman 2002]. During agent-oriented design, there is an emphasis on defining software agents and how they collaborate to fulfill the requirements.

Step 6 Internal Use Cases

The Internal Use Case concerns interactions among elements inside the System. The reason analysts call such Use Cases *internal* Use Cases, 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 plans or sub-goals if it is necessary as well as other agents that analysts have not found yet. Based on the external use cases, the scenario of each service is decomposed into details that help us find plans for the goal and discover more agents who are not easy to be seen from external point of view. In order to create the brief internal use cases analysts read the external use cases and decompose the scenarios while keep the rest of parts unchanged. Since this is the design phase, analysts do not have to discover the plans or agents from previous clues. Analysts have to create some plans or agents if analysts think they are necessary.

Design phase is the most difficult process in the software engineering progress, because it needs to create new goals and plans. To help analyst identifies new goals and plans for new roles, analysts provide some agent design patterns that are built upon the object-oriented design patterns created by Einhorn and Jo [Einhorn and Jo, 2003]. They generate three common patterns that are suitable for the agents: agent identification patterns, agent creational patterns, and agent goal assignment patterns.

The agent identification pattern tries to identify common agents in agent software systems, which includes three sub patterns: manager agent pattern, service agent pattern, and broker agent pattern. The manager agent pattern suggests creating a manager agent between the interacted internal and external agents. Hence, the main duty of the manager agent is to communicate the external agent with multiple internal agents. Not only it can locate the proper internal agents based on its interaction with the external agent, but also removes the need for the external agent to know detailed information about internal agents. In large software systems analysts can create several manager agents to bridge the external and internal agents. If a system has multiple manager agents, analysts can build a special manager agent, named delegation agent, above these manager agents.

The service agent pattern guides us to create a service agent based on certain kind of service. For instance, analysts could create a database access agent for a database access service. So, other agents can access database through this database access agent.

The broker agent pattern suggests if several different agents provide similar services, they can register their services with the broker agent. Then they can communicate with the broker agent when they want to use those services. After the broker agent has the request, it will choose the proper service agent to serve the requester.

The second category pattern is the agent creational pattern. It suggests creating an agent while a system needs, such as the long-lived agent. Whenever the system is running, the long-lived agent is needed. It is going to handle the start and shutdown processes for the system. “An agent A should be created by agent B if agent A is the only used by agent B. An agent A should be created by B if agent B has agent A’s initialization data. Agent B should create agent A if it aggregates, contains, records or closely uses agent A.” [Einhorn and Jo, 2003]

The third category pattern is the goal assignment pattern, which suggests us remain the coupling low between agents when analysts assign goals to agents. An agent, with vastly different goals, can be treated as an overly complex agent. It would be better to decompose it into several smaller agents.

Case Study

Now, analysts have to expand the scenarios and explore how the system processes internally in that analysts can identify the plans or find more goals. To provide the hot numbers to the player, the system must obtain the current draw, if there is one available, and all past winning numbers. That means that the HotNumber requires two more services: CurrentResult Service and PastWinningNumber Service. The HotNumber only performs a specific algorithm to generate some hot numbers. It does not care how to retrieve those current result and past winning numbers. In order to accomplish the player's request, the HotNumber needs the assistance from both CurrentResult and PastWinningNumber. Therefore, analysts create two agents based on the service agent pattern: CurrentResult and PastWinningNumber. Here is the complete the HotNumber use case:

Use Case	HotNumber
Goal	Find hot numbers
Primary Actor	The player
Stakeholders	The player (wants to know what numbers might more likely be drawn out next time)
Preconditions	The system has been run The player launches his requirement
Postconditions	All hot numbers are displayed on screen
Main Success Scenario	1. The CSLFS launches the Hot Number Service. 2. The Hot Number Service asks the current result from the Current Result Service. 3. The Current Result Service asks the current numbers from the California Super Lotto web sit 4. The Current Result Service returns the current numbers to the Hot Number Service

	5. The Hot Number Service asks all past winning numbers from the Past Winning Number Service. 6. The Past Winning Number Service returns all past winning numbers to the Hot Number Service 7. The Hot Number Service displays all hot numbers on the screen.
Extensions	4a. The current result is same to the latest result. The Current Result Service will supply nothing to the Hot Number Service.

Similarly, when analysts look into the SearchNumber, analysts find that the SearchNumber needs a draw of numbers and the assistance of the PastWinningNumber to approach its goal. The following is the use case for the SearchNumber.

