Component-Based Platform for a Virtual University Information System

Dr. IVAN GANCHEV, Dr. MAIRTIN O’DROMA, FERGAL McDONNELL
Department of Electronics and Computer Engineering
University of Limerick
National Technological Park, Limerick
IRELAND

Abstract:- Design ideas for a Virtual University Information System, VUIS, are presented. The design is an interface-centric component-based architecture. Basic services are decomposed into sets of monadic services, each of which is implemented as a reusable software EJB component. The interaction patterns and communication interfaces between components to realise foreseen and un-foreseen basic services are described. Considerations of scalability, security, latency, user interface design issues (e.g. acceptable response time, and speed) are addressed. The paper includes an EJB application example showing this approach towards modelling and how it benefits the design of the VUIS.

Key-Words: virtual university, system architecture, component-based system, EJB, monadic services, eLearning, eLectures

1 Introduction
The Virtual University Information System (VUIS) proposed here is designed as a web-based interactive eLearning system for students, through which instructors and tutors may supplement the learning exercises for courses/modules through the provision of eLectures, eLabs, eTutorials, eSeminars, online/off-line consultation facilities, eTests, eExams, eAssignments etc.

A principal design approach adopted for the VUIS development is that VUIS functionality should be decomposed into several standardized atomic functions called basic services, such as listed above. These in turn have their functionality further decomposed into what we call monadic services [1]. For a particular basic service request to be effected several monadic services (grouped into one VUIS transaction) could be activated. A list of the VUIS basic services, monadic services and their parameters is shown in Table 1. An example of the monadic services and their parameters associated with one particular basic service (eLecture) is displayed in Table 2.

<table>
<thead>
<tr>
<th>Basic Services</th>
<th>Monadic Services</th>
<th>Monadic Service Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>eLecture, eTutorial, eLab, eTest, eAssignment, eSeminars, notice board, discussion board, eMail contact, virtual classroom, groups, course settings and administration</td>
<td>Create, Download, Add/Upload, View, Modify/Edit, Remove/Delete, Submit/Complete, Grade, Track, Import, Export, Group, Contact, Report, Search, Track, Query, Select/Mark</td>
<td>serviceTypeBasicServiceName, fileName, material, BasicServiceSubmission, BasicServiceResult, StudentID, QueryType</td>
</tr>
</tbody>
</table>

The VUIS monadic services are implemented as reusable software (composite) components, which allow different functions to be carried out, depending on the parameters supplied to the component. As VUIS deals with distributed computing, there is a demand for encapsulated code, which can be delivered over a short time frame as near instantaneous as possible. Also, it may be desirable to include/incorporate software that is already on the market into VUIS. For example, there is a student assessment product called "Question Mark" [2], which generates electronic tests, provides on-line assessment of students and contains management functions available to the instructor. It may be desirable to integrate this software into the VUIS.
The design of this component-based VUIS approach providing for such functionality is described further.

2 Component Based Platform Approach

Software components are self-contained, reusable software units [3], [4], [5]. Using visual application builder tools, components can be composed into applets, applications, servlets, and composite components. The VUIS will employ a platform approach [6]. A platform is a set of generally reusable components. It can be viewed as a substrate that allows for installation of components and their frameworks, so that they can be instantiated and activated. The VUIS platform is virtual to some degree. It consists of a number of components, each with clearly defined interfaces. Information models will be used to describe the semantics of data exchanged over the component interfaces. Product builders (e.g. instructors composing a course/module) can choose which components to produce a product (a course/module). For example, a Create component will create an eLecture if that is the basic service type parameter passed to it. Similarly, if the basic service type parameter specifies an eTutorial, then an instance of that will be produced. Other parameters can be added, which may specify what facilities the basic service has (e.g. whether or not to include an interactive example in the eTutorial). Figure 1 shows the Create component and the methods (presented user options) it can carry out (it can have also other methods, which may be specific to certain basic services and could only be invoked when that basic service is being created). For instance, for a new lecture the method uploadFile() could be invoked. This installs a pre-prepared lecture into the appropriate VUIS course. The question of the presentation software of this lecture (e.g. MS Word or MS PowerPoint) should be resolved by the VUIS either by embedded application or Java applet transformation.

