Constructing an EJB Application in a WFMS

Jian-Wei Wang, Ta-Chun Lin, Huai-Jong Hsu, Feng-Jian Wang

Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan, R.O.C
E-mail: {jwwang,djlin,hjhsu,fjwang}@csie.nctu.edu.tw

Abstract
A workflow system provides enterprises the automatic and paperless process management. The idea of Enterprise JavaBeans (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. Designing a workflow system used within an enterprise becomes an unavoidable trend. In an internet workflow management system of a single database server, the performance is limited to network traffic, the system's computing power, and the access handle of the database server. For example, Agentflow system [1] [2] originated from our lab, is one of the internet workflow management systems based on a database server.

On the other hand, with the thirst of rapid software development and deployment, the growing requirements of easy manageability, security, and software reusability are emphasized. However, components used to form a document in most workflow systems are usually static. These components have their own properties and can only do some predefined jobs. Examples are the 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 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 do not have to beware of the underlying network connection, database access, transaction, multithreading, and other complex low-level APIs when developing applications. They can devote themselves to business logic design. On the other hand, the design of enterprise beans also affects the performance a lot. Well analysis and design of an enterprise bean can largely reduce the network traffic in a system.

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 advantages of flexibility, reusability, rapid component/application development, secure, and safe transaction to Agentflow system. Different Agentflow systems can also communicate and exchange data through the EJB module and its corresponding EJBs.

In this paper, section 2 discusses the background of the workflow and EJB architecture. Section 3 introduces methodologies and design strategies of the EJB module. Implementation of plug-in interfaces, methods, and corresponding configurations of application server are presented in section 4. Section 5 describes the conclusion and future work.

2 Background and Motivation

2.1 Enterprise JavaBeans

With EJB, one can quickly and easily construct server-side components in Java by leveraging a prewritten distributed infrastructure provided by the industry. EJB is a server-side component architecture that simplifies the process of building enterprise-class distributed component applications in Java.

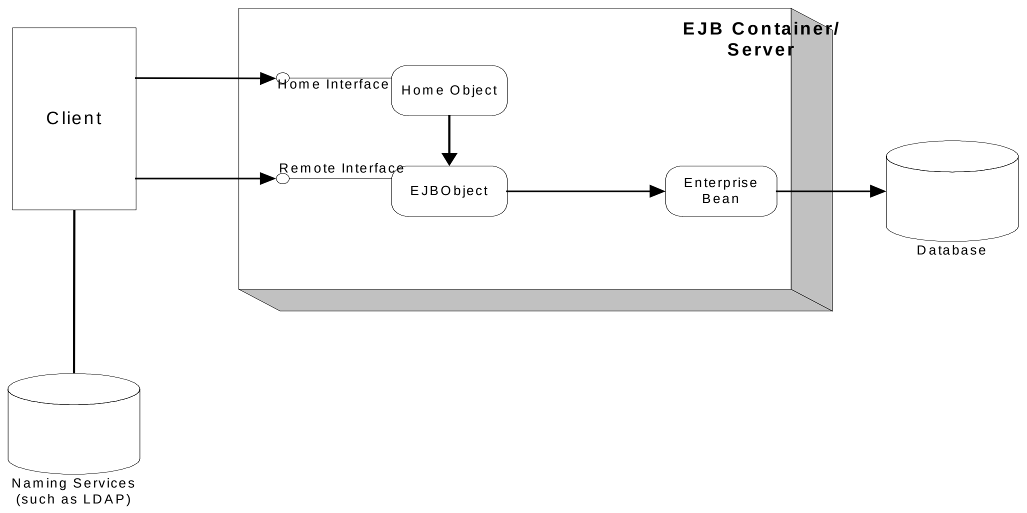


Figure 1 The EJB model

An EJB is a server-side software component that can be deployed in a distributed multi-tier environment. An enterprise bean can comprise one or more Java objects because a component may be more than just a simple object. Regardless of an enterprise bean's composition, the clients of the bean deal with a single exposed component interface. This interface, as well as the enterprise bean itself, must conform to the EJB specification. The specification requires that the beans expose a few required methods; these required methods allow the EJB container to manage beans uniformly, regardless of which container the bean is running in.

A basic EJB architecture is shown in Figure 1 and consists of an EJB server/container, home interface/object, remote interface/object, EJB client, and auxiliary systems such as JNDI [5], JTS [6], security, and so on.