Use Case	SearchNumber
Goal	Search favorite numbers
Primary Actor	The player
Stakeholders	The player (likes to know if his expected numbers have ever been won)
Preconditions	The player must input five winning numbers and a mega number
Postconditions	The results in the search format are displayed on screen

Main Success Scenario	1. The CSLFS launches the Search Number Service. 2. The Search Number Service provides the input form. 3. The player inputs expected five winning numbers and one mega number. 4. The Search Number Service asks all past winning numbers from the Past Winning Number Service. 5. The Past Winning Number Service returns all past winning numbers to the Hot Number Service 6. The Search Number Service displays the search results on the screen.
Extensions	3a. Submitted data is incomplete: 3a1. The system requires missing numbers 3a2. The player provides the missing numbers

Step 7 Sequence Diagram

From the Use Case scenarios, the Sequence Diagram illustrates the sequence of events that are transmitted and the relationship between roles. It shows how the roles communicate with one another over time.

A sequence diagram is used to determine the minimum set of events that must be passed between roles. There must be a corresponding communication path between them, if an event is passed between two roles. Roles with high frequency communication are suggested to merge into a single agent class. The role that initiates the communication becomes the initiator, and the receiver becomes the responder.

The use cases can be transformed to Sequence Diagram straightly. 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.

Normally, analysts create one Sequence Diagram for each use case. However, if a use case has several alternate resolutions, then multiple Sequence Diagrams may be created. After identifying the participating roles, analysts should draw the arrow lines for all events in the use cases.

Case Study

The Sequence Diagrams created by analyst from the Internal Use Cases are shown below. The use cases have clearly identified participants. In this case, analysts create a "manager" for each service in the Internal Use Case. After identifying the participating roles, analysts create the Sequence Diagram by reading through the use case and finding all instances of an event that occurs between two of the roles. Events identified in the use case are drawn as arrows on the Sequence Diagram in the order they occur.

