Applying Enterprise JavaBean in an Internet Workflow Management System

Chia-Lin Hsu, Feng-Jian Wang, Ta-Chun Lin
Department of Computer Science and Information Engineering
National Chiao Tung University
1001 TA HSUEH ROAD, HSINCHU, TAIWAN 30050, ROC
{glshu, fjwang, djlin}@csie.nctu.edu.tw

Abstract

A workflow system provides enterprises the automatic and paperless process management. The idea of Enterprise JavaBean (EJB) is to utilize components from various vendors to construct an application, which has the characteristics of scalability, security, distributed, and fast development. In the paper, an EJB module and the enterprise beans are implemented. Workflow designer may use this module to invoke the business methods defined in the enterprise bean on the outer application server. The EJB module may facilitate the development of a component of a workflow system, reduce the designing time, and reuse the existing components to fulfill the function needed.

Keywords: workflow management, component development, Enterprise JavaBeans (EJB)

1 Introduction

The workflow systems have come out for decades. In recent years, owing to the maturity of internet environment, designing a workflow system used within an enterprise, even across enterprises, becomes an unavoidable trend. In an internet workflow management system based on a single database server, the performance of the system is limited to network traffic, the system's computing power, and the access handle of the database server. For example, Agentflow system [1] is one of the internet workflow management systems based on a database server and originated from our lab. [2].

On the other hand, with the thirst of rapid software development and deployment, the growing requirements of easy manageability, security, and the aspect of software reusability are emphasized. However, components used to form a document in most workflow systems are usually static items. These components have their owned properties and can only do some predefined jobs, such as components in Agentflow's FormDesigner [1]. Therefore, how to enhance a workflow system with a component which provides the characteristics mentioned above is an interesting issue.

A new software design paradigm has occurred to solve above problems since the Enterprise JavaBeans (EJB in short) specification [3][4] first presented in 1998. The EJB architecture defines a component-based model to simplify the development of distributed enterprise applications, which develops and deploys the reusable components via the network. Following the EJB specification, component/application developers can devote themselves to business logic design.

This paper proposes an EJB module which introduces the features of EJB into Agentflow system. An EJB module plugged into Agentflow system becomes a bridge between Agentflow system and EJBs. An electronic form carrying the EJB module can utilize business methods provided in an EJB which is deployed to an application server. Such an EJB module brings the following advantages to Agentflow System, such as flexibility, reusability, rapid component/application development, security, and safe transaction, etc. With this mechanism, different Agentflow systems can also communicate and exchange data to each other through the .EJB module and the application server in which the corresponding enterprise beans deployed.

In this paper, Section 2 introduces some background of this paper. Section 3 describes methodologies and design strategies of the EJB module. Section 4 simply introduces implementation of plug-in interfaces, methods, and corresponding configurations of application. Section 5 describes the conclusion and future work.

2 Backgrond

The first section introduce the EJB technology, the second and the third section will give some brief of Workflow Management System and, Agentflow system. And the last of all is the problem we see in the real world.

2.1 Enterprise JavaBean

Enterprise JavaBean (EJB) is a server-side distributed component-based architecture that simplifies the process of building enterprise applications in Java. By using EJB, one can write scalable, reliable, and secure applications without writing his own complex distributed component framework. EJB is about rapid application development for the middle tier business application which between the client and database and designed to support application portability and reusability across any vendor's enterprise middleware services.

Every EJB must be deployed in an EJB container which resides in an application server. Methods for locating, creating, and removing instances of EJB classes are defined in the home interface and implement in the home object. The remote interface lists the business methods available for the enterprise bean and the EJBObject is the client's view of the enterprise bean and implements the remote interface. Engineers only define the home interface and the remote interface. The EJB system would generate the implementation of them automatically. During runtime, EJB container is responsible for managing the EJBObject. Each time the client invokes the EJBObject's methods, the EJB container first handles the request before delegating it to the Enterprise Bean.

