JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2005**

Vol. 4, No. 9, November-December 2005

A BDI Agent-Based Software Process

Chang-Hyun Jo, California State University Fullerton, USA,
Jeffery M. Einhorn, University of North Dakota, USA,

Abstract

Agent-based programming comes us as a next generation programming paradigm. However, we have not been ready yet to fully use it without having sound and concrete software engineering methods and tools to facilitate agent-based software development. In this paper we propose a new software engineering process based on the BDI agent concept. We have refined and extend substantially our previous work, Agent-based Modeling Technique (AMT) and Agent-based Software Development Process (ASP), so that a systematic and realistic process has been born to construct BDI agent-based software. This paper introduces our new approach to the BDI agent-based software development process.

The Belief-Desire-Intention (BDI) model has been proved as a dominant view in contemporary philosophy of human mind and action. We utilize BDI as a tool to analyze agents' environments, goals, and behaviors. Use cases have been proved as a useful tool for requirement analysis. However, use cases cannot be neither agent-oriented nor object-oriented even though it has been used as a tool for analysis for a while. We have extended the existing use cases, and use a new sort of use cases to identify BDIs of agents in the real-world problem.

We use external use cases to get the basic behaviors (intentions) needed to provide the services in the system. We use then internal use cases to define goals (desires) of the system and to discover more specified behaviors (intentions) to achieve the goals. By analyzing the behaviors (intentions) for each goal (desire), we can obtain environments (beliefs) on which the system behaves to perform the goal.

The goal of this paper is to provide a very practical and systematic way to analyze and design the agent software based on the BDI concepts. We have started by using the existing proven tools and methods such as the use case approach, however, we have made a substantial modifications and improvements to these existing techniques so that we can analyze and design the system very realistically based on the BDI agent concept.

This paper provides a systematic and seamless approach to the BDI agent-based software development. The way we suggest here to find BDI agents through requirements analysis is a unique and novel approach. This technique suggests a new way of thinking for BDI agent-based modeling.

1 INTRODUCTION

Agent-based software development provides a next generation of software construction. Agent-based software consists of agents cooperating to achieve a common goal. To succeed the common goals, agents can be working in the form of highly distributed, mobile, autonomous, intelligent and cooperative entities. Building systems based on agents gives a more natural way to simulate complex real-world systems [Jennings 2000]. Therefore, agents have been next generation entities to model the complex systems in many research works [Jennings 2001] [Wooldridge 2000] [Wooldridge and Jennings 1995] [Wooldridge and Jennings 1999] [Wooldridge et al. 1999] [Depke 2001] [Iglesias et al. 1998] [Jo 2001] [Petrie 2001].

The rationality of intentional action is viewed as a primary function of the agent's desire-belief reasons for action [Bratman 1987]. An agent's desires and beliefs at a certain time provide her with reasons for acting in various ways. This Belief-Desire-Intention (BDI) model has been used to describe the behavior of agents with certain goals on a certain environment [Rao and Georgeff 1995] [Wooldridge 2000].

The goal of this paper is to provide a very practical and systematic way to analyze and design the agent-based system based on the BDI agent model. In this paper we present a new way of development process by using various kind of artifacts to model agents of the system based on their belief, desire, and intentions. In our modeling, real-world entities are described as agents when we assign their beliefs, desires, and intentions.

Previously we have once introduced the Agent-based Modeling Technique, AMT [Jo 2001]. We have also briefly described a draft definition of our Agent-based Software Development Process (ASP) in the previous work. This paper is to provide more mature and concrete steps and artifacts in each step. We have also adopted other techniques such as different kinds of use cases to analyze the requirements and to aid in finding beliefs, desires and intentions of agents in the system. Our method presented here is a very unique and systematic approach to find BDIs and to assign them to appropriate agents to perform the system services. The new technique we presented here is realistically useful. It provides a seamless and systematic approach from the analysis to the implementation.

Adoption of use case approach has made our process more realistic and systematic. Use cases have been a proven tool in analyzing systems [Cockburn 2001]. Use cases are neither object-oriented nor agent-oriented approach. Using use cases is rather functional approach. However, use cases have been well used to gather requirements in the analysis phase. Functional approach has also been well used in object-oriented analysis [Larman 2002]. We have adopted this approach first and then extended substantially to find beliefs, desires, and intentions of the system.

To properly model both BDIs and agents in the system, we have developed a unique technique for requirements analysis to use various kinds of use cases. We have extended the existing use cases technique by categorizing external use cases and internal use cases. We have developed a new way to describe a system by using both external use cases and internal use cases. We use external use cases to get the basic behaviors (intentions) needed to provide the services in the system. We use then internal use cases to define goals (desires) of the system and to discover more specified behaviors (intentions) to achieve the goals. By analyzing the behaviors (intentions) for each goal (desire), we can obtain environments (beliefs) on which the system behaves to perform the goal.

In our model, a system consists of cooperative and communicating agents. An agent is defined by a set of BDIs [Jo 2001]. To model the system by using the BDI agents, we go through two steps: (1) In analysis, we use various artifacts including use cases to find BDIs and agents in the system; (2) In design, we assign BDIs to appropriate agents in the system.

We propose a new BDI agent-based software development process in the next sections. We show this step by step, and we will show how the information previously we found in the analysis is correlated and transferred into the later artifacts. The artifacts we generate in the requirements analysis will be naturally used in the artifacts in other phases of both analysis and design. We explain here briefly how the rest of phases will go through, but the more detailed version for other phases will be introduced in separate papers.

