CHAPTER 3

Managing Connections

THIS chapter discusses how an application creates and uses connections to an underlying EIS. In particular, it focuses on the need for connection pooling and describes the different scenarios under which connection pooling is accomplished.

To provide some background and context, we begin by discussing the need for connection pooling. Enterprise applications that integrate with EISs run in either a two-tier or a multi-tier application environment. (Note that a two-tier environment is also called a nonmanaged environment, whereas a multi-tier environment is called a managed environment.) Figure 3.1 provides a simplified illustration of these two environments.

In a two-tier application environment, a client accesses an EIS that resides on a server. The client application creates a connection to an EIS. In this case, a resource adapter may provide connection pooling, or the client application may manage the connection itself.

In a multi-tier application environment, Web-based clients or applications use an application server residing on a middle tier to access EISs. The application server manages the connection pooling and provides this service to the applications deployed on the application server.

Applications require connections so that they can communicate to an underlying EIS. They use connections to access enterprise information system resources. A connection can be a database connection, a Java Message Service (JMS) connection, a SAP R/3 connection, and so forth. From an application's perspective, an application obtains a connection, uses it to access an EIS resource, then closes the connection. The application uses a connection factory to obtain a connection. Once it has obtained the connection, the application uses the connection to connect to the underlying EIS. When the application completes its work with the EIS, it closes the connection.

Why is there a need for connection pooling? Connection pooling is a way of managing connections. Because connections are expensive to create and destroy, it is imperative that they be pooled and managed properly. Proper connection pooling leads to better scalability and performance for enterprise applications.

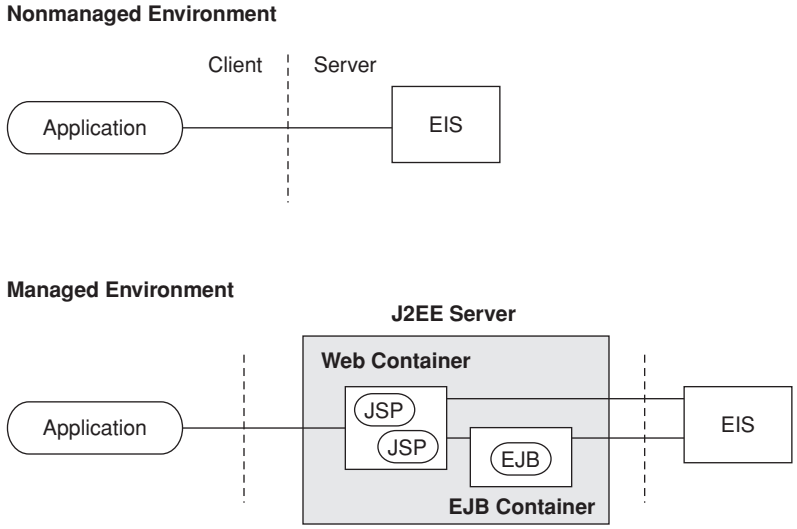

Figure 3.1 Managed and Nonmanaged Environments

Often many clients want concurrent access to the EISs at any one time. However, access to a particular EIS is limited by the number of concurrent physical connections that may be created to that EIS. The number of client sessions that can access the EIS is constrained by the EIS's physical connection limitation. An application server, by providing connection pooling, enables these connections to be shared among client sessions so that a larger number of concurrent sessions can access the EIS.

Web-based applications, in particular, have high scalability requirements. Note that the Connector architecture does not specify a particular mechanism or implementation for connection pooling by an application server. (Our example implementation presented later does demonstrate one possible approach to connection pooling.) Instead, an application server does its own implementation-specific connection pooling mechanism, but, by adhering to the Connector architecture, the mechanism is efficient, scalable, and extensible.

Prior to the advent of the J2EE Connector architecture, each application server implementation provided its own specific implementation of connection pooling. There were no standard requirements for what constituted connection pooling. As a result, it was not possible for EIS vendors to develop resource adapters that would work across all application servers and support connection pooling. Applications also could not depend on a standard support from the application server for connection pooling.

J2EE application servers that support the Connector architecture all provide standard support for connection pooling. At the same time, they keep this connec-

tion pooling support transparent to their applications. That is, the application server completely handles the connection pooling logic and applications do not have to get involved with this issue.

3.1 Connection Management Contract

The Connector architecture provides support for connection pooling and connection management through its connection management contract, one of the three principal contracts defined by the Connector architecture. The connection management contract is of most interest to application server vendors and resource adapter providers because they implement it. However, application developers will also benefit from understanding the application programming model based on the connection management contract.