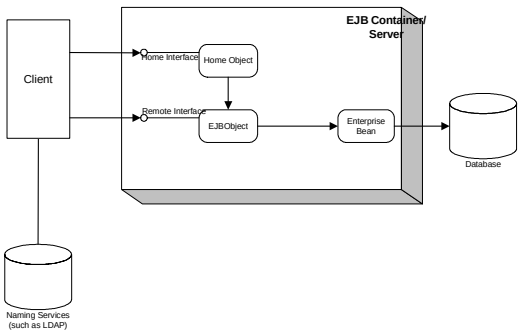

Figure 1 The EJB Model

2.2 Workflow Management System

Workflow Management Systems (WFMS) [8] are specialized types of software systems used to assist in computer supported collaborative work. WFMS are often referred to as workflow automation since they can automate the tasks or activities undertaken by both people and computer resources of an organization. WFMS are often introduced since they support new ways of working as businesses reengineer. They are used in mission critical areas such as in financial services for issuing loans and for common administrative functions such as processing purchase orders.

The Workflow Management Coalition (WfMC in short) [9][10] describes workflow as: "The computerized facilitation or automation of a business process in whole or part" and a Workflow Management System as: "A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications."

All workflow systems are process oriented [8]. A process definition, a representation of what should happen, is created, and it typically comprises some sub-processes. Each process and sub-process comprises some activities. Making a payment or not making a payment is an activity. And an activity consists of work items which utilize workflow queue to schedule the process and cooperate with resources such as human or computer. Figure 2 illustrates the WfMCs work flow reference model. The original idea comes from an initial study sponsored by the U.S. Department of Defense which kick-start the WfMC work in this area.

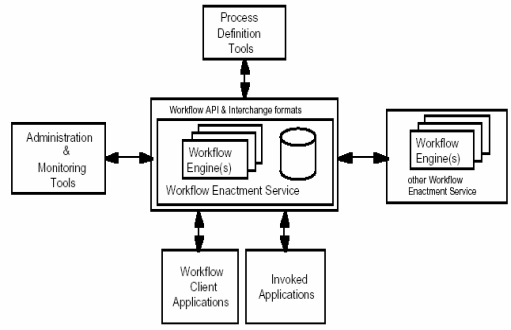

Figure 2 WfMC reference model

2.3 Agentflow

The real workflow management system studied in the paper is Agentflow system. The idea of Agentflow system comes from "Software

Process Management Environment on the Internet" [2]. Agentflow system is completed based on the n-tier software architecture, and is composed of 3 parts which are Client/User Interface, Flow Designer/User Interface, and Workflow Server. The system architecture is illustrated in following Figure 3.

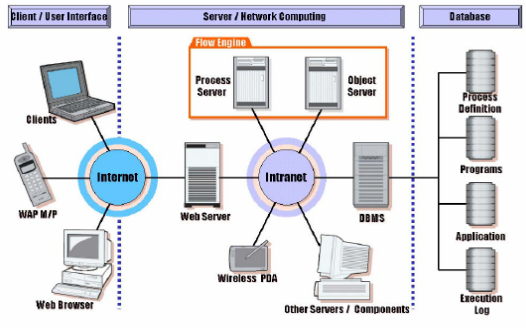

Figure 3 Agentflow architecture

The main part of client/user interface is Agenda [1], which is a client side program of Agentflow. Users can install Agenda on different platform and access Agentflow server through internet. Users can also know which jobs are currently needed to deal with, which process can be initialized, and which explicit programs needed to be called, and how to trace the processes. Users even do not require to install Agenda. They can just download Agenda through Web.

The user interface for workflow designer, named FormDesigner, takes the responsibility for integrating software component in the system. FormDesigner is a graphical development tool to construct electronic forms. It supports some basic user-interface and database related components, and user can build a form with FormDesigner simply by drag-and-drop the components.

Agentflow utilizes the concept of network centric computation. In this concept, the network environment is thought of as a super computer and every program which user runs currently is directly downloaded to the client side. The workflow server plays the role of dispatching tasks to each user. And each client can receive the newest task list.

2.4 Motivation

Currently, Agentflow System, a WfMC model, extends the editing power called FormDesigner to design the artifact or data in the workflow system. Here, an artifact is extended as an interactive electronic form with components predefined and developed. Within FormDesinger, many components are available. And developer can also design some specific components which can be plugged into electronic form through "add component" function provided in FormDesigner. Except plug-in components, the general characteristic of the previous components is that they are static items. Designers can not define their own properties of a component in an electronic form. A typical example is component usability.

