Design of the OSS IP Billing API Reference Implementation (JSR130)

IPBilling-API-RI-DG.1.1

OSS through Java™ Initiative

James Jouett

Executive Summary

The OSS IP Billing API specifies both J2EE/EJB and based interfaces for managing usage data record collection activities. The interfaces and mechanisms between billing components are supported via standard J2EE communication mechanisms.

This document describes how the Reference Implementation was designed to implement the OSS IP Billing API.

Table of Contents

Preface

Objectives

The purpose of this document is to describe the design of the OSS IP Billing Reference Implementation.

Audience

The target audience of this document is as follows:

- Developers who seek information about how the API can be implemented
- Developers who want to make use of this API and extend its implementations

Revision History

Date	Version	Author	State	Comments
08/30/2002	1.0	James Jouett	Final	Final Version
11/01/2003	1.1	James Jouett	Final	Updates to include XML interface and editorial changes

1 Introduction and concepts

This document describes the design of the OSS IP Billing API Reference Implementation.

The Reference Implementation (RI) can be used either as a proof-of-concept for the OSS IP Billing API specification or the components in the RI can be directly used as building blocks for a custom implementation. The RI serves to fulfill multiple needs:

- It acts as a tutorial, because the Reference Implementation might offer solutions to problems that arise during implementation.

- It can be a toolbox, because some parts can be found to be reusable during a custom implementation of the API.

- It can serve as a prototype, since a developer can add new usage record types to the Reference Implementation. Using the RI as a starting point allows a developer to rapidly create a prototype that supports a custom implementation.

  The use of the prototype is limited in a production environment though, as it probably will not scale perfectly and is not optimized for speed, memory footprint or security.

This document shows how the Reference Implementation was designed, any limitations and problems, and how it is to be used.

The RI was developed using J2EE 1.3 as the system platform and it was tested using the following application servers:

- Weblogic Server 6.1 [BEA01]
- JBoss Server 3.2.1 [JBOSS01]

IMPORTANT: Before installing the OSS IP Billing Reference Implementation, please read and follow the license agreement [NEC01].

2 General

The Reference Implementation has been tested successfully with Weblogic Server 6.1 (using Cloudscape as the system database) and JBoss 3.2.1 (using Hypersonic as the system database). However, no reasons are known why the Reference Implementation would not work on other application servers or database systems. Future versions of the RI will be compatible with the J2EE Reference Implementation as well as with other application servers. Also, the initial version of the RI does not contain a built-in scheduling manager process or back-end mediation process to simulate the actual firing of schedules and collection of usage data; adding this advanced capability will be addressed in future versions of the RI.

The RI provides both a Java Value Type (JVT) access interface implementation and an XML Value Type Request/Response (XVT) access interface implementation. The following sections provide information on the organization of the RI for these implementations.

2.1 Java Value Type Implementation

The JVT RI provides a Stateless Session Bean *JVTProducerSessionBean* that serves as the Java access interface to API clients. The *JVTProducerSessionBean* supports the interaction with the *ProducerEntityHome*, which manages the *ProducerEntity* Entity Beans. The JVT RI uses container-managed persistence (CMP) to support the storage and retrieval of the Entity Bean representation of a Producer object. Figure 1 shows how these components interact in the JVT RI.

