Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Theory and Practice of Enterprise JavaBean™ Portability

Santiago Comella Dorda
John Robert
Robert Seacord

June 1999

COTS-Based Systems Initiative

Technical Note
CMU/SEI-99-TN-005

Contents

Abstract		**vii**
1	**Introduction**	**1**
2	**Background**	**5**
3	**Bean Portability**	**7**
4	**Deployment Portability**	**10**
5	**Summary and Conclusions**	**13**
References		**14**

List of Figures

Figure 1. Development Cycle in EJB 3

List of Examples

Example 1. Middleware Effect on Bean Lookup 8
Example 2. JNDI Context Initialization in WebLogic 9
Example 3. WebLogic's Mapping Description 11
Example 4. Ejipt's Mapping Description 11
Example 5. Declarative Definition of a Relation in PowerTier 12

Abstract

The modern enterprise information system (EIS) requires the integration of numerous technologies such as distribution, transactions, data management, security, and naming. Off-the-shelf architectures such as Enterprise JavaBeans™ (EJB) provide a pre-integrated solution that supports the quick development and deployment of information systems. Unfortunately, the EJB specification is extremely porous, leading to portability problems. In addition, the line between vendor extensions and EJB standard functionality is blurred, making it difficult for bean providers to know what functionality can be depended upon across server implementations. This paper presents sources of portability problems in EJB and illustrates them with some real examples. We also present our opinion about the direction the EJB specification should take to enable effective reuse of Enterprise Beans™ between servers.

1 Introduction

Enterprise JavaBeans™ (EJB) is a specification for a component model that promises to simplify the development of multi-tier application systems capable of supporting high-volume business transactions [Spitzer 98]. EJB is not an implementation, but a specification owned by JavaSoft. JavaSoft is acting in the role of a standards organization to expedite the evolution of EJB technology.

Prior to the development of the Enterprise JavaBeans specification in March of 1998, the application server market was segregated into proprietary camps. The EJB specification offered a common model for Java application servers bringing coherence to an otherwise chaotic application server market.

EJB encourages innovation by allowing multiple vendors to develop different implementations of the specification. Most vendors add unique features to core application server functionality to differentiate themselves from their competitors. However, the EJB specification maintains that software developed in an EJB-compliant server[1] can run in another EJB-compliant server seamlessly and without adaptation. In this paper, we examine Enterprise Bean portability among EJB-compliant servers and identify practical obstacles to portability.

To evaluate EJB portability, we created a small EJB test application or *model problem*. This model problem uses many EJB features, including entity and session beans, container-managed persistence, and container-managed transaction demarcation. Using this model problem, we tested single and concurrent clients, different security settings and transaction isolation levels and different naming service implementations. Four EJB platforms were evaluated: WebSphere 2.0 from IBM, WebLogic 3.1.6 from BEA, Ejipt 1.0.2 from Valto, and PowerTier™ from Persistence. However, the specific EJB servers evaluated is not critical as the results of this work can be extrapolated to any EJB platform.

All four EJB servers evaluated claim to be EJB 1.0 compliant—no EJB 1.1 compliant servers were available as of May 1999. The 1.1 release of the specification has made progress in addressing portability issues by clarifying some confusing aspects of the EJB 1.0 specification. However, it is far from a definitive solution and most of our conclusions are

[1] The EJB specification distinguishes between functionality implemented in an EJB server and in an EJB container. To simplify the discussion, we have not made this distinction in this paper because EJB vendors currently bundle both components together as inseparable parts of their implementations.

valid for EJB 1.1 compliant servers. We identify portability problems present in the EJB 1.0 specification and state when these problems have been corrected by the 1.1 specification.

Before presenting the results of these experiments, we briefly discuss some EJB fundamentals and the importance of portability in EJB. This paper is not an introduction to EJB; basic knowledge of the EJB framework and functionality is assumed. For those that are not familiar with EJB please see [Thomas 98, Johnson 98].