Background

There has been much debate on the definition of an agent or even an intelligent agent. The simplest definitions of an agent usually are described as an object with a goal or an entity that acts upon the environment it exists in [Cockburn 2001]. Wooldridge and Jennings describe agents as having autonomy, proactiveness, reactivity and social ability [Wooldridge and Jennings 1995]. In our research an agent-based system is a system that is made up of agents defined by a set of beliefs, desires and intentions (BDI) [Bratman 1987] [Rao and Georgeff 1995] [Wooldridge 2000]. In our research entities become agents when we can assign beliefs, desires and intentions to them.

In the analysis phase of our agent-based software development process will strive to discover potential agents and the BDI's that make up our system. Defining the BDI's does border on design instead of analysis because we are describing how something will be done. In the design phase of our agent-based software development process we will assign the BDI's to software agents.

Our development process builds upon successful strategies that can be found in object-oriented development. We propose new methods for use in agent-based software development whenever previous tools found in other development processes such as UML, the Unified Development Process [Booch et al. 1999] [Larman 2002] and use cases [Cockburn 2001] prove inadequate for agent-based development.

Use cases are another tool that will be fundamental to our agent-based software development process. Use cases are a proven tool that helps drive the development process forward and helps capture the requirements of a system [Cockburn 2001]. Use cases provide a functional approach to gathering requirements [Larman 2002]. Jennings also supports functional analysis by describing it as more natural than data or object type analysis [Jennings 2001]. The functional approach will also be useful when building agent-based systems because it is necessary to gather requirements for agent-based systems. We will use a modified use case called an external use case for discovering the functions or services that our systems should provide. We will also use another kind of use case called an internal use case for identifying plans (intentions), goals (desires), and their beliefs from the system services discovered from the external use case.

In our agent-based development process we will first identify the services that our system should provide. The system can be thought of as an agent, since we will describe our entire system as an encapsulated entity, which will have state and behavior. After we have identified the services that our system will provide we can then identify the goals that are necessary to provide each service. Identifying the proper goals and assigning them to agents becomes a major focus of the agent-based software development process.

An agent's beliefs correspond to the knowledge an agent has about its environment. The desires of an agent can be described as the goals an agent can choose to achieve. An agent’s intentions are the plans that will allow the fulfillment of a goal. In our agent-based software development process we define an agent as an entity that we can assign BDI to. In our development process we will identify the possible agents and the goals that will provide the system’s functionality. In the process of discovering goals we will also assign beliefs and intentions to each goal. We define the software agents in the system as we assign BDI to candidate agents.

In studying the research that was been done in the area of agent software systems we have found two general types of works to be useful. The first is the research that has been done to solve problems from a software engineering perspective. Research into such tools as CRC cards [Bellin and Simone 1997], UML diagrams [Fowler and Scott 2000], use cases [Cockburn 2001] and software patterns [Gamma et al. 1995] [Larman 2002] have been invaluable for use in constructing object-oriented systems [Booch 1994]. Agent UML is one of the pioneer work in extending UML for agent development [Odell et al. 2000]. In our research we have modified several of the tools that have proved successful for object-oriented software construction for use in agent-based software systems.

The second area of research is in the agent theory. Jennings, Wooldridge and others [Wooldridge et al. 1995, 1999, 2000] [Iglesias et al. 1998] [Depke et al. 2001] [Jennings 2000, 2001] [Petrie 2001] provide research into the theory of agent software development. Depke et al. [Depke et al. 2001] takes the approach of describing a system using roles. It provides a brief description of a development process based on roles. They also make the necessary additions to UML in order to provide diagrams to describe their process. Wooldridge, Jennings and Kinny [Wooldridge et al. 1999] also talk about defining a system based upon its organization. They state by looking at the roles played by agents in the system you can then model the system based upon those roles. Instead of focusing on roles we focus on describing the beliefs, desires and intentions for each agent.

Michael Wooldridge has published several research documents on agent theory and agent software development techniques [Wooldridge et al. 1995, 1999, 2000]. Wooldridge presents a detailed BDI architecture, which is designed for building BDI agent systems [Wooldridge 2000]. Rao and Georgeff [Rao and Georgeff 1995] provide a paper describing a BDI architecture for use in building agents. They formalize their work using BDI logic and provide a model that can be used to describe BDI agents. Their research provides us with a better understanding of BDI and how it can be formally described. One of the most related works we found was Kinny's work [Kinny et al. 1996]. They similarly view the agent system from external viewpoint and internal viewpoint. Their work describes the framework necessary for agent-based modeling very well. However, our modeling views and captures external view and internal view using our own coherent methods and techniques. Our technique using two different kinds of use cases provides a very systematic way to develop BDI agent-based systems.

2 A PROCESS FOR BDI-AGENT SOFTWARE DEVELOPMENT

In our previous research, Agent-based Software Development Process (ASP) and Agent-based Modeling Technique (AMT) have defined four development phases such as requirements analysis, modeling, construction, and deployment. Each phase suggests four steps analysis, design, build and test iteratively and evolutionally [Jo 2001].