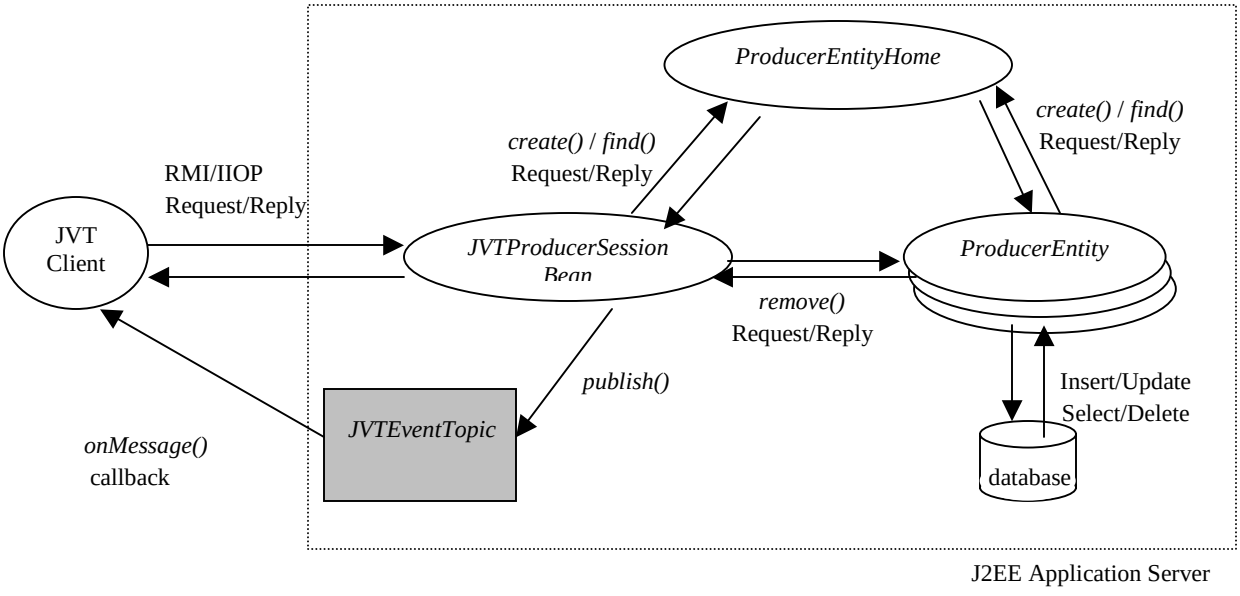

Figure 1 JVT RI Component Interaction

The following sections provide information on the organization of the Java classes in the JVT RI.

2.1.1 Package Names

Naming of the RI packages follows the structure of the JSR130 specification [IPB01], with the package names for the RI taking the form *com.nec.oss.ipb.**SPEC_PKG_NAME**.ri*, where ***SPEC_PKG_NAME*** is the name of the corresponding package name in the JSR130 specification. For example, the package *javax.oss.ipb.producer* in the JSR130 specification is implemented as the package *com.nec.oss.ipb.producer.ri* in the RI.

2.1.2 Class Names

Naming of the RI classes follows the convention of ***SPEC_IF_NAME****Impl*, where ***SPEC_IF_NAME*** is the name of the corresponding interface name in the JSR130 specification. For example, the interface *ProducerValue* in the JSR130 specification is implemented as the class *ProducerValueImpl* in the RI. The only exception to this convention is the naming of some of the EJB classes, which are described in the following section.

2.2 XML Value Type Implementation

The XVT RI provides a Message Driven Bean that provides a Message Queue to service requests from API clients. The request messages sent to this queue should follow the format of the *XXXrequest* element structures in the XML Schema of the JSR130 specification [IPB01]. Each *XXXrequest* message has a matching *XXXresponse* message that informs the client of the result of the request. Following the OSS Design Guidelines [COM01], these request/response pairs correspond to the methods provided by the *JVTProducerSession*. For the XVT RI, the implementation is realized by using the Message Driven Bean as a wrapper around the JVT RI. Figure 2 shows how these components interact in the XVT RI.

