# Detection of Java EE EJB Antipattern Instances using Framework-Specific Models

by

Matthew Stephan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2009

© Matthew Stephan 2009

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Matthew Stephan

## Abstract

Adding flexibility to a process or technology often comes with a price. This holds true in the case of the amendments made to Java EE platform to upgrade to version 5. Java EE 5 allows Enterprise Java Bean (EJB) developers the ability to configure EJBs via Java 5 annotations, through XML deployment descriptors, or through a combination of both. While this adds flexibility to the EJB configuration process, it also comes with the price of an EJB project's stakeholder not being able to ascertain the current configuration of an EJB project until runtime, due to the multiple sources of configuration and the complex overriding rules. Furthermore, to detect errors in configuration or perform antipattern instance detection it is clearly beneficial to have a representation of an EJB project that accurately represents the current configuration of the system.

This thesis first presents an EJB Framework Specific Modeling Language (FSML) that formalizes the EJB domain's specific components in the form of a cardinality-based feature model. By having such a model and using and extending the existing FSML infrastructure, one retrieves a Framework Specific Model (FSM) through reverse engineering that represents all the information from the various sources of EJB configuration. By analyzing this FSM, we can create another model that represents the resolved configuration of an EJB project. We employ model filtration to highlight specific sources of configuration. We then use open-source and custom EJB projects to evaluate the EJB FSML and the resolved model.

Models admit antipattern instance detection. This thesis presents two methods for running antipattern instance detection on an EJB project using existing EJB antipatterns in literature: 1) queries in Java that execute against the resolved configuration model; and 2) queries written in .QL, an object-oriented query language, against the EJB project's source code. We compare these two techniques qualitatively and propose a new approach based on this comparison that entails modeling the antipatterns and their symptoms within an FSML model declaratively.

We then discuss possible extensions to the work presented in this thesis including the ability to support round-trip engineering for the EJB domain, the detection of new EJB antipatterns, and techniques that account for the strength of symptoms within the context of their respective antipatterns.

## Acknowledgements

I would like to give my thanks to the ones who have contributed to this thesis.

Firstly, I want to thank my supervisor Professor Krzysztof Czarnecki. He provided me enough freedom to allow me to explore various topics independently but at the same time provided ideas, ensured I stayed on track, and kept my vision properly scoped. He was always there to challenge me to go farther and deeper into the work and never let me get complacent, which is something I will carry with me as I continue my studies. I am grateful that I was able to work with him early in my academic career so that I can take the lessons and advice he gave me and apply them in both my professional and personal life.

Next, I want to thank my colleague Michał Antkiewicz, the mind behind FSMLs. Without him my masters would have been a much more difficult journey, not to mention this thesis topic would be completely different. No matter how busy he was or what I was working on, he always provided a helping hand with a smile and with patience. He is a model researcher and person and one that I will attempt to emulate during the next stage of my academic career.

I would also like to thank Professor Patrick Lam and Professor Sebastian Fischmeister for agreeing to be my thesis readers. I can only imagine how busy they are and for them to be willing to do this for me is something that I am extremely grateful for.

I am grateful to my entire family for always checking up on me and supporting me in every way possible.

Lastly, I would be remiss if I did not thank my fraternity of friends for ensuring that I do not spend my entire time reading papers or on the computer doing work and showing me that balance is important to one's happiness.

## Dedication

I would like to dedicate this thesis to my family and to my fraternity of friends.

# Contents

**List of Tables** ix

**List of Figures** xi

**1 Introduction** 1

1.1 Research Contributions . . . 2

1.2 Thesis Organization . . . 3

**2 Background Material** 5

2.1 Framework-Specific Modeling Languages . . . 5

2.1.1 Reverse Engineering with FSMLs . . . 6

2.1.2 Notation . . . 7

2.2 Enterprise JavaBeans Architecture . . . 7

2.3 Antipatterns . . . 11

2.3.1 Enterprise JavaBeans Antipatterns . . . 12

2.4 .QL: An Object-Oriented Query Language . . . 13

**3 Engineering the EJB FSML** 15

3.1 Iteration 1: Information from All Sources . . . 15

3.1.1 Inception . . . 15

3.1.2 Elaboration . . . 16

3.1.3 Construction . . . 18

3.1.4 Transition . . . 22

3.2 Iteration 2: EJB 2.1 Projects . . . 23

3.2.1 Inception . . . 23

3.2.2 Elaboration . . . 23

3.2.3 Construction . . . 24
3.2.4 Transition . . . 25
3.3 Iteration 3: Facilitate Antipattern Instance Detection . . . 26
3.3.1 Inception . . . 26
3.3.2 Elaboration . . . 26
3.3.3 Construction . . . 28
3.3.4 Transition . . . 31
3.4 Evaluation . . . 32
3.4.1 Sample EJB Projects Tested . . . 32
3.4.2 Threats to Correctness of FSML . . . 38
3.4.3 Further Limitations of the EJB FSML . . . 38
3.5 Discussion . . . 38

**4 Providing a Resolved Model of an EJB3 Project 40**
4.1 Model Filtration of EJB FSM . . . 40
4.2 Model Analysis of EJB FSM . . . 42
4.2.1 Meta Model of Resolved Model . . . 43
4.2.2 Model Analysis . . . 43
4.3 Evaluation . . . 47
4.3.1 Sample EJB Projects Tested . . . 47
4.4 Discussion . . . 47

**5 Detecting Antipattern Instances 49**
5.1 Existing Antipatterns . . . 49
5.2 Detecting Antipattern Instances via Resolved EJB Model . . . 52
5.2.1 Categorizing Antipattern Symptoms for Analysis . . . 52
5.2.2 EJB Antipattern Meta model . . . 53
5.3 Detecting Antipattern Instances via .QL . . . 54
5.4 Discussion . . . 56
5.4.1 Comparison of Techniques . . . 57
5.4.2 Combination of Techniques . . . 59
5.5 Evaluation . . . 61
5.5.1 Detection Results . . . 61
5.5.2 Limitations . . . 62

**6 Future Work** **64**

6.1 Round-Trip and Refactoring Capabilities . . . . . . . . . . . . . . . . 64

6.2 Detecting New Antipatterns . . . . . . . . . . . . . . . . . . . . . . . 65

6.3 Compare Multiple Projects . . . . . . . . . . . . . . . . . . . . . . . . 65

6.4 Techniques that Account for Strength of Symptoms . . . . . . . . . . 66

6.5 New Detection Technique on Other Frameworks . . . . . . . . . . . . 66

**7 Related Work** **68**

7.1 Java EE Development Tools . . . . . . . . . . . . . . . . . . . . . . . 68

7.2 Static Code Analysis Tools . . . . . . . . . . . . . . . . . . . . . . . . 68

7.3 Dynamic Analysis for EJB Antipatterns . . . . . . . . . . . . . . . . 70

**8 Conclusion** **71**

**APPENDICES** **73**

**A Equivalent Ecore Models for EJB FSML Feature Models** **74**

A.1 Iteration 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74

A.2 Iteration 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81

A.3 Iteration 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87

**B Equivalent Ecore Models for Resolved Meta Model Feature Models** **94**

**C Extended J2EE library for Semmle .QL** **97**

**References** **99**

# List of Tables

2.1 Cardinality-based feature modeling notation . . . . . . . . . . . . . . 7

3.1 Additional mapping types added for Iteration 1 . . . . . . . . . . . . 22

3.2 Additional mapping types added for Iteration 3 . . . . . . . . . . . . 31

3.3 EJB Projects Used For Testing . . . . . . . . . . . . . . . . . . . . . 33

5.1 Antipattern Instances detected in Open-Source Projects . . . . . . . . 62

# List of Figures


1.1 Overview of Process Yielding Current Configuration Model . . . . . 3

2.1 Excerpt from Workbench Part Interactions FSML . . . . . . . . . . . 6
2.2 Scope of EJBs within a Java EE System . . . . . . . . . . . . . . . . 8
2.3 Annotated JavaBean Class . . . . . . . . . . . . . . . . . . . . . . . . 9
2.4 Excerpt from session-BeanType from EJB 3.0 Specification . . . . . 10
2.5 Layout of an EJB Jar file . . . . . . . . . . . . . . . . . . . . . . . . 10
2.6 Example of .QL Query . . . . . . . . . . . . . . . . . . . . . . . . . 13
2.7 Example of a .QL Class definition . . . . . . . . . . . . . . . . . . . 14

3.1 Overall Structure for Feature Model . . . . . . . . . . . . . . . . . . 17
3.2 Iteration 1-Annotation Information Feature Model . . . . . . . . . . 18
3.3 Iteration 1-Deployment Descriptor Information Feature Model . . . 19
3.4 Iteration 2-Additional Features . . . . . . . . . . . . . . . . . . . . 25
3.5 Sample Deployment Descriptor EJB References . . . . . . . . . . . 27
3.6 Iteration 3-Features for Clients and Entity Users . . . . . . . . . . . 29
3.7 Iteration 3-Additional Antipattern-Specific Features . . . . . . . . . 30
3.8 FSM yielded from eMal Payment System Project . . . . . . . . . . 34
3.9 FSM yielded from Time to Work Project . . . . . . . . . . . . . . . 35
3.10 FSM yielded from Redwood Web Log Mining Project . . . . . . . . 36
3.11 FSM yielded from Custom EJB Project . . . . . . . . . . . . . . . . 37

4.1 Information from Annotation View . . . . . . . . . . . . . . . . . . . 41
4.2 Information from Deployment Descriptor View . . . . . . . . . . . . 42
4.3 Feature Model of Resolved Meta Model . . . . . . . . . . . . . . . . 44
4.4 Bean Component of Resolved Meta Model . . . . . . . . . . . . . . 45
4.5 Overview of Model Analysis/Merging Process . . . . . . . . . . . . 46

4.6 Example of a Resolved Session Bean . . . . . . . . . . . . . . . . . . 48

5.1 Bloated Session Symptoms in Predicate Logic . . . . . . . . . . . . . 50
5.2 Data Cache Symptom in Predicate Logic . . . . . . . . . . . . . . . . 50
5.3 Fragile Links Symptom in Predicate Logic . . . . . . . . . . . . . . . 50
5.4 Sessions a Plenty Symptoms in Predicate Logic . . . . . . . . . . . 51
5.5 Transparent Facade Symptom in Predicate Logic . . . . . . . . . . 51
5.6 Thin Session Symptoms in Predicate Logic . . . . . . . . . . . . . . . 52
5.7 Meta Model for Discovered EJB Antipatterns . . . . . . . . . . . . 54
5.8 Example of Antipattern Instances in Antipattern View . . . . . . . 55
5.9 .QL Queries for Bloated Session Symptoms . . . . . . . . . . . . . . 57
5.10 .QL Query for Data Cache Symptom . . . . . . . . . . . . . . . . . . 57
5.11 .QL Queries for Sessions-A-Plenty Symptoms . . . . . . . . . . . . . 57
5.12 .QL Queries for Thin Session Symptoms . . . . . . . . . . . . . . . . 58
5.13 .QL Queries for Transparent Facade Symptom . . . . . . . . . . . . 58
5.14 FSML elements for Bloated Session Antipattern . . . . . . . . . . . 60
5.15 FSML elements for Data Cache Antipattern . . . . . . . . . . . . . 60
5.16 FSML elements for Thin Sessions Antipattern . . . . . . . . . . . . 61

# Chapter 1

# Introduction

Adding flexibility to an existing mechanism or technology often comes with a cost. This holds true in the case of the amendments made to the Java 2 Enterprise Edition 1.4 (J2EE 1.4) platform to create the current version, Java Enterprise Edition 5 (Java EE 5). Specifically, the Enterprise JavaBean architecture in J2EE 1.4 requires developers configure properties of Enterprise Java Bean(EJB)s and other system properties via an extended markup language (XML) file known as a deployment descriptor. Java EE 5, specifically the EJB 3.0 specification [16] for the EJB 3.0 architecture, provides the additional flexibility of allowing configuration of EJBs in the deployment descriptor, as done in J2EE 1.4; via Java 5 annotations; or by using a combination of both and following complex merge and overriding rules. While this benefits Java EE developers because configuration is more flexible, it is much more difficult to understand the current configuration of a Java EE system during development because the two sources of configuration, the overriding rules, and default values given to certain properties must be considered. Developers and others involved an EJB project will know the final result of their configuration only once the project is deployed and executed, which involves a large investment of time and effort.

Furthermore, one issue that Object-Oriented Framework developers and maintainers, such as those that work on Java EE projects, should be concerned with is the notion of Antipatterns. Software Antipatterns, contrary to patterns, are commonly found errors in software projects that arise from any number of factors including lack of understanding or misunderstanding of a framework's application programming interface (API), the need for a quick fix, incorrect documentation, or many others [10]. There is also the notion of project or organisation-specific violations, which are violations that are relevant only in the context of a specific project or organisation. Many times, these can be detected statically by looking for indicators within a project's artifacts. In the context of Java EE projects, however, it would be difficult and less useful to perform static antipattern detection, either manual or automatic, without consulting the current configuration of a Java EE system, that is, it would be difficult without having some representation of the system that accounts for the various sources of configuration and other factors

mentioned earlier. This problem of determining the current/resolved configuration at development time of a Java EE project and detecting antipatterns on it was posed by Rational engineers during the IBM Rational/WebSphere University Day at CAS Toronto in September 2007 [20].

One mechanism that has been shown to assist with understanding Object-Oriented Frameworks, like Java EE, is the use of a *Framework-Specific Modeling Language* (FSML). FSMLs are created by a framework expert and formalize framework concepts. Furthermore, once formalized, the FSML generic infrastructure defined in [1] allows for reverse engineering of a framework's artifacts into a framework specific model (FSM) that represents a specific instance of the framework. This paper outlines the creation of such an FSML for the Java EE EJB framework. This FSML facilitates the construction of a model that represents the current configuration of the system at development time. Figure 1.1 provides a bottom-to-top view of this process. Reverse engineering through the FSML infrastructure using the EJB FSML produces the EJB FSM and we analyze this FSM, represented by the cloud within the figure, to present the current configuration. Furthermore, this paper shows that we can detect antipattern instances within an EJB project by analyzing this current configuration model. We compare this method of detecting antipatterns against a solution using the .QL object-oriented query language, and, based on the experiences with both methods, we suggest a new method that entails modeling the antipatterns within the FSML declaratively.

## 1.1 Research Contributions

The following are novel contributions made by this thesis:

- A Java EE EJB framework specific modeling language. This FSML expresses a framework specific model that represents the EJB properties configured from the various sources of configuration. This FSM then undergoes analysis to create another model that represents the configuration at development time and provides traceability to the configuration sources. The analysis takes into account the merge and overriding rules. Furthermore, we present other models generated from analysis on the FSM that showcase the information obtained from the deployment descriptor alone or Java annotations alone.

- Two techniques for static EJB antipattern detection. The first is queries formulated against the FSM. The second is queries written in .QL, an object-oriented query language, on the project's code. Based on the experiences of creating and working with these two approaches, we propose a new approach that entails modeling the antipatterns within an FSML declaratively.

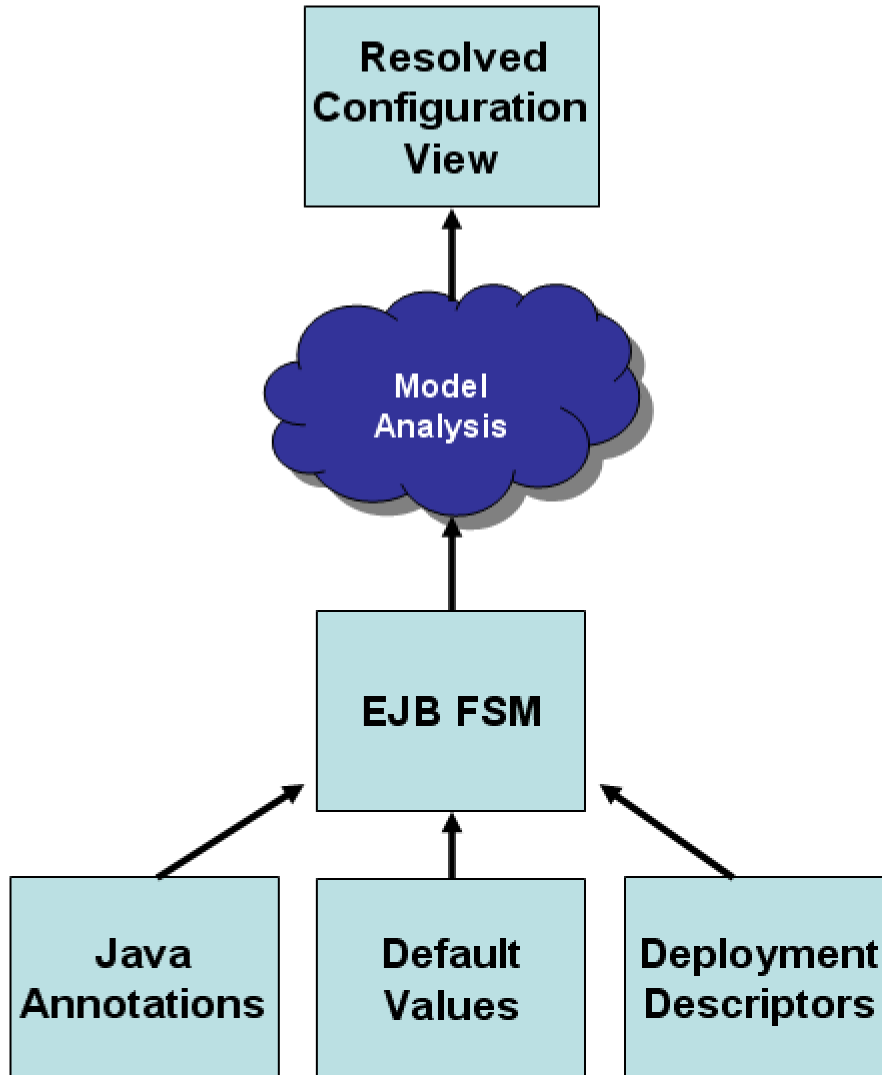


Figure 1.1: Overview of Process Yielding Current Configuration Model

## 1.2 Thesis Organization

The remainder of this thesis begins with Chapter 2, which presents the necessary background material required to appreciate the content of this thesis, notably, the relevant information on Framework-Specific Modeling Languages, the Enterprise JavaBeans Architecture, Antipatterns, and the .QL query language. Chapter 3 presents the EJB FSML created in three separate sections, with each section containing the elements and information corresponding to each iteration within the development process of the EJB FSML. Chapter 4 discusses the analysis that we perform on the FSM expressed from the EJB FSML to achieve a resolved configuration model/view of an EJB project. Chapter 5 then continues by discussing the analysis of such a model to present existing EJB antipattern instances of EJB antipatterns found in literature. Chapter 6 and Chapter 7 discuss future

work and related work, respectively, and the thesis concludes in Chapter 8 with some concluding remarks.

# Chapter 2

# Background Material

The following chapter provides background information on Framework-Specific Modeling Languages in 2.1; the Enterprise JavaBeans Architecture in 2.2; Antipatterns in 2.3, including Enterprise JavaBeans antipatterns in 2.3.1; and the .QL object-oriented query language in 2.4. A person unfamiliar in any of these topics can understand the contents of the paper following this chapter.

## 2.1 Framework-Specific Modeling Languages

Object-oriented frameworks allow for developers to use framework components without being concerned with underlying component code. Rather, developers using object-oriented frameworks need be concerned only with the various ways of instantiating a framework. This is accomplished by creating completion code using the appropriate application programming interfaces [1]. One way of representing the different possible implementations of a framework is framework-specific models (FSM). A way of expressing these models, proposed in [1], is the use of *Framework-Specific Modeling Languages* (FSML). An FSML is developed by an expert well versed with the specific framework. The expert models declaratively the components and constraints that exist within that framework as well as all the points of variability among these components.

In order to express these components and variability, each FSML metamodel is modeled declaratively as a feature model, which is a modeling notation that is an established method for modeling commonality and variability [12]. Figure 2.1 taken from [1] provides a small excerpt from the Eclipse Workbench Part Interactions(WPI) FSML showcasing the basic components that comprise the Eclipse WPI framework. Interpreting this feature model, we see that a WPI project can contain, as indicated by the empty circle over a filled circle, zero to many *ViewPart*s or *EditorPart*s, both of which are inherited from *Part*. They contain an optional, as indicated by the empty circle, part identification string. A ViewPart component must implement, as indicated by the filled circle, the

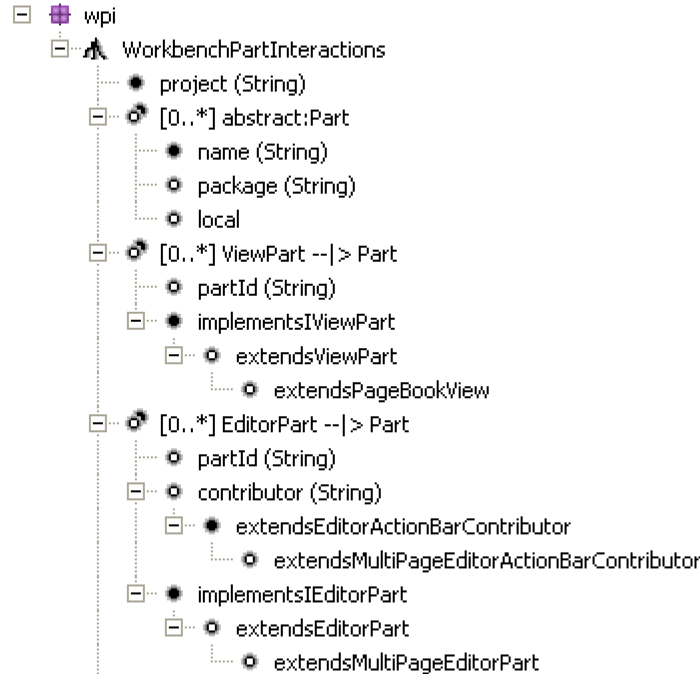


Figure 2.1: Excerpt from Workbench Part Interactions FSML

IViewPart interface and may optionally extend ViewPart. An EditorPart feature must implement the IEditorPart interface and may optionally extend EditorPart class. In the context of FSMLs, each of these features are related to Java completion code through a mapping type and definition, which is utilized by the FSML mapping interpreter [2]. So, for example, the feature *implementsIViewPart* has a mapping type *implementsInterface* and a mapping definition *IViewPart* that can be written as implementsInterface=IViewPart, which determines if the class being evaluated implements the IViewPart interface. We omit the mapping types and definitions from this figure but include them in later FSML feature models presented in this paper as text between ⟨angular brackets⟩ beside the specific features.

### 2.1.1 Reverse Engineering with FSMLs

One of the facilities provided by FSMLs and the FSML infrastructure is the ability to reverse engineer a framework specific model from existing completion code. Specifically, the infrastructure traverses the FSML's metamodel; then the infrastructure queries the completion code for features from the metamodel based on the mapping types and definitions; and, for each feature found in the code, the infrastructure instantiates the corresponding model element in the framework specific model [2]. Furthermore, each model element includes a traceability link that links the specific model element/feature to the completion code that it represents.

### 2.1.2 Notation

As noted in [1], we define FSMLs using cardinality-based feature models [13]. Specifically, the FSML we define in this thesis is an Eclipse Modeling Framework Ecore model [49] and we use the tool defined in [50] to render the Ecore model as a feature model. As such, the notation we use in this thesis is the slight variation of cardinality-based feature models defined in [50] and presented in table 2.1.

Table 2.1: Cardinality-based feature modeling notation

| **icon** | **feature model or configuration element** |
|---|---|
| | package |
| | root feature |
| | optional feature [0..1] |
| | mandatory feature [1..1] |
| | optional multiple feature [0..*] |
| [0..m] | optional multiple feature [0..m] |
| | mandatory multiple feature [1..*] |
| [n..m] | mandatory multiple feature [n..m], $n > 0$ |
| --\|> | inheritance |
| | feature group <1-1> (XOR) |
| <n..m> | feature group <n..m> |
| | grouped feature [0..1] |

## 2.2 Enterprise JavaBeans Architecture

Java Enterprise Edition (Java EE) is a large set of technologies that work together to allow organizations to easily develop, place in production, and manage multi-tier enterprise applications that are developed in the Java programming language [39]. The latest version of Java EE, Java EE 5, contains the latest version of the Enterprise JavaBeans (EJB) architecture, version 3.0. Figure 2.2, modified from [30], showcases the scope of EJBs within a Java EE project and, therefore, the scope of the work done in this thesis within the context of Java EE. An enterprise bean is a component inside a Java EE program that encapsulates business logic that operates on an enterprise's data [16]. The EJB architecture contains three types of enterprise bean objects: session beans, message-driven beans, and entities. Session beans are enterprise beans that perform a task for a single client. Session beans come in two variants, either stateful or stateless, where stateful session beans are transaction aware and can retain their state variables throughout multiple transactions. Message-driven beans are beans that act as message listening objects within an enterprise application and have the ability to receive messages asynchronously and react to them. Lastly, there are entities, which provide a Java object view of enterprise data that is stored in the system's

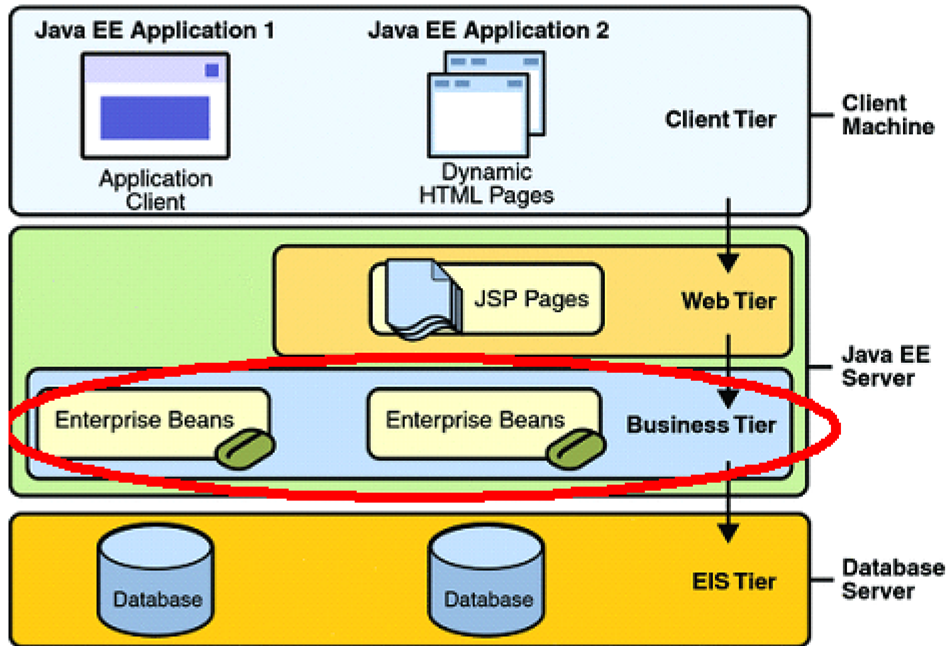


Figure 2.2: Scope of EJBs within a Java EE System

database. Session beans and Message-driven beans typically act on entities during transactions.