- EJB Server/Container
  The EJB server provides an organized framework or execution environment in which EJB containers can run. It makes available system services for multiprocessing, load balancing, and device access for EJB containers.
  An EJB container acts as the interface between an enterprise bean and lower-level, platform-specific functionality that supports the bean. In essence, the EJB container is an abstraction that manages one or more EJB classes, while making the required services available to EJB classes through standard interfaces as defined in the EJB specification. An EJB client never accesses a bean directly. Any bean access is done through the methods of the container-generated classes, which, in turn, invoke the bean's methods.
  Conceptually, an EJB server may have many containers, each of which may contain one or more types of enterprise beans.
- Home Interface and Home Object
  In EJB specification, methods for locating, creating, and removing instances of EJB classes are defined in the home interface. The home object is the implementation of the home interface. The EJB developer first defines the home interface for their bean. Then the EJB container vendor provides tools that automatically generate the implementation code for the home interface defined by the EJB developer.
- Remote Interface and Remote Object
  The remote interface lists the business methods available for the enterprise bean. The EJBObject is the client's view of the enterprise bean and implements the remote interface. While the enterprise bean developer defines the remote interface, the container vendor provides the tools necessary to generate the implementation code for the corresponding EJBObject.
- Enterprise Bean
  The real enterprise bean itself is contained within an EJB container and should never be directly accessed by anyone but the container. Although direct access may be possible, this is inadvisable as it breaks the contract between the bean and the container. The EJB container should mediate all enterprise bean accesses. For this reason, the enterprise bean developer does not implement the remote interface within the enterprise bean itself. The implementation code for the remote interface is generated automatically by tools the container vendor provides. This prevents inadvertent direct accesses from clients or other beans.

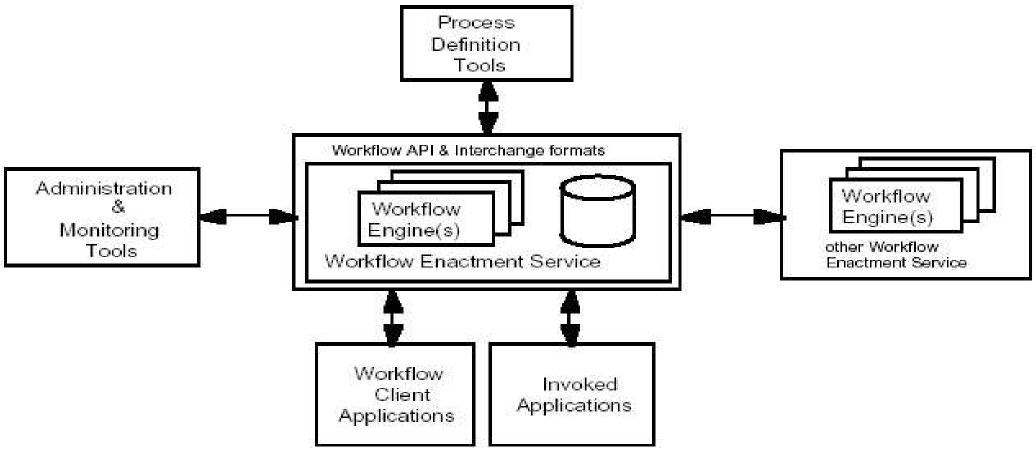

Process Definition Tools
Workflow API & Interchange formats
Administration & Monitoring Tools
Workflow Engine(s)
Workflow Enactment Service
Workflow Engine(s)
other Workflow Enactment Service
Workflow Client Applications
Invoked Applications

Figure 2 WfMC reference model

- Client
  EJB clients locate the specific EJB container that contains the enterprise bean through the Java Naming and Directory Interface (JNDI). They then make use of the EJB container to invoke bean methods. The EJB client only gets a reference to an EJBObject instance and never really gets a reference to the actual enterprise bean instance itself. When the client invokes a method, the EJBObject instance receives the request and delegates it to the corresponding bean instance while providing any necessary wrapping functionality.
  The client uses the home object to locate, create, or destroy instances of an enterprise bean. It then uses the EJBObject instance to invoke the business methods of a bean instance.

2.2 Workflow Management System

2.2.1 Workflow Definition

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. For example, making a payment or not is an activity. An activity consists of work items which utilize workflow queue to schedule the processes 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.

2.2.2 Agentflow Architecture

The 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]. The real system of Agentflow is completed based on the n-tier software architecture. Figure 3 illustrates the system architecture.

- Client/User Interface
  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.