With the thirst of rapid software development and deployment, the growing requirements of easy manageability and security, and the aspect of software reuse, the EJB technology and its application server, which provides EJB container/server, need be introduced into Agentflow system. As mentioned at section 2.1, the EJB component, might be applied to solve such a problem. After the EJB component is plugged into the FormDesigner, one can just design the business logic defined as requirements and skip the low-level network implementation, which heavily reduces the development time. Through the "deploytool" [11] or other tools provided by application server, one can easily manage the states of an application. Take Weblogic Server [12] version 6.0 for example, it provides an user-friendly web interface by which administrators can modify the application state, deploy applications and setup the server. The consideration of security is fulfilled with the secure hypertext transfer protocol (HTTPS) provided by application server. Last but not least, software reuse is the fundamental of EJB architecture that reduces the timely cost for developers.

This research might provide a useful reference for researchers to increase productivity through integration of the EJB technology and workflow system.

3 Agentflow Enhancement with EJB

The design of Agentflow and EJB interoperability is proposed in this section. The first section introduces the architecture of enhanced Agentflow FormDesigner with EJB module. The second section presents the process of how the enterprise bean is designed, connected, and used. In the third section, the strategies of designing EJBs are introduced. The fourth section shows the method of authentication and authorization access to enterprise bean. At last, a pattern is introduced into EJB to improve the performance of EJB implementation and facilitate developer's work.

3.1 Architecture of Enhanced Agentflow FormDesigner with EJB Module

All workflow systems are process-oriented, and one of the most important parts that come along with process flow is artifacts. Artifacts, also named documents or electronic forms, are the information needed in processing workflow. Within an Agentflow system, developers use the FormDesigner to combine existing components to design an artifact. For remote components, which are used to enhance FormDesigner's function and reduce development time, the EJB module is introduced into Agentflow system.

Through the newly added EJB module, an artifact can utilize the business methods designed in an enterprise bean outside the Agentflow system. Although invoking remote methods generates additional network traffic, when security, safe transaction, and distributed computing are taken into consideration, EJB module might provide a good, convenient, and total solution.

On the other hand, inside the Agentflow, the corresponding code of EJB button generally needs to implement the following interfaces.

* IPlugInComponent interface
* IPlugInCallback interface
* IPlugInDesignTimePolicy interface
* IPlugInRunTimePolicy interface

3.2 Collaboration of FormDesigner and EJB component

The goal of an EJB component module is to bridge the function of EJB to Agentflow system. The steps of plugging an EJB component module into an Agentflow system are listed below and shown in Figure 4.

* Collect data needed to perform action from electronic form.
* Access naming service (1. and 2. in Figure 4).
* Create remote object (3. and 4. in Figure 4).
* Invoke business method (5. and 6. in Figure 4).
* Activate real enterprise bean (7. in Figure 4).
* Return computation results to electronic form.