AMT suggests the analysis step include the following sub-steps:

- Analyze the system requirements
- Construct BDI agent cards
- Identify agents and concepts
- Identify relationships among agents
- Build agent scenarios
- Identify agent boundary
- In the design step, AMT suggests the following sub-steps:
- Build agent relationship diagrams
- Build agent interaction diagrams
- Use agent patterns
- Build agent component diagrams

In this research we refine the ASP by defining more precise activities and concrete artifacts. In the next section we will introduce our new version of software development process named the BDI Agent-based Software Process (BDI ASP). We should present both a brief discussion of each artifact created in each phase as principles and an example of the artifact as practices. Because of the limitation of the paper length we, however, cannot provide examples of all artifacts. To see a more detailed example and a more realistic case study, you may refer to our web page at http://jo.ecs.fullerton.edu/research.

Brief Process Description

A salient point in our research is the use of BDI [Rao and Georgeff 1995] for describing agents. BDI provides us with a clear view of what makes up an agent. We will assign beliefs, desires and intentions to each agent. Our process will provide the tools that will be necessary to systematically build agent-based software systems.

Figure 1 provides a high level view of how we will use BDI in our agent-based development process. Figure 1 describes a general approach of how an agent BDI attributes are discovered in our BDI agent software development process. 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. Before we discuss each step of our development process in detail it is useful to take a high level view of the entire BDI agent software development process. Our process stresses a goal-oriented approach for developing agent-based systems. Use cases play an important role in discovering the goals that will be necessary to provide the services for our system.

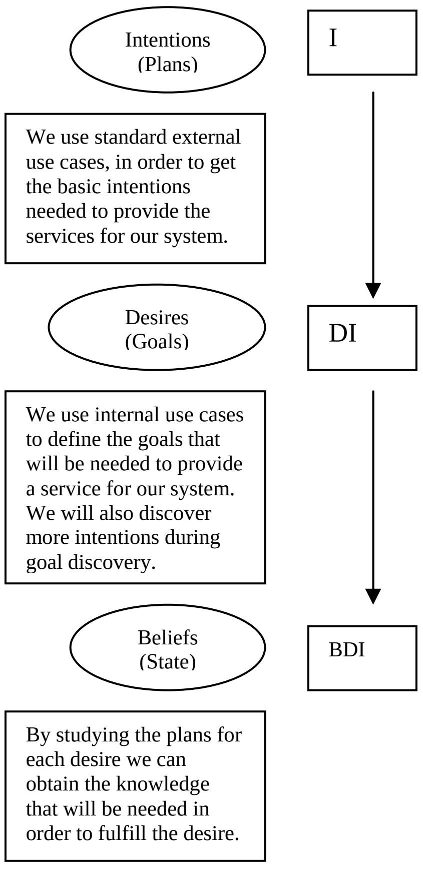

Intentions
(Plans)
I
We use standard external use cases, in order to get the basic intentions needed to provide the services for our system.
Desires
(Goals)
DI
We use internal use cases to define the goals that will be needed to provide a service for our system. We will also discover more intentions during goal discovery.
Beliefs
(State)
BDI
By studying the plans for each desire we can obtain the knowledge that will be needed in order to fulfill the desire.

Figure 1: BDI Discovery in the BDI Agent Software Process

Figure 2 is a diagram of the artifacts that will be created during our BDI agent development process. The arrows show the general order of creation for the artifacts in our process. It is important to understand that the artifacts can be created in any order that is useful to the developer. The arrows represent a loose order that we suggest for artifact creation at this time.