Two types of component are used in the VUIS. The service component (e.g. Create) contains functionality that can be reused in applications (i.e. for creation of any new basic service object). The application component (e.g. Course Settings) contains functionality realised by using the service components’ functionality. Application components are integrated to form products.

To allow self-contained components to communicate a set of logically related interfaces is implemented. The same interface may be used by several components [7]. In Figure 2, the interface CreateBasicService is shown that allows the Course Settings component (which manages the instructors’ module) to create a basic service as required by the instructor.

![Fig. 2. Component interaction over an interface](image)

A component has two types of interface: i) a provided interface (the Create component’s interface), whose functionality must be implemented by the component; ii) a required interface (the Course Settings component’s interface) whose functionality is implemented outside of the component. Information models define the structure and semantics of data objects required by components thus allowing them to request a particular service with one method call, using these specially structured data objects. This approach is more efficient than using a number of methods, which have their own parameters and require the component to specify these attributes. Information models can be extended for use in a particular situation. Any component that uses this extended information model may or may not use the additional elements, depending on what the component is requesting. For example, if an eLecture is being created by the Create component, it may have access to extended areas of the data object that would not be relevant to, say, an eLab.

In the VUIS, an information model for communication between the Course Settings and the Create components would have the following data structure (all of the following attributes are required in the set-up of a basic service):

- `string basicServiceType`
- `boolean retrieveBasicServiceFromRepository`
- `string specifyBasicService`
- `boolean addBasicServiceToRepository`
- `String[] learningComponentsRequired`
- `boolean uploadFile, string filename`

The concepts discussed above form a platform from which products can be built (e.g. components to build a basic service, like an eLecture, can be chosen from the platform and combined to produce that service). Once a component is constructed, it can be added to the platform.
Three main technologies could be used to implement the components monadic services in the VUIS. These included Enterprise JavaBeans (EJB) [8], the Common Object Request Broker Architecture (CORBA) [9] and Microsoft’s Component Object Model (COM) [10]. Ultimately it was decided to utilise the EJB structure because it is simple to use, portable and reusable. Also, Java operates strongest of the three on general Web clients and can use ActiveX on Windows web clients. COM’s implementations and applications are for desktops. CORBA is strong when working with servers but Java works well here also. Java is also found to work best for infrastructure vendors.

3 VUIS EJB application example

EJB is a state-of-the-art architecture for developing, deploying and managing reliable enterprise applications in the production environment. Beans are components that are used as parts of distributed enterprise logic. Each bean encapsulates a part of the business logic of the system. In the VUIS, beans are in communication with web applications (using Java Server Page - JSP), servlets, other beans and of course system users. The type of applications they will be in contact with will include resource managers and databases management systems. Any such entities that can invoke bean functionality are referred to from here as “clients”. Many of these clients can access one or more beans at any one time [8]. However, beans are not a complete operational application. In order to deploy a bean as a runnable application, it must have an EJB container associated with it. This container provides the run-time environment for the bean. Containers with communicating beans can be located on different servers and more than one bean can be placed in each container. It also provides a number of other services such as concurrency, security, transaction and instance life cycle management thereby eliminating these concerns of the VUIS developers. When deploying the beans additional classes called container artefacts are generated. These classes “bind” the beans and the container run-time. They deal with issues described above and generate classes, which implement the bean “home” and “remote” interfaces. So that in total the complete application consists of beans, the container and the container artefacts. The VUIS will use the Remote Method Invocation – Internet Inter ORB Protocol (RMI-IIOP) as the distributed object protocol since many container implementations use this. Two RMI-IIOP object types are created, each containing multiple Java classes. In our system, CreateRMI and CreateHomeRMI are two such object types. These implement the client-bean communication.