The connection management contract is defined between an application server and a resource adapter. It provides support for an application server to implement its connection pooling facility. The contract enables an application server to pool its connections to an underlying EIS. It also enables individual application components to connect to an EIS.

The connection management contract defines the fundamentals for the management of connections between applications and underlying EISs. The application server uses the connection management contract to:

- Create new connections to an EIS.
- Configure connection factories in the JNDI namespace.
- Find the matching physical connection from an existing set of pooled connections.

The connection management contract provides a consistent application programming model for connection acquisition. This connection acquisition model is applicable to both managed and nonmanaged environments. More details on the connection acquisition model are given later in this chapter in the section "Application Programming Model." Chapter 12, Connection Management Contract, provides more information on the connection contract itself.

3.2 Connection Management Architecture

To understand how the connection management architecture works, let's look at Figure 3.2, which shows connection management for an application in a managed environment.

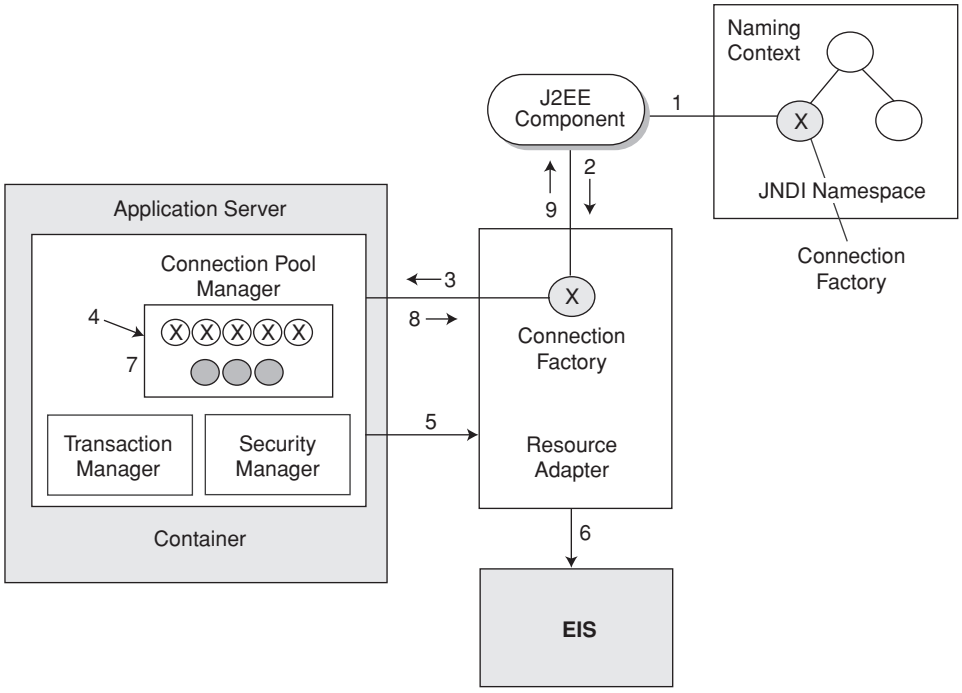

Figure 3.2 Connection Management

The resource adapter provides interfaces for an application component to create a connection to an EIS using a connection factory. An application component that wants to connect to an underlying EIS uses the services of the resource adapter.

Specific steps occur when an application component attempts to establish a connection to an EIS. However, prior to this, the deployer configures the JNDI namespace to include a connection factory. (This is marked as step 1 in Figure 3.2.) The connection factory carries the configuration information, specifically the EIS server and port number, required to create connections to the EIS. Briefly, here is what happens when an application component tries to establish a connection to an EIS.

1. The application component begins by doing a JNDI lookup of a connection factory from the JNDI namespace.
2. The application component, after successfully locating a connection factory, calls a method on the connection factory to create a connection to the EIS. It uses the connection factory to obtain a connection to the underlying EIS.
3. Before it creates the connection, the connection factory, which is provided by the resource adapter, delegates the connection request to the application server

using the connection management contract. The application server provides a connection pool, along with other services such as transaction management, security management, and error logging.

4. The application server, when it receives a request to create a connection, attempts to find a suitable existing connection in the application server's connection pool. The connections in the pool are called managed connections. A managed connection represents an actual physical connection to the underlying EIS. If the application server finds a matching connection in the pool, it uses that matching managed connection to satisfy the application's connection request.
5. If the application server cannot find a matching connection in the pool, then it uses the resource adapter to create a new physical connection—also represented by a managed connection—to the underlying EIS.
6. The resource adapter for the EIS creates a new managed connection by establishing a physical connection to the EIS. A resource adapter returns the newly created managed connection to the application server.
7. The application server adds the new managed connection to its connection pool. As part of this process, the application server creates an application-level connection handle for the managed connection. The application server returns this connection handle to the resource adapter, which in turn returns it to the application component.
8. The application component uses the connection handle returned by the resource adapter to access the EIS.
9. When the application component completes its work with the connection, it closes the connection handle.