1.1 Enterprise JavaBeans

EJB has emerged from the now critical intersection of the Internet and business enterprises. Business enterprises have realized that the Web provides a means to share information and offer services to customers (Internet), business partners (extranets), or even their own employees (intranets). EJB provides several advantages for building Web-based enterprise systems.

The “Write Once, Run Anywhere™” capability makes Java uniquely qualified for building enterprise systems in the multi-platform environment of enterprises. Until now, Java has been primarily used for client-side development because server-side business logic requires more complex services such as transactions, scalability, database integration, naming, and security services. These requirements have been historically addressed using a mix of “traditional” technologies including relational databases, transaction monitors, and naming servers. However, difficulties often arise in the integration of technologies from different vendors that can only be addressed by the vendors concerned. When this occurs, development is effectively held hostage to the whims of vendor priorities, a state of affairs inhibitive to the use of these technologies in enterprise applications [Seacord 99]. In contrast, EJB vendors provide a pre-integrated solution, effectively removing integration issues.

An important benefit of EJB is the component-based approach to application development. The challenge of “better, faster, cheaper” software solutions is driving component-based software engineering (CBSE) to the forefront of EIS development solutions. This building block development process can help organizations reduce software development time by enabling reuse of custom components and the purchase of pre-built third party components. EJB provides a component framework where software components are combined to create complete systems.

Component-based development differs in some aspects from custom development. One difference is that in custom development, all development tasks are performed directly by, or under the direction of, a single organization. In contrast, in a component-based development effort, different organizations can perform different roles in the development.

The Enterprise JavaBeans architecture defines distinct roles in the application development and deployment workflow as shown in Figure 1. The bean *provider* is an application domain expert that develops reusable Enterprise Beans. An application *assembler* integrates beans from multiple bean providers to compose a complete application, developing custom beans when necessary. An EJB *deployer* adapts and customizes EJB applications to run in a specific environment. Enterprise Beans output from each step of the development process become inputs to the next step.

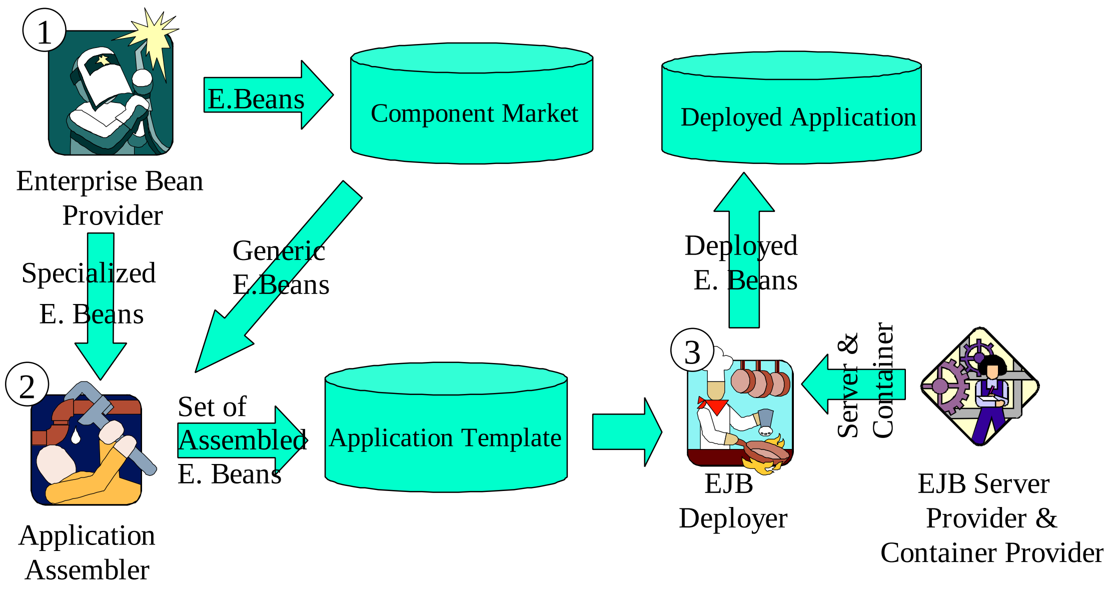


Figure 1. Development Cycle in EJB

As each development role may be performed by a different party using a different EJB server, every step in the development process can be a porting task between different servers. When all roles are performed within a single organization that has selected a common EJB server, portability is a lesser concern.