In our example, the CreateHomeRMI object type provides the implementation of the enterprise bean home interface, CreateHome. The instance of CreateHome is shared by all objects. The CreateRMI object type provides the implementation of the enterprise bean remote interface, Create. A client can communicate with objects of these types. It never communicates with an instance of the enterprise bean class (Figure 3). These objects can inject or add container services when delegating invoked methods to the bean instances.

3.1 VUIS implementing EJB architecture

In the construction of the VUIS, it is important that the developers do not have to concern themselves with system-level problems. That is why the EJB architecture is used.

Firstly Java classes, called enterprise bean classes, will be used. In the production of monadic service components, which can be reused for a number of functions depending on the parameters passed to them, bean classes will be used. An example of such a class would be the Create monadic service used to produce an eLecture. This bean class accepts parameters, one of which specifies that an eLecture is the basic service to be created. The bean component uses these parameters to build an eLecture.

Once a component bean has been produced, some means of communication between components must be implemented. Communication between beans on the VUIS system is achieved by using a bean’s client-view Application Program Interface (API). Web
application and servlets can access this API. The API consists of the enterprise beans “home interface” and “remote interface”. The home and remote objects are distributed objects that implement communication between a remote client and the bean deployed in the container. They also communicate internally with the container at run-time to inject the services mentioned earlier into the client method invocation path.

The life cycle of bean objects are controlled by the home interface. It defines methods such as create(), find() and remove(). In Figure 4, the methods of Create’s home interface are displayed. The create methods must be named “create” but several can be introduced each with a different parameter list. The find methods can have different names. All these methods return remote interface beans, although the find methods can return a collection of them. The EJBHome Interface defines methods that are supported by all bean home interfaces (e.g. remove).

In the implementation of the creation of a basic service on the VUIS, the class CreateBean is required to implement the life cycle methods and the methods to be used by a client. (Container callback methods are also defined here). These methods are used by a client but are invoked by the Create and CreateHome interfaces (Figure 6).

The remote interface defines the methods that a client can use from each individual bean. Its methods reflect the needs of the individual beans in the system. The interface extends the EJBObject interface, which defines methods that are supported by all bean remote interfaces (Figure 5).

The VUIS requires a means of configuring the environment entries of a bean, which can be set so that a bean’s operational environment can be customised. The “deployment descriptor” is used to do this. It is an XML document that contains declarative information about a bean. It contains information that is used by the customer. When the customer receives the beans in their JAR (archive) file, it may be required to change certain settings (e.g. database set-up or security model used). Figure 7 shows which details are editable in the Create enterprise bean.

3.2 Sessions in the VUIS

Because VUIS is web-based system involving client-server communication/web application sessions, session beans were chosen to build VUIS components.
<table>
<thead>
<tr>
<th>Deployment Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name = CreateEJB</td>
</tr>
<tr>
<td>Class = CreateBean</td>
</tr>
<tr>
<td>Home = CreateHome</td>
</tr>
<tr>
<td>Remote = Create</td>
</tr>
<tr>
<td>Type = Session</td>
</tr>
<tr>
<td>Transaction = Required</td>
</tr>
</tbody>
</table>

This is used to implement the system’s conversational business process. A session lasts for the duration of the clients’ session. The session stores the current client’s information (i.e. what access level is allowed to each type of user). The scope of a session is called a conversation.

Here we discuss how session beans are used when an instructor wishes to create an eLecture. The session drives the conversation between the web application – the JSP web page looking for the service required by the user – and the user (Figure 8). The web application is a collection of JSPs and servlets deployed in a web container. The user selects Create eLecture from the list of options in order to create an eLecture basic service. Using the HTTP protocol, the web application processes these requests. It is responsible for generating HTML pages with responses that return to the browser.

The response in this case would be display the creation options for an eLecture. To do this, the web application invokes the Create bean. This bean processes the user data sent to it and generates data for the JSP to format into a HTML page response.

The web application architecture (i.e. JSP) defines the concept of a HTTP session. Such a session spans many HTTP requests from the user. The web application can retain session-specific state across multiple requests from the same user.