What does all this mean to an application? Essentially, the connection management contract enables an application server to offer a number of important benefits to an application.

- It eliminates any dependencies of the application on the connection pooling and keeps the connection pooling transparent to an application. Because connection pooling is implemented independently from an application, the application need not have any knowledge of how the application server accomplishes connection pooling, nor does it have any dependencies on a particular connection pooling mechanism.
- It simplifies the application programming model for the management of connections. An application is not exposed to how the application server and resource adapter use the connection management contract.

- It enables the application server to provide different qualities of services related to EIS integration. Examples of services include transaction management, security management, and error logging and tracing support. An application server can also implement different levels of connection pooling using this connection management contract.
- It increases the scalability of an application. An application can now support a greater number of concurrent client sessions accessing EISs.

3.3 Application Programming Model

An application uses a standard application programming model when obtaining connections. The model is similar whether an application obtains the connection using an application server in a managed environment or whether it obtains the connection independent of the application server.

An application developer follows defined steps and relies on particular information to establish a connection to an underlying EIS. Before the developer can establish a connection, the necessary connection factory needs to be properly deployed and configured. The deployer is responsible for the deployment of a resource adapter and an application. The deployer also deploys and configures the connection factory. What is entailed in deploying and configuring a connection factory? The deployer configures the connection factory by providing configuration information—port number, server name, and so forth. This configuration information represents the information required by a resource adapter to create physical connections to an EIS. Once the connection factory is configured, the application can use the connection factory to create connections to the EIS. See Code Example 3.1.

Code Example 3.1 Establishing a Connection

```
package com.aci;
public class InventoryManagerEJB implements javax.ejb.SessionBean {
    private javax.resource.cci.ConnectionFactory cf;
    ...
    public int getQuantityAvailable(String productId)
             throws InventoryException {
        try {
            javax.resource.cci.Connection cx = getConnection();
            CheckInventoryCommand command =
                new CheckInventoryCommand(cx, cf.getRecordFactory());
```

```
        command.setProductId(productId);
        command.execute();
        // Close the connection.
        cx.close();
        return command.getProductQuantity();
    }
    catch (Exception e) {
        throw new InventoryException();
    }
    ...
}

public void removeFromInventory(String productId,
                            int quantity)
         throws InventoryException {
}

public Connection getConnection() {
    try {
        // Get a connection using the ConnectionFactory.
        Connection cx = cf.getConnection();
        return cx;
    }
    catch (ResourceException re) {
        throw new EJBException(re);
    }
}

private void initialize() {
    try {
        // Use JNDI interface to look up connection
        // factory instance.
        Context nc = new InitialContext();

        // Lookup ConnectionFactory from the JNDI namespace.
        cf = (ConnectionFactory)nc.lookup(
            "java:comp/env/eis/MainframeCxFactory");
    }
    catch (NamingException ne) {
        throw new EJBException(ne);
    }
}
```

```
    public void ejbCreate() throws RemoteException {
        initialize();
    }

    public void setSessionContext(SessionContext sc) {}
    public void ejbRemove() throws RemoteException {}
    public void ejbActivate() { /** Never Called **/ }
    public void ejbPassivate() { /** Never Called **/ }
}
```

Code Example 3.1 uses the Connection and ConnectionFactory interfaces defined in the Common Client Interface. (See Chapter 7, Common Client Interface, for more information on the CCI.) When the application component needs to establish a connection, the developer uses the JNDI naming context to look up a ConnectionFactory instance.

Once the application component has a ConnectionFactory instance, the developer can obtain one or more connections to the EIS by invoking the factory's `getConnection` method. This method returns an application-level connection handle to the underlying physical connection. At this point, the developer may use the common client interface (CCI) application programming model to access the underlying EIS. See Chapter 12, Connection Management Contract, for details on a connection handle, and how an application server and resource adapter collaborate to manage connection pooling.

When the application component completes its work using the connection, the developer calls the `close` method on the connection handle.

3.4 Conclusion

We've seen in this chapter how an application can create connections to underlying EISs. In a managed environment, the application servers manage connection pooling, thus simplifying connection handling for an application. The connection pooling mechanism, because it is handled by the application server, remains transparent to the application.

The connection management contract results in a simplified application programming model. The contract also serves to increase the scalability of application integration with EISs.

Chapter 4 describes the transaction management contract, also from the point of view of an application developer.