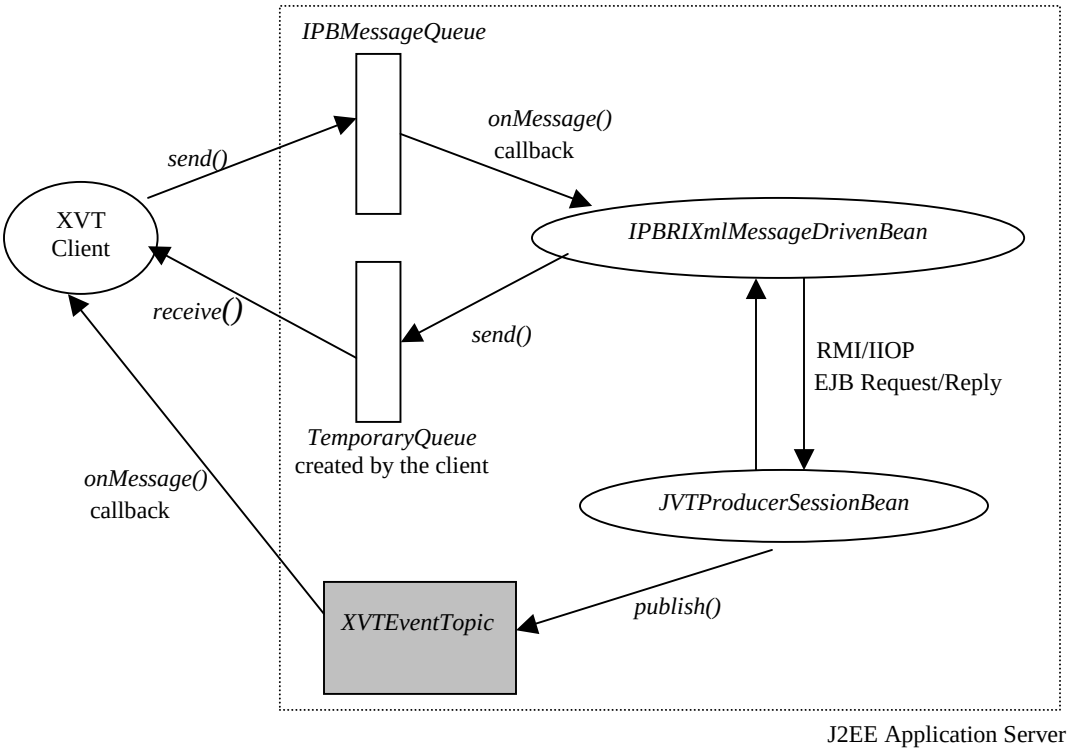

Figure 2 XVT RI Component Interaction

3 Component Overview

The following subsections provide details on the components that comprise the JVT and XVT Reference Implementations.

3.1 JVT Component Overview

The JVT Reference Implementation contains the following high-level EJB components, which are described in more detail throughout the document:

- *JVTProducerSessionBean*: The *JVTProducerSessionBean* Stateless Session Bean is the main component of the JVT Reference Implementation. It implements the *JVTProducerSession* Java Value Type (JVT) interface defined in the IP Billing specification.
- *ProducerEntity*: This entity bean controls the persistence state of all Producers in the system.
- *ProducerValueIteratorImpl*: This Stateful Session Bean provides an iterator capability for retrieving the results of a query of *ProducerValue* objects.
- *ProducerKeyResultIteratorImpl*: This Stateful Session Bean provides an iterator capability for retrieving the results of an operation that provides *ProducerKeyResult* objects.
- *UsageRecValueIteratorImpl*: This Stateful Session Bean provides an iterator capability for retrieving the results of a query of UsageRecValue objects.
- *ReportInfoIteratorImpl*: This Stateful Session Bean provides an iterator capability for retrieving the results of Usage Data report information.
- *TransferStatusValueIteratorImpl*: This Stateful Session Bean provides an iterator capability for retrieving the results of a query of TransferStatusValue objects
- *EventHelper*: This class provides all of the functionality needed to send JMS events for Producer state change and Usage Data availability notification.

Figure 3 below shows the relationship between these components.

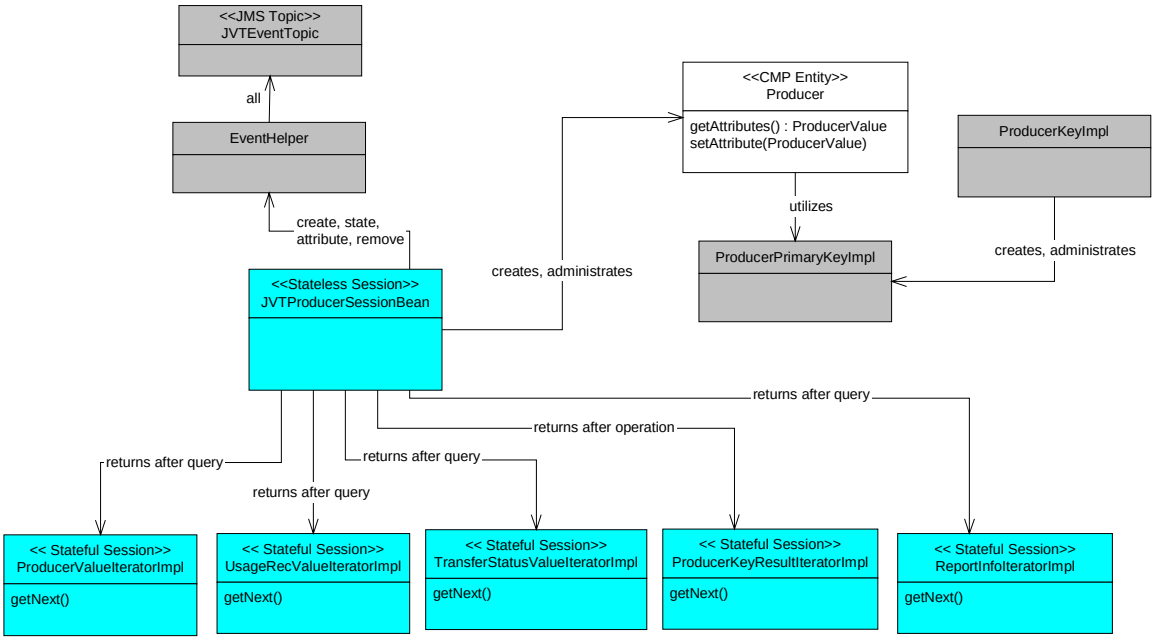

Figure 3 JVT Reference Implementation Component Overview

3.1.1 JVTProducerSessionBean

The *JVTProducerSessionBean* class is the focal point of the JVT implementation of the IP Billing API. This Stateless Session Bean provides the implementation of the *JVTProducerSession* interface methods defined in the JSR130 specification. The following restrictions and limitations exist with the initial version of the RI:

