Service Activator

Context

Enterprise beans and other business services need a way to be activated asynchronously.

Problem

When a client needs to access an enterprise bean, it first looks up the bean's home object. The client requests the EJB home to provide a remote reference to the required enterprise bean. The client then invokes business method calls on the remote reference to access the enterprise bean services. All these method calls, such as lookup and remote method calls, are synchronous. The client has to wait until these methods return.

Another factor to consider is the life cycle of an enterprise bean. The EJB specification permits the container to passivate an enterprise bean to secondary storage. As a result, the EJB container has no mechanism by which it can provide a process-like service to keep an enterprise bean constantly in an activated and ready state. Because the client must interact with the enterprise bean using the bean's remote interface, even if the bean is in an activated state in the container, the client still needs to obtain its remote interface via the lookup process and still interacts with the bean in a synchronous manner.

If an application needs synchronous processing for its server-side business components, then enterprise beans are an appropriate choice. Some application clients may require asynchronous processing for the server-side business objects because the clients do not need to wait or do not have the time to wait for the processing to complete. In cases where the application needs a form of asynchronous processing, enterprise beans do not offer this capability in implementations prior to EJB 2.0.

EJB 2.0 provides integration by introducing message-driven bean, which is a special type of stateless session bean that offers asynchronous invocation capabilities. However, the new specification does not offer asynchronous invocation for other types of enterprise beans, such as stateful or entity beans.

In general, a business service such as a session or entity bean provides only synchronous processing and thus presents a challenge to implementing asynchronous processing.

Forces

- Enterprise beans are exposed to their clients via their remote interfaces, which allow only synchronous access.
- The container manages enterprise beans, allowing interactions only via the remote references. The EJB container does not allow direct access to the bean implementation and its methods. Thus, implementing the JMS message listener in an enterprise bean is not feasible, since this violates the EJB specification by permitting direct access to the bean implementation.
- An application needs to provide a publish/subscribe or point-to-point messaging framework where clients can publish requests to enterprise beans for asynchronous processing.
- Clients need asynchronous processing capabilities from the enterprise beans and other business components that can only provide synchronous access, so that the client can send a request for processing without waiting for the results.
- Clients want to use the message-oriented middleware (MOM) interfaces offered by the Java Messaging Service (JMS). These interfaces are not integrated into EJB server products that are based on the pre-EJB 2.0 specification.

- An application needs to provide daemon-like service so that an enterprise bean can be in a quiet mode until an event (or a message) triggers its activity.
- Enterprise beans are subject to the container life cycle management, which includes passivation due to time-outs, inactivity and resource management. The client will have to invoke on an enterprise bean to activate it again.
- EJB 2.0 introduces a message-driven bean as a stateless session bean, but it is not possible to invoke other types of enterprise beans asynchronously.

Solution

Use a Service Activator to receive asynchronous client requests and messages. On receiving a message, the Service Activator locates and invokes the necessary business methods on the business service components to fulfill the request asynchronously.

The ServiceActivator is a JMS Listener and delegation service that requires implementing the JMS message listener—making it a JMS listener object that can listen to JMS messages. The ServiceActivator can be implemented as a standalone service. Clients act as the message generator, generating events based on their activity.

Any client that needs to asynchronously invoke a business service, such as an enterprise bean, may create and send a message to the Service Activator. The Service Activator receives the message and parses it to interpret the client request. Once the client's request is parsed or unmarshalled, the Service Activator identifies and locates the necessary business service component and invokes business methods to complete processing of the client's request asynchronously.

The Service Activator may optionally send an acknowledgement to the client after successfully completing the request processing. The Service Activator may also notify the client or other services on failure events if it fails to complete the asynchronous request processing.

The Service Activator may use the services of a Service Locator to locate a business component. See "Service Locator" on page 351.

Structure

Figure 1.1 represents the class relationships for the Service Activator pattern.