1.2 Importance of Portability in EJB

The objective of portability in EJB is to allow an Enterprise Bean to be used across EJB servers. The Sun EJB specification [Sun 98a] defines the following goal:

> *Enterprise JavaBeans applications will follow the "write-once, run anywhere" philosophy of the Java programming language. An Enterprise Bean can be developed once, and then deployed on multiple platforms without recompilation or source code modification.*

This goal conflicts with other objectives such as compatibility with existing application servers, differentiation across multiple implementations from different vendors and encouraging continued innovation. Portability of Enterprise Beans between competing server implementations is not a precondition for success, as demonstrated by the success of technologies such as SQL, but a basic level of portability is desirable and beneficial for the EJB community.

The importance of portability depends greatly upon business objectives. Portability has increased importance for

- component vendors that want a broad-based market for their components
- application assemblers that want to reuse pre-built components
- EJB server providers that want to expand the number of third-party components available for their platforms

In contrast, portability may be less important for

- enterprises that have made a strategic decision to use a particular EJB server to take advantage of proprietary features
- organizations that need to custom develop beans to meet non-negotiable requirements or to differentiate their application
- application server providers that want to offer non-standard extensions as a business strategy

Please note that reuse is not dependent on portability—Enterprise Beans can be reused in other applications implemented on the same application server without any concern for portability. However, lack of portability of Enterprise Beans across EJB servers fragments the component market, restricting the number of Enterprise Beans available for a given application server.

2 Background

The Java 2 platform and the Java Virtual Machine (JVM), when correctly implemented, provide an ideal level of portability. Standardized application program interfaces (APIs) in the Java 2 platform provide source code portability, while standardization of the JVM provides for portability of compiled classes.

Although EJB is implemented in Java, this degree of portability is no longer pragmatic. EJB needs to provide for product differentiation as well as for portability. Differences in capabilities arise form the following market necessities, the first two of which are listed as goals in the EJB specification:

1. The Enterprise JavaBeans architecture needs to be compatible with existing server platforms.
2. Vendors need to be able to extend their existing products to support Enterprise JavaBeans.
3. Vendors need be able to differentiate their products by providing implementation-specific enhancements.

These market necessities arise from the significantly different market positions enjoyed by EJB and Java. The initial success of Java was based on the ability to allow users of the World Wide Web to access applications from anywhere on the Internet. Having established a dominant position early, competing vendors did not feel that they could successfully challenge Java in the marketplace. Sun supported this position by licensing Java technology to competitors, allowing them to collaborate in making Java a success rather than forcing them to compete.

Enterprise JavaBeans can be viewed as a push to galvanize support for Java on the server, and move Java beyond the applet paradigm. However, a large number of vendors collaborating with Sun in making Java successful have a vested interest in application servers. For the Enterprise JavaBeans specification to be a success, JavaSoft needed to establish consensus between 19 partners, including IBM, BEA, Oracle, GemStone, and Netscape, each of which contributed to the Enterprise JavaBeans specification.

The EJB specification is actually more of a classification scheme than a traditional specification. Common elements in application servers were identified and gaping holes plugged with new interfaces. As a result, the specification is intentionally vague in areas where existing implementations took conflicting approaches, and no short-term resolution was possible between competing vendors. In some respects, the EJB standardization process more closely resembles the CORBA standardization process managed by the Object

Management Group (OMG). Sun is acting as a central authority to bring about consensus in application server domain, much the same way that the OMG brought about consensus in the area of distributed object technology.

EJB standardization is a multiyear process geared towards bringing application server vendors closer together by developing an increasingly detailed specification, while allowing continued vendor innovations to grow and extend the specification. It is not expected or necessary that absolute portability be achieved at the start. The problem is that the existing specification makes exaggerated claims, leading to heightened expectations and initial disappointments.