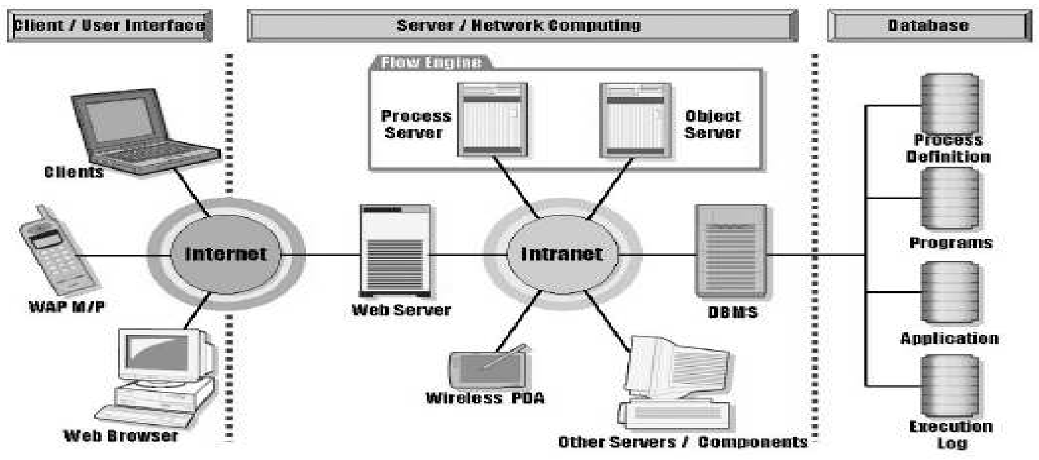


Figure 3 Agentflow architecture

- Flow Designer/User Interface
  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.
- Workflow Server
  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.

2.3 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 built-in components are available. These components are static, i.e. not changeable. Developers are allowed to design some components which can be plugged into electronic form through “add component” function.

As mentioned at sub-section 2.1, the EJB component, might a good choice. 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. Let Weblogic Server [12] version 6.0 be an example, it provides a 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.

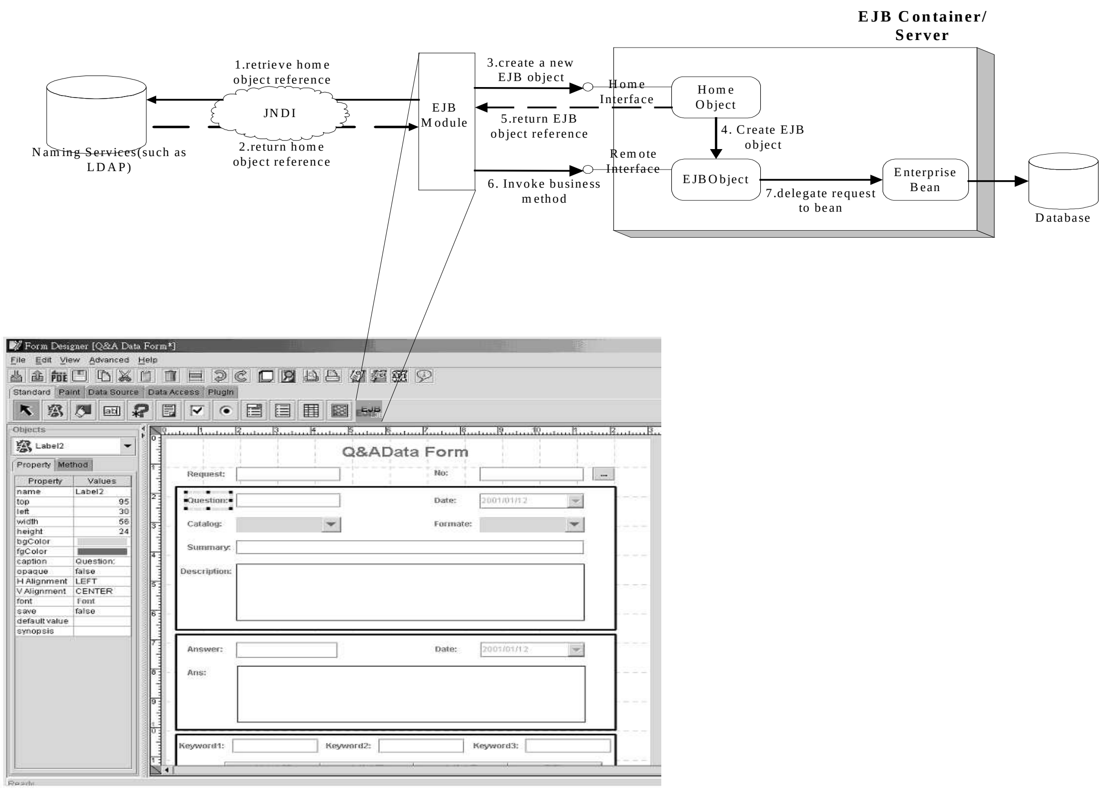
This paper might provide a useful reference for researchers to increase productivity through integration of the EJB technology and workflow system.

