UML Profile For EJB

This is a draft of the UML Profile For EJB Specification. The UML Profile For EJB is a mapping from the Unified Modeling Language (UML) to the Enterprise JavaBeans (EJB) architecture. The mapping specifies the standard representation for elements of the EJB architecture in UML models.

Please send comments to: uml-ejb-specification-lead@rational.com.

Public Draft

Jack Greenfield

Rational Software Corporation

DISCLAIMER

This document and its contents are furnished "as is" for informational purposes only, and are subject to change without notice. Rational Software Corporation (Rational Software) does not represent or warrant that any product or business plans expressed or implied will be fulfilled in any way. Any actions taken by the user of this document in response to the document or its contents shall be solely at the risk of the user.

RATIONAL SOFTWARE CORPORATION MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, WITH RESPECT TO THIS DOCUMENT OR ITS CONTENTS, AND HEREBY EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR USE OR NON-INFRINGEMENT. IN NO EVENT SHALL RATIONAL SOFTWARE BE HELD LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING FROM THE USE OF ANY PORTION OF THE INFORMATION.

Copyright

Copyright © 1999, 2000, 2001 Rational Software Corporation. All rights reserved.

Copyright © 1999, 2000, 2001 Fujitsu Limited. All rights reserved.

Copyright © 1999, 2000, 2001 IBM Corporation. All rights reserved.

Copyright © 1999, 2000, 2001 IONA Technologies, PLC. All rights reserved.

Copyright © 1999, 2000, 2001 Number Six Software, Inc. All rights reserved.

Copyright © 1999, 2000, 2001 Oracle Corporation. All rights reserved.

Copyright © 1999, 2000, 2001 SolNet Limited. All rights reserved.

Copyright © 1999, 2000, 2001 Sun Microsystems, Inc. All rights reserved.

Copyright © 1999, 2000, 2001 Unisys Corporation. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or documentation may be reproduced in any form by any means without prior written authorization of the copyright holders, or any of their licensors, if any. Any unauthorized use may be a violation of domestic or international law.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the U.S. Government and its agents is subject to the restrictions of FAR 52.227- 14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

Trademarks

Rational, Rational Software Corporation, the Rational Software logo and Rational Rose are trademarks or registered trademarks of Rational Software Corporation.Sun, Sun Microsystems, the Sun logo, Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc.

Sun, Sun Microsystems, the Sun Logo, Solaris, Java, and Enterprise JavaBeans, are trademarks or registered trademarks of Sun Microsystems, Inc in the U.S. and other countries.

OMG, CORBA and all CORBA-based trademarks and logos are trademarks or registered trademarks of the Object Management Group.

PostScript is a registered trademark of Adobe Systems, Inc.

OMG, OBJECT MANAGEMENT GROUP, CORBA, CORBA ACADEMY, CORBA ACADEMY & DESIGN, THE INFORMATION BROKERAGE, OBJECT REQUEST BROKER, OMG IDL, CORBAFACILITIES, CORBASERVICES, CORBANET, CORBAMED, CORBADOMAINS, GIOP, IIOP, OMA, CORBA THE GEEK, UNIFIED MODELING LANGUAGE, UML, and UML CUBE LOGO are registered trademarks or trademarks of the Object Management Group, Inc.

All other product or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

Table Of Contents

1 INTRODUCTION 6

1.1 TARGET AUDIENCE 6
1.2 ACKNOWLEDGMENTS 6
1.3 CONVENTIONS 6
1.4 ORGANIZATION 7

2 OVERVIEW 8

2.1 GOALS 8
2.2 RELATED SPECIFICATIONS 9
2.3 UML EXTENSIONS 9
2.4 UML PROFILES 10
2.5 SPECIFICATION SCOPE 10

3 UML PROFILE 11

3.1 DEFINITION OF TERMS 11
3.2 SUPPORTED META-MODEL ELEMENTS 12
3.3 META-MODEL EXTENSIONS 12
3.4 PRE-DEFINED COMMON MODEL ELEMENTS 12
3.5 STANDARD EXTENSIONS 12
3.6 SEMANTICS 18
3.7 WELL-FORMEDNESS RULES 52

4 UML DESCRIPTOR 53

4.1 OVERVIEW 53
4.2 UML DESCRIPTOR DTD 53

5 VIRTUAL METAMODEL 54

5.1 BACKGROUND 54
5.2 VMM PACKAGE STRUCTURE 55
5.3 PACKAGE JAVA::UTIL::JAR 55
5.4 PACKAGE JAVA::LANG 56
5.5 PACKAGE JAVAX::EJB 57

6 RATIONALE 60

6.1 PROCESS 60
6.2 NOTATION 60
6.3 REPRESENTATION 61
6.4 RELATIONSHIPS 63
6.5 IMPLEMENTATION 63

7 EXAMPLES 66

7.1 ITSO BANK EXAMPLE 66

8 RELATED DOCUMENTS 73

9 REVISION HISTORY 74

9.1 CHANGES SINCE PARTICIPANT DRAFT CANDIDATE 1 74
9.2 CHANGES SINCE PARTICIPANT DRAT CANDIDATE 4 75

Table Of Figures

TABLE 3-1. DEFINITION OF TERMS 11
TABLE 3-2. JAVA DESIGN MODEL STEREOTYPES 13
TABLE 3-3. EJB DESIGN MODEL STEREOTYPES - EXTERNAL VIEW 13
TABLE 3-4. EJB DESIGN MODEL STEREOTYPES - INTERNAL VIEW 14
TABLE 3-5. JAVA IMPLEMENTATION MODEL STEREOTYPES IN PACKAGE JAVA::LANG 14
TABLE 3-6. JAVA IMPLEMENTATION MODEL STEREOTYPES IN PACKAGE JAVA::UTIL::JAR 14
TABLE 3-7. EJB IMPLEMENTATION MODEL STEREOTYPES 15
TABLE 3-8. JAVA DESIGN MODEL TAGGED VALUES 16
TABLE 3-9. EJB DESIGN MODEL TAGGED VALUES - EXTERNAL VIEW 16
TABLE 3-10. EJB DESIGN MODEL TAGGED VALUES - INTERNAL VIEW 17
TABLE 3-11. JAVA CLASS TAGGED VALUES 20
TABLE 3-12. JAVA INTERFACE STEREOTYPES 22
TABLE 3-13. JAVA INTERFACE TAGGED VALUES 22
TABLE 3-14. JAVA FIELD TAGGED VALUES 25
TABLE 3-15. JAVA METHOD TAGGED VALUES 27
TABLE 3-16. JAVA METHOD PARAMETER TAGGED VALUES 27
TABLE 3-17. EJB METHOD STEREOTYPES 28
TABLE 3-18. EJB REMOTE INTERFACE STEREOTYPES 30
TABLE 3-19. EJB HOME INTERFACE STEREOTYPES 31
TABLE 3-20. EJB SESSION HOME STEREOTYPES 32
TABLE 3-21. EJB SESSION HOME TAGGED VALUES 32
TABLE 3-22. EJB ENTITY HOME STEREOTYPES 33
TABLE 3-23. EJB PRIMARY KEY CLASS STEREOTYPES 34
TABLE 3-24. EJB METHOD TAGGED VALUES 36
TABLE 3-25. EJB REMOTE INTERFACE STEREOTYPES 36
TABLE 3-26. EJB HOME INTERFACE STEREOTYPES 37
TABLE 3-27. EJB IMPLEMENTATION CLASS STEREOTYPES 38
TABLE 3-28. EJB ENTERPRISE BEAN STEREOTYPES 41
TABLE 3-29. EJB ENTERPRISE BEAN TAGGED VALUES 41
TABLE 3-30. EJB SESSION BEAN STEREOTYPES 43
TABLE 3-31. EJB SESSION BEAN TAGGED VALUES 43
TABLE 3-32. EJB ENTITY BEAN STEREOTYPES 45
TABLE 3-33. EJB ENTITY BEAN TAGGED VALUES 45
TABLE 3-34. JAVA CLASS FILE STEREOTYPES 48
TABLE 3-35. JAVA ARCHIVE FILE STEREOTYPES 49
TABLE 3-36. EJB-JAR STEREOTYPES 50
TABLE 3-37. EJB DEPLOYMENT DESCRIPTOR STEREOTYPES 51
TABLE 3-38. EJB DEPLOYMENT DESCRIPTOR CLIENT JAR STEREOTYPES 51
FIGURE 5-1. VIRTUAL METAMODEL PACKAGES 55
FIGURE 5-2. STEREOTYPE DEFINED IN PACKAGE JAVA::UTIL::JAR 55
FIGURE 5-3. STEREOTYPES AND TAGGED VALUES DEFINED IN PACKAGE JAVA::LANG 56
FIGURE 5-4. STEREOTYPES DEFINED IN EJB DESIGN MODEL - EXTERNAL VIEW 57
FIGURE 5-5. STEREOTYPES AND TAGGED VALUES DEFINED IN EJB DESIGN MODEL - INTERNAL VIEW 58
FIGURE 5-6. STEREOTYPES DEFINED IN EJB IMPLEMENTATION MODEL 59
FIGURE 7-1. ITSO BANK EXAMPLE ANALYSIS MODEL – CORE CLASSES 66
FIGURE 7-2. ITSO BANK EXAMPLE ANALYSIS MODEL 67
FIGURE 7-3. ITSO BANK EXAMPLE - EXTERNAL VIEW 68
FIGURE 7-4. ITSO BANK EXAMPLE CUSTOMER ENTITY - INTERNAL VIEW 69
FIGURE 7-5. ITSO BANK EXAMPLE - INTERNAL VIEW - EJB REFERENCES 69
FIGURE 7-6. ITSO EXAMPLE - INTERNAL VIEW - SECURITY ROLE REFERENCES 70

FIGURE 7-7. ITSO BANK EXAMPLE CUSTOMER ENTITY - INTERNAL VIEW DETAIL 71
FIGURE 7-8. ITSO BANK EXAMPLE CUSTOMER ENTITY - IMPLEMENTATION VIEW 72

1 Introduction

This document describes a standard mapping between the Enterprise JavaBeans[1] (EJB) architecture, including the portions of the Java language on which it depends[2], and the Unified Modeling Language[3] (UML).

The EJB architecture is a component architecture for the development and deployment of component-based distributed business applications. Applications written using the EJB architecture are scalable, transactional, and multi-user secure. These applications may be written once, and then deployed on any server platform that supports the EJB Specification.

The UML is a graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML Profile For EJB lets software designers use the UML to describe software systems based on the EJB architecture. It defines standard representations for EJB-based software artifacts in the UML, providing a basis for forward engineering artifacts from UML models and for reverse engineering UML visualizations from artifacts.

1.1 Target Audience

The target audience for this specification includes:

- *Vendors of transaction processing platforms, enterprise application tools and other products based on the EJB architecture;*
- *Vendors of modeling tools, enterprise application tools and other products based on the UML;*
- *Software developers who design or implement EJB-based software systems, or who use UML to design or implement enterprise applications.*

1.2 Acknowledgments

The UML Profile for EJB is a collaborative effort that has benefited from contributions by numerous groups and individuals throughout the industry. Sun Microsystems and the Object Management Group provided the technical foundations and collaborative frameworks for this effort. Special thanks go to James Abbott and Loïc Julien, who worked closely with Jack in order to release this specification, to David Frankel, who prepared the virtual metamodel, to Scott Rich, who prepared the example. CT Arrington, Don Baisley, Grady Booch, Steve Brodsky, Jim Conallen, Linda DeMichiel, Desmond D'Souza, Bill Dudney, Don Ferguson, Shel Finkelstein, Jeffrey Hammond, Mark Hapner, Sridhar Iyengar, Cris Kobryn, Vlada Matena, Jeff Norton, David Page, Dmitri Plotnikov, Guus Ramackers, Jim Tremlett, Kevin Wittkopf and Hiroyuki Yoshida also helped to make this specification possible through logistical or technical assistance, or through material contributions.

1.3 Conventions

The regular Times font is used for information that is prescriptive.

The italic Times font is used for paragraphs that contain descriptive information, such as notes describing typical use, or notes clarifying the prescriptive specification.

The Helvetica font is used for code examples and keywords.

[1] This document is based on the Enterprise JavaBeans Specification Version 1.1 Public Release, available from Sun Microsystems, Inc.

[2] This document is based on the Java Language Specification, Second Edition, available from Sun Microsystems, Inc.

[3] This document is based on the Unified Modeling Language Specification, version 1.3, available from the Object Management Group, Inc.

1.4 Organization

Chapter 1, Introduction, describes the target audience for this specification, acknowledges contributors, describes the organization of the document, and defines typographical conventions.

Chapter 2, Overview, lists the goals of the specification, describes the standard UML extension mechanisms, explains the reasons for extending UML, and defines the scope of the specification.

Chapter 3, UML Profile, presents the formal definition of the UML profile for EJB.

Chapter 4, UML Descriptor, defines the facility used to associate UML models with EJB-based artifacts in EJB-JARs.

Chapter 5, Virtual Metamodel, describes the UML standard extensions defined by this specification using a virtual metamodel.

Chapter 6, Rationale, describes salient aspects of the approach used to develop the profile.

Chapter 7, Examples, presents examples of the UML Profile.

Chapter 8, Related Documents, identifies other documents used or referenced by this specification.

Chapter 9, Revision History, describes changes made in each published version of the document.

2 Overview

The UML is widely used to specify, visualize, construct, and document the types of enterprise applications for which the EJB architecture was designed. Conversely, the EJB architecture is widely used to implement the types of enterprise applications most frequently described by UML models. There is therefore significant motivation for ensuring that the UML can be used to describe EJB-based software systems.

While the UML already provides standards for the design of object-oriented systems in general, including enterprise computing systems, it does not provide everything necessary for the design of systems based on specific implementation architectures. In particular, it does not explicitly capture the semantics expressible in the EJB architecture[4].

The UML was designed to be extensible, however, and provides standard extension mechanisms for defining new semantic constructs. These mechanisms can be used to define constructs describing EJB-based software artifacts. Ad hoc attempts to do this have been made by the industry over the past several years. To ensure the interoperability of tools and frameworks from different vendors, and the portability of the applications they support, this specification defines a standard set of UML extensions for modeling EJB-based software systems.

In addition, since tool and framework vendors often place UML models describing EJB-based artifacts in the EJB-JARs that contain those artifacts to support display, automation and reflection, this specification defines a standard format for storing a UML model that describes the contents of an EJB-JAR within the EJB-JAR.

Collectively, these definitions comprise the UML Profile for EJB. The UML Profile for EJB can be used to develop models describing EJB-based implementation artifacts, and to round trip engineer between the models and the artifacts.

2.1 Goals

The UML Profile for EJB has the following goals:

- *To define a standard approach to modeling for the Java programming language and the EJB architecture.*
- *To support all practical requirements commonly encountered in the design of EJB-based systems.*
- *To define complete, accurate and unambiguous representations of EJB artifacts and their semantics for modeling purposes.*
- *To support the construction of models of EJB assemblies by combining packages created with tools from different vendors.*
- *To simplify the design of EJB-based software, so that developers do not have to define ways to model EJB-based artifacts or to round trip engineer between models and EJB-based artifacts.*
- *To support the development, deployment, and runtime phases of the EJB life cycle.*
- *To enable tools from multiple vendors to interchange models of EJB-based artifacts using standard interchange formats.*
- *To be compatible with UML 1.3. Vendors will be able to extend UML 1.3 compliant products to support the profile.*
- *To be compatible with all applicable Java programming language APIs and standards.*
- *To be compatible with UML profiles for related technologies, including CORBA and the CORBA Component Model.*

[4] At the time of writing, the UML 1.4 Revision Task Force (RTF) of the OMG's Analysis & Design Platform Task Force (ADPTF) has released a draft of the UML specification version 1.4 that supports a more direct approach to modeling distributed components such as EJBs. This specification will be revised to take full advantage of these change to the UML.

2.2 Related Specifications

In addition to the Java Language Specification, the Enterprise JavaBeans Specification, and the Unified Modeling Language specification, the documents listed below deal with subjects relevant to the UML Profile for EJB.

- *Meta Object Facility Specification (OMG documents ad/99-09-05, et. al.)*

  The Meta Object Facility (MOF) defines a meta-metamodel that can describe many types of metamodels. In addition, the MOF defines a set of IDL interfaces for creating and manipulating metamodels. MOF 1.3 was used to define the physical metamodel for UML 1.3. That metamodel is the basis of the UML profile defined by this document.

- *XML Metadata Interchange Format Specification (OMG document ad/99-10-02, et. al.)*

  The XML Metadata Interchange Format (XMI) defines a mechanism for mapping between MOF based metamodels and XML DTD*s. These DTDs define interchange formats for the mapped metamodels. XMI 1.1 was used to produce the DTD that defines the physical representation of the UML 1.3 metamodel. Without XMI, the mapping of UML constructs to EJB constructs could be ambiguous and open to proprietary interpretation by tools and applications. Because only standard extension mechanisms of UML 1.3 are used by the profile, the UML descriptor defined by this document is a strict subset of the OMG UML 1.3* XML DTD*.*

- *Java Metadata Interface Specification (Java Community Process JSR-000040)*

  The Java Metadata Interface (JMI) defines a set of Java interfaces for defining and manipulating MOF-based metamodels and the metadata they describe. These interfaces form the basis of the reference implementation for the UML Profile for EJB, and can be used to access the UML descriptor defined by this specification.

2.3 UML Extensions

This specification defines a set of UML extensions that capture the structure and semantics of EJB-based artifacts.

The UML defines several standard extension mechanisms, including Stereotypes, Constraints, Tagged Values *and icons. These mechanisms can be used to specialize or refine the UML for a specific purpose.*

A Stereotype *creates a virtual UML metaclass based on an existing UML metaclass, providing a way to classify the base metaclass instances, and optionally to specify additional* Constraints *or required* Tagged Values*. For example, the* Stereotype «topLevelPackage» *extends the UML metaclass* Package*. When applied to a* Package *instance, it constrains the instance to be not nested within another instance.*

A Tagged Value *acts like an attribute of a UML metaclass, allowing arbitrary information to be attached to an instance. A set of* Tagged Values *can be associated with a* Stereotype*, to be applied to model elements carrying the* Stereotype*.*

A Constraint *allows semantics for a model element to be specified linguistically. The specification is written as an expression in a designated constraint language. The formal* Constraint *language used by the UML metamodel is the Object Constraint Language (OCL).* Constraints *can also be specified informally, using natural language.*

One or more Constraints *can be applied to any model element to govern the use of its instances. In addition,* Constraints *can be associated with a* Stereotype*, so that they are applied to model elements classified by the* Stereotype*.*

Stereotypes *are* GeneralizableElements *in the UML metamodel. They can therefore participate in* Generalization *and* Dependency *relationships. This specification uses* Generalizations *between* Stereotypes*. Because* Stereotypes *are not* Classifiers*, they can not participate in* Association *relationships.*

An abstract GeneralizableElement *can not be instantiated in a UML model. Abstract* Stereotypes *can be used to generalize common properties of related* Stereotypes*. This specification uses abstract* Stereotypes*.*

Common model elements are pre-defined instances of UML metamodel elements. A set of common model elements for Java 2, Standard Edition, for example, might define a UML Class *describing the* Java Class java.lang.String*. The other standard extension mechanisms may be used in defining common model elements.*

2.4 UML Profiles

The UML extensions introduced by this specification are organized as UML profiles.

According to the UML specification, a UML profile specializes or refines the UML for a specific purpose using the standard UML extension mechanisms. A profile does not add any basic concepts. Instead, it specializes existing concepts and defines conventions for applying them. ADPTF RFP7[5] adds to this definition the requirement that a UML profile specification include a specification of the subset of the UML metamodel that may be used when defining models based on the profile. The entire UML metamodel may be included in this subset. In addition, a profile may include one or more of the following:

- *Standard extensions beyond those specified by the identified subset of the UML metamodel. A standard extension is an instance of the* UML Stereotype, Tagged Value *or* Constraint *metaclasses.*
- *Semantics, beyond those supplied by the specified subset of the UML metamodel, defined using natural language.*
- *Well-formedness rules, beyond those supplied by the specified subset of the UML metamodel, expressed as* Constraints *written in the Object Constraint Language (OCL).*
- *Common model elements, which are pre-defined instances of UML metamodel elements. The definitions of common model elements may use the standard extensions defined by the profile, and are constrained by the formal and informal semantics defined by the profile.*
- *UML metamodel extensions created by defining new metaclasses using the Meta Object Facility[6]. UML metamodel extensions should be introduced only when the standard extension mechanisms can not be used to accomplish the desired result, since their use may prevent some tools from reading and writing the resulting models.*

2.5 Specification Scope

This specification contains two major components, a UML profile and a UML descriptor. The UML profile supports the capture of semantics expressible with EJB and the subset of the Java language used in the construction of EJBs[7], including the relationships between logical and physical Java constructs and UML model elements, and the forward and reverse engineering transformations between UML model elements and Java artifacts. It is defined in section 3. The UML descriptor contains a UML model of the Java and EJB-based artifacts stored within an EJB-JAR. It is defined in section 4.