The EJB architecture version prior to 3.0, version 2.1, required that each JavaBean has an entry in an Extended Markup Language (XML) file called the deployment descriptor. Each entry specifies the properties of the JavaBean including its EJB name, the corresponding Java class for the bean, the type of bean, the bean's various EJB interfaces provided to clients, and more [17]. Furthermore, each JavaBean's Java class must implement the appropriate interface for its JavaBean type, for example, a session bean must implement the javax.ejb.SessionBean interface [17].

In the current version of the EJB architecture, version 3.0, developers configure a JavaBean in a variety of ways. Firstly, they can use Java 5 annotations on the JavaBean's Java class. As seen in Figure 2.3, many of the configuration properties for the JavaBean corresponding to the Java class `StatefulClass` are specified via annotations and the details of the annotations. Specifically, the annotation `@Stateful` indicates that the class annotated represents a stateful session JavaBean. The details inside the parameters indicate that the bean will have the ejb-name `StatefulClassAnnotationName` as entered via the `name` detail, a description of `StatefulClassEJBDescription`, and a mapped name of `StatefulMappedNameFromAnot`. In version 2.1, each of these were required elements of the JavaBean's deployment descriptor element. The `@Local` annotation indicates that this JavaBean's local business interface is the interface `DerivedLocalInterface1`.

In order to support backwards compatibility and interoperability, version 3.0

```
import interfacefolder.DerivedLocalInterface1;
import javax.ejb.Local;
import javax.ejb.Stateful;

@Stateful(name=``StatefulClassAnnotationName'',description=``
StatefulClassEJBDescription'',mappedName=``StatefulMappedNameFromAnot'')
@Local(DerivedLocalInterface1.class)
public class StatefulClass {
        //EJB class implementation goes here...
}
```

Figure 2.3: Annotated JavaBean Class

also allows JavaBeans to be configured via the XML deployment descriptor as was done in version 2.1 and earlier [39]. The XML schema for the version 3.0 deployment descriptor differs: it needs to account for the possibility in which configuration is done by both annotations and the deployment descriptor. The XML schema needs to allow for the XML elements that configure EJBs that can be configured by annotations be optional in the schema whereas they were mandatory before. For example, Figure 2.4 taken from [15] displays an excerpt of the 3.0 version of the XML schema. This excerpt presents the schema for a session bean, which has optional XML elements: `minOccurs=‘‘0’’` means the element is optional and the minimum number of occurrences defaults to one/mandatory if not specified. The `ejb-class` element indicates the fully-qualified name of the Java class implementing the bean. It is optional because if the current XML bean entry has its mandatory `ejb-name` XML element set to the same ejb-name specified in one of the Java classes via the annotation detail `name` then the ejb-class value is taken as the fully-qualified name of the annotated class [16]. Therefore, the `ejb-name` element in the bean's XML entry maps to the ejb-name specified via the name detail on the @Stateless,@Stateful, or @MessageDriven annotation on a Java class. In this case, the XML entry can be used in conjunction with Java annotations on the class to configure the bean. When EJB users use both sources of configuration, there are specific override rules and default values. For example, per [16], the deployment descriptor cannot override the value of the bean type (Stateless, Stateful, or Message-Driven) if it has been specified by a Java annotation. If the type is specified in the deployment descriptor, it has to be the same as the one specified in the annotation. [42] provides a listing of some overriding rules extracted from the EJB 3.0 specification.

If no match in the descriptor's ejb-name is found in any Java Class, then the `ejb-class` element must be present and must identify a Java class within the EJB project. In this case, the deployment descriptor must specify all the appropriate information because there is no configuration information arising from the annotations.

As shown in Figure 2.5 taken from [30], once EJB developers create and configure the enterprise Java beans, either through annotations or the deployment descriptor, the deployer places class files for the beans and their business interfaces within an EJB Java Archive (JAR) file. They include this with the appropriate

```
<xsd:complexType name="session-beanType">
        <xsd:sequence>
                <xsd:element name="ejb-name" type="javaee:ejb-nameType"/>
                <xsd:element name="mapped-name" type="javaee:xsdStringType"
                minOccurs="0"/>
                <xsd:element name="home" type="javaee:homeType" minOccurs="0"/>
                <xsd:element name="remote" type="javaee:remoteType" minOccurs="0"/
                >
                <xsd:element name="local-home" type="javaee:local-homeType"
                minOccurs="0"/>
                <xsd:element name="local" type="javaee:localType" minOccurs="0"/>
                <xsd:element name="business-local" type="javaee:fully-qualified-
                classType" minOccurs="0"
                        maxOccurs="unbounded"/>
                <xsd:element name="business-remote" type="javaee:fully-qualified-
                classType" minOccurs="0"
                        maxOccurs="unbounded"/>
                <xsd:element name="ejb-class" type="javaee:ejb-classType"
                minOccurs="0">
                <xsd:element name="session-type" type="javaee:session-typeType"
                minOccurs="0"/>
        </xsd:sequence>
</xsd:complexType>
```

Figure 2.4: Excerpt from session-BeanType from EJB 3.0 Specification

meta data including the deployment descriptor (ejb-jar.xml). The deployer then places this file within an Application Server that will host the enterprise application. During runtime, the EJB container interprets information from all the various sources of EJB configuration and makes the enterprise beans' interfaces available through dependency injection and name space lookups [16]. The EJB container performs this runtime resolution of EJBs and their configuration and is not available during development time.

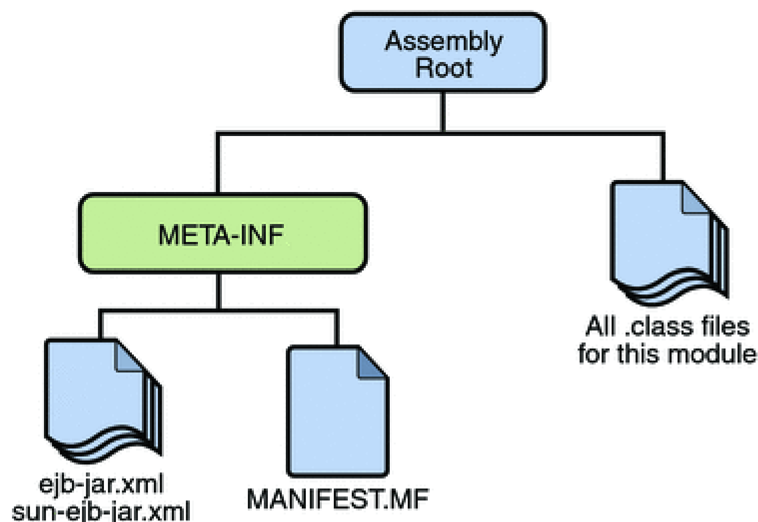


Figure 2.5: Layout of an EJB Jar file

## 2.3 Antipatterns

Much like software patterns are well-established methods and techniques to solve frequent problems or inherent issues within a specific context, such as Object-Oriented programming, software antipatterns are commonly made mistakes and non-optimal solutions that arise in a software project. As noted in [10], an antipattern must be a commonly repeated mistake that at first seems correct and/or beneficial but in fact is not. Each antipattern must have a solution associated with it that corrects the antipattern. Patterns distill allegedly useful and positive experiences to emulate. Antipatterns distill allegedly useful negative experiences to avoid and how to fix them.

Antipatterns are not limited to code and can be found at various levels within the field of Software Engineering. Brown and others introduce antipatterns that exist within the context of software configuration management [8]. Rather than antipatterns that exist within the design and architecture of software code, antipatterns deal with commonly occurring problems in the way that software is stored and changes are tracked and controlled. They later on, in [9], discuss antipatterns at an even higher level, specifically, those that are found within software project management. They identify three types or categories of software project management antipatterns: people-management antipatterns, technology-management antipatterns, and process-management antipatterns. Similarly, Kuranuki introduces Antipractices, which are antipatterns for the Extreme Programming (XP) development methodology that were discovered when observing XP projects [34].

Architecture is another level at which the notion of both patterns and antipatterns can be applied. Lauder and Kent examine a number of mid-sized legacy systems and discover 12 general antipatterns that deal with the architecture of the software [35]. Kis, on the other hand, outlines a number of architectural antipatterns that result in poor application security [32]. Even the relatively new architectural paradigm of Service Oriented Architecture (SOA) has antipatterns that are associated with it, as discussed by Kral and others [33]. Based on the authors' experiences with SOA and analogous projects, they identify 5 antipatterns that are crucial to look for within an SOA project.

Antipatterns have also been discovered that are applicable to specific areas of computer science. Multi-threaded applications have a number of antipatterns that can be detected statically [23]. Furthermore, attempts have been made to detect dynamic antipatterns in multi-threaded applications through the use of custom antipattern detectors represented by finite state machines [6]. Database antipatterns are elaborated on in [31] and Smith discusses antipatterns that relate specifically to software performance [47, 48]. Even the modeling of software, for example, object-oriented concept modeling, has antipatterns as elaborated on in [18].

This concept of antipatterns extends to frameworks: Having detailed application

programming interfaces (API)s, specifications, and documentation for a framework do not imply that the framework will be implemented or used correctly by framework implementors. Furthermore, framework components can be used in ways that are either unintended or ways that are unanticipated by framework developers when creating the framework. Thus, frameworks are also susceptible to antipatterns. Some examples include Ajax antipatterns [24] and JSP, Web Services, and Servlet antipatterns [19].

### 2.3.1 Enterprise JavaBeans Antipatterns

Enterprise JavaBeans are no exception. Although EJBs provide many capabilities, such as transaction support, security, portability, and more [39], they can often be misused in a variety of ways. In [19], Dudney et. al. provide a catalog of various types of Java EE antipatterns including EJB antipatterns. They choose to categorize the EJB antipatterns based on the type of bean that the antipattern is related to, namely Entity Bean antipatterns, Session EJB antipatterns, and Message-Driven Bean antipatterns. For each antipattern listed, they provide a general description, symptoms and consequences of the antipattern, refactorings to correct the antipattern, and real-world examples. Crawford and Kaplan discuss some more EJB antipatterns in a chapter within their book about J2EE design patterns [11]. The 3 antipatterns in this book consist of 1) using EJBs when not necessary, 2) using stateful beans when stateless beans are more appropriate, and 3) performance problems that arise when using remote EJBs incorrectly by requesting too many small pieces of data rather than combining multiple requests into larger ones [11]. These 3 antipatterns are not considered for the work presented in this thesis as their symptoms are not as statically-identifiable, concretely enumerated, and programmable as the ones presented by Dudney in [19] or are already accounted for.

Because FSMLs formalize and work with only the static implementation of a framework, the only EJB antipatterns that are considered are the ones that have symptoms that can be analyzed statically. So, antipatterns that have symptoms with dynamic behavior, such as symptoms that deal with system performance; or antipatterns that have symptoms that deal with social implications, such as reduced maintainability or project-team effectiveness, are unable to be discovered via the models expressed through the FSML. Thus, they are considered out of scope for the work done in this thesis but are addressed in the related work section discussing Parsons' dynamic EJB antipattern detection solution [40].

There also exists a notion of organisation-specific violations that are relevant and important only within a specific organisation or project within an organisation. Sources of these types of violations can be those related to regulations, company-wide policies, project-specific requirements, and others. Some EJB examples include naming, such as all EJBs must begin with “ejb” or all EJB names must end with a number; security with respect to configuration, for example, only certain

roles are allowed to be used in certain places, some roles are prohibited, or all configurations must be done via annotations; or security with respect to code, such as the overriding of certain methods being prohibited, limiting or restricting public visibility, and other restrictions. Because tools can detect statically many of these types of violations, an FSML should be able to detect them assuming there is some facility to specify them.

## 2.4 .QL: An Object-Oriented Query Language

.QL is an object-oriented query language that is implemented as an Eclipse plug in. This language supports queries of both Java code and XML files. As noted in [14], .QL stores Java projects as relational databases, thereby allowing standard SQL-like queries to be executed. .QL is based on Datalog [21], a logic programming language used for querying databases. Any Datalog program is comprised of 3 parts: a query, which is the predicate to be computed; intensional predicates, which are the relations to be computed that are specified by a user; and extensional predicates, which are the components stored in the database that are being evaluated [21]. Each .QL query is first translated into a Datalog program/query as an intermediate step and then these Datalog queries are implemented over the relational database(s) containing the Java project(s) [14]. Figure 2.6 provides an example .QL query taken from [14]. It identifies the case where a class implements the compareTo method but does not overwrite the equals method. This is usually indicative of a bug: to compare something, it is mandatory to have the ability to know if it is equal to something else or not. There are three parts to each .QL query; note that the order differs from that of an SQL query. The *from* clause indicates the variables that will be used throughout the query and the type of the variable, with Class being the type and c being the variable name in the example. The *where* clause contains predicates that use the variable and test certain conditions and, as a whole, must resolve to true for the current set of variables being evaluated for them to be matched or of interest. Lastly, the *select* clause indicates the return results of the query and can be the variable itself, properties of the variable, or any other data available during the query. These return results are returned in various output formats provided by the .QL plug in and appear in the order given in the select clause [14].

```
from Class c
where c.declaresMethod("compareTo")
        and
        not(c.declaresMethod("equals"))
select c.getPackage(), c
```

Figure 2.6: Example of .QL Query

The type *Class* in the from clause is an example of a .QL class and can have certain properties and queries associated with it, such as *declaresMethod(...)* in

the example. .QL classes are matches of logical properties, that is, "when a value satisfies that property, it is a member of the corresponding type" [14]. So, each .QL class' constructor contains properties that must hold true for it to be an instance of the .QL class. For example, if one wanted to do a query of all private classes, the where clause in the example from Figure 2.6 could instead be c.hasModifer("private"). Rather than doing this query, however, one could create a new .QL class that extends the existing .QL class entitled *Class* as in Figure 2.7. This example defines a new class, *PrivateClass*, that matches all private classes, as noted by the constructor. If only static-private classes are desired, then an *and* clause could follow the private modifier test in the constructor followed by this.hasModifer("static"). As a result, .QL is extensible as new libraries of classes and queries can be built on top of existing .QL libraries of .QL classes and queries.

```
class PrivateClass extends Class {
        PrivateClass () {
                this.hasModifer("private")
                }
}
```

Figure 2.7: Example of a .QL Class definition

# Chapter 3

# Engineering the EJB FSML

This chapter outlines the steps we took and design decisions we made in engineering an EJB FSML that facilitates model analysis for the purpose of understanding the current configuration at development time. We introduce a new FSML for the EJB framework as none yet exists. We engineer the EJB FSML using the FSML Engineering Method defined in [3], which is a method that follows the Rational Unified Process [29]. The method is iterative and the EJB FSML went through multiple iterations. As such, rather than presenting the EJB FSML and its elements all at once, the following sections in this chapter discuss the iterations, the activities we performed, and the FSML components we created in each of the iterations of development in the creation of the EJB FSML. We discuss only steps that differ among subsequent iterations, that is, anything that does not change from one iteration to the next does not appear in the iterations that follow.

## 3.1 Iteration 1: Information from All Sources

### 3.1.1 Inception

The problem we identify for this iteration is the need to have a single source/model that accounts for all methods of configuration. Without such a model, applying the override rules defined in the EJB 3.0 specification would be difficult. Such a model will help assist the people in the various roles of an EJB Project. As such, the purpose for this iteration is to create an FSML that is able to capture all the information necessary from an EJB project's XML deployment descriptor and annotated Java classes. At this point, we consider only projects that follow the EJB 3.0 specification to simplify the functionality provided by this iteration.

[3] defines many use cases for an FSML but not all of them are applicable for every FSML and for every iteration. The first use case we consider applicable to achieving the goal of this iteration for EJB FSML is *Completion Code Understanding*. This use case describes when the framework-specific model is to

provide users with an understanding of the specific instance of the framework the completion code represents. So, in the case of the EJB FSML, EJB developers and deployers should be able to read through the EJB FSM and determine what configuration has taken place and from what sources. The second and main use case that is to be supported for this iteration is *Model Analysis*. This use case involves using the EJB FSM extracted through reverse engineering to perform model analysis on it to ensure that the EJB FSM accounts for all the necessary sources of configuration. Lastly, this iteration must support the use case of relatively simple *API Constraint Checking*, namely cardinality constraints and ensuring referential integrity of interfaces specified by beans.

For this iteration, the FSML's sources of domain knowledge need to cover all the various means of configuration for an EJB 3.0 project. Our initial reference is the Java EE 5 tutorial [30], specifically the section on Enterprise Java Beans. This provides a high-level walk-through example of configuring an Enterprise Java Bean via Java 5 annotations and covers the case where an EJB is configured through annotations only. The EJB 3.0 specification [16] and included EJB XML deployment descriptor schema [15] provide detailed information about configuring an EJB via XML and Java annotations and the accompanying override rules. The specification, although a bit terse, contains the most information about the resolution by EJB containers at deployment-time.

The horizontal scope of the FSML refer to which top-level API concepts / components are included in the language [3]. For this iteration we must include EJBs and EJB business interfaces identified either by a Java 5 Annotation or through a 3.0 XML deployment descriptor entry. That is, if either source of configuration identifies or configures either an EJB or an EJB interface, then it must be included in the FSML for this iteration. The vertical scope of the FSML means the depth/detail level of the feature models specifying the concepts [3]. For this iteration, the FSML must facilitate the identification and merging of an EJB that has been overridden by another source of configuration and must be deep enough to provide basic details in order to assist completion code understanding.

### 3.1.2 Elaboration

As mentioned previously, the top-level concepts in scope for this iteration are each EJB and EJB business interface defined via Java 5 annotations and the XML deployment descriptor. Per the EJB 3.0 tutorial [30], it is clear that the fully qualified class name for each annotated EJB is a concept that must be included in the language. Furthermore, the specific EJB Java annotation (*@Stateless*, *@MessageDriven*, et cetera) must be a concept in scope as it contains useful information regarding the EJB within its details.

The tutorial mentions that a developer can specify a EJB business interface in two ways via a Java annotation: Either the Java class file representing the interface can be annotated with *@Remote* or *@Local* or the bean class

implementing the interface can be annotated with *@Remote(InterfaceName.class)* or *@Local(InterfaceName.class)* where InterfaceName is the name of the Java file representing the interface. Four framework-provided concepts arise out of this: local and remote EJB interfaces that are explicitly EJB interfaces by having @Local or @Remote, respectively, annotated on their class; and local and remote EJB interfaces that are derived to be EJB interfaces from the fact that the class implementing them has used the @Local or @Remote annotation, respectively. For the XML deployment descriptor concepts, the top-level concepts, as mentioned previously, are the entries for the the various EJB types (Session, Message-Driven, and Entity). Also, the EJB interfaces the infrastructure identifies within the XML bean entries are framework-provided concepts that are within scope for this iteration.

The enterprise beans the FSML infrastructure detects via annotations and the deployment descriptors are component-oriented concepts as they do not relate to other concepts. The EJB interfaces the infrastructure recognizes from both of the configuration sources are port-oriented concepts because they provide access to EJBs to users of EJBs. The annotations for the most part are component-oriented as they do not relate multiple concepts, however, the *@Local* and *@Remote* annotations that are on EJBs are connector-oriented concepts because they represent a connection between an EJB and an EJB interface.

The root feature for the EJB FSML feature model is the EJB project containing the Java files and the XML deployment descriptors. While an EJB project root feature contains all the concepts the FSML infrastructure identifies, there is a logical grouping that can be represented in the feature model for grouping concepts that come from annotations and those that come from deployment descriptors. Figure 3.1 presents the overall structure of the feature model for the EJB FSML. The root element, indicated by the root image, is the *EJBProject* feature. We represent the grouping by the two sub features of the EJBProject feature: *informationFromAnnotations* and *deploymentDescriptors*. The feature *informationFromAnnotations* is optional because there are EJB projects that may not have any information derived from annotations. The same goes for the *deploymentDescriptors* feature, except it has multiple optional cardinality because an Eclipse EJB project may contain multiple deployment descriptors. It also has a mandatory project name, which we represent as the feature of type String entitled *projectName*.

Figure 3.1: Overall Structure for Feature Model

### 3.1.3 Construction

Figure 3.2 and Figure 3.3 display the resulting feature models we constructed during this iteration that will be discussed in this section. The text between the angular brackets for each feature represents the feature's annotation and is related to the mapping types and definitions discussed below. Appendix A contains the equivalent Ecore class models for these feature models, as these feature models are generated from Ecore models using Ecore.FMP [50].

Figure 3.2: Iteration 1-Annotation Information Feature Model

Using the feature model structure we outlined previously, the *informationFromAnnotations* feature is the parent feature for all the EJB project concepts/features that the infrastructure discovers from looking at annotations or Java source alone. That is, the sub features of the *informationFromAnnotations* feature are the EJBs

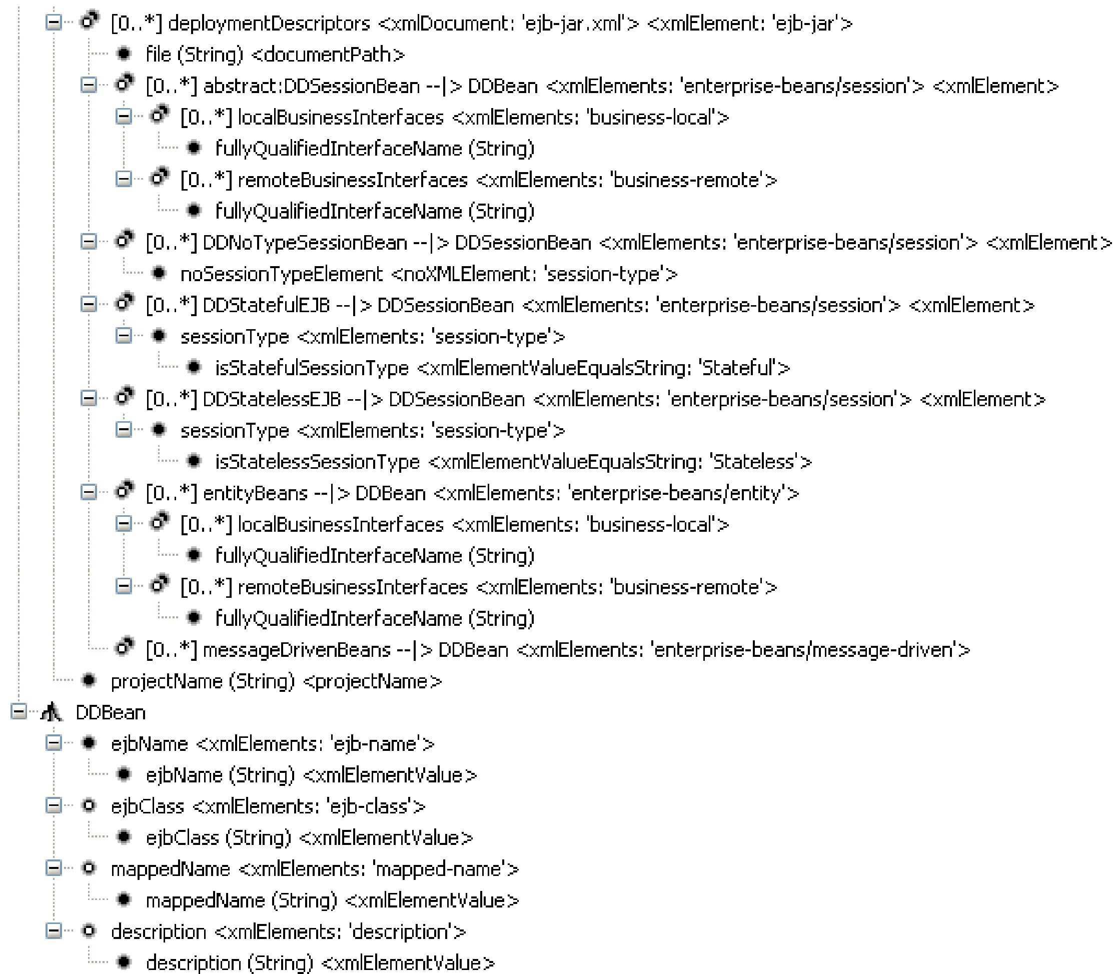


Figure 3.3: Iteration 1-Deployment Descriptor Information Feature Model

and the EJB interfaces, both derived and explicit, found from annotations as they semantically imply the information acquired from annotations. The same holds true for the EJBs the FSML infrastructure discovers via the XML deployment descriptor(s), so they are sub features of the *deploymentDescriptors* parent feature. The EJB interfaces the infrastructure discovers from the XML deployment descriptor(s) are sub features of the EJB features under *deploymentDescriptors* because they are defined within the EJB XML deployment descriptor entry and are, thus, logically dependent.

Within the *informationFromAnnotations* feature, the features representing Java annotations themselves are sub features of the respective EJB or EJB interface features they were discovered in as the annotations are logically dependent on the classes those features represent. For each of these annotation features, the Java annotation details belonging to the annotations are logically dependent on the annotations and are, thus, sub features. Also, sub features of the session beans found from annotations, that is sub features of the *SessionBean—|⟩EJBean* feature, include the local and remote annotations as well as a String list of and references

to the implemented explicit local and remote interfaces. Lastly, each feature representing an EJB that the FSML infrastructure discovers within the deployment descriptor has sub features representing the ejbName, ejbClass, mappedName, and description XML elements found within the XML entry and each of those has a sub feature representing the textual value found within that element.

Throughout the feature model, there are a number of instances where we can generalize features, thus allowing for features to be abstracted and be inherited from, analogously to class modeling. All abstract features within the feature models begin with the text *abstract:* followed by the feature's name. One notable case of an abstract feature within the information from annotations hierarchy is the abstract feature *abstract:SessionBean—|⟩EJBean*, which represents a session bean and contains all the interface related sub features that can be present on a Java session bean class. We specialize it with the features *StatelessEJB* and *StatefulEJB*, which differ only in the type of annotation the classes are annotated with. Regarding deployment descriptors, the abstract feature *DDBean*, contains sub features that are common to all features representing EJB entries the FSML infrastructure finds within a deployment descriptor. Note that the Ecore.FMP tool renders the *DDBean* feature as a Root feature (and not abstract) because of the fact that it is not directly contained within the feature model even though subclasses of it are. This is the way that the Ecore.FMP tool chooses to interpret the Ecore model and render the feature model [50].

We discussed previously the cardinality of *informationFromAnnotations* and *deploymentDescriptors* in the elaboration section. The EJBs and EJB interface features the FSML infrastructure discovers from Java annotations have a cardinality of 0..*, that is, these features are multiple features with an unbounded upper bound as there are no restrictions as to the number of these that must be present in any EJB project. For each EJB and explicit EJB interface feature the infrastructure discovers from annotations, the sub feature representing the Java annotation is an essential feature with cardinality 1..1 meaning that there must be one and only one instance of the relevant annotation for the parent feature to be considered an instance. For example, the *StatefulEJB* feature cannot exist without its sub feature *statefulAnnotation*. The features representing the annotation details from these annotations are optional features having a cardinality of 0..1 as they are not necessary and can have at most one instance. The features representing a local or remote annotation on an EJB session bean Java class that indicate a derived local or derived remote interface specification, respectively, have an optional cardinality (0..1) and have an essential (1..*) sub feature, *derivedLocalInterfaces* and *derivedRemoteInterfaces*, respectively, that represent the EJB interface Java classes specified in the annotation's detail. Furthermore, an EJB session bean class feature, *abstract:SessionBean—|⟩EJBean*, additionally has optional (0..1) sub features representing both a String list of names and references to all the explicit local and remote interfaces that are implemented by the class the features represents.