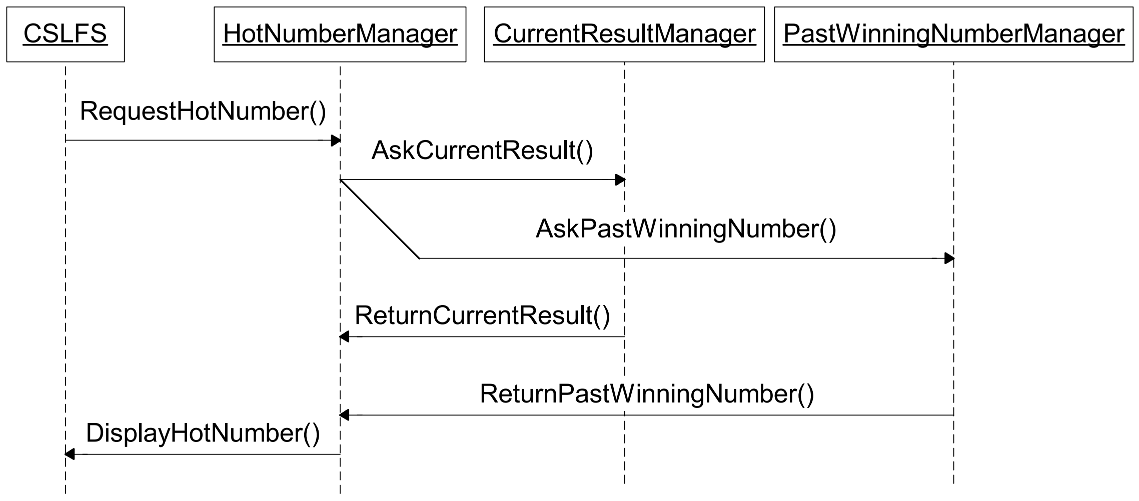

Figure 6. HotNumber Sequence Diagram

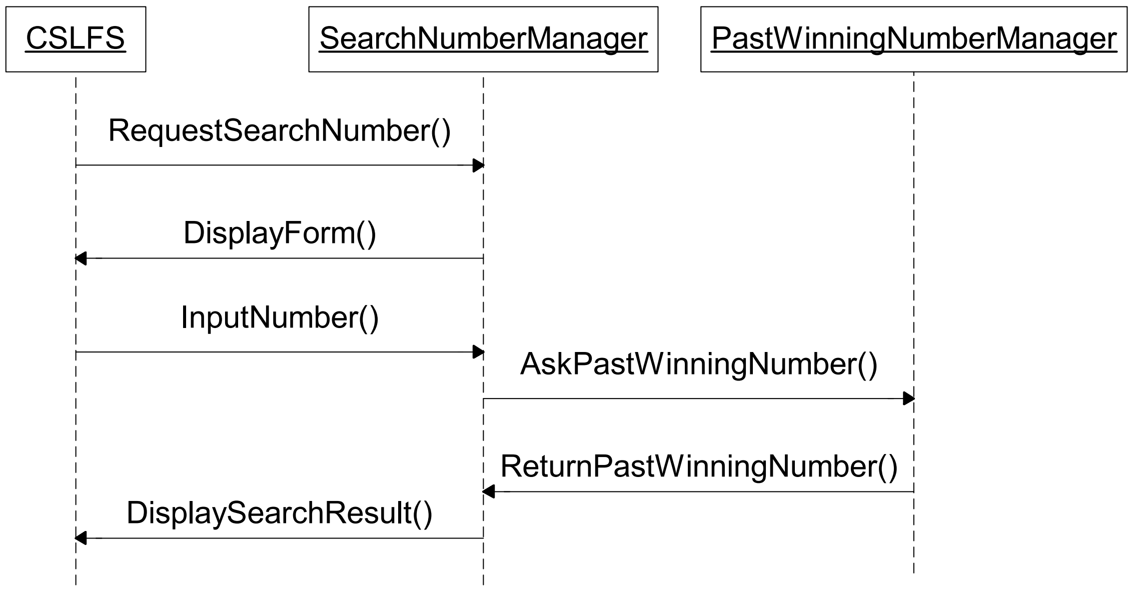

Figure 7. SearchNumber Sequence Diagram

Step 8 Conceptual Agent Activity Diagram

The activity diagram expresses operations and the events that trigged them [Odell 2000]. It is much like the flowcharts of old. It shows steps (called, appropriately enough, ***activities***) as well as decision points and branches [UML 2003].

First and foremost, an activity diagram is designed to be a simplified look at what happens during an operation or a process. The state diagram shows the states of an agent and represents activities as arrows connecting the states. The activity diagram highlights the activities. The activity diagram depicts all activities of those agents. Since an agent plan is a set of actions, analysts can collect the intention from the activity diagram.

The Agent Activity Diagram can help us to further understand the system. It is not necessary to have this step. If the developers feel comfortable, they can simply skip it.

Each activity is represented by a rounded rectangle - narrower and more oval-shaped than the state icon. The processing within an activity goes to completion and then an automatic transmission to the next activity occurs. An arrow represents the transition from one activity to the next. Also an activity diagram has a starting point represented by filled-in circle, and endpoint represented by a bull's eye.

The activity diagram is an extension of the state diagram. State diagrams highlight states and represent activities as arrows between states. Activity diagrams put the spotlight on the activities. Each activity is represented as a rounded rectangle, more oval in appearance than the state icon. The activity diagram uses the same symbols as the state diagram for the startpoint and the endpoint.

<u>Case Study</u>

To construct the activity diagram is quite simple. What analysts do is retrieving the sequences from the previous step, drawing a rounded rectangle for each description, and connecting them with arrow lines. If analysts find some activities take place at same time, analysts just use the fork arrow line to connect them. The HotNumber and SearchNumber activity diagrams are look like the following:

