Xi'an Jiaotong-Liverpool University
西交利物浦大學

Department of Computer Science and Engineering

Assignment 2

Component Models

Sahand Sabour

CSE306

Software Engineering II

Student ID

1614650

Introduction

Software reuse has become a pivotal topic in modern software engineering, mainly due to its impact on the development cost and process risk. Evidently, as object-oriented development has failed to effectively support software reuse, Component-based Software Engineering (CBSE) has emerged as a well-known approach for professional software architecture and it can be defined as a branch of software engineering that focuses on the development of standardized components with a reuse-based approach. Essentially, CBSE consists of independent components, component models and middleware that provide support for component distribution, security, resource allocation, etc.

Components are loosely coupled deployable unit compositions that have a contractually specified interface and dependencies that implement or describe services of real world entities. Hence, components could be distinguished from objects as they are service-oriented and can be employed independently while objects are identity-oriented and must be bind to an application. A component model is a definition of standards for implementation and deployment (i.e. interfaces) of components as well as their documentation (Figure 1). The standards of component models must be met by all the components to ensure that they are able to operate effectively with each other. In this report, Enterprise Java Bean (EJB), which is a highly prominent component model, is thoroughly analyzed and discussed. Moreover, EJB's implementation of the component model is investigated.

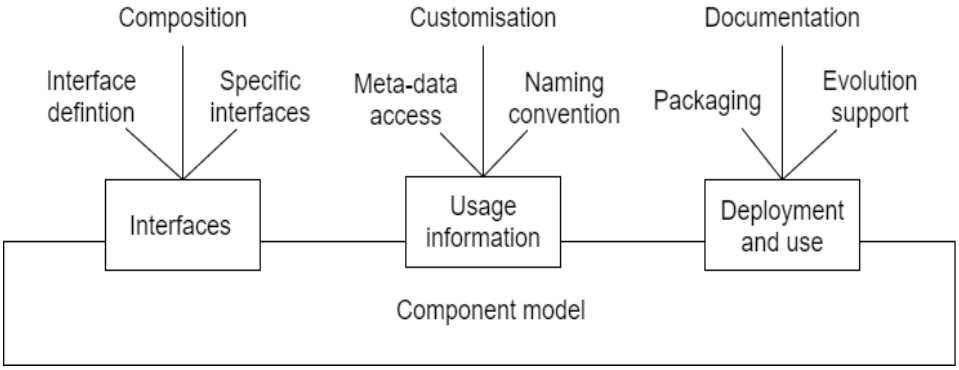

Figure 1: Component model

Overview

According to Oracle's official documentation, Enterprise Java Bean (EJB) is a server-side component architecture that allows development of distributed and secure applications with the Java Enterprise Edition (Java EE) [1]. EJBs are commonly implemented in scalable application that need encapsulated businesses logic or remote access. EJBs encapsulate, execute or summarize specific business logic of distributed applications, where business logic refers to the code that demonstrates the application purpose. In EJB architecture, components run inside a container and for each EJB component, there exists an obligatory contract between the component and the container. By adhering to the contract, EJB components would be developed based on a clearly-defined structure and consequently, are enabled to utilize the services provided by the container based on a set of standards. These standards also ensure that the execution infrastructures are enabled to support component operations.

As previously mentioned, the obligation for each EJB component to follow a specific structure is to ensure interoperation between different components. For instance, when purchasing a monitor, it could be noticed that the monitors from different brands share the same input slots, such as HDMI or VGA, as they should be able to connect to other devices regardless of their brands. In this case, the manufacturers must adhere to specific industry standards to produce consistent interfaces. The following figure illustrates a simplified version of the EJB component model (Figure 2).

