Case Study
Use-Case Based BDI Agent Software Construction

Jeffery M. Einhorn
Department of Computer Science
University of North Dakota
Grand Forks, ND 58202-9015

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

ABSTRACT

As computer software continues to grow increasingly complex with each passing year, researchers continue to try and develop means to simplify software development. In this paper, we propose a BDI agent software development process as the next evolution in software development. The goal of this research is to develop a process, which can be used to enable the creation of agent-based systems.
This research strives to present a practical software development process, which is useful to today's software engineer, by building upon current agent research and proven software engineering practices. Our BDI agent software development process is a systematic process, which enables the decomposition of a system into agents. The Belief-Desire-Intention Model is a fundamental ingredient to our development process. We utilize BDI as a natural method for describing agents in our development process. Our software development process utilizes several forms of use cases, which are useful for defining the architecture of a system in our process. We have also leveraged many other existing software development tools such as CRC cards, patterns and the Unified Development Process. We have made modifications to many of these existing tools so they can be used for agent-based development. These are just some of the tools that provide valuable insight into the development of our BDI agent software development process.

Here we provide a case study to clarify the description of our BDI agent software development process. Basically, our BDI agent software development process strives to model both the dynamic and static structure of the agents that make up the system. Once we have modeled the structure, which makes up the agents in the system the structure can then be created in software.

The way we suggest here showing how to form BDI agents through the software development process is a unique and novel concept. This technique suggests a new way of thinking for BDI agent-based modeling.

Keywords

Agent-based modeling, BDI agent software development process

CASE STUDY

This *case study* is for the *Notice Management System* in the *Weather Forecasting System*.

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.

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.

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

In the following case study we have created external use cases for the services createNotice, viewNotice and start. At this time there was no need to create an external use case for stop because we did not feel that the external use case for stop would provide any new insight into the operation of the system.

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)

External use case: ViewNotice

Primary Actors: Forecaster, System

Stakeholders:

-Forecaster wants view current valid notices.

-Company wants forecaster to be able to easily view the valid notices.

Preconditions: Forecaster decides to view a notice.

Sucess/Postcondition: Forecaster is able to view the contents of a notice.

Scenario:

The forecaster wants to view current valid notices.

The forecaster starts the view notice interface.

The forecaster selects the customer to view notices for.

The interface indicates to the forecaster what districts have valid notices.

The forecaster can choose a district and view the valid notice for that district.

Extensions:

5a)

-If if we try and view a notice in a district that currently doesn't have any valid notices, then we should get a message, which says "No valid notice available".

External use case: Start

Primary Actor: System Administrator, System

Stakeholders:

-The System Administrator wants to be able to start and stop the notice delivery system as desired.

-Company wants System Administrator to be able to easily manage the system.

Preconditions: System Administrator decides to start the system.

Sucess/Postcondition: The system is started.

Scenario:

The System Administrator wishes to start the notice management and delivery system.

The System Administrator starts the notice management and delivery system.

The System starts the proper components to enable management and notice delivery.

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

Case Study: Conceptual Agent Relation Diagram

The following figure is the agent relation diagram for the createNotice service. The diagram of the createNotice service provides a general view of how a conceptual system might provide the createNotice service.

In the following case study we create an agent relation diagram for the service createNotice. Figure 3 is the agent relation diagram for the createNotice service. The diagram of the createNotice service provides a general view of how a conceptual system might provide the createNotice service.

The conceptual agent relation diagram provides a conceptual view of a possible system. The oval circles represent external agents and the boxes represent internal agents. There is a large rectangle that surrounds all thee internal agents, which represents the system boundary.