[5] OMG document ad/99-03-11. This definition was created to support the solicitation of proposals for extensions to UML, and is considered the current working definition. The UML 1.4 RTF is developing a formal definition of the mechanisms used to specify and apply profiles.

[6] Recent developments on mapping between UML metamodel extensions and profiles suggests that profiles may be specified or accompanied by metamodel extensions in the future. See OMG document number ad/99-09-05 for more information.

[7] Because the EJB architecture is built on a foundation of general Java language constructs and packages, the mapping from EJB to UML defined by this specification is built on a mapping of general Java language constructs to UML. This specification was developed without waiting for the standardization of a general UML Profile for Java because of time to market considerations. It therefore defines UML mappings for the subset of Java language constructs used by the EJB architecture. This material will constrain any general UML Profile for Java developed in the future.

3 UML Profile

This section describes the UML profile component of this specification. The profile defines standard UML extensions that combine and/or refine existing UML constructs to create a dialect that can be used to describe EJB-based artifacts in a design or implementation model.

An EJB Design Model contains class diagrams that describe the interfaces and optionally the implementation classes of an enterprise bean. Because of the strong separation between specification and implementation in the EJB architecture, there are two different views of an EJB that can be modeled: an external view and an internal view.

The external view defines an EJB as seen by its clients. In this view, an EJB is modeled as a pair of stereotyped UML Classes representing the EJB Home and Remote Interfaces. The implementation of the EJB is not described by this view.

The internal view defines an EJB as seen by its developers, and describes its composition. The elements that comprise the internal view of an EJB are modeled using a stereotyped UML Subsystem. The specification section of the subsystem contains stereotyped UML Classes that describe the EJB Home and Remote Interfaces. The realization section contains stereotyped UML Classes and other model elements that describe the implementation of the EJB.

An EJB Implementation Model contains component diagrams describing the physical artifacts in which the logical elements of an EJB reside. These include Java class and resource files, and EJB-JARs that physically contain and organize the Java class and resource files. The profile uses Stereotypes of the UML Component construct to represent Java class and resource files and EJB-JARs.

3.1 Definition Of Terms

Term	**Definition**
Java Primitive Type	A Java Primitive Type is one of the primitive types defined by the Java Language Specification, i.e., boolean, byte, short, int, long, char, float and double.
Java Reference Type	A Java Reference Type is a class type, interface type, array type or the special null type, as defined by the Java Language Specification. An object is an instance of a Java Reference Type.
Java Collection Type	A Java Collection Type is either one of the Java Reference Types, Bitset, Collection, Iterator or Map, in the java.util package, or a Java Reference Type that realizes or specializes one of them.
Java Boolean Collection Type	A Java Boolean Collection Type is either the Java Class, java.util.Bitset, or a Java Reference Type that specializes it. A Java Boolean Collection Type is the only Java Collection Type whose element type is known at compile time.
Java Primitive Class	A Java Primitive Class is a Java Class that wraps a Java Primitive Type, i.e., Boolean, Byte, Short, Integer, Long, Float, Double and String, in the java.lang package.
Component	A Component is a UML construct that represents an implementation artifact, such as a file or archive. The term Component is used in this specification to refer to a physical artifact, such as an EJB-JAR or a Java Class File.
EJB Enterprise Bean	An EJB Enterprise Bean is a logical and physical construct defined by the EJB architecture. It is modeled logically as stereotyped Classes or a stereotyped Subsystem, and physically as stereotyped Components that represent the physical artifacts in which the logical elements reside.
Operation	In UML, an Operation is a declaration of behavior.
Method	In UML, a Method is an implementation of an Operation. The EJB architecture uses the term Method to describe a declaration of behavior. In other words, it uses the term Method to describe what UML calls an Operation. In this specification, the term Method is used as defined by the EJB Specification when describing EJB artifacts. All other usages of the terms Operation and Method are according to the UML terminology.

Table 3-1. Definition Of Terms

3.2 Supported Meta-Model Elements

This profile supports the UML Foundation and Model Management packages.

The Foundation package is the infrastructure that specifies the static structure of models. It contains the Core, Extension Mechanisms, and DataTypes packages. The Core package specifies the architectural backbone of the UML. The Extension Mechanisms package specifies how model elements are customized and extended with new semantics. The DataTypes package defines basic data structures for the language.

The Model Management package is dependent on the Foundation package. It specifies constructs used to organize model elements, including models, packages, and subsystems.

3.3 Meta-Model Extensions

This profile does not define any meta-model extensions.

3.4 Pre-Defined Common Model Elements

The common model elements pre-defined by this profile include UML DataTypes that represent the Java Primitive Types[8] and the types null and void, and model elements describing the Java language constructs defined by the Java Package named javax.ejb, and by all of the Java Packages on which javax.ejb depends, either directly or indirectly[9]. The definitions of the UML DataTypes reside in the UML Package named java::lang. The definitions of the other pre-defined common model elements can be produced by reverse engineering the Java language constructs contained in the Java Packages listed above using the mappings defined in this profile, and reside in the UML Packages produced by the reverse engineering process. See section 6.5.2, Reverse Engineering, for a description of the reverse engineering process.

3.5 Standard Extensions

This section summarizes the standard extensions, i.e., the Stereotypes and Tagged Values, introduced by this profile.

The standard extensions are defined in UML Packages defined by the profile, including java::lang, java::util, java::util::jar and javax::ejb. These Packages are introduced by the pre-defined common model elements described in the preceding section. Note that the fully qualified name of a model element introduced by this profile, e.g., java::lang::JavaStatic, must be used when the unqualified name, e.g., JavaStatic, does not uniquely identify the model element.

3.5.1 Stereotypes

The following table summarizes the Stereotypes introduced by this profile. It does not list the standard Stereotypes defined by UML that are used by this profile, such as the standard «file» Stereotype, used for a Component that represents a file.

[8] The standard UML DataTypes, Boolean and Integer, have semantics that are similar, but not identical, to the semantics of the Java Primitive Types, boolean and int, respectively, and are not used by the profile.

[9] Many of the Java language constructs defined by the EJB architecture, including the methods ejbLoad, ejbStore, ejbRemove, ejbActivate, ejbPassivate, setSessionContext, afterBegin, beforeCompletion, afterCompletion, setEntityContext and unsetEntityContext, and the exception classes RemoveException, ObjectNotFoundException and DuplicateKeyException, are described by these common model elements, and therefore are not explicitly defined in the body of this specification.

3.5.1.1 Java Design Model

The following Stereotypes are defined in package java::lang.

Stereotype	Applies To	Definition
«JavaInterface»	Class	A Stereotype indicating that the Class represents a Java Interface. Specializes the standard UML Stereotype «type».

Table 3-2. Java Design Model Stereotypes

3.5.1.2 EJB Design Model

The following Stereotypes are defined in package javax::ejb.

3.5.1.2.1 External View

Stereotype	Applies To	Definition
«EJBCreateMethod»	Operation	Specializes «EJBHomeMethod». Indicates that the Operation represents an EJB Create Method.
«EJBFinderMethod»	Operation	Specializes «EJBHomeMethod». Indicates that the Operation represents an EJB Finder Method.
«EJBRemoteMethod»	Operation	Indicates that the Operation represents an EJB Remote Method.
«EJBRemoteInterface»	Class	Specializes the standard UML Stereotype «type». Indicates that the UML Class represents an EJB Remote Interface.
«EJBHomeInterface»	Class	An abstract Stereotype indicating that the UML Class represents an EJB Home Interface. Specializes the standard UML Stereotype «type».
«EJBSessionHomeInterface»	Class	Indicates that the Class represents an EJB Session Home. Specializes the Stereotype «HomeInterface».
«EJBEntityHomeInterface»	Class	Indicates that the Class represents an EJB Entity Home. Specializes the Stereotype «HomeInterface».
«EJBPrimaryKey»	Usage	Indicates that the supplier of the Usage represents the EJB Primary Key Class for the EJB Entity Home represented by the client.

Table 3-3. EJB Design Model Stereotypes - External View

3.5.1.2.2 Internal View

Stereotype	Applies To	Definition
«EJBCmpField»	Attribute	Indicates that the Attribute represents a container-managed field for an EJB Entity Bean with container-managed persistence
«EJBPrimaryKeyField»	Attribute	Specializes «EJBCmpField». Indicates that the Attribute is the primary key field for an EJB Entity Bean with container-managed persistence.
«EJBRealizeHome»	Abstraction	Indicates that the supplier of the Abstraction represents an EJB Home Interface for the EJB Implementation Class represented by the client.
«EJBRealizeRemote»	Abstraction	Indicates that the supplier of the Abstraction represents an EJB Remote Interface for the EJB Implementation Class represented by the client.
«EJBImplementation»	Class	Specializes the standard UML Stereotype «implementationClass». Indicates that the Class describes an EJB Implementation Class, distinguishing it from other Classes that may appear within a UML Subsystem that represents an EJB Enterprise Bean.
«EJBEnterpriseBean»	Subsystem	An abstract Stereotype indicating that the Subsystem represents an EJB Enterprise Bean.
«EJBSessionBean»	Subsystem	Indicates that the Subsystem represents an EJB Session Bean. Specializes «EJBEnterpriseBean».
«EJBEntityBean»	Subsystem	Indicates that the Subsystem represents an EJB Entity Bean. Specializes «EJBEnterpriseBean».
«EJBReference»	Association	A Stereotype indicating that the navigable end of the UML Association represents a referenced EJB Enterprise Bean.
«EJBAccess»	Association	A Stereotype indicating that the UML Association defines a security role name relationship between a UML Actor and an «EJBEnterpriseBean».

Table 3-4. EJB Design Model Stereotypes - Internal View

3.5.1.3 Java ImplementationModel

The following Stereotypes are defined in package java::lang.

Stereotype	Applies To	Definition
«JavaClassFile»	Component	Specializes the standard UML Stereotype «file». Indicates that the Component describes a Java Class File.

Table 3-5. Java Implementation Model Stereotypes In Package java::lang

The following Stereotypes are defined in package java::util::jar.

Stereotype	Applies To	Definition
«JavaArchiveFile»	Package	Indicates that the Package represents a JAR.

Table 3-6. Java Implementation Model Stereotypes In Package java::util::jar

3.5.1.4 EJB Implementation Model

The following Stereotypes are defined in package javax::ejb.

Stereotype	Applies To	Definition
«EJB-JAR»	Package	Specializes the Stereotype «JavaArchiveFile». Indicates that the Package represents an EJB-JAR.

«EJBDescriptor»	Component	Specializes the standard Stereotype «file». Indicates that the Component represents an EJB Deployment Descriptor.
«EJBClientJAR»	Usage	Indicates that the client of the Usage represents an ejb-client-jar for the EJB-JAR represented by the supplier of the Usage.

Table 3-7. EJB Implementation Model Stereotypes

3.5.2 Tagged Values

The following table summarizes the Tagged Values introduced by this profile. It does not list the standard Tagged Values defined by UML that are used by this profile, such as the standard Tagged Value, persistence, used to indicate whether or not a Classifier is persistent.

3.5.2.1 Java Design Model

Tagged Value	**Applies To**	**Definition**
JavaStrictfp	Class	A Boolean value indicating whether or not the Java Class is FP-strict
JavaStatic	Class or Interface	A Boolean value indicating whether or not the Java Class or Interface is static.
JavaVolatile	Attribute or AssociationEnd	A Boolean value indicating whether or not the Java Field is volatile.
JavaDimensions	Attribute or AssociationEnd	An Integer indicating the number of array dimensions declared by the Java Field.
JavaCollection	Attribute or AssociationEnd	A String containing the name of the Java Collection Type used to implement an Attribute or AssociationEnd with complex multiplicity.
JavaNative	Operation	A Boolean value indicating whether or not the Java Method is native.
JavaThrows	Operation	A comma-delimited list of names of Java Exception Classes.
JavaFinal	Parameter	A Boolean value indicating whether or not the Parameter is final.
JavaDimensions	Parameter	An Integer indicating the number of array dimensions declared by the Parameter.

Table 3-8. Java Design Model Tagged Values

3.5.2.2 EJB Design Model

3.5.2.2.1 External View

Tagged Value	**Applies To**	**Definition**
EJBSessionType	Class «EJBSessionHomeInterface»	Stateful or Stateless. Indicates whether or not the EJB Session Bean maintains state.

Table 3-9. EJB Design Model Tagged Values - External View

3.5.2.2.2 Internal View

Tagged Value	Applies To	Definition
EJBRoleNames	Operation	A comma-delimited list of Strings, designating the security roles that may invoke the Operation.
EJBTransAttribute	Operation	An enumeration with values Not Supported, Supports, Required, RequiresNew, Mandatory, or Never. Defines the transaction management policy for the Operation.
EJBEnvEntries	Subsystem «EJBEnterpriseBean»	A comma-delimited list of tuples, designating the environment entries used by the EJB Enterprise Bean, of the form <name, type, value>.
EJBNameInJAR	Subsystem «EJBEnterpriseBean»	The name used for the EJB Enterprise Bean in the EJB-JAR. Defaults to the name of the EJB Remote Interface.
EJBReferences	Subsystem «EJBEnterpriseBean»	A comma-delimited list of tuples, designating the other EJB Enterprise Beans referenced by the EJB Enterprise Bean, of the form <name, type, home, remote>.
EJBResources	Subsystem «EJBEnterpriseBean»	A comma-delimited list of tuples, designating the resource factories used by the EJB Enterprise Bean, of the form <name, type, auth>.
EJBSecurityRoles	Subsystem «EJBEnterpriseBean»	A comma-delimited list of tuples, designating the role names that may invoke ALL operations on the EJB Enterprise Bean, of the form <name, link>.
EJBTransType	Subsystem «EJBSessionBean»	An enumeration with values Bean or Container. Indicates whether the transactions of the EJB Session Bean are managed by the EJB Session Bean or by its container, respectively.
EJBPersistenceType	Subsystem «EJBEntityBean»	An enumeration with values Bean or Container. Indicates whether the persistence of the EJB Entity Bean is managed by the EJB Entity Bean or by its container, respectively.
EJBReentrant	Subsystem «EJBEntityBean»	A Boolean value indicating whether or not the EJB Entity Bean can be called reentrantly.

Table 3-10. EJB Design Model Tagged Values - Internal View

3.6 Semantics

This section defines a mapping from Java source files to UML model elements. It also describes UML model elements and standard extensions that represent EJBs in design and implementation models, and supports a mapping of those constructs to EJB-based artifacts.

3.6.1 Java Design Model

This section defines the mechanisms used to produce a design model of Java language constructs defined by the Java Language Specification. It enumerates the subset of Java logical constructs supported by this profile, and for each construct defines a mapping to UML model elements, including abstract syntax and applicable Stereotypes, Tagged Values and Constraints. This mapping does not attempt to describe any Java logical constructs that can not be used in the implementation of EJBs, nor does it attempt to describe any Java logical constructs whose representation in UML requires the use of Java language expressions.

Strings used as names for Java logical constructs must be sequences of Java Letters and Java Digits, as defined by the Java Language Specification, the first of which must be a Java Letter. The grammar used to describe the syntax of declarations for Java logical constructs is based on the following conventions: *[x]* denotes zero or one occurrences of *x*; *{x}* denotes zero or more occurrences of *x*; *x* | *y* means one of either *x* or *y*.

3.6.1.1 Java Package

3.6.1.1.1 Syntax

The Java Package construct is declared by the following Java syntax.

```
package PackageName ;
```

3.6.1.1.2 Mapping

A Java Package maps to a UML Package. The simple name of the UML Package is the simple name of the Java Package. A hierarchy of Java Packages maps to a hierarchy of UML Packages. The elements of the declaration map as follows.

- *PackageName* is the fully-qualified[10] name of the Java Package. The fully-qualified name of a top level Java Package is its simple name. The fully-qualified name of a Java Package contained by another Java Package is the fully-qualified name of the containing Java Package, followed by ".", followed by the simple name of the Java Package. The fully-qualified name of a Java Package maps to the fully-qualified name of the corresponding UML Package by replacing every occurrence of "." with "::".

[10] Unless explicitly stated otherwise, all Java Package names described in this document are fully-qualified.

3.6.1.2 Java Single Type Import

3.6.1.2.1 Syntax

The Java Single Type Import construct is declared by the following Java syntax. When the name of a Java Class or Interface appears in the declaration of another construct, it must be fully-qualified, unless the referenced Java Class or Interface is imported by the Java Class File that contains the declaration, in which case, the name must be properly qualified in the context of the import. A Java Class or Interface is imported either by a Java Single Type Import, or by a Java Type Import On Demand. A Java Class or Interface name referenced in the declaration of another construct is properly qualified in the context of a Java Single Type Import that names the referenced Java Class or Interface.

```
import TypeName ;
```

3.6.1.2.2 Mapping

A Java Single Type Import maps to a UML Permission, stereotyped as «access», whose client represents the Java Class File that contains the declaration, and whose supplier represents the imported Java Class or Interface. The elements of the declaration map as follows.

- *TypeName* is the fully-qualified[11] name of the imported Java Class or Interface, and maps to the fully-qualified name of the supplier of the UML Permission. The fully-qualified name of a Java Class or Interface is the fully-qualified name of the containing Java Class, Interface or Package, followed by ".", followed by the simple name of the Java Class or Interface. The simple name of a Java Class or Interface is the simple name of the corresponding model element. The fully-qualified name of a Java Class or Interface maps to the fully-qualified name of the corresponding model element by replacing every occurrence of "." with "::".

3.6.1.2.3 Constraints

These Constraints apply to a UML Permission, stereotyped as «access», that represents a Java Single Type Import.

1) The client of the Permission must represent a Java Class File.
2) The supplier of the Permission must represent a Java Class or Interface.

3.6.1.3 Java Type Import On Demand

3.6.1.3.1 Syntax

The Java Type Import On Demand construct is declared by the following Java syntax.

```
import PackageOrTypeName . * ;
```

A Java Class or Interface name referenced in the declaration of another construct is properly qualified in the context of a Java Type Import On Demand that names a Java Class, Interface or Package directly or indirectly containing the referenced Java Class or Interface if the name is fully-qualified when appended to "." appended to the imported name.

3.6.1.3.2 Mapping

A Java Type Import On Demand maps to a UML Permission stereotyped as «access», whose client represents the Java Class File that contains the declaration, and whose supplier represents the imported Java Class, Interface or Package. The elements of the declaration map as follows.

- *PackageOrTypeName* is the name of the imported Java Class, Interface or Package, and maps to the name of the supplier of the UML Permission.

3.6.1.3.3 Constraints

These Constraints apply to a UML Permission, stereotyped as «access», that represents a Java Type Import On Demand.

1) The client of the Permission must represent a Java Class File.
2) The supplier of the Permission must represent a Java Package, Class or Interface.

[11] Unless explicitly stated otherwise, all Java Class or Interface names described in this document are fully-qualified.

3.6.1.4 Java Class

3.6.1.4.1 Syntax

The Java Class construct is declared by the following Java syntax.

```
[ visibility ] [ modifiers ] class Identifier [ extends ClassType ] [ implements TypeList ] {
    { ClassMember }
    { InterfaceMember }
    { FieldMember }
    { MethodMember }
}
visibility = public | protected | private
modifiers = [ abstract ] [ static ] [ final ] [ stricfp ]
TypeList = InterfaceType | TypeList , InterfaceType
```

The Java Class java.lang.Throwable and its direct and indirect subclasses are called Java Exception Classes.

3.6.1.4.2 Mapping

A Java Class maps to a UML Class contained directly or indirectly by a model element that represents a Java Package, or resident in a model element that represents a Java Class File. The UML Class is an implementation class and may optionally carry the standard UML Stereotype «implementation». The elements of the declaration map as follows.

- *visibility* maps to the visibility property of the UML ElementOwnership that references the UML Class.
- *modifiers* maps as follows.
  - *abstract* maps to a value of true, if present, or false, if absent, for the isAbstract property of the UML Class.
  - *static* maps to a value of true, if present, or false, if absent, for the JavaStatic tag on the UML Class.
  - *final* maps to a value of true, if present, or false, if absent, for the isLeaf property of the UML Class.
  - *stricfp* maps to a value of true, if present, or false, if absent, for the JavaStricfp tag on the UML Class.
- *Identifier* maps to the name of the UML Class.
- *ClassType* maps to the name of a model element specialized by the UML Class.
- *InterfaceType* maps to the name of a model element realized by the UML Class.
- *ClassMember* maps to a model element, contained by the UML Class, that represents a Java Class.
- *InterfaceMember* maps to a model element, contained by the UML Class, that represents a Java Interface.
- *FieldMember* maps to a model element, contained by the UML Class, that represents a Java Field.
- *MethodMember* maps to a model element, contained by the UML Class, that represents a Java Method.

3.6.1.4.3 Tagged Values

These Tagged Values apply to a UML Class that represents a Java Class.