The cardinality of the EJBs the FSML infrastructure discovers from the XML

deployment descriptor(s) via XML entries is 0..* as there are no constraints on the number of the entries in the schema. Each of the three types of XML EJB entries, session; entity; and message-driven, generalize the abstract feature *DDBean* and have a mandatory (1..1) ejbName feature representing an *ejb-name* XML element and optional (0..1) ejbClass, mappedName, and description features representing the corresponding XML elements. Each of these have a sub feature representing the String value within the element that has mandatory (1..1) cardinality. These cardinalities correspond to those specified in the EJB 3.0 schema [15]. The local and remote EJB 3.0 business interfaces the infrastructure identifies for both session beans (feature *abstract:SessionBean*—|⟩*DDBean*) and entity beans (feature *abstract:entityBeans*—|⟩*DDBean*) within the deployment descriptor(s) are of type multiple optional, having cardinality of 0..*. The features belonging to *DDStatefulEJB* and *DDStatelessEJB* that represent session-type and the corresponding sub feature that indicates if it is the appropriate session type are mandatory (1..1) and essential. The justification for this is all deployment descriptor session beans identified as either Stateful or Stateless must have their session type specified as the expected type, otherwise it is not the instance of interest.

The features in the displayed EJB FSML feature models shown also contain annotations, listed in the figures between the angular brackets, which correspond to mapping types and definitions. Mapping types refer to the type/nature of the query to be performed on the code/artifact while mapping definitions are the exact values to be used in the query. For example, the feature *statefulAnnotation* with parent feature *StatefulEJB* has a mapping type of *annotatedWith* and a mapping definition of *javax.ejb.Stateless*, meaning it will look for Java classes that are annotated with that type of annotation. Mapping definitions for the information from Java annotation elements use the Java mapping interpreter while the mapping definitions for the information obtained from the deployment descriptor use the XML mapping interpreter. All mapping types and definitions that were in existence before the construction of the EJB FSML can be found in [1]. Furthermore, there are some mapping types that are unique to EJB projects, that is, some feature to code mappings that are relevant only in the EJB domain. Thus, we create a new mapping interpreter, *EJBMappingInterpreter*. Examples of mapping types are the *implementsExplicitLocalInterface* and *implementsExplicitRemoteInterface* mapping types, which gather all the string representations of the interfaces that are EJB local or remote interfaces, respectively, implemented by the class in context. Table 3.1 presents all the new mapping types that we introduced in iteration 1. Following the same structure of the analogous tables presented in [1], the first column presents a SmallTalk-like representation of the pattern the infrastructure is matching, the second column presents a description of the elements being matched, and the last column is the abbreviated version of the pattern that we use within the feature model and in this paper.

Table 3.1: Additional mapping types added for Iteration 1

| **Structural Pattern Expression** | **Structural Element(s) Matched** | **Abbreviation** |
|---|---|---|
| c ImplementsExplicitLocalInterface eli | matches the fully qualified interface name of Java interface eli that is annotated with @Local and implemented by class c | ImplementsExplicitLocalInterface |
| c ImplementsExplicitRemoteIntreface eri | matches the fully qualified interface name of Java interface eri that is annotated with @Remote and implemented by class c | ImplementsExplicitRemoteInterface |
| e noXMLElement s | matches if XML element e has no sub element/child with element name s | noXMLElement |
| e xmlElementValueEqualsString s | matches if the value of xml element e is equal to string s | xmlElementValueEqualsString |

### 3.1.4 Transition

Once we construct this iteration's feature model, the FSML infrastructure reverse engineers various EJB projects and generates the corresponding FSMs. Section 3.4 discusses these projects and FSMs in detail. At this point we perform various forms of evaluation that correspond/are equivalent to those identified in [3].

Semantically, we show the feature model to be satisfiable as valid feature configurations come from reverse engineering the projects. During this phase, we discovered that the XML code queries for some of the mapping types in the XML mapping interpreter were incorrect, so they we corrected them promptly. Also, at first, the infrastructure was not identifying some of the explicit Local and Remote interfaces as sub features of the appropriate session bean features, so we determined that a String list of these interfaces, represented by the features*implementedLocalInterface* and *implementedRemoteInterface*, are necessary in order to check this list against the already identified explicit interfaces the FSML infrastructure has found in the project.

We consider the pragmatic quality of the EJB FSML feature model thus far to be high given the fact that we utilized abstracted features whenever we felt that it was both logical and helpful. The FSML feature model can be easily modified as the only dependences existing in the model are the containment features that are obtained by the modeling of the framework/domain. So the majority of changes would be trivial except in cases of modifying containment relationships, in which case the model may cease to be semantically correct.

The pragmatic quality of the generated EJB FSM in regards to comprehension was originally not very high due to the use of generic/non-helpful icons and the showing of containment references only (with non-containment references being shown in the properties menu instead of the model). We added custom icons for various EJB components and both containment and non-containment references are shown when thought to be of assistance to viewers. Section 3.4 has examples of FSM models and these pragmatic improvements. Pragmatic quality of the generated EJB FSM in terms of ease of modification of the model is not applicable as modification is disabled due to the fact that the only use case supported is reverse engineering, so modification is unnecessary. Furthermore, the FSM is able to perform the three

identified use cases, *Completion Code Understanding*, *Model Analysis*, and basic *API Constraint Checking*.

The organizational evaluation of the EJB FSML for this phase entails performing the model analysis on the EJB FSM, discussed in Section 4.2, and ascertaining the ability to resolve the current configuration of the EJB 3.0 project that the EJB FSM describes. Also, we verify traceability links as they add to the organizational value for the EJB FSM.

## 3.2 Iteration 2: EJB 2.1 Projects

### 3.2.1 Inception

The goal for this iteration is the same as that in iteration 1, that is, to have an FSML that expresses an FSM that accounts for all the methods of configuration when reverse engineering is performed on an EJB project. However, in this iteration, the FSML and associated mapping definitions must support EJB 2.1 projects. The use cases supported are the same as those supported in iteration 1. The sources of domain knowledge for this iteration are all those included in iteration 1 as well as the materials that are related to 2.1 EJBs. This includes the Java 1.4 tutorial's section on 2.1 Java Beans [4], which provides a high-level description of how to create and configure 2.1 EJBs; the EJB 2.1 specification [17], which provides detailed and specific information; and the EJB 2.1 deployment descriptor schema [44].

As noted in the EJB 2.1 specification, "the Bean Provider is responsible for providing the structural information for each enterprise bean in the deployment descriptor." As such, the horizontal scope (top-level concepts) for this iteration is the same as the previous iteration because beans found in the EJB 2.1 deployment descriptor can be considered the same as those found in the EJB 3.0 deployment descriptor. However, the fact that the beans are the same implies that the vertical scope (depth of the concepts) of these concepts from the last iteration needs to be extended to include XML elements necessary for 2.1 EJBs, such as the different interfaces.

### 3.2.2 Elaboration

As a developer must configure properties for EJB 2.1 beans via the deployment descriptor, the new concepts for this iteration are all deployment descriptor related. While all EJB 2.1 beans must implement the appropriate EJB 2.1 interface, such as javax.ejb.SessionBean for session beans, the implementation of this interface alone does not constitute a valid bean [17]. Since this iteration deals with identifying beans and their main properties only, the only additional concepts in this iteration are the 2.1 EJB local and remote interfaces and the local and remote home interfaces for session beans and entity beans. Message-driven beans do not implement these

interfaces [17, 44]. In an EJB project, EJB Home, local or remote, interfaces are used by a client (bean user) to create a session or entity object or a remote session or entity object and to acquire meta data about the session or entity [17]. The only difference between the 2.1 and 3.0 interfaces is that the EJB 2.1 interfaces are not implemented directly by the Java classes as they are in EJB 3.0 projects. As such, the new concepts within scope are the local and remote EJB 2.1 interfaces and local and remote home interfaces for session beans and entities. All of these interfaces are port-oriented concepts as they facilitate access to the beans.

One option for the overall structure of the feature model is to have a different feature model or a different branch in the previous model for an EJB 2.1 project. However, some projects may use a combination of 2.1 and 3.0 EJB beans as supported by the framework [39]. As such, another option is to have the overall structure of the feature model for this iteration remain the same because information is still being retrieved from either annotations or deployment descriptors. In the case of a pure (non 3.0) EJB 2.1 project, the *informationFromAnnotations* feature in an FSM retrieved from reverse engineering will have no feature instances underneath it, but we found this to be an acceptable trade off for having a single feature model for the EJB FSML.

### 3.2.3 Construction

Figure 3.4 exhibits the amendments to the feature model underneath the deploymentDescriptors feature. Appendix A contains the equivalent Ecore class model for these feature model, as this feature model was generated from Ecore models using Ecore.FMP [50].

We make the features representing local home and remote home interfaces and the corresponding 2.1 local and remote interfaces as sub features of the features representing the respective session bean and entity deployment descriptor XML entries they are defined within. This is because these interface entries are logically dependent on the bean entries. Each interface then has a sub feature that represents the fully qualified class/interface name for the Java file embodying the interface. We declare the cardinality for the interfaces are as optional, per the schema. The string sub features of the interfaces representing the fully qualified interface name are mandatory features.

We already have defined the mapping types xmlElements and xmlElementValue within the XML mapping interpreter, so no new mapping types are required. We obtain the mapping definitions for the xmlElements mapping type from the XML schema. Specifically, 2.1 local and remote interfaces have mapping definitions of *local* and *remote*, respectively, and the local home and remote home interfaces have mapping definitions of *local-home* and *home*, respectively. The string feature representing the interface qualified name uses xmlElementValue with no definition, indicating that it is the string value located within the XML element in context.

Figure 3.4: Iteration 2-Additional Features

### 3.2.4 Transition

At this point, we use projects that conform the EJB 2.1 architecture to the verify manually the FSMs. Section 3.4 outlines these projects and provides the FSMs generated.

For semantic evaluation, we show the feature model for this iteration to be satisfiable through the various feature configurations that we discover from reverse engineering the projects. Since no new mapping types are used, we need verify the mapping definitions only.

Pragmatically, the feature model gained relatively few elements, so pragmatic quality remains relatively constant compared to the last iteration. In terms of comprehensibility and modification, an option is to abstract away the commonalities between *DDSessionBean* and *entityBeans* into a feature that represents beans that utilize interfaces. While this would aid modification, since the interfaces and their mapping definitions would need to be modified only once, we believe that the comprehension of the feature model would suffer as a result. The pragmatic quality

of the EJB FSM is unaffected by the addition of the new EJB 2.1 features. We provide additional custom icons to the appropriate feature instances to assist with understanding. The use cases we identified for this iteration are still supported.

We examine the organizational quality of the EJB FSML feature model by analyzing, via the method discussed in Section 4.2, the EJB FSMs to create the resolved model. If projects that use a combination of the EJB 3.0 and EJB 2.1 architectures or projects that use the EJB 2.1 architecture in isolation can have their current configuration established and displayed through this analysis, then the organizational quality is satisfactory.

## 3.3 Iteration 3: Facilitate Antipattern Instance Detection

### 3.3.1 Inception

The goal for this iteration is to have an EJB FSML that is able to express an EJB FSM that can indicate whether or not antipattern instances are present within an EJB project. We discuss the specific antipatterns considered in Chapter 5. Thus, the FSML needs to contain features specified in the feature model that can provide the information necessary to detect antipattern instances. This iteration does not support additional use cases because the only use case necessary to achieve the iteration's purpose of detecting antipattern instances is advanced *Model Analysis*.

Information regarding the additional features and concepts within this iteration's scope is mainly the material from Dudney's antipattern book [19], the predominant source of existing EJB antipatterns we found. The horizontal scope of the language remains relatively unchanged as the antipattern properties/features are related to existing top-level elements, except for the addition of clients that use EJBs. We extend the vertical scope, however, to include these properties of concepts within an EJB project that represent symptom properties and other indicators of antipattern instances.

### 3.3.2 Elaboration

Many of the antipatterns within scope have symptoms that are related to properties of clients, that is, Java classes that reference and use beans defined within the project. Beans can also be clients themselves (EJB clients) as they may reference and utilize other beans. Such clients need to be accounted for in the current iteration. Any client using the EJB 3.0 architecture will be using the *@EJB* annotation on fields to specify the reference to other beans [30]. Thus, we must include this annotation, its details, and the field it is placed on in the language. For clients using the EJB 2.1 architecture, a developer must specify these references in

the appropriate deployment descriptor entries: all EJB clients must specify all remote and local references within the XML deployment descriptor by placing an *ejb-ref* or an *ejb-local-ref* entry, respectively, within their XML bean entry. Figure 3.5, modified from [16], provides an example of an EmployeeService session bean that has two references to other beans, a 2.1 EJB EmplRecord and a 3.0 EJB Payroll, within the project. These XML reference entries and all sub entries within the deployment descriptor are concepts we include in the language.

```
<session>
...
        <ejb-name>EmployeeService</ejb-name>
        <ejb-class>com.wombat.empl.EmployeeServiceBean</ejb-class>
        ...
        <ejb-ref>
                <description>
                This is a reference to an EJB 2.1 entity bean that
                encapsulates access to employee records.
                </description>
                <ejb-ref-name>ejb/EmplRecord</ejb-ref-name>
                <ejb-ref-type>Entity</ejb-ref-type>
                <home>com.wombat.empl.EmployeeRecordHome</home>
                <remote>com.wombat.empl.EmployeeRecord</remote>
        </ejb-ref>
        <ejb-local-ref>
                <description>
                This is a reference to the local business interface
                of an EJB 3.0 session bean that provides a payroll
                service.
                </description>
                <ejb-ref-name>ejb/Payroll</ejb-ref-name>
                <local>com.aardvark.payroll.Payroll</local>
        </ejb-local-ref>
</session>
```

Figure 3.5: Sample Deployment Descriptor EJB References

A number of the antipatterns are concerned specifically with usage of entities by EJBs. When using an entity within an EJB, no annotations are necessary. In EJB 3.0 an entity manager is used to populate a local variable/field that is typed with the corresponding Java entity class [30]. In EJB 2.1, one uses the entity's home interface as an entity manager in that it also populates a local variable/field [17]. We include these entity references as concepts in this iteration.

We include any other properties of existing EJB concepts that occur in one or more antipattern definitions, such as the number of public methods implemented by a bean, whether or not a bean implements a specific interface, and others.

The clients identified are component-oriented concepts. The concepts involved in referencing other EJB beans and entities, such as the @EJB annotation, entity fields and local variables, and the XML reference entries, are connector-oriented concepts as they relate concrete concepts like EJBs and entities to other concepts like clients, EJB clients, or entity users.

The overall structure of the feature model for this iteration remains the same as the previous iteration as the concepts we identify are still either derived from

information from annotations or from XML deployment descriptors. As such, the top-level structure/organization of the feature model remains the same.

### 3.3.3 Construction

The first concepts we add and decompose into features are the features necessary to represent clients and EJB clients that are derived from annotation information. Figure 3.6 highlights these additional features, along with their mapping definitions, within the *informationFromAnnotations* feature. The *clients* feature represents clients and is a child of *informationFromAnnotations* as it is logically dependent. To ensure that it is not a bean, it has the mandatory and essential feature *isNotAnEJB*. Both clients and all EJBs (client or not) contain a child feature *fieldsAnnotatedWithEJB* however the cardinalities are different. Clients have this feature with an essential mandatory unbounded cardinality as it is necessary to have at least one field annotated with EJB to be considered a client. EJBs (client or not) represented by the *EJBean* feature from the previous iteration contain the *fieldsAnnotatedWithEJB* feature with an optional unbounded cardinality as it is not necessary for all EJBs to be EJB clients. The *fieldsAnnotatedWithEJB* feature itself contains both its field name and the EJB annotation feature *EJBInterfaceAnnotation* as a child. The *EJBInterfaceAnnotation* contains children features, which have an optional cardinality and represent the corresponding annotations' details.

Figure 3.6 exhibits more additional features to facilitate EJB 3.0 entity uses within an EJB. We add two children features to the *EJBean* feature: *entityFields* and *entityLocalVariables*. *entityFields* represents fields within an EJB Java class that are typed with a Java class that is an entity while *entityLocalVariables* represents local variables within an EJB java class that are typed with a Java class that is an entity. A Java class is known to be an EJB 3.0 entity if it is denoted with the *@Entity* annotation. If the entity is an EJB 2.1 entity, then there must be a corresponding XML entry in the deployment descriptor. So EJB 2.1 entity uses can not be detected until the model analysis stage. The *entityFields* feature contains a mandatory essential feature, *entityReference*, that indicates the corresponding field is typed with an EJB 3.0 entity. The *entityLocalVariables* feature also contains the same *entityReference* feature, but this feature indicates the local variable is typed with an EJB 3.0 entity.

Figure 3.7 displays new features added to detect antipattern instances. The child feature *publicMethods* underneath both the *SessionBean* and *Entity* features represents the names of public methods the FSML infrastructure finds within the respective session bean and entity Java classes. The cardinality of this child feature is mandatory unbounded as every one of these classes must have at least one public method beyond the excluded public methods from [16]. *SessionBean* also contains the feature *implementsSessionSynchronization* that indicates whether the session bean within context implements the SessionSynchronization interface included in the Java EE framework.

Figure 3.6: Iteration 3-Features for Clients and Entity Users

Many of the mapping types used in this iteration are already implemented within the Java Mapping Interpreter. However, we still added some mapping type additions to the interpreter, as shown in Table 3.2. Firstly, there is no support yet for local variable declarations as it was not necessary for the FSMLs

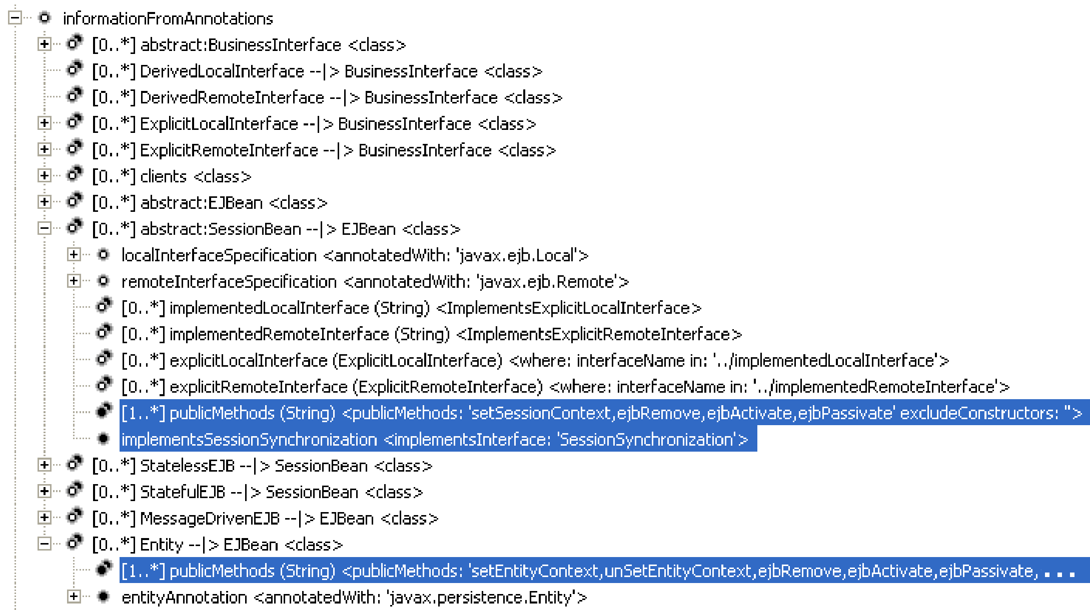


Figure 3.7: Iteration 3-Additional Antipattern-Specific Features

developed to date. We therefore add the *localVariable* context, *localVariableName*, and *localVariableType* mapping types to the interpreter along with the appropriate mappings. The mapping type *publicMethods* provides a string list of all the public methods in the class. Furthermore, the mapping definition for this mapping type includes the parameter *excludes*, which indicates methods to ignore and the parameter *excludeConstructors* with no value, which indicates that constructors should be excluded in the string list. We add the mapping types *fieldTypeAnnotatedWith* and *localVariableTypeAnnotatedWith* to match entity uses by EJBs and to find all fields and local variables, respectively, that reference types that are entities.

There were also some mapping type additions for the EJB mapping interpreter. We include these in Table 3.2 along with their descriptions. *isNotAnEJB*, for example, is one such mapping type that ascertains whether a Java Class is not an EJB by ensuring that it does not implement any interfaces or contain any annotations that would indicate that it is an EJB. *referenceNameEJBAttribute* is another example. It determines the unique reference name/namespace based on the annotation detail values within the @EJB annotation. Depending on the type, marker or normal, of the annotation and the value of the name detail, it obtains the reference name according to the EJB 3.0 specification.

In order to test the feature model for this iteration, we modify the custom/test EJB project by adding a number of various EJBs and clients to it that contain both antipattern instances and any borderline cases, if applicable.

Table 3.2: Additional mapping types added for Iteration 3

| **Structural Pattern Expression** | **Structural Element(s) Matched** | **Abbreviation** |
|---|---|---|
| c IsNotAnEJB | matches if the class c is an not an EJB according to annotations or implemented interfaces | IsNotAnEJB |
| f referenceNameEJBAttribute | matches the proper EJB reference name used to reference a field, f, that is annotated with EJB based the annotation's details. This is done according to the rules in [16] for the resolution of EJB reference name | referenceNameEJBAttribute |
| field beanInterfaceEJBAttribute | matches the appropriate bean interface name used for a field, f, that is annotated with EJB based on its details. Calculated as per [16] | beanInterfaceEJBAttribute |
| f fieldTypeAnnotatedWith t | matches if the qualified type of the field f is annotated with qualified annotation type t | fieldTypeAnnotatedWith |
| localVariable | matches a local variable | localVariable |
| lv localVariableName | matches the name of local variable lv | localVariableName |
| lv localVariableType | matches the type of local variable lv | localVariableType |
| lv localVariableTypeAnnotatedWith t | matches if the qualified type of the local variable lv is annotated with the qualified annotation type t | localVariableTypeAnnotatedWith |
| c publicMethods excludingList excludeConstructors | matches methods that have public visibility; do not have a method name contained in the excludingList, which is a comma delimited string list; and is not a constructor if excludeConstructors is in the definition | publicMethods |

### 3.3.4 Transition

For semantic evaluation, we determine the feature model to be satisfiable as different feature configurations are generated by running reverse engineering on the various projects. We test the new mapping types and definition parameters accordingly. So, for example, we test the mapping type *isNotAnEJB* by reverse engineering projects that have non-EJB clients. We verify manually the mapping type *referenceNameEJBAttribute* after reverse engineering the various reference name resolution outcomes listed in the specification.

In regards to pragmatics of the feature model, there are few issues in this iteration. Some of the EJB specific mapping types, such as *beanInterfaceEJBAttribute* and *referenceNameEJBAttribute* have relatively cryptic names, which may hinder feature model understanding. The breadth of the model increases only by the addition of clients and the depth increases only minimally by the addition of the *EJBInterfaceAnnotation*. The pragmatics of the FSM obtained from reverse engineering stay relatively constant with respect to the last iteration as the new feature instances added are mainly sub features within existing top-level features.

We evaluate the organizational quality of the feature model through reverse engineering and manually verifying that the FSML infrastructure discovers all the appropriate instances and that there are no false positives or false negatives. As discussed in Section 2.3.1, we display the antipatterns in another Eclipse view and in a tree-like structure.

## 3.4 Evaluation

### 3.4.1 Sample EJB Projects Tested

One notable limitation of the EJB FSML presented in this chapter is the lack of depth in the examples that are used for testing and verification. Seeing as Enterprise Java Beans are, by nature, intended for enterprise ventures, it is difficult to obtain concrete examples that organizations are willing to share. However, a number of instances of open-source EJB projects do exist and we use them for testing the EJB FSML. Furthermore, to address the lack of depth of the open-source examples, we construct and use a custom EJB project that ensures that the EJB FSML accomplishes its goals for the specific iteration in question and also tests all special or unique cases that may not exist in many projects. Overall, we use 11 EJB open-source projects in conjunction with the custom EJB project. Table 3.3 lists these open-source projects; a brief description of the project and its depth; and a website location for each project and where its documentation can be found. Rather than discuss each of the projects listed in the table, we discuss only the ones with both significant depth and purpose in detail in this paper.

Table 3.3: EJB Projects Used For Testing

| **Project** | **Description & URL** |
|---|---|
| eMal Payment System | Internet-Based Payment System. EJB 3.0 project with 19 entities, 4 interfaces, and 2 session beans. FSM presented in Figure 3.8. `http://sourceforge.net/projects/emal/` |
| WebDiary | WebDiary is multi-user web application where users define tasks. EJB 3.0 with 1 interface, 1 session bean, and 3 entities. `http://sourceforge.net/projects/web-diary/` |
| BellWhistle | An mp3 jukebox server. Few EJB 2.1 session beans, entities, and interfaces. `http://sourceforge.net/projects/bellwhistle/` |
| Storm | Web based management tool for XP user stories. Entity Beans only `http://sourceforge.net/projects/xpstorm/` |
| Redwood Web Log Mining System | A tool for mining web logs for information and statistics. Huge EJB 2.1 project with many sub projects and multiple deployment descriptors. 51 resolved entities, 698 EJB and EJB Home Interfaces, 6 Message Driven beans, and 47 Session Beans. FSM presented in Figure 3.10. `http://sourceforge.net/projects/redwood/` |
| Rowing Club Management Software | Manages the activities of a rowing club. Contains only entities for persistence. `http://sourceforge.net/projects/clra/` |
| Time to Work | Web based employee time tracking tool. No source currently, but contains deployment descriptor with 7 entities, 20 interfaces, and 3 session beans. FSM presented in Figure 3.9. `http://sourceforge.net/projects/timetowork` |
| WordNet Web Application | A four-tier web application that queries the WordNet lexical database. Only a single session bean. `http://sourceforge.net/projects/wnwa/` |
| Saeed | Web-based document management system. Uses entities only. `http://sourceforge.net/projects/saeed` |
| JUnitEJB | JUnitEJB can run Junit on a remote EJB server. Uses only a single EJB 2.1 session bean. `http://sourceforge.net/projects/junitejb` |
| Java Issue Tracker | Provides issue-tracking abilities to an EJB project. Contains EJBs but uses hibernate instead entities for persistence. `http://sourceforge.net/projects/jissuetracker/` |

The eMal payment system is an open-source payment system that is intended to facilitate use by merchants and consumers that owe them money. It provides an example of an EJB 3.0 system with a notable amount of EJBs, including 19 entities that represent users, admins, merchants, and others. The FSM for this project is in Figure 3.8. In the figure, the session bean element ConsumerSessionBeanBean shows 2 entity uses via local variables of the *model.User* entity and implementation of a local and remote explicit EJB interface, ConsumerSessionBeanLocal and ConsumerSessionBean, respectively. There is also an example of an entity using one or more other entities as the *model.Agent* entity uses two entities, *model.Account* and *model.Country*, as fields.