A similar session bean class that is bound to a client user is called a stateful session bean. Every client that logs-on has a session bean class object assigned to them. Client method invocations are delegated to the same stateful session bean instance. Thus, client-specific information can be retained on a server, which the system can use. This is useful because otherwise the required data, which is used many times in a particular access (i.e. the student’s ID number), would have to be continuously fetched from a database. This would be quite inefficient. Stateful session beans keep the information that is initially read from databases about the user and information that s/he enters. Such structured information can be called the conversational state. Also included are all the objects that a user can reach.

Stateless session objects do not retain any client-specific information. The container maintains a pool of instances of the session bean class and it delegates a client method invocation to any available instance. The VUIS uses such session beans to perform procedural services on a database. A Query component will exist to perform some type of find function on a specified database. An instructor may wish to view students’ exam records and can use this stateless session bean to do so.

### 3.3 Create EJB in Operation

Figure 9 illustrates how the Create Basic Service is invoked and what databases it communicates with. The CreateEJB session bean implements the rules of the VUIS when a basic service needs to be created. It must account for all consequences of creating a new basic service (i.e. where it is stored, where it appears in the VUIS, how it is ordered etc.). The process displayed in Figure 9 consists of a conversation between a user and the server. As a result, and as has already been stated, CreateEJB is implemented as a stateful session.

The CreateWeb web application is written in JSP. This creates the HTML pages (that the user accesses) and formats the data input and interactions of the user. This application is deployed in a web container on a web server and is maintained by the system administrator.

The Create bean accesses databases in order to retrieve and submit information. In this case, there are two databases, which the bean interacts with – **Basic Services Database** and **User Database**. The Basic Services Database is a repository where basic services of all types are stored. If a lecturer wants to add these to a course, a link is created to the application in the
repository. The basic services are stored and divided into their own categories. The User Database is used to specify which users will have access to the basic service. Because this is a sensitive operation (i.e., alteration of the VUIS databases), access on the Create bean is limited to the instructor and any tutors he grants such access to.

When a user enters the VUIS URL, a HTML page with JSP appears asking the user to log on (by submitting his/her username and password). The authentication logic is implemented by the web container. Once verified, the user has access to all the services available to one with that access level. An instructor can use the CreateWeb web application to invoke the create() method on the CreateHome interface. This creates a new Create bean session (described above) and an instance of the CreateBean class. This CreateBean object sets up an initial state by reading information from the Basic Services database.

At this stage, the user will have indicated that a new basic service is being added to an already specified course. Firstly, the user specifies the type of basic service required. This is achieved when the CreateWeb application invokes the createBasicService() method on the Create bean. Let us say the user wants to create a new eLecture. The invocation informs the Create bean of this. Now, the user has the option of choosing an existing eLecture from the repository or creating one from scratch. To choose an existing eLecture, CreateWeb invokes the Create bean’s getFromRepository() method. Otherwise, CreateWeb displays the new eLecture option to the user. The user enters the details of the eLecture in the form produced. On this page the user can elect to add an additional file to the eLecture. If chosen, then CreateWeb invokes the Create bean’s uploadFile() method. The user will be allowed to specify the file and this will be added to the repository with the eLecture itself. Finally, the user can decide that this eLecture should be shareable with other instructors. If so, CreateWeb invokes the Create bean’s addToRepository() method and it prompts the CreateWeb web application to display a HTML page with options on who to share with and what access to allow them. The eLecture is then submitted to the repository and becomes a part of the instructors’ course/module.

4 Conclusion
This paper has proposed an architectural design for a VUIS platform, particularly focussing on showing how EJB and JSP technology may be used beneficially for this task. It introduces and describes how the functionality and requirements of each VUIS basic service may be decomposed in a set of monadic services. These in turn are implemented as a reusable software (composite) components – EJB beans – which communicate over well defined interfaces with other components on the VUIS platform in order to provide the basic service required by the user. It is shown how this design approach facilitates the addressing of important attributes of VUIS, such as scalability, security, latency, and user interface design issues such as acceptable response time and speed.

References:
[5] Component Primer: 