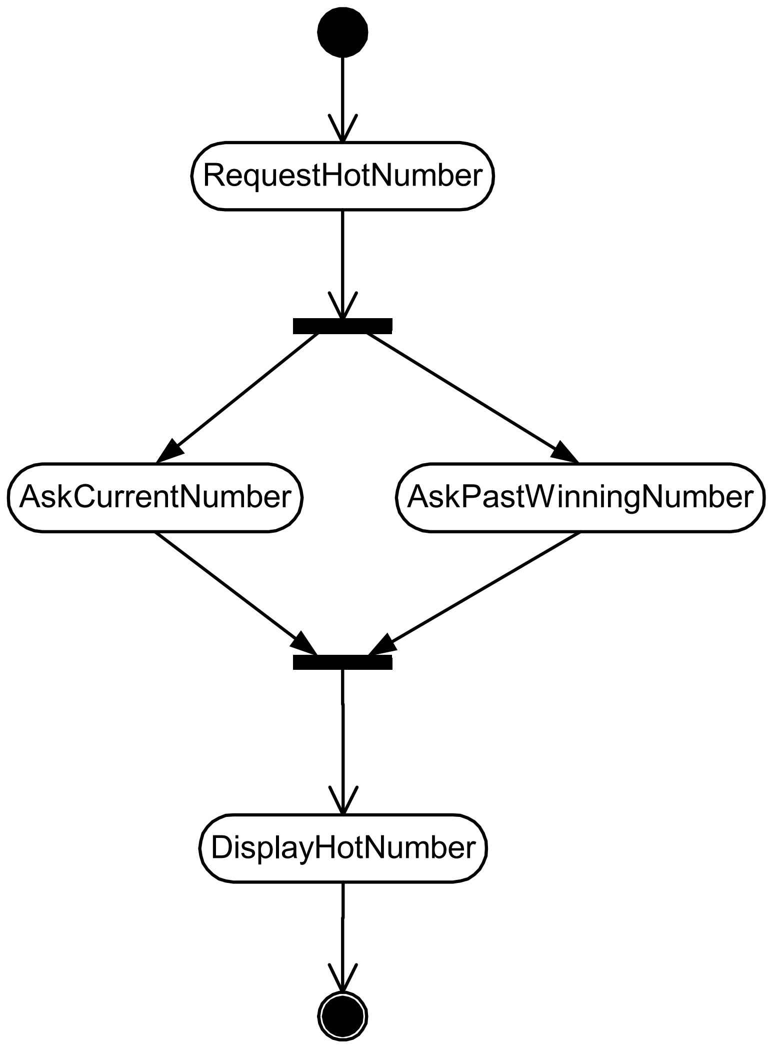

FIGURE 8. ACTIVITY DIAGRAM FOR HOTNUMBER

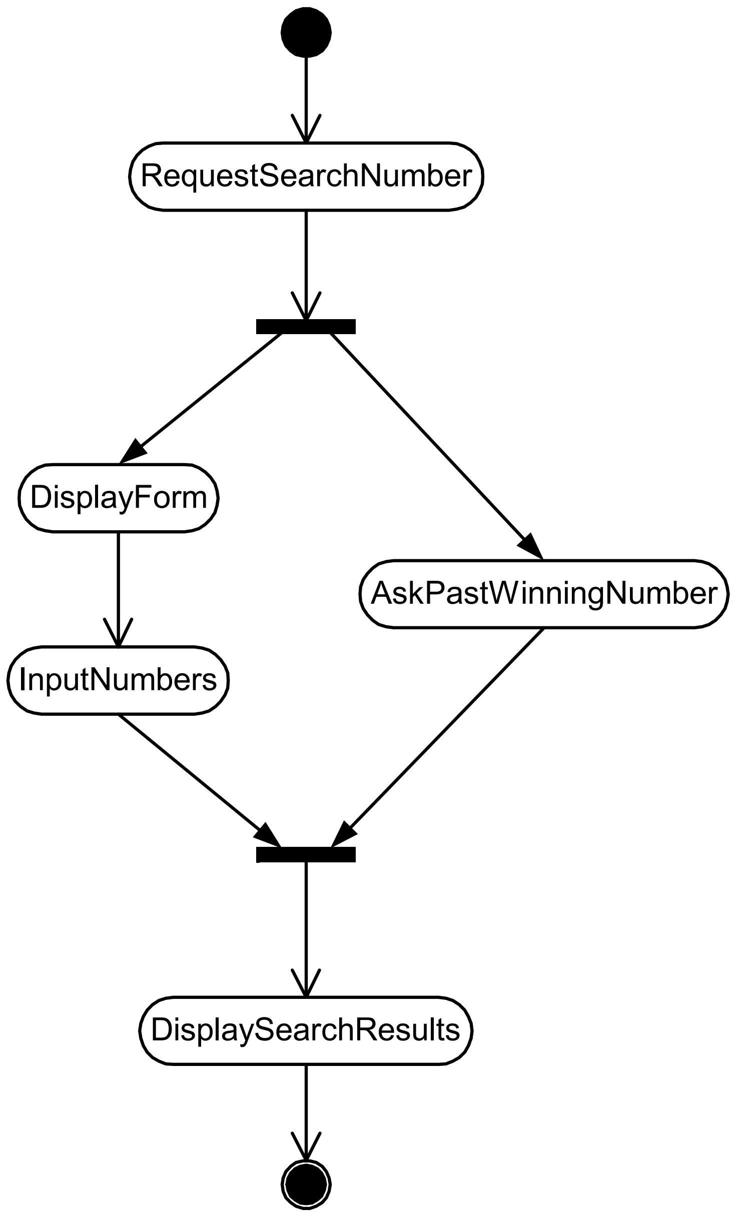

FIGURE 9. ACTIVITY DIAGRAM FOR SEARCHNUMBER

Step 7 Agent Belief List

An agent's *beliefs* are a set of data describing the state of the environment. The basic idea of this step is using the Data Flow Diagram to find out all beliefs that associate to each goal or its sub-goal discovered in the detailed internal use case. This is the most important job in the entire BDI agent software development process. When analysts assign goals to agents, analysts have to discover the proper beliefs for them to build complete agents. Beliefs are the knowledge that intentions use them to fulfill their goals. After analysts finish this step, analysts get all three necessary parts of agents: Desires, Intentions, and Beliefs. Therefore, analysts can form the completed BDI agents.

The purpose and value of the Data Flow Diagram is primarily *data* discovery, not *process* mapping. Analysts utilize the Data Flow Diagram (DFD) in our agent software development process because the DFD can help us to find the data moving around in the environment. Analysts use step-wise refinement approach, starting from the context diagram and then the next level. In the context diagram, which is also named Level 0, analysts depict the actors, the whole system, and the data that flow between them.