3 Agentflow Enhancement through EJB

The design of Agentflow and EJB interoperability is proposed in this section. The first sub-section introduces the architecture of enhanced Agentflow FormDesigner with EJB module. The second sub-section presents the process of how the enterprise bean is designed, connected, and used. In the third sub-section, the strategies of designing EJBs are introduced. The fourth sub-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 the most important part which comes along with process flow is artifact. 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.



Property	Values
name	Label2
top	95
left	30
width	56
height	24
bgColor	▭
fgColor	■
caption	Question:
opaque	false
H Alignment	LEFT
V Alignment	CENTER
font	Font
save	false
default value	
synopsis	

Figure 4 The process of EJB component module design

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.

The first step is to add an EJB module form to the ToolBar. Some properties and methods are then required, such as JNDI name and methods, to connect between the module and the outer EJB Server. After collecting all the resources needed to initiate a method on the server, the action listener of the module receives the order and starts accessing the naming service (LDAP) [13]. The 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 complete, 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 designing the real business logic rather than devoting 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 complex business logic. Three methods are presented to optimize the enterprise bean.

(1) Use container-managed persistence as possible

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 `lookup` calls for obtaining EJB resources, such as `DataSources`, bean references, or even environment entries can be very costly, and it is simple to avoid redundant `lookup` calls. To solve this problem, one can:

- Define a reference as an instance variable.
- Look up it in the method `setEntityContext for entity beans, or in` the 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 setting up 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

In fact, remote communication is no good as expected in a distributed system. And thus the biggest bottleneck is probably 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 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[15] is then proposed to solve this problem.