Service: createNotice

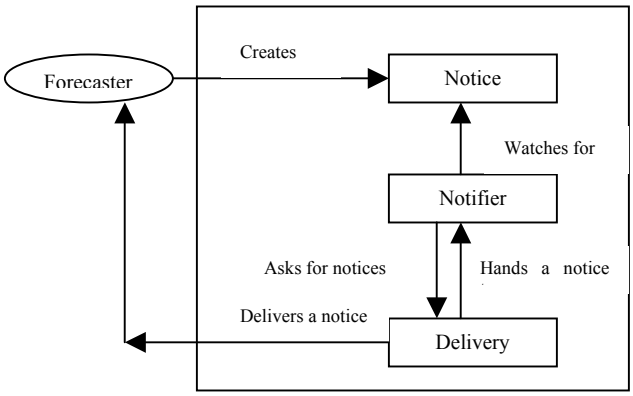

Figure 3. Conceptual Agent Relation Diagram.

Case Study: Candidate Agent List

After looking at Figure 3, the agent relation diagram for createNotice, and applying our agent identification patterns we discover a number of potential agents. We see that we have two external actors that interact with our internal agents and we suggest manager agents that handle the interaction between the internal and external agents. We also discover that the delivery agent wants to be notified of any new notices and we would create a notifier agent, but we have already discovered that agent. We notice that several of our agents will need to interact with the database and add the database agent to our list. Upon further review the Delivery agent may need help in delivering notices, so we suggest a deliveryservice agent. We think that it may be possible that we want several different agents to register their delivery services with a broker agent. For example the Web Page and Fax agents may want to provide their services through a DeliveryBroker agent. Table 1 describes the candidate agent list for our Notice Management System.

Table. 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.

Case Study: Brief Internal Uses Cases

The following case study lists the brief internal uses cases for our Notice Management System. The brief internal use cases are grouped under the service that they should provide. The service name is in bold in order to easily distinguish which brief internal use cases belong to which service.

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.

Service: ViewNotice

Name: ViewNotice

Description:

The forecaster decides to view notices and starts the view notice interface to do so. The NoticeManager provides a view notice interface to the forecaster. The forecaster indicates the state it wishes to view valid notices in. The interface indicates which districts have valid notices to the forecaster. The forecaster selects a district to view a valid notice for. The valid notice is retrieved from the database and displayed to the forecaster.

Service: Start

Name: Start

Description:

The System Administrator decides to start the notice delivery system. The SystemManager creates the proper agents that will be needed in order for the system to allow management and delivery of notices.

Service: Stop

Name: Stop

Description:

The System Administrator decides to shutdown the system.

Case Study: Internal Use Cases

The following case study lists the internal use cases for our Notice Management System. The internal use cases are structured similarly to the brief internal use cases. Some of these internal use cases where not created until after the creation of some of the agent interaction diagrams. The Create goal that can be found under the start service in our internal use cases is an example of new use case that was discovered during the creation of the agent interaction diagrams.

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

Extends: CreateNotice, intention 3

Actors: NoticeManager, Database

Preconditions: The database receives a notice to save.

Success/Postconditions: The notice is saved.

1) The NoticeManager sends a notice to the database to be saved.

Internal use case: WatchForNoticesToDeliver

Actors: Notifier, Delivery, DeliveryService

Stakeholders:
-The notifier watches the information that is inserted into the database.
-The delivery agent wants to know when/what/who it should deliver.
-The deliveryservice agent will format the notice as specified and deliver it.
Preconditions: The system needs to recognize when notices should be delivered.
Sucess/Postcondition: The system recognizes that a notice should be delivered.

Scenario (intentions):
1) The delivery agent creates the notifier.
2) The delivery agent asks the notifier to watch the database for notices, which are a type of weather product.
3) The delivery agent listens to notifier for notices.
The notifier watches the database for notices to be entered.
5) The notifier gives the delivery agent a notice.
The delivery agent sends a request to the delivery service agent to deliver the notice.

Internal use case: DeliverNoticesToDistricts
Actors: DeliveryService, Notice
Preconditions: DeliveryService agent has received a notice to be delivered and how it should be delivered.
Sucess/Postcondition: The notice is delivered in the proper format to the proper districts.
Scenario (intentions):
1) The deliveryservice agent checks the notice for who it should be delivered too.
2) The deliveryservice agent then checks the database on how to properly format the notice for delivery.
3) The deliveryservice agent formats the notice for delivery.
4) The deliveryservice agent delivers the notice as a fax, web page or both.