In the second step, analysts decompose the system into Level 1. The processes in the Level 1 are from the previous artifact (step 5). The number of goals analysts have in the step 5 is same as the number of processes analysts will have in this step. What analysts have to do is to extract the data flowing between them.

Finally, analysts collect all the data and group them based on the goals. When analysts group the data, analysts will determine what data analysts need to accomplish the goal.

<u>Case Study</u>

In order to get the hot numbers or search results, the player will generate the Hot_Number_Request and Search_Number_Request, and input the Expected_Numbers into the CSLFS. What the CSLFS returns to the player are the Hot_Numbers and Search_Results. The context diagram depicts as following.

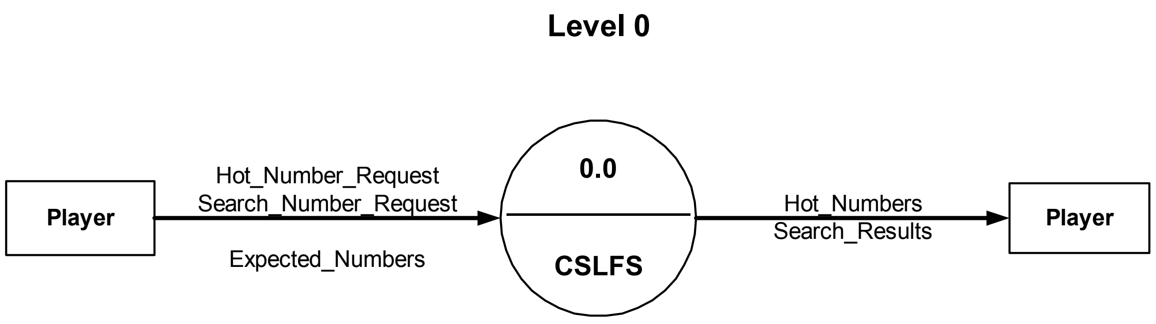

FIGURE 10. CONTEXT DIAGRAM OF DFD

As thrilling down the CSFLS analysts find that, when the CSFLS receives the request, it will determine which service the player chooses. The Hot_Number_Request will trigger the next processes to get current winning numbers and past winning numbers. The Find Hot Numbers process will gather these data and generate the Hot_Numbers to the player.