As a result of the market necessities just described, the EJB specification is quite porous and many of the vendors that claim compliance to the specification provide significantly different capabilities in their EJB containers or servers. Of course, not every difference between servers is a threat for portability. Vendors can make enhancements to the server that do not impact portability—for example

- smart caching and pooling of objects and resources to improve performance—e.g, database connection pooling
- improved development and management tools
- virtual machines optimizations to run server-side code

In the remainder of this paper, we present some portability problems that we encountered in porting our model problem. We have not attempted to provide a comprehensive list of every difficulty that may be encountered when deploying an Enterprise Bean in a different server. This paper instead attempts to illustrate the kinds of problems that a developer faces when making these migrations.

3 Bean Portability

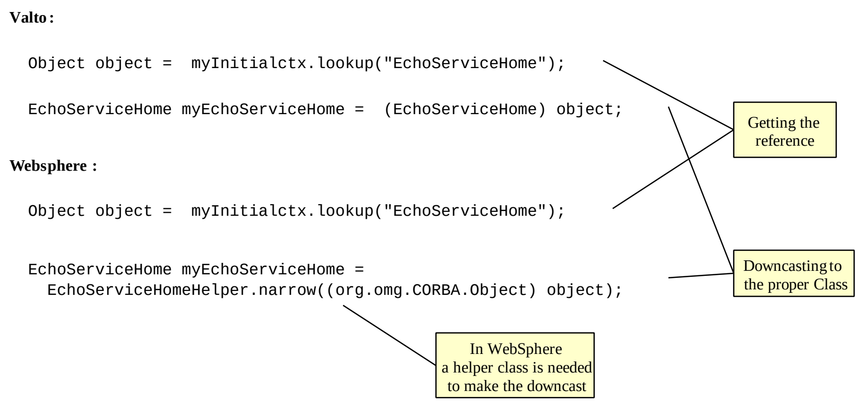
To provide portability for the source code of an Enterprise Bean, EJB relies on API specifications to

- homogenize access to services or tools at source code level
- define a common interface to access resources and functionality

EJB defines its own API for server/container interfaces to Enterprise Beans. EJB also relies on an alphabet soup of high level APIs including JTS, JDBC, RMI, JIDL, and JNDI. These APIs, along with additional APIs such as JMS and the Java servlet API, are not part of EJB, but rather form the Java Platform for the Enterprise (JPE).

Surprisingly, basic services of JPE in different servers do not present homogeneous interfaces. For example, the EJB 1.0 Specification does not specify which JDK version should be used [IONA 98]. Most EJB server vendors support JDK1.1.x and some support Java 2, and all claim to be EJB 1.0 compliant. JDK 1.1 and Java 2 have multiple incompatibilities with significant incompatibilities in APIs. The EJB 1.1 specification states that to be portable an Enterprise Bean must be careful to use only JDK 1.1 APIs. This does not address how semantic differences, for example, in the Java remote method protocol (JRMP) are addressed. Of the four EJB servers we evaluated, two supported JDK 1.1 (WebSphere, WebLogic) and two supported Java 2 (Ejipt, PowerTier).

Even using the same JDK release, servers can have API incompatibilities. For example, the RMI API can use different middleware protocols, like the native Java Remote Method Protocol (JRMP) or OMG standard inter-ORB interoperability protocol (IIOP). Unfortunately, there is no middleware transparency in RMI, because different capabilities in IIOP and JRMP make it difficult to hide the underlying protocol from RMI users. Example 1 illustrates these differences with two examples of source code, one from a Valto client using native RMI and another from a WebSphere client using IIOP. Each client is attempting to get an instance of a bean home interface from the Java Naming and Directory Interface (JNDI) naming service.



