Business Delegate

Context

A multitiered, distributed system requires remote method invocations to send and receive data across tiers. Clients are exposed to the complexity of dealing with distributed components.

Problem

Presentation-tier components interact directly with business services. This direct interaction exposes the underlying implementation details of the business service application program interface (API) to the presentation tier. As a result, the presentation-tier components are vulnerable to changes in the implementation of the business services: When the implementation of the business services change, the exposed implementation code in the presentation tier must change too.

Additionally, there may be a detrimental impact on network performance because presentation-tier components that use the business service API make too many invocations over the network. This happens when presentation-tier components use the underlying API directly, with no client-side caching mechanism or aggregating service.

Lastly, exposing the service APIs directly to the client forces the client to deal with the networking issues associated with the distributed nature of EJB technology.

Forces

- Presentation-tier clients need access to business services.
- Different clients, such as devices, Web clients, and thick clients, need access to business service.
- Business services APIs may change as business requirements evolve.
- It is desirable to minimize coupling between presentation-tier clients and the business service, thus hiding the underlying implementation details of the service, such as lookup and access.
- Clients may need to implement caching mechanisms for business service information.
- It is desirable to reduce network traffic between client and business services.

Solution

Use a Business Delegate to reduce coupling between presentation-tier clients and business services. The Business Delegate hides the underlying implementation details of the business service, such as lookup and access details of the EJB architecture.

The Business Delegate acts as a client-side business abstraction; it provides an abstraction for, and thus hides, the implementation of the business services. Using a Business Delegate reduces the coupling between presentation-tier clients and the system's business services. Depending on the implementation strategy, the Business Delegate may shield clients from possible volatility in the implementation of the business service API. Potentially, this reduces the number of changes that must be made to the presentation-tier client code when the business service API or its underlying implementation changes.

However, interface methods in the Business Delegate may still require modification if the underlying business service API changes. Admittedly, though, it is more likely that changes will be made to the business service rather than to the Business Delegate.

Often, developers are skeptical when a design goal such as abstracting the business layer causes additional upfront work in return for future gains. However, using this pattern or its strategies results in only a small amount of additional upfront work and provides considerable benefits. The main benefit is hiding the details of the underlying service. For example, the client can become transparent to naming and lookup services. The Business Delegate also handles the exceptions from the business services, such as java.rmi.Remote exceptions, JMS exceptions and so on. The Business Delegate may intercept such service level exceptions and generate application level exceptions instead. Application level exceptions are easier to handle by the clients, and may be user friendly. The Business Delegate may also tranparently perform any retry or recovery operations necessary in the event of a service failure without exposing the client to the problem until it is determined that the problem is not resolvable. These gains present a compelling reason to use the pattern.

Another benefit is that the delegate may cache results and references to remote business services. Caching can significantly improve performance, because it limits unnecessary and potentially costly round trips over the network.

A Business Delegate uses a component called the Lookup Service. The Lookup Service is responsible for hiding the underlying implementation details of the business service lookup code. The Lookup Service may be written as part of the Delegate, but we recommend that it be implemented as a separate component, as outlined in the Service Locator pattern.

When the Business Delegate is used with a Session Facade, typically there is a one-to-one relationship between the two. This one-to-one relationship exists because logic that might have been encapsulated in a Business Delegate relating to its interaction with multiple business services (creating a one-to-many relationship) will often be factored back into a Session Facade.

Finally, it should be noted that this pattern could be used to reduce coupling between other tiers, not simply the presentation and the business tiers.

Structure

Figure 1.1 shows the class diagram representing the Business Delegate pattern. The client requests the BusinessDelegate to provide access to the underlying business service. The BusinessDelegate uses a LookupService to locate the required BusinessService component.

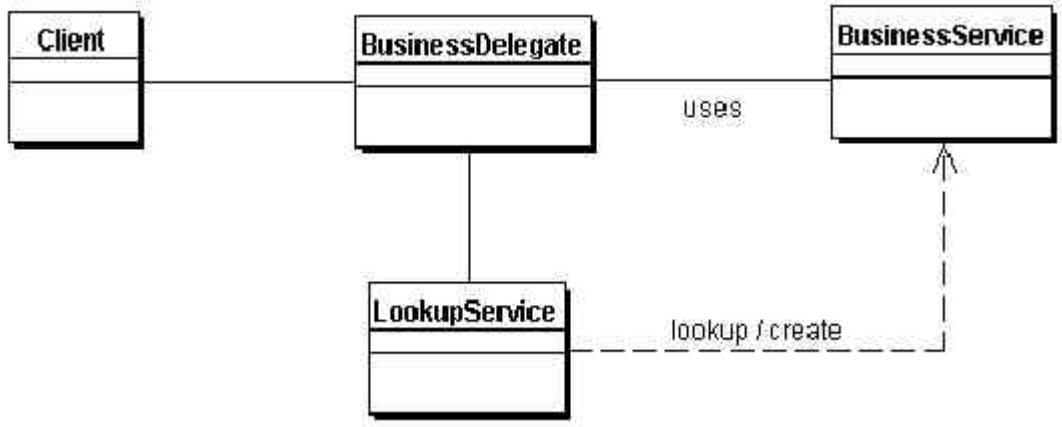

Figure 1.1 BusinessDelegate class diagram.