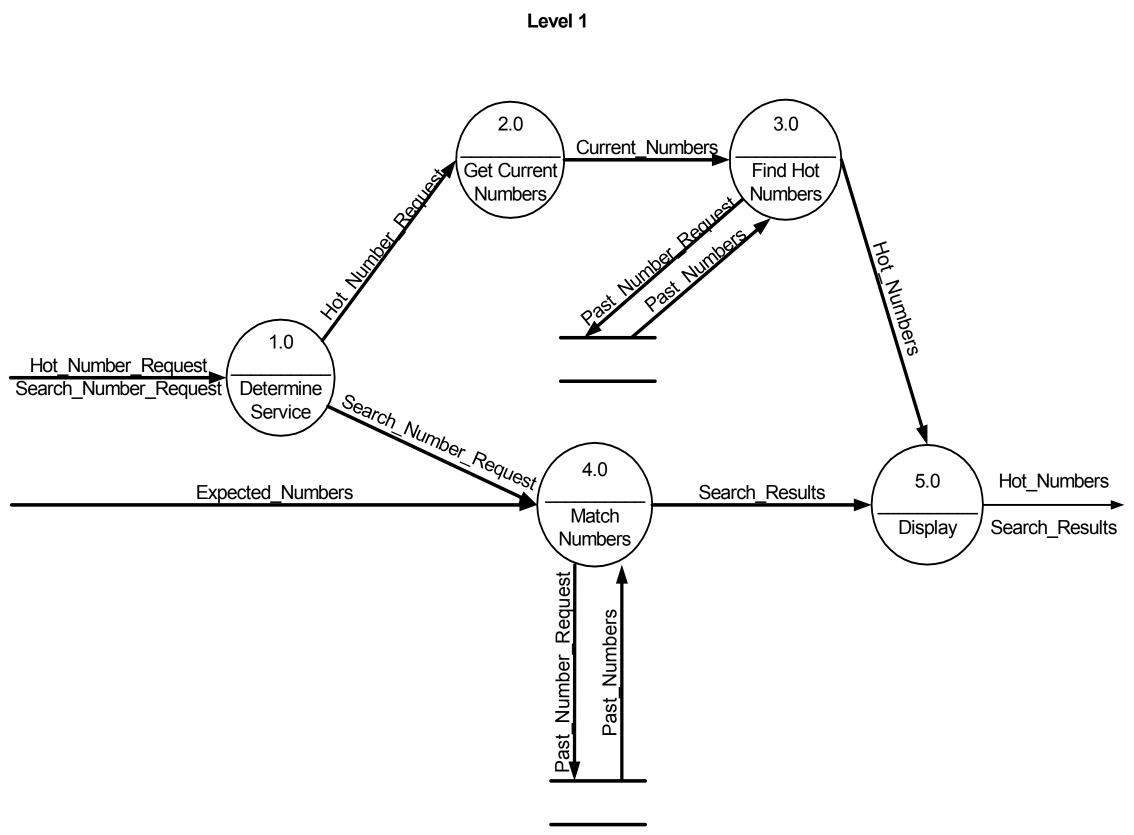

FIGURE 11. LEVEL 1 OF DFD

Now, analysts collect the data in the above Data Flow Diagrams and insert them into the use case form.

Service:

HotNumber

Goal:

Find hot numbers

Belief:

Hot_Number_Request

Current_Numbers

Past_Number_Request

Past_Numbers

Hot_Numbers

Service:

SearchNumber

Goal:

WinningNumberSearch

Belief:

Search_Number_Request

Expected_Numbers

Past_Number_Request

Past_Numbers

Search_Results

Step 8 BDI Agent Cards

The BDI Agent Cards are the collecting formats to bring together all parts of agents into entities. Since the purpose of this approach is to find the BDI agent, the BDI card certainly includes the most important parts: Belief, Desire, and Intention. At previous steps, analysts have already figured out the materials analysts demand. Here, analysts just simply collect them and form them into group based on the goals analysts 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. The following is the format of the BDI Agent Cards:

Agent:

        (Name of the agent)

BDI list:

        Belief:

                (List of the beliefs)

        Desire:

                (The description of the goal)

        Intention:

                (The scenario)

Preconditions:

Postconditions:

Extensions:

The end product of this phase is an Agent Class Diagram. The Agent in the Agent Class Diagram was designed as in Figure, similar to the object in Object Diagram. It contains the agent name, desire, belief, and intention. The Agent Class Diagram reflects the relationship between agents.

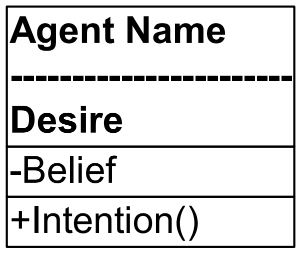

Figure 12. Agent Class

<u>Case Study</u>

Since analysts have extracted the desires, intentions, and beliefs from previous steps, what analysts do now is to insert them into the agent cards to form the complete BDI agents.