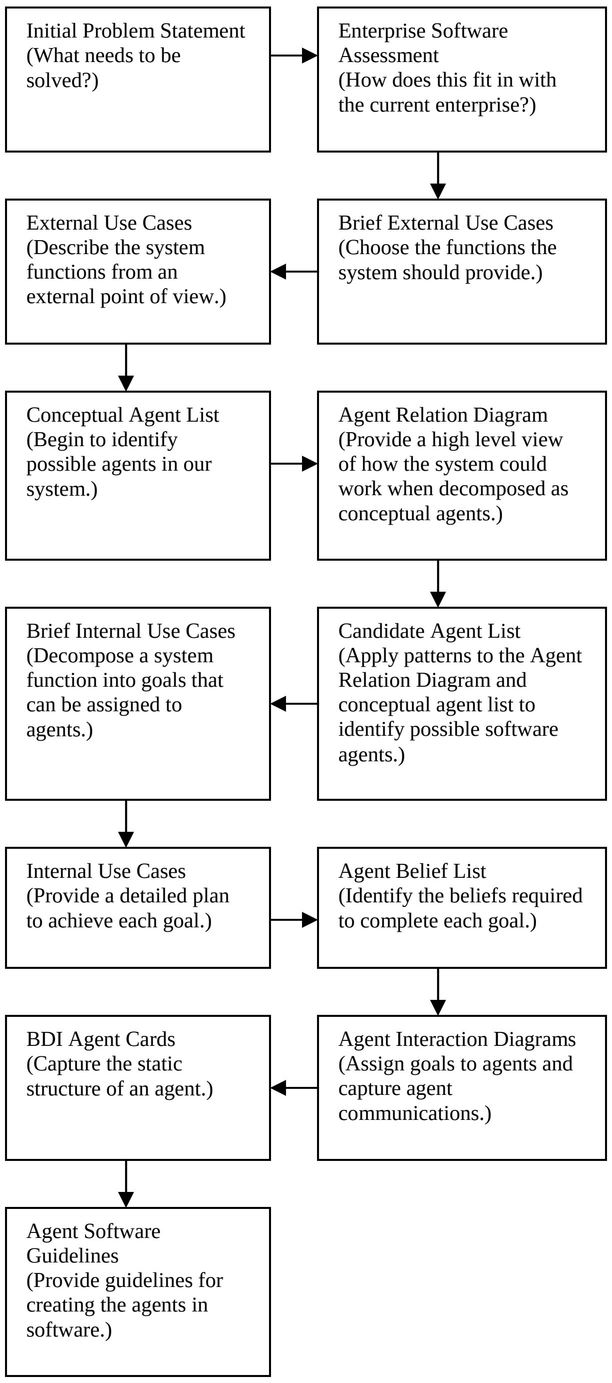

Initial Problem Statement (What needs to be solved?)
Enterprise Software Assessment (How does this fit in with the current enterprise?)
Brief External Use Cases (Choose the functions the system should provide.)
External Use Cases (Describe the system functions from an external point of view.)
Conceptual Agent List (Begin to identify possible agents in our system.)
Agent Relation Diagram (Provide a high level view of how the system could work when decomposed as conceptual agents.)
Candidate Agent List (Apply patterns to the Agent Relation Diagram and conceptual agent list to identify possible software agents.)
Brief Internal Use Cases (Decompose a system function into goals that can be assigned to agents.)
Internal Use Cases (Provide a detailed plan to achieve each goal.)
Agent Belief List (Identify the beliefs required to complete each goal.)
Agent Interaction Diagrams (Assign goals to agents and capture agent communications.)
BDI Agent Cards (Capture the static structure of an agent.)
Agent Software Guidelines (Provide guidelines for creating the agents in software.)

Figure 2: BDI Agent Software Development Process

3 BDI AGENT SOFTWARE ANALYSIS

Here we describe the steps and artifacts for the analysis phase in our BDI Agent-based Software Development Process (BDI ASP). The steps described here are rather concurrent than strictly sequential steps. Many artifacts here are correlated one another, therefore they can be concurrently constructed and use information to construct one part of artifacts from others.

Problem Statements

Our requirements analysis phase starts by building initial problem statements. After an iteration of full phases in the process, the initial problem statements are revisited and refined.

Case Study: Initial Problem Statement

A customer would like to receive special notices of certain types of weather events. The business will direct the forecasters that they need to create these new notices. We need to develop a tool that will aid the forecasters in providing notices to districts inside a state. The system should be able to provide notices for a variety of events (frost, severe-weather, freezing rain). The customer wishes to use these notices as a warning that they may need to take action in order to respond to an event. Our business would like the interface to be fast and easy to use in order to minimize both the time and cost to the forecaster in creating the notices. We do not want the forecaster to have to worry about the delivery of the notices. Instead we would like to develop a system that will automatically deliver the notices to the districts once they are created. The forecasters job is identifying when to create a notice. The forecaster will use an interface to create the notices. The system should be able to format and deliver the notices, created by the forecasters, as needed. The customer often wants the notices delivered in a variety of formats (web pages, faxes or both). The customers usually want the notices delivered to each district where the notice is valid.

Enterprise Assessment

We then assess the enterprise software to find how new requirements fit the existing enterprise. In each iteration we redo the enterprise assessment.

Case study: Enterprise Software Assessment

We currently have a system that stores all our weather products in a database. The notices could be added to a database as a new weather product. Once a notice is received in the database we could provide another process that will handle the delivery and formatting of a notice. We currently have an internal system set up called the notifier that can watch the database for different kinds of weather products to be inserted. We can use the notifier to signal the system when new notices are created.

External System View

The next step is to discover all the functions that system should provide for the new requirements. This is gone through by writing external use cases. With external use cases, we can describe the system functions from the external point of view. This system functions will be foundations of behavioral plans (intentions). External use cases become to the detailed external use cases during the several iterative steps.

Case study: Brief External Use Cases

Name: CreateNotice
Description:
A forecaster identifies the need to submit a notice or notices in a region. The forecaster starts the notice interface. The forecaster selects the state to submit notices in. The forecaster then selects the districts to submit notices to. The forecaster then creates and submits the notice to the system. The system recognizes that a notice needs to be delivered. The system formats the notice properly for delivery and then delivers the notice properly.

Name: ViewNotice
Description:
A forecaster wishes to view the notices that are currently valid. The forecaster starts the interface and selects the region to view notices in. The forecaster is able to easily see where valid notices are and can bring up the details of a notice as desired.

Name: Start
Description:
The system manager needs to start the system.

Name: Stop
Description:
The system manager needs to be able to stop the system.

Case study: (Detailed) External Use cases

External use case: CreateNotice
Primary Actors: Forecaster, System, District
Stakeholders:
-Forecaster wants fast and accurate entry of the notices.
-Customer wants accurate and timely delivery of the warnings to districts
-Districts are interested in taking appropriate action for each warning.
-Company wants to satisfy customer interests in a cost effective manner.
Preconditions: Forecaster has identified a need to submit a severe weather warning for an area.
Sucess/Postcondition: A notice is delivered to the district and a copy is saved.
Scenario:

A customer requests, from the company, that they receive notices of certain kinds of weather events.
The company directs the forecaster to create the notices for the customers.
A Forecaster recognizes the need to create a notice.
Forecaster starts the notice creation interface.
Forecaster selects the proper customer to issue a notice for.
Forecaster creates the text of the notice.
Forecaster submits the finished notice to the system.
The system recognizes that a notice needs to be delivered.
The system formats the product for delivery.
The notice is delivered in the proper format to each district.
Forecaster repeats steps 5-6 as needed.