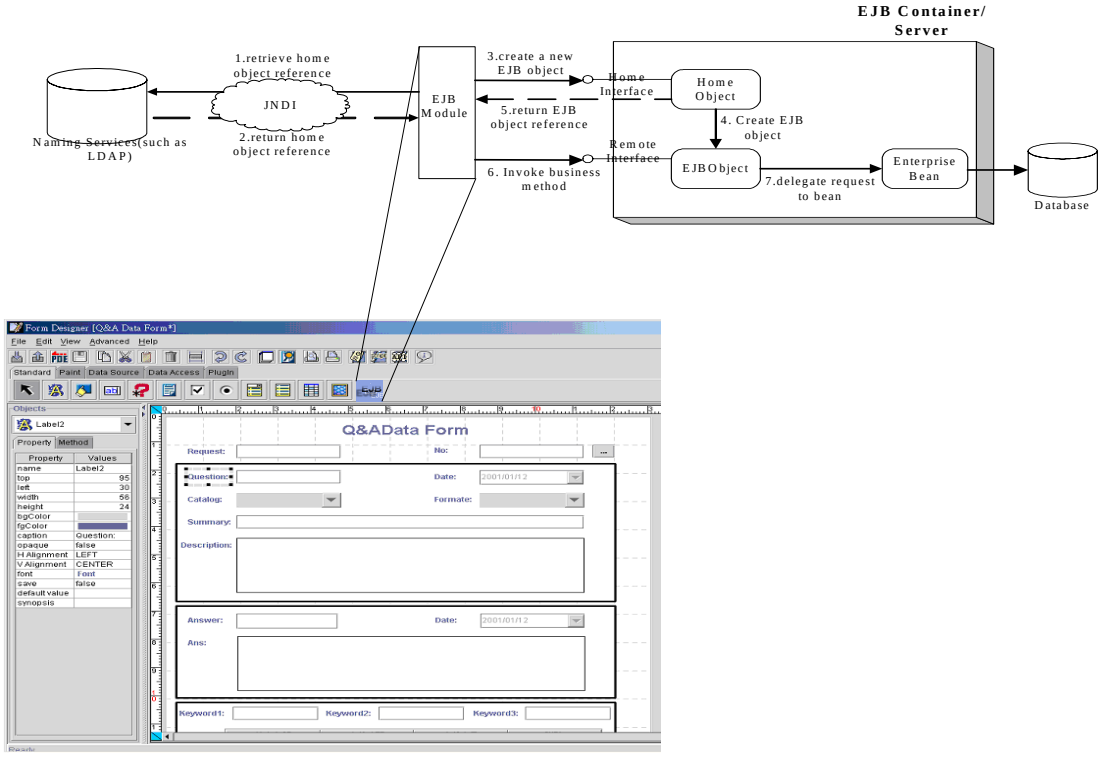

Figure 4 The process of EJB component module design

Adding an EJB module to the ToolBar is the first step. Next, place the module into the form itself. Some properties and methods are now required to setup, such as JNDI name and methods, to make the connection between the module and the outer EJB Server. After collecting all the resources needed to initial a method reside on the server, the action listener of the module receives the order and start accessing the naming service (LDAP) [13]. Through naming service

looks up the JNDI name and returns a home object reference. The EJB module then uses this reference to create an EJBObject. When previous moves are finished, EJB module may now invoke business methods on remote interface which will be delegated to real enterprise beans. Finally, the result is either written into repository or returned to EJB module which will then be filled into a column or be displayed.

Agentflow component developers can benefit from EJB approach. They need not handle the low-level transaction and state management details, multi-threading, connection pool, and other complex low-level APIs. Actually, the EJB container deals with the underlying process, stub, skeleton, and remote method invocation (RMI) [14] on behalf of the enterprise bean developers. Therefore, enterprise bean developers can focus on design the real business logic rather than devote too much time on coding the communication protocol.

3.3 Design of Enterprise Bean

Designing the session and entity beans follows the EJB specification. Namely, one shall define the interfaces and implement the bean code. And the design of enterprise bean affects the performance of workflow process when a transaction has very complex business logic. Some methods are presented to optimize the enterprise bean.

1. *Use container-managed persistence*
Container-managed persistence (CMP) [3] not only reduces the coding effort but also enable potential optimization within the container and container-generated database access code. Rather than coding JDBC operations in the bean class, the container implicitly performs all data operations on behalf of the bean. The container has access to the in-memory buffer of the bean which allows it to monitor for any change in the buffer. Storing the buffer to the database before committing a transaction can be avoided if the buffer has not changed. This avoids unnecessary expensive database calls.

Another instance of this optimization is called find methods, which

* Gets the reference of an entity bean from the instance pool.
* Retrieves the primary key.

A find method usually follows a method which retrieves the record data into the buffer. CMP allows for optimizing the above two methods into a single database access. In this way, the access time might be shorter.

2. *Always cache references obtained from lookups and find calls*
Reference caching is useful for both entity beans and session beans. JNDI lookups of EJB resources, such as DataSources, bean references, or even environment entries can be very costly, and it is simple to avoid redundant lookups. To solve this problem, one can:

* Define a reference as an instance variable.
* Look it up in the setEntityContext (for entity beans; in method setSessionContext for session beans).

The setEntityContext method is called only once for a bean instance, so the time of redundantly looking up all required references can be saved, especially at the database access methods, ejbLoad and ejbStore. These two methods might be called frequently.
Calls to the find methods of other entity beans are also heavyweight. These calls may or may not be done at bean initialization callbacks methods like setEntityContext, so developers shall write the code to cache the references resulting from find methods whenever applicable. If the reference is only valid for the current entity, it is necessary to clear the references before the instance gets reactivated to represent other entities. This should be done inside the ejbActivate method.

3. *Close all statements properly*
During BMP implementations, dealing with database access code never leaves the query statements open after database access calls. Each open statement corresponds to an open cursor in the database. (The garbage collector claims the open statement and closes it at garbage collection (GC) time eventually, while users have no right to control over the time the GC kicks in; namely, it is useless to enforce GC by calling the System.gc method.) Leaving statements open causes the database to have excessive open cursors, which use resources in the database. Thus, these statements have to be closed to remove the corresponding database resources for performance.