- For the current Reference Implementation, the following methods of the *JVTProducerSession* are not supported:
  - *queryTransferStatusValues()*
  - *getUsageRecordsByIterator()*

- The following query methods of the *JVTProducerSession* will return a fixed set of data on each invocation:
  - *queryUsageRecords()*
  - *getUsageDataInformation()*
  - *queryUsageDataInformation()*

Since these methods involve optional features or dynamic capabilities that require a functional back-end system to simulate a mediation application, the initial release of the RI does not support these methods. Subsequent versions of the Reference Implementation will provide support for these advanced and optional capabilities. The main goal of this initial version of the RI is to provide an example implementation of the JSR130

specification that will serve as a significant starting point and generate feedback within the technical community.

3.1.2 JMS Notification

A *JVTEventTopic* JMS topic is defined for transmitting notifications such as *MediationCapabilityChangeEvent* and activity status change events (*ActivityCreationEvent*, *ActivitySuspendEvent*, etc). The *EventHelper* class provides the functionality to send these JMS events.

3.1.3 ProducerEntity

The *Producer* Entity Bean is the persistent version of the corresponding *ProducerValue* Java Value Type class. This class is implemented as EJB 2.0 Container Managed Persistence (CMP), meaning that the application server performs the reading and writing of the data to the database. For the RI, the default database provided with the application server is used to store the Producer entities.

This Entity Bean provides methods to set and retrieve the attributes from a *Producer* JVT object and to create a JVT object with specified values populated. Note that the primary key for the *Producer* Entity Bean is represented by the *ProducerPrimaryKeyImpl* class, and the *ProducerPrimaryKeyImpl* is contained within the *ProducerKeyImpl* class. The *ejbCreate()* method of the *Producer* Entity Bean has an argument of the base class *ProducerKey*. This approach allows a developer to subclass *ProducerKeyImpl* and *ProducerPrimaryKeyImpl* to provide the specifics for uniquely identifying their Producer, instead of relying on only a single *String* to serve as the key. Further information on these JVT classes is found in Section 3.2.

3.1.4 ProducerValueIteratorImpl

To support Producer queries without consuming system resources due to transferring large result sets, the *ProducerValueIteratorImpl* encapsulates a *ProducerValueStatefulIterator* Stateful Session Bean and provides the implementation mechanism to support retrieval of client-defined data set sizes. *JVTProducerSessionBean* methods such as *getProducersByTemplate()* return a *ProducerValueIteratorImpl* object to the client to allow batch-type retrieval of the *Producer* Java Value Type objects.

3.1.5 UsageRecValueIteratorImpl

The *UsageRecValueIteratorImpl* encapsulates a *UsageRecValueStatefulIterator* Stateful Session Bean and provides the client with the ability to step through the results of a usage data query and control the number of values returned from *JVTProducerSessionBean* methods such as *queryUsageRecords()*. Note that as mentioned in Section 3.1.1, the methods for querying usage records are implemented to return a fixed set of data for each invocation.

3.1.6 ReportInfoIteratorImpl

The *ReportInfoIteratorImpl* encapsulates a *ReportInfoStatefulIterator* Stateful Session Bean and provides the client with the ability to step through the results of a usage report information query and control the number of values returned from *JVTProducerSessionBean* methods such as *getUsageDataInformation()*. Note that as mentioned in Section 3.1.1, the methods for querying usage report information are implemented to return a fixed set of data for each invocation.

3.1.7 TransferStatusValueIteratorImpl

Similar to the *UsageRecValueIteratorImp* and *ReportInfoIteratorImpl,* the *TransferStatusValueIteratorImpl* encapsulates a *TransferStatusValueStatefulIterator* Stateful Session Beans and provides the client with the ability to step through the results of a query of transfer status records and control the number of values returned. As mentioned in Section 3.1.1, the methods for querying transfer status values aren't supported in this initial version of the RI. However, this iterator class is fully defined in the RI, and it is deployed with the other EJBs when the RI is installed.

3.1.8 ProducerKeyResultIteratorImpl

The *ProducerKeyResultIteratorImpl* encapsulates a *ProducerKeyResultStatefulIterator* Stateful Session Bean and provides the same iterator capability as the previously mentioned iterator beans, but this class is used to traverse the results of a bulk operation. *JVTProducerSessionBean* methods such as *trySuspendByKeys()* return a *ProduceKeyResultIteratorImpl* object to the client to allow batch-type retrieval of the bulk operation results.