Tag	Value
JavaStrictfp	A Boolean value indicating whether or not the Java Class is FP-strict
JavaStatic	A Boolean value indicating whether or not the Java Class is static.

Table 3-11. Java Class Tagged Values

3.6.1.4.4 Constraints

These Constraints apply to a UML Class that represents a Java Class.

1) The Class may not carry the standard UML Stereotype «type».
2) The value of the JavaStatic tag must be either true or false.
3) The value of the JavaStricfp tag must be either true or false.
4) The Class must be directly or indirectly contained by a model element that represents a Java Package, or it must reside in a model element that represents a Java Class File.
5) The Class may specialize at most one other model element, which must represent a Java Class.
6) The Class may realize any number of model elements, which must represent Java Interfaces.
7) The Class may not be public unless directly contained by a model element that represents a Java Class, Interface or Package.
8) The Class may not be protected or private unless directly contained by a model element that represents a Java Class.
9) The Class may not be static unless directly contained by a model element that represents a Java Class or Interface.
10) The Class must be static and public if directly contained by a model element that represents a Java Interface.
11) The Class may not contain any model elements that represent Java Interfaces unless it is static, or is directly contained by a model element that represents a Java Package.
12) The Class may not be both final and abstract.
13) The Class may not specialize a model element that represents a Java Class that is final.
14) The Class may not have the same simple name as any enclosing model elements that represent Java Classes or Interfaces.
15) The Class must be abstract if any of its Operations, either contained or inherited, are abstract and are not referenced by the specification property of a UML Method.
16) If the Class does not contain an Operation that represents a Java Constructor Method, then the inherited Class, if any, must not contain an Operation that represents a Java Constructor Method, unless that Operation has no in Parameters.

3.6.1.5 Java Interface

3.6.1.5.1 Syntax

The Java Interface construct is declared by the following Java syntax.

```
[ visibility ] [ modifiers ] interface Identifier [ extends ExtendsInterfaces ] {
    { ClassMember }
    { InterfaceMember }
    { FieldMember }
    { MethodMember }
}
visibility = public | private | protected
modifiers = [ abstract ] [ static ]
ExtendsInterfaces = InterfaceType | ExtendsInterfaces , InterfaceType
```

3.6.1.5.2 Mapping

A Java Interface maps to a UML Interface, or UML Class stereotyped as «JavaInterface»[12]. The UML Class is contained directly or indirectly by a model element that represents a Java Package, or is resident in a model element that represents a Java Class File. The elements of the declaration map as follows.

- *visibility* maps to the visibility property of the UML ElementOwnership that references the declared model element.
- *modifiers* maps as follows.
  - *abstract* maps to the value of the isAbstract property of the declared model element, which is always true.
  - *static* maps to a value of true, if present, or false, if absent, for the JavaStatic tag on the declared model element.
- *Identifier* maps to the name of the declared model element.
- *InterfaceType* maps to the name of a model element specialized by the declared model element.
- *ClassMember* maps to a model element, contained by the declared model element, that represents a Java Class.
- *InterfaceMember* maps to a model element, contained by the declared model element, that represents a Java Interface.
- *FieldMember* maps to a model element, contained by the declared model element, that represents a Java Field.
- *MethodMember* maps to a model element, contained by the declared model element, that represents a Java Method.

3.6.1.5.2.1 Stereotypes

These Stereotypes apply to a UML Class that represents a Java Interface

Stereotype	Definition
«JavaInterface»	An Stereotype indicating that the Class represents a Java Interface.

Table 3-12. Java Interface Stereotypes

3.6.1.5.3 Tagged Values

These Tagged Values apply to a UML Interface, or to a UML Class stereotyped as «type», that represents a Java Interface.

Tag	Value
JavaStatic	A Boolean value indicating whether or not the Java Interface is static.

Table 3-13. Java Interface Tagged Values

3.6.1.5.4 Constraints

These Constraints apply to a UML Interface, or to a UML Class stereotyped as «JavaInterface», that represents a Java Interface.

1) The model element must be directly or indirectly contained by a model element that represents a Java Package, or it must reside in a model element that represents a Java Class File.
2) The value of the JavaStatic tag must be either true or false.
3) The model element may specialize any number of model elements, which must represent Java Interfaces.
4) The model element must be abstract, regardless of the presence or absence of the modifier.
5) The model element may not be protected or private unless directly contained by a model element that represents a Java Class.
6) The model element may not be static unless directly contained by a model element that represents a Java Class or Interface.

[12] Because a UML Interface can not have Attributes or AssociationEnds or contain other model elements, a UML Class must be used to represent a Java Interface that has Java Field, Class or Interface members. The Stereotype «JavaInterface» that inherits from the UML Stereotype «type» denotes that the Class is a specification construct, not an implementation construct.

7) The model element must be static if directly contained by a model element that represents a Java Class or Interface.
8) The model element must be public if directly contained by a model element that represents a Java Interface.
9) The model element may not contain model elements that represent Java Fields, Methods, Classes or Interfaces and are not public.
10) The model element may not have the same simple name as any enclosing model elements that represent Java Classes or Interfaces.

3.6.1.6 Java Field

3.6.1.6.1 Syntax

One or more Java Fields may be declared within a Java Class or Interface declaration by the following Java syntax.

```
[ visibility ] [ modifiers ] Type { [] } IdentifierList ;
visibility = public | protected | private
modifiers = [ static ] [ final ] [ transient ] [ volatile ]
IdentifierList = IdentifierDeclarator | IdentifierList , IdentifierDeclarator
IdentifierDeclarator = Identifier { [] } [ = InitialValue ] ;
```

3.6.1.6.2 Mapping

A Java Field maps to a UML Attribute or AssociationEnd[13] contained by a model element that represents a Java Class or Interface. If *Type* is a Java Primitive Type,or if the declaration is directly enclosed by a Java Interface declaration, then the Java Field maps to a UML Attribute. The elements of the declaration map as follows.

- *visibility* maps to the visibility property of the declared model element.
- *modifiers* maps as follows.
  - *static* maps to a value of classifier, if present, or instance, if absent, for the ownerScope property, in the case of a UML Attribute, or for the targetScope property of the other end, in the case of a UML AssociationEnd.
  - *final* maps to a value of frozen, if present, or changeable, if absent, for the changeability property of the declared model element.
  - *volatile* maps to a value of true, if present, or false, if absent, for the JavaVolatile tag on the declared model element.
  - *transient* maps to a value of transient, if present, or persistent, if absent, for the persistence tag on the declared model element in the case of a UML Attribute and to the UML Association that owns the UML AssociationEnd in the case of a UML AssociationEnd
- *Type* maps as follows.
  - If *Type* is a Java Boolean Collection Type, it maps to the value of the JavaCollection tag on the declared model element. In addition, the type of the declared model element represents the Java Primitive Type, boolean.
  - If *Type* is a Java Collection Type, it maps to the value of the JavaCollection tag on the declared model element. In addition, the type of the declared model element represents a Java Reference Type other than null. If no Java Reference Type is specified, it represents the Java Class, java.lang.Object[14].
  - Otherwise, Type maps to the name of the Type of the declared model element.
- *Identifier* maps to the name of the declared model element; The number of pairs of square brackets appended to *Type* plus the number of pairs of square brackets appended to *Identifier* maps to the value of the JavaDimensions tag on the declared model element.
- *InitialValue* maps to the initialValue property of the declared model element.

---

[13] Because the distinction articulated by the UML between these two constructs relies on information that is not captured by the declaration of a Java Field, a deterministic mapping can not be specified, except in the circumstances indicated by the text. Extensions of this mapping, such as a mapping for EJB, may be able to use other information sources, such as an EJB Deployment Descriptor, to specify a deterministic mapping.

[14] The declaration of a Java Field does not provide enough information to specify the type. The text makes java.lang.Object the default, but allows for the specification of a different type if additional information is available from another source, such as an EJB Deployment Descriptor.

3.6.1.6.3 Tagged Values

These Tagged Values apply to a UML Attribute or UML AssociationEnd that represents a Java Field.

Tag	Value
JavaVolatile	A Boolean value indicating whether or not the Java Field is volatile.
JavaDimensions	An Integer indicating the number of array dimensions declared by the Java Field.
JavaCollection	A String containing the name of the Java Collection Type used to implement an Attribute or AssociationEnd with complex multiplicity.

Table 3-14. Java Field Tagged Values

3.6.1.6.4 Constraints

3.6.1.6.4.1 Constraints For A UML Attribute Or UML AssociationEnd

These Constraints apply to a UML Attribute or UML AssociationEnd that represents a Java Field.

1) The model element may not be both volatile and final.
2) The value of the JavaVolatile tag must be either true or false.
3) The value of the JavaCollection tag, if specified, must be the name of a Java Collection Type.
4) The value of the JavaDimensions tag, if specified, must be an Integer.
5) If the value of the JavaDimensions tag is zero and the JavaCollection tag is not specified, then:
   a) The value of the upper bound of the multiplicity property must be exactly one.
   b) The value of the lower bound of the multiplicity property must be exactly one, or zero if the type represents a Java Reference Type.

3.6.1.6.4.2 Additional Constraints For A UML Attribute

These additional Constraints apply to a UML Attribute that represents a Java Field.

1) The type must represent a Java Primitive or Reference Type other than null.
2) The Attribute must have an initialValue if contained by a model element that represents a Java Interface.
3) The Attribute may not be protected, private, transient or volatile if contained by a model element that represents a Java Interface.
4) The Attribute must be public, static, and final if contained by a model element that represents a Java Interface.

3.6.1.6.4.3 Additional Constraints For A UML AssociationEnd

These additional Constraints apply to a UML AssociationEnd that represents a Java Field.

1) The type must represent a Java Reference Type other than null.
2) The value of the isNavigable property must be true.

3.6.1.7 *Java Method*

3.6.1.7.1 Syntax

One or more Java Methods may be declared within a Java Class or Interface declaration by the following Java syntax.

[visibility] [modifiers] Type { [] } [Identifier] (*[* Parameter*List]*) *{* [] *} [* throws Class*TypeList] MethodBody*

visibility = public | protected | private

modifiers = *[* abstract *] [* static *] [* final *] [* synchronized *] [* native *]*

Parameter*List* = Parameter | Parameter*List* , Parameter

Parameter = *[final]* Parameter*Type { [] }* Parameter*Declarator { [] }*

MethodBody = { … } | ;

A Java Method whose name is the simple name of the Java Class in which it is declared is called a Java Constructor Method.

3.6.1.7.2 Mapping

A Java Method maps to a UML Operation contained by a model element that represents a Java Class or Interface. The elements of the declaration map as follows.

- *visibility* maps to the visibility property of the UML Operation.
- *modifiers* maps as follows.
  – *abstract* maps to a value of true, if present, or false, if absent, for the isAbstract property of the UML Operation.
  – *static* maps to a value of classifier, if present, or instance, if absent, for the ownerScope property of the UML Operation.
  – *final* maps to a value of true, if present, or false, if absent, for the isLeaf property of the UML Operation.
  – *native* maps to a value of true, if present, or false, if absent, for the JavaNative tag on the UML Operation.
  – *synchronized* maps to a value of guarded, if present, for the concurrency property of the UML Operation.
- *Type* maps to the name of the Type of a UML Parameter that is referenced by the parameter association of the UML Operation, and that has no name and a kind property value of return. The number of pairs of square brackets appended to *Type* plus the number of pairs of square brackets appended to the parameter list maps to the value of the JavaDimensions tag of the UML Parameter.
- *Identifier* maps to the name of the UML Operation. If not specified, the name of the UML Operation is the simple name of the model element that contains the Operation.
- *Parameter* maps to a UML Parameter that is referenced by the parameter property of the UML Operation, and that has a kind property value of in, as follows.
  – *final* maps to a value of true, if present, or false, if absent, for the JavaFinal tag on the UML Parameter.
  – *ParameterType* maps to the name of the Type of the UML Parameter.
  – *ParameterDeclarator* maps to the name of the UML Parameter. The number of pairs of square brackets appended to *ParameterType* plus the number of pairs of square brackets appended to *ParameterDeclarator* maps to the value of the JavaDimensions tag on the UML Parameter.
- If the values of the isAbstract property and the JavaNative tag on the UML Operation are false, *MethodBody* maps to the value of the body property of a UML Method whose specification property references the UML Operation.
- *ClassTypeList* maps to the value of the JavaThrows tag of the UML Operation.

3.6.1.7.3 Tagged Values

These Tagged Values apply to a UML Operation that represents a Java Method.

Tag	Value
JavaNative	A Boolean value indicating whether or not the Java Method is native.
JavaThrows	A comma-delimited list of names of Java Exception Classes.

Table 3-15. Java Method Tagged Values

These Tagged Values apply to an in Parameter of a UML Operation that represents a Java Method.

Tag	Value
JavaFinal	A Boolean value indicating whether or not the Parameter is final.
JavaDimensions	An Integer indicating the number of array dimensions declared by the Parameter.

Table 3-16. Java Method Parameter Tagged Values

3.6.1.7.4 Constraints

These Constraints apply to a UML Operation that represents a Java Method.

1) All of the Parameters referenced by the parameter property must have an no defaultValue property.
2) The Operation must have a single unnamed Parameter with a kind property value of return, and a type property value that references a Classifier that describes a Java Primitive Type, a Java Reference Type other than null, or void.
3) All of the other Parameters referenced by the parameter property must have names, and must have a kind property value of in.
4) The Operation must not be referenced by the specification property of a UML Method if it is abstract or native.
5) The body property of a UML Method whose specification property references the Operation must be a UML Expression whose language property has the value "Java Language Specification, Second Edition", and whose body property has a value that begins with the character "{" and ends with the character "}".
6) The value of the JavaNative tag must be either true or false.
7) The value of the JavaThrows tag must be a comma-delimited list of names of Java Exception Classes.
8) The value of the JavaFinal tag on a Parameter of the Operation must be either true or false.
9) The value of the JavaDimensions tag, if specified, on a Parameter of the Operation must be an Integer.
10) The Operation may not be abstract unless it is contained by a model element that represents an abstract Java Class or a Java Interface.
11) The Operation may not be private, static, native or final if it is abstract.
12) The Operation must be final if it is private or if it is contained by a model element that represents a final Java Class.
13) The Operation must public and abstract if it is contained by a model element that represents a Java Interface.
14) The Operation may not be synchronized if it is contained by a model element that represents a Java Interface.
15) The Operation may not have a different ownerScope property value than an inherited Operation with the same signature.
16) The Operation must have the same return type, and must throw the same Java Exception Classes, subclasses of the same Java Exception Classes, or a subset of either, as any inherited Operations with the same name and Parameters.
17) The Operation may not have the same name and Parameters as an inherited Operation that is final.
18) The Operation must be public if it has the same name and Parameters as an inherited Operation that is public.
19) The Operation must be protected or public if it has the same name and Parameters as an inherited Operation that is protected.
20) The Operation may not be abstract, static, final, native, or synchronized if it represents a Java Constructor Method.

3.6.2 EJB Design Model

This section defines the mechanisms used to construct a design model of EJB-based artifacts. It enumerates the logical constructs defined by the EJB Specification, and for each construct defines a mapping to UML model elements, including abstract syntax and applicable Stereotypes, Tagged Values and Constraints. There are two views that can be modeled, an external view and an internal view.

All standard tags and Stereotype names defined by the EJB Design Model start with the prefix "javax.ejb.".

3.6.2.1 External View

This section describes the representation of logical constructs visible to the clients of an EJB Enterprise Bean. The constructs described in this section are included in the internal view, and may be extended there.

3.6.2.1.1 EJB Method

3.6.2.1.1.1 Syntax

An EJB Method is declared by a Java Method declaration within an EJB Home or Remote Interface. The EJB Method is an EJB Home Method, if declared within an EJB Home Interface, or an EJB Remote Method, if declared within an EJB Remote Interface.

3.6.2.1.1.2 Mapping

The mapping for an EJB Method extends the mapping for a Java Method as follows. The UML Operation that represents the Java Method is contained by a model element that represents an EJB Home or Remote Interface.

If the UML Operation is contained by a model element that represents an EJB Home Interface, then the EJB Method is an EJB Home Method. An EJB Home Method is an EJB Create Method, except in the case of an EJB Entity Bean, where it may be an EJB Finder Method. The UML Operation may be stereotyped more specifically as «EJBCreateMethod» or «EJBFinderMethod».

If the UML Operation is contained by a model element that represents an EJB Remote Interface, then the EJB Method is an EJB Remote Method, and the UML Operation may be stereotyped as «EJBRemoteMethod».

3.6.2.1.1.3 Stereotypes

These Stereotypes optionally apply to a UML Operation that represents an EJB Method.

Stereotype	Definition
«EJBCreateMethod»	Specializes «EJBHomeInterfaceMethod». Indicates that the Operation represents an EJB Create Method.
«EJBFinderMethod»	Specializes «EJBHomeInterfaceMethod». Indicates that the Operation represents an EJB Finder Method.
«EJBRemoteMethod»	Indicates that the Operation represents an EJB Remote Method.

Table 3-17. EJB Method Stereotypes

3.6.2.1.1.4 Constraints

3.6.2.1.1.4.1 CONSTRAINTS FOR AN EJB METHOD

These Constraints apply to a UML Operation that represents an EJB Method.

1) The types of the Parameters must be valid types for RMI/IIOP.
2) The value of the JavaThrows tag must contain the name of the Java Class, java.rmi.RemoteException.
3) The value of the JavaThrows tag must not contain the name of a Java Exception Class that subclasses the Java Class, java.rmi.RemoteException.
4) The Operation must have a concurrency property value of sequential.

3.6.2.1.1.4.2 ADDITIONAL CONSTRAINTS FOR AN EJB CREATE METHOD

These additional Constraints apply to a UML Operation that represents an EJB Create Method.

1) The Operation must have an ownerScope property value of classifier.
2) The name of the Operation must be "create".
3) The Operation may be stereotyped as «EJBCreateMethod».
4) The type of the return Parameter must be the model element that represents the EJB Remote Interface.
5) The value of the JavaThrows tag must contain the name of the Java Class, javax.ejb.CreateException.

3.6.2.1.1.4.3 ADDITIONAL CONSTRAINTS FOR AN EJB FINDER METHOD

These additional Constraints apply to a UML Operation that represents an EJB Finder Method.

1) The Operation must have an ownerScope property value of classifier.
2) The name of the Operation must begin with "find".
3) The Operation may be stereotyped as «EJBFinderMethod».
4) The type of the return Parameter must be either the model element that represents the EJB Remote Interface, or a model element that represents a Java Collection Type.
5) The value of the JavaThrows tag must contain the name of the Java Class, javax.ejb.FinderException.

3.6.2.1.1.4.3.1 Additional Constraints For An EJB Primary Key Finder Method

These additional Constraints apply to a UML Operation that represents an EJB Primary Key Finder Method.

1) The name of the Operation must be "findByPrimaryKey".
2) The Operation must have a single in Parameter whose type represents the EJB Primary Key Class.
3) The type of the return Parameter must represent the EJB Remote Interface of the EJB Enterprise Bean.

3.6.2.1.1.4.4 ADDITIONAL CONSTRAINTS FOR AN EJB REMOTE METHOD

These additional Constraints apply to a UML Operation that represents an EJB Remote Method.

1) The Operation must have an ownerScope property value of instance.
2) The Operation must have a visibility property value of public and an isLeaf property value of false.
3) The name must not start with "ejb" or conflict with the names of common model elements pre-defined by this profile (e.g., ejbCreate, ejbActivate, etc.).

3.6.2.1.2 EJB Remote Interface

3.6.2.1.2.1 Syntax

The declaration of an EJB Remote Interface extends the declaration of a Java Interface with EJB Deployment Descriptor elements for an EJB Enterprise Bean. The name of the EJB Remote Interface and the related EJB Home Interface are specified by remote and home elements in the entity or session element for the EJB Enterprise Bean. These elements have the following XML DTD syntax[15].

```
<!ELEMENT entity (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class, persistence-type, prim-key-class, reentrant, cmp-field*, primkey-field?, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
<!ELEMENT home (#PCDATA)>
<!ELEMENT remote (#PCDATA)>
<!ELEMENT session (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class, session-type, transaction-type, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
```

3.6.2.1.2.2 Mapping

The mapping for an EJB Remote Interface extends the mapping for a Java Interface as follows. The EJB Remote Interface maps to a UML Class[16] stereotyped as «EJBRemoteInterface». The elements used in the declaration map as follows.

- *home* maps to the name of the client of a UML Usage, stereotyped as «instantiate», whose supplier is the UML Class.
- *remote* maps to the name of the UML Class.

3.6.2.1.2.3 Stereotypes

These Stereotypes apply to a UML Class that represents an EJB Remote Interface.

Stereotype	Definition
«EJBRemoteInterface»	Specializes the standard UML Stereotype «type»[17]. Indicates that the UML Class represents an EJB Remote Interface.

Table 3-18. EJB Remote Interface Stereotypes

3.6.2.1.2.4 Constraints

These Constraints apply to a UML Class that represents an EJB Remote Interface.