Agent:

HotNumber

BDI list:

Belief:

Hot_Number_Request

Current_Numbers

Past_Number_Request

Past_Numbers

Hot_Numbers

Desire:

Find hot numbers

Intention:

- A player requests the Hot Number Service.
- The Hot Number Service asks the current result from the Current Result Service.
- The Hot Number Service asks all past winning numbers from the Past Winning Number Service.
- The Hot Number Service displays all hot numbers on the screen.

Preconditions:

The system has been run.

Postconditions:

All hot numbers are displayed on screen.

Extensions:

2a. The current result is same to the latest result.

2a1. The Current Result Service will supply nothing to the Hot Number Service.

Agent:

WinningNumberSearch

BDI list:

Belief:

Search_Number_Request

Expected_Numbers

Past_Number_Request

Past_Numbers

Search_Results

Desire:

Search if the player's favorite numbers have ever won

Intention:

- A player requests the Search Number Service.
- The Search Number Service provides the input form.
- The player inputs expected five winning numbers and one mega number.
- The Search Number Service asks all past winning numbers from the Past Winning Number Service.
- The Search Number Service displays the search results on the screen.

Preconditions:

The player must input five winning numbers and a mega number.

Postconditions:

The results in the search format are displayed on screen.

Extensions:

2a. Submitted data is incomplete:

2a1. The system requires missing numbers

2a2. The player provides the missing numbers

Agent:

CurrentResult

BDI list:

Belief:

Hot_Number_Request

Current_Numbers

Desire:

Get current result from the Internet

Intention:

- The Hot Number Service requests the Current Result Service the current numbers
- The Current Result Service gets the current numbers from the California Super Lotto web sit
- The Current Result Service sends the current numbers to the Hot Number Service

Preconditions:

The player must ask for the Hot Number Service

Postconditions:

The Hot Number Service has the current numbers

Extensions:

3a. The current numbers are same to the latest numbers

3a1. The Current Result Service sends nothing to the Hot Number Service

After finishing the BDI Agent Cards, analysts can draw the Agent Class Diagram.

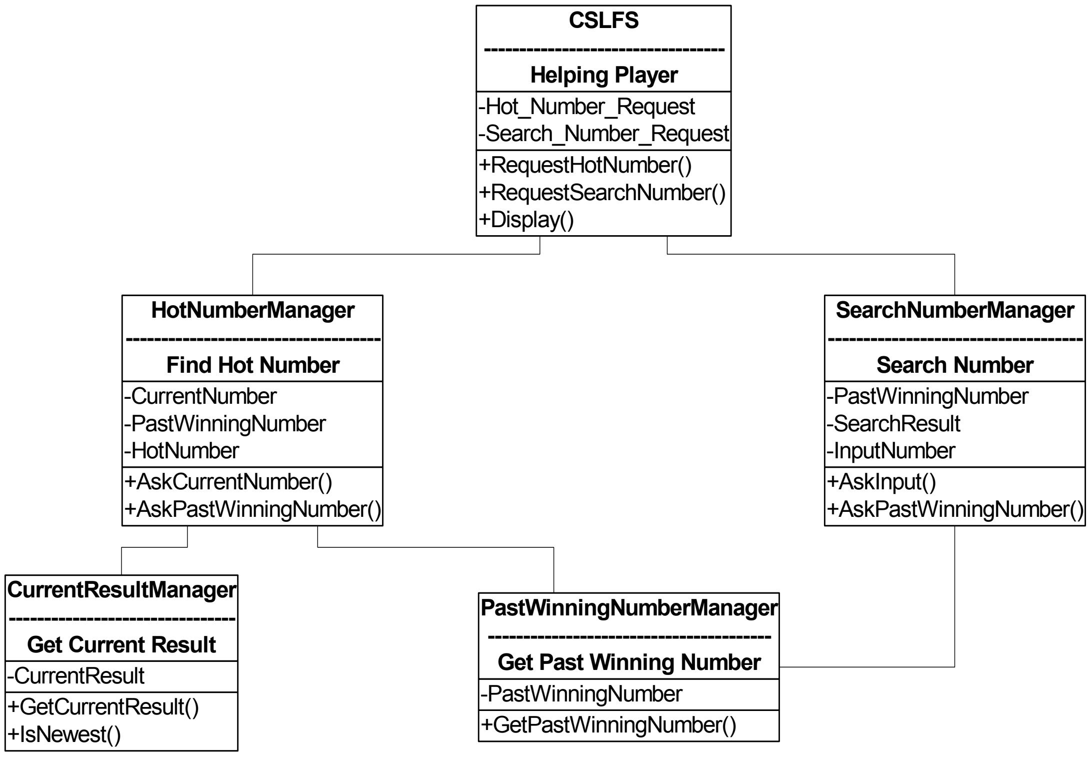

Figure 13. CSLFS BDI Agent Class Diagram

REFERENCE

[1] Booch, G., Object-Oriented Analysis and Design with Applications, Addison Wesley, 1994.