3.2 Java Value Types

The Java Value Types are objects that are exchanged between the *JVTProducerSessionBean* and a client to represent the managed entities. For the RI, the *ProducerValueImpl* is used to represent the *Producer* Entity to the client. Figure 4 below shows the definition of the *ProducerValueImpl* class and its relationship to other classes in the RI.

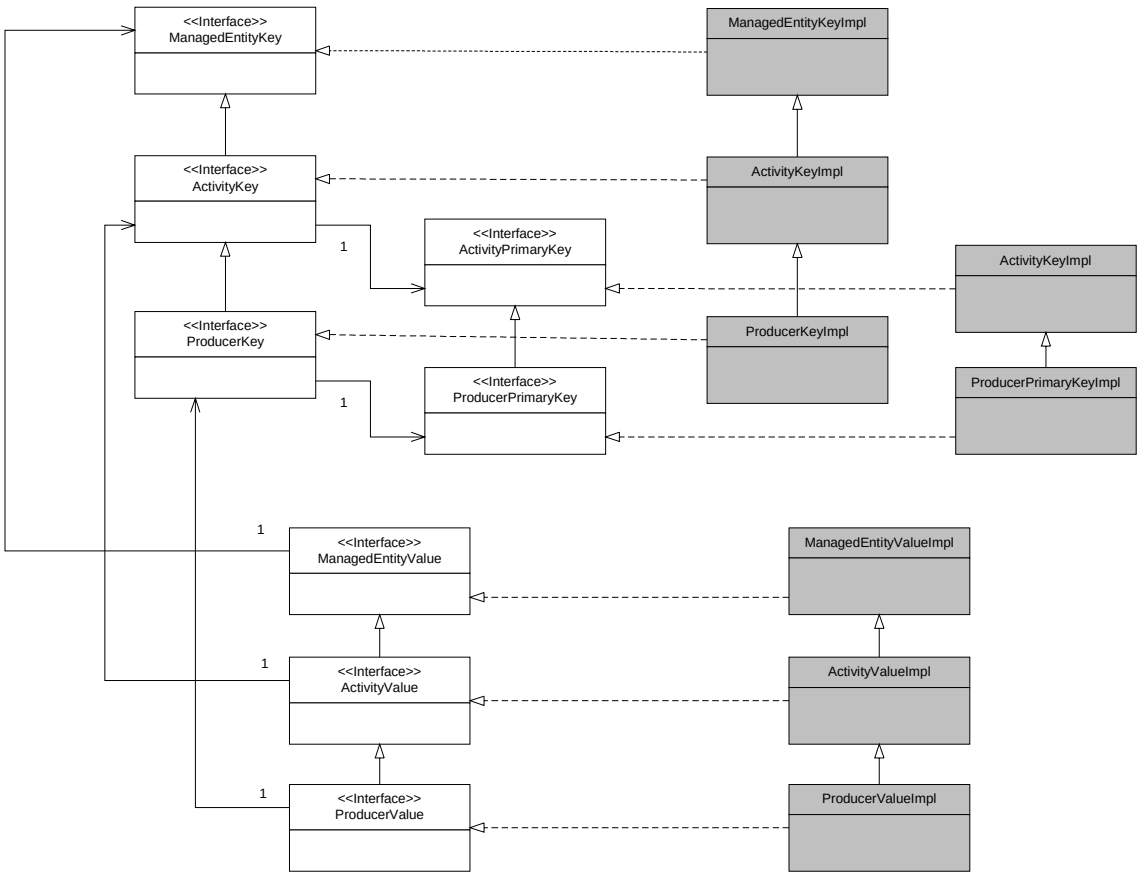

Figure 4 Java Value Types

The RI JVT classes utilize the common *ManagedEntity* classes to control the state and changes of the populated attributes [COM01]. As was mentioned in Section 3.1.3, an approach was taken in the JSR130 specification to define a *ProducerPrimaryKey* that defines the primary key fields for the *Producer* object, allowing more complex primary key definition than a single *String* object. For the RI, a *ProducerPrimaryKeyImpl* is defined that has attributes of *activityId* and *producerType* that act as the primary key for the *Producer* entity. A developer wishing to extend the RI would just need to modify the implementation classes shown (or define custom classes that extend the implementation classes) and provide the specifics that define the characteristics of a Producer for their system.