Valto :

```
  Object object =  myInitialctx.lookup("EchoServiceHome");

  EchoServiceHome myEchoServiceHome =  (EchoServiceHome) object;
```

Websphere :

```
  Object object =  myInitialctx.lookup("EchoServiceHome");

  EchoServiceHome myEchoServiceHome =
    EchoServiceHomeHelper.narrow((org.omg.CORBA.Object) object);
```

Getting the reference
Downcasting to the proper Class
In WebSphere a helper class is needed to make the downcast

Example 1. Middleware Effect on Bean Lookup

While these interface or syntactic problems can be found at compilation time and are relatively easy to resolve, this specific example is alarming because it affects the portability of the client to run with the same bean in different EJB servers. This fact is clearly in contradiction with the goals of EJB, as presented in the following statement from the EJB specification:

> *A client's view of an EJB object is the same, irrespective of the implementation of the Enterprise Bean and its container. This ensures that a client application is portable across all container implementations in which the Enterprise Bean might be deployed.*

This middleware transparency issue has been addressed in the 1.1 release of the EJB specification. This release states that the type narrowing must be performed using the narrow method of the portable remote object. In the EJB 1.0 specification this practice was only recommended.

A more difficult problem to find occurs when two implementations of a service share the same syntax but have different semantics. Semantic differences are harder to find, because they cannot be detected at compile time and they usually produce cryptic errors at runtime.

For example, the JNDI API defines an initial context factory interface containing a method that returns the initial context. This method accepts a single parameter: a hash-table of property-value pairs. This table of property-value pairs represents the environment in which the naming service is accessed, including requirements for security and level of service. This construction is extremely flexible: any set of properties can be passed to the factory. The JNDI specification defines a minimal set of properties, but enables vendors to extend this set with properties specific to their products.

In Example 2, we create an initial context that uses another remote naming service as delegate of service. This example can be compiled and deployed in any EJB compliant server, but produces a run time error in any server not using WebLogic's implementation of the naming service. That is because the delegate environment property is specific to WebLogic's JNDI. [2]

Creating the environment of the remote service provider

```
Hashtable delegateEnv = new Hashtable();
delegateEnv.put(...); //properties of delegated environment
...
```

Adding the delegate provider to the local provider

```
Hashtable env = new Hashtable();
env.put(TengahContext.DELEGATE_ENVIRONMENT, delegateEnv);
env.put(...); // rest of properties on the local environment
...

Context ctx = new InitialContext(env);
```

Getting the initial context

Example 2. JNDI Context Initialization in WebLogic

Almost every EJB server includes proprietary libraries. Some of these APIs make available functionality that is not included (or not yet included) by the JPE—such as a time service or extensible markup language (XML) manipulation. If the EJB developer needs any of these services, it is difficult to avoid getting locked into a proprietary solution. For example, the event service for JPE has not yet been released by Sun. Users that need reliable asynchronous communication have to employ proprietary versions of this service, or build their own event service with Java (a non-trivial problem).

[2] In JNDI 1.2, now in beta release, a new mechanism for using resource files has been added that allows applications to provide configuration dependent properties. This will allow these properties to be established at deployment time, partially addressing the portability problem.

4 Deployment Portability

Deployment descriptors are used to establish the runtime service properties for an Enterprise Bean. These properties tell the EJB container how to manage and control the Enterprise Bean [Thomas 98]. Deployment properties are attributes of the server that can be used as required, and vary with the capabilities of the server. Servers that support fine control over object pools, for example, have a property for the number of instances of a bean in the pool, while servers that provide course control do not.

4.1 Vendor-Specific Notation

The most readily apparent fact when porting a deployment descriptor to a different server is the differences in notations. As of the 1.0 release of the EJB specification, syntax of the deployment descriptor is vendor dependent. JavaSoft is aware of this problem and has standardized deployment description notation using XML in the 1.1 release of the EJB specification [Sun 99].

4.2 Server Functionality

There is a basic set of deployment properties defined by the standard that every vendor must support, including the name of the home and remote interface in JNDI, the access control list for the Bean and the container-managed field. In EJB 1.0, these properties are described in three classes defined by the specification: the deployment descriptor, entity descriptor, and session descriptor. The EJB 1.1 specification deprecates these classes, and substitutes the serialized deployment descriptor for an XML file. It also adds more standard properties and dictates the properties that must be defined by each role. However, it is difficult, if not impossible, to limit a development effort to the use of only these properties. In most applications, it is necessary to use proprietary properties to deploy an Enterprise Bean.