Participants and Responsibilities

Figure 1.2 and Figure 1.3 show sequence diagrams that illustrate typical interactions for the Business Delegate pattern.

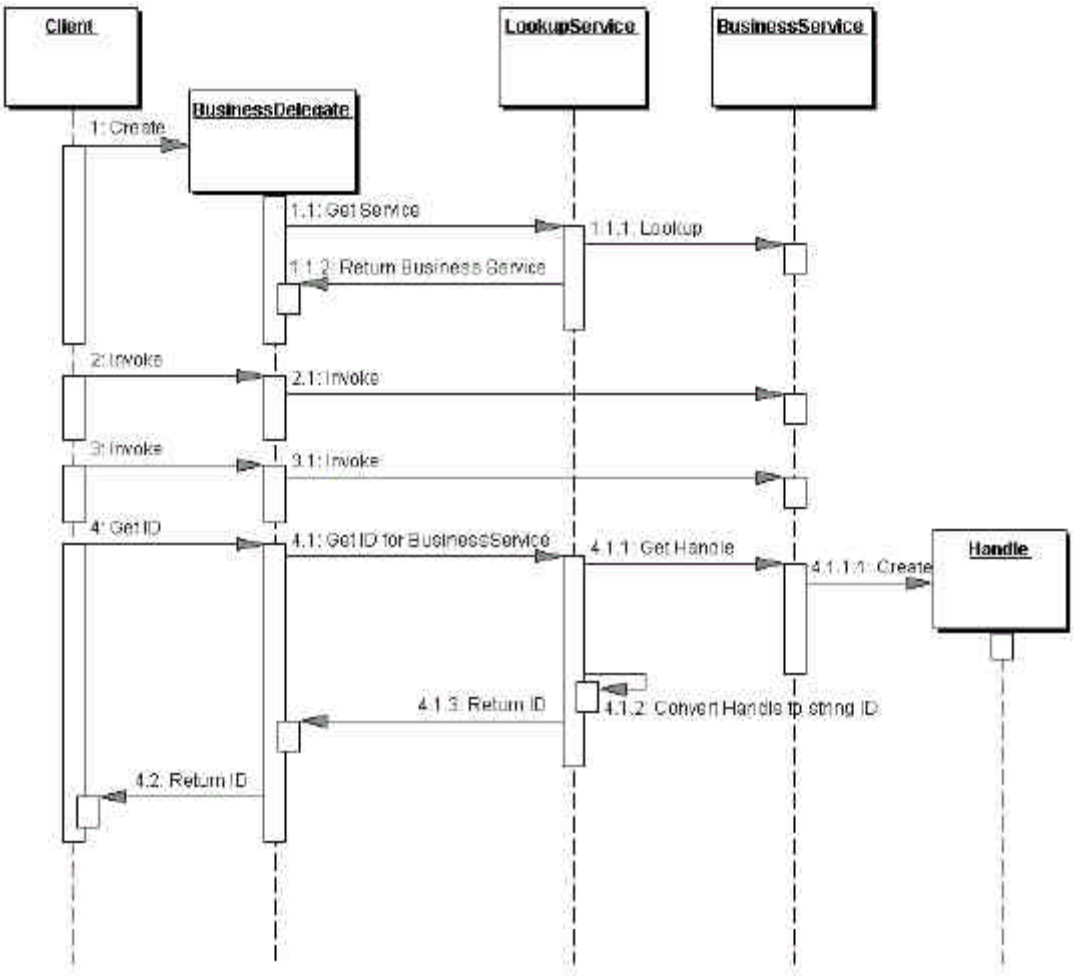

Figure 1.2 BusinessDelegate sequence diagram.

The BusinessDelegate uses a LookupService for locating the business service. The business service is used to invoke the business methods on behalf of the client. The Get ID method shows that the BusinessDelegate can obtain a String version of the handle (such as EJBHandle object) for the business service and return it to the client as a String. The client can use the String version of the handle at a later time to reconnect to the business service it was using when it obtained the handle. This technique will avoid new lookups, since the handle is capable of reconnecting to its business service instance. It should be noted that handle objects are implemented by the container provider and may not be portable across containers from different vendors.