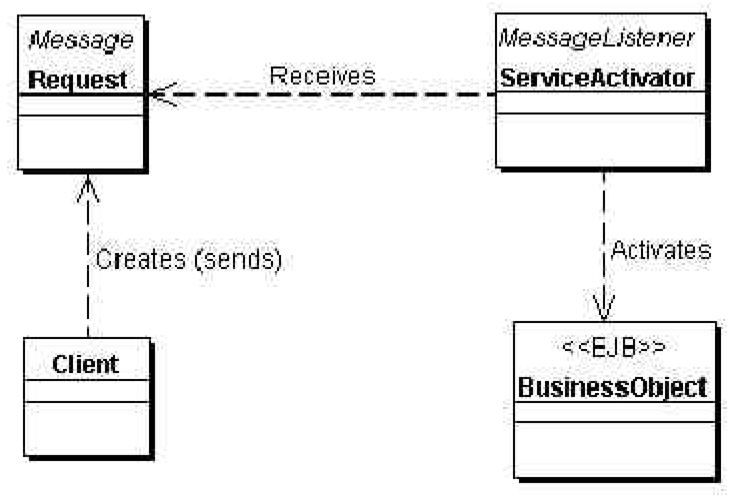

Figure 1.1 Service Activator class diagram.

Participants and Responsibilities

Figure 1.2 shows the interactions between the various participants in the Service Activator pattern.

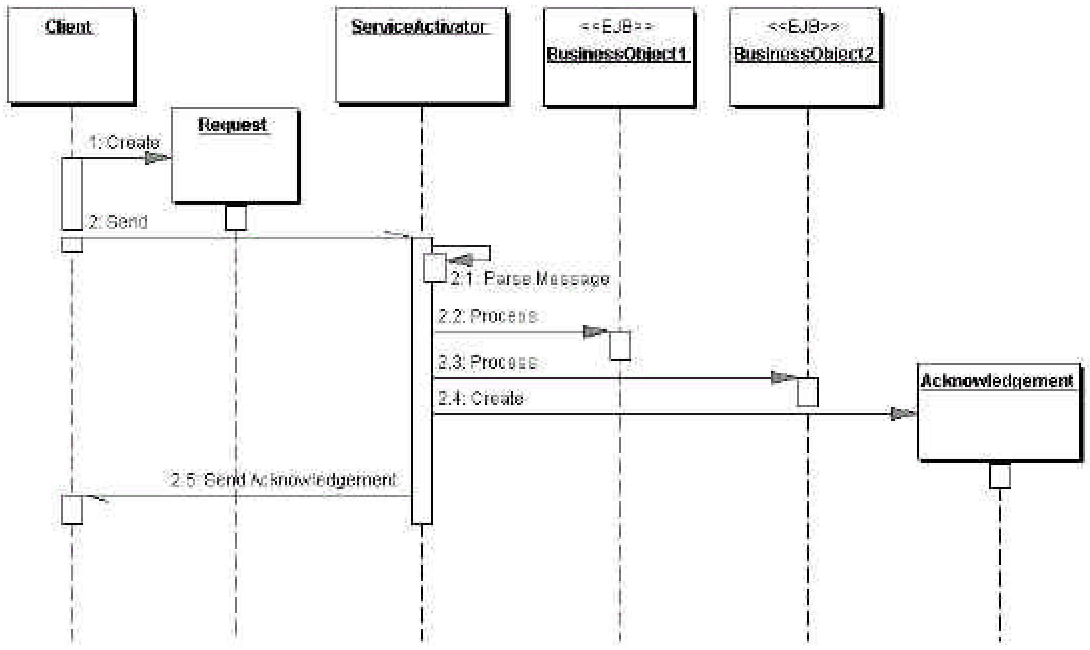

Figure 1.2 Service Activator sequence diagram.

Client

The client requires an asynchronous processing facility from the business objects participating in a workflow. The client can be any type of application that has the capability to create and send JMS messages. The client can also be an EJB component that needs to invoke another EJB component's business methods in an asynchronous manner. The client can use the services offered by the Service Locator pattern to look up or create EJB components, JMS services, and JMS objects, as necessary.

Request

The Request is the message object created by the client and sent to the ServiceActivator via the MOM. According to the JMS specification, the Request is an object that implements the javax.jms.Message interface. The JMS API provides several message types, such as TextMessage, ObjectMessage, and so forth, that can be used as request objects.

ServiceActivator

The ServiceActivator is the main class of the pattern. It implements the javax.jms.MessageListener interface, which is defined by the JMS specification. The ServiceActivator implements an `onMessage()` method that is invoked when a new message arrives. The ServiceActivator parses (unmarshals) the message (request) to determine what needs to be done. The ServiceActivator may use the services offered by a Service Locator (see Service Locator) pattern to look up or create Business Service components such as enterprise beans.

BusinessObject

BusinessObject is the target object to which the client needs access in an asynchronous mode. The business object is a role fulfilled by either a session or entity bean. It is also possible that the BusinessObject is an external service instead of an entity bean.