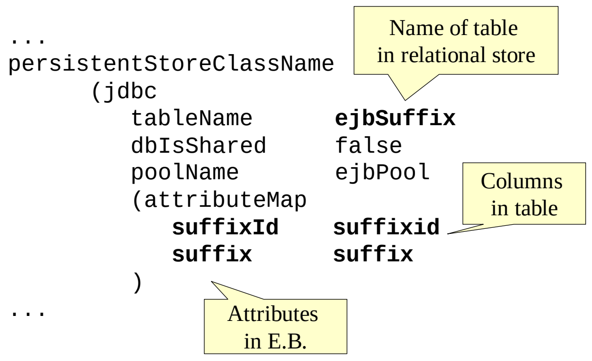
Differences in EJB server capabilities are mirrored in the deployment attributes. When migrating an Enterprise Bean to a new EJB server, deployers may find that the new server's deployment descriptor does not support properties upon which an Enterprise Bean is reliant. Sometimes the only solution is to transfer this functionality to the business logic of the Enterprise Bean. This is an unpleasant task—assuming the deployer has access to the Enterprise Bean's source code and can make the changes at all.

We believe the standardized core set of properties must be extended to enable the development of applications without reliance on proprietary properties. Of course, specialized needs or specific niche markets could make use of these extensions.

The extension of the standard set of properties is a difficult problem because of the varied capabilities of EJB servers. For example, the mapping between EJB model and persistent store may be completely different if the persistent store is a relational database or an object oriented database. It may be necessary to restrict the specification to provide actual portability across a more narrow range of technologies than to provide limited portability across a broad range of technologies.

Object Persistence

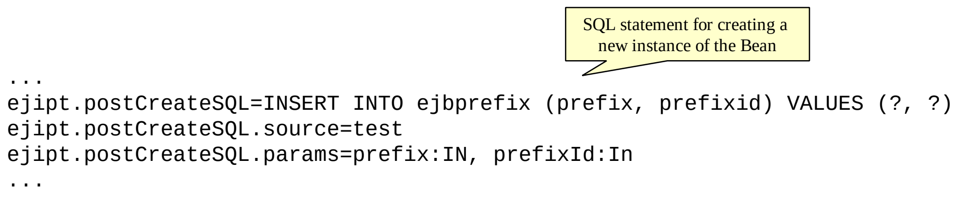
One area where differences between server implementations are substantial is the mapping between the EJB object model and the underlying data store. Most EJB servers use a relational database as a persistent repository, while others may use either an object repository or flat file. With container-managed Beans, this mapping must be described in the deployment descriptor. Current EJB server implementations have different approaches for providing this mapping. For example, in the WebLogic server an entity bean is always mapped to a row in a table. Every state data variable in the Enterprise Bean is mapped to a specific column in a table in the relational database as shown in Example 3.



```
...
persistentStoreClassName
      (jdbc
         tableName      ejbSuffix
         dbIsShared     false
         poolName       ejbPool
         (attributeMap
            suffixId    suffixid
            suffix      suffix
         )
...
```

Example 3. WebLogic's Mapping Description

Ejipt’s relational mapping is more complex. EJB deployers write a set of SQL statements to transfer data. This solution, shown in Example 4, enables the deployer to specify more intricate mappings, but also requires greater development effort.



```
...
ejipt.postCreateSQL=INSERT INTO ejbprefix (prefix, prefixid) VALUES (?, ?)
ejipt.postCreateSQL.source=test
ejipt.postCreateSQL.params=prefix:IN, prefixId:In
...
```