Service: ViewNotice
Internal use case: ViewNotice
Service: ViewNotice
Actors: NoticeManager, Forecaster, Notice
Stakeholders:
-Forecaster wants fast viewing of valid notices.
-NoticeManager handles the interaction with the forecaster and desires proper valid notice viewing.
-Notice contains all the information about a notice.
Preconditions: Forecaster desires to view a notice area.
Postcondition: Forecaster is able to view the contents of a valid notice.
Scenario (intentions):
1) Forecaster asks the NoticeManager agent to view a Notice.
2) The NoticeManager agent provides an interface to the forecaster for notice viewing.

Internal use case: View
Extends: ViewNotice, intention 2
Actors: NoticeManager, Notice, Forecaster, Database
Preconditions: We need to view a notice.
Success/Postconditions: A notice is viewed.

Scenario (intentions):

1) The notice interface is started for notice viewing.

2) The NoticeManager gets the State from the user.

3) The NoticeManager gets the valid notices from the database.

4) The NoticeManager provides an interface with the districts that have valid notices.

5) The Forecaster selects a district to view a notice for.

6) The NoticeManager displays the notice contents to the Forecaster.

Internal use case: GetValidNotices

Extends: viewNotice, intention 3

Actors: NoticeManager, Database

Preconditions: The database receives a request for valid notices in a state.

Success/Postconditions: All the valid notices are returned.

Scenario (intentions):

1) The NoticeManager requests all the valid notices for a state from the Database.

Service: Start

Internal use case: Start

Actors: SystemManager, Database, Delivery

Preconditions: The system is started.

Success/Postconditions: The proper services are started to enable management and notice delivery.

The SystemManager creates the database to provide database access to the various agents of the system.

The SystemManager creates the delivery agent to enable the delivery of notices.

Internal use case: Create

Extends: Start, intention 2

Actors: SystemManager, Delivery, Notifier, DeliveryService

Preconditions: The delivery agent is created.

Success/Postconditions: The proper agents are created which allow the delivery agent to deliver notices.

1) The delivery agent creates the Notifier so it can be notified of new notices.

2) The delivery agent creates the DeliveryService agent, which it will hand the notices that need to be delivered to.

3) The delivery agent gives a list of notices to the Notifier, which it wishes to be notified for.

4) When the delivery agent receives a notice from the Notifier it hands the notice to the DeliveryService for delivery.

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.

Service: ViewNotice
Goal: ViewNotice
Belief: StateDB
Reason: We need a district id for the notice we wish to view.

Goal: View
Belief: Notice
Reason: We need a notice to view.

Goal: GetValidNotices
Belief: StateDB, NoticeDB
Reason: We need a list of district ids to check for valid notices for those districts.

Service: Start
Goal: Start
Belief: Delivery, Database
Reason: This goal needs to know which delivery and database services to start.

Goal: Create
Belief: Notifier, DeliveryService
Reason: This goal needs to know, which Notifier and DeliveryService agents to start.