1) The Class must specialize a model element that represents the Java Interface, javax.ejb.EJBObject.
2) All of the Operations contained by the Class must represent EJB Remote Methods.
3) Any realization of model elements by the Class is subject to the RMI/IIOP rules for Java Interface inheritance.
4) The Class must be the supplier of a UML Usage, stereotyped as «instantiate», whose client represents the EJB Home Interface of the same EJB Enterprise Bean.

[15] The syntax of an XML DTD is defined by XML 1.0, W3C Recommendation February 1998, and Namespaces, January 1999. The content of an element is case sensitive.

[16] Version 1.1 of the EJB Specification does not define a mechanism to expose state to the client of an EJB Enterprise Bean. This profile therefore does not define mappings for Attributes or AssociationEnds. It uses a UML Class to model EJB Home and Remote Interfaces, however, so that profile extensions can use Attributes and AssociationEnds to describe state exposed to the client by extensions to the EJB programming model.

[17] In the UML, a Type is used to specify a domain of Objects and applicable Operations, without defining their implementation. A type may contain Attributes and AssociationEnds that declare named pieces of state, but these declarations do not define the implementation of the state. In addition, a Type may not define Methods, i.e., implementations, for its Operations.

3.6.2.1.3 EJB Home Interface

3.6.2.1.3.1 Syntax

The declaration of an EJB Home Interface extends the declaration of a Java Interface with EJB Deployment Descriptor elements for an EJB Enterprise Bean. The name of the EJB Home Interface and the related EJB Remote Interface are specified by home and remote elements in the entity or session element for the EJB Enterprise Bean. These elements have the following XML DTD syntax.

```
<!ELEMENT entity (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class, persistence-type, prim-key-class, reentrant, cmp-field*, primkey-field?, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
<!ELEMENT home (#PCDATA)>
<!ELEMENT remote (#PCDATA)>
<!ELEMENT session (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class, session-type, transaction-type, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
```

3.6.2.1.3.2 Mapping

The mapping for an EJB Home Interface extends the mapping for a Java Interface as follows. The EJB Home Interface maps to a UML Class stereotyped as «EJBHomeInterface». The elements used in the declaration map as follows.

- *home* maps to the name of the UML Class.
- *remote* maps to the name of the supplier of a UML Usage, stereotyped as «instantiate», whose client is the UML Class.

3.6.2.1.3.3 Stereotypes

These Stereotypes apply to a UML Class that represents an EJB Home Interface.

Stereotype	Definition
«EJBHomeInterface»	Specializes the standard UML Stereotype «type». Indicates that the UML Class represents an EJB Home Interface.

Table 3-19. EJB Home Interface Stereotypes

3.6.2.1.3.4 Constraints

These Constraints apply to a UML Class that represents an EJB Home Interface.

1) The Class must specialize a model element that represents the Java Interface, javax.ejb.EJBHome.
2) All of the Operations contained by the Class must represent EJB Home Methods.
3) Any realization of model elements by the Class is subject to the RMI/IIOP rules for Java Interface inheritance.
4) The Class must be the client of a UML Usage, stereotyped as «instantiate», whose supplier represents the EJB Home Interface of the same EJB Enterprise Bean.

3.6.2.1.4 EJB Session Home

3.6.2.1.4.1 Syntax

The declaration of an EJB Session Home extends the declaration of an EJB Home Interface with additional EJB Deployment Descriptor elements. Properties of the EJB Session Bean accessed through the EJB Session Home are specified by the session element. These elements have the following XML DTD syntax.

```
<!ELEMENT session-type (#PCDATA)>
<!ELEMENT session (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class, session-type, transaction-type, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
```

3.6.2.1.4.2 Mapping

The mapping for an EJB Session Home extends the mapping for an EJB Home Interface as follows. The UML Class that represents the EJB Home Interface is stereotyped as «EJBSessionHomeInterface». The elements used in the declaration map as follows.

- *session-type* maps to the value of the EJBSessionType tag on the UML Class.

3.6.2.1.4.3 Stereotypes

These Stereotypes apply to a UML Class that represents an EJB Session Home.

Stereotype	Definition
«EJBSessionHomeInterface»	Indicates that the Class represents an EJB Session Home. Specializes the Stereotype «EJBHomeInterface».

Table 3-20. EJB Session Home Stereotypes

3.6.2.1.4.4 Tagged Values

These Tagged Values apply to a UML Class, stereotyped as «EJBSessionHomeInterface», that represents an EJB Session Home.

Tag	Value
EJBSessionType	Stateful or Stateless. Indicates whether or not the EJB Session Bean maintains state.

Table 3-21. EJB Session Home Tagged Values

3.6.2.1.4.5 Constraints

These Constraints apply to a UML Class that represents an EJB Session Home.

1) The Class must not be tagged as persistent.
2) The value of the EJBSessionType tag must be either Stateful or Stateless.
3) The Class may not contain any Operations that represent EJB Finder Methods.
4) The Class must contain at least one Operation that represents an EJB Create Method.
5) If the value of the EJBSessionType tag is Stateless, then the Class must contain exactly one Operation that represents an EJB Create Method. The type of its return Parameter must be the supplier of a UML Usage, stereotyped as «instantiate», whose client is the Class.

3.6.2.1.5 EJB Entity Home

3.6.2.1.5.1 Syntax

The declaration of an EJB Entity Home extends the declaration of an EJB Home Interface with additional EJB Deployment Descriptor elements. The name of the EJB Primary Key Class used by the EJB Entity Home is specified by a prim-key-class element in the entity element for the EJB Entity Bean. These elements have the following XML DTD syntax.

```
<!ELEMENT entity (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class, persistence-type, prim-key-class, reentrant, cmp-field*, primkey-field?, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
<!ELEMENT prim-key-class (#PCDATA)>
```

3.6.2.1.5.2 Mapping

The mapping for an EJB Entity Home extends the mapping for an EJB Home Interface as follows. The UML Class that represents the EJB Home Interface is stereotyped as «EJBEntityHomeInterface». The elements used in the declaration map as follows.

- *prim-key-class* maps to the name of the supplier of a UML Usage, stereotyped as «EJBPrimaryKey», whose client is the UML Class.

3.6.2.1.5.3 Stereotypes

These Stereotypes apply to a UML Class that represents an EJB Entity Home.

Stereotype	Definition
«EJBEntityHomeInterface»	Indicates that the Class represents an EJB Entity Home. Specializes the Stereotype «EJBHomeInterface».

Table 3-22. EJB Entity Home Stereotypes

3.6.2.1.5.4 Constraints

These Constraints apply to a UML Class that represents an EJB Entity Home.

1) The Class must be tagged as persistent.
2) The Class must contain exactly one Operation that represents an EJB Primary Key Finder Method. The type of its return Parameter must be the supplier of a UML Usage, stereotyped as «instantiate», whose client is the Class.
3) The Class must be the client of a UML Usage, stereotyped as «EJBPrimaryKey», whose supplier represents an EJB Primary Key Class. The supplier must be the type of the in Parameter of the Operation that represents the EJB Primary Key Finder Method.

3.6.2.1.6 EJB Primary Key Class

3.6.2.1.6.1 Syntax

The declaration of an EJB Primary Key Class extends the declaration of a Java Class with EJB Deployment Descriptor elements for an EJB Entity Bean. The name of the EJB Entity Bean and the name of the EJB Primary Key Class are specified by the home and prim-key-class elements, respectively. These elements have the following XML DTD syntax.

```
<!ELEMENT entity (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class, persistence-type, prim-key-class, reentrant, cmp-field*, primkey-field?, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
<!ELEMENT home (#PCDATA)>
<!ELEMENT prim-key-class (#PCDATA)>
```

3.6.2.1.6.2 Mapping

The mapping for an EJB Primary Key Class extends the mapping for a Java Class. The elements used in the declaration map as follows.

- *home* maps to the name of the client of a UML Usage, stereotyped as «EJBPrimaryKey», whose supplier is the UML Class.
- *prim-key-class* maps to the name of the UML Class.

3.6.2.1.6.3 Stereotypes

These Stereotypes apply to a UML Usage whose supplier represents an EJB Primary Key Class and whose client represents an EJB Entity Home.

Stereotype	Definition
«EJBPrimaryKey»	Indicates that the supplier of the Usage represents the EJB Primary Key Class for the EJB Entity Home represented by the client.

Table 3-23. EJB Primary Key Class Stereotypes

3.6.2.1.6.4 Constraints

These Constraints apply to a UML Class that represents an EJB Primary Key Class.

1) The Class must be a valid RMI/IIOP value type[18].
2) The Class must contain implementations for Operations named hashCode and equals.
3) The Class must be the supplier of a UML Usage, stereotyped as «EJBPrimaryKey», whose client represents an EJB Entity Home

[18] The EJB Specification imposes this constraint, but does not define the meaning of the term "valid RMI/IIOP value type".

3.6.2.2 Internal View

This section describes the representation of logical constructs visible to the developers of an EJB Enterprise Bean. It includes and may extend the constructs included in the external view.

3.6.2.2.1 EJB Method

3.6.2.2.1.1 Syntax

The declaration of an EJB Method in the internal view extends the declaration in the external view with EJB Deployment Descriptor elements and an optional corresponding Java Method declaration in the EJB Implementation Class[19]. Properties of the EJB Method are specified by the container-transaction and method-permission elements. These elements have the following XML DTD syntax.

```
<!ELEMENT container-transaction (description?, method+, trans-attribute)>
<!ELEMENT ejb-name (#PCDATA)>
<!ELEMENT method (description?, ejb-name, method-intf?, method-name, method-params?)>
<!ELEMENT method-intf (#PCDATA)>
<!ELEMENT method-name (#PCDATA)>
<!ELEMENT method-param (#PCDATA)>
<!ELEMENT method-params (method-param*)>
<!ELEMENT method-permission (description?, role-name+, method+)>
<!ELEMENT role-name (#PCDATA)>
<!ELEMENT trans-attribute (#PCDATA)>
```

3.6.2.2.1.2 Mapping

The mapping for an EJB Method in the internal view extends the mapping in the external view as follows. The corresponding Java Method declaration in the EJB Implementation Class, if present, maps to a UML Operation contained by the model element that represents the EJB Implementation Class. The elements used in the declaration map as follows.

- If the EJB Method is described by a method element in the container-transaction element, then *trans-attribute* maps to the value of the EJBTransAttribute tag on the UML Operation.
- If the EJB Method is described by a method element in the method-permission element, then the *role-names*, separated by commas, form the value of the EJBRoleNames tag on the UML Operation.

A method element used in either of the elements described above maps as follows.

- *ejb-name* maps to the value of the EJBNameInJAR tag on the model element that represents the EJB Enterprise Bean.
- *method-intf*, if present, has the value Home, if the EJB Method is an EJB Home Interface Method, or the value Remote, if the EJB Method is an EJB Remote Method. *method-intf* is used when EJB Methods in the EJB Home and Remote Interfaces have the same name and Parameters.
- *method-name* maps to the simple name of the UML Operation.
- If method-params is present, each *method-param* maps to the name of the Type of the corresponding in Parameter contained by the UML Operation.

3.6.2.2.1.3 Stereotypes

The Stereotypes that optionally apply to a UML Operation that represents an EJB Method in the external view also optionally apply to the corresponding UML Operation, contained by the model element that represents the EJB Implementation Class, in the internal view.

[19] Local Java Methods declared by the EJB Implementation Class are not EJB constructs, per se. and are therefore described by the Java Logical Model, not the EJB Logical Model.

3.6.2.2.1.4 Tagged Values

These Tagged Values apply to a UML Operation, contained by the model element that represents the EJB Implementation Class, that represents an EJB Method in the internal view.

Tag	Value
EJBRoleNames	A comma-delimited list of Strings, designating the security roles that may invoke the Operation.
EJBTransAttribute	Not Supported, Supports, Required, RequiresNew, Mandatory, or Never. Defines the transaction management policy for the Operation.

Table 3-24. EJB Method Tagged Values

3.6.2.2.1.5 Constraints

These additional Constraints apply to a UML Operation, contained by the model element that represents the EJB Implementation Class, that represents an EJB Method in the internal view.

1) The value of the EJBTransAttribute tag must be Not Supported, Supports, Required, RequiresNew, Mandatory, or Never.

3.6.2.2.2 EJB Remote Interface

3.6.2.2.2.1 Syntax

The declaration of an EJB Remote Interface in the internal view extends the declaration in the external view with additional EJB Deployment Descriptor elements. The name of the EJB Implementation Class is specified by the ejb-class element in the entity or session element for the EJB Enterprise Bean. These elements have the following XML DTD syntax.

```
<!ELEMENT entity (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class, persistence-type, prim-key-class, reentrant, cmp-field*, primkey-field?, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
<!ELEMENT ejb-class (#PCDATA)>
<!ELEMENT session (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class, session-type, transaction-type, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
```

3.6.2.2.2.2 Mapping

The mapping for an EJB Remote Interface in the internal view extends the mapping in the external view as follows. The UML Class that represents the EJB Remote Interface is contained or imported by the specification part of a UML Subsystem that represents the EJB Enterprise Bean. The elements used in the declaration map as follows.

- *ejb-class* maps to the name of the client of a UML Abstraction, stereotyped as «EJBRealizeRemote», whose supplier is the UML Class.

3.6.2.2.2.3 Stereotypes

These Stereotypes apply to a UML Abstraction whose client represents an EJB Implementation Class and whose supplier represents an EJB Remote Interface.

Stereotype	Definition
«EJBRealizeRemote»	Indicates that the supplier of the Abstraction represents the EJB Remote Interface for the EJB Implementation Class represented by the client.

Table 3-25. EJB Remote Interface Stereotypes

3.6.2.2.2.4 Constraints

These additional Constraints apply to a UML Class that represents an EJB Remote Interface in the internal view.

1) The Class must be the supplier of a UML Abstraction, stereotyped as «EJBRealizeRemote», whose client represents the EJB Implementation Class of the same EJB Enterprise Bean.

3.6.2.2.3 EJB Home Interface

3.6.2.2.3.1 Syntax

The declaration of an EJB Home Interface in the internal view extends the declaration in the external view with additional EJB Deployment Descriptor elements. The name of the EJB Implementation Class is specified by the ejb-class element in the entity or session element for the EJB Enterprise Bean. These elements have the following XML DTD syntax.

```
<!ELEMENT entity (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class, persistence-type, prim-key-class, reentrant, cmp-field*, primkey-field?, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
<!ELEMENT ejb-class (#PCDATA)>
<!ELEMENT session (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class, session-type, transaction-type, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
```

3.6.2.2.3.2 Mapping

The mapping for an EJB Home Interface in the internal view extends the mapping in the external view as follows. The UML Class that represents the EJB Home Interface is contained or imported by the specification part of a UML Subsystem that represents the EJB Enterprise Bean. The elements used in the declaration map as follows.

- *ejb-class* maps to the name of the client of a UML Abstraction, stereotyped as «EJBRealizeHome», whose supplier is the UML Class.

3.6.2.2.3.3 Stereotypes

These Stereotypes apply to a UML Abstraction whose client represents an EJB Implementation Class and whose supplier represents an EJB Home Interface.

Stereotype	Definition
«EJBRealizeHome»	Indicates that the supplier of the Abstraction represents the EJB Home Interface for the EJB Implementation Class represented by the client.

Table 3-26. EJB Home Interface Stereotypes

3.6.2.2.3.4 Constraints

These additional Constraints apply to a UML Class that represents an EJB Home Interface in the internal view.

1) The Class must be the supplier of a UML Abstraction, stereotyped as «EJBRealizeHome», whose client represents the EJB Implementation Class of the same EJB Enterprise Bean.

3.6.2.2.4 EJB Implementation Class

3.6.2.2.4.1 Syntax

The declaration of an EJB Implementation Class extends the declaration of a Java Class with EJB Deployment Descriptor elements. Properties of the EJB Implementation Class are specified by the entity and session elements. These elements have the following XML DTD syntax.

```
<!ELEMENT ejb-class (#PCDATA)>
<!ELEMENT entity (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class,
persistence-type, prim-key-class, reentrant, cmp-field*, primkey-field?, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
<!ELEMENT home (#PCDATA)>
<!ELEMENT remote (#PCDATA)>
<!ELEMENT session (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class,
session-type, transaction-type, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
```

3.6.2.2.4.2 *Mapping*

The mapping for an EJB Implementation Class extends the mapping for a Java Class as follows. The UML Class stereotyped as «EJBImplementation». The elements used in the declaration map as follows.

- *ejb-class* maps to the name of the UML Class.
- *home* maps to the name of the model element that represents the EJB Home Interface. The model element is the supplier of a UML Abstraction, stereotyped as «EJBRealizeHome», whose client is the UML Class.
- *remote* maps to the name of the model element that represents the EJB Remote Interface. The model element is the supplier of a UML Abstraction, stereotyped as «EJBRealizeRemote», whose client is the UML Class.

3.6.2.2.4.3 *Stereotypes*

These Stereotypes apply to a UML Class that represents an EJB Implementation Class.

Stereotype	**Definition**
«EJBImplementation»	Specializes the standard UML Stereotype «implementationClass». Indicates that the Class describes an EJB Implementation Class, distinguishing it from other Classes that may appear within a UML Subsystem that represents an EJB Enterprise Bean.

Table 3-27. EJB Implementation Class Stereotypes

3.6.2.2.4.4 *Constraints*

These Constraints apply to a UML Class that represents an EJB Implementation Class.

1) The Class must have a visibility property value of public, and isLeaf, isAbstract and isActive property values of false.
2) The Class must contain a public Operation that represents a Java Constructor Method, and that has no in Parameters.
3) The Class must not contain a public Operation named finalize.
4) All of its Operations must be have a concurrency property value of sequential.
5) The Class must be contained or imported by the realization part of a UML Subsystem that represents an EJB Enterprise Bean.
6) The Class must be the client of a UML Abstraction, stereotyped as «EJBRealizeHome», whose supplier represents the EJB Home Interface.
7) The Class must be the client of a UML Abstraction, stereotyped as «EJBRealizeRemote», whose supplier represents the EJB Remote Interface.
8) For every Operation that represents an EJB Create Method, the Class must contain a matching Operation with the same in Parameters that satisfies the following Constraints.
   a) Its name must be ejbCreate.
   b) It must have a visibility property value of public, an isLeaf property value of false, and an ownerScope property value of instance.
   c) The name of every Java Exception Class in its JavaThrows tag must appear in the JavaThrows tag on the matching Operation.
9) For every Operation that represents an EJB Remote Method, other than getEJBHome, getHandle, getPrimaryKey, isIdentical or remove, the Class must contain a matching Operation with the same name and Parameters that satisfies the following Constraints.
   a) The Operation must have a visibility property value of public, isLeaf property and isStatic tag values of false, and an ownerScope property value of instance.
   b) The name of every Java Exception Class in its JavaThrows tag must appear in the JavaThrows tag on the matching Operation.

3.6.2.2.5 EJB Enterprise Bean

3.6.2.2.5.1 Syntax

An EJB Enterprise Bean is declared by an EJB Home Interface, an EJB Remote Interface, an EJB Implementation Class, supplemental Java Classes or interfaces, and EJB Deployment Descriptor elements. Properties of the EJB Enterprise Bean are specified by the entity and session elements. These elements have the following XML DTD syntax.

```
<!ELEMENT ejb-class (#PCDATA)>
<!ELEMENT ejb-link (#PCDATA)>
<!ELEMENT ejb-name (#PCDATA)>
<!ELEMENT ejb-ref (description?, ejb-ref-name, ejb-ref-type, home, remote, ejb-link?)>
<!ELEMENT ejb-ref-name (#PCDATA)>
<!ELEMENT ejb-ref-type (#PCDATA)>
<!ELEMENT entity (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class,
persistence-type, prim-key-class, reentrant, cmp-field*, primkey-field?, env-entry*, ejb-ref*, security-role-ref*, resource-
ref*)>
<!ELEMENT env-entry (description?, env-entry-name, env-entry-type, env-entry-value?)>
<!ELEMENT env-entry-name (#PCDATA)>
<!ELEMENT env-entry-type (#PCDATA)>
<!ELEMENT env-entry-value (#PCDATA)>
<!ELEMENT home (#PCDATA)>
<!ELEMENT remote (#PCDATA)>
<!ELEMENT res-auth (#PCDATA)>
<!ELEMENT res-ref-name (#PCDATA)>
<!ELEMENT res-type (#PCDATA)>
<!ELEMENT resource-ref (description?, res-ref-name, res-type, res-auth)>
<!ELEMENT role-link (#PCDATA)>
<!ELEMENT role-name (#PCDATA)>
<!ELEMENT security-role-ref (description?, role-name, role-link?)>
<!ELEMENT session (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class,
session-type, transaction-type, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
```

3.6.2.2.5.2 Mapping

An EJB Enterprise Bean maps to a UML Subsystem stereotyped as «EJBEnterpriseBean». The UML Subsystem contains model elements that represent the Java Classes and Interfaces that comprise the EJB Enterprise Bean, and may contain model elements that represent the Java Class Files in which the Java Classes and Interfaces reside. The EJB Deployment Descriptor elements used in the declaration of both EJB Session and Entity Beans map as follows.