[2] Bradshaw, Jeffrey, An Introduction to Software Agents, Software Agents, American Association for Artificial Intelligence, 1997.

[3] Bratman, Michael E., *Intention, Plans, and Practical Reason*, Harvard Univ. Press, 1987

[4] Cohen, P.R. and Levesque, H.J., *Intention is choice with commitment,* Artificial Intelligence, 1990.

[5] DeLoach, Scott A. and Wood, Mark F., Multiagent Systems Engineering: The Analysis Phase, Afit/En-Tr-00-02 Technical Report June 2000.

[6] Einhorn, Jeffery M. and Jo, Chang-Hyun, *A Use-Case Based BDI Agent Software Development Process,* under a review process for publication, 2003.

[7] Fantechi, A., Gnesi, S., Lami, G., and Maccari, A., *Application of Linguistic Techniques for Use Case Analysis, 2003.* http://matrix.iei.pi.cnr.it/FMT/WEBPAPER/RE02-revfin.PDF.

[8] Feng, Xin and Jo, Chang-Hyun, *Agent-Based Stock Trader,* ISCA CATA-03 March 26-28, 2003, Honolulu, Hawaii.

[9] Jo, Chang-Hyun and Arnold, Allen J., *Real-Time Communication On A Portable Distributed Programming Environment*, under a review process for publication, 2003.

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

[11] Jo, Chang-Hyun, *A Seamless Approach To The Agent Development,* ACM SAC'01, 641-647, March 11-14, 2001, Las Vegas, Nevada, USA.

[12] Kinny, David, Georgeff, Michael, and Rao, Anand, *A Methodology and Modelling Techniqure for Systems of BDI Agents*, Agents Breaking Away: 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Maamaw '96, Eindhoven, the Netherlands, January 22-25, 1996: proceedings (Lecture Notes in Artificial Intelligence), 61, 1996

[13] Larman, Craig, Applying UML and Patterns: Second Edition, Prentice-Hall, 2002.

[14] Lin, Dongqing, Wiggen, Thomas P., and Jo, Chang-Hyun, *A Restaurant Finder Using Belief-Desire-Intention Agent Model And Java Technology,* ISCA CATA-03 March 26-28, 2003, Honolulu, Hawaii.

[15] Merriam-Webster's Collegiate Dictionary, 2000

[16] Negroponte, N., *Agents: From Direct Manipulation to Delegation*, In software Agents, ed. J. M. Bradshaw, Menlo Park, Calif.: AAAI Press, 1997.

[17] Nwana, H.S., Software Agents: An Overview, Knowledge Engineering Review, 205-244, 1996.

[18] Odell, J., Parunak, V. D., and Bauer, B., Extending UML for Agents, 2000. http://www.jamesodell.com/ExtendingUML.pdf.

[19] Ovum Report, Intelligent agents: the new revolution in software.

[20] Sargent, P. Back to school for a brand new ABC, In: *The Guardian*, March 12, 1992, p 28.

[21] Shoham, Y., *Agent-Oriented Programming*, Artificial Intelligent, 1993.

[22] UML Tutorial in 7 Days, http://odl-skopje.etf.ukim.edu.mk/uml-help, Apr.

2003.

[23] Wooldridge, M. and Jennings, N., Intelligents: Theory and Practice, The Knowledge Engineering Review, 115-152, 1995.

[24] Wooldridge, M., Jennings, N., and Kinny, D., A Methodology for Agent-Oriented Analysis and Design, 2003. http://citeseer.nj.nec.com/wooldridge99methodology.html

[25] Wooldridge, Michael J., Reasoning about rational agents, *Massachusetts Institute of Technology*, p7, 2000.

[26] Zhao, Wei and Jo, Chang-Hyun, *A Compiler Design For The Agent-Based Programming Language,* ISCA CATA-03 March 26-28, 2003, Honolulu, Hawaii.