Case Study: Agent Interaction Diagrams

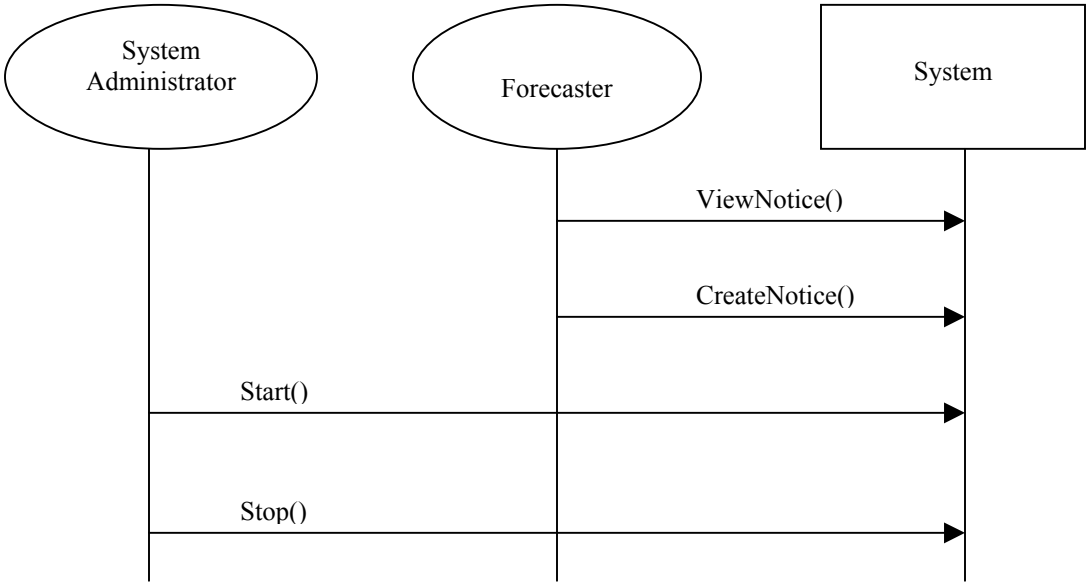

Figure 4. System Services Interaction Diagram.

Figure 4 is a variation of the agent interaction diagram that shows the services that will be provided by our system. The services that our system should provide can be extracted from the titles of the brief external use cases. This system service interaction diagram provides the developer with a visual picture of the external services that the system will provide. In our notice management case study we have described the CreateNotice, ViewNotice and Start services in detail. We choose not to provide a detailed documentation of the Stop service because it does not seem to provide any useful insight into the system architecture.

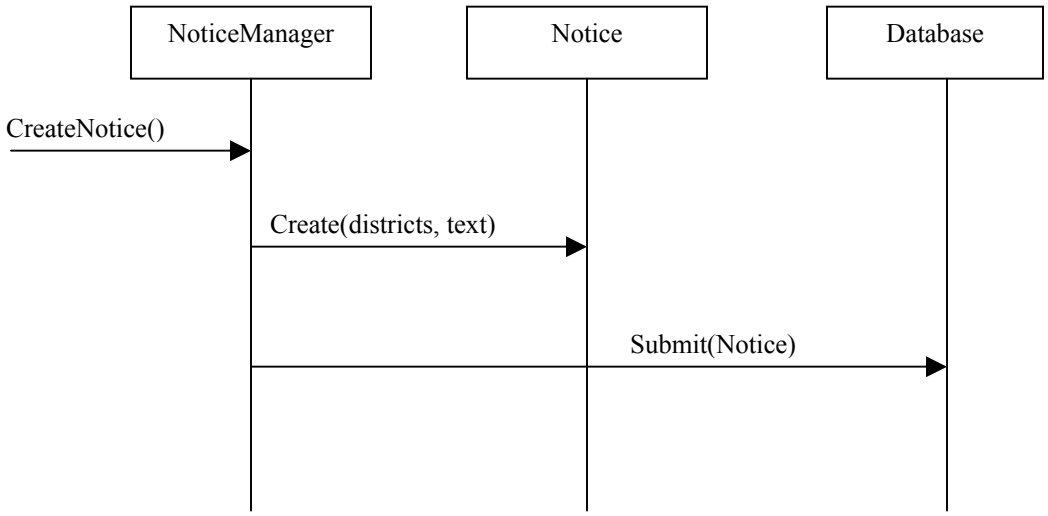

Figure 5. Agent Interaction Diagram: NoticeManager.CreateNotice(). The Forecaster invokes the CreateNotice goal of the NoticeManager. The NoticeManager then uses the Notice.Create() goal in order to create a Notice. The NoticeManager uses the Database.Submit(Notice) goal in order to store the notice in the database.