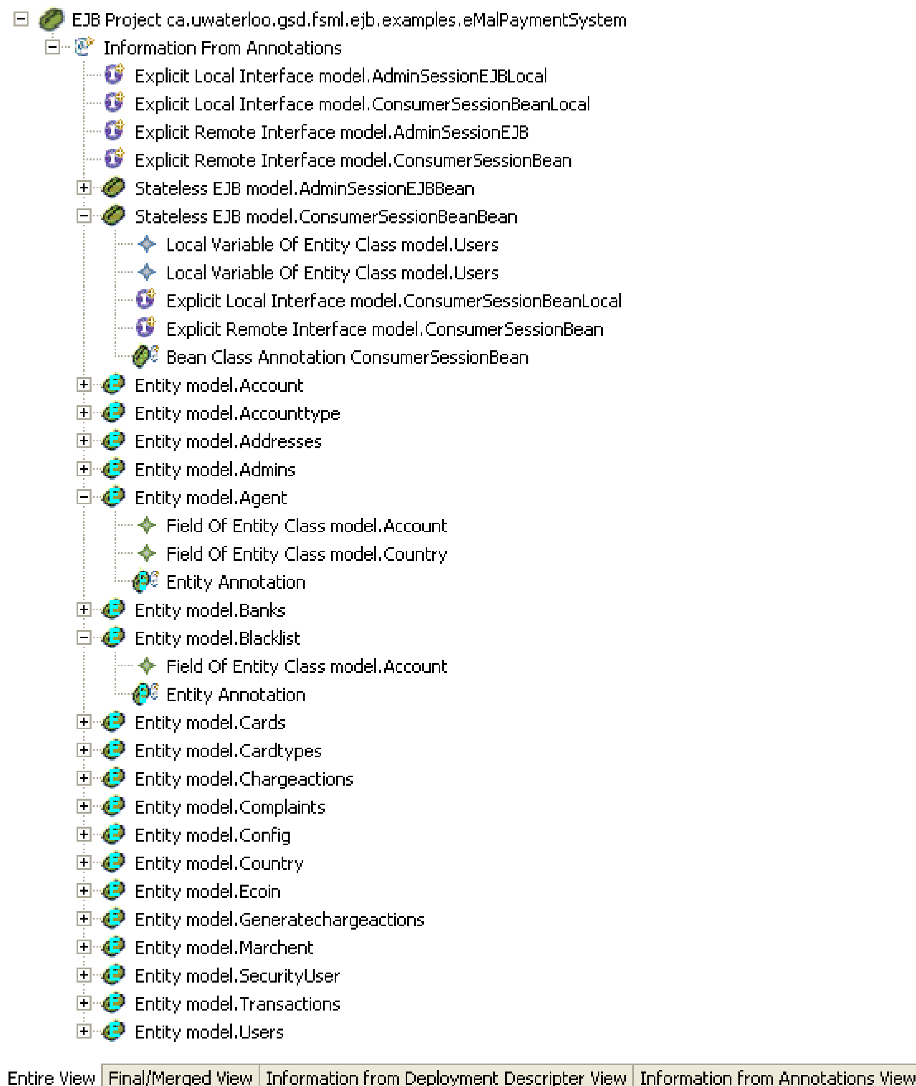


Figure 3.8: FSM yielded from eMal Payment System Project

The Time to Work project, presented as a reverse engineered FSM in Figure 3.9, is a system that allows employees to track their time spent working on various tasks. It is an EJB 2.1 system with 20 interfaces and 10 EJBs. This system can

be used for testing EJB 2.1 projects and it also has no source/java code and has compiled .class files, which makes it useful for testing the configuration of sources / resolution of the current configuration as discussed in Section 4.3. As shown in Figure 3.9, the expanded stateless EJB from deployment descriptor information contains local references and provides an example of EJB 2.1 local and remote interfaces and corresponding local home and remote home interfaces that are defined in the deployment descriptor.

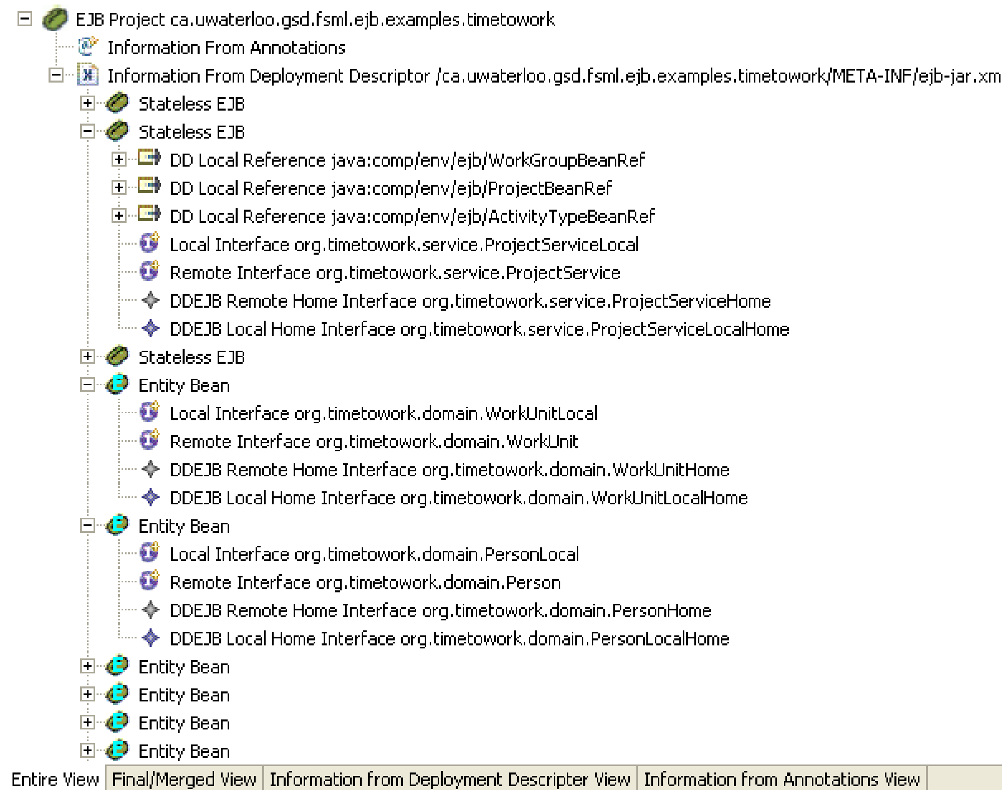


Figure 3.9: FSM yielded from Time to Work Project

The Redwood Web Log Mining System extracts information and statistics from website logs. It is an EJB 2.1 open-source project with all the corresponding Java/source code. This project is important in testing the FSML infrastructure due to is its large size. It is a Java project that contains many EJB 2.1 sub projects and has, in total, 51 resolved entities, 698 EJB and EJB Home Interfaces, 6 message driven beans, and 47 session beans. Furthermore, it has multiple deployment descriptors, something the FSML did not support until we discovered this project. Because the reverse engineered FSM in its entirety is too large to display in this paper, we present an excerpt of the FSM in Figure 3.10. The FSM accounts for multiple deployment descriptors in this project as multiple FSM elements with the XML icon and subsequent file path information. It is also a good stress test and can be used for testing and optimizing performance of the FSML infrastructure in the future as it takes significant time to perform the reverse engineering of the entire project.

Figure 3.10: FSM yielded from Redwood Web Log Mining Project

To test all the configuration override rules and EJB 3.0 cases not covered in any of the open-source projects, a custom project is created. This project ensures we verify all the mapping types/definitions at least once and verifies that the resolved configuration is correct as discussed in Section 4.3. We present the reverse engineered FSM in Figure 3.11. As expected, it contains both information from annotations and information from deployment descriptors. A non-EJB client, *test.client.Client1*, shows the use of an EJB annotation and that the appropriate reference name is shown following the dependency element. Further down the FSM, the *test.combination.StatefulClass* session bean element and its session bean Java 5 annotation is highlighted. The importance of highlighting is that whenever a user selects an element within the FSM its properties come up in the property view as seen near the bottom of the figure. We display the description, mapped name, and name of the session bean as indicated by the annotation's name/detail pairs. Within the deployment descriptor information, the FSM has a no-type session bean, that is, a session bean that does not have its type indicated in the deployment descriptor and therefore must correspond to an annotated EJB Java class to map

to a valid session bean. There is also an expanded Stateless EJB within the deployment descriptor information that contains both remote and local references and that implements the remote interface *test.FullyQualifiedName3*, according to the deployment descriptor.

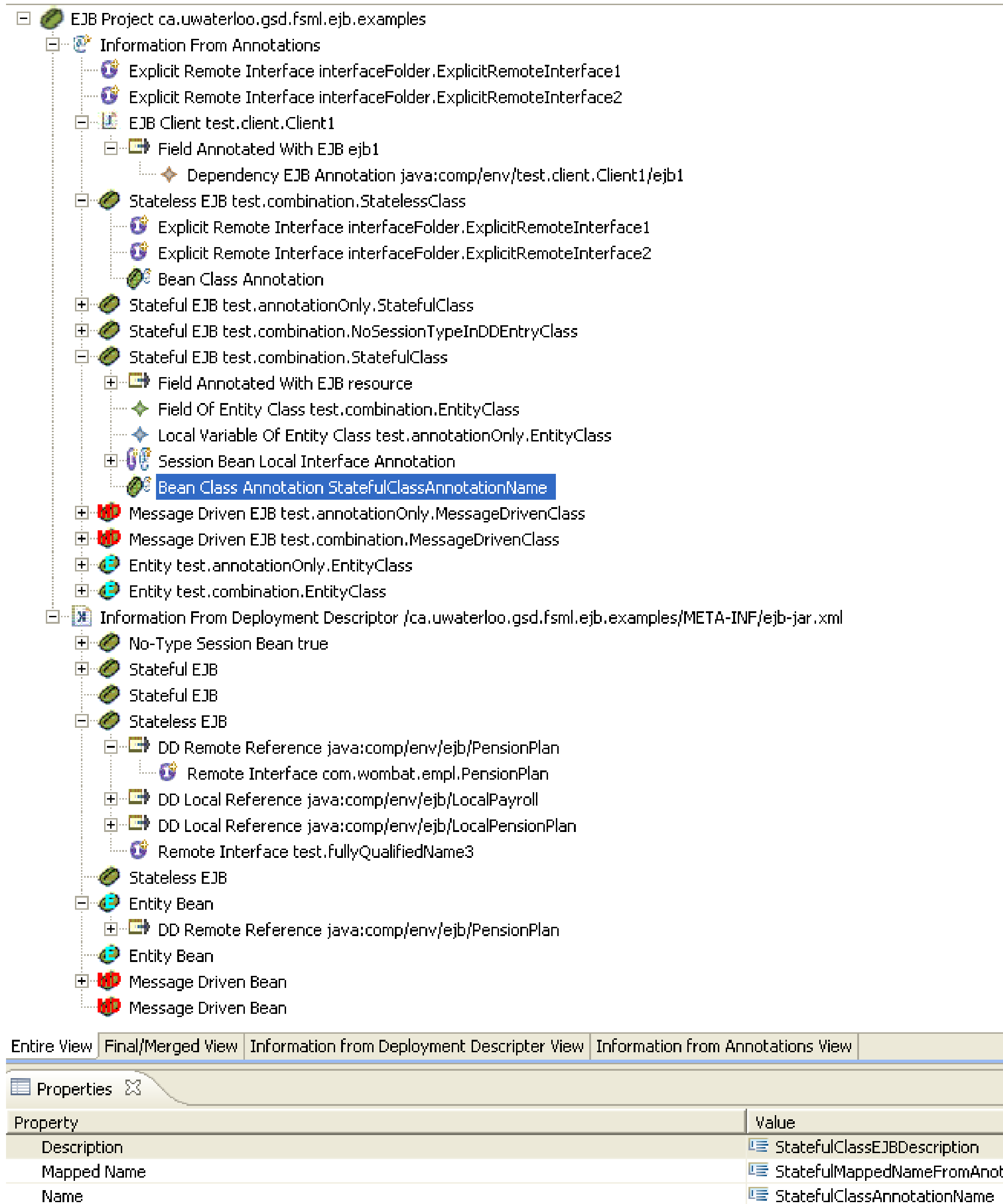


Figure 3.11: FSM yielded from Custom EJB Project

We use each of these projects as well as those listed in Table 3.3 to ensure that, for each iteration, the horizontal and vertical scope of the FSML feature model / FSML meta model are encompassing enough to accomplish the goal of the iteration. This evaluation does not show that our EJB FSML is the minimal language required

to accomplish each iteration's goal. Rather, this evaluation shows only that our EJB FSML accomplishes its goals. While a minimal language is desirable to maintain simplicity, we do not consider it a necessary step at this time.

### 3.4.2 Threats to Correctness of FSML

The main threat to correctness of the FSML for each iteration is the fact that its is validation is manual, so, of course, human error is a possibility. Specifically, the FSML and infrastructure generate each FSM and we verify the corresponding elements using either the traceability functionality of the FSML infrastructure or by manually navigating to the element. This threat to correctness is more likely in the case of the newly defined mapping types and definitions, that is, the ones defined in Tables 3.1 and 3.2, because the pre-existing mapping types and definitions have been used and verified by others through use [1, 2, 3]. The newly defined mapping types and definitions have been used for only this FSML to date.

### 3.4.3 Further Limitations of the EJB FSML

One problem with the current implementation of the EJB FSML is that it can not detect entity usage of EJB 2.1 or earlier entities via reverse engineering of the project into an FSM. EJB 3.0 entities are known to be valid entities once a developer annotates them with the @Entity annotation. As such, any usage of EJB 3.0 entities the FSML infrastructure detects from annotations can be ascertained because each field and local variable within scope can have their Class/type checked for the @Entity annotation. EJB 2.1 and earlier entities, however, require the developer both implement the appropriate interface witin the Java class and have the proper entry within the deployment descriptor to be vaid. While the class/type of the fields and local variables could be checked if the class implements the Entity 2.1 interface, this implementation of the Entity interface is more difficult to check as it requires FSML model traversal [1]. It is possible to implement this, but we choose not to implement it as EJB 2.1 entity usage is covered during the model analysis phase, discussed in Chapter 4.

## 3.5 Discussion

We consider the order of the iterations. Iteration 2 and 1 can switch in that the first iteration can support 2.1 EJBs and the iteration that follows can support 3.0 EJBs. This is because there are no conflicting or dependent changes/additions made between the two iterations. Iteration 2 makes additions only to the deployment descriptor and changes none of the existing elements. The iteration dealing with antipatterns must always follow the iteration dealing with annotation/Java information because many of the properties are source specific. The only other

way that the antipatterns could come before the iteration supporting 3.0 EJBs is if the FSML for supporting 2.1 EJBs assumed that it was sufficient to have a bean class implement the appropriate EJB session bean, entity, or message-driven interface without the corresponding entry in the deployment descriptor. In regards to breakdown of the iterations, another alternative is to focus on a specific type of bean in each iteration. That is, for one iteration, focus on supporting all versions of session beans and for the next, focus on supporting the different versions of entities. There would also need to be an iteration that focuses on aspects of EJB domain that are common to all beans, such as EJB references, entity usage, and others.

Another important consideration is the fact that the EJB FSML we present in this thesis is for the purpose of future analysis. Specifically, iterations 1 and 2 are focused on creating an FSM that we analyze later to present the current configuration of the project. Iteration 3 focuses on the analysis in that it deals with adding FSML elements that will help with the EJB antipattern detection that follows. This differs from to the way FSMLs have been created to date [3] in that FSML creation is typically API-driven. That is, FSMLs to date have been created by choosing scope related API elements of interest within a framework and then modeling those elements as FSML elements. The way we modeled the final FSML may have been impacted by the fact that we developed the EJB FSML with future analysis in mind.

# Chapter 4

# Providing a Resolved Model of an EJB3 Project

After reverse engineering an EJB project using the EJB FSML we discussed in the previous chapter, the resulting FSM contains both elements the FSML infrastructure discovers through means of code patterns in Java code and elements it discovers via analyzing patterns in XML deployment descriptors. This chapter discusses the way in which this FSM is filtered, in Section 4.1, and analyzed, in Section 4.2, to provide various views and new models useful to those involved in an EJB project. The resolved model refers to the configuration model that comes from resolving the current configuration through analysis of the EJB FSM.

## 4.1 Model Filtration of EJB FSM

Providing various views that allow EJB users to see the configuration sources of the beans in isolation may help in a number of scenarios. Before model analysis, we perform simple model filtration to show only the information from annotations and only the information from deployment descriptors in the Information from Annotation View, shown in figure 4.1, and the Information from Deployment Descriptor View, shown in figure 4.2, respectively. The models in the two figures come from filtering the EJB FSM from the custom EJB project. Selecting the option Navigate to Code on either of the views will open up the Model-Code Navigation window, which provides traceability from the currently selected element in the view to the respective code/artifact pattern allowing navigation to the pattern instance. The use of element keys that the FSML expert specifies via key annotations, such as *key*, *parentKey*, *indexKey*, on the FSML elements [1] facilitate traceability. The FSML infrastructure uses these keys to uniquely identify the instances within the FSM.

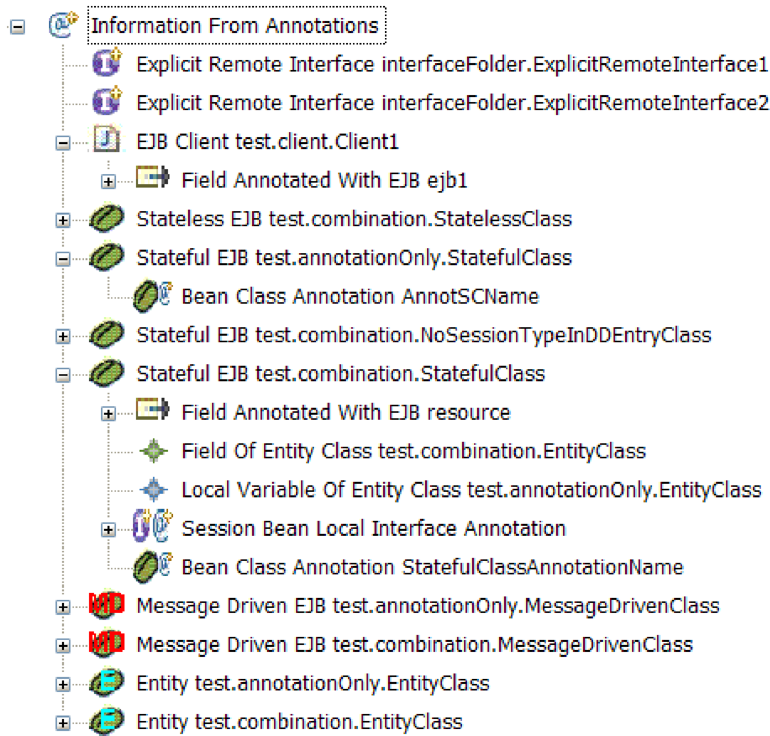


Figure 4.1: Information from Annotation View

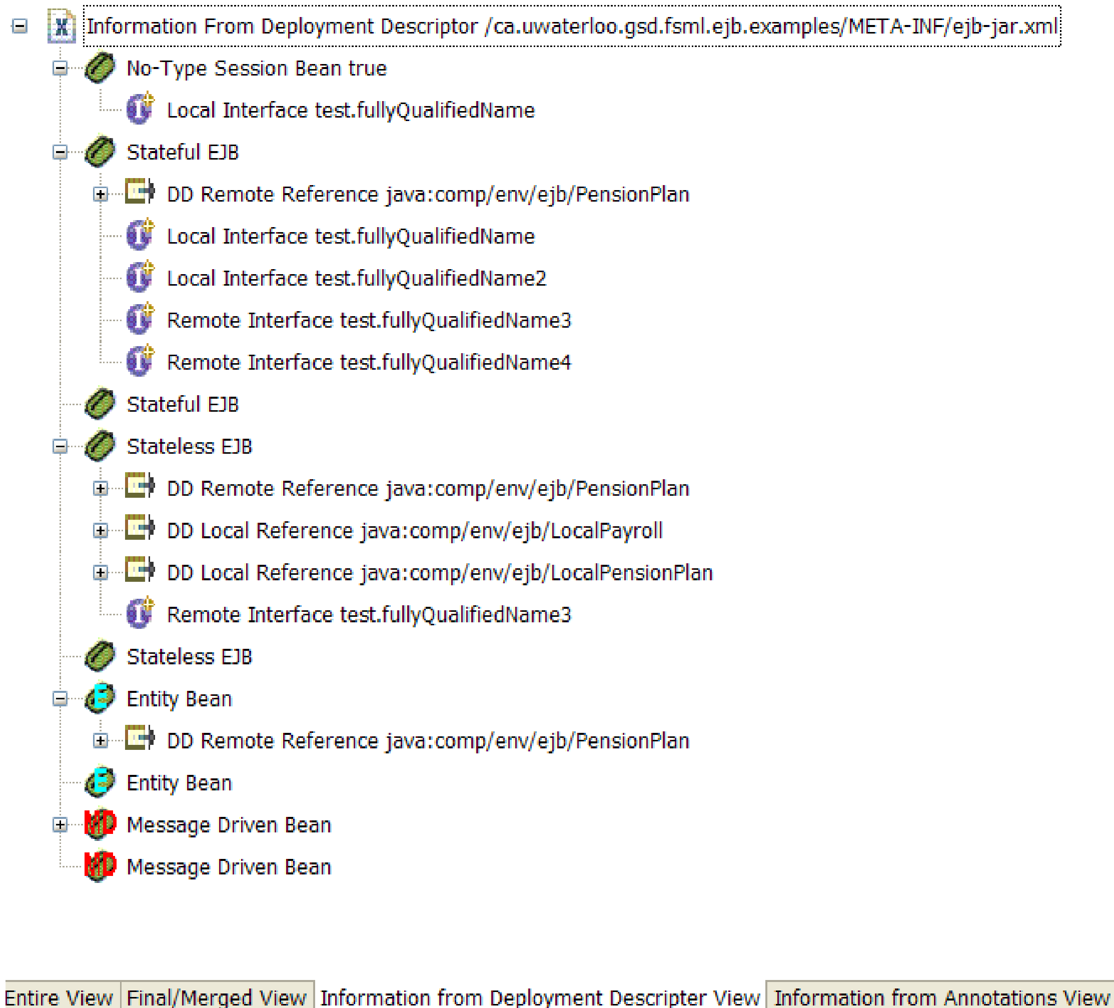


Figure 4.2: Information from Deployment Descriptor View

## 4.2 Model Analysis of EJB FSM

Beans that developers configure in the deployment descriptor but that are not configured/specified in annotations are still valid EJB 2.1 beans, assuming the Java class implements the appropriate Java interface [17]. The same is true for beans that the FSML infrastructure discovers from Java annotation code patterns that have no corresponding deployment descriptor reference, except that these are EJB 3.0 beans. Furthermore, some, but not necessarily all, of the beans the infrastructure finds from the XML deployment descriptors may correspond to beans found from Java annotations. To have an artifact that is useful to developers we must merge all this information into a single model that represents the current configuration of the EJB project, which requires model analysis of the EJB FSM. Figure 1.1 provides a bottom-to-top view of the process involved in creating the resolved configuration of an EJB project and the context of Model Analysis, represented by the cloud, within the process. Initially, there are three sources of configuration:

Java 5 annotations, XML deployment descriptors, and default values from the EJB specification [16]. The FSML infrastructure generates a project specific EJB FSM via reverse engineering through the FSML infrastructure described in the previous chapter. We then perform model analysis, providing a resolved model as the final product of the process.

### 4.2.1 Meta Model of Resolved Model

Before we can perform FSM model analysis and merging of the information from the various configuration sources, we must determine the structure of the resolved model representing the current configuration. As such, we present the meta model for the resolved model in figure 4.3 using a feature model to facilitate comparison with the EJB feature model presented in Chapter 3. The *clients* feature in the model is left collapsed and simply represents a non-EJB class that contains one or more local or remote EJB references. The *SessionBean* feature is abstract because all session beans must be specified as either stateful or stateless in either the annotations or in a deployment descriptor, so the final configuration would not contain any session beans that have not been further specified in regards to their type. Furthermore, many of the features extend the feature *Bean*, which we extracted from figure 4.3. Because *Bean* is not directly contained by another feature it is considered a Root Feature by the Ecore.FMP's interpretation of the corresponding Ecore class model [50]. Figure 4.4 presents the *Bean* feature. It contains properties that are common to all session beans, message-driven beans, and entities. The meta model also contains some of the properties necessary for antipattern detection, namely, the feature *publicMethods* beneath the *SessionBean* and *Entity* features, the feature *implementsSessionSynchronization* beneath *SessionBean*, and the *EntitiesUsed* feature that applies to all beans and is thus a child feature under the *Bean* feature.

### 4.2.2 Model Analysis

We develop and specify the EJB FSML as an Ecore model, which generates the corresponding Java code and object structure [49], and then later render it as a feature model using the Ecore.FMP tool. The FSM from reverse engineering the project becomes an Ecore instance model, which contains Java object instances. Because the FSM artifacts are already in Java form, we choose to do the model analysis and merging of information in Java. This also gives us the code-generation benefits of Ecore modeling via the EMF framework. We discuss alternative approaches in Section 4.4. We create the meta model presented in Figures 4.3 and 4.4 as Ecore models and then render them as feature models via the Ecore.FMP plugin [50], similar to the EJB FSML. The full Ecore models can be found in Appendix B.

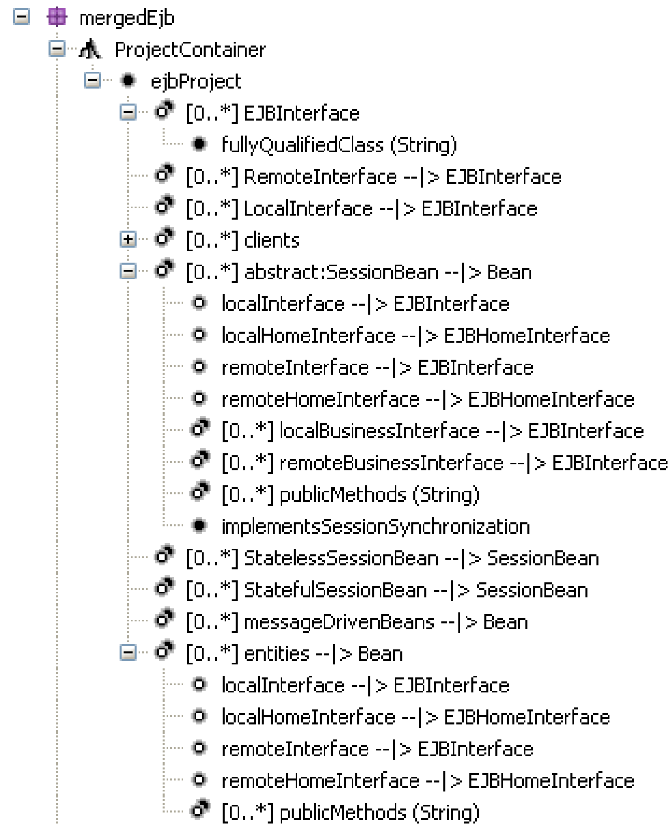


Figure 4.3: Feature Model of Resolved Meta Model

Figure 4.5 provides a high-level overview of the process in which we analyze an EJB FSM and merge its information from annotations and deployment descriptors. It begins with the resolution of clients. This entails finding all the non-EJB bean users by looking at the information from the annotations component of the FSM and seeing what feature instances represent classes that reference beans. After we find clients, the analysis continues by searching the FSM for all valid instances of EJB interfaces. The interfaces discovered from deployment descriptor information via the FSM are trivial to resolve as they are defined explicitly. The same goes for the interfaces contained in the information from annotations component of the FSM that are defined explicitly, that is, the Java interfaces that are annotated with either the *@Local* or the *@Remote* annotations. This interface resolution is complicated in the case of derived local and remote interfaces declared via annotations as it requires that we iterate through each bean in the EJB FSM and we include any derived interfaces found within a bean.