- *description*, if specified, maps to the value of the documentation property of the UML Subsystem.
- *display-name*, if specified, maps to the value of the name of the UML Subsystem. If *display-name* is not specified, the name of the UML Subsystem defaults to the simple name of the EJB Remote Interface.
- *ejb-class* maps to the name of the model element that represents the EJB Implementation Class.
- *ejb-name* maps to the value of the EJBNameInJAR tag on the UML Subsystem.
- *ejb-ref* maps to a tuple of the form <name, type, home, remote>. The *ejb-ref* tuples, separated by commas, form the value of the EJBReferences tag on the UML Subsystem. The elements of *env-entry* map as follows.
  - *ejb-ref-name* maps to the name element of the tuple.
  - *ejb-ref-type* maps to the type element of the tuple.
  - *home* maps to the home element of the tuple.
  - *remote* maps to the remote element of the tuple.
  - *ejb-link*, if specified, maps to the value of the EJBNameInJAR tag on a model element that represents another EJB Enterprise Bean. The model element is the type of a navigable UML AssociationEnd, whose name is the value of the name element of the tuple, contained by the UML Subsystem.
- *ejb-ref* may additionally and optionally map to a unidirectional UML Association stereotyped as «EJBReference» navigable to another UML subsystem stereotyped as «EJBEnterpriseBean». The navigable UML associationEnd must have a targetScope of Classifier.
- *env-entry* maps to a tuple of the form <name, type, value>. The *env-entry* tuples, separated by commas, form the value of the EJBEnvEntries tag on the UML Subsystem. The elements of *env-entry* map as follows.
  - *env-entry-name* maps to the name element of the tuple.
  - *env-entry-type* maps to the type element of the tuple.
  - *env-entry-value* maps to the value element of the tuple.
- *home* maps to the name of the model element that represents the EJB Home Interface. The model element is the supplier of a UML Abstraction, stereotyped as «EJBRealizeHome», whose client represents the EJB Implementation Class.
- *remote* maps to the name of the model element that represents the EJB Remote Interface. The model element is the supplier of a UML Abstraction, stereotyped as «EJBRealizeRemote», whose client represents the EJB Implementation Class.
- *resource-ref* maps to a tuple of the form <name, type, auth>. The *resource-ref* tuples, separated by commas, form the value of the EJBResources tag on the UML Subsystem. The elements of *resource-ref* map as follows.
  - *res-auth* maps to the auth element of the tuple.
  - *res-ref-name* maps to the name element of the tuple.
  - *res-type* maps to the type element of the tuple.
- *security-role-ref* maps to a tuple of the form <name, link>. The *security-role-ref* tuples, separated by commas, form the value of the EJBSecurityRoles tag on the UML Subsystem. The elements of *security-role-ref* map as follows.
  - *role-link*, if specified, maps to the link element of the tuple.
  - *role-name* maps to the name element of the tuple, and may additionally and optionally map to the name of a UML Actor at the opposite end of a UML Association stereotyped as «EJBAccess».

3.6.2.2.5.3 Stereotypes

These Stereotypes apply to a UML Subsystem that represents an EJB Enterprise Bean.

Stereotype	Definition
«EJBEnterpriseBean»	Indicates that the Subsystem represents an EJB Enterprise Bean.

Table 3-28. EJB Enterprise Bean Stereotypes

3.6.2.2.5.4 Tagged Values

These Tagged Values apply to a UML Subsystem, stereotyped as «EJBEnterpriseBean».

Tag	Value
EJBEnvEntries	A comma-delimited list of tuples, designating the environment entries used by the EJB Enterprise Bean, of the form <name, type, value>.
EJBNameInJAR	The name used for the EJB Enterprise Bean in the EJB-JAR. Defaults to the name of the EJB Remote Interface.
EJBReferences	A comma-delimited list of tuples, designating the other EJB Enterprise Beans referenced by the EJB Enterprise Bean, of the form <name, type, home, remote>.
EJBResources	A comma-delimited list of tuples, designating the resource factories used by the EJB Enterprise Bean, of the form <name, type, auth>.
EJBSecurityRoles	A comma-delimited list of tuples, designating the role names that may invoke ALL operations on the EJB Enterprise Bean, of the form <name, link>.

Table 3-29. EJB Enterprise Bean Tagged Values

3.6.2.2.5.5 Constraints

These Constraints apply to a UML Subsystem that represents an EJB Enterprise Bean.

1) The Subsystem must have a visibility property value of public, and isLeaf and isAbstract property values of false.
2) The specification part must contain or import exactly one model element that represents an EJB Home Interface and exactly one model element that represents an EJB Remote Interface.
3) The realization part must contain or import exactly one model element that represents an EJB Implementation Class.
4) The Subsystem may not define any Features beyond those defined by the model elements in its specification section.
5) The model element that represents the EJB Implementation Class must be the client of a UML Abstraction, stereotyped as «EJBRealizeHome», whose supplier represents the EJB Home Interface.
6) The model element that represents the EJB Implementation Class must be the client of a UML Abstraction, stereotyped as «EJBRealizeRemote», whose supplier represents the EJB Remote Interface.
7) The value of the type element of a tuple in the value of the EJBEnvEntries tag must be the name of a Java Primitive Class.
8) The value of the EJBNameInJAR tag must be unique among the model elements that represent EJB Enterprise Beans in the same EJB-JAR.
9) The value of the type element of a tuple in the value of the EJBReferences tag must be either Entity or Session.
10) These Constraints apply to a UML AssociationEnd named by a tuple in the EJBReferences tag.
    a) The AssociationEnd must be navigable, and must have a targetScope property value of classifier.
    b) The AssociationEnd must have a multiplicity property lower bound of zero and upper bound of exactly one.
    c) The AssociationEnd must have no qualifier, and must have an aggregation property value of none.
    d) The model element referenced by its type property must be a UML Subsystem that represents an EJB Enterprise Bean.
    e) The value of the home element must be the name of the model element that represents the EJB Home Interface in the specification part of the Subsystem.

f) The value of the remote element must be the name of the model element that represents the EJB Remote Interface in the specification part of the Subsystem.

11) The value of the auth element of a tuple in the EJBResources tag must be either Application or Container.
12) The value of the type element of a tuple in the EJBResources tag must be the name of a Java Class or Interface.
13) The value of the name element of a tuple in the EJBSecurityRoles tag must satisfy the lexical rules for an NMTOKEN.
14) The value of the link element of a tuple in the value of the EJBSecurityRoles tag, if specified, must be one of the security roles defined in the security-role element of the deployment descriptor.
15) A UML Association stereotyped as «EJBAccess» must correspond to an entry in the EJBSecurityRoles tag.
16) A UML Association stereotyped as «EJBReference» must correspond to an existing entry in the EJBReferences tag.
17) A UML Association stereotyped as «EJBReference» must be navigable in only one direction.
18) A UML Association stereotyped as «EJBReference» must have a targetScope value of Classifier.

3.6.2.2.6 EJB Session Bean

3.6.2.2.6.1 Syntax

The declaration of an EJB Session Bean extends the declaration of an EJB Enterprise Bean with additional EJB Deployment Descriptor elements. Properties of the EJB Session Bean are specified by the session element. These elements have the following XML DTD syntax.

```
<!ELEMENT session (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class,
session-type, transaction-type, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
<!ELEMENT transaction-type (#PCDATA)>
```

3.6.2.2.6.2 Mapping

The mapping of EJB Session Bean extends the mapping for an EJB Enterprise Bean as follows. The UML Subsystem that represents the EJB Enterprise Bean is stereotyped as «EJBSessionBean». The EJB Home Interface is an EJB Session Home. The elements used in the declaration map as follows.

- *transaction-type* maps to the value of the EJBTransType tag on the UML Subsystem.

3.6.2.2.6.3 Stereotypes

These Stereotypes apply to a UML Subsystem that represents an EJB Session Bean.

Stereotype	Definition
«EJBSessionBean»	Indicates that the Subsystem represents an EJB Session Bean. Specializes «EJBEnterpriseBean».

Table 3-30. EJB Session Bean Stereotypes

3.6.2.2.6.4 Tagged Values

These Tagged Values apply to a UML Subsystem, stereotyped as «EJBSessionBean», that represents an EJB Session Bean.

Tag	Value
EJBTransType	Bean or Container. Indicates whether the transactions of the EJB Session Bean are managed by the EJB Session Bean or by its container, respectively.

Table 3-31. EJB Session Bean Tagged Values

3.6.2.2.6.5 Constraints

3.6.2.2.6.5.1 CONSTRAINTS FOR AN EJB ENTITY BEAN

These additional Constraints apply to a UML Subsystem that represents an EJB Session Bean.

1) The value of the EJBTransType tag must be either Container or Bean.
2) The EJB Home Interface represented by a model element in the specification part must be an EJB Session Home.

3.6.2.2.6.5.2 CONSTRAINTS FOR AN EJB IMPLEMENTATION CLASS FOR AN EJB SESSION BEAN

These additional Constraints apply to a UML Class that represents an EJB Implementation Class for an EJB Session Bean.

1) The Class must realize a model element that represents the Java Interface, EJBSessionBean.
2) The Class must not be tagged as persistent.
3) If the value of the EJBSessionType tag on the model element that represents the EJB Session Home is Stateless, then the Class must not realize a model element that represents the Java Interface, javax.ejb.SessionSynchronization.
4) The type of the return Parameter of any Operation named ejbCreate contained by the Class must represent the Java Special Type void.

3.6.2.2.7 EJB Entity Bean

3.6.2.2.7.1 Syntax

The declaration of an EJB Entity Bean extends the declaration of an EJB Enterprise Bean with the declaration of an EJB Primary Key Class and additional EJB Deployment Descriptor elements. Properties of the EJB Entity Bean are specified by the entity element. These elements have the following XML DTD syntax.

```
<!ELEMENT cmp-field (description?, field-name)>
<!ELEMENT entity (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class,
persistence-type, prim-key-class, reentrant, cmp-field*, primkey-field?, env-entry*, ejb-ref*, security-role-ref*, resource-
ref*)>
<!ELEMENT field-name (#PCDATA)>
<!ELEMENT persistence-type (#PCDATA)>
<!ELEMENT primkey-field (#PCDATA)>
<!ELEMENT reentrant (#PCDATA)>
```

If the value of the persistence-type element is Container, then the EJB Entity Bean has container-managed persistence, and the following additional rules apply.

If the type of the argument of the findByPrimaryKey method on the EJB Home Interface, the return type of the ejbCreate method in the EJB Implementation Class, and primary key class named in the prim-key-class element in the EJB Deployment Descriptor are the Java Class, java.lang.Object, then the specification of the EJB Primary Key Class is deferred.

If the specification of the EJB Primary Key Class is not deferred, then the EJB Primary Key Class corresponds to one or more Java Fields in the EJB Implementation Class, or one of its superclasses, as follows.

If a primkey-field element is present in the EJB Deployment Descriptor, then its value is the name of a Java Field whose type is the EJB Primary Key Class.

In the case where multiple primary key fields map to multiple cmp-field elements, then they identify Java Fields that have the same names and types as Java Fields in the EJB Primary Key Class.

3.6.2.2.7.2 Mapping

The mapping for an EJB Entity Bean extends the mapping for an EJB Enterprise Bean as follows. The UML Subsystem that represents the EJB Enterprise Bean is stereotyped as «EJBEntityBean». The EJB Home Interface is an EJB Entity Home. The elements used in the declaration map as follows.

- *cmp-field maps to the model element that represents a Java Field in the EJB Implementation Class or one of its* superclasses. The UML Attribute that represents the *cmp-field* is stereotyped as «EJBCmpField». The *field-name* maps to the name of the UML Attribute. *reentrant* maps to the value of the EJBReentrant tag on the UML Subsystem.
- *persistence-type* maps to the value of the EJBPersistenceType tag on the UML Subsystem.

If the EJB Entity Bean has container-managed persistence, the specification of the EJB Primary Key Class is not deferred, and the EJB Primary Key Class corresponds to a single Java Field, then additional elements map as follows.

- *primkey-field* maps to the name of a model element that represents a Java Field in the EJB Implementation Class or one of its superclasses. The type of the model element maps to the model element that represents the EJB Primary Key Class. The UML Attribute that represents the *primkey-field* is stereotyped as «EJBPrimaryKeyField».

3.6.2.2.7.3 Stereotypes

These Stereotypes apply to a UML Subsystem that represents an EJB Entity Bean.

Stereotype	Definition
«EJBCmpField»	Indicates that the Attribute represents a container-managed field for an EJB Entity Bean with container-managed persistence.
«EJBPrimaryKeyField»	Indicates that the Attribute is the primary key field for an EJB Entity Bean with container-managed persistence.
«EJBEntityBean»	Indicates that the Subsystem represents an EJB Entity Bean. Specializes «EJBEnterpriseBean».

Table 3-32. EJB Entity Bean Stereotypes

3.6.2.2.7.4 Tagged Values

These Tagged Values apply to a UML Subsystem, stereotyped as «EJBEntityBean», that represents an EJB Entity Bean.

Tag	Value
EJBPersistenceType	Bean or Container. Indicates whether the persistence of the EJB Entity Bean is managed by the EJB Entity Bean or by its container, respectively.
EJBReentrant	True or False. Indicates whether or not the EJB Entity Bean can be called reentrantly.

Table 3-33. EJB Entity Bean Tagged Values

3.6.2.2.7.5 Constraints

3.6.2.2.7.5.1 CONSTRAINTS FOR AN EJB ENTITY BEAN

These additional Constraints apply to a UML Subsystem that represents an EJB Entity Bean.

1) The value of the EJBPersistenceType tag must be either Bean or Container.
2) The value of the java.lang.Reentrant tag must be either true or false.
3) The EJB Home Interface represented by a model element in the specification part must be an EJB Entity Home.
4) The specification part must contain or import the model element that represents the EJB Primary Key Class.

3.6.2.2.7.5.1.1 Additional Constraints For An EJB Entity Bean With Bean Managed Persistence

These additional Constraints apply if the value of the EJBPersistenceType tag is Bean.

1) The model element that represents the EJB Primary Key Class must not represent the Java Class, java.lang.Object.

3.6.2.2.7.5.1.2 Additional Constraints For An EJB Entity Bean With Container-managed persistence

These additional Constraints apply if the value of the EJBPersistenceType tag is Container.

1) If model element that represents the EJB Primary Key Class does not represent the Java Class, java.lang.Object, then the following additional Constraints apply.
    a) The single UML Attribute stereotyped as «EJBPrimaryKeyField» represents a single public Java Field in the EJB Implementation Class, or one of its superclasses, whose type is the model element that represents the EJB Primary Key Class.
    b) If no model elements of the EJB Implementation Class that represent Java Fields are stereotyped as «EJBPrimaryKeyField» , then all of the model elements that represent Java Fields in the model element that represents the EJB Primary Key Class must be public, and must have the same names and types as model elements that represent public Java Fields in the EJB Implementation Class, or one of its superclasses. In addition, the model element that represents the EJB Primary Key Class must be public, must realize a model element that represents the Java Interface, java.io.Serializable, and must contain a public Operation that represents a Java Constructor Method, and that has no in Parameters.

3.6.2.2.7.5.2 CONSTRAINTS FOR AN EJB IMPLEMENTATION CLASS FOR AN EJB ENTITY BEAN

These additional Constraints apply to a UML Class that represents an EJB Implementation Class for an EJB Entity Bean.

1) The Class must realize a model element representing the Java Interface, EJBEntityBean.
2) The Class must be tagged as persistent.
3) The Class must not realize a model element that represents the Java Interface, javax.ejb.SessionSynchronization.
4) The type of the return Parameter of any Operation named ejbCreate must be the model element that represents the EJB Primary Key Class.
5) For every Operation named ejbCreate, the Class must contain a matching Operation named ejbPostCreate with the same in Parameters. The matching Operation must satisfy the following Constraints.
   a) It must have a visibility property value of public, an isLeaf property value of false, and an ownerScope property value of instance.
   b) Its return Parameter type must represent the Java Special Type void.
   c) The name of every Java Exception Class in its JavaThrows tag must appear in the JavaThrows tag on an Operation that represents an EJB Create Method with the same Parameters.

3.6.2.2.7.5.2.1 Additional Constraints For An EJB Implementation Class For An EJB Entity Bean With Bean Managed Persistence

These additional Constraints apply if the value of the EJBPersistenceType tag on the EJB Entity Bean is Bean.

1) The model element that represents the EJB Primary Key Class must not represent the Java Class, java.lang.Object
2) For every Operation that represents an EJB Finder Method, the Class must contain a matching Operation with the same in Parameters that satisfies the following Constraints.
   a) Its name must have the prefix ejbFind.
   b) The remainder of its name must be the remainder of the name of the matching Operation.
   c) It must have a visibility property value of public, an isLeaf property value of false, and an ownerScope property value of instance.
   d) If the return Parameter type of the matching Operation is the model element that represents the EJB Remote Interface, its return Parameter type must be the model element that represents the EJB Primary Key Class. Otherwise, its return Parameter type must be the return Parameter type of the matching Operation, and must represent a Java Collection Type.
   e) The name of every Java Exception Class in its JavaThrows tag must appear in the JavaThrows tag on the matching Operation.

3.6.2.2.7.5.2.2 Additional Constraints For An EJB Implementation Class For An EJB Entity Bean With Container Managed Persistence

These additional Constraints apply if the value of the EJBPersistenceType tag on the EJB Entity Bean is Container.

1) Every Attribute stereotyped as «EJBCmpField» must be tagged as Persistent.
2) Every Attribute tagged as Persistent must be stereotyped as «EJBCmpField».

3.6.2.2.8 EJB Primary Key Class

3.6.2.2.8.1 Syntax

The declaration of an EJB Primary Key Class in the internal view extends the declaration in the external view with additional EJB Deployment Descriptor elements. These elements have the following XML DTD syntax.

```
<!ELEMENT cmp-field (description?, field-name)>
<!ELEMENT entity (description?, display-name?, small-icon?, large-icon?, ejb-name, home, remote, ejb-class,
persistence-type, prim-key-class, reentrant, cmp-field*, primkey-field?, env-entry*, ejb-ref*, security-role-ref*, resource-ref*)>
<!ELEMENT field-name (#PCDATA)>
<!ELEMENT primkey-field (#PCDATA)>
```

3.6.2.2.8.2 Mapping

The mapping for an EJB Primary Key Class in the internal view extends the mapping in the external view as follows. The UML Class that represents the EJB Primary Key Class is contained or imported by the specification part of the UML Subsystem that represents the EJB Entity Bean. The elements used in the declaration map as follows.

- *field-name* maps to the name of a model element that represents a Java Field in the EJB Implementation Class, or one of its superclasses. The model element has the same name and type as a model element that represents a public Java Field in the UML Class.

If the EJB Entity Bean has container-managed persistence, the specification of the EJB Primary Key Class is not deferred, and the EJB Primary Key Class corresponds to a single Java Field, then additional elements map as follows.

- *primkey-field* maps to the name of a model element that represents a Java Field in the EJB Implementation Class or one of its superclasses. The type of the model element maps to the model element that represents the EJB Primary Key Class. The UML Attribute that represents the *primkey-field* is stereotyped as «EJBPrimaryKeyField».

3.6.2.2.8.3 Constraints

3.6.2.2.8.3.1.1 Constraints For An EJB Primary Key Class For An EJB Entity Bean With Bean Managed Persistence

These additional Constraints apply to a UML Class that represents an EJB Primary Key Class in the internal view if the value of the EJBPersistenceType tag on the EJB Entity Bean is Bean.

1) The Class must not represent the Java Class, java.lang.Object.

3.6.2.2.8.3.1.2 Constraints For An EJB Primary Key Class For An EJB Entity Bean With Container-managed persistence

These additional Constraints apply to a UML Class that represents an EJB Primary Key Class in the internal view if the value of the EJBPersistenceType tag on the EJB Entity Bean is Container.

1) If the Class does not represent the Java Class, java.lang.Object, then the following additional Constraints apply.
   a) If a single UML Attribute stereotyped as «EJBPrimaryKeyField» is specified on the EJB Implementation Class, its type must be the Class.
   b) If no model elements of the EJB Implementation Class that represent Java Fields are stereotyped as «EJBPrimaryKeyField» , then all of the model elements that represent Java Fields in the model element that represents the EJB Primary Key Class must be public, and must have the same names and types as model elements that represent public Java Fields in the EJB Implementation Class, or one of its superclasses. In addition, the model element that represents the EJB Primary Key Class must be public, must realize a model element that represents the Java Interface, java.io.Serializable, and must contain a public Operation that represents a Java Constructor Method, and that has no in Parameters.

3.6.3 Java Implementation Model

This section defines the mechanisms used to produce a implementation model of Java language constructs included in the Java Design Model. It enumerates the subset of Java physical constructs supported by this profile, and for each construct defines a mapping to UML model elements, including abstract syntax and applicable Stereotypes, Tagged Values and Constraints.