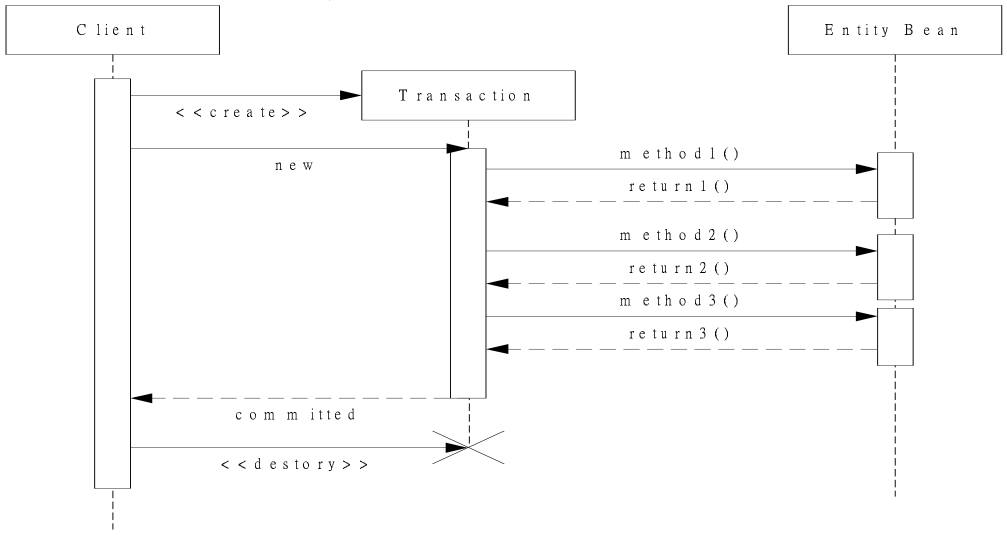


Figure 5 Transaction with multiple entity beans

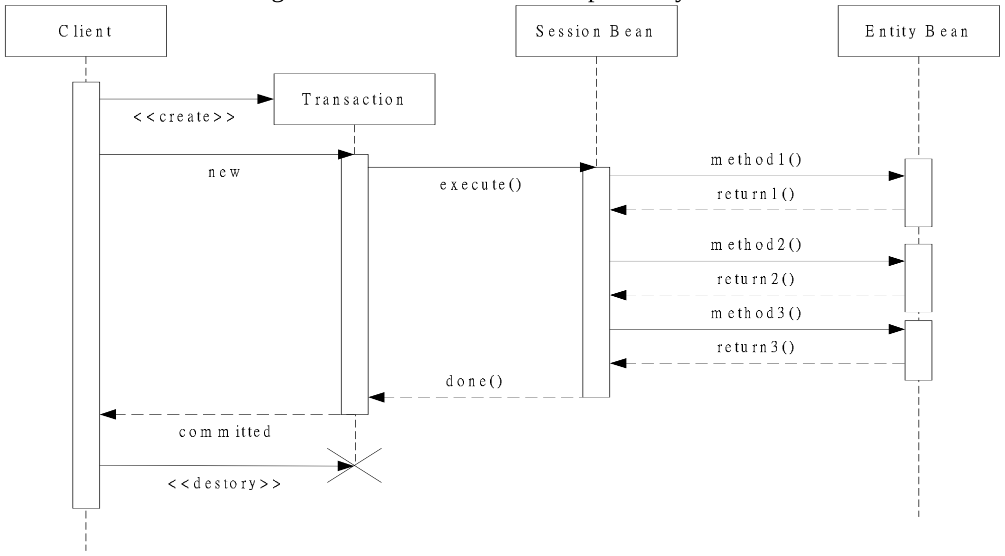


Figure 6 Stateless helper model

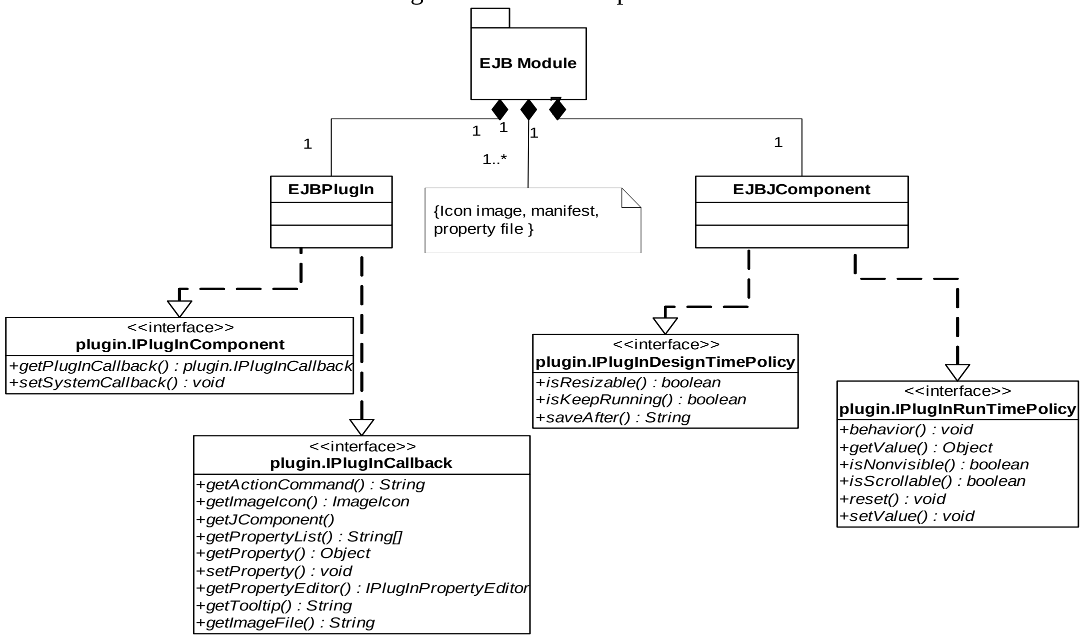

EJB Module
1
1
1
1
1
1..*
EJBPlugIn
{Icon image, manifest, property file }
EJBJComponent
<<interface>>
plugin.IPlugInComponent
+getPlugInCallback() : plugin.IPlugInCallback
+setSystemCallback() : void
<<interface>>
plugin.IPlugInDesignTimePolicy
+isResizable() : boolean
+isKeepRunning() : boolean
+saveAfter() : String
<<interface>>
plugin.IPlugInRunTimePolicy
+behavior() : void
+getValue() : Object
+isNonvisible() : boolean
+isScrollable() : boolean
+reset() : void
+setValue() : void
<<interface>>
plugin.IPlugInCallback
+getActionCommand() : String
+getImageIcon() : ImageIcon
+getJComponent()
+getPropertyList() : String[]
+getProperty() : Object
+setProperty() : void
+getPropertyEditor() : IPlugInPropertyEditor
+getTooltip() : String
+getImageFile() : String

Figure 7 Architecture of a plug-in EJB module

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

Sub-section 4.1 will introduce the components making up the EJB module which help to apply EJBs to FormDesigner, and the interfaces must be implemented to achieve the goal. Sub-section 4.2 is a simple case to indicate how the mechanism works in real case.