There are various exceptions, two of which may not be proper.

* It causes the later statements to be ignored.
* It causes the later statements to be opened.

It is a necessary factor to catch these two kinds of exceptions for an exception catch-up algorithm.

3.4 Design Strategy

3.4.1 Secure Access to Enterprise Bean

According to the EJB specification, the EJB container provides the implementation of the security infrastructure; the developer and system administrator define the security policies. In the J2EE implementation, one can utilize "realmtool" [11] for creating the user group, roles, and password. And through the "deploytool", he can setup the permission of each business method in a table that determines the access privilege of each role.

The realm is a collection of users that are controlled by the same authentication policy. First of all, a standalone Java application (client) tries to access a protected business method. The authentication service then verifies the identification of the client. At the third step, the client invokes business method of the enterprise bean and the container performs authorization. When the user group, to which the client belongs, has the right to access enterprise bean, the client is permitted to invoke business methods in enterprise bean.

After setup all the groups, roles, and their relative permission of business methods, one can call the enterprise bean's method under the protection of secure protocols. The intellectual property and some private business methods can be hidden from the client's access. Furthermore, the number of J2EE certificated application servers grow day by day. So, most EJB developers do not have to worry about the application server's support of authentication and authorization.

3.4.2 Design Pattern

The pattern Stateless Helper [15] presented here is about to raise the performance of EJB implementation.

3.4.2.1 Stateless Helper

First, remote communication is in fact no good as expected. Probably the biggest bottleneck in a distributed system is the network. Even if running the system on a LAN, one may still encounter delays when making calls across the network. In general, the more one eliminates remote communication, the better the system will perform. Second, developers should avoid lengthy distributed transactions. In other words, the client should not interact with the entity bean directly. Because entity beans are transactional by nature, they consume significant resources both on the server and in the client. They store their state information in a database and usually have some sort of access control mechanism. Because of these characteristics, function calls to remote entity beans tend to be expensive. Therefore, developers shall keep transactions short and minimize the number of remote method calls.

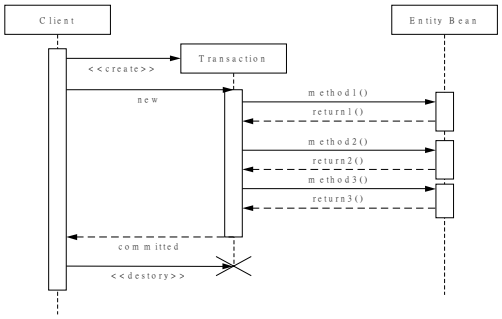

Figure 5 Transaction with multiple entity beans

Figure 5 describes the transaction in which multiple entity beans involve. In this transaction, messages go back and forth between transaction and entity beans that produce a lot of network load. As mentioned above, this situation should be avoided. The Stateless Helper pattern is then proposed to solve this problem.

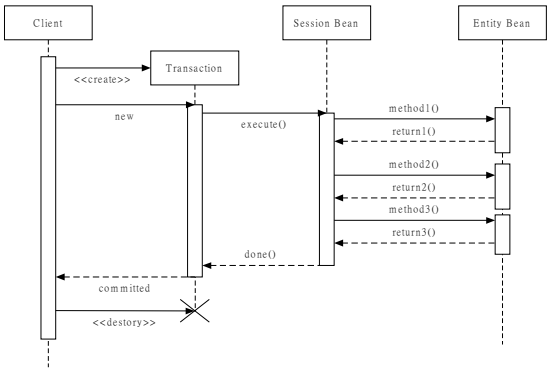

Figure 6 Stateless helper model

In Figure 6, a stateless session bean, which is added as a wrapper of entity beans, accepts the execute() instruction from and returns the done() instruction to the original transaction. The session bean then takes the work of the transaction inside the server instead of internetworking. Obviously, the traffic load can be reduced a lot with this mode, as long as the data consistency can be protected. It is easy inside a server-based system.

4 Implementation

This section discusses how to implement the predefined plug-in interfaces and methods in order to add a new EJB module into FormDesigner.