This mapping does not attempt to describe any Java physical constructs that can not be used in the implementation of EJBs.

All standard tags and Stereotype names defined by the Java Design Model start with the prefix "java.lang." or "java.util.".

3.6.3.1 Java Class File

3.6.3.1.1 Syntax

A Java Class File is a file that contains the compiled representation of a Java Compilation Unit that declares one top level Java Class or Interface, and zero or more nested Java Classes or Interfaces[20]. The name of the file is the name of the top level Java Class or Interface, plus a host specific extension, e.g., ".class". The Java Compilation Unit is declared by the following Java syntax.

[PackageDeclaration] { ImportDeclaration } [TypeDeclaration ; *]*

3.6.3.1.2 Mapping

A Java Class File maps to a UML Component stereotyped as «JavaClassFile». The name of the UML Component is the simple name of the Java Class File without the extension. The elements of the declaration map as follows.

- *PackageDeclaration* maps to the name of a UML Package that represents the Java Package that contains the single top level Java Class or Interface.
- *ImportDeclaration* maps to a UML Permission, stereotyped as «access», that represents a Java Single Type Import or Java Type Import On Demand. The UML Component is the client of the UML Permission.
- *TypeDeclaration* maps to a model element, resident in the UML Component, that represents the top level Java Class or Interface. Java Class or Interface declarations nested within *TypeDeclaration* map to model elements, resident in the UML Component, that represent Java Classes or Interfaces.

3.6.3.1.3 Stereotypes

These Stereotypes apply to a UML Component that represents a Java Class File.

Stereotype	Definition
«JavaClassFile»	Specializes the standard UML Stereotype «file». Indicates that the Component describes a Java Class File.

Table 3-34. Java Class File Stereotypes

3.6.3.1.4 Constraints

These Constraints apply to a UML Component that represents a Java Class File.

1) Every resident of the Component must represent a Java Class or Interface.
2) At most one of the residents of the Component may represents a top level Java Class or Interface. The model element must directly or indirectly contain the other residents of the Component, and must have the same simple name as the Component.
3) The ElementResidence that references the model element that represents the top level Java Class or Interface must have a visibility property value of public. The ElementResidence that references every other resident of the Component must have the same visibility property value as the ElementOwnership that references the resident.

[20] The compilation unit in which a Java Reference Type is declared can have other forms. However, the Java Language Specification requires that systems using other forms provide an option to convert compilation units to this form.

3.6.3.2 Java Archive File

3.6.3.2.1 Syntax

A Java Archive File or JAR is declared by a file in the Zip format that may contain a META-INF directory[21]. The file name may include a host specific extension, e.g., ".jar". The META-INF directory may contain a JAR Manifest named "MANIFEST.MF".

3.6.3.2.2 Mapping

A JAR maps to a UML Package stereotyped as «JavaArchiveFile». The name of the UML Package is the simple name of the JAR without the extension.

A directory in the JAR maps to a UML Package. The simple name of the UML Package is the simple name of the directory.

The hierarchy of directories in the JAR maps to a hierarchy of UML Packages contained by the UML Package that represents the JAR. The fully-qualified name of a top level directory is its simple name. The fully-qualified name of a directory contained by another directory is the fully-qualified name of the containing directory, followed by "/", followed by the simple name of the directory. The fully-qualified name of a directory maps to the fully-qualified name of the corresponding UML Package by replacing every occurrence of "/" with "::".

A file in the JAR maps to a model element contained directly or indirectly by the UML Package that represents the JAR. If the file has a specific mapping, e.g., Java Class File or EJB Deployment Descriptor, then the model element is defined by the mapping. If not, then the model element is a UML Component stereotyped as «file».

If the file resides at the top level of the JAR, then the model element is contained directly by the UML Package that represents the JAR. If not, then the model element is contained directly by the UML Package that represents the directory that contains the file in the JAR.

The simple name of the model element is the simple name of the file. The fully-qualified name of a top level file is its simple name. The fully-qualified name of a file contained by a directory is the fully-qualified name of the directory, followed by "/", followed by the simple name of the file. The fully-qualified name of a file maps to the fully-qualified name of the corresponding model element by replacing every occurrence of "/" with "::".

3.6.3.2.3 Stereotypes

These Stereotypes apply to a UML Package that represents a JAR.

Stereotype	Definition
«JavaArchiveFile»	Indicates that the Package represents a JAR.

Table 3-35. Java Archive File Stereotypes

[21] For a more detailed description of the format, see the *JAR File Specification* contained in the *Java 2 Platform Standard Edition Specification*.

3.6.4 EJB Implementation Model

This section defines the mechanisms used to produce an implementation model of constructs defined in the EJB Design Model. It enumerates the physical constructs defined by the EJB Specification, and for each construct defines a mapping to UML model elements, including abstract syntax and applicable Stereotypes, Tagged Values and Constraints.

All standard tags and Stereotype names defined by the EJB Design Model start with the prefix "javax.ejb.".

3.6.4.1 EJB-JAR

3.6.4.1.1 Syntax

An EJB-JAR is a JAR that contains a file named META-INF/ejb-jar.xml that contains an EJB Deployment Descriptor, and Java Class Files containing the declarations of Java Classes and Interfaces that comprise the EJB Enterprise Beans declared by the EJB Deployment Descriptor.

3.6.4.1.2 Mapping

The mapping for an EJB-JAR extends the mapping for a JAR as follows. The UML Package that represents the JAR is stereotyped as «EJB-JAR». The file named META-INF/ejb-jar.xml maps to an EJB Deployment Descriptor.

3.6.4.1.3 Stereotypes

These Stereotypes apply to a UML Package that represents an EJB-JAR.

Stereotype	Definition
«EJB-JAR»	Specializes the Stereotype «JavaArchiveFile». Indicates that the Package represents an EJB-JAR.

Table 3-36. EJB-JAR Stereotypes

3.6.4.1.4 Constraints

These Constraints apply to a UML Package, stereotyped as «EJB-JAR», that represents an EJB-JAR.

1) The Package must contain a model element that represents an EJB Deployment Descriptor.
2) The Package must contain or import model elements that represent Java Class Files for the Java Classes and Interfaces that comprise every EJB Enterprise Bean declared by the EJB Deployment Descriptor, and for the Java Classes and Interfaces on which the EJB Enterprise Beans depend.

3.6.4.2 EJB Deployment Descriptor

3.6.4.2.1 Syntax

An EJB Deployment Descriptor is an XML file with the following DOCTYPE specification.

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
"http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
```

The contents of the file have the following XML DTD syntax, per the DOCTYPE specification.

```
<!ELEMENT assembly-descriptor (security-role*, method-permission*, container-transaction*)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT ejb-client-jar (#PCDATA)>
<!ELEMENT ejb-jar (description?, display-name?, small-icon?, large-icon?, enterprise-beans, assembly-descriptor?, ejb-client-jar?)>
<!ELEMENT enterprise-beans (session | entity)+>
<!ELEMENT role-name (#PCDATA)>
<!ELEMENT security-role (description?, role-name)>
```

3.6.4.2.2 Mapping

An EJB Deployment Descriptor maps to a UML Component, stereotyped as «EJBDescriptor», whose name is the simple name of the file without any extension. The contents of the file map as follows.

- *container-transaction* maps to model elements that represent an EJB Method, as described in the EJB Design Model.
- *description*, if specified, maps to the value of the documentation property of the UML Component.
- *ejb-client-jar*, if specified, maps to the fully-qualified name of a model element that represents an EJB-JAR. The model element is the client of a UML Usage, stereotyped as «EJBClientJAR», whose supplier represents the EJB-JAR that contains the UML Component.
- *entity* maps to model elements that represent an EJB Entity Bean, as described in the EJB Design Model.
- *method-permission* maps to model elements that represent an EJB Method, as described in the EJB Design Model.
- *session* maps to model elements that represent an EJB Session Bean, as described in the EJB Design Model.

3.6.4.2.3 Stereotypes

These Stereotypes apply to a UML Component that represents an EJB Deployment Descriptor.

Stereotype	Definition
«EJBDescriptor»	Specializes the standard Stereotype «file». Indicates that the Component represents an EJB Deployment Descriptor.

Table 3-37. EJB Deployment Descriptor Stereotypes

These Stereotypes apply to a UML Usage whose client and supplier represent EJB-JARs.

Stereotype	Definition
«EJBClientJAR»	Indicates that the client of the Usage represents an ejb-client-jar for the supplier.

Table 3-38. EJB Deployment Descriptor Client JAR Stereotypes

3.7 Well-Formedness rules

This section specifies OCL well-formedness rules that formally define the semantics of the profile.

This section is not yet complete.

4 UML Descriptor

This section defines the UML descriptor component of this specification. The UML descriptor is an XML file that contains a UML model of the Java and EJB-based artifacts stored within an EJB-JAR. Section 4.1 provides an overview of the UML descriptor. Section 4.2 defines the UML descriptor's XML DTD.

4.1 Overview

The UML descriptor can be used to generate documentation describing the contents of the EJB-JAR, to support the modification or assembly of the contents of the EJB-JAR within a tool, and to support the execution of the contents of the EJB-JAR. The descriptor must use the profile defined by this specification to describe the contents of the EJB-JAR. In addition to information that can be captured using the profile defined in this specification, the UML descriptor may contain information captured using extensions to this profile.

The UML descriptor must be stored with the name META-INF/ejb-uml.xml in the ejb-jar file. The UML descriptor must describe all of the classes stored within the EJB-JAR that are mentioned in the deployment descriptor, and may optionally describe any classes or other artifacts stored within the EJB-JAR that are not mentioned in the deployment descriptor.

Java language programs should access the metadata stored in the UML descriptor using the Java Metadata Interface (JMI), a specification developed under JSR-40 of the Java Community Process. JMI defines the Java language APIs for metadata repositories based on the OMG's Meta Object Facility (MOF). The reference implementation for this specification uses JMI.

4.2 UML Descriptor DTD

This section defines the XML DTD for the UML descriptor. The DTD describes the format of the UML descriptor. Because this profile uses only standard UML extension mechanisms, the UML descriptor uses the standard XML DTD for version 1.3 of UML. This DTD is based on version 1.1 of the XML Metadata Interchange Format (XMI), and is specified by OMG document ad/99-10-15, available at http://cgi.omg.org/cgi-bin/doc?ad/99-10-15.dtd. In practice, only a subset of the DTD, describing the Foundation and Model Management packages, is used by the UML descriptor. The UML descriptor uses the following declaration to reference the DTD[22].