Extensions:
a) System fails:
-any work that hasn't been submited by the forecaster should be lost.
-restart the interface and recreate the notice.

Special Requirements:
-Once the system is loaded it must have a very quick response time (less than a sec or two from the forecasters perspective)

Conceptual Agent Identification

Once we create external system view, we try to create a conceptual agent list. Conceptual agents are candidate agents, which might or might not be adopted as software agents in the design phase. Conceptual agents provide us with information for possible candidates so that we can focus on those agents to assign proper BDIs later. Conceptual agents can be found by using linguistic analysis as widely used in the object-oriented analysis. We use nouns or noun phrases to find candidate agents from the previous artifacts such as problem statements and external use cases.

Case Study: Conceptual Agent List

The following case study lists the conceptual agents of the Notice Management System.

Notice	Weather
Notifier	District
System	Web Page
Forecaster	Fax
Customer	Delivery

Scenario Construction

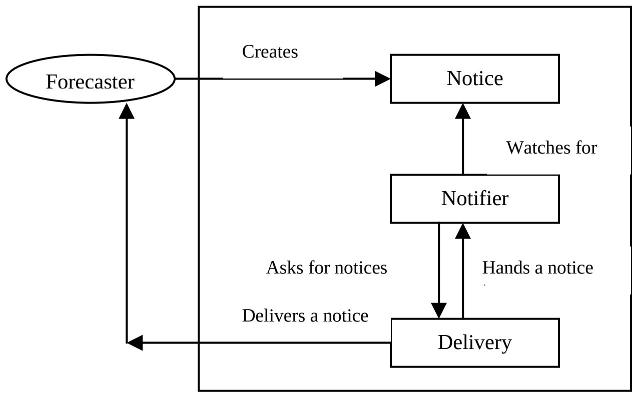
An initial scenario describes the system process in plain sentences. The scenarios will become more formalized by using a form to describe a scenario name, a set of clients, a set of servers, goals and plans. Scenario helps us to find out the system process, related client agents and server agents, their goals, and plans.

Agent Relationship Identification

After we are familiar with conceptual agents in the system and the system process through the scenarios, we build the conceptual agent relationship diagram (ARD). A conceptual ARD provides conceptual relationships to show how the system functions are achieved among the conceptual agents. Conceptual ARDs will be used to find out related agents in the design step, and conceptual ARDs do not guarantee any software implementation yet. During the iteration, conceptual ARDs will become detailed ARDs. ARDs use oval representation for external agents and rectangular representation for internal agents.

Case Study: Conceptual Agent Relation Diagram

Service: createNotice



Case Study: Candidate Agent List

Agent	Reason
Notice	Conceptual Agent List
Weather	Conceptual Agent List
Notifier	Conceptual Agent List
District	Conceptual Agent List
System	Conceptual Agent List
Web Page	Conceptual Agent List
Forecaster	Conceptual Agent List
Fax	Conceptual Agent List
Customer	Conceptual Agent List
Delivery	Conceptual Agent List

NoticeManager	By studying the agent relation diagram for the createNotice service we can see that the external Forecaster agent is communicating directly with the notice agent. By applying the manager pattern, we identify the possible need for a NoticeManager.
DeliveryManager	We recommend this agent based on applying the manager pattern.
Database	We discover this agent by applying the service pattern. The service pattern provides a single agent that is available to all the internal agents in our system. Many agents will need to get and store information in the database and we may provide a single database agent to handle this.
DeliveryService	We recommend this agent based on applying the service pattern.
DeliveryBroker	We identify this possible software agent based on the broker pattern. We have several possible agents such as Fax and Web page, which provide the function of delivering notices to the districts. The broker could provide a single agent that decides which agent to use for notice delivery.
SystemManager	This agent is recommended from looking at the start brief internal use case. The system manager will ensure the proper agents are created at the systems initialization.

Internal System View

We decompose the system functions investigated from the external use cases by building internal use cases into the goals (desires) that are supposed to be assigned to appropriate agents. Agent decomposition is done in the reverse order of belief-desire-intention lists suggested in Jo [2001]. From the external system view through the external use cases construction, we list the goals. A system service described in an external use case is decomposed into one or more goals. Each decomposition step creates an internal use case. An internal use case describes the plans for each goal that is identified to provide a system service described in an external use case. Each goal has a plan that may include other goals, which in turn include their own plans.

Case Study: Brief Internal Uses Cases

Service: CreateNotice
Name: CreateNotice
Description:
The forecaster has identified a need to submit a notice for an region. The forecaster starts the notice interface. The forecaster selects the proper state to issue a notice for. The forecaster enters the notice text. The notice is formatted and submitted and is stored in the database.

Name: WatchForNoticesToDeliver
Description:
The notifier is started and asked to watch the database for new notices. When a new notice arrives it is handed to the interested party.
Name: DeliverNoticesToDistricts

Description:
A notice arrives for delivery. The notice contains the information about whom it should be delivered to. The database is checked on how to properly format the notice for delivery. The database is checked on how to format the notice properly. The notice could be delivered as a fax, web page or both. We also want to add the ability to deliver the notice in new formats.