The subsequent 3 steps that follow, namely, resolution of entities, resolution of session beans, and the resolution of message-driven beans, follow the same general process. In Figure 4.5, we present this process to the right and link it to the 3 steps via a dashed line to show the steps' compositions. At a high level, each bean from the FSM is first configured appropriately depending on the source of its configuration. We then add each bean to the resolved model, along with any

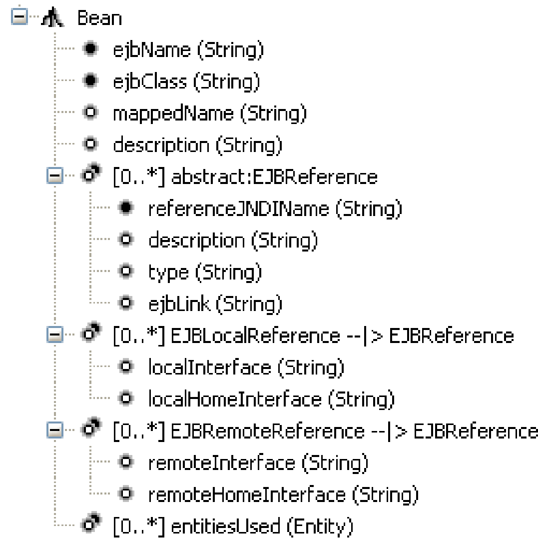


Figure 4.4: Bean Component of Resolved Meta Model

related interfaces, references, and entity uses.

A lower level description of the configuration process is as follows. We iterate through each step-specific bean type that is found within the FSM's information from annotation element. We discover, using the bean class of the annotated bean as a primary key and the EJB bean name provided as a second primary key in case there is no match for the first, any corresponding deployment descriptor entries. This accounts for the case where an annotated bean has properties overwritten by the deployment descriptor. We base the primary search keys on the matching used in the specification [16]. We then configure each of the beans discovered through annotations according to the properties retrieved from annotations and the properties that are overwritten via deployment descriptor, if applicable. We then add any interfaces from either of the applicable sources by going through the interface instances within the FSM related to the current bean. We then resolve entity usage and EJB references similarly. The final step for each individual bean is to configure properties related to antipattern discovery. This varies per bean type. For example, the property of whether or not a bean class implements the SessionSynchronization interface is only relevant in the case of session beans. After we iterate through and evaluate the annotated-step-specific beans from the FSM, we consider the beans defined from deployment descriptors alone individually. Once we confirm the existence of the bean class identified in the deployment descriptor information in the FSM, the process is the same as is done for annotated beans discovered in the FSM without the overwriting check as there is no other source of configuration in this instance. Default values come into play when a non-mandatory value is not present in the FSM in the annotation or deployment descriptor information elements. An example of this is the case where an "enterprise bean's name is not explicitly specified in meta data annotations or in the deployment descriptor, it defaults to the unqualified name of the bean class" as noted in [16].

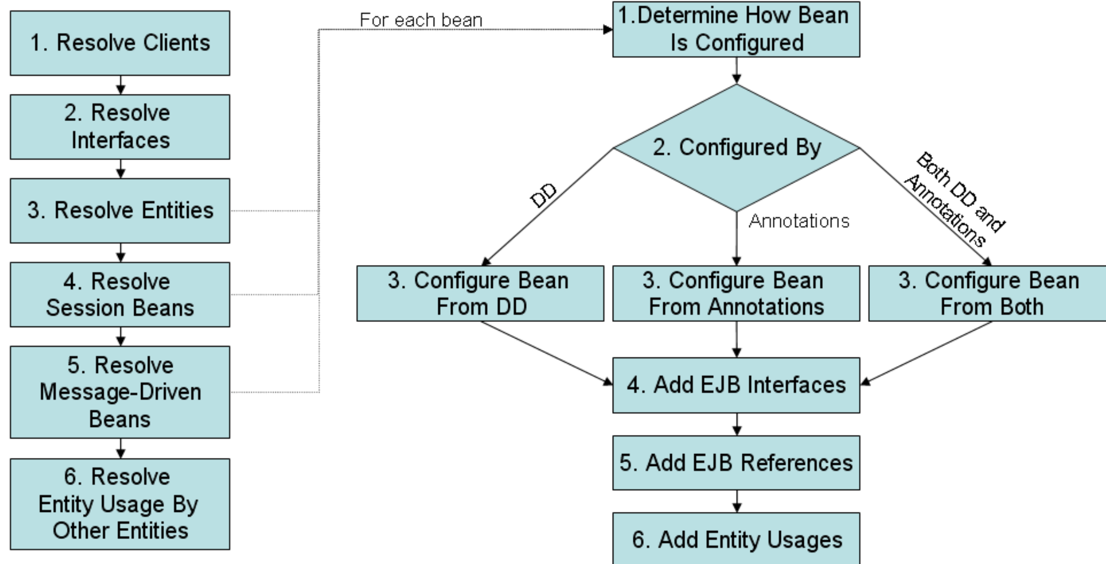


Figure 4.5: Overview of Model Analysis/Merging Process

Note the order of the 3 resolution steps of the three bean types, that is, the fact that entities are resolved before the two other types. The usage of entities is an important consideration for the analysis of any EJB project, as such, some of the usage information is contained within the FSM and the resolved model that follows needs to ascertain all entity usage. So, in order to detect entity usage, one first needs to be aware of all entities in existence within an EJB project, regardless of how they are configured. So, before entity usage is ascertained, we first detect all the entities belonging to the resolved model. This is also the justification for resolving the EJB interfaces before resolving the EJBs themselves. We accomplish entity detection for each bean, the final step in the step-specific bean configuration in Figure 4.5, differently based on the sources that the bean is configured from. If the bean is configured from annotations alone, then we search the annotated Java class' fields and methods to find references to any entities that are already present in the resolved model by checking against the entities' qualified class name. For any beans that have been configured by either a combination of Java annotations and deployment descriptors or deployment descriptors alone, we first check the deployment descriptors for their local and remote reference XML entries as any entity usage for an EJB 2.1 bean must be indicated there [17]. After this is done, we search the corresponding Java class for entities the same way as for beans configured by annotations alone. This method accounts for entities that are configured as either EJB 3.0 or EJB 2.1 beans. We must do entity usage by other entities separately however, as noted by step # 6 on the main (left) flow in Figure 4.5. This is because when we perform entity usage detection during the entity resolution step of the flow, step # 3 of the main flow, we will miss out of any entities that have yet to be resolved. As such, we add this additional step.

## 4.3 Evaluation

### 4.3.1 Sample EJB Projects Tested

Just as we evaluate and correct the EJB FSML manually using the projects listed in Table 3.3, so is the resolved model from FSM analysis. We reverse engineer each of the projects and we view the final/resolved configuration by clicking the Final/Merged view tab in the FSM window. We then manually verify the model by ensuring that only the elements that are supposed be present in the model are present, that is, check for false positives. We then check that each of the elements that are not supposed to be in the model are not, that is, check for false negatives. We then verify manually the properties configured for each element. As shown in figure 4.6, we accomplish this by selecting the element and using the properties view to ensure the analysis executes the correct overriding rules. In this example, the analysis has the name and mapped name of the stateful session bean *StatefulClass* overridden in the deployment descriptor and properly listed in the resolution view, as noted by the values taken from the deployment descriptor. The values within the deployment descriptors end with with the string *FromDD* to make verification easier. So, in this specific case, we see the deployment descriptor had an entry for this specific class and the appropriate overriding took place. We evaluate and verify manually these tests on all the sample projects, with corrections being made to the analysis/resolution accordingly. To ensure we execute all cases, we modify the custom EJB project to have the various situations that will lend itself to cover all overriding scenarios.

## 4.4 Discussion

One possible alternative to providing a resolved view of an EJB project's configuration is to use .QL. As discussed previously, .QL classes encapsulate .QL queries and can be extended much in the way Java classes are. Seeing as .QL is a general query language that has an abstraction mechanism, it is likely that it will be a very good tool for doing the type of merging/analysis that is done in this chapter. From a resolution of configuration perspective, the existing .QL J2EE library [37] can be extended to include information from the deployment descriptor, as .QL also supports XML [14], and the overriding rules can be applied within the extended .QL classes. Appendix C contains the extension of the .QL J2EE library to support EJB 3.0 beans that we create for detecting EJB antipatterns via .QL as we discuss in the subsequent chapter. The beans listed in this extended library can be extended to take in information from the deployment descriptors. For example, the .QL class *SessionBean* can be extended into a class entitled *ResolvedSessionBean* and can have methods that return its configuration values based on the information from both the source and the deployment descriptor(s). To do this, .QL must be able to retrieve the Java 5 annotation information, specifically the annotation's

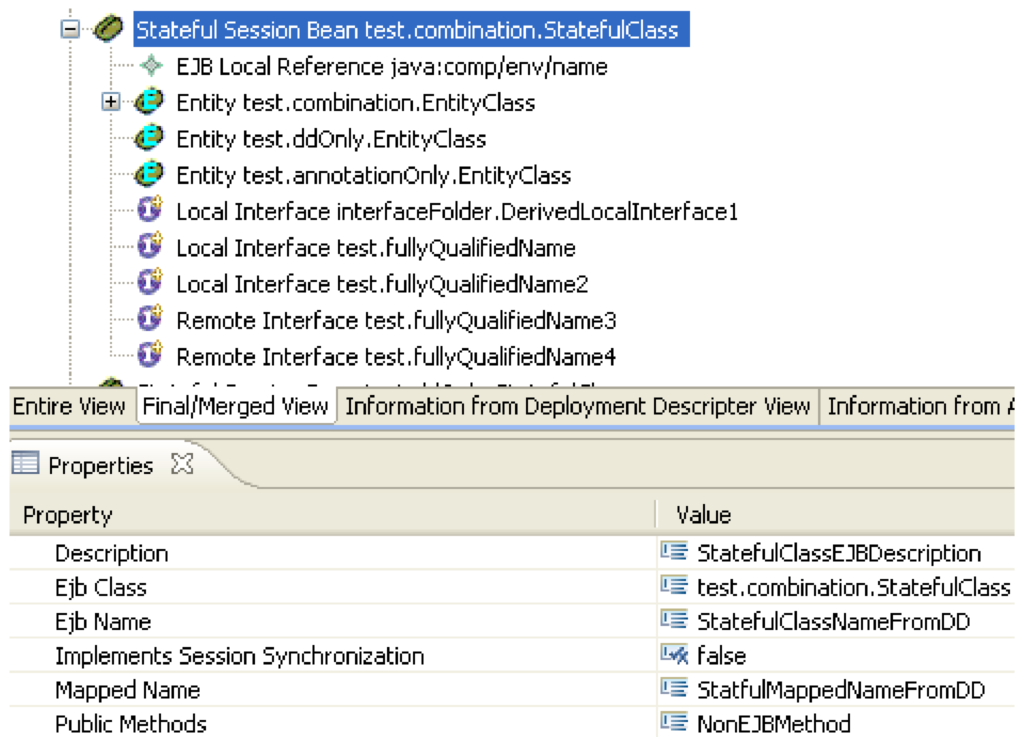


Figure 4.6: Example of a Resolved Session Bean

detail/value pair information, for example, the value of the *name* detail within the Stateless annotation. This functionality is in the .QL documentation and encoded within the .QL libraries, however, when we attempted this it was found that there was a bug in the .QL implementation of this. We informed the .QL developers of this, they have acknowledged the problem, and they have placed the fix within their development build, which will be available sometime in the future during the tool's next release. Once this functionality is available, resolution of an EJB project's configuration can be attempted via .QL.

# Chapter 5

# Detecting Antipattern Instances

This chapter discusses the detection of EJB antipattern instances within an EJB project. Section 5.1 presents the EJB Antipatterns that are used/detected within the scope of this thesis. Section 5.2 discusses how we detect antipattern instances within the resolved models from the model analysis presented in chapter 4 while Section 5.3 demonstrates how antipattern instance detection can be accomplished via the .QL query language. We compare both of the techniques and a new technique is proposed that combines the strengths of both in Section 5.4. We present the evaluation and limitations of the work presented in this chapter in Section 5.5.

## 5.1 Existing Antipatterns

The EJB antipatterns we use for the purpose of this thesis are from Dudney's work on J2EE antipatterns [19]. We choose to use these antipatterns because the book defines them well, provides concrete examples, and explicitly lists the corresponding symptoms for each antipattern. We describe below briefly each of the antipatterns in scope. We present each statically-detectable symptom and then we translate them manually into a predicate logic formula. A more detailed description and more information about them can be found in [19]. We take the of threshold numbers for the antipattern symptoms, for example having more than 20 methods in a Bloated Session, from the Dudney book. We propose in Chapter 6 one possible way of discovering new threshold values.

***Bloated Session***

Bloated Session is used to refer to a session bean that, essentially, attempts to do too much. Ideally, each session bean should deal with only one abstraction/entity, but in the case of a bloated session, the session bean works with multiple abstractions [19]. This antipattern is analogous with the God Object antipattern from the Object-Oriented programming domain that has an Object taking on too much responsibility. Figure 5.1 presents the symptoms associated with the Bloated

Session antipattern. The first symptom to check whether the number of entities used is greater than 1. The second symptom checks whether the session bean has a large number of methods and the third symptom determines if the project contains a small number of session beans relative to entities.

$$1)\exists bean(bean \in ProjectBeans), \forall entity, entity2(entity, entity2 \in ProjectEntities)[entity! = entity2 \wedge bean.Uses(entity) \wedge bean.Uses(entity2)]$$

$$2)\exists bean(bean \in ProjectBeans)[bean.PublicMethodCount > 20]$$

$$3)\exists P(P \in Projects)[P.BeansInProject < P.EntitiesInProject]$$

Figure 5.1: Bloated Session Symptoms in Predicate Logic

***Data Cache***

The Data Cache antipattern deals with the case where one extends a session bean to support data caching [19]. This violates the very nature of session beans: they are intended for a single session and not for multiple users, where caching is most useful. Caching should therefore not be necessary and will create performance problems and possible deadlocking [19]. The symptom for this antipattern, presented in Figure 5.2, is relatively straightforward, as the only way a session bean can perform data caching is when it extends the interface SessionSynchronization in the javax.ejb package.

$$\exists sbean(sbean \in ProjectSessionBeans)\exists interface(interface \in sbean.ImplementedInterfaces)interface = javax.ejb.SessionSynchronization$$

Figure 5.2: Data Cache Symptom in Predicate Logic

***Fragile Links***

Fragile Links is an antipattern where entities or other beans are referenced using hard-coded links, typically within the default EJB context, java:comp/env/ejb/ [19]. This makes projects more susceptible to breakage as beans are not always deployed within the default ejb context. Rather, it is better to add a level of indirection by using an EJB reference alias, which creates a context that decouples the application code reference with the actual location of the bean. The only telltale symptom in this case is the situation where the default EJB location is used. The symptom presented in Figure 5.3 uses of this default location.

$$\forall bean(bean \in ProjectBeans)\exists reference(reference \in ProjectReferences)[reference \subseteq bean.References \wedge (java : comp/env/ejb \subseteq reference.getJNDIName)]$$

Figure 5.3: Fragile Links Symptom in Predicate Logic

***Sessions a Plenty***

Sessions a Plenty is an antipattern that refers to unnecessary uses of session beans. Session beans are intended for transactional processes that work with entities within a single session. This antipattern identifies session beans that are purely algorithmic, such as for currency conversion or product pricing [19]. The symptoms for this antipattern are listed in Figure 5.4. The first symptom reflects the fact that having sessions of an algorithmic nature will result in an inordinate amount of sessions within the project. The second symptom accounts for the fact that sessions that are algorithmic in nature will likely have few public methods. The last symptom notes that a session bean that is purely algorithmic will likely not use any persistence/entities.

$$1)\exists Project(Project \in Projects)Project.SessionBeans > 30$$

$$2)\exists sbean(sbean \in ProjectSessionBeans)sbean.PublicMethodCount < 4$$

$$3)\exists sbean(sbeaninProjectSessionBeans)sbean.EntitiesUsed = \emptyset$$

Figure 5.4: Sessions a Plenty Symptoms in Predicate Logic

#### ***Transparent Facade***

Transparent Facade is a more common EJB antipattern according to [19]. It refers to the case where there is a one-to-one direct mapping of public methods in an entity to those in the corresponding session. Rather than having a useful facade that hides details within an entity, a transparent facade between the session and the entity is created that is essentially useless. The symptom for the transparent facade, demonstrated in Figure 5.5, verifies if the majority of public methods in the session bean are in the entity.

$$\exists sbean(sbean \in ProjectSessionBeans), \exists entity(entity \in sbean.EntitiesUsed)sbean.publicMethodNames \bigcap entity.publicMethodNames \geq (0.75 * entity.PublicMethodCount)$$

Figure 5.5: Transparent Facade Symptom in Predicate Logic

#### ***Thin Session***

Thin Session is an EJB antipattern that refers to sessions that perform only one or a few methods on an entity, that is, sessions are too fine grained [19]. Ideally, only one session should deal with each entity. This antipattern causes problems for program understanding, as logic is spread across multiple sessions, and causes worsened performance of the system because of the large number of sessions. We present the symptoms for the Thin Session antipattern in Figure 5.6. The first symptom ascertains if there is more than one bean that works with a specific entity. The second symptom checks for individual session beans that utilize more than one entity. The last symptom represents the fact that projects with thin sessions will likely have a large number of session beans within it.

$$1)\exists bean, bean2(bean, bean2 \in ProjectBeans)bean! = bean2 \wedge (bean.EntitiesUsed \bigcap bean2.EntitiesUsed = \emptyset)$$

$$2)\exists bean(bean \in ProjectBeans)bean.NumberEntitiesUsed = 1 \wedge bean.NumberOfPublicMethods < 4$$

$$3)\exists Project(Project \in Projects)Project.NumberOfSessionBeans > 30$$

Figure 5.6: Thin Session Symptoms in Predicate Logic

## 5.2 Detecting Antipattern Instances via Resolved EJB Model

Once we perform model analysis on an FSM, we can do antipattern analysis on the EJBs resolved. Like the reasons we chose for doing the analysis in Java, we perform the antipattern detection on the resolved model in Java. That is, because we model the resolved model is as an EMF meta model and the instances of the resolved model are EMF instance models, the artifacts of interest are already Java objects / class instances. To accomplish antipattern instance detection, we use the iterator Java design pattern from [22] to iterate through each of the antipattern detectors and analyze the model and code for the EJB antipatterns. That is, each antipattern detector extends an abstract EJBAntipatternDetector class and provides a specific implementation depending on the antipattern instances of interest. Each concrete antipattern detector registers itself with the collection of EJBAntipatternDetectors and is executed when instructed to via the detect antipattern view.

### 5.2.1 Categorizing Antipattern Symptoms for Analysis

Antipatterns can be categorized along various dimensions. For example, in the J2EE antipattern book [19], the EJB antipatterns are classified based on what specific bean type that the antipattern is applicable to. In [9], the project management antipatterns are classified by the aspects of management that they relate to: people, technology, or process. For the purpose of antipattern detection, however, categorization of the antipatterns themselves is not important. We categorize the symptoms associated with the antipatterns and, specifically, we categorizing the symptoms based on the elements of interest within the symptom. This is helpful from the perspective of the antipattern detectors in that we can scope a particular symptom check at a given time accordingly based on the scope of the symptom. From the developers perspective this categorization is beneficial as it helps them find the symptoms' locations and also assists with traceability to the actual elements in question.

The first category of symptom is the individual-bean symptom. This category consists of symptoms that can be applied to a single bean independently of any other

elements. Straightforward examples of this include the symptom that indicates a bloated session: a single session has an inordinate amount of methods; as well as the symptom for a thin session or sessions-a-plenty antipatterns: a session has a relatively low number of public methods beyond the default EJB ones. While other elements might appear in symptoms belonging to this category, the problem itself exists with a single bean only. An example of this is the symptom indicating a bloated session exists when a session bean works with multiple entities/persistent objects. Even though multiple elements appear within this symptom, the problem exists with only a single session bean and not the entities it references.

Symptoms that describe multiple elements are categorized as Multi-Bean or relational symptoms. Consider the symptom where multiple beans are utilizing the same entity, suggesting that the thin session antipattern is present. In this case, the problem exists with the two or more session beans working on the same entity. Looking at one of the session beans without the other offending session beans would fail to capture the case the symptom is describing, thus this symptom is a multi-bean symptom rather than an individual bean symptom.

Lastly, there exist a number of symptoms that describe antipattern indicators at the project level and, as such, lend themselves to the category of Project-Wide symptoms. These symptoms do not refer to a problem with a bean or a group of beans in particular. They instead refer to antipattern indicators at a project level. Examples of this are the symptoms for the thin and sessions-a-plenty antipattern that check for an inordinate amount of session beans as well as the symptom for the bloated-session antipattern that checks for a relatively low number of session beans within a project relative to the number of entities.

### 5.2.2 EJB Antipattern Meta model

Like the other artifacts in this thesis, the EJB FSML meta model and resolved meta model, we model the antipattern instance meta model as an Ecore model and display it as a feature model, as shown in Figure 5.7. Each project contains 0 to many antipattern instances. Antipatterns have both a name and text description. Each antipattern has 0 to many associated symptoms. Multi-bean symptom instances contain the various beans that match the symptom, while individual-bean instances have the specific bean that matches the properties of symptom.

The meta model is instantiated as follows and produces the results of Figure 5.8, which displays antipattern detection results from antipattern detection run on the custom EJB project in the detect antipattern view. For each antipattern detector registered within the antipattern view, the view creates an antipattern instance and lists its associated symptoms, regardless of whether or not an instance is found. The antipattern detector then checks first for project-wide symptoms on the resolved model in context, followed by multi-bean symptoms and individual bean symptoms. The view adds each symptom instance discovered to the antipattern's list of symptom instances as well as any related information for the symptom, such

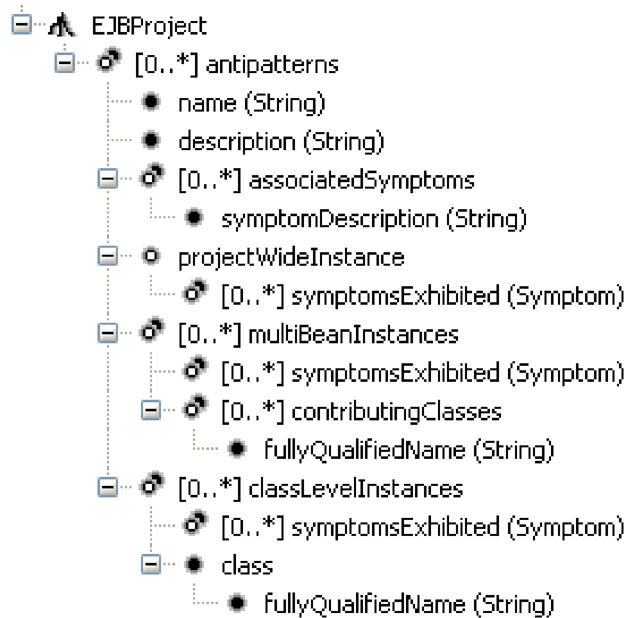


Figure 5.7: Meta Model for Discovered EJB Antipatterns

as the related bean(s). For example, the Sessions-A-Plenty antipattern expanded within the figure lists multiple individual bean instances followed by the possible symptoms associated with it. If one was to expand one one of these instances, it would be similar to the individual bean instance expanded within the Bloated Session antipattern in the figure, which displays the offending class file associated with the bean and the particular bean related to the symptom. In the case of a multi-bean instance, such as the Thin Session antipattern within Figure 5.8, the view displays the multiple offending classes underneath the multi-bean instance along with the associated symptom.

## 5.3 Detecting Antipattern Instances via .QL

Some of the more substantial uses of .QL to date have been auditing code and checking certain rules within Java projects [14]. Because the tool performs static analysis of Java source code, some of the EJB antipatterns detected via analyzing the resolved model from FSM analysis should also be detectable using .QL. As noted in Section 4.4, providing a current resolved view of an EJB project via .QL is not possible at this time, but will can be in the future. Hence, only antipatterns or beans that do not require a resolved view/model can be detected with .QL. That is, antipatterns that deal with any of the configuration properties of beans can not be detected at this time with .QL. The only antipattern identified earlier that fits this category of needing a resolved model showing configuration is the *Fragile Links* antipattern, as the EJB name resolution is a configuration property and can have many sources of configuration. Furthermore, this inability to have a current

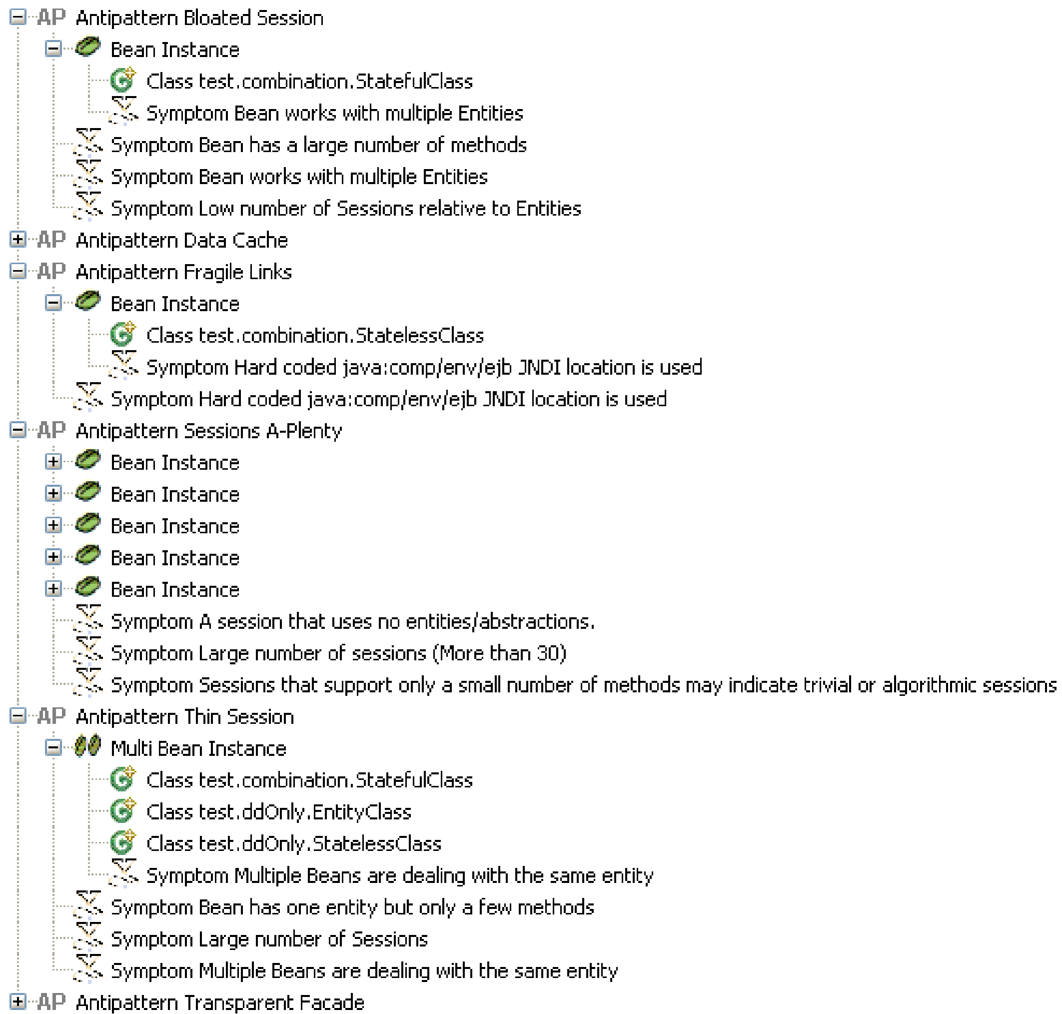


Figure 5.8: Example of Antipattern Instances in Antipattern View

resolved view implies that any EJB 2.1 beans would not be able to be analyzed by the .QL method because EJB 2.1 beans must implement the appropriate interface in the Java Class and must be specified along with its properties in a deployment descriptor [17]. In order to allow for EJB 2.1 beans to be analyzed with .QL, we assume that the implementation of the appropriate EJB 2.1 interface, such as javax.ejb.SessionBean;javax.ejb.EntityBean; or javax.ejb.MessageDrivenBean, by Java classes is sufficient to indicate a valid bean.