3.3 XVT Component Overview

As shown in Figure 2, the XVT RI is implemented as a wrapper around the JVT RI. Therefore, the number of components that are specific to the XVT RI is quite small. The OSS/J XML Tooling API [XML01] provides the XVT RI with the base Message Driven Bean functionality as well as the utility classes to support conversion between Java objects and XML messages. Figure 5 shows the Java components that are specific of the XVT RI and their relationship with the OSS/J

XML Tooling classes. All of the classes that are specific to the XVT RI are located in the package *com.nec.oss.ipb.producer.ri.xml*

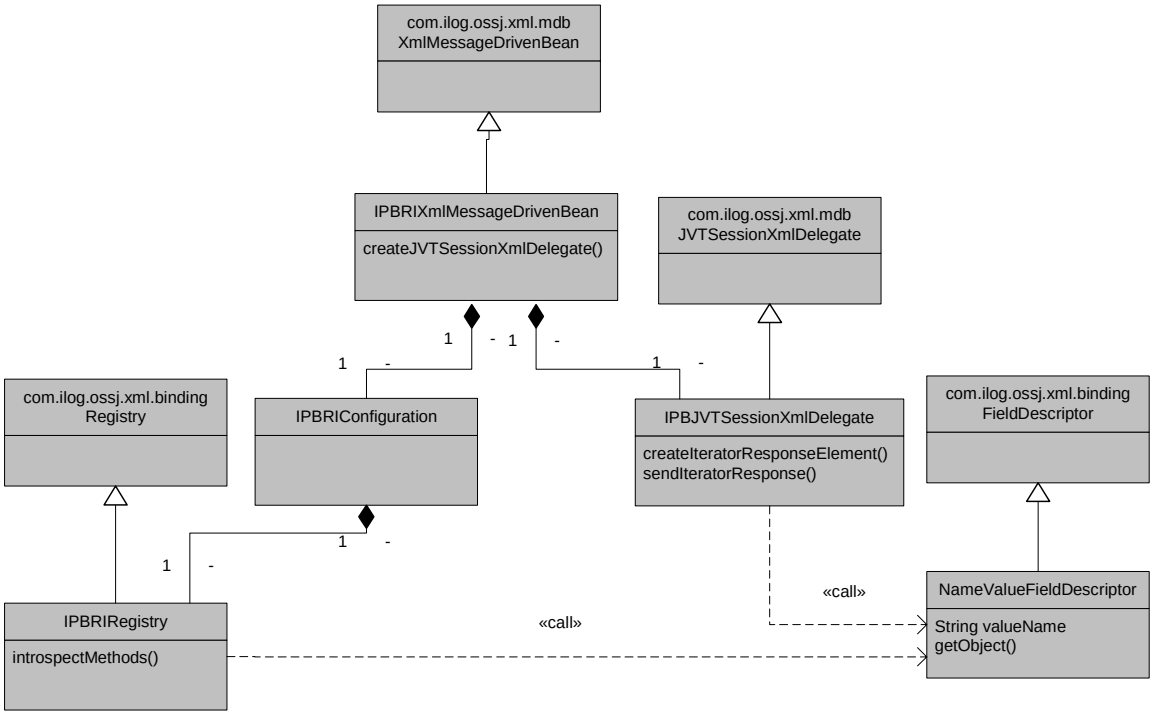

Figure 5 XVT Reference Implementation Component Overview

3.3.1 IPBRIXmlMessageDrivenBean

The *IPBRIXmlMessageDrivenBean* class extends the functionality provided by the OSS/J XML Tooling class *XmlMessageDrivenBean* by overriding the method *createJVTSessionXmlDelegate()*. The overridden method sets the *JVTSessionXmlDelegate* object used by the class to an instance of *IPBJVTSessionXmlDelegate*, which allows *IPBJVTSessionXmlDelegate* to be used in handling request and responses to the *JVTProducerSession* Stateless Session Bean.

3.3.2 IPBRIJVTSessionXmlDelegate

The *IPBRIJVTSessionXmlDelegate* class extends the functionality provided by the OSS/J XML Tooling class *JVTSessionXmlDelegate* by overriding the methods *createIteratorResponseElement()* and *sendIteratorResponse()*. The overridden method are needed to handle the special case of servicing requests that return *UsageRecValueIterator*. As described in the OSS IP Billing API Specification [IPB01], the XML schema elements representing *UsageRecValueIterator* differ from standard OSS/J schema representations of iterators since the response

element must contain both the *UsageRecValue* data array as well as an instance of *RecordDescriptor*, which describes the data contained in the *UsageRecValue* array. Since this iterator response message differs from the standard OSS/J XML response message format, the overridden methods provided by *IPBRIJVTSessionXmlDelegate* create the appropriate iterator response message.