Case Study: Internal Use Cases

Service: CreateNotice
Internal use case: CreateNotice
Actors: NoticeManager, Forecaster, Notice
Stakeholders:
-Forecaster wants fast and accurate creation of the notices.
-NoticeManager handles the interaction with the forecaster and desires proper creation and maintenance of notices.
-Notice contains all the information about a notice.
Preconditions: Forecaster has identified a need to submit a notice for an area.
Postcondition: Notice is delivered/saved to the database.
Scenario (intentions):
 Forecaster asks the NoticeManager agent to create a Notice.
 The NoticeManager agent provides an interface to the forecaster for notice creation.
 NoticeManager agent properly formats and submits the notice to the database.

Internal use case: Create
Extends: CreateNotice, intention 2
Actors: NoticeManager, Notice
Preconditions: We need to create a notice.
Success/Postconditions: A notice is created.
Scenario (intentions):
 The notice interface is started for notice creation.
 The NoticeManager gets the State from the user.
 The NoticeManager gets the district from the user.
 The NoticeManager gets the notice text from the user.
 The NoticeManager passes the DistrictIds and the notice text to the Notice.
 A new notice is created.
 The NoticeManager now has a notice.

Internal use case: Submit
:

Internal use case: WatchForNoticesToDeliver
Actors: Notifier, Delivery, DeliveryService
:

Internal use case: DeliverNoticesToDistricts
Actors: DeliveryService, Notice
:

Agent Belief Lists

We have found so far plans (intentions) and goals (desires) through the previous steps. Now it is time for us to identify the environments (beliefs) required to achieve goals for agents. Goals and plans certainly require information to access and update. Such information forms environment on which agents are working to achieve goals by using plans. This environment can be represented as states in a knowledge base, and it is called as belief in our model. The agent belief lists (ABLs) show what kinds of beliefs are needed to achieve goals. An agent belief list consists of a system service name, a set of triples (goal, belief, reason) to perform this service.

Case Study: Agent Belief List

Service: CreateNotice
Goal: CreateNotice
Belief: NoticeDB
Reason: We need to know the database to submit notices to.

Goal: Create
Belief: StateDB
Reason: Agent needs to provide a list of districts for a state to the user.

Goal: Submit
Belief: NoticeDB
Reason: Agents needs access to the Notice DB to insert new notices.

Goal: WatchForNoticesToDeliver
Belief: NoticeDB
Reason: Agent needs to watch for new notices entering the NoticeDB.

Goal: DeliverNoticesToDistricts
Belief: StateDB
Reason: Agent formats the notice for delivery based upon how the states desire it delivered.

Brief DBI Agent Cards Construction

A BDI agent card summarizes the system requirements, agents, and their BDIs. A BDI agent card lists agents' names with a set of their BDIs, collaborators, pre-conditions, and post-conditions respectively. BDI agent cards explain the static property of the agent system. The goal and belief identified from agent belief lists (ABLs) become concrete desire and belief. The plans we identified by external system view and internal system view are used to define intentions. In the first iteration, a BDI agent card might be brief and abstract. It may not include any detailed idea, but it gives a guide to identify proper agents and their BDIs. In the consecutive iterations, brief BDI agent cards become more detailed and concrete.

Agent Boundary Identification

An agent boundary diagram (ABD) provides conceptual view of the system with the external services. All conceptual agents will be categorized into their agent boundaries. ABDs will be eventually foundations of components diagrams in the design phase that will be useful in the construction and deployment phases.

4 BDI AGENT SOFTWARE DESIGN

In the previous sections we have shown the steps and artifacts for in the analysis phase of our BDI agent-based software development process. The next consequence is the design phase of our BDI agent-based software development process.

Detailed Agent Relationship Diagrams

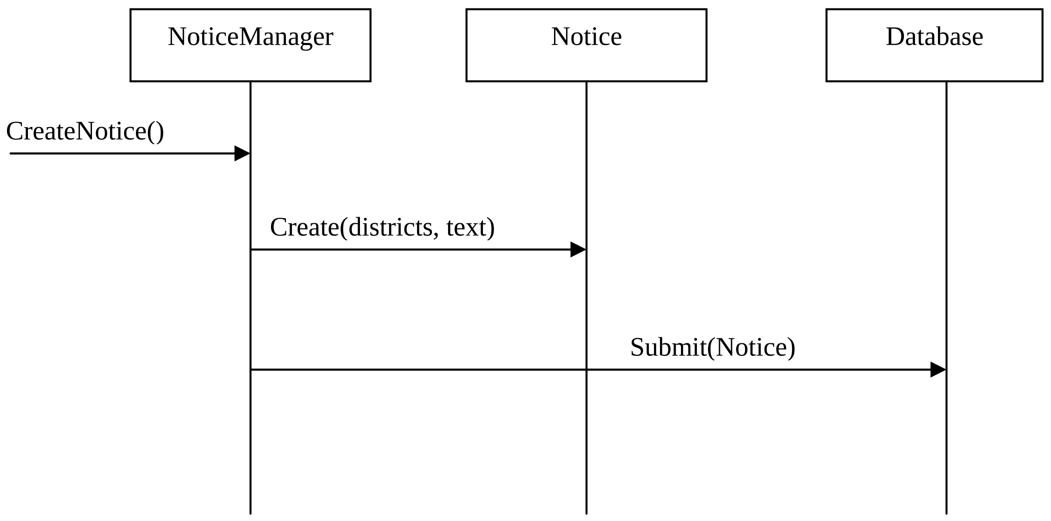
In the design step, we suggest to refine the conceptual agent relationship diagrams (ARDs). Conceptual ARDs will become more detailed and concrete through the several iterative and evolutionally steps in different phases. With the concrete agent relationship diagrams we can identify collaborators to achieve common goals among cooperative agents in the system.