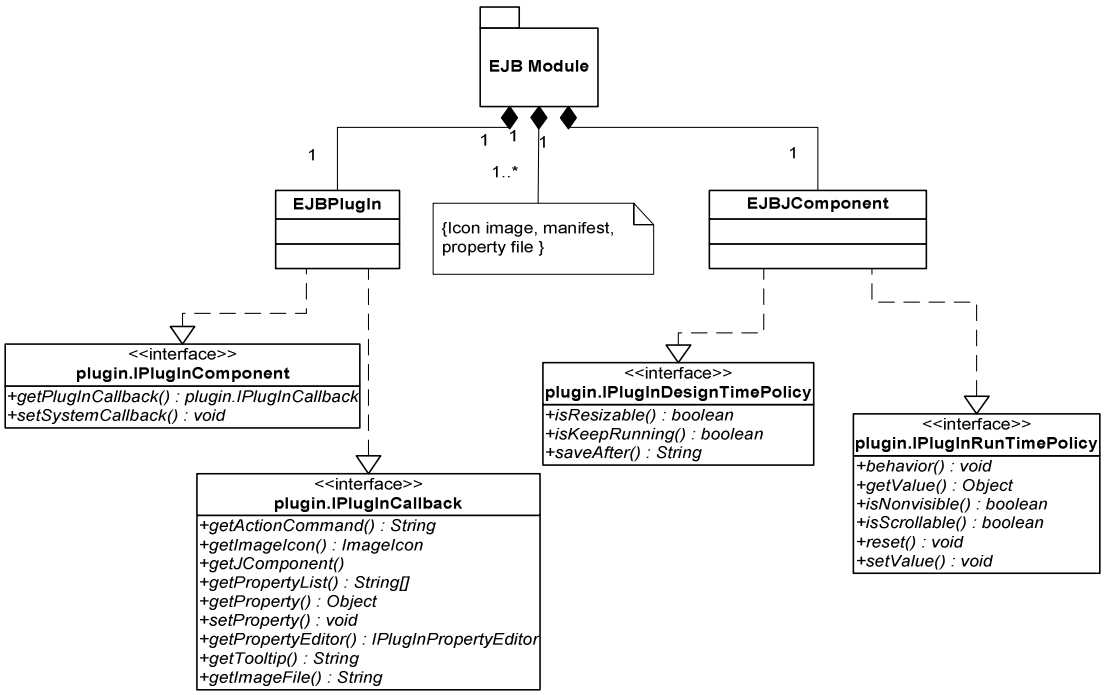

Figure 7 Architecture of a plug-in EJB module

4.1 Interfaces and Methods to FormDesigner

To write a component that can be plugged into the FormDesigner, developers have to implement the plug-in interfaces defined in the FormDesigner. The interfaces can be classified into two categories. One is the handshaking between the plug-in component and the FormDesigner, and the other is the policy when using the plug-in component.

From the categories of the interfaces, such a component can be separated into two parts. One is the basic component such as EJBPlugIn, and the other is its corresponding JComponent such as EJBJComponent). The 2 glue interfaces:

plugin.IplugInComponent
plugin.IplugInCallback

have to be implemented in the basic component, while the policy interfaces:

plugin.IplugInDesignTimePolicy
plugin.IplugInRunTimePolicy

have to be implemented in the corresponding JComponent. Figure 7 depicts the architecture of a plug-in EJB module. The methods and functions in these interfaces are described as following subsections.

4.2 IPlugInComponent Interface

The interface is the glue between the plug-in component and the FormDesigner. The methods defined in this interface are called when the plug-in component is loaded and registered into the FormDesigner. Only two methods are defined in this interface:

* plugin.IPlugInCallback getPlugInCallback():

The method returns the plugin.IPlugInCallback reference of the plug-in component, which is used to retrieve the information of the plug-in component.

* void
setSystemCallback(plugin.ISystemCallback)

The method is used to set the plugin.ISystemCallback reference into the plug-in component. plugin.IsystmCallback interface provides a way that the plug-in component can used to communicate with the FormDesigner.

4.3 IPlugInCallback Interface

The interface provides the related information of the plug-in component, such as the action name of the component, the properties of the component, and so on. The methods defined in this interface are introduced below.

* String getActionCommand():

The method is called after the plug-in component is loaded and registered into the FormDesigner. The FormDesigner uses the action command returned from this method to identify the plug-in component the user asked for and to trigger the action provided by the plug-in component.