```
<!DOCTYPE XMI SYSTEM 'UMLX13.dtd' >
```

[22] More information about XMI can be found in the specification, available at ftp://ftp.omg.org/pub/docs/formal/00-06-01.pdf.

5 Virtual Metamodel

5.1 Background

A *virtual metamodel* (VMM) is a formal model of a set of UML extensions, expressed in UML. The VMM for the UML Profile for EJB is presented in this chapter as a set of class diagrams. More information about virtual metamodels can be found in the UML Profile for CORBA[23].

5.1.1 Representation of Stereotypes

The VMM represents a Stereotype as a Class stereotyped «stereotype» (sic). The Class that represents the Stereotype is the client of a Dependency stereotyped «baseElement», whose supplier is the UML metamodel element being extended.

In UML, Stereotype is a GeneralizableElement. Therefore, inheritance hierarchies of Stereotype instances can be constructed, and a Stereotype can be designated as abstract. This VMM makes use of these capabilities.

5.1.2 Representation of TaggedValues

The VMM represents a TaggedValue associated with a Stereotype as an Attribute of the Class that represents the Stereotype. The Attribute is stereotyped «TaggedValue». In UML 1.3, a TaggedValue does not have a type, so the VMM specifies the type of data used by the value portion of the TaggedValue informally using the attribute type expression. An expression of the form <x,y,…,z> indicates that the value is a comma-delimited tuple. An expression of the form (x,y,…,z) indicates that the value is an enumeration.

The VMM can also specify TaggedValues that are not related to any Stereotype. The VMM represents this kind of TaggedValue as an Attribute of a nameless Class stereotyped «TaggedValues». The Class stereotyped «TaggedValues» is the client of a Dependency stereotyped «baseElement», whose supplier is the UML metamodel element being extended. When multiple TaggedValues extend the same base element of the UML metamodel, all of them may be grouped together as Attributes of a single «TaggedValues» stereotyped Class.

[23] OMG document ad/00-05-07.

5.2 VMM Package Structure

The VMM adds new Packages to the base UML 1.3 metamodel. These Packages contain the Stereotypes and TaggedValues that make up the profile.

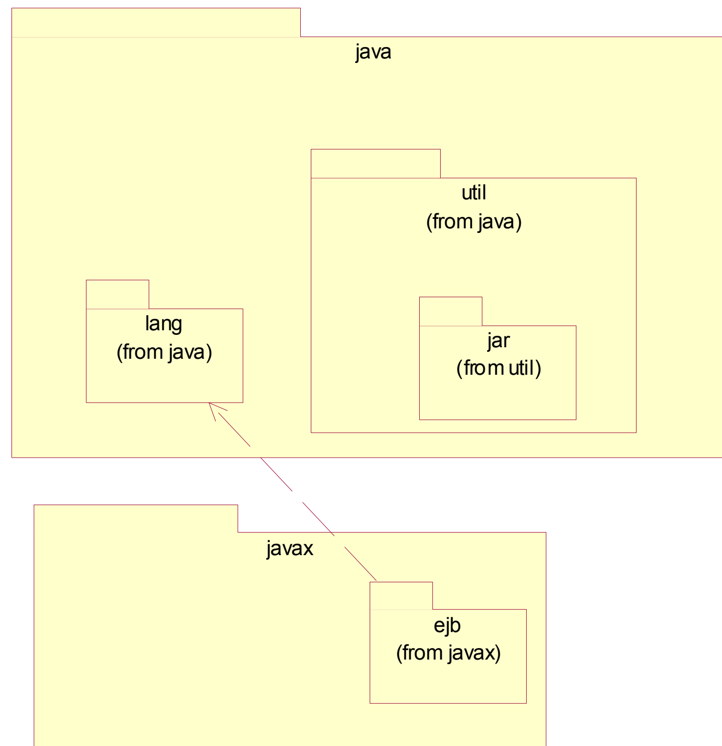

Figure 5-1. Virtual Metamodel Packages

5.3 Package java::util::jar

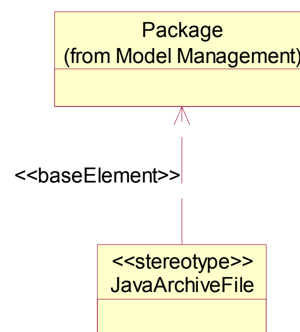

Figure 5-2. Stereotype Defined in Package java::util::jar

5.4 Package java::lang

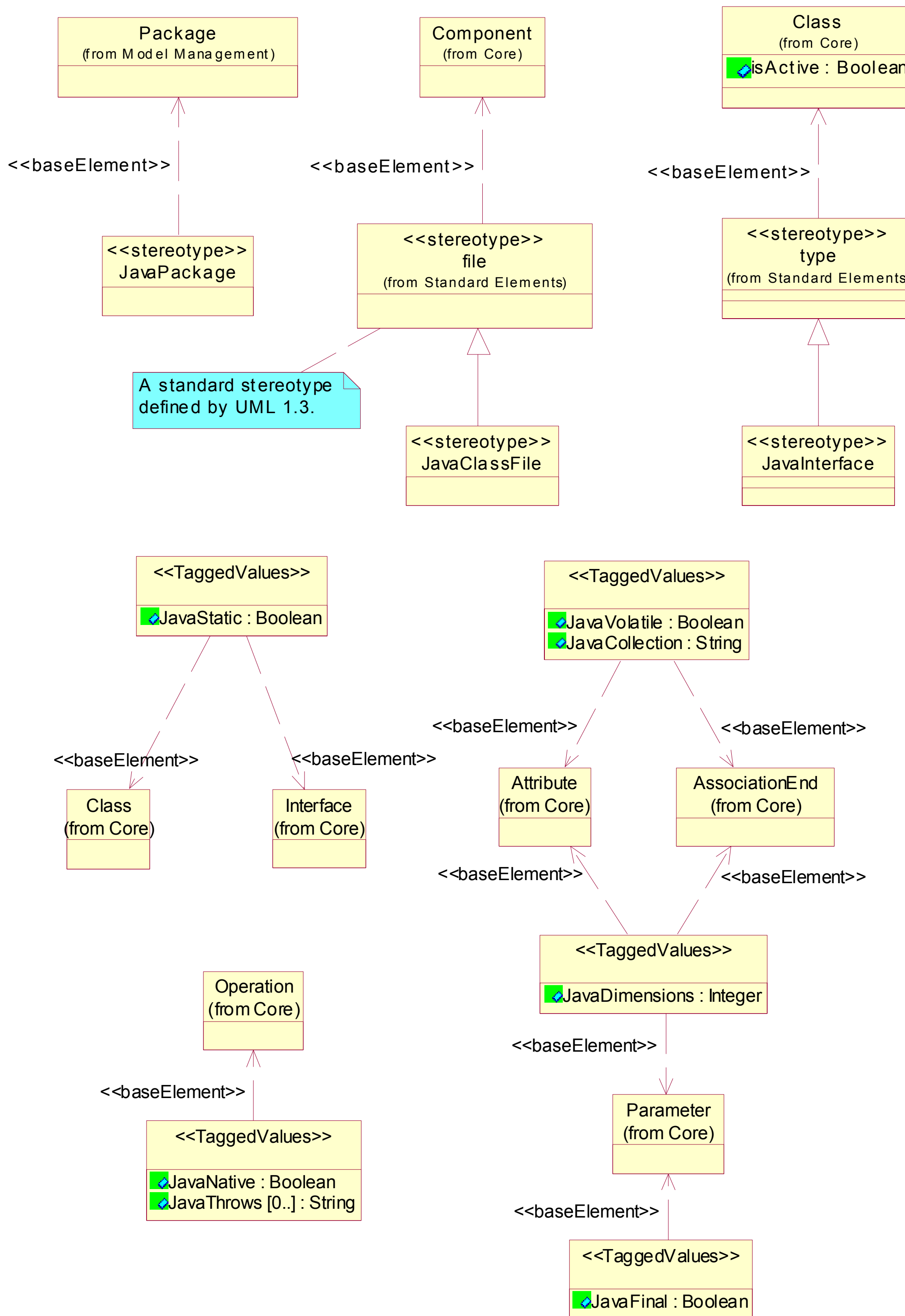

Figure 5-3. Stereotypes and Tagged Values Defined in Package java::lang

5.5 Package javax::ejb

5.5.1 Design Model

5.5.1.1 External View

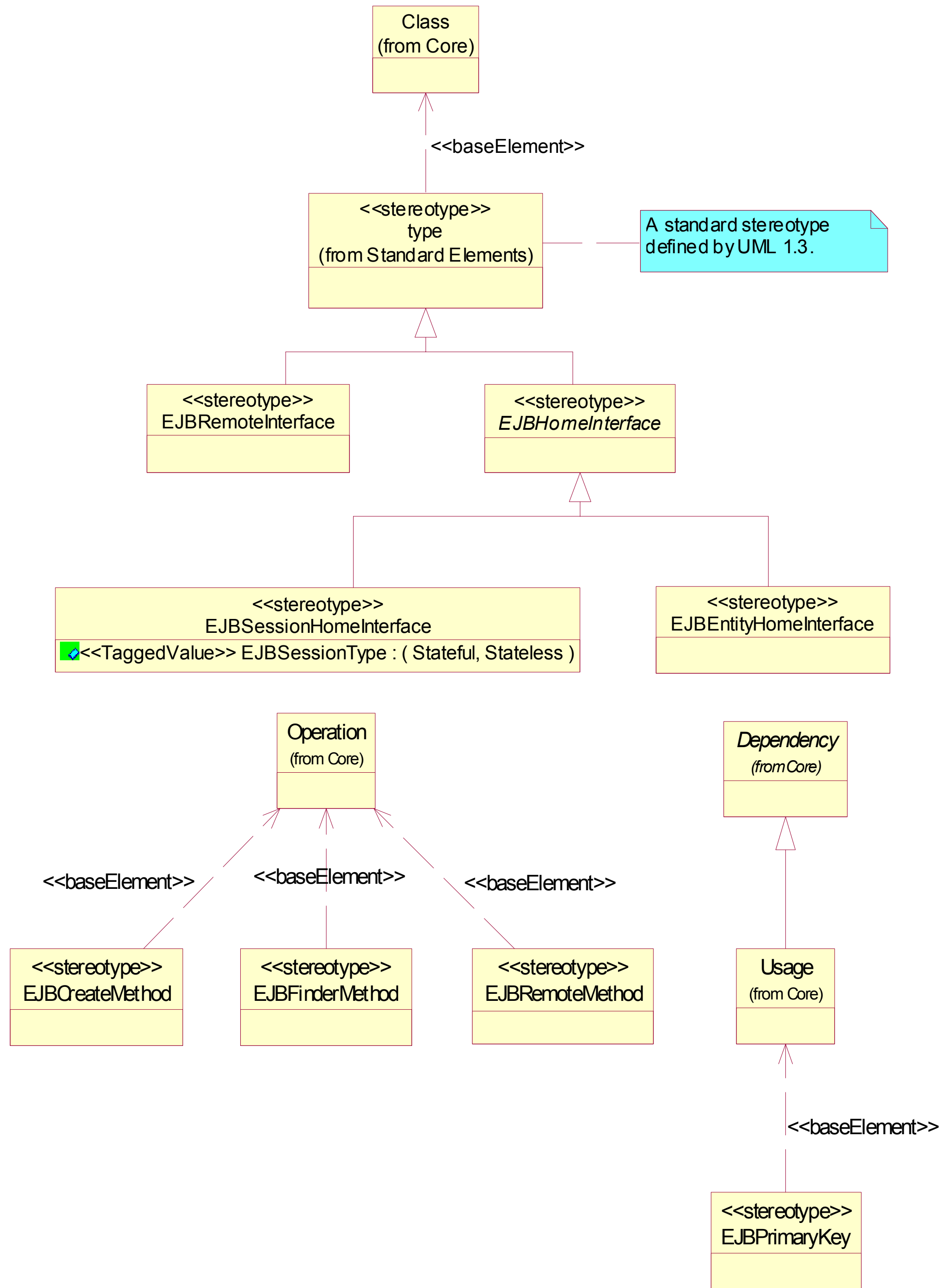

Figure 5-4. Stereotypes Defined in EJB Design Model - External View

5.5.1.2 Internal View

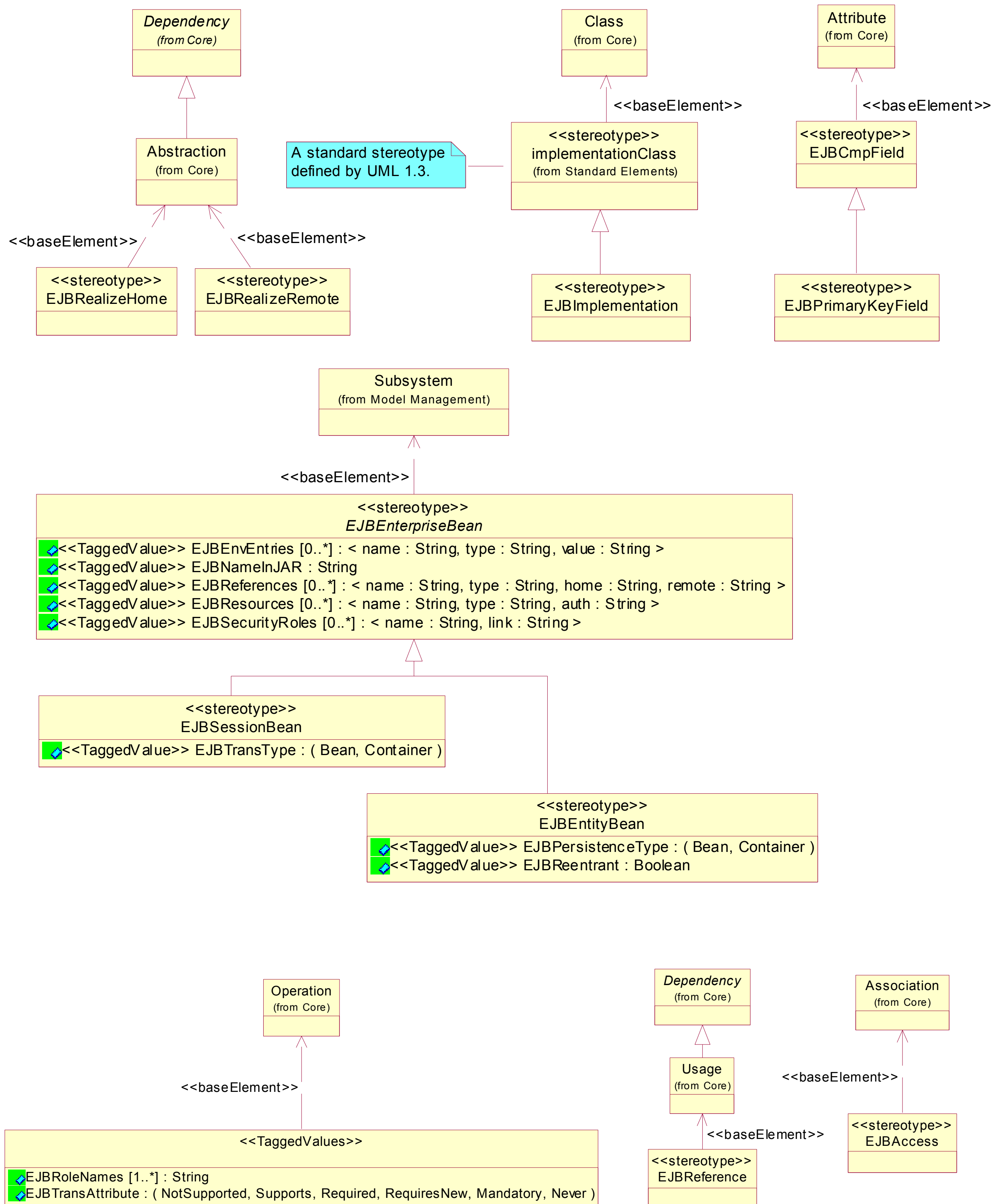

Figure 5-5. Stereotypes and Tagged Values Defined in EJB Design Model - Internal View

5.5.2 Implementation Model

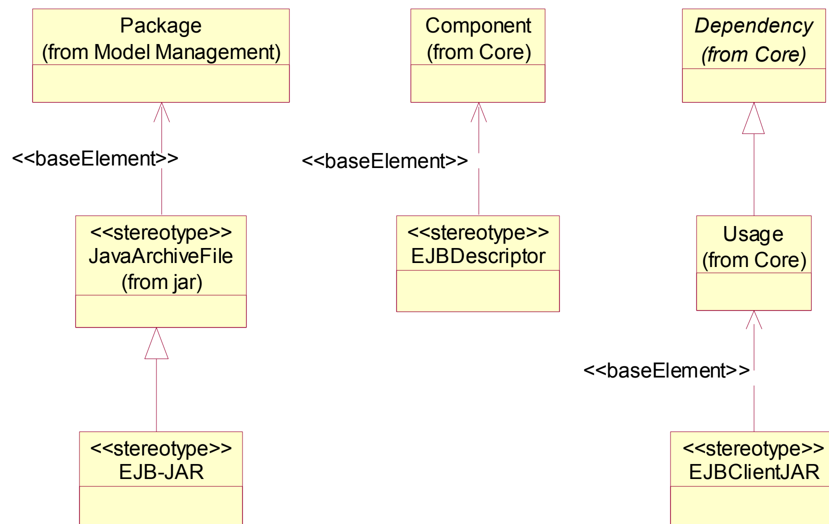

Figure 5-6. Stereotypes Defined in EJB Implementation Model

6 Rationale

This section describes the rationale used in developing this profile, examining process, notation, representation and implementation issues.

6.1 Process

This specification draws a distinction between analysis and design models, as defined by Jacobsen, et. al. [9]. An analysis model is an abstract description of a system's functionality, while a design model is a concrete description of a system's implementation. This definition suggests that analysis models should not depend on specific implementation technologies. In practice, analysis models are influenced by assumptions about implementation technologies, such as the use of an object oriented or component based paradigm.

Because this profile depends on a specific implementation technology, it is not suitable for analysis modeling, and does not attempt to provide analysis modeling constructs. Models based on this profile are design models.

Analysis models for EJB-based applications should be created using an implementation technology independent profile that supports functional descriptions of component based systems. A mapping from the analysis profile to this profile, introducing EJB-specific implementation information, can then be used to generate EJB-specific design models from analysis models.

A profile that can be used to create analysis models for EJB-based applications may be standardized pursuant to the Request For Proposals (RFP) for a UML Profile for Enterprise Distributed Object Computing[24] (EDOC) issued by the Analysis and Design Platform Task Force (ADPTF) of the Object Management Group (OMG). The RFP solicits mechanisms for modeling the security, transactional, persistence, packaging and deployment characteristics of enterprise-class component based systems.

6.2 Notation

Because this profile is intended to support the creation of design models of EJB-based implementations, it must permit the capture all of the information needed to describe EJB-based artifacts. This requirement competes with the need for notational simplicity to make models based on this profile easy to read and write.

For notational simplicity, one would elide the differences between the remote interface and the implementation class, permitting the use of a single lifeline for an enterprise bean in sequence diagrams. These differences are as follows:

1. The create/ejbCreate, remove/ejbRemove *and* find/ejbFind *methods have different* names*.*

2. The methods on the public *interfaces throw* java.rmi.RemoteException*.*

3. The home and remote interfaces inherit methods that are not delegated to the implementation class, such as getHome, getHandle, getPrimaryKey *and* isIdentical*.*

4. Some of the methods on the implementation class are not exposed through the home or remote interfaces, including methods defined by the EntityBean*,* SessionBean *and* SessionSynchronization *interfaces, and methods defined by the bean provider intended to be accessible only to objects that are local to the implementation of the bean.*

However, hiding these differences would prevent that capture of sufficient information to the automatic generation of the bean from a model, and would prevent the accurate description of interactions involving clients of the bean and objects local to its implementation.

To resolve this dilemma, this profile supports the definition of two views of an enterprise bean.

The external view models the bean as seen by its clients, in terms of only its home and remote interfaces and the relationship between them. The external view is simple and easy to manage.

[24] The RFP is publicly available at ftp://ftp.omg.org/pub/docs/ad/99-03-11.pdf.

The internal view models the bean as seen by the implementation class and objects local to the implementation class. In the internal view, each part of the bean is explicitly modeled, and the parts are collected into a Subsystem*. The internal view is significantly more complex than the external view:*

1. The description of a single enterprise bean requires the use of at least five model elements.

2. The configuration of the model elements is subject to numerous Constraints*.*

3. Many of the model elements carry multiple Tagged Values *that map to deployment descriptor elements.*

The internal view is not easy to manipulate without automation, but permits concise description of the bean, including collaborations between the implementation class and other objects that can not be captured using the external view.

In a development process based on iterative refinement, EJB-based systems can be modeled initially using only the external view, so that modelers can focus on the external interfaces of the beans, and the interactions between them. Subsequently, the internal view can be used to concisely describe their implementations.

In the external view, since every business method on the remote interface has a corresponding method on the implementation class, a single life line for the Classifier *that represents the remote interface can be used in sequence diagrams. In the internal view, sequence diagrams can describe the invocation of methods on the implementation class, using one life line for the home object, one for the remote object, and one for the implementation class.*

6.3 Representation

This section discusses the choice of UML metamodel elements used to represent EJB artifacts.

Though unfamiliar to most UML users, and not well supported by popular modeling tools, the UML Subsystem *is the appropriate construct for modeling the internal view of an EJB. According to the UML specification:*

"A subsystem is a group of model elements that represent a behavioral unit in a physical system. A subsystem offers interfaces and has operations. In addition, the model elements [in] a subsystem can be partitioned into specification and realization elements, where the former, together with the operations of the subsystem, are realized, i.e., implemented, by the latter. In the metamodel, Subsystem *is a subclass of both* Package *and* Classifier*. As such it may have a set of Features, which are constrained to be* Operation*s and Receptions."*

Two other UML metamodel elements were considered: Component*, which would seem to be the natural choice on the basis of its name, and Collaboration.*

As it turns out, the UML Component *is like an EJB in name only. According to the UML specification[25]:*

"A component represents a physical piece of implementation of a system, including software code (source, binary or executable) or equivalents, such as scripts or command files. As such, a Component *may itself conform to and provide the realization of a set of interfaces, which represent services implemented by the elements resident in the component. In the metamodel,* Component *is a child of* Classifier*. It provides the physical packaging of its associated specification elements."*

In other words, a UML Component *describes a physical artifact. This is illustrated by its standard* Stereotypes*: document, file, executable, library and table.*

The UML Collaboration captures more of the qualities of an EJB than a UML Component*. According to the UML specification:*

"A collaboration describes how an operation, or a Classifier *like a use case, is realized by a set of* Classifiers *and associations used in a specific way. The collaboration defines a set of roles to be played by instances and links, as well as a set of interactions that define the communication between the instances when they play the roles. In the metamodel, a Collaboration contains a set of* Classi*fierRoles and AssociationRoles, which represent the* Classifiers *and Associations that take part in the realization of the associated* Classifier *or* Operation*."*

[25] The notion of a UML Component as described by this document refers to the UML Specification version 1.3. At the time of writing, the UML 1.4 Revision Task Force (RTF) of the OMG's Analysis & Design Platform Task Force (ADPTF) has released a draft of the UML specification version 1.4 that modify the semantic of the UML Component: "A component represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces". This specification will be revised to take full advantage of these change to the UML.

A Collaboration is not a Classifier*, however, and so can not be instantiated. In addition, it does not physically contain the participating model elements. In the typical model, an additional* Package *would have to be provided to contain the pieces of each EJB.*

6.4 Relationships

Version 1.2 of the Java 2 Platform, Enterprise Edition (J2EE), which includes EJB 1.1, does not define mechanisms to create, destroy or manage relationships between enterprise beans or any other kind of class, except JavaBeans[26]. JavaBeans features are outside the scope of this specification, however, since they are not generally used in the construction of EJBs.

While it may be possible to use finder methods to implement some types of relationships, EJB 1.1 does not define a standard way to do this. Models containing relationships between EJBs or other Java classes therefore describe conditions that can be established or maintained only by proprietary or implementation dependent mechanisms.

Conversely, UML defines association end properties that imply the existence of mechanisms that are not explicitly defined by EJB 1.1. For example, aggregation on the target end of an association coupled with containment by value implies cascading delete semantics, a multiplicity of exactly one implies an existence constraint, a qualifier implies selective access to associated instances, and so on. Because the mechanisms required to implement these features are not defined by EJB 1.1, no mappings can be established for them by this specification.

In this profile, instance level associations between beans map to field members containing references to the remote interfaces of the related beans. In the case of a to-one relationship, the field member holds the remote interface of the related bean directly. In the case of a to-many relationship, the field member holds a reference to a Java collection type that holds references to the remote interfaces of the related beans. The class of the collection must be java.util.Collection or java.util.Set.

When forward engineering a to-many relationship from a UML model to Java-based implementation artifacts, information is lost because the class of the related objects is not captured by the syntax of the Java field declaration. During reverse engineering, a collection valued field member must therefore assumed to implement a collection of instances of java.lang.Object. In addition, when reverse engineering, it is not possible to tell whether the collection describes an attribute or an association end.

6.5 Implementation

This section describes the forward and reverse engineering transformations between UML model elements and Java artifacts. These descriptions are suggestive, not prescriptive, meaning that tools supporting this profile are not required to implement them, or to use the approach suggested here for round trip engineering.

6.5.1 Forward Engineering

This section describes the process of forward engineering from a UML model to Java-based artifacts. New artifacts are generated from an existing model. Java source files and directories are generated for elements defined in the model.