Agent Interaction Diagrams

Here we assign goals (desires) to appropriate agents, plans (intentions) by which agents achieve the goals, and environments (beliefs) on which agents work to achieve the goals by plans. Not only an agent interaction diagram shows goals and participating agents, but also it describes the dynamic characteristic of the agent system.

To achieve the system service we have to find out which intentions are proper to fulfill the system requirements gathered through the external view of the system. Then we find what are the goals to perform these intentions, and which agents are responsible for it. We assign goals to appropriate agents at this time. Certain patterns such as the agent goal assignment pattern [Einhorn 2002] are useful to find the most appropriate agents to fulfill the goals. An agent interaction diagrams shows a set of agents and their communications via invoking their goals.

Case Study: Agent Interaction Diagrams



Agent Patterns

During the software design through the several steps and artifacts listed above such as agent interaction diagrams, some patterns for BDI agents can be very useful. There are not many patterns for agent development [Hayden et al. 1999], but we may use patterns for object-oriented programming to some extent [Gamma et al 1995]. However, they do not provide enough information to find BDIs and appropriate agents, we have to develop own patterns to be used for BDI agents construction. Einhorn [2002] suggests some patterns useful for the BDI agent software design.

Detailed BDI Agent Cards

The brief BDI agent cards in the analysis phase are used for capturing system requirements and finding corresponding responsibilities. The detailed BDI agent cards we build in the design phase are used to assign the responsibilities to the appropriate agents by defining their BDIs and collaborators. Using patterns, we can assign BDIs to appropriate agents more efficiently. The brief BDI agent cards constructed in the previous phase are refining more precisely at this time. Within a few iterations, the detailed BDI agent cards show enough information ready to implement from which programmers can construct software agents and testers can construct test cases at least for prototyping. BDI agent cards with other artifacts will be refined and evolved during continuous iterations.

Mostly detailed BDI agent cards are constructed in parallel with agent interaction diagrams.

Case Study: BDI Agent Cards

Agent: NoticeManager
BDI list:
Desire: CreateNotice
Pre-condition: Forecaster decides to create a Notice.
Belief: NoticeDB
Post-condition: Notice is saved in the database.
Collaborators: Forecaster (external), Notice
Intentions:

> Forecaster asks the NoticeManager agent to create a Notice.
> The NoticeManager agent provides an interface to the forecaster for notice creation.
> NoticeManager agent properly formats and submits the notice to the database.

Desire: ViewNotice
Pre-conditions: Forecaster desires to view a notice area.
Belief: StateDB
Post-condition: Forecast is able to view the contents of a valid notice.
Collaborators: Forecaster (external)
Intentions:

> Forecaster asks the NoticeManager agent to view a Notice.
> The NoticeManager agent provides an interface to the forecaster for notice viewing.

Agent Component Diagrams

We also build agent component diagrams (ACDs) based on artifacts, we constructed during the analysis phase and design phase, such as agent boundary diagrams (ABDs), agent relationship diagrams (ARDs), agent interaction diagrams (AIDs), and BDI agent cards. The agent component diagrams will be eventually used as guidance for packaging of agents and their BDI packages in the construction and deployment phases.

Guidelines

A BDI agent software development guideline suggests how we can develop software agents and how we can implement our models exactly by suggestive mapping models into agent-based programming languages. This is also a very important guideline because it gives both precise and exact steps we have to follow to develop the BDI agent software applications from our models.

Mapping to Codes

The next step is the implementation of the BDI agents figured out from the requirements analysis and the agent design models. The artifacts in both analysis and design and the guidelines can show us how to implement BDI agent-based software in the real programming languages.

There are many ways to implement our BDI agent models in different languages. A good way might be choosing a proper agent-based programming language in which BDI models can be implemented properly and naturally. One of our research works in agent computing includes a design of a new agent-based programming language, APL [Jo 2002]. APL provides a natural syntax and semantics to implement the BDI agent models suggested by the Agent-Modeling Technique (AMT) [Jo 2001].

We will show this mapping in a separate paper. In this paper, we emphasize the agent software analysis and design only based on the BDI agent concept.

Concurrent Artifacts Construction

Some artifacts can be (or must be) constructed in parallel. For example, the BDI agent cards can be concurrently constructed with agent interaction diagrams. While BDI agent cards describe the agent system architecture globally and statically, each agent interaction diagram describes the system service locally but dynamically.

Evolutionary and Iterative Approach

In the rest of phases, construction and deployment, while we evolutionally and iteratively refine BDIs and assign BDIs to proper agents, we implement software agents in agent-based programming languages. The artifacts that are modeled and their corresponding software that is implemented are evolving more and more, not only through the iterative software construction but also through the whole agent software life cycle.

5 CONCLUSION

Systems go more complex and embedded so that software engineers are hard to analyze and design the system with a simple model. The BDI model teaches us how agents plan their intentions with reasoning desires on their belief naturally.

In this paper we use the BDI concept to model agent-based system. We have shown our development process for the agent-based software construction based on the BDI agent model.

Even though at first we have started to use the existing proven methods and tools used in the object-oriented modeling techniques [Bellin and Simone 1997] [Booch 1994] [Booch et al. 1999] [Fowler and Scott 2000], we have refined and extended substantially the existing methods to adapt into the BDI agent-based software construction. As a result, we have concluded with our own BDI agent-based software development process. We have briefly but precisely defined the phases and their steps in the BDI agent-based software development process, and described them reasonably.