4.1 EJB Module Implementation

Figure 7 depicts the architecture of a plug-in EJB module. The EJB Module primarily contains three parts. They are basic component (e.g `EJBPlugin`), its corresponding JComponent (e.g `EJBJcomponent`) and some resources i.e icon image, manifest, and property files. The basic component needs to implement following two interfaces, `IPluginComponent` and `IPluginCallback`. And interfaces, `IPluginDesignTimePolicy` and `IPluginRunTimePolicy`, for plug-in component using policy would be implemented in the corresponding JComponent.

`IPluginComponent` interface defines the methods which are needed for gluing 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. In the mean time, `IPluginCallback` interface provide the related information of the plug-in component, such as the action name of the component, the properties of the component, and so on.

Form has dual states, design state and running state. During the design state, users draw an electronic form in FormDesigner with pre-defined components. Subsequently, during the running state, users access the electronic form designed by FormDesigner through Agenda within a running workflow. `IPlugInRunTimePolicy` interface is used to specify some values of a component needed during run-time state. Comparatively, `IPlugInDesignTimePolicy` interface defines methods offering values of a component during design state.

With the module, a user can put the component relating to EJB on an electronic form with FormDesigner during design time, and invoke them through JNDI services during run-time.

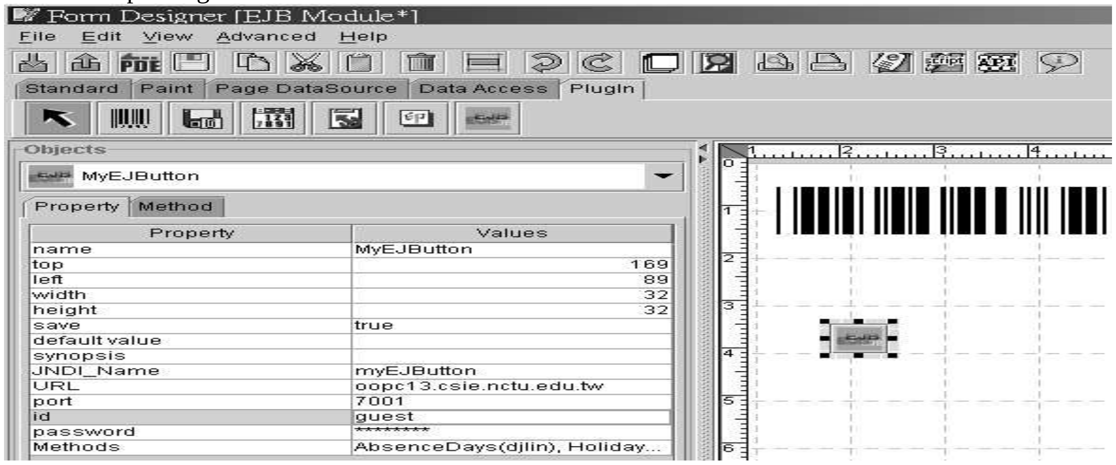
The interfaces implemented will accomplish the task one need with the methods they offer.

4.2 An Example

For Example, in figure 8, users can add the JComponent, MyEJButton, on the electronic form with FormDesigner during design state. After setting up the following features:

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 passwd of the role for special access privilege
4. the methods to invoke in enterprise beans


Form Designer [EJB Module*]
File Edit View Advanced Help
Standard Paint Page DataSource Data Access PlugIn
Objects
MyEJButton
Property Method

Property	Values
name	MyEJButton
top	169
left	89
width	32
height	32
save	true
default value	
synopsis	
JNDI_Name	myEJButton
URL	oopc13.csie.nctu.edu.tw
port	7001
id	guest
password	*********
Methods	AbsenceDays(djlin), Holiday...

Figure 8 Setup the connection between EJB module and enterprise beans

Information of the component would be set into the electronic form. When the form is used during run-time with Agenda, through the component, EJBmodule could invoke the methods in enterprise bean for various purposes. Then we can claim that we have reached the achievement using the components which has been existed on application server through EJB module.

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.

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. 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.

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™ 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