2. Foundations

Contents

2.1. Enterprise JavaBeans

The Enterprise JavaBeans (EJB) technology is a standardized framework for the development of multi-layered software systems. It encapsulates business logic and provides services like security handling, transaction management, persistence, networking, resource management or concurrency handling [20]. The EJB architecture is based on server-side components for application development, which are called enterprise beans. Enterprise Beans differ in their functionality from JavaBeans. In contrast to JavaBeans, enterprise beans are components, which manage objects for distributed applications, that are located on various machines with many clients [27].

The EJB technology implements the so called “three tier architecture”, where the business logic with its services and the client applications are separated. The three tier concept guarantees performance, scalability and flexibility of systems during the access of the back-end resources like databases, if the amount of clients is big [27].

The EJB framework is a Java Enterprise Edition (Java EE) runtime environment, where the EJB container runs. The EJB container is a software, which consists of EJB components (enterprise beans). It manages all interactions of the EJB components during their runtime as well as their lifecycle. This takeover of the control of business logic managed by the EJB container is called Inversion of Control and was defined by Rod Johnson [21], [5]. Due to the approach of Inversion of Control, the EJB application developer does not need to care about transaction and security management. As a consequence, creation and maintenance of EJB applications becomes less complex. The EJB programming API includes a set of protocols (agreements) between the EJB container and the EJB components, interfaces and classes in order to enable platform

independency, portability and a large number of functions for the development of business applications.

As mentioned above, the component architecture forms the basis of the EJB technology. Each EJB container includes a pool of EJB instances and is responsible for their monitoring, performing as well as for the lifecycle of EJB instances in compliance with contract rules established in the EJB protocols. The lifecycle management process includes the creation and destruction of the EJB instances [15], [27]. There are three types of the EJB components:

- **A Persistent Entity** represents a table in a database, where an entity instance is a row in this table. Persistent Entities are the real domain objects, which are represented by Plain Old Java Objects (POJOs). The persistence approach of EJB 3 defines a reworked Java persistence API (JPA), whose implementation is realized by the persistence provider. The persistence provider enables the availability of the entity manager. The persistent entities can be considered as *"a high-performance data cache"* [18], where the entity manager arranges for the loading, saving and updating of the persistent entities. Additionally, the Entity Manager is responsible for the search of persistent entities, which can be performed by using the Java Persistence Query Language.

- **A Session Bean** is in contrast to the entity bean a non-persistent element of the EJB components and is used to map business processes and the interactions between business objects. Session beans are divided into stateful session beans and stateless session beans. Stateful session beans hold the information state of each client. They are used to perform tasks where many interaction steps (methods) for one client are needed. Thus, the corresponding session object has to store the state of the one client during many method calls. The stateless beans are connected to the client for single method calls. In this case each method invocation is independent of the stored resources of the previous method. Therefore no client state information can be stored in a stateless session object [18], [2].

- **Message-Driven Beans** are used for asynchron communication between the client and the server by using the Java Messaging Service, where the client is the message sender and the server is the message receiver. Message-driven beans are not called by the client directly. The client sends its request to the messaging service. Message-driven beans do not have an individual state or unique identifier of the client and have a similar lifecycle like stateless session beans [29].

Figure 2.1 illustrates the EJB architecture, where the client and the EJB container running on the server side are separated. A single EJB server can include many EJB containers with various types of EJBs. The remote client represents either a servlet, a mobile device, a stand-alone Java application or another enterprise bean [29]. As shown in 2.1, the client does not communicate to the bean

of the EJB container directly. Instead of that it calls the EJB object through the business interface. The EJB object then invokes either an enterprise bean or a service. The invocation of an enterprise bean is also realized by using an interface. This concept of the EJB architecture ensures automatic persistence, transaction and security for the bean.

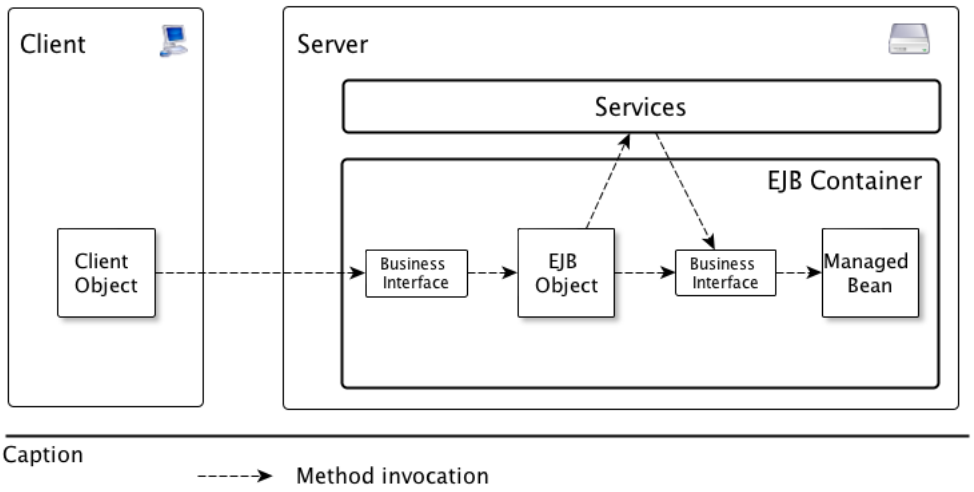

Figure 2.1.: EJB Architecture.

Another task of the EJB container is to provide the required resources for EJB instances. The communication between bean and EJB container can be performed either via callback methods, the EJBContext or the Java Naming and Directory Interface (JNDI). Callback methods are implemented by each bean and used to notify the bean, which took part in the particular lifecycle event. The disadvantage of this mechanism is, that the beans are forced to implement all callback methods even if only one of them is used.

An EJB container provides the service of resource management via the Enterprise Naming Context (ENC). The ENC registers a references of the resources and allocates a unique Name for each registered resource. The access to this names can be gained by using the JNDI-lookup mechanism [18]. Using the EJBContext object the bean can obtain the information of its environment or transaction status [27].

2.2. JavaServer Faces

The expansion of the World Wide Web increases the demands for modern web development tools. As a consequence, in the last ten years many web application frameworks have been produced. One of the first attempts to develop Java-based technology for Web applications were the Servlet API and the later emerged