The agent interaction diagram in Figure 5 describes the internal actions that take place when an external agent (a Forecaster in this case) invokes the CreateNotice goal that can be found in Figure 4. The internal use cases for the CreateNotice goal are used for the creation of Figure 5. The CreateNotice goal should not be confused with the CreateNotice service. The CreateNotice service embodies all the goals listed under the CreateNotice service in our brief internal use cases.

Figure 6 details the internal actions that take place when the Start goal is invoked. We choose to have the SystemManager create both the Database agent and the Delivery agent in Figure 6. The Database agent meets the description of the long-lived agent pattern because it needs to be available to many different agents inside our system and it needs to be available for the entire life of the system. We have the SystemManager create the Delivery agent because we want notices to be delivered as soon as the system is started.

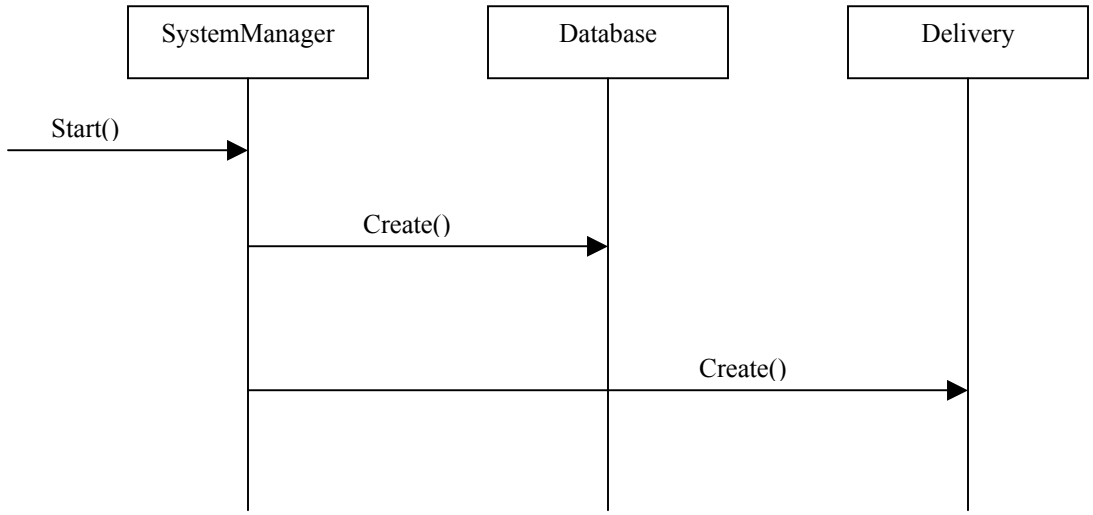

Figure 6. Agent Interaction Diagram: SystemManager.Start(). When the SystemManager.Start() goal is invoked by the SystemAdministrator then the SystemManager invokes the Create() goals for the Database and Delivery agents.