As noted at [37], currently the .QL J2EE libraries do not support EJB 3.0 beans. The library accounts only for beans that extend the EJB 2.1 interfaces by identifying beans as the classes that implement the appropriate interface. As such, we extend the library to include EJB 3.0 beans so that these beans can be tested for the EJB antipatterns and we include this library in Appendix C. Specifically, each enterprise bean definition now includes both classes that extend the appropriate EJB 2.1 interface and those classes that annotate themselves with the appropriate EJB 3.0 Java 5 annotation. Furthermore, we create two completely

new .QL classes that represent EJB 3.0 local and remote business interfaces that are annotated appropriately.

Now that we can detect the beans after updating the J2EE .QL library, we can write queries that detect antipatterns through their symptoms. The approach we chose entails writing each symptom for each antipattern as an individual .QL query. The alternative to this is to encapsulate some of the symptoms together in a .QL class. For example, one could write a .QL class that extends the .QL *SessionBean* class called *BloatedSessionBean* with the two symptoms that are the individual-bean symptoms for that antipattern from Figure 5.1. Unfortunately, symptoms that are multi bean or project wide are still required to be individual .QL queries, so it just seems more fitting to stay consistent and have each symptom be an individual .QL query. Even if there was a .QL class like *BloatedSessionAntipatternInstance* that encapsulated all the symptoms/queries, it would be difficult to obtain the granular level of traceability to the specific bean, beans, or elements in question, because it would need to be made clear which symptom(s) are shown to be in existence for the antipattern instances and the symptoms would need to contain the traceability information. The EJB antipattern instance meta model presented earlier in Figure 5.7 did this but it would be a non-trivial task to accomplish in .QL and would also cause a loss of much of the declarative power of .QL as noted in Section 5.4.1 because of this inability to encapsulate all the symptoms and queries.

Figures 5.9, 5.10, 5.11, 5.12, and 5.13 present the .QL queries implemented for the Bloated Session, the Data Cache, the Sessions-A-Plenty, the Thin Session, and the Transparent Facade antipatterns, respectively. In the case of single-bean instance symptoms, the bean in conflict is returned via the select clause. Multi-bean and project-wide instance symptom queries return all the offending beans within the specific symptom and the offending projects discovered, respectively. Each return/select clause also includes some text that helps explain the symptom. All of these queries presented work, except for the project-wide instance ones, which yield a runtime error. The reasoning for this is at this time unknown to us, but we informed the Semmle .QL team of the problem and sent them the extended library in use. They are working on the problem.

## 5.4 Discussion

We next compare the two techniques presented for finding EJB antipattern instances: finding EJB antipattern instances via a resolved model retrieved through analysis of an FSM model and finding EJB antipattern instances detected through .QL. We then propose how to combine aspects of both techniques to achieve the benefits of both.

```
1)
from SessionBean sb
where sb.getNumberOfNonDefaultPublicMethods() > 20
select sb, ''Session Bean contains many public methods indicating a bloated
session.''

2)from SessionBean sb
where sb.getNumberOfEntitiesReference() > 1
select sb, ''Session Bean uses more than one entity.  Session Beans should ideally
 deal with only one abstraction/entity.''

3)from JavaEJBProject jp
where jp.getNumberOfEntities() > jp.getNumberOfSessionBeans()
select jp, ''Java Project has more entities than it has Session Beans, indicating
Bloated Sesssions may be present''
```

Figure 5.9: .QL Queries for Bloated Session Symptoms

```
1) from SessionBean
where sb.getASupertype().getQualifiedName().matches(''javax.ejb.
SessionSynchronization'')
select sb, ''Session Bean should not be implementing the Session Synchronization
interface as it should not be doing data caching.''
```

Figure 5.10: .QL Query for Data Cache Symptom

```
1)from JavaEJBProject jp
where jp.getNumberOfSessionBeans() > 30
select jp, ''Java Project has a large number of sessions (more than 30) indicating
 that too many sessions may be in play.''

2)from SessionBean sb
where sb.getNumberOfNonDefaultPublicMethods() < 4
select sb, ''Sessions that support only a small number of methods may indicate
trivial or algorithmic sessions.''

3)from SessionBean sb
where sb.getNumberOfEntitiesReference() = 0
select sb, ''Session Bean is not dealing with any entities/abstractions and is
likely algorithmic or trival.''
```

Figure 5.11: .QL Queries for Sessions-A-Plenty Symptoms

### 5.4.1 Comparison of Techniques

We first compare how we express symptoms. For resolved models from the FSM, we express antipattern symptoms as Java code that ascertains if a symptom is present. In the .QL method, we express symptoms in the SQL-like .QL format. Note that the .QL method of expressing the symptoms is much more declarative than the Java code representation. As noted in [36], declarative approaches focus on the logic or components of the program irrespective of how the program accomplishes it. Furthermore, declarative programming and modelling enable users to disregard control flow and focus only on what they are attempting to create/model [36]. As such, the .QL approach has an advantage with respect to expressiveness and representation of the antipattern symptoms. This advantage is also apparent when

```
1)from SessionBean sb, EntityBean entityUsed, SessionBean sb2
where depends(sb,entityUsed) and depends(sb2,entityUsed) and sb != sb2
select sb,sb2, ``Multiple Session Beans deal with the same entity implying they
are thin and should be combined.''

2)from SessionBean sb
where sb.getNumberOfNonDefaultPublicMethods() < 4 and sb.
getNumberOfEntitiesReference() = 1
select sb, ``Bean has one entity but only a few methods meaning it may be thin and
 can be combined with another Session Bean.''

3)from JavaEJBProject jp
where jp.getNumberOfSessionBeans() > 30
select jp, ``Java Project has a large number of sessions (more than 30) indicating
 that there may be thin sessions present.''
```

Figure 5.12: .QL Queries for Thin Session Symptoms

```
1)from SessionBean sb
where count (Method m, Method m2 | declaresMember(sb,m) and m.isPublic() and
declaresMember(sb.getEntityUsed(),m2) and m2.isPublic() and m.getSignature().
toLowerCase().matches(m2.getSignature().toLowerCase())
                                            and (not m.getName().matches(``
                                            ejbActivate''))
                                            and (not m.getName().matches(``
                                            ejbPassivate''))
                                            and (not m.getName().matches(``ejbRemove'
                                            '))
                                            and (not m.getName().matches(``
                                            setSessionContext''))
                                            >= (0.75 * sb.getEntityUsed().
                                            getNumberOfNonDefaultPublicMethods()) and
                                             sb.getEntityUsed().
                                            getNumberOfNonDefaultPublicMethods() != 0
select sb, sb.getEntityUsed(), ``The majority of methods in the Session Bean are
in the Entity implying the session bean is a transparent facade"
```

Figure 5.13: .QL Queries for Transparent Facade Symptom

one considers the steps necessary to facilitate user-defined antipatterns for both approaches. The .QL approach already allows this:users can simply write new queries for each new symptom/case they want to check for. For the FSM/resolved model approach, one could create an interface that would use the existing elements of the FSM or resolved model and specify acceptable or unacceptable values for the specific elements. Essentially, this interface would be a customizable model query tool, similar to what .QL is for Java code, that, given a specific FSM or resolved model, a user could specify all the attributes and elements along with the values to search for. Obviously, this is more work than having users write another .QL query.

Each antipattern symptom in the .QL method is analyzed as an individual query, due to the varying categories of the symptoms and the difficulty in encapsulating these queries into a single .QL class. As such, the antipattern instance analysis results are symptoms displayed as single query results. While the results of a query can be output in various ways, such as a table or graph [14], this approach

supports only one query at a time. The FSM method instead allows for an antipattern instance meta model to be configured in a variety of ways assuming the analysis continues to use the appropriate resolved model elements. The results of the FSM method support more structure than those from the .QL method. Another advantage of the FSM approach over the .QL approach is in regards to traceability. The traceability of the FSM approach is at the model level: the antipattern symptom instances presented have a traceability link back to the FSM, which in turn provide a traceability link back to the source. The .QL approach provides traceability to the source code only. It is better to have the option of viewing the antipattern instance at a higher level and then going to a lower one if so desired.

### 5.4.2 Combination of Techniques

After detecting EJB antipattern instances using both methods, it is clear that there should be a way to accomplish antipattern instance detection via an FSML or a model derived from an FSML more declaratively, as done with .QL. The mutation of the EJB FSML in iteration 3 to facilitate antipattern detection presented in Section 3.3 should be possible in a more declarative manner, rather than by specifying properties for antipattern detection and then performing antipattern analysis later. It should be possible to specify the antipatterns and symptoms as elements and mapping types directly within the FSML, rather than having FSML elements that represent the information necessary to detect antipatterns as is done in iteration 3.

Since the antipatterns and symptoms are going to be expressed declaratively within the FSML, the analysis of the FSM to retrieve the resolved model, presented in Chapter 4, will be ignored in this context. That is, as was done for the .QL queries, we focus only on antipatterns that do not deal with configuration properties and assume that extending the appropriate EJB 2.1 interface alone is enough to indicate a valid bean within Java source. We consider only the information from annotation/Java code components in the FSML. In terms of generalizing this approach to other domains and their FSMLs, ignoring FSM analysis and the other stated assumption should not negate the use of this declarative approach to FSML antipattern detection for those domains unless there are similar issues in that domain. For example, this approach could be attempted on the FSMLs in existence such as WPI, struts, and applets [1] as they have no similar FSM analysis requirements.

Figure 5.14 presents how this integrated approach would apply to detecting a bloated session bean. An even better way to accomplish this would be to have the list of symptoms/children be an essential 1 to many (!1..*) feature group, thus the existence of any symptom is necessary for the parent, but such functionality is not yet supported by the current FSML infrastructure presented in [1]. As such, the *bloatedSessionAntipatternExists* mapping type presented in the figure checks

if the bean uses multiple entities or checks if there are an inordinate amount of methods. Beneath this feature are the specific symptoms that have specific mapping types that duplicate the check done for the overall antipattern. The mapping type *multipleEntitiesUsed* checks if there is more than one entity used by the session bean. The mapping type *tooManyPublicMethods* checks if there are too many public methods given a mapping definition that contains the upper bound of methods used, any excluded methods, and if constructors should be excluded by the parameters upperBound(hidden in the feature model since the first parameter is hidden by Ecore.FMP), excludes, and excludesConstructors, respectively. In other words, the parent feature checks for the existence of either symptom and the children recheck for each symptom individually, allowing for the specific symptoms in existence to be identified. Furthermore, if one was to eventually consider strength of symptoms, this method is preferable as the instance of the antipattern itself may not be dependent on the existence of any one symptom but, instead, a specific combination of symptoms. Regarding the third, project-wide, bloated session symptom, we could add a feature that has a mapping type that checks the number of session beans relative to entities as a child under the EJB project feature.

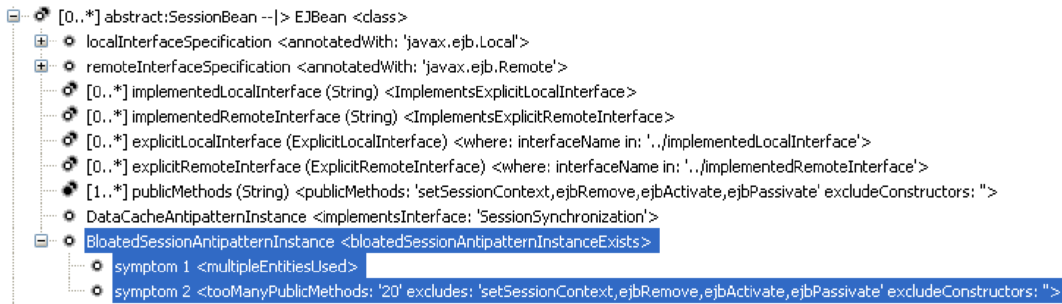


Figure 5.14: FSML elements for Bloated Session Antipattern

A trivial example of this declarative approach is the DataCache antipattern instance. Considering this antipattern has only one symptom that already has an existing mapping type, no new mapping type would be required. As shown in Figure 5.15, only the feature would need to be renamed to indicate whether or not the antipattern is present for the specific bean. The mapping type would remain unchanged as it is already present in the existing FSML infrastructure.

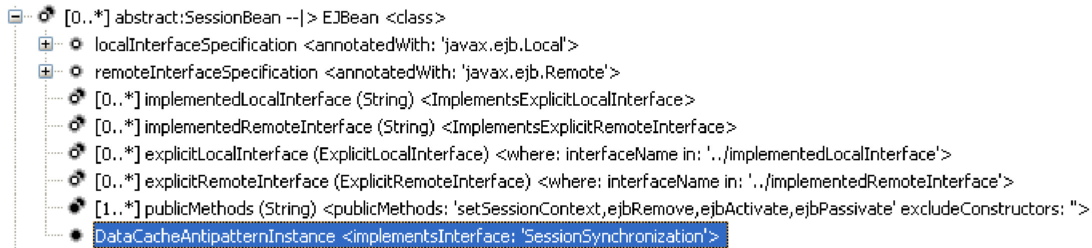


Figure 5.15: FSML elements for Data Cache Antipattern

We now consider multi-bean instance symptoms. In this case, the features

representing the antipattern instances can be placed under the EJB project but they must have non-containment references to the elements of interest within the symptom in order to differentiate itself from a project-wide symptom. Figure 5.16 demonstrates one possible way of specifying the thin session antipattern instance within the EJB FSML. Specifically, the *thinSessionAntipatternInstance* mapping type ascertains if there are any beans that utilize the same entity. The *moreThanOneSessionUsesSameEntity* does the same thing but it also populates its children with the appropriate references to the two or more beans, of type *EJBean*, that share the same entity and the entity being shared. Since there is only one symptom in this case, an alternative to this is to make the single symptom essential, which would require only one check.

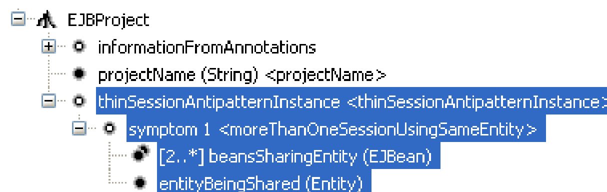


Figure 5.16: FSML elements for Thin Sessions Antipattern

This approach to declaratively modelling antipattern instances and their symptoms applies to other domains if and only if the domain in question requires no intermediate analysis to obtain information necessary for the antipattern instance detection. Furthermore, the antipatterns being detected must exhibit static properties/patterns that can be matched by evaluation of the artifacts in context. Essentially, each antipattern symptom can be a feature with a mapping type containing mapping definitions that specify qualities of interest. These features should be placed in the context that they are relevant in. For example, individual-bean symptoms should be placed within the context of a bean, while Applet-specific antipattern symptoms should be placed within the specific Applet. Ideally, with the implementation of essential feature groups within the FSML infrastructure, an antipattern can be a feature with no mapping type on its own, but which contains a mandatory essential feature group containing its relevant symptoms. We assume that the strength of symptoms with respect to each antipattern are not known or not considered. Chapter 6 considers the implications of strength of symptoms in a declarative approach.

## 5.5 Evaluation

### 5.5.1 Detection Results

We test both of the antipattern instance detection approaches presented in this chapter using the EJB projects presented in Section 4.3.1. Furthermore, we extend

the custom EJB application to ensure that each symptom is properly identified within an EJB project. That is, we included an example of each symptom for each EJB antipattern in the custom EJB project to ensure that it can be detected appropriately. In regards to the open-source projects, table 5.1 presents a listing of the projects and the antipattern instances present. The eMal payment system project contains examples of both bloated and thin sessions. The Webdiary project contains one bean that, as expected, is an example of a bloated session bean. The Redwood logging system contains examples of sessions-a-plenty, thin sessions, and bloated sessions. An interesting case is the Employee Timesheet EJB Project, which has Java class files and XML deployment descriptors but no corresponding Java source. Because there is no source, we could not perform .QL analysis on it. However, we could do antipattern analysis on the resolved model showing that there are instances of bloated, thin, and sessions-a-plenty antipatterns from the information retrieved from the deployment descriptors alone.

Table 5.1: Antipattern Instances detected in Open-Source Projects

| **Project** | **Antipatterns Instances Found** |
|---|---|
| eMal Payment System | 2 bloated beans and 1 thin session bean. |
| WebDiary | 1 bloated bean and project-wide bloated symptom. |
| BellWhistle | 2 Sessions-A-Plenty beans. |
| Storm | Project contains entities only. Tool dectected no antipatterns. |
| Redwood Web Log Mining System | 8 bloated session beans and 1 project-wide symptom. 42 sessions-a-plenty bean symptoms and 1 project-wide symptom. 10 thin session single-bean symptoms, 7 thin session multi-bean symptom, and 1 project-wide. |
| Rowing Club Management Software | Project contains entities only. Tool dectected no antipatterns. |
| Time to Work | 3 bloated session beans and 1 project-wide symptom. 3 session-a-plenty bean symptoms. 4 thin session bean symptoms. |
| WordNet Web Application | The tool detected no antipattern instances. |
| Saeed | The tool detected no antipattern instances. |
| JUnitEJB | The tool detected no antipattern instances. |
| Java Issue Tracker | The tool detected no antipattern instances. |

### 5.5.2 Limitations

One major limitation with all the techniques presented is that none of them account for the strength of the symptoms in terms of how indicative they are of a specific antipattern. That is, no symptom has a stronger implication of an antipattern than any other and only one symptom is necessary to indicate the existence of an instance. Possible techniques for accomplishing this are discussed further in the Future Work chapter, Chapter 6.

#### Categorization of Antipattern Symptoms

The categorization of the transparent facade symptom is somewhat ambiguous and may indicate a weakness/limitation in the categorization: the symptom is both a multi-bean or an individual-bean symptom. The individual-bean symptom

argument is as follows; seeing as the session bean is acting as the facade and the problem is with the facade itself, in that it is transparent, then the symptom should be categorized as an individual-bean symptom. However, the problem need not necessarily lie with the facade alone; it could be that the methods in the entity are at too coarse-grained a level rather than fine grained as entities are intended to be considering they are essentially wrappers or a facade to the data in the underlying database/persistent layer. In either case, the fact that a ambiguous area like this exists may lend itself to the idea that this categorization is not strong enough as ideal categorizations typically do not have any ambiguities.

# Chapter 6

# Future Work

## 6.1 Round-Trip and Refactoring Capabilities

As noted in [1], FSMLs and the FSML infrastructure support both round-trip and forward engineering. The work presented in this thesis utilizes reverse engineering only, that is, going from code/other artifacts to models. Forward engineering entails going from models to code/artifacts and round-trip engineering involves going in both directions and being able to identify conflicts and merge the deltas that arise from each direction. That is, changes to the model must be propagated to the code and vice versa. [1] elaborates on round-trip engineering via an FSML. To facilitate round-trip engineering, the mapping type must contain mappings in both the forward and reverse direction. The EJB FSML created in this thesis implements the reverse engineering mappings only as reverse mappings suffice for the language's goals. That being said, however, value may come out of implementing the forward engineering mappings and this is left for future work. By having both the forward and reverse engineering mappings implemented for this EJB FSML, the language would be much more useful.

Having both mappings would allow EJB developers to migrate all of a project's configuration to a single source of configuration rather than having configuration interspersed between source code and deployment descriptors. So, for example, if an organization prefers to have all its configuration done via deployment descriptors rather than Java annotations or vice versa, then this can be facilitated without changing the final resolved configuration. Furthermore, another possible benefit of implementing the forward mappings is the ability of EJB developers to configure an EJB from scratch through the model rather than ever using code or XML directly. Lastly, having the forward mappings would help facilitate the possibility of refactoring capabilities based on antipattern instance detection. That is, once an antipattern instance has been detected, refactoring solutions can be presented in the form of an FSM instance and the actual refactorings can be accomplished via the appropriate forward mappings.

## 6.2 Detecting New Antipatterns

Another possible extension to the work is the notion of discovering/cataloging new EJB antipatterns beyond those already in existence by analyzing various EJB projects. The main obstacle in this, however, is having a good working sample set of EJB projects to analyze due to the fact that the majority of EJB projects are proprietary. One technique that may be helpful in accomplishing this is the feature/pattern mining work performed in [45]. Feature mining refers to reverse engineering a set of specific configurations to get a probabilistic feature model [45]. Such a feature model has both hard and soft constraints, in addition to the existing components in a feature model, that indicate the probability that a specific feature is in existence given the specific hierarchy preceding it. Essentially, many Java EE projects could be evaluated by the tool and one could use the results to determined what methods are typically overwritten, how attributes are set, et cetera, and this information can then be catalogued in order to derive new EJB antipatterns. Furthermore, one could update the EJB FSML to include the discovered probabilities, if the sample set is determined to be a reasonable sample. Another approach to detecting new antipatterns may be to emulate what is done in the CodeNose tool [46], that is, to have existing symptom instances, or smells, as termed in the context of CodeNose, be detected and then placed through a reasoner to see if any new EJB antipatterns can be derived.

## 6.3 Compare Multiple Projects

Currently, the reverse engineering of an EJB project through the presented EJB FSML is limited to a single EJB project. While this may be sufficient for smaller organizations that have few EJB projects, organizations with many EJB projects may desire cross-project analysis that can help assist with standardization, enforcing code practices, et cetera. Some companies may have typical settings or rules that exist on the majority of their projects that they are unaware of. As such, an extension of the EJB to provide the facility to reverse engineer multiple projects at once may allow for additional analysis to be performed on a single FSM representing multiple projects. This analysis can help determine existing standards being used or help enforce standardization where it is lacking. A concrete example of this is a case in which the majority of EJB projects within scope have their entity-bean-implementing Java classes' fully qualified name begin with the string "entity". For the EJB projects that do not enforce this apparent company standard, the tool would present an option that would allow the appropriate forward engineering to be executed that would update the appropriate artifacts.

## 6.4 Techniques that Account for Strength of Symptoms

One of the major limitations of our techniques, including the declarative FSML antipattern technique, is that the antipattern instance detector fails to account for the strength of symptoms. That is, currently, the existence of any individual symptom belonging to an antipattern is enough to imply the existence of an antipattern instance. Realistically, however, symptoms are likely to be stronger or weaker indicators of their respective antipatterns relative to one another. While some symptoms may indeed be strong enough alone to imply existence, there may be cases in which a symptom requires one or more other symptoms to imply the instance, as the symptom itself does not have sufficient strength/implicative power. One possible way of obtaining these strengths is to use the feature model mining tool presented in [45]. Specifically, projects that are known to contain or projects that are labeled as having particular antipattern instances could be mined, and for each antipattern, the probability that the symptom is present could be used as an indicator of strength. For a concrete example, imagine one has access to many projects with examples of bloated session beans. The individual offending beans can be analyzed to see what percentage have more than one entity being used and what percentage of beans have the number of non-default public methods over a certain threshold. Furthermore, such thresholds could be fine tuned by observing what threshold yields the highest correct bloated session identification percentage. While automatic detection of thresholds sounds ideal in theory, the prerequisite problem of acquiring such projects remains and so does the problem of having someone familiar with each project identify bloated beans based on their familiarity with the project and EJBs in general.

## 6.5 New Detection Technique on Other Frameworks

Lastly, as alluded to earlier, a notable area of exploration is to try the declarative FSML antipattern modeling approach in other areas. While the current FSMLs in existence from [1, 3] do not have an enumerated list of antipatterns in their respective domains like EJBs, there are likely common errors in existence based on user experiences such as those found for applets in [43], Eclipse WPI development [41], and for struts [7, 38]. Based on examples such as these, one could extend the existing FSMLs by declaratively modeling these problems and creating the appropriate mapping types. Also, many of the antipatterns discussed in Section 2.3 for the various domains discussed could be statically detectable if an FSML existed for them. For example, the SQL antipatterns in [31] or the SOA antipatterns from [33] may be able to detected if an FSML was created for those domains. Lastly, as mentioned previously, the book containing the EJB

antipatterns, [19], also contains antipatterns for Servlet and JSP projects so these domains are a viable application domain for the declarative FSML antipattern modeling approach.

# Chapter 7

# Related Work

## 7.1 Java EE Development Tools

IBM's WebSphere Application Server(WAS) is built on the Eclipse framework and supports EJB 3.0 [5]. Its main functionality for EJBs includes creation wizards for session beans and entities, packaging support, automatically generated EJB test clients, and automatic generation of EJB clients [5]. In regards to its configuration checking, however, it supports only individual beans: it will display the properties configured and sources of configuration for a specific bean based on its annotations and/or its deployment descriptor through a properties view on the specific bean [5]. This differs from the solution presented in this thesis because in the WAS solution, each bean must be navigated to individually rather than having project-wide tools that analyze an entire EJB project and showcase all the configuration sources at once or provide an overall resolved view such as the one provided in the solution presented in this thesis.

## 7.2 Static Code Analysis Tools

The antipattern detection work presented in this thesis is similar to a number of existing static code analysis tools. Structural Analysis for Java(SA4J) is one such tool developed by IBM that looks for structural antipatterns within Java programs [28]. Essentially, it calculates what elements within the system will be impacted significantly by changes other elements. It accomplishes this via analyzing program dependencies and identifies structural problems or antipatterns [28]. This work differs from the FSML and .QL approach as SA4J analyzes dependency webs/graphs among components focusing on structure and the amount of impact a change on an element will have on the entire system, something that the approaches in this thesis do not consider. The FSML and .QL approaches do consider structure and dependencies of EJB elements but they also consider properties of individual

constituents within a project as well as properties of the elements, the relationships, and the project itself.