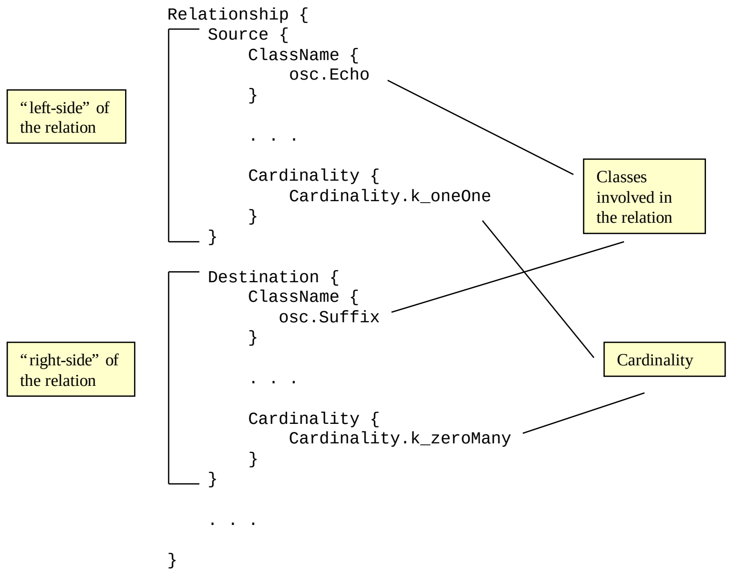
Example 4. Ejipt's Mapping Description

Mappings that can be easily described in one EJB server’s deployment descriptor may be impossible to describe in another. If these complex mappings are used, and the Enterprise

Bean is migrated to a server that does not support them, the bean must be converted from container to bean managed.

Container-Managed Relations

Another example of deployment descriptor incompatibilities can be found in container-managed relations. In most enterprise information systems, business objects are related. For example, customers have accounts and providers offer products. The EJB specification does not describe how to express relations between container-managed entity beans. Some EJB servers allow these relations to be expressed declaratively (i.e., in the deployment descriptor), while others do not. Of the evaluated servers, only PowerTier™ for EJB supports relations managed in the deployment descriptor. WebSphere claims support for relations in a future release and WebLogic provides support through TOPLink™, a plug-in from the Object People. Example 5 shows a segment of the deployment illustrating the specification of a relation in a deployment descriptor.



```
Relationship {
    Source {
        ClassName {
            osc.Echo
        }

        . . .

        Cardinality {
            Cardinality.k_oneOne
        }
    }

    Destination {
        ClassName {
           osc.Suffix
        }

        . . .

        Cardinality {
            Cardinality.k_zeroMany
        }
    }

    . . .

}
```

"left-side" of the relation
"right-side" of the relation
Classes involved in the relation
Cardinality

Example 5. Declarative Definition of a Relation in PowerTier

Finder Methods

A finder method is a service of the home interface used to locate entity beans. Depending on the EJB Server, a finder method may be defined in source code or declaratively in the deployment descriptor. For example, WebLogic defines a restricted language that permits declaration in the deployment descriptor of simple queries. In contrast, WebSphere requires developers to create Java helper classes with methods that return the SQL sentences of the finder. The WebSphere solution is more flexible, but must be implemented in source code, including detailed representations of the Enterprise Bean in the database. This defeats the principal benefit of container-managed persistence—allowing the Enterprise Bean source code to be independent of the underlying data source.

5 Summary and Conclusions

Pourousness in the EJB specification is a source of portability problems in Enterprise JavaBeans. Sun has recognized this problem and is taking steps to address it. At the end of 1998 Sun announced a roadmap [Sun 98b] consisting of a three-phased plan in which increased portability is a major consideration:

1. Phase one of this roadmap will provide enhancements to the specification to allow developers to write Enterprise Beans that can easily install and run anywhere. [3]
2. Enhancements made to the specification in phase two will provide vendors with a universal way to connect to existing systems without sacrificing portability.
3. Phase three will take this a step further by making the mapping process between enterprise class systems and Enterprise JavaBeans seamless and automatic. Developers will be able to create enterprise-class apps without concern for the underlying enterprise infrastructure. A developer could write a checking account bean, for instance, without regard for the underlying database, transaction server, or directory server.