This work is a very unique and new approach in the agent-oriented software engineering. Our method will provide a sound, realistic and practical modeling technique for agent-oriented software development.

With the limitation of the paper length we could not list all steps and examples in the whole process. To see a more detailed example and a more realistic case study, you may refer to our web page at http://jo.ecs.fullerton.edu/research.

REFERENCES

[Beck 2000] Beck, K. 2000. Extreme Programming Explained-Embrace Change, *Addison-Wesley*, 2000.

[Bellin and Simone 1997] Bellin, David and Simone, Susan, The CRC Card Book, *Addison-Wesley*, 1997.

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

[Booch et al. 1999] Booch, G., Rumbaugh, J., and Jacobson, I., The Unified Software Development Process, *Addison-Wesley*, 1999.

[Bratman 1987] Bratman, M. E., Intention, Plans, and Practical Reason, *Harvard University Press*, 1987.

[Cockburn 2001] Cockburn, Alistair, Writing Effective Use Cases, *Addison-Wesley*, 2001.

[Depke et al. 2001] Depke, Ralph, Heckel, Reiko, and Kuster, Jochen, Improving the Agent-Oriented Modeling Process by Roles, *AGENTS'01*, 640-647, June 2001.

[Einhorn 2002] Einhorn, M. Jeffrey, A BDI Agent Software Development Process, MS Thesis, (Advisor: Chang-Hyun Jo), University of North Dakota, USA, May 2002.

[Fowler and Scott 2000] Fowler, Martin and Scott, Kendall, UML Distilled Second Edition: A Brief Guide to the Standard Object Modeling Language, *Addison-Wesley*, 2000.

[Gamma et al. 1995] Gamma, E., r. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object Oriented Software, *Addison-Wesley*, 1995.

[Hayden et al. 1999] Hayden, Sandra, Carrick, Christina, and Yang, Qiang, A Catalog of Agent Coordination Patterns, ACM Press, 412-413, 1999.

[Iglesias et al. 1998] Iglesias C. A., Garijo M, Gonzalez J. C., and Juan R. Velasco, Analysis and Design of Multiagent Systems using MAS-CommonKADS, In M.P. Singh, A. Rao, and M.J. Wooldridge, editors, *Proc. 4th Int. Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, volume 1365 of LNAI, 313-328, Springer-Verlag, July 24-26, 1998.

[Jennings 2000] Jennings, Nicholas R., On agent-based software engineering, *Artificial Intelligence*, volume 117, 277-296, February 2000.

[Jennings 2001] Jennings, Nicholas R., An Agent-Based Approach for Building Complex Software Systems, *Communications of the ACM*, 44(4), 35-41, April 2001.

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

[Jo 2002] Jo, Chang-Hyun and Allen J. Arnold, Agent-Based Programming Language (APL), ACM SAC 2002, Madrid, Spain, (March, 2002), 27-31.

[Kinny, D., Georgeff, M., and Rao], A. A Methodology and Modelling Techniques for Systems of BDI agents. Proc. of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, (LNAI Vol. 1038): 56-71, Springer, 1996.

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

[Odell et al. 2000] Odell, J., Parunak, H. V. D., and Bauer, B., Extending *UML for Agents*, Proc. of the Agent-Oriented Information System Workshop at the National Conf. on AI, (AOIS Workshop at AAAI 2000), (2000).

[Petrie 2001] Petrie, Charles, Agent-Based Software Engineering, *Agent-Oriented Software Engineering*, *Lecture Notes in AI*, *Springer-Verlag*, 58-76, 2001.

[Rao and Georgeff 1995] Rao, Anand S. and Georgeff, Michael P., BDI Agents: From Theory to Practice, *Australian Artificial Intelligence Institute*, April, 1995.

[Weiss 1999] Weiss G., editor, Multi-Agent Systems, *The MIT Press: Cambridge, MA*, 1999.

[Wooldridge and Jennings 1995] Wooldridge, M. and Jennings, N. R., Intelligent Agents: Theory and Practice, *Knowledge Engineering Review, Cambridge Univ. Press*, 10(2), 115-152, June 1995.

[Wooldridge and Jennings 1999] Wooldridge, M. and Jennings, N. R., Software Engineering With Agents: Pitfalls and Pratfalls, *IEEE Internet Computing*, 20-27, May-June 1999.

[Wooldridge et al. 1999] Wooldridge, M., Jennings, N. R., and Kinny, D., A Methodology for Agent-Oriented Analysis and Design, *Autonomous Agents 1999, Seattle, WA*, 69-76, 1999.

[Wooldridge 2000] Wooldridge, M., Reasoning about Rational Agents, *The MIT Press: Cambridge, MA*, 2000.

About the authors

Chang-Hyun Jo is an Associate Professor of the Computer Science Department at the California State University Fullerton where he teaches Programming Languages and Software Engineering. He has published more than 50 technical papers in journals, conferences and international standards. E-Mail: jo@ecs.fullerton.edu.

Jeffrey M. Einhorn works as a Distributed Systems Engineer for Futuresource, Inc. in Chicago, IL. He worked on developing a process for designing Agent-Based Software Systems, while pursuing his Masters Degree in Computer Science at the University of North Dakota. He is currently designing and building distributed software systems using Erlang (www.erlang.org) and K (www.kx.com). E-Mail: einhorn@uhhh.org.