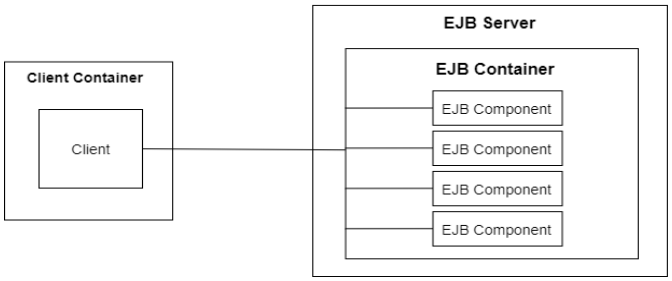

Figure 2: Simplified EJB component model

In the above illustration, the client would be regarded as an application that is executed on a local or remote system and uses the functionality of EJB components. EJB components that call methods and services from other EJB containers are considered as clients likewise.

The EJB ecosystem

As mentioned, the EJB deployment and development requires an EJB container and EJB components. In addition, there are six assets that are included in the EJB ecosystem [2]:

1. Bean provider: provides the business components to enterprise applications.

2. Application assembler: integrates and combines components to build the target application.

3. EJB deployer: conducts the required configurations, such as resource allocation and security parameter setting. Accordingly, it deploys the built application on the container.

4. System administrator: provides the monitoring and management tools for the deployed application.

5. Container provider: used as the runtime environment for the beans. It also provides the middleware services to the beans and is responsible for the management of these services likewise.

6. Tool vendors: consist of numerous IDEs, such as Eclipse and NetBeans, which allow developers to rapidly and simply develop and debug the written components.

The summarized ecosystem of EJB is provided in the above figure (Figure 3).

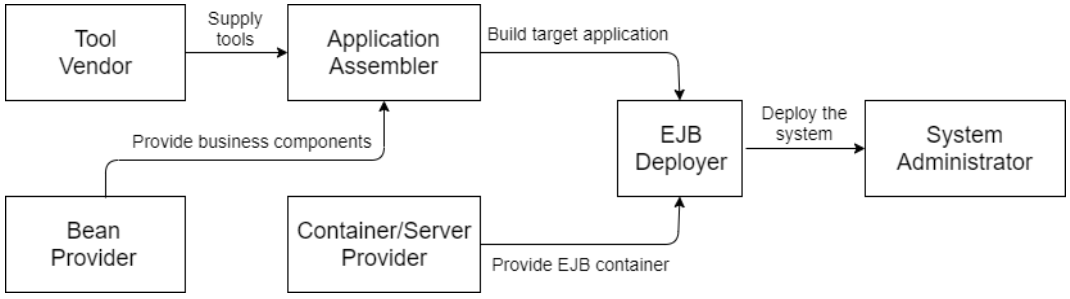

Figure 3: EJB ecosystem

Types of EJB

There are three types of Enterprise Java Beans:

1. **Session beans**

   Session beans encapsulate actions and business logic that can be invoked and executed by the client. Session beans can be Stateful or Stateless. Stateful session beans perform business tasks while keeping their state by storing the state in an instance variable. For example, when using an ATM machine, the customer may want to carry out a number of tasks. Since the user for all these task is

the same person, the ATM machine must store this information and keep track of this state for each task. On the contrary, stateless beans are implemented for requests that include only one method invocation. That is, it implements the business logic without having to store any information. For instance, when validating student IDs, the request would include the student ID and the invoked method verifies the correctness of the given ID. After the verification, the ID will no longer be used and therefore, it is not necessary to store this piece of information.

2. **Message-driven beans**

   Similar to session beans, message-driven beans encapsulate actions and business logic. However, message-driven beans act as listeners for particular messaging type, such as Java Message Service API. Another distinction between the two would be that message-driven beans process the incoming messages in an asynchronous manner while session beans implement synchronous processing. Moreover, similar to stateless beans, message-driven beans do not retain any data or information as a form of state for specific users.