Structural Constraint Language (SCL) is another static code analysis tool that can be used to detect antipatterns in a system. A system expert first specifies constraints on the design of the system using the SCL syntax which is used to express their design intent [26]. Once the model is complete, the developer can run the SCL "conformance checking tool" to ensure that all constraints are not in violation [26]. Like the techniques presented in this thesis, SCL focuses on structure rather than behavior and is also a declarative language. SCL differs from the EJB FSM and .QL antipattern detection approaches in two main ways. Fundamentally, SCL is a constraint language comprised of declarations and formulas, whereas both .QL and the EJB FSM analysis approach are query based. Also, the intent of SCL differs from the presented techniques. SCL is intended as a tool that is to be used during the design phase to specify constraints that should be enforced throughout the implementation of a project [26], whereas the EJB FSM analysis and .QL technique are intended to be used on existing projects. While SCL can add constraints to existing projects, it is more geared toward use in the design phase.

Another tool that is intended to be used on existing projects is the FindBugs tool that discovers bugs in projects [27]. Its "bug detectors" perform analysis at the level of byte code and can even do data and control flow analysis. FindBugs differs from the FSM analysis approach as its analysis is at a much lower level. As noted in [27], the level of analysis performed is at the level of individual class structure/hierarchy and at the linear code level through linear code scans, which is somewhat similar to .QL except for the byte-code analysis aspect.

Lastly, CodeNose is an Eclipse-based tool that detects generic Java code smells and visualizes them within in Eclipse [46]. It utilizes the Java Development Tools (JDT) parser to construct abstract syntax trees representing the code and, thus, forms a structural view of the system. The tool visits each node within the abstract syntax tree and records any primitive smells that can be detected directly within the source [46]. Furthermore, the tool places these primitive smells into a relational algebra reasoner called Grok [25] and Grok derives more complicated code smells. As noted in [46], the code smells targeted are related to individual classes or methods and on dependency problems. Because CodeNose performs analysis at the source code level, it is more related to the .QL approach for antipattern detection than the EJB FSM analysis approach. .QL visits/analyzes the source in the form of database elements, while CodeNose visits abstract syntax trees. It differs from both in that it visits all classes and nodes within the syntax tree and looks for smells at that point, whereas the approaches proposed in this paper scope their searches and perform the analysis on different types of artifacts, namely Java source and XML. It may be possible, however, to extend CodeNose to search only syntax trees that are for EJBs and look for symptoms the way that is currently done for smells.

## 7.3 Dynamic Analysis for EJB Antipatterns

Parsons outlines a prototype in [40] that monitors a running J2EE system and uses data mining techniques to analyze performance and identify EJB performance antipatterns. These antipatterns are detected using rule-based inference on the performance data. The tool works in three phases: monitoring, in which the system is monitored and data is collected; analysis, which parses the XML deployment descriptors to provide context to the currently executing code and uses data mining and statistical techniques; and detection using a rule-engine approach [40]. Parsons' work is similar to ours in that it is working in the same domain as the approaches presented in this paper and the rules used in the Parsons' tool are analogous to the symptoms used within this work. The obvious difference between Parsons' detector and the work presented in this thesis is that Parsons' detector is able to identify the existence of dynamic symptoms only and is able to work with running EJB projects. Combining Parsons' approach in conjunction with the approaches presented in this paper may prove fruitful. For example, in [40], they consider the Sessions-A-Plenty dynamic symptom that states suboptimal performance will result when using sessions unnecessarily. This symptom can not be accounted for in the static techniques presented in this thesis. As such, combining the results yielded from Parsons' approach and results yielded from the techniques presented in this thesis would entail the majority of symptoms being accounted for. The only remaining symptoms would be those containing social/project management implications, such as lengthened change cycles or reduced maintainability.

# Chapter 8

# Conclusion

In this thesis we present an EJB FSML that formalizes a subset of the EJB framework and all the various sources of configuration for EJBs. Using the FSML infrastructure, our tool can generate an FSM that represents a view of the configuration sources for a specific EJB project. This FSM can then, in turn, be analyzed to generate a resolved model/view that represents the current configuration of the underlying EJB project. Developers in an EJB project can now use this work to gain various views on their EJB project: namely the entire configuration view, the view showing annotation configuration information only, the view showing deployment descriptor configuration information only, and the view showing a resolved model that represents the current configuration. The EJB FSML we presented utilizes only the reverse engineering aspect of the FSML infrastructure. Round-trip and forward engineering are a viable extension to the work done in this thesis; only the forward mappings need to be implemented. This extension may prove to be useful in that EJB developers may be able to work at the model level rather than the Java/XML level and may also prove useful for migrating from one source of configuration to another.

This thesis also presented two EJB antipattern instance detection techniques: namely, codified model queries run against the resolved model from FSM analysis and .QL queries written against EJB project source code. The codified model queries provided better-structured results and provided traceability at the model level rather than the source level, but, the .QL queries were easier to develop and are easier to understand, as they are declarative. As such, we propose the notion of a third technique that entails modeling the antipatterns and their symptoms within the original FSML model declaratively. So, rather than ensuring the FSML elements contain the properties necessary for antipattern detection, as was done originally, the antipattern detection itself should be done by modeling the antipatterns and their symptoms declaratively within the FSML, including the corresponding FSML mapping types and definitions.

The antipattern detection techniques presented in this thesis have limitations. Firstly, all of the techniques fail to account for the notion of symptom strength

within the context of a specific antipattern. Furthermore, these techniques are limited to statically-detected symptoms only. However, this work could be combined with work on dynamic-antipattern instance detection to cover a wider range of symptoms.

The antipattern detection work presented in this thesis can be extended to other frameworks. We hope that the results from specifying antipatterns declaratively within FSMLs will prove to be another benefit of modeling a framework as an FSML. Ideally, assuming the appropriate reverse and forward mappings exist within an FSML, a project implementing a framework will be reverse engineered into an FSM including antipattern instances and a tool will present and enforce refactoring solutions in both the FSM and the artifacts it represents. This will be a great step toward developing better quality software projects that utilize object-oriented frameworks.

# APPENDICES

# Appendix A

# Equivalent Ecore Models for EJB FSML Feature Models

This section contains the corresponding XML Ecore Model representations of the Feature Models presented in Chapter 3. Each XML snippet, saved as a .ecore file, is the Ecore model that is rendered as a feature model in the corresponding figure.

## A.1 Iteration 1

### Figure 3.2

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema
    -instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="ejb"
    nsURI="http://ca.uwaterloo.gsd.ejb" nsPrefix="ejb">
  <eClassifiers xsi:type="ecore:EClass" name="InformationFromAnnotations">
    <eStructuralFeatures xsi:type="ecore:EReference" name="ejbInterfaces"
    upperBound="-1"
        eType="#//BusinessInterface" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ejbs" upperBound="-1"
    eType="#//EJBean"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="EJBean" abstract="true" eSuperTypes=
  "platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
    <eAnnotations source="class"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="className" lowerBound="
    1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="fullyQualifiedName"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="BusinessInterface" abstract="true"
  eSuperTypes="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
    <eAnnotations source="class"/>
    <eAnnotations source="parentKey"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="interfaceName"
    lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
```

```
    <eAnnotations source="fullyQualifiedName"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="derivedLocalInterfaces" eSuperTypes=
"#//BusinessInterface">
  <eAnnotations source="parentKey"/>
  <eAnnotations source="indexKey"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DerivedRemoteInterface" eSuperTypes=
"#//BusinessInterface">
  <eAnnotations source="parentKey"/>
  <eAnnotations source="indexKey"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="ExplicitLocalInterface" eSuperTypes=
"#//BusinessInterface">
  <eStructuralFeatures xsi:type="ecore:EReference" name="localAnnotation"
  lowerBound="1"
      eType="#//InterfaceMarkerAnnotation" containment="true">
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.Local"/>
    </eAnnotations>
    <eAnnotations source="essential"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="ExplicitRemoteInterface" eSuperTypes
="#//BusinessInterface">
  <eStructuralFeatures xsi:type="ecore:EReference" name="remoteAnnotation"
  lowerBound="1"
      eType="#//InterfaceMarkerAnnotation" containment="true">
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.Remote"/>
    </eAnnotations>
    <eAnnotations source="essential"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="SessionBean" abstract="true"
eSuperTypes="#//EJBean">
  <eStructuralFeatures xsi:type="ecore:EReference" name="
  localInterfaceSpecification"
      eType="#//SessionBeanLocalInterfaceAnnotation" containment="true">
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.Local"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="
  remoteInterfaceSpecification"
      eType="#//SessionBeanRemoteInterfaceAnnotation" containment="true">
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.Remote"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="
  implementedLocalInterface"
      upperBound="-1" eType="ecore:EDataType␣http://www.eclipse.org/emf/2002/
      Ecore#//EString">
    <eAnnotations source="ImplementsExplicitLocalInterface"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="
  implementedRemoteInterface"
      upperBound="-1" eType="ecore:EDataType␣http://www.eclipse.org/emf/2002/
      Ecore#//EString">
    <eAnnotations source="ImplementsExplicitRemoteInterface"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="explicitLocalInterface"
      upperBound="-1" eType="#//ExplicitLocalInterface">
    <eAnnotations source="where">
      <details key="attribute" value="interfaceName"/>
      <details key="in" value="../implementedLocalInterface"/>
```

```
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="explicitRemoteInterface
    "
        upperBound="-1" eType="#//ExplicitRemoteInterface">
      <eAnnotations source="where">
        <details key="attribute" value="interfaceName"/>
        <details key="in" value="../implementedRemoteInterface"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="StatelessEJB" eSuperTypes="#//
  SessionBean">
    <eStructuralFeatures xsi:type="ecore:EReference" name="statelessAnnotation"
    lowerBound="1"
        eType="#//BeanClassAnnotation" containment="true">
      <eAnnotations source="essential"/>
      <eAnnotations source="annotatedWith">
        <details key="fullyQualifiedType" value="javax.ejb.Stateless"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="StatefulEJB" eSuperTypes="#//
  SessionBean">
    <eStructuralFeatures xsi:type="ecore:EReference" name="statefulAnnotation"
    lowerBound="1"
        eType="#//BeanClassAnnotation" containment="true">
      <eAnnotations source="essential"/>
      <eAnnotations source="annotatedWith">
        <details key="fullyQualifiedType" value="javax.ejb.Stateful"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="MessageDrivenEJB" eSuperTypes="#//
  EJBean">
    <eStructuralFeatures xsi:type="ecore:EReference" name="messageDrivenAnnotation
    "
        lowerBound="1" eType="#//BeanClassAnnotation" containment="true">
      <eAnnotations source="essential"/>
      <eAnnotations source="annotatedWith">
        <details key="fullyQualifiedType" value="javax.ejb.MessageDriven"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Entity" eSuperTypes="#//EJBean">
    <eStructuralFeatures xsi:type="ecore:EReference" name="entityAnnotation"
    lowerBound="1"
        eType="#//EntityAnnotation" containment="true">
      <eAnnotations source="essential"/>
      <eAnnotations source="annotatedWith">
        <details key="fullyQualifiedType" value="javax.persistence.Entity"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="BeanClassAnnotation" eSuperTypes="
  platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
    <eAnnotations source="annotation"/>
    <eAnnotations source="parentKey"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="
    ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="attribute">
        <details key="attributeName" value="name"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="mappedName" eType="
    ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="attribute">
        <details key="attributeName" value="mappedName"/>
```

```
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" eType="
    ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="attribute">
        <details key="attributeName" value="description"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="EntityAnnotation" eSuperTypes="
  platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
    <eAnnotations source="annotation"/>
    <eAnnotations source="parentKey"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="
    ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="attribute">
        <details key="attributeName" value="name"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="InterfaceMarkerAnnotation"
  eSuperTypes="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
    <eAnnotations source="annotation"/>
    <eAnnotations source="parentKey"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="isMarker" lowerBound="1
    "
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean">
      <eAnnotations source="essential"/>
      <eAnnotations source="hasNoAttribute"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="SessionBeanLocalInterfaceAnnotation"
      eSuperTypes="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//
      Concept">
    <eAnnotations source="annotation"/>
    <eAnnotations source="parentKey"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="derivedLocalInterfaces"
        lowerBound="1" upperBound="-1" eType="#//derivedLocalInterfaces"
        containment="true">
      <eAnnotations source="essential"/>
      <eAnnotations source="attribute">
        <details key="attributeName" value="value"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="SessionBeanRemoteInterfaceAnnotation
  "
      eSuperTypes="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//
      Concept">
    <eAnnotations source="annotation"/>
    <eAnnotations source="parentKey"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="derivedRemoteInterfaces
    "
        lowerBound="1" upperBound="-1" eType="#//DerivedRemoteInterface"
        containment="true">
      <eAnnotations source="essential"/>
      <eAnnotations source="attribute">
        <details key="attributeName" value="value"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
</ecore:EPackage>
```

### Figure 3.3

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
```

```xml
  xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema
  -instance"
  xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="ejb"
  nsURI="http://ca.uwaterloo.gsd.ejb" nsPrefix="ejb">
<eClassifiers xsi:type="ecore:EClass" name="InformationFromDeploymentDescriptor"
    eSuperTypes="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//
    Concept">
  <eAnnotations source="xmlDocument">
    <details key="fileName" value="ejb-jar.xml"/>
  </eAnnotations>
  <eAnnotations source="xmlElement">
    <details key="name" value="ejb-jar"/>
  </eAnnotations>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="file" lowerBound="1"
  eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="documentPath"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="sessionBeans"
  upperBound="-1"
      eType="#//DDSessionBean" containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="enterprise-beans/session"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="entityBeans" upperBound
  ="-1"
      eType="#//DDEntityBean" containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="enterprise-beans/entity"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="messageDrivenBeans"
  upperBound="-1"
      eType="#//DDMessageDrivenBean" containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="enterprise-beans/message-driven"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDBean" abstract="true" eSuperTypes=
"platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
  <eAnnotations source="xmlElement"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="ejbName" lowerBound="1"
      eType="#//DDEJBName" containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="ejb-name"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="ejbClass" eType="#//
  DDEJBClass"
      containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="ejb-class"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="mappedName" eType="#//
  DDEJBMappedName"
      containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="mapped-name"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="description" eType="#//
  DDDescription"
      containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="description"/>
    </eAnnotations>
```

```
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDEJBName">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ejbName" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="xmlElementValue"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDEJBClass">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ejbClass" lowerBound="1
    "
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="xmlElementValue"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDEJBMappedName">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="mappedName" lowerBound=
    "1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="xmlElementValue"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDDescription">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" lowerBound
    ="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="xmlElementValue"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDNoTypeSessionBean" eSuperTypes="
#//DDSessionBean">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="noSessionTypeElement"
    lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean">
      <eAnnotations source="noXMLElement">
        <details key="path" value="session-type"/>
      </eAnnotations>
      <eAnnotations source="essential"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDSessionBean" abstract="true"
eSuperTypes="#//DDBean">
    <eStructuralFeatures xsi:type="ecore:EReference" name="localBusinessInterfaces
    "
        upperBound="-1" eType="#//DDLocalInterface" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="business-local"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
    remoteBusinessInterfaces"
        upperBound="-1" eType="#//DDRemoteInterface" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="business-remote"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDStatefulEJB" eSuperTypes="#//
DDSessionBean">
```

```
  <eStructuralFeatures xsi:type="ecore:EReference" name="sessionType" lowerBound
  ="1"
      eType="#//DDStatefulSessionType" containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="session-type"/>
    </eAnnotations>
    <eAnnotations source="essential"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDStatelessEJB" eSuperTypes="#//
DDSessionBean">
  <eStructuralFeatures xsi:type="ecore:EReference" name="sessionType" lowerBound
  ="1"
      eType="#//DDStatelessSessionType" containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="session-type"/>
    </eAnnotations>
    <eAnnotations source="essential"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDStatefulSessionType">
  <eAnnotations source="xmlElement"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="isStatefulSessionType"
      lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
      Ecore#//EBoolean">
    <eAnnotations source="xmlElementValueEqualsString">
      <details key="StringToSearchFor" value="Stateful"/>
    </eAnnotations>
    <eAnnotations source="key"/>
    <eAnnotations source="essential"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDStatelessSessionType">
  <eAnnotations source="xmlElement"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="isStatelessSessionType"
      lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
      Ecore#//EBoolean">
    <eAnnotations source="xmlElementValueEqualsString">
      <details key="StringToSearchFor" value="Stateless"/>
    </eAnnotations>
    <eAnnotations source="key"/>
    <eAnnotations source="essential"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDEntityBean" eSuperTypes="#//DDBean
">
  <eStructuralFeatures xsi:type="ecore:EReference" name="localBusinessInterfaces
  "
      upperBound="-1" eType="#//DDLocalInterface" containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="business-local"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="
  remoteBusinessInterfaces"
      upperBound="-1" eType="#//DDRemoteInterface" containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="business-remote"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDMessageDrivenBean" eSuperTypes="
#//DDBean"/>
<eClassifiers xsi:type="ecore:EClass" name="DDLocalInterface">
  <eAnnotations source="xmlElement"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="
  fullyQualifiedInterfaceName"
```

```
        lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
        Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDRemoteInterface">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
    fullyQualifiedInterfaceName"
        lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
        Ecore#//EString"/>
  </eClassifiers>
</ecore:EPackage>
```

## A.2 Iteration 2

### Figure 3.4

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema
    -instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="ejb"
    nsURI="http://ca.uwaterloo.gsd.ejb" nsPrefix="ejb">
  <eClassifiers xsi:type="ecore:EClass" name="InformationFromDeploymentDescriptor"
      eSuperTypes="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//
      Concept">
    <eAnnotations source="xmlDocument">
      <details key="fileName" value="ejb-jar.xml"/>
    </eAnnotations>
    <eAnnotations source="xmlElement">
      <details key="name" value="ejb-jar"/>
    </eAnnotations>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="file" lowerBound="1"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="documentPath"/>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="sessionBeans"
    upperBound="-1"
        eType="#//DDSessionBean" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="enterprise-beans/session"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="entityBeans" upperBound
    ="-1"
        eType="#//DDEntityBean" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="enterprise-beans/entity"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="messageDrivenBeans"
    upperBound="-1"
        eType="#//DDMessageDrivenBean" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="enterprise-beans/message-driven"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDBean" abstract="true" eSuperTypes=
  "platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ejbName" lowerBound="1"
        eType="#//DDEJBName" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="ejb-name"/>
```

```
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ejbClass" eType="#//
    DDEJBClass"
        containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="ejb-class"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="mappedName" eType="#//
    DDEJBMappedName"
        containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="mapped-name"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="description" eType="#//
    DDDescription"
        containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="description"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="remoteReference"
    upperBound="-1"
        eType="#//DDRemoteReference" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="ejb-ref"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="localReference"
    upperBound="-1"
        eType="#//DDLocalReference" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="ejb-local-ref"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDEJBName">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ejbName" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="xmlElementValue"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDEJBClass">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ejbClass" lowerBound="1
    "
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="xmlElementValue"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDEJBMappedName">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="mappedName" lowerBound=
    "1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="xmlElementValue"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDDescription">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" lowerBound
    ="1"
```

```
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="xmlElementValue"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDReference" abstract="true">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="description" upperBound
    ="-1"
        eType="#//DDDescription" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="description"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="refName" lowerBound="1"
        eType="#//DDEJBRefName" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="ejb-ref-name"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="refType" eType="#//
    DDEJBRefType"
        containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="ejb-ref-type"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="JNDIName" lowerBound="1
    "
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="concatenate">
        <details key="element" value="../refName/refName"/>
        <details key="string" value="java:comp/env/"/>
        <details key="position" value="before"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ejbLink" eType="#//
    DDEJBLink"
        containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="ejb-link"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDEJBLink">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="link" lowerBound="1"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="xmlElementValue"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDLocalReference" eSuperTypes="#//
DDReference">
    <eStructuralFeatures xsi:type="ecore:EReference" name="localInterface" eType="
    #//DDLocalInterface"
        containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="local"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="localHomeInterface"
    eType="#//DDEJBLocalHomeInterface"
        containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="local-home"/>
      </eAnnotations>
    </eStructuralFeatures>
```

```
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDRemoteReference" eSuperTypes="#//
DDReference">
  <eStructuralFeatures xsi:type="ecore:EReference" name="remoteInterface" eType=
  "#//DDRemoteInterface"
      containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="remote"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="remoteHomeInterface"
  eType="#//DDEJBRemoteHomeInterface"
      containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="home"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDEJBRefName">
  <eAnnotations source="xmlElement"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="refName" lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="xmlElementValue"/>
    <eAnnotations source="key"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDEJBRefType">
  <eAnnotations source="xmlElement"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="refType" lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="xmlElementValue"/>
    <eAnnotations source="key"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDEJBRemoteHomeInterface">
  <eAnnotations source="xmlElement"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="remoteHomeInterface"
  lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="xmlElementValue"/>
    <eAnnotations source="key"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDEJBLocalHomeInterface">
  <eAnnotations source="xmlElement"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="localHomeInterface"
  lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="xmlElementValue"/>
    <eAnnotations source="key"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDNoTypeSessionBean" eSuperTypes="
#//DDSessionBean">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="noSessionTypeElement"
  lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean">
    <eAnnotations source="noXMLElement">
      <details key="path" value="session-type"/>
    </eAnnotations>
    <eAnnotations source="essential"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDSessionBean" abstract="true"
eSuperTypes="#//DDBean">
  <eStructuralFeatures xsi:type="ecore:EReference" name="ddLocalInterface" eType
  ="#//DDLocalInterface"
      containment="true">
```

```
      <eAnnotations source="xmlElements">
        <details key="path" value="local"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ddRemoteInterface"
    eType="#//DDRemoteInterface"
        containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="remote"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="localBusinessInterfaces
    "
        upperBound="-1" eType="#//DDLocalInterface" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="business-local"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="
    remoteBusinessInterfaces"
        upperBound="-1" eType="#//DDRemoteInterface" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="business-remote"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="remoteHomeInterface"
    eType="#//DDEJBRemoteHomeInterface"
        containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="home"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EReference" name="localHomeInterface"
    eType="#//DDEJBLocalHomeInterface"
        containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="local-home"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDStatefulEJB" eSuperTypes="#//
  DDSessionBean">
    <eStructuralFeatures xsi:type="ecore:EReference" name="sessionType" lowerBound
    ="1"
        eType="#//DDStatefulSessionType" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="session-type"/>
      </eAnnotations>
      <eAnnotations source="essential"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDStatelessEJB" eSuperTypes="#//
  DDSessionBean">
    <eStructuralFeatures xsi:type="ecore:EReference" name="sessionType" lowerBound
    ="1"
        eType="#//DDStatelessSessionType" containment="true">
      <eAnnotations source="xmlElements">
        <details key="path" value="session-type"/>
      </eAnnotations>
      <eAnnotations source="essential"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDStatefulSessionType">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="isStatefulSessionType"
        lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
        Ecore#//EBoolean">
      <eAnnotations source="xmlElementValueEqualsString">
```

```
      <details key="StringToSearchFor" value="Stateful"/>
    </eAnnotations>
    <eAnnotations source="key"/>
    <eAnnotations source="essential"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDStatelessSessionType">
  <eAnnotations source="xmlElement"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="isStatelessSessionType"
      lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
      Ecore#//EBoolean">
    <eAnnotations source="xmlElementValueEqualsString">
      <details key="StringToSearchFor" value="Stateless"/>
    </eAnnotations>
    <eAnnotations source="key"/>
    <eAnnotations source="essential"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDEntityBean" eSuperTypes="#//DDBean
">
  <eStructuralFeatures xsi:type="ecore:EReference" name="ddLocalInterface" eType
  ="#//DDLocalInterface"
      containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="local"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="ddRemoteInterface"
  eType="#//DDRemoteInterface"
      containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="remote"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="localBusinessInterfaces
  "
      upperBound="-1" eType="#//DDLocalInterface" containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="business-local"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="
  remoteBusinessInterfaces"
      upperBound="-1" eType="#//DDRemoteInterface" containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="business-remote"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="remoteHomeInterface"
  eType="#//DDEJBRemoteHomeInterface"
      containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="home"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="localHomeInterface"
  eType="#//DDEJBLocalHomeInterface"
      containment="true">
    <eAnnotations source="xmlElements">
      <details key="path" value="local-home"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DDMessageDrivenBean" eSuperTypes="
#//DDBean"/>
<eClassifiers xsi:type="ecore:EClass" name="DDLocalInterface">
  <eAnnotations source="xmlElement"/>
```