The basic unit of forward engineering is a UML Component stereotyped as a Java Class File. A Java language source file should be generated, containing Java language declarations for the model elements resident in the Component, using the mappings defined in the Java Design and Implementation Models. The file should have the same name as the Component, and may be given a tool or system specific extension, such as ".java". If the file already exists, its contents should be updated to reflect the model elements resident in the Component.

[26] Version 2.0 of the EJB Specification, developed and published under JSR-000033, defines an architecture for container managed instance level relationships between entity beans through their local interfaces.

The hierarchy of Packages in the model enclosing the Component, excluding the top level Package, maps to the Java Package for the Java Classes and Interfaces generated from the Component, as specified by the Java Design Model. In addition, it maps to a directory tree within the file system. Every Package in the hierarchy maps to a directory in the tree. The simple name of each Package maps to the simple name of the corresponding directory. Directories that do not already exist should be generated. The Java language source file generated from the Component should be placed in the innermost directory. The name of the directory where the tree is rooted may optionally be added to the CLASSPATH[27] used for reverse engineering.

UML Packages, Classes and Subsystems stereotyped as Java or EJB based constructs may also be forward engineered, as follows. When a Package is forward engineered, all of the Components contained directly or indirectly by the Package should be forward engineered. In addition, the Classes contained directly or indirectly by the Package should be forward engineered as follows. For every Class stereotyped as a Java Class or Interface, the associated Component should be forward engineered. If there is no associated Component, a Component that represents a Java Class File for the Class should be generated and forward engineered. Finally, all of the Subsystems contained directly or indirectly by the Package should be considered, and all of the Components defined by a Subsystem stereotyped as EJB Entity or Session Bean should be forward engineered. Classes and Subsystems may also be forward engineered explicitly, outside the context of a Package, using these same mechanisms.

If a Package is used as the basis for forward engineering, an XML file implementing the EJB Deployment Descriptor should also be generated from the model elements within the Package, as specified by the EJB Design Model.

It is not possible to meaningfully generate a JAR by forward engineering. A JAR holds compiled Java Class Files. These files can not be meaningfully produced until method bodies have been provided and the source code prepared for compilation, either manually, or by mechanisms based on extensions to this profile.

[27] The CLASSPATH is a system property that describes the set of locations searched by the standard class loader to find classes required by a Java language program. The mechanism used to define the CLASSPATH for forward or reverse engineering is tool specific.

6.5.2 Reverse Engineering

This section describes the process of reverse engineering from Java-based artifacts to a UML model.

The basic unit of reverse engineering is a Java Class File that resides either in a directory or in a JAR.

The Java Package declaration defined in the Java Class File maps to a UML Package hierarchy, as specified by the Java Design Model. This hierarchy should contain the model elements generated from the Java Class File by the reverse engineering process, and should be created if it does not already exist. All of the Java and EJB logical constructs declared in the Java Class File should be reverse engineered using the mappings defined by the Java and EJB Design Models.

A UML Component that represents the Java Class File should be generated in the innermost UML Package, using the mapping specified by the Java Implementation Model. The Component should have the same name as the Java Class File, without an extension. If a Component by that name already exists in the Package, then it should be updated to reflect the contents of the Java Class File.

6.5.2.1 Reverse Engineering A Directory

When reverse engineering a directory, all of the Java Class Files contained directly or indirectly by the directory should be reverse engineered. The CLASSPATH may be used to locate directories containing Java Class Files to be reverse engineered.

6.5.2.2 Reverse Engineering a JAR

Reverse engineering a JAR is similar to reverse engineering a directory. All of the Java Class File contained by the JAR should be reverse engineered. In addition, the JAR should be reverse engineered to produce a hierarchy of UML Packages, as specified by the Java Implementation Model.

If the JAR is an EJB-JAR, then the EJB Deployment Descriptor should be reverse engineered, as specified by the Java Implementation Model. The Component generated from the EJB Deployment Descriptor should be placed in a Package named META-INF at the top level of the «EJB-JAR» Package.

7 Examples

7.1 ITSO Bank Example

The diagrams in this example are based on the IBM ITSO Bank model. This model is used in many of the RedBooks published by IBM to illustrate best practices using IBM runtimes and tools for Enterprise JavaBeans, and will therefore be familiar to many readers.

The model has four main objects: Bank, BankAccount, Customer, and TransactionRecord. A Bank has many BankAccounts, which may be one of three types: CheckingAccount, SavingsAccount and CorporateAccount. Each transaction executed on a BankAccount is recorded as a TransactionRecord. Each BankAccount has an owner: a Customer, which has its own authentication information. The CorporateAccount is used to refer to corporations with which the customer does business. For example, the gas company would be a corporate account. Figure 7-1, below, shows the core of the analysis model for the ITSO Bank example.

Figure 7-1. ITSO Bank Example Analysis Model – Core Classes

Figure 7-2, below, shows supporting classes for the analysis model for the ITSO Bank example.

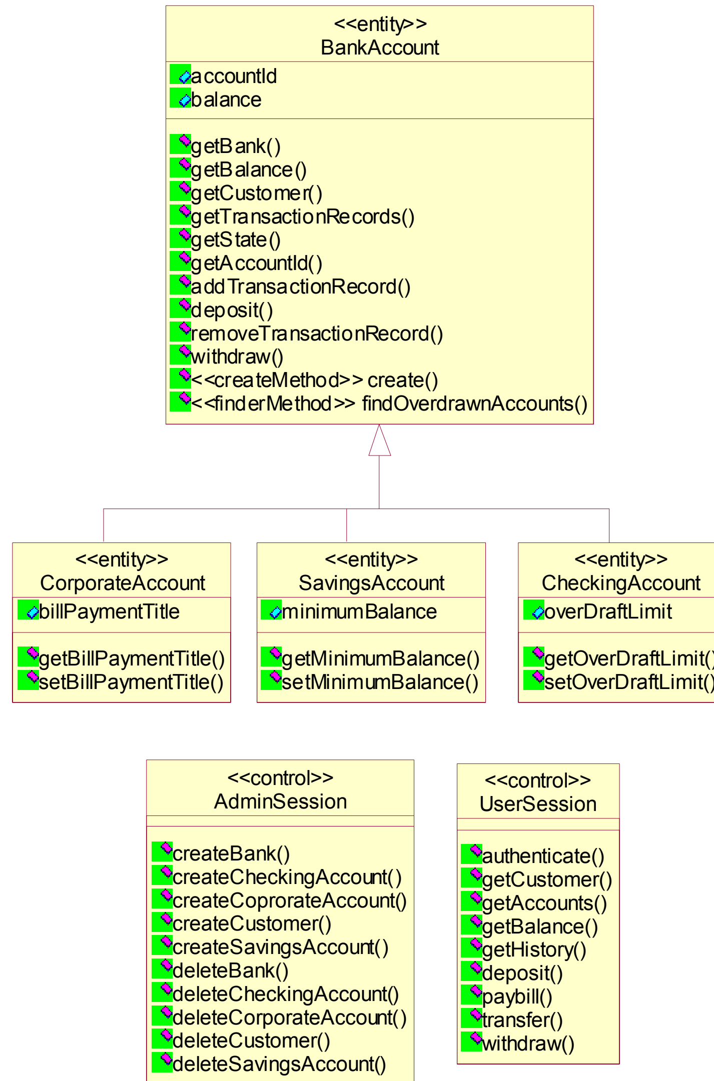

Figure 7-2. ITSO Bank Example Analysis Model

Three views can be created from the analysis model using the profile to describe an EJB-based implementation of the model: an external view, an internal view, and a implementation view. The external view, in Figure 7-3, above, shows the parts of EJBs visible to client programmers: the home and remote interfaces, and the primary key classes. For the sake of brevity, only the User session and Customer entity are shown.

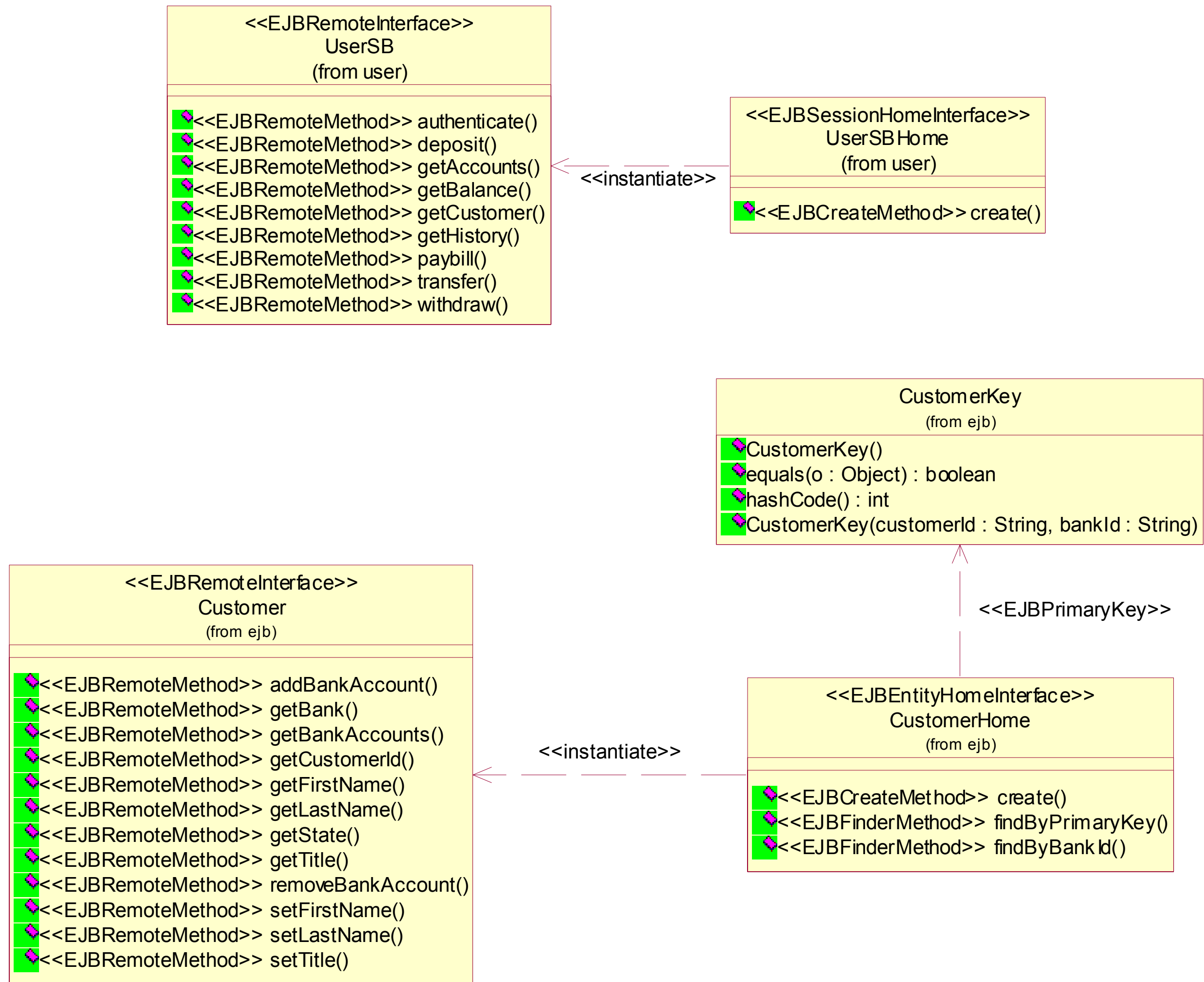

Figure 7-3. ITSO Bank Example - External View

Figure 7-4, below, shows the internal view of the Customer EJB. In the internal view, an EJB is modeled as a Subsystem, which has specification and implementation compartments. For Customer, the specification is expressed by Customer and CustomerHome, and the implementation by CustomerBean and CustomerKey.

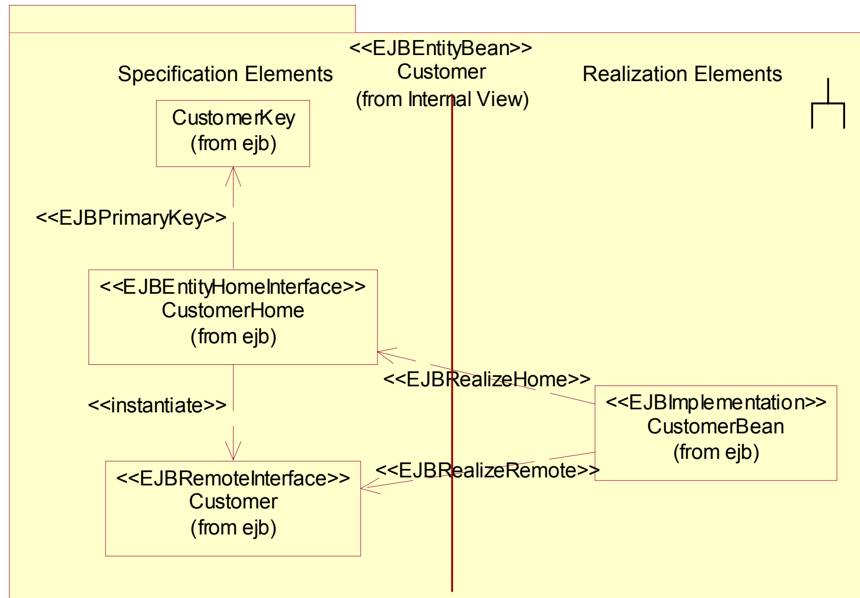

Figure 7-4. ITSO Bank Example Customer Entity - Internal View

Figure 7-5, below, shows that the Bank EJB will reference to the home of the Customer EJB through EJB References. The references are drawn between the two Subsystems in order to simplify the overall representation.

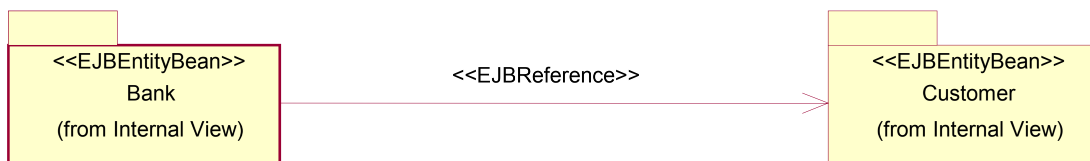

Figure 7-5. ITSO Bank Example - Internal view - EJB References

Figure 7-6, below, shows that the security role name bankManager is allowed to invoke all the operations on the Customer entity bean. The Customer bean will use this reference in its code.

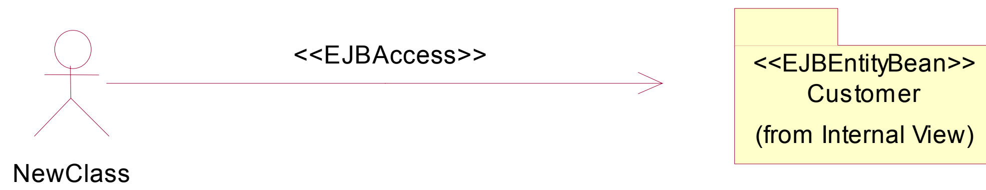

Figure 7-6. ITSO Example - Internal View - Security Role References

Figure 7-7, below, shows the implementation details and exposes the mappings between the specification and realization elements.

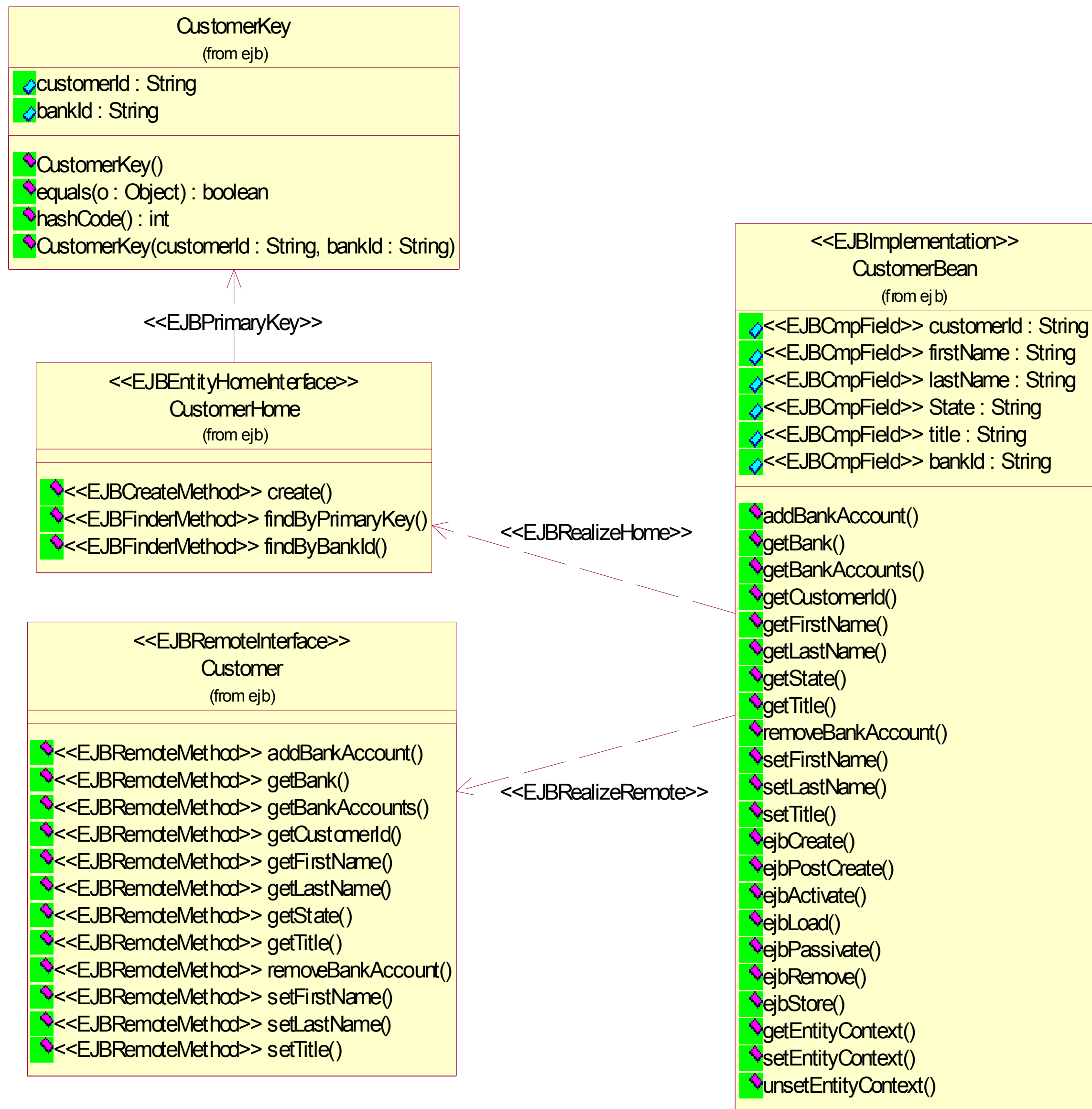

Figure 7-7. ITSO Bank Example Customer Entity - Internal View Detail

To complete the example, Figure 7-8, below, shows the physical artifacts that comprise the EJB-based implementation of the model. These include an EJB-JAR file, which contains a deployment descriptor named ejb-jar.xml and the implementation artifacts for the model. For the sake of brevity, the implementation artifacts are shown only for the Customer EJB. The implementation model also includes a client EJB-JAR file, which contains the interfaces and stubs used by a Java client to access the EJBs.

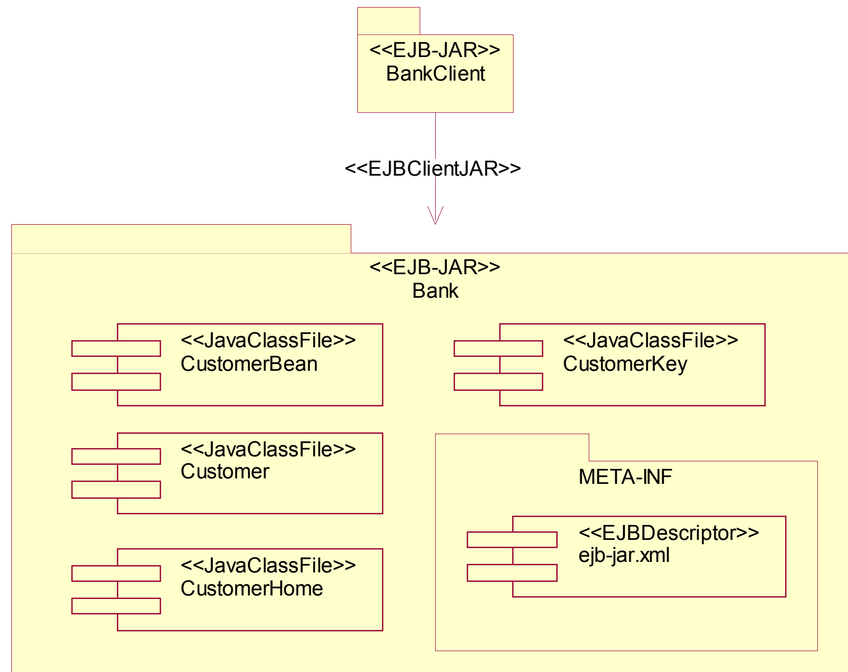

Figure 7-8. ITSO Bank Example Customer Entity - Implementation View

8 Related Documents

This specification refers to the following documents.

1. ***Java 2 Platform, Enterprise Edition Specification, Version 1.2*** *(J2EE Specification). Copyright 1999, Sun Microsystems, Inc. Available at* http://java.sun.com/j2ee/docs.html.
2. ***Java 2 Platform, Standard Edition, v1.2.2 API Specification*** *(J2SE Specification). Copyright 1993-1999, Sun Microsystems, Inc. Available at* http://java.sun.com/products/jdk/1.2/docs/api/index.html.
3. ***Enterprise JavaBeans Specification, Version 1.1*** *(EJB Specification). Copyright 1998, 1999, Sun Microsystems, Inc. Available at* http://java.sun.com/products/ejb.
4. ***Java Language Specification, Second Edition*** *(Java Language Specification). Copyright 2000, Sun Microsystems, Inc. Available at* http://java.sun.com/docs/books/jls/html/index.html.
5. ***Meta Object Facility Specification, Version 1.3*** *(MOF Specification). Copyright 1997-2000, Object Management Group, Inc., et. al. Available at* ftp://ftp.omg.org/pub/docs/formal/00-04-03.pdf.
6. ***Unified Modeling Language Specification, Version 1.3*** *(UML Specification). Copyright 1997-2000, Object Management Group, Inc., et. al. Available at* ftp://ftp.omg.org/pub/docs/formal/00-03-01.pdf.
7. ***XML Meta-Data Interchange, Version 1.1*** *(XMI Specification). Copyright 1998-2000, Object Management Group, Inc., et. al. Available at* ftp://ftp.omg.org/pub/docs/formal/00-06-01.pdf.
8. ***Java Metadata Interface Specification*** *(JMI Specification). Java Community Process (JCP) Specification JSR-000040. Available at* http://java.sun.com/aboutJava/communityprocess/final/jsr040/index.html.
9. Jacobson, et al., *The Unified Software Development Process*, Addison-Wesley, 1999.

9 Revision History

9.1 Changes Since Participant Draft Candidate 1

1. Interface Stereotypes now apply to Interface, instead of specializing Interface and applying to Class.
2. The specification was migrated from EJB 1.1 Public Draft 2 to EJB 1.1 Public Release 2.
3. The implementation class is no longer constrained to be sequential, and is constrained to be not active, and to have operations that are not guarded.
4. An abstract stereotype was introduced to describe the elements common to javax.ejb.SessionBean and javax.ejb.EntityBean.
5. A section was added to describe the rational for the approach used to model components.
6. The profile was extended to support physical models of EJB-based artifacts.
7. EJB references were mapped to navigable Classifier scoped association ends.
8. Associations were used to model field members containing references to the remote interfaces of the related beans.
9. The definition of the pre-defined common model elements was clarified.
10. The profile was extended to support logical and physical models of the subset of Java required to support EJB.
11. An example containing both UML and EJB constructs was added to illustrate the application of the mapping.
12. Assumptions regarding analysis, design, and logical and physical modeling were made explicit.
13. The definition of Stereotype was replaced with a more complete and accurate definition from the UML 1.3 specification.
14. The language from the UML 1.3 specification was used to provide a formal definition of a UML profile.
15. The OMG MOF specification was added to the list of related specifications.
16. XMI 1.1, which is used to produce the UML 1.3 XML DTD that defines the structure of UML 1.3 constructs, was added to the definition of the target platform.
17. The specification now describes how sequence diagrams based on the UML Profile for EJB should be constructed.
18. A Stereotype was added to identify the bean implementation class as convenience for development tools that must differentiate among the classes that may appear within the subsystem that represents an enterprise bean.
19. The relationship of this specification to the Java MetaData Interface was defined.
20. The applicability of Stereotypes for methods on the home and remote interfaces was clarified.
21. The relationship between the public interfaces and the implementation class was clarified.
22. The disposition of Java and EJB constructs that are pre-defined common model elements, and therefore not explicitly defined, was explained.
23. The names of all the EJB stereotypes have been updated to be prefixed by EJB.
24. The names of all the Java stereotypes have been updated to be prefixed by Java.
25. The names of all the EJB Tagged Values have been updated to be prefixed by EJB.
26. The names of all the Java Tagged values have been updated to be prefixed by Java.
27. The chapter on UML Descriptor has been completed.
28. The chapter on the Rationale of the specification has been completed.
29. The virtual metamodel has been introduced.

9.2 Changes Since Participant Draft Candidate 4

1. The optional stereotype <<JavaPackage>> has been removed.
2. The constraints that applied to a JavaPackage have been removed.
3. The Logical model is now a Design model.
4. The Physical model is now an Implementation model.
5. The tag value JavaStricfp has been introduced.
6. The tag value <<EJBPrimaryKeyField>> has been replaced by a stereotype named <<EJBPrimaryKeyField>>.
7. The abstract stereotype <<EJBHomeMethod>> has been removed.
8. The stereotype <<EJBReference>> has been introduced.
9. A new stereotype <<JavaInterface>> has been introduced.
10. The definition of the Tag value <<EJBRoleNames>> has been updated.
11. The stereotype <<EJBCmpField>> has been introduced.
12. The stereotype <<EJBAccess>> has been introduced.
13. The mapping for an EJB Security Role reference has been clarified.
14. The constraints on an EJB Entity Bean in the internal view have been updated to reflect the changes made with EJBCmpField and EJBPrimaryKeyField.
15. The constraints on an EJB Primary Key Class in the internal view have been updated to reflect the changes made with EJBPrimaryKeyField.
16. The mapping for an EJB Reference has been clarified.
17. The note referencing the UML 1.4 specification has been updated.