3.3.3 NameValueFieldDescriptor

The *NameValueFieldDescriptor* class extends the functionality provided by the OSS/J XML Tooling class *FieldDescriptor* by overriding the method *getObject()* and adding an attribute to represent a name for the value. This class is needed to describe the fields of the *UsageRecValue* class in the OSS IP Billing specification. As described in the OSS IP Billing API Specification [IPB01], the *UsageRecValue* class differs from other OSS/J classes in that it uses name/value pairs to specify the attributes of the class instead of using *getXXX()*/*setXXX()* accessor methods. Since the name of the attribute is an additional parameter to the accessor methods instead of being part of the accessor name, *NameValueFieldDescriptor* is used to allow *IPBRIJVTSessionXmlDelegate* and classes of the OSS/J XML Tooling to get and set attributes of the associated class.

3.3.4 IPBRIConfiguration

The *IPBRIConfiguration* class is used to initialize the *Registry* used by the OSS/J XML Tooling library to define how to convert between Java classes and XML messages. This class follows the pattern given by the *SampleConfiguration* class provided as an example in the OSS/J XML Tooling distribution.

3.3.5 IPBRIRegistry

The *IPBRIRegistry* class extends the functionality provided by the OSS/J XML Tooling class *Registry* by overriding the method *introspectMethods()*. The overridden method is used to determine whether an accessor method of an OSS/J class should be represented by a *FieldDescriptor* or a *NameValueFieldDescriptor*.

3.3.6 JMS Notification

A *XVTEventTopic* JMS topic is defined for transmitting notifications such as *MediationCapabilityChangeEvent* and activity status change events (*ActivityCreationEvent*, *ActivitySuspendEvent*, etc). The *EventHelper* class of the JVT RI provides the functionality to send these JMS events.

4 Database Schema

The database schema for the initial version of the RI defines a single table that maintains the *Producer* entities.

4.1 ProducerEntity Table

The **ProducerEntity** table stores all *Producer* entities in the system.

Column Name	Type	PK
ACTIVITYID	VARCHAR(255) NOT NULL	Yes
ACTIVITYNAME	VARCHAR(255) NOT NULL	
PRODUCERTYPE	VARCHAR(255) NOT NULL	Yes
CONTROLSTATE	INTEGER	
EXECUTIONSTATUS	INTEGEGER	
ACTIVITYCONTROLPARAMS	LONG BIT VARYING	
ACTIVITYEXECPARAMS	LONG BIT VARYING	
ACTIVITYREPORTPARAMS	LONG BIT VARYING	
SUBSCRIPTIONPARAMS	LONG BIT VARYING	
SCHEDULE	LONG BIT VARYING	
MEDIATIONCAPABILITYVALUES	LONG BIT VARYING	

5 Adding New Data Types

After installing the RI [IPB03], it can serve as a starting point for adding custom Producer or Usage Rec definitions. The steps to add these new data types are described in the following sections.

5.1 JVT Reference Implementation

5.1.1 Custom Producer Types

Since the Producer JVT is a managed entity (it is derived from the *ManagedEntityValue* class), the following steps are required to define the Producer classes and incorporate them in the EJB framework:

1. Create an interface class that is derived from the JSR130 base interface *javax.oss.ipb.producer.ProducerValue*.
2. Create an implementation class for the custom interface class that also extends the *com.nec.oss.ipb.producer.ri.ProducerValueImpl* class. The name of that class should follow the same naming strategy as the RI, adding the text *Impl* to the interface name.
3. Add the new Producer to the list of producers supported in the *JVTProducerSessionBean* class.
4. Add the new classes to the deployment JAR file for RI, so the server will have access to these new classes.
5. Deploy the modified session bean, which will now be aware of this new Producer type.

5.1.2 Custom Usage Rec Types

The following steps are required to define the Usage Rec classes and incorporate them in the EJB framework:

1. Create an interface class that is derived from the JSR130 base interface *javax.oss.ipb.producer.UsageRecValue*.
2. Create an implementation class for the custom interface class that also extends the *com.nec.oss.ipb.producer.ri.UsageRecValueImpl* class. The name of that class should follow the same naming strategy as the RI, adding the text *Impl* to the interface name.
3. Add the new classes to the deployment JAR file for the RI, so the server will have access to these new classes.
4. Deploy the modified JAR file, so the session bean will now be aware of this new Usage Rec type

5.2 XVT Reference Implementation

The following sections provide details on the steps to use in defining extensions of the base XVT type definitions. All of the extension types can be

included in a single XML schema file, but the following sections provide the steps for each individual extension type.