```
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
    fullyQualifiedInterfaceName"
        lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
        Ecore#//EString">
      <eAnnotations source="xmlElementValue"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DDRemoteInterface">
    <eAnnotations source="xmlElement"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
    fullyQualifiedInterfaceName"
        lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
        Ecore#//EString">
      <eAnnotations source="xmlElementValue"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
  </eClassifiers>
</ecore:EPackage>
```

## A.3 Iteration 3

### Figure 3.6 and Figure 3.7 Combined

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema
    -instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="ejb"
    nsURI="http://ca.uwaterloo.gsd.ejb" nsPrefix="ejb">
  <eClassifiers xsi:type="ecore:EClass" name="InformationFromAnnotations">
    <eStructuralFeatures xsi:type="ecore:EReference" name="ejbInterfaces"
    upperBound="-1"
        eType="#//BusinessInterface" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="clients" upperBound="-1
    "
        eType="#//EJBClient" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ejbs" upperBound="-1"
    eType="#//EJBean"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="EJBClient" eSuperTypes="platform:/
  plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
    <eAnnotations source="class"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="className" lowerBound="
    1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
      <eAnnotations source="fullyQualifiedName"/>
      <eAnnotations source="key"/>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
    hasFieldAnnotatedWithEJB"
        lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
        Ecore#//EBoolean">
      <eAnnotations source="essential"/>
      <eAnnotations source="hasFieldAnnotatedWith">
        <details key="fullyQualifiedType" value="javax.ejb.EJB"/>
      </eAnnotations>
    </eStructuralFeatures>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="isNotAnEJB" lowerBound=
    "1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean">
      <eAnnotations source="essential"/>
      <eAnnotations source="isNotAnEJB"/>
```

```
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="fieldsAnnotatedWithEJB"
      lowerBound="1" upperBound="-1" eType="#//FieldAnnotatedWithEJB"
      containment="true"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="EJBean" abstract="true" eSuperTypes=
"platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
  <eAnnotations source="class"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="className" lowerBound="
  1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="fullyQualifiedName"/>
    <eAnnotations source="key"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="fieldsAnnotatedWithEJB"
      upperBound="-1" eType="#//FieldAnnotatedWithEJB" containment="true"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="entityFields"
  upperBound="-1"
      eType="#//FieldOfEntityClass" containment="true"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="entityLocalVariables"
  upperBound="-1"
      eType="#//LocalVariableOfEntityClass" containment="true"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="FieldAnnotatedWithEJB" eSuperTypes="
platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
  <eAnnotations source="field"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="fieldName" lowerBound="
  1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="fieldName"/>
    <eAnnotations source="key"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="EJBInterfaceAnnotation"
      lowerBound="1" eType="#//DependencyEJBAnnotation" containment="true">
    <eAnnotations source="essential"/>
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.EJB"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="FieldOfEntityClass" eSuperTypes="
platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
  <eAnnotations source="field"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="fieldName" lowerBound="
  1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="fieldName"/>
    <eAnnotations source="key"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="fieldType" lowerBound="
  1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="fieldType"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="entityReference"
  lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean">
    <eAnnotations source="essential"/>
    <eAnnotations source="fieldTypeAnnotatedWith">
      <details key="fullyQualifiedType" value="javax.persistence.Entity"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="LocalVariableOfEntityClass"
eSuperTypes="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
  <eAnnotations source="localVariable"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="localVariableName"
  lowerBound="1"
```

```
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="localVariableName"/>
    <eAnnotations source="key"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="localVariableType"
  lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="localVariableType"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="entityReference"
  lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean">
    <eAnnotations source="essential"/>
    <eAnnotations source="localVariableTypeAnnotatedWith">
      <details key="fullyQualifiedType" value="javax.persistence.Entity"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="BusinessInterface" abstract="true"
eSuperTypes="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
  <eAnnotations source="class"/>
  <eAnnotations source="parentKey"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="interfaceName"
  lowerBound="1"
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="fullyQualifiedName"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DerivedLocalInterface" eSuperTypes="
#//BusinessInterface">
  <eAnnotations source="parentKey"/>
  <eAnnotations source="indexKey"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DerivedRemoteInterface" eSuperTypes=
"#//BusinessInterface">
  <eAnnotations source="parentKey"/>
  <eAnnotations source="indexKey"/>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="ExplicitLocalInterface" eSuperTypes=
"#//BusinessInterface">
  <eStructuralFeatures xsi:type="ecore:EReference" name="localAnnotation"
  lowerBound="1"
      eType="#//InterfaceMarkerAnnotation" containment="true">
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.Local"/>
    </eAnnotations>
    <eAnnotations source="essential"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="ExplicitRemoteInterface" eSuperTypes
="#//BusinessInterface">
  <eStructuralFeatures xsi:type="ecore:EReference" name="remoteAnnotation"
  lowerBound="1"
      eType="#//InterfaceMarkerAnnotation" containment="true">
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.Remote"/>
    </eAnnotations>
    <eAnnotations source="essential"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="SessionBean" abstract="true"
eSuperTypes="#//EJBean">
  <eStructuralFeatures xsi:type="ecore:EReference" name="
  localInterfaceSpecification"
      eType="#//SessionBeanLocalInterfaceAnnotation" containment="true">
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.Local"/>
    </eAnnotations>
```

```
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="
  remoteInterfaceSpecification"
      eType="#//SessionBeanRemoteInterfaceAnnotation" containment="true">
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.Remote"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="
  implementedLocalInterface"
      upperBound="-1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
      Ecore#//EString">
    <eAnnotations source="ImplementsExplicitLocalInterface"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="
  implementedRemoteInterface"
      upperBound="-1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
      Ecore#//EString">
    <eAnnotations source="ImplementsExplicitRemoteInterface"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="explicitLocalInterface"
      upperBound="-1" eType="#//ExplicitLocalInterface">
    <eAnnotations source="where">
      <details key="attribute" value="interfaceName"/>
      <details key="in" value="../implementedLocalInterface"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="explicitRemoteInterface
  "
      upperBound="-1" eType="#//ExplicitRemoteInterface">
    <eAnnotations source="where">
      <details key="attribute" value="interfaceName"/>
      <details key="in" value="../implementedRemoteInterface"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="publicMethods"
  lowerBound="1"
      upperBound="-1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
      Ecore#//EString">
    <eAnnotations source="publicMethods">
      <details key="excludes" value="setSessionContext,ejbRemove,ejbActivate,
      ejbPassivate"/>
      <details key="excludeConstructors" value=""/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="
  implementsSessionSynchronization"
      lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
      Ecore#//EBoolean">
    <eAnnotations source="implementsInterface">
      <details key="name" value="javax.ejb.SessionSynchronization"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="StatelessEJB" eSuperTypes="#//
SessionBean">
  <eStructuralFeatures xsi:type="ecore:EReference" name="statelessAnnotation"
  lowerBound="1"
      eType="#//BeanClassAnnotation" containment="true">
    <eAnnotations source="essential"/>
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.Stateless"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="StatefulEJB" eSuperTypes="#//
SessionBean">
```

```
  <eStructuralFeatures xsi:type="ecore:EReference" name="statefulAnnotation"
  lowerBound="1"
      eType="#//BeanClassAnnotation" containment="true">
    <eAnnotations source="essential"/>
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.Stateful"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="MessageDrivenEJB" eSuperTypes="#//
EJBean">
  <eStructuralFeatures xsi:type="ecore:EReference" name="messageDrivenAnnotation
  "
      lowerBound="1" eType="#//BeanClassAnnotation" containment="true">
    <eAnnotations source="essential"/>
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.ejb.MessageDriven"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="Entity" eSuperTypes="#//EJBean">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="publicMethods"
  lowerBound="1"
      upperBound="-1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
      Ecore#//EString">
    <eAnnotations source="publicMethods">
      <details key="excludes" value="setEntityContext,unSetEntityContext,
      ejbRemove,ejbActivate,ejbPassivate,ejbLoad,ejbStore"/>
      <details key="excludeConstructors" value=""/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EReference" name="entityAnnotation"
  lowerBound="1"
      eType="#//EntityAnnotation" containment="true">
    <eAnnotations source="essential"/>
    <eAnnotations source="annotatedWith">
      <details key="fullyQualifiedType" value="javax.persistence.Entity"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="BeanClassAnnotation" eSuperTypes="
platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
  <eAnnotations source="annotation"/>
  <eAnnotations source="parentKey"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="
  ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="attribute">
      <details key="attributeName" value="name"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="mappedName" eType="
  ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="attribute">
      <details key="attributeName" value="mappedName"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" eType="
  ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="attribute">
      <details key="attributeName" value="description"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="DependencyEJBAnnotation" eSuperTypes
="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
  <eAnnotations source="annotation"/>
  <eAnnotations source="parentKey"/>
```

```
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="referenceName" eType="
  ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="referenceNameEJBAttribute"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="beanName" eType="
  ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"
      defaultValueLiteral="">
    <eAnnotations source="attribute">
      <details key="attributeName" value="beanName"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="beanInterface" eType="
  ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="beanInterfaceEJBAttribute"/>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="mappedName" eType="
  ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="attribute">
      <details key="attributeName" value="mappedName"/>
    </eAnnotations>
  </eStructuralFeatures>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" eType="
  ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="attribute">
      <details key="attributeName" value="description"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="EntityAnnotation" eSuperTypes="
platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
  <eAnnotations source="annotation"/>
  <eAnnotations source="parentKey"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="
  ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString">
    <eAnnotations source="attribute">
      <details key="attributeName" value="name"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="InterfaceMarkerAnnotation"
eSuperTypes="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//Concept">
  <eAnnotations source="annotation"/>
  <eAnnotations source="parentKey"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="isMarker" lowerBound="1
  "
      eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean">
    <eAnnotations source="essential"/>
    <eAnnotations source="hasNoAttribute"/>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="SessionBeanLocalInterfaceAnnotation"
    eSuperTypes="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//
    Concept">
  <eAnnotations source="annotation"/>
  <eAnnotations source="parentKey"/>
  <eStructuralFeatures xsi:type="ecore:EReference" name="localInterfaces"
  lowerBound="1"
      upperBound="-1" eType="#//DerivedLocalInterface" containment="true">
    <eAnnotations source="essential"/>
    <eAnnotations source="attribute">
      <details key="attributeName" value="value"/>
    </eAnnotations>
  </eStructuralFeatures>
</eClassifiers>
<eClassifiers xsi:type="ecore:EClass" name="SessionBeanRemoteInterfaceAnnotation
"
    eSuperTypes="platform:/plugin/ca.uwaterloo.gsd.fsml/model/fsml.ecore#//
    Concept">
```

```
    <eAnnotations source="annotation"/>
    <eAnnotations source="parentKey"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="remoteInterfaces"
    lowerBound="1"
        upperBound="-1" eType="#//DerivedRemoteInterface" containment="true">
      <eAnnotations source="essential"/>
      <eAnnotations source="attribute">
        <details key="attributeName" value="value"/>
      </eAnnotations>
    </eStructuralFeatures>
  </eClassifiers>
</ecore:EPackage>
```

# Appendix B

# Equivalent Ecore Model for Resolved Meta Model Feature Models

The following is the XML source of the Ecore Model represented by the Feature Model presented in both Figure 4.3 and Figure 4.4, with Figure 4.4 being the bean root feature extracted from the main feature model presented in Figure 4.3. The following XML, saved as a .ecore file, is the Ecore model for the resolved configuration meta model.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema
    -instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="mergedEjb"
    nsURI="mergedEjb" nsPrefix="mergedEjb">
  <eClassifiers xsi:type="ecore:EClass" name="ProjectContainer">
    <eStructuralFeatures xsi:type="ecore:EReference" name="ejbProject" lowerBound=
    "1"
        eType="#//EJBProject" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="EJBProject">
    <eStructuralFeatures xsi:type="ecore:EReference" name="interfaces" upperBound=
    "-1"
        eType="#//EJBInterface" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="clients" upperBound="-1
    "
        eType="#//Client" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="sessionBeans"
    upperBound="-1"
        eType="#//SessionBean" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="messageDrivenBeans"
    upperBound="-1"
        eType="#//MessageDrivenBean" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="entities" upperBound="
    -1"
        eType="#//Entity" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Client">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="class" lowerBound="1"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="references" upperBound=
    "-1"
```

```
        eType="#//EJBReference" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="EJBReference" abstract="true">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="referenceJNDIName"
    lowerBound="1"
        eType="ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" eType="
    ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="type" eType="
    ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ejbLink" eType="
    ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="EJBLocalReference" eSuperTypes="#//
  EJBReference">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="localInterface" eType="
    ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="localHomeInterface"
    eType="ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="EJBRemoteReference" eSuperTypes="#//
  EJBReference">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="remoteInterface" eType=
    "ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="remoteHomeInterface"
    eType="ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="EJBInterface">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="fullyQualifiedClass"
    lowerBound="1"
        eType="ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="EJBHomeInterface" abstract="true">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="fullyQualifiedClass"
    lowerBound="1"
        eType="ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="LocalHomeInterface" eSuperTypes="#//
  EJBHomeInterface"/>
  <eClassifiers xsi:type="ecore:EClass" name="RemoteHomeInterface" eSuperTypes="
  #//EJBHomeInterface"/>
  <eClassifiers xsi:type="ecore:EClass" name="RemoteInterface" eSuperTypes="#//
  EJBInterface"/>
  <eClassifiers xsi:type="ecore:EClass" name="LocalInterface" eSuperTypes="#//
  EJBInterface"/>
  <eClassifiers xsi:type="ecore:EClass" name="Bean" abstract="true">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ejbName" lowerBound="1"
        eType="ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="ejbClass" lowerBound="1
    "
        eType="ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="mappedName" eType="
    ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="description" eType="
    ecore:EDataType_http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="ejbReferences"
    upperBound="-1"
        eType="#//EJBReference" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="entitiesUsed"
    upperBound="-1"
        eType="#//Entity"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="SessionBean" abstract="true"
  eSuperTypes="#//Bean">
    <eStructuralFeatures xsi:type="ecore:EReference" name="localInterface" eType="
    #//LocalInterface"
        containment="true"/>
```

```
    <eStructuralFeatures xsi:type="ecore:EReference" name="localHomeInterface"
    eType="#//EJBHomeInterface"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="remoteInterface" eType=
    "#//RemoteInterface"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="remoteHomeInterface"
    eType="#//RemoteHomeInterface"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="localBusinessInterface"
        upperBound="-1" eType="#//LocalInterface" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="remoteBusinessInterface
    "
        upperBound="-1" eType="#//RemoteInterface" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="publicMethods"
    upperBound="-1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="
    implementsSessionSynchronization"
        lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/
        Ecore#//EBoolean"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="StatelessSessionBean" eSuperTypes="
  #//SessionBean"/>
  <eClassifiers xsi:type="ecore:EClass" name="StatefulSessionBean" eSuperTypes="
  #//SessionBean"/>
  <eClassifiers xsi:type="ecore:EClass" name="Entity" eSuperTypes="#//Bean">
    <eStructuralFeatures xsi:type="ecore:EReference" name="localInterface" eType="
    #//LocalInterface"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="localHomeInterface"
    eType="#//EJBHomeInterface"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="remoteInterface" eType=
    "#//RemoteInterface"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="remoteHomeInterface"
    eType="#//RemoteHomeInterface"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="publicMethods"
    upperBound="-1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="MessageDrivenBean" eSuperTypes="#//
  Bean"/>
</ecore:EPackage>
```

# Appendix C

# Extended J2EE library for Semmle .QL

The following is an extended version of the J2EE library [37] that comes from the current version of .QL. It is extended for the purpose of identifying EJB 3.0 beans appropriately.

```
/** --- J2EE ---

   A library of classes to represent J2EE bean types.
*/

import Type
import JavaProject

/** An entity bean */
class EntityBean extends Class {
   EntityBean() {
      exists(Interface i, Annotation a | (i.hasQualifiedName("javax.ejb","
      EntityBean") and this.hasSupertype+(i))
                        or (a.getType().hasQualifiedName("javax.persistence","
                        Entity") and a = this.getAnAnnotation()))
   }

  /** return the number of public member methods declared in this type */
  int getNumberOfNonDefaultPublicMethods() { result = count(Method m |
  declaresMember(this,m) and m.isPublic()
                and (not m.getName().matches("ejbActivate"))
                and (not m.getName().matches("ejbPassivate"))
                and (not m.getName().matches("ejbRemove"))
                and (not m.getName().matches("ejbLoad"))
                and (not m.getName().matches("ejbStore"))
                and (not m.getName().matches("setEntityContext"))
                and (not m.getName().matches("unsetEntityContext")))}
}

/** An enterprise bean */
class EnterpriseBean extends RefType {
   EnterpriseBean() {
      exists(Interface i, Annotation a | i.hasQualifiedName("javax.ejb","
      EnterpriseBean") and this.hasSupertype+(i)
                              or (a.getType().hasQualifiedName("javax.ejb","
                              MessageDriven") and a = this.getAnAnnotation())
                              or (a.getType().hasQualifiedName("javax.
                              persistence","Entity") and a = this.
                              getAnAnnotation())
```

```
                                or (a.getType().hasQualifiedName("javax.ejb","
                                Stateful") and a = this.getAnAnnotation())
                                or (a.getType().hasQualifiedName("javax.ejb","
                                Stateless") and a = this.getAnAnnotation()))
   }
}

/** A local EJB home interface */
class LocalEJBHomeInterface extends Interface {
   LocalEJBHomeInterface() {
      exists(Interface i | i.hasQualifiedName("javax.ejb","EJBLocalHome") and this
      .hasSupertype+(i))
   }
}

/** A remote EJB home interface */
class RemoteEJBHomeInterface extends Interface {
   RemoteEJBHomeInterface() {
      exists(Interface i | i.hasQualifiedName("javax.ejb","EJBHome") and this.
      hasSupertype+(i))
   }
}

/** A local 2.1 EJB interface */
class LocalEJBInterface extends Interface {
   LocalEJBInterface() {
      exists(Interface i | i.hasQualifiedName("javax.ejb","EJBLocalObject") and
      this.hasSupertype+(i))
   }
}

/** A remote 2.1 EJB interface */
class RemoteEJBInterface extends Interface {
   RemoteEJBInterface() {
      exists(Interface i | i.hasQualifiedName("javax.ejb","EJBObject") and this.
      hasSupertype+(i))
   }
}

/** A remote 3.0 EJB interface */
class RemoteEJBBusinessInterface extends Interface {
   RemoteEJBBusinessInterface() {
      exists(Annotation a | a.getType().hasQualifiedName("javax.ejb","Remote") and
       a = this.getAnAnnotation())
   }
}

/** A local 3.0 EJB interface */
class LocalEJBBusinessInterface extends Interface {
   LocalEJBBusinessInterface() {
      exists(Annotation a | a.getType().hasQualifiedName("javax.ejb","Local") and
      a = this.getAnAnnotation())
   }
}

/** A message bean */
class MessageBean extends Class {
   MessageBean() {
      exists(Interface i,Annotation a | (i.hasQualifiedName("javax.ejb","
      MessageDrivenBean") and this.hasSupertype+(i))
                        or (a.getType().hasQualifiedName("javax.ejb","
                        MessageDriven") and a = this.getAnAnnotation()))
   }
}

/** A session bean */
class SessionBean extends Class {
   SessionBean() {
```

```
      exists(Interface i,Annotation a | (i.hasQualifiedName("javax.ejb","
      SessionBean") and this.hasSupertype+(i))
                                              or (a.getType().hasQualifiedName("
                                              javax.ejb","Stateful") and a =
                                              this.getAnAnnotation())
                                              or (a.getType().hasQualifiedName("
                                              javax.ejb","Stateless") and a =
                                              this.getAnAnnotation()))
    }

  /** return the number of public member methods declared in this type */
  int getNumberOfNonDefaultPublicMethods() { result = count(Method m |
  declaresMember(this,m) and m.isPublic()
               and (not m.getName().matches("ejbActivate"))
               and (not m.getName().matches("ejbPassivate"))
               and (not m.getName().matches("ejbRemove"))
               and (not m.getName().matches("setSessionContext")))}

        /* get the number of Entities Referenced by this SessionBean */
        int getNumberOfEntitiesReference(){ result = count (EntityBean e | depends
        (this,e))}

        /** return an entity used by this SessionBean */
  EntityBean getEntityUsed() { depends(this,result) }

}

/** An EJB 3.0 stateful session bean */
class StatefulSessionBean extends Class {
   StatefulSessionBean() {
                        exists(Annotation a | a.getType().hasQualifiedName("javax.
                        ejb","Stateful") and a = this.getAnAnnotation())
   }
}

/** An EJB 3.0 stateless session bean */
class StatelessSessionBean extends Class {
   StatelessSessionBean() {
                        exists(Annotation a | a.getType().hasQualifiedName("javax.
                        ejb","Stateless") and a = this.getAnAnnotation())
   }
}

/* An Java EJB Project */
class JavaEJBProject extends JavaProject{

int getNumberOfSessionBeans () { result = count(SessionBean sb| exists(this.
getASourcePackage()) and isInPackage(sb, this.getASourcePackage()))}
int getNumberOfEntities () { result = count(EntityBean eb| exists(this.
getASourcePackage()) and isInPackage(eb, this.getASourcePackage()))}

}
```

# References


[1] Michal Antkiewicz. *Framework-Specific Modeling Languages*. PhD thesis, University of Waterloo, 2008. 2, 5, 7, 21, 38, 40, 59, 64, 66

[2] Michal Antkiewicz, Thiago Tonelli Bartolomei, and Krzysztof Czarnecki. Automatic extraction of framework-specific models from framework-based application code. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 214–223, New York, NY, USA, 2007. ACM. 6, 38

[3] Michal Antkiewicz, Krzysztof Czarnecki, and Matthew Stephan. Engineering of framework-specific modeling languages. *Transactions of Software Engineering,Special Issue on Language Engineering*, 2009. Accepted for publication. Available at `http://swen.uwaterloo.ca/~mantkiew/2009-antkiewicz-engineering-fsmls.pdf`. 15, 16, 22, 38, 39, 66

[4] Eric Armstrong, Jennifer Ball, Stephanie Bodoff, Debbie Carson, Ian Evans, Dale Green, Kim Haase, and Eric Jendrook. The java ee 1.4 tutorial: Getting started with enterprise beans, December 2005. `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/EJB.html`. 23

[5] Roland Barcia and Jeffrey Sampson. Building ejb 3.0 applications with websphere application server, 2007. `http://www.ibm.com/developerworks/websphere/techjournal/0712_barcia/0712_barcia.html`,last accessed Saturday March 28 2009. 68

[6] S. Boroday, A. Petrenko, J. Singh, and H. Hallal. Dynamic analysis of java applications for multithreaded antipatterns. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, 2005. 11

[7] Daniel Bray. Struts: avoiding some common problems, 2003. `http://www.braindelay.com/danielbray/struts-top-tips/stt.html`,last accessed Saturday March 28 2009. 66

[8] William Brown, Hays McCormick, and Scott W. Thomas. *Anti-Patterns and Patterns in Software Configuration Management*. John Wiley and Sons, 1999. 11

[9] William Brown, Hays McCormick, and Scott W. Thomas. *AntiPatterns in Project Management*. John Wiley and Sons, 2000. 11, 52

[10] William J. Brown, Raphael C. Malveau, Skip McCormick, and Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. Wiley Publishing Inc, 1st edition, March 1998. 1, 11

[11] William Crawford and Jonathan Kaplan. *J2EE Design Patterns*. O'Reilly Publishing, 2003. 12

[12] Krysztof Czarnecki and Ulrich Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, June 2000. 5

[13] Krysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. In *Software Process Improvement and Practice, special issue of best papers from SPLC04*, volume 10, pages 7 – 29, 2005. 7

[14] Oege de Moor, Damien Sereni, Mathieu Verbaere, Elnar Hajiyev, Pavel Avgustinov, Torbj orn Ekman, Neil Ongkingco, and Julian Tibble. .ql: Object-oriented queries made easy. In *Generative and Transformational Techniques in Software 2007*, page 78 133. Springer-Verlag Berlin Heidelberg, 2007. 13, 14, 47, 54, 58

[15] Linda DeMichiel and Michael Keith. Enterprise JavaBeans deployment descriptor schema 3.0, May 2006. `http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd`. 9, 16, 21

[16] Linda DeMichiel and Michael Keith. *JSR 220: Enterprise JavaBeans*. Sun Microsystems Inc, 3.0 edition, May 2006. http://java.sun.com/products/ejb/docs.html. 1, 7, 9, 10, 16, 27, 28, 31, 43, 45

[17] Linda G. DeMichiel. *Enterprise JavaBeansTM Specification*. Sun Microsystems Inc, 2.1 edition, November 2003. http://java.sun.com/products/ejb/docs.html. 8, 23, 24, 27, 42, 46, 55

[18] Frank Devos. *Patterns and Anti-Patterns in Object-Oriented Analysis*. PhD thesis, Katholieke Universiteit Leuven, 2004. 11

[19] Bill Dudney, Stephen Asbury, Joseph K. Krozak, and Kevin Wittkopf. *J2EE AntiPatterns*. Wiley Publishing Inc, 2003. 12, 26, 49, 50, 51, 52, 67

[20] IBM Centers for Advanced Studies. Cas toronto. https://www-927.ibm.com/ibm/cas/toronto/index.shtml. 2

[21] H Gallaire and J Minker. *Logic and Databases*. Plenum Press, New York, 1978. 13

[22] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995. 52

[23] H. H. Hallal, E. Alikacem, W. P. Tunney, S. Boroday, and A. Petrenko. Antipattern-based detection of deficiencies in java multithreaded software. *Quality Software, International Conference on*, 0:258–267, 2004. 11

[24] Jack D Herrington. Ajax and xml: Five ajax anti-patterns, March 2007. http://www.ibm.com/developerworks/xml/library/x-ajaxxml3/index.html. 12

[25] Richard C. Holt. Structural manipulations of software architecture using tarski relational algebra. In *WCRE '98: Proceedings of the Working Conference on Reverse Engineering (WCRE'98)*, page 210, Washington, DC, USA, 1998. IEEE Computer Society. 69

[26] Daqing Hou and H. James Hoover. Using scl to specify and check design intent in source code. *IEEE Transactions on Software Engineering*, 32(6):404–423, 2006. 69

[27] David Hovemeyer and William Pugh. Finding bugs is easy. In *ACM SIGPLAN Notices*, pages 132–136. ACM Press, 2004. 69

[28] Alex Iskold, Daniel Kogan, and Goran Begic. Structural analysis for java, 2004. http://www.alphaworks.ibm.com/tech/sa4j,last accessed Saturday March 28 2009. 68

[29] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc., 1999. 15

[30] Eric Jendrock, Jennifer Ball, Debbie Carson, Ian Evans, Scott Fordin, and Kim Haase. The java ee 5 tutorial, September 2007. http://java.sun.com/javaee/5/docs/tutorial/doc/. 7, 9, 16, 26, 27

[31] Bill Karwin. Sql antipatterns. MySql Conference and Expo 2008, April 2008. http://en.oreilly.com/mysql2008/public/schedule/detail/1639. 11, 66

[32] Miroslav Kis. Information security antipatterns in software requirements engineering. In *9th Conference of Pattern Languages of Programs (PloP)*, 2002. 11

[33] Jaroslav Kral and Michal Zemlicka. The most important service-oriented antipatterns. In *ICSEA '07: Proceedings of the International Conference on Software Engineering Advances*, page 29, Washington, DC, USA, 2007. IEEE Computer Society. 11, 66

[34] Yoshihito Kuranuki and Kenji Hiranabe. Antipractices: Antipatterns for xp practices. In *Proceedings of the Agile Development Conference (ADC04)*, 2004. 11

[35] Anthony Lauder and Stuart Kent. Legacy system anti-patterns and a pattern-oriented migration response. In *Henderson, P. (ed.) Systems Engineering for Business Process Change*. Springer-Verlag, 2000. 11

[36] J. W. Lloyd. Practical Advantages of Declarative Programming. In *Joint Conference on Declarative Programming*, 1994. 57

[37] Semmle Ltd. Standard libraries: J2ee, 2009. 47, 55, 97

[38] Dustin Marx. Common struts errors and causes. http://www.geocities.com/dustinmarx/SW/struts/errors.html,last accessed Saturday March 28 2009. 66

[39] Sun Microsystems. Java ee technologies at a glance, 2008. http://java.sun.com/javaee/technologies/. 7, 9, 12, 24

[40] Trevor Parsons. A framework for detecting performance design and deployment antipatterns in component based enterprise systems. In *DSM '05: Proceedings of the 2nd international doctoral symposium on Middleware*, pages 1–5, New York, NY, USA, 2005. ACM. 12, 70

[41] G.R. Prakash. Top 10 mistakes in eclipse plugin development, 2009. http://eclipse.dzone.com/articles/top-10-mistakes-eclipse-plugin,last accessed Saturday March 28 2009. 66

[42] Peter Purich, Debu Panda, and Raghu Kodali. What are enterprise javabeans?, 2007. http://download-uk.oracle.com/docs/cd/B31017_01/web.1013/b28221/undejbs001.htm. 9

[43] David Reilly. What causes applets to fail, 2006. http://www.javacoffeebreak.com/articles/greyboxapplets/index.html,last accessed Saturday March 28 2009. 66

[44] Bill Shannon. Enterprise JavaBeans deployment descriptor schema 2.1, November 2004. http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd. 23, 24

[45] Steven She. Feature model mining. Master's thesis, University of Waterloo, 2008. 65, 66

[46] Stefan Slinger. Code smell detection in eclipse. Master's thesis, Delft University of Technology, 2005. 65, 69

[47] Connie U. Smith. Software performance antipatterns. In *Proceedings of the Second International Workshop on Software and Performance (WOSP2000*, pages 127–136. Technology, Inc, 2000. 11

[48] Connie U. Smith. Software performance antipatterns: Common performance problems and their solutions. In *In Int. CMG Conference*, pages 797–806, 2001. 11

[49] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF Eclipse Modeling Framework*. Addison Wesley, second edition, 2009. 7, 43

[50] Matthew Stephan and Michal Antkiewicz. Ecore.fmp: A tool for instantiating class models as feature models. Technical Report 2008-08, Electrical and Computer Engineering, University of Waterloo, 2008. 7, 18, 20, 24, 43