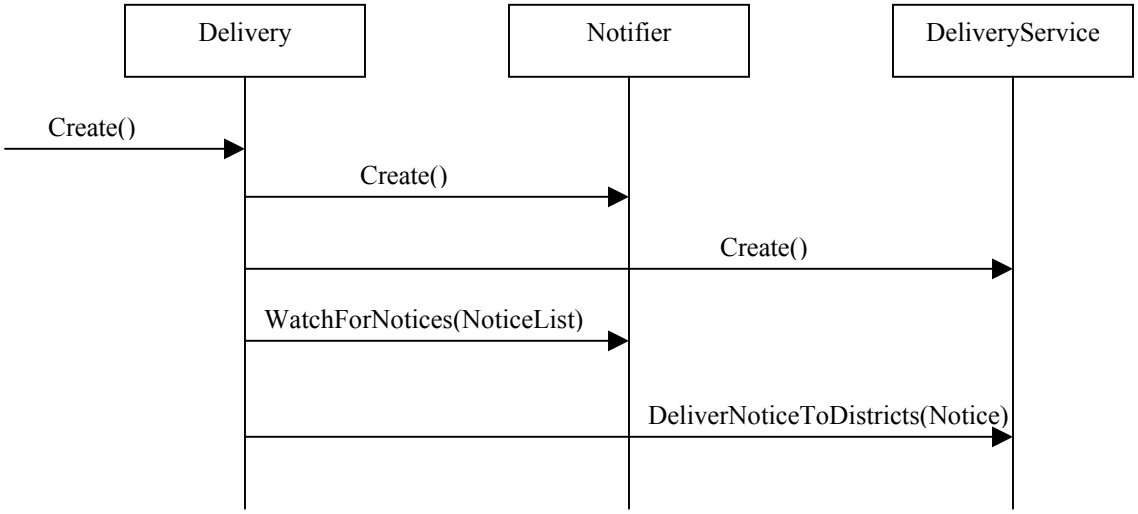

Figure 7. Agent Interaction Diagram: Delivery.Create(). When the Delivery agent receives a create request from the SystemManager it invokes the Notifier.Create(), DeliveryService.Create() and Delivery.WatchForNotices(NoticeList) goals. The Notifier will transparently pass new notices to the Delivery agent, described by the WatchForNotices(NoticeList) goal, which will then invoke the DeliveryService.deliverNoticeToDistricts(Notice) goal.

Figure 7 is a more detailed description of the actions that take place when the SystemManager invokes the Create goal of the Delivery agent in Figure 6. The Delivery agent creates the Notifier agent, so it can be receive new notices from the system. The Delivery agent creates the DeliveryService agent, which it will use to deliver any notices it receives. The Notifier.WatchForNotices(NoticeList) goal is invoked which tells the Notifier which notices we wished to be notified of. The DeliveryService.DeliverNoticeToDistricts(Notice) goal is invoked whenever a notice is received from the Notifier.

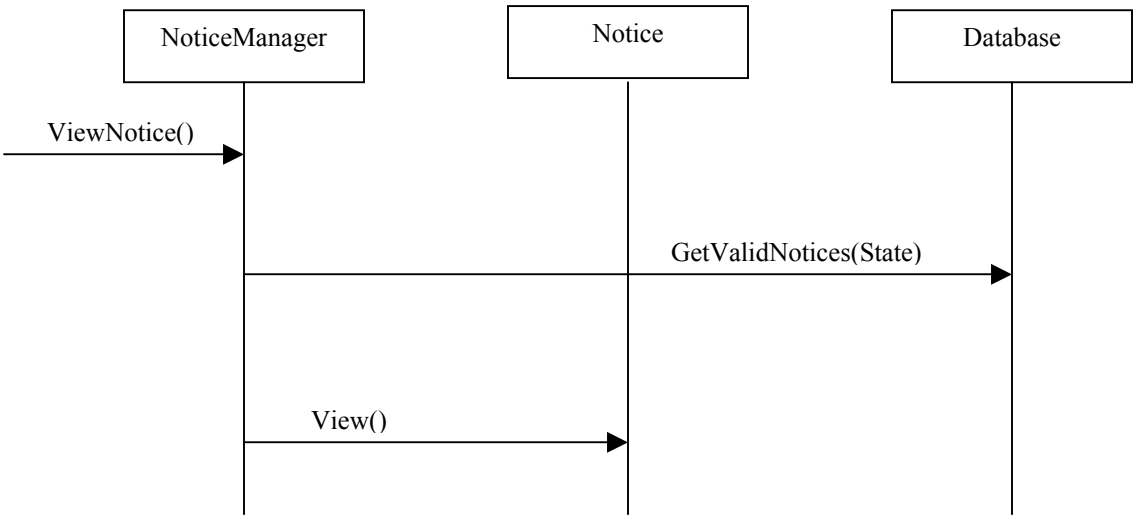

Figure 8. Agent Interaction Diagram: NoticeManger.ViewNotice(). The NoticeManager.ViewNotice() goal is invoked by the Forecaster. The NoticeMangar invokes the Database.GetValidNotices(State) goal and receives a list of Notices. The NoticeManager can then invoke the Notice.View() on each Notice.