3. **Entity beans**

   Entity beans are concerned with data and mainly model business concepts of real-world objects. For instance, entity beans can be used to demonstrate payment records, a scheduled flight, etc. In simple Java programs, after the program is terminated, all of the created objects and instances are lost and cannot be retrieved. By implementing entity beans, the program could store these objects (entities) and reuse them after being restarted. Moreover, any program on the network can find and use entity beans by using Java Naming Directory Interface (JNDI). However, entity beans are deprecated in the new versions of EJB and have been replaced by Java Persistence API entities [3].

Component reuse in EJB

In software engineering, software processes that support CBSE are referred to as CBSE processes. There are two types of CBSE processes: CBSE for reuse and CBSE with reuse. The prior type focuses on developing components or services that could be reused by other applications while the latter is concerned with developing applications that reuse existing components or services. In order for a component to be reusable, it should reflect stable domain abstractions, hide its state, be independent, and demonstrate its exceptions in its interface rather than catching these exceptions.

The aim of EJBs is to produce reusable components that can be assembled or reassembled in different applications. As illustrated in the below figure (Figure 3), EJB components can be considered as LEGO building block; they are produced by a set of specific standards to ensure that all the pieces fit together. Accordingly, different pieces could be assembled together in different ways to build various constructions.

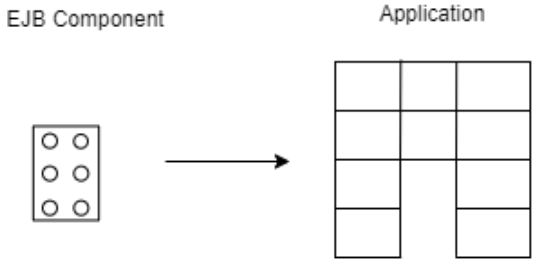

Figure 3: EJB LEGO analogy

In EJB, the components are developed based on a specific standard provided by the EJB container to ensure that the components can interoperate. The EJB container is responsible for creating the enterprise bean and binding it to a name service so that it can be used by other applications. The container is also responsible for saving and caching the bean's state to enable the components to hide state representations. In addition, the container provides the exceptions through its components' interface. Accordingly, as previously mentioned, the components in EJB are independent. Hence, the developed EJB components are reusable

and can be implemented in different applications. This represents CBSE for reuse. Accordingly, these components could be used in other remote or local client applications as well as other beans. The reuse of existing EJB components in new applications demonstrates CBSE with reuse. Therefore, both of the CBSE processes are included in the EJB architecture.

Summary

Due to the failure of object oriented development to efficiently reuse software, Component-based Software Engineering (CBSE) has become fairly well-known. CSBE includes independent components, component models and middleware. Components are reusable building blocks of software and component models are definitions of component architecture. One of the important component models of CBSE is Enterprise Java Bean (EJB). In EJB, components are executed within EJB containers. Containers provide the standard for component development to ensure that different components can interoperate and function properly in different applications. The EJB ecosystem consists of six parties: the bean provider, the application assembler, the EJB deployer, the system administrator, the container providers, and the tool vendors. The standards of EJB containers allows for the development of reusable independent components that could be used by other beans and applications. In summary, EJB is a beneficial component model for scalable and secure apps that require rapid development. However, the complications of the EJB implementation and its large specifications could be considered as the major drawbacks of this component model.

Conclusions

In this report, the Enterprise Java Bean (EJB) component model was thoroughly investigated and discussed. Moreover, the related terminology was explained in detail. A number of examples to further enhance the explanations and definitions were provided likewise. It is believed that by doing the required research, the students were enabled to improve their critical thinking as well as resource finding skills. In conclusion, this assignment was considered as a valuable learning experience regarding components, component models, and EJBs in particular, which greatly assisted the students with further understanding of the lecture notes and materials.

References

[1] *Enterprise JavaBeans Technology* [Online]. Available: https://www.oracle.com/technetwork/java/ejb-141389.html

[2] *What are EJBs?* [Online]. Available: https://tutorialseye.com/what-are-ejbs.html

[3] *What is an Enterprise Bean?* [Online]. Available: https://docs.oracle.com/javaee/5/tutorial/doc/bnblt.html