* String getImageFile(),
* ImageIcon getImageIcon():

These two methods are called after the plug-in component is load and registered into the FormDesigner. The FormDesigner uses the information returned from these two methods to generate a new button in the toolbar. The difference of these two methods is the location of the image file. The method, getImageFile(), ask the FormDesigner to find the image, while the method, getImageIcon(), provide it's own image in the package of the component.

* JComponent getJComponent():

The method is called when the user asks for the plug-in component. The method returns the corresponding JComponent described before.

* String[] getPropertyList()
* Object getProperty(String)
* void setProperty(Stirng,Object)
* IplugInPropertyEditor
  getPropertyEditor(String)

These four methods are called after the FormDesigner retrieves the corresponding JComponent from the plug-in component. The FormDesigner uses these methods to initialize or persist the corresponding JComponent. The pseudo-code bellowed described the actions for the initialization of the component and for the persistence of the component.

```
Pseudo-code for the initialization of the component
String[] properties = comp.getPropertyList();
for(int i=0; i<properties.length; i++)
{
// get the value of the property
   Object value =
      comp.getProperty(properties[i]);
// get the editor of the property
   IPlugInPropertyEditor editor =
      comp.getPropertyEditor(properties[i]);
// get the string format of the value.
   String str_value =
      editor.convertValueToString(value);
// save the string format to DB
}
```

```
Pseudo-code for the persistence of the component
String[] properties = comp.getProperyList();
for(int i=0; i<properties.length;i++)
{
// get the string format of the value from DB

// get the editor of the property
   IPlugInPropertyEditor editor =
      comp.getPropertyEditor(properties[i]);
// convert the string format of the value
   Object value =
      editor.convertStirngToValue(str_value);
// set the value of the property
      comp.setProperty(properties[i],value);
}
```

4.4 IPlugInRunTimePolicy Interface

The FormDesigner has two modes, design-time mode and run-time mode. When a form designer uses the pre-defined components to draw an electronic form, the form is in design-time mode. And a form is switched to run-time mode when it runs in Agenda.

The IPlugInRunTimePolicy interface is used to specify some values of a component which is needed at FormDesigner's run-time state.

* behavior():
This method is used to set up the pre-action of a component before the component is activated.

* setValue(), getValue():
These two methods are used to set and get the run-time values of a component.

* isNonvisible():
This method is used to define whether a component is visible in Agenda. If the return value of isNonvisible() is true, a component will not show up in Agenda.

* isScrollalbe():
This method is used to claim the scrollable state of a component. If it returns a "true" value, the component is scrollable in Agenda.

* reset():
This method is used to reset the data held by the component before a componet ends its action (post-action).For example, the DbComboBox component uses JCheckBox.removeAllItems() in reset() to remove all items from the item list.

4.5 IPlugInDesignTimePolicy Interface

The IPlugInDesignTimePolicy interface is used to specify some values of a component which is needed at FormDesigner's design-time state.

* isKeepRunning():
This method is used to define whether this component can respond to user input and generate events at design-time. If the return value is false, this component can not respond to events.

* isResizable():
This method is used to define whether this component is resizable or not at design-time. If the return value of this method is true, this component can be resized by drag-and-drop on the edge of the component.

* saveAfter():
Some components can utilize other components' functions in FormDesigner. And a component which is loaded as soon as the component's value/state is saved. Therefore, the component, which makes use of other components' function, should save its values/states after the other components' values/states are saved. The saveAfter() method is used to define whether a component has to be saved after the other components are saved or not.

```
Context ctx =
   getInitialContext(URL, user, password);
…
Object home = ctx.lookup(JNDI_Name);
EJButtonHome home = (EJButtonHome)
      PortableRemoteObject.narrow
         (home, EJButtonHome.class);
```

```
Static Context getInitialContext(String url, String
user, String password) {
   Properties p = new Properties();
   p.put(Context.INITIAL_CONTEXT_FACTORY,
     “weblogic.jndi.WLInitialContextFactory");
   p.put(Context.PROVIDER_URL, url);
   p.put(Context.SECURITY_PRINCIPAL,
     user);
return new InitialContext(p);
```

4.6 Connection between EJB Module and Enterprise Bean

As described in section 3.2, one has to retrieve the EJB object reference to invoke the business logic in enterprise bean, and the EJB object reference comes from JNDI lookup and home object's create() method. The following code presents the methods to invoke the business logic in enterprise bean.