The sequence diagram in Figure 1.3 shows obtaining a BusinessService (such as a session or an entity bean) using its handle.

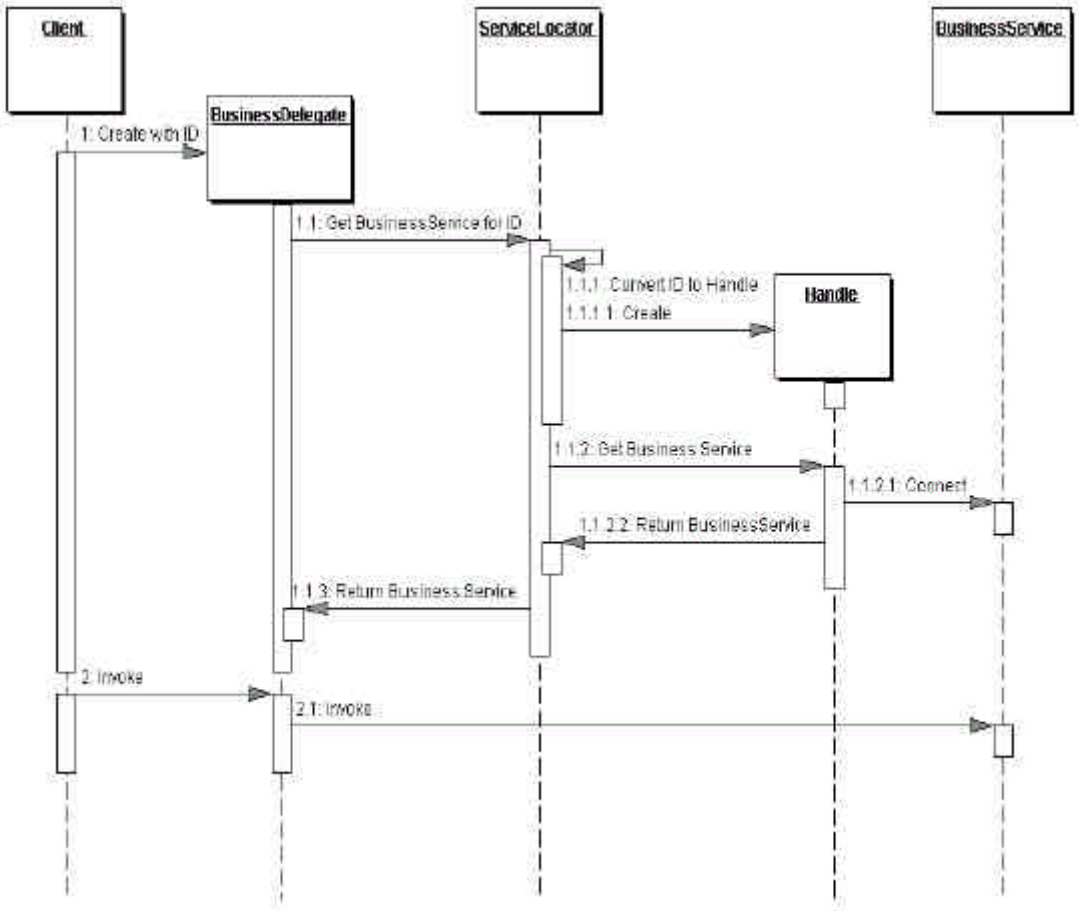

Figure 1.3 BusinessDelegate with ID sequence diagram.

BusinessDelegate

The BusinessDelegate's role is to provide control and protection for the business service. The BusinessDelegate can expose two types of constructors to clients. One type of request instantiates the BusinessDelegate without an ID, while the other instantiates it with an ID, where ID is a String version of the reference to a remote object, such as EJBHome or EJBObject.

When initialized without an ID, the BusinessDelegate requests the service from the Lookup Service, typically implemented as a Service Locator, which returns the Service Factory, such as EJBHome. The BusinessDelegate requests that the Service Factory locate, create, or remove a BusinessService, such as an enterprise bean.

When initialized with an ID string, the BusinessDelegate uses the ID string to reconnect to the BusinessService. Thus, the BusinessDelegate shields the client from the underlying implementation details of BusinessService naming and lookup. Furthermore, the presentation-tier client never directly makes a remote invocation on a BusinessSession; instead, the client uses the BusinessDelegate.

LookupService

The BusinessDelegate uses the LookupService to locate the BusinessService. The LookupService encapsulates the implementation details of BusinessService lookup.

BusinessService

The BusinessService is a business-tier component, such as an enterprise bean or a JMS component, that provides the required service to the client.