An underlying source of portability problems in EJB is the conflict between the needs of a multi-vendor/multi-niche market and the necessity of an acceptable level of portability. Vendors should be free to provide enhanced features, but there should be a well-defined minimal common set of capabilities. This standard EJB core must comprise both source code interfaces and deployment descriptor's properties, and be complete enough to implement most enterprise applications. Without this common core functionality, the establishment of a component market in Enterprise Beans is seriously jeopardized. Currently, the line between vendor extensions and EJB standard functionality is blurred, making it difficult for bean providers to know what functionality can be depended upon across server implementations. Moreover, the capabilities standardized by EJB are often insufficient to develop commercial applications without using proprietary extensions.

[3] A public draft of the EJB 1.1 specification was released on May 11, 1999. However, most EJB products will not be EJB 1.1. compliant until late 1999 or early 2000.

References

Iona 98	IONA Technologies.® "Outstanding Issues" [online]. Dublin, Ireland: IONA Technologies. Available WWW: <URL: http://www.ejbhome.com/holes.htm> 1998.
Johnson 98	Johnson, Mark. "A Beginner's Guide to Enterprise JavaBeans" [online]. *Java World 3,* 10 (October 1998). Available WWW: <URL: http://www.javaworld.com/javaworld/jw-10-1998/jw-10-beans.html>.
Seacord 99	Seacord, Robert C.; Wallnau, Kurt; Robert, John; Comella Dorda, Santiago; & Hissam, Scott A. *Custom vs. Off-The-Shelf Architecture* (CMU/SEI-99-TN-006). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, May 1999.
Spitzer 98	Spitzer, Tom. "Component Assembler." *DBMS Online*. August 1998. Available WWW: <URL: http://www.dbmsmag.com/9808d18.html>.
Sun 98a	Sun Microsystems.® *Enterprise JavaBeans™ 1.0* [online]. Palo Alto, Ca.: Sun Microsystems. Available FTP: <URL: ftp://ftp.javasoft.com/docs/ejb/ejb.10.pdf>, 1998.
Sun 98b	Sun Microsystems. "Sun Delivers Enterprise JavaBeans™ Technology Roadmap" [online]. Palo Alto, Ca.: Sun Microsystems. Available WWW: <URL: http://java.sun.com/pr/1998/12/pr981208-05.html>, December 8, 1998.
Sun 99	Sun Microsystems. *Enterprise JavaBeans™ 1.1* [online]. Palo Alto, Ca.: Sun Microsystems. Available FTP: <URL: http://www.javasoft.com/products/ejb/docs.html>, 1999.
Thomas 98	Thomas, Anne. "Enterprise JavaBeans ™ Technology. Server Component Model for the Java Platform" [online]. Palo Alto, Ca.: Sun Microsystems. Available WWW: <URL: http://java.sun.com/products/ejb/white_paper.html>, 1998.

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.		
1. AGENCY USE ONLY (LEAVE BLANK)	2. REPORT DATE June 1999	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Theory and Practice of Enterprise JavaBean Portability		5. FUNDING NUMBERS C — F19628-95-C-0003
6. AUTHOR(S) Santiago Comella Dorda, John Robert, Robert Seacord		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-99-TN-005
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/DIB 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12.A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12.B DISTRIBUTION CODE
13. ABSTRACT (MAXIMUM 200 WORDS) The modern enterprise information system (EIS) requires the integration of numerous technologies such as distribution, transactions, data management, security, and naming. Off-the-shelf architectures such as Enterprise JavaBeans™ (EJB) provide a pre-integrated solution that supports the quick development and deployment of information systems. Unfortunately, the EJB specification is extremely porous, leading to portability problems. In addition, the line between vendor extensions and EJB standard functionality is blurred, making it difficult for bean providers to know what functionality can be depended upon across server implementations. This paper presents sources of portability problems in EJB and illustrates them with some real examples. We also present our opinion about the direction the EJB specification should take to enable effective reuse of Enterprise Beans™ between servers.		
14. SUBJECT TERMS component-based software engineering (CBSE), Enterprise JavaBeans (EJB), information system, Java, off-the-shelf architecture, portability, reuse		15. NUMBER OF PAGES 14 pp. 16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102