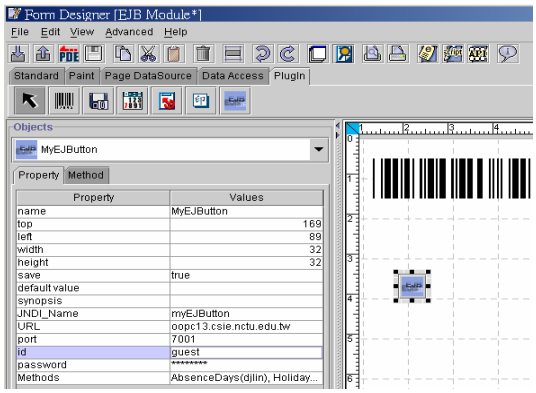

Figure 8 Setup the connection between EJB module and enterprise beans

For example, in figure 8, after setting up

1. JNDI_Name of the home object registered in naming service,
2. URL and port of the application server which enterprise beans are deployed,
3. the id and password of the role for special access privilege,
4. the methods to invoke in enterprise beans,

the EJB module can now invoke the methods in enterprise bean at runtime in Agenda.

5 Conclusions and Future Work

In this paper, an EJB module which can build connection between Agentflow system and EJB is presented. Through the characteristics of EJB, the EJB module helps the Agentflow system to reduce the development time of components and cut down the coding effort of underlying connection. Especially in a distributed internet environment, developers can focus on the business logic without taking care of the network connection and multithreaded consideration. On

the other hand, using the EJB module in Agentflow increases the flexibility and reusability. Workflow designers now have more choices to finish a job rather than use only the components in FormDesigner. And the EJB components from various vendors can be assembled to do a specific work needed by designers. This research provides a useful reference for researchers to increase productivity through integration of the EJB module and Agentflow system.

Nowadays, the Agentflow system uses the centralized n-tier architecture, which may suffer from the bottleneck of network traffic, server's computing power, and limitation of CPU-to-Memory bandwidth, and so on. Therefore, how to distribute the Agentflow server in several platforms, which can share the load and enhance fault tolerance, is a future topic. On the other hand, the database connections are through JDBC in the Agentflow system. Some related database accesses, which can be grouped into a single transaction, are not necessary to call the JDBC functions for each access. The works in the future may include how to retrieve these kinds of database accesses, how to group them into a single transaction, and implement it with session and entity beans. On the other hand, to make the EJB modules in Agentflow to communicate and exchange data with each other is another issue. Last but not least, how to integrate the application server into Agentflow system is another interesting and important issue.

6 References

[1] Flowring Technology Corp., Agentflow system, http://www.flowring.com.tw

[2] Bin-Shiang Liang, Shung-Bin Yan, Ching-Hong Tsai, and Feng-Jian Wang, "Software Process Management Environment on the Internet"

[3] Sun Microsystem, "Enterprise JavaBeans 1.1 Specification", in http://java.sun.com/products/ejb/docs.html

[4] Sun Microsystem, "Enterprise JavaBeans White Papers", in http://java.sun.com/products/ejb/white/

[5] Sun MicroSystem, "Java Naming and Directory Interface (JNDI)"

[6] Sun MicroSystem, "Java Transaction Service (JTS)"

[7] Richard Monson-Haefel, "Enterprise JavaBeans", second edition, O'Reilly, 2000

[8] Layna Fischer, "Workflow Handbook 2001", Future Strategies Inc., Book division, 2001

[9] "The Workflow Management Coalition", http://www.wfmc.org

[10] The Workflow Management Coalition, "The Workflow Reference Model", version 1.1, 1995

[11] Sun Microsystem, "Java 2 Platform, Enterprise Edition", in http://java.sun.com/j2ee

[12] BEA System, Inc., "BEA Weblogic Server", http://www.bea.com/products/weblogic/server/index.shtml

[13] "Lightweight Directory Access Protocol (LDAP)", RFC 2551, 1997

[14] Sun Microsystem, "Java(tm) Remote Method Invocation (RMI)", http://java.sun.com/products/jdk/rmi/

[15] Jeff Gallimore, " Tips on Implementing Enterprise JavaBeans", 1999, http://www.devx.com/upload/free/features/javapro/1999/10mid99/jg1399/jg1399.asp