5.2.1 Custom Producer Types

The following steps are required to define a custom Producer definition in the XVT framework:

1. Create a new XML schema file that specifies a namespace for this schema extension and includes a reference to the JSR 130 schema.
2. In this new XML schema, define a *complexType* with a stated extension of *ipb:ProducerValue*. Within this *complexType* definition, provide a *sequence* that includes the attributes that are required of the custom Producer definition. If any of these sequence elements are of a custom *complexType*, then this schema needs to include the definition of this new *complexType* as well.

5.2.2 Custom Usage Rec Types

The following steps are required to define a custom Usage Rec classes and incorporate it in the XVT framework:

1. Create a new XML schema file that specifies a namespace for this schema extension and includes a reference to the JSR 130 schema.
2. In this new XML schema, define a *complexType* with a stated extension of *ipb:UsageRecValue*. Within this *complexType* definition, provide a *sequence* that includes the attributes that are required of the custom Usage Rec definition. If any of these sequence elements are of a custom *complexType*, then this schema needs to include the definition of this new *complexType* as well.

6 Client Access Information

6.1 JVT Reference Implementation

A client that wants to use the JVT RI needs to do the following:

- The **CLASSPATH** environment variable needs to include the following JAR files:
  - JSR130 specification JAR ([IPB01])
  - Application server JAR files ([BEA01] or [JBOSS01])
- Reference the *JVTProducerSessionBean* home remote interface, which is bound to the following JNDI Name:

WLS 6.1:
System/DFW/ApplicationType/Producer/Application/1-1;1-0;NEC-IPB-RI/Comp/ProducerSessionHome
JBoss 3.2.1:
System/DFW/ApplicationType/Producer/Application/1-1;1-0;NEC-IPB-RI/Comp/ProducerSessionHome

- Reference the JMS Topic Factory, which is bound to the following JNDI name:

WLS 6.1:
System/DFW/ApplicationType/Producer/Application/1-1;1-0;NEC-IPB-RI/Comp/TopicConnectionFactory
JBoss 3.2.1:
ConnectionFactory

- Reference the JVT JMS Event Topic, which is bound to the following JNDI Name:

WLS 6.1:
System/DFW/ApplicationType/Producer/Application/1-1;1-0;NEC-IPB-RI/Comp/JVTEventTopic
JBoss 3.2.1:
topic/System/DFW/ApplicationType/Producer/Application/1-1;1-0;NEC-IPB-RI/Comp/JVTEventTopic

For an example of a client configuration, the JSR130 TCK [IPB02] provides scripts for setting the client environment variables and property files defining the JNDI names. These files would serve as a good starting point for a custom client.

6.2 XVT Reference Implementation

A client that wants to use the XVT RI needs to do the following:

- Reference the *IPBMessageQueue*, which is bound to the following JNDI Name:

WLS 6.1:
System/DFW/ApplicationType/Producer/Application/1-1;1-0;NEC-IPB-RI/Comp/IPBMessageQueue
JBoss 3.2.1:
queue/System/DFW/ApplicationType/Producer/Application/1-1;1-0;NEC-IPB-RI/Comp/IPBMessageQueue

- Reference the JMS Queue Factory, which is bound to the following JNDI name:

WLS 6.1:
System/DFW/ApplicationType/Producer/Application/1-1;1-0;NEC-IPB-RI/Comp/QueueConnectionFactory
JBoss 3.2.1:
ConnectionFactory

- Reference the JMS Topic Factory, which is bound to the following JNDI name:

WLS 6.1:
System/DFW/ApplicationType/Producer/Application/1-1;1-0;NEC-IPB-RI/Comp/TopicConnectionFactory
JBoss 3.2.1:
ConnectionFactory

- Reference the XVT JMS Event Topic, which is bound to the following JNDI Name:

WLS 6.1:
System/DFW/ApplicationType/Producer/Application/1-1;1-0;NEC-IPB-RI/Comp/XVTEventTopic
JBoss 3.2.1:
topic/System/DFW/ApplicationType/Producer/Application/1-1;1-0;NEC-IPB-RI/Comp/XVTEventTopic

Appendix A: References

[BEA01] BEA WebLogic 6.1, http://www.bea.com
[COM01] OSS Design Guidelines http://java.sun.com/products/oss/Com-arch-dg.1.1.pdf.zip
[IPB01] OSS IP Billing API Specification http://jcp.org/jsr/detail/130.jsp
[IPB02] OSS IP Billing API TCK User Guide (IPBilling-API-TCK-UG.1.1)
[IPB03] OSS IP Billing API RI Installation Guide (IPBilling-API-RI-IG.1.1)
[JBOSS01] JBoss Application Server 3.2.1, http://www.jboss.org
[NEC01] RI License Terms (LGL-HML-RI-IPB.1.1.htm)
[XML01] OSS/J XML Tooling, http://java.sun.com/products/oss/xml_tooling.html