Figure 8 is an agent interaction diagram that provides a more detailed description of what happens when an external agent invokes the systems ViewNotice goal. The Database.GetValidNotices(State) allows the NoticeManager to get the list of valid notices for a state from the database. The Notice.View() goal allows the NoticeManager to view the contents of each notice.

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: Notice

BDI list:

Desire: Create

Pre-condition: NoticeManager needs to create a new notice.

Belief: StateDB

Post-condition: Notice is created.

Collaborators: NoticeManager, Database

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.

Desire: View

Pre-condition: We need to view a notice.

Belief: Notice

Post-condition: A notice is viewed.

Collaborator: NoticeManager

Intentions:

The notice interface is started for notice viewing.

The NoticeManager gets the State from the user.

The NoticeManager gets the valid notices from the database.

The NoticeManager provides an interface with the districts that have valid notices.

The Forecaster selects a district to view a notice for.

The NoticeManager displays the notice contents to the Forecaster.

Agent: Delivery

BDI list:

1) Desire: Create

Preconditions: The delivery agent is created.

Belief: Notifer, DeliveryService

Success/Postconditions: The proper agents are created which allow the delivery agent to deliver notices.

Collaborators: SystemManager, Delivery, Notifier, DeliveryService

1) The delivery agent creates the Notifier so it can be notified of new notices.

2) The delivery agent creates the DeliveryService agent, which it will hand the notices that need to be delivered to.

3) The delivery agent gives a list of notices to the Notifier, which it wishes to be notified for.

4) When the delivery agent receives a notice from the Notifier it hands the notice to the DeliveryService for delivery.

Agent: DeliveryService

BDI list:

Desire: DeliverNoticesToDistricts

Pre-condition: We receive a notice that needs to be delivered.

Belief: StateDB

Post-condition: Notice is delivered in the proper format to the proper districts.

Collaborator: Delivery

Intentions:

The deliveryservice agent checks the notice for which it should be delivered too.

The deliveryservice agent then checks the database on how to properly format the notice for delivery.

The deliveryservice agent formats the notice for delivery.

The deliveryservice agent delivers the agent as a fax, web page or both.

Agent: Database

BDI list:

1) Desire: Submit

Pre-condition: The database agent receives a notice to save.

Belief: NoticeDB

Post-condition: The database agent stores the notice properly.

Collaborator: NoticeManager

Intentions:

1) The NoticeManager sends a notice to the database to be saved.

2) Desire: GetValidNotices

Pre-condition: The database receives a request for valid notices in a state.

Belief: NoticesDB, StateDB

Post-condition: All the valid notices are returned.

Collaborator: NoticeManager

Intentions:

1) The NoticeManager requests all the valid notices for a state from the Database.

Agent: Notifier

BDI list:

1) Desire: WatchForNotices

Pre-condition: The system needs to recognize when notices should be delivered.

Belief: NoticeDB

Post-condition: Waiting to receive notices.

Collaborator: Notifier

Intentions:

1) The delivery agent creates the notifier.

2) The delivery agent asks the notifier to watch the database for notices, which are a type of weather product.

3) The delivery agent listens to notifier for notices.

4) The notifier watches the database for notices to be entered.

5) The notifier gives the delivery agent a notice.

6) The delivery agent sends a request to the delivery service agent to delivery the agent.

Agent: SystemManager

1) Desire: Start

Pre-condition: The system is started.

Belief: Database, Delivery

Collaborator: System Administrator (external)

Intentions:

1) The SystemManger creates the database to provide database access to the various agents of the system.

2) The SystemManager creates the delivery agent to enable the delivery of notices.