Java™ Metadata Interface and the J2EE™ Connector Architecture

A JMI white paper by

John D. Poole

November 2002

Abstract. This paper provides an illustration of how managed meta data might be accessed from a JMI-enabled meta data resource via the J2EE™ Connector Architecture.

Introduction

This paper illustrates how access to JMI-enabled meta data resources might be accomplished. In this example, the meta data resource is treated as a JMI service embedded within a generic Enterprise Information System (EIS). The EIS is accessed via the J2EE™ Connector Architecture. Note that this is just an example of one possible strategy for connecting to a JMI service. The JMI specification itself does not prescribe how clients connect to JMI-enabled meta data services. This is generally outside the scope of a meta data management API.

In this example, an EIS ("MyEIS") uses *design patterns* prescribed by the J2EE™ Connector Architecture's Common Client Interface (CCI) to define its own ConnectionFactory and Connection interfaces. A client of MyEIS uses JNDI to obtain an instance of the ConnectionFactory interface, which is registered as a node in a local JNDI tree. The client then uses one of two variants of the ConnectionFactory::getConnection() method to obtain an instance of the Connection interface. This is illustrated in Figure 1 below:

Copyright © 2002 John D. Poole. All rights reserved.

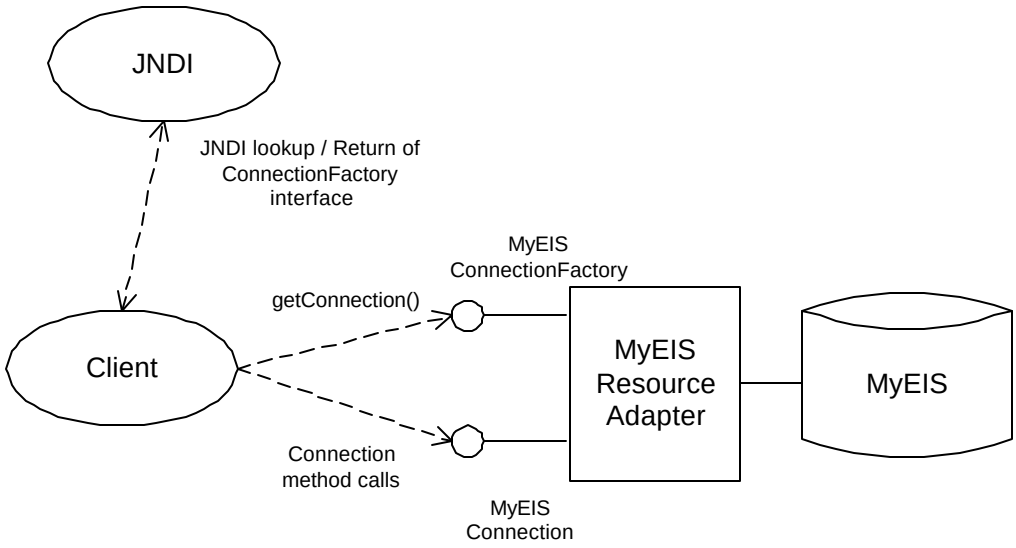

Figure 1: Example EIS Resource

The ConnectionFactory and Connection interfaces are defined by MyEIS as follows:

```
public interface com.myeis.ConnectionFactory
      extends java.io.Serializable, javax.resource.Referenceable {

      public com.myeis.Connection getConnection()
            throws com.myeis.ResourceException;

      public com.myeis.Connection getConnection(
            javax.resource.cci.ConnectionSpec properties )
            throws com.myeis.ResourceException;

      public javax.resource.cci.ConnectionSpec createConnectionSpec()
            throws com.myeis.ResourceException;

      public javax.resource.cci.ResourceAdapterMetaData getMetadata()
            throws com.myeis.ResourceException;
}

public interface com.myeis.Connection {

      public void close() throws com.myeis.ResourceException;

      public javax.resource.cci.ConnectionMetadata getMetadata()
            throws com.myeis.ResourceException;

      public javax.jmi.reflect.RefPackage getTopLevelPackage()
            throws com.myeis.ResourceException;
}
```

Copyright © 2002 John D. Poole. All rights reserved.

As stated previously, MyEIS's custom version of ConnectionFactory provides two factory methods for creating Connection instances:

```
com.myeis.Connection getConnection()

com.myeis.Connection getConnection( javax.resource.cci.ConnectionSpec
properties )
```

The first variant does not require the client to supply any properties. This variant is used in cases where a client component defers connection management to its container. The second variant accepts the J2EE™ Connector Architecture's ConnectionSpec interface as a means of specifying client-level connection properties. ConnectionSpec is essentially a *marker* or *tag* interface for a JavaBean implementation class that provides resource-specific properties required for establishing client-managed connections to EIS resources. A client introspects ConnectionSpec to determine the supported properties. These properties are accessible as JavaBean-style getter/setter methods. Default properties for ConnectionSpec, as defined by the J2EE™ Connector Architecture, are UserName and Password. An EIS may define additional properties, if necessary.

MyEIS uses an embedded JMI service as a means of providing advanced meta data management. Access to JMI services implemented by MyEIS is provided through the getTopLevelPackage() method of the Connection interface:

```
javax.jmi.reflect.RefPackage getTopLevelPackage()
```

This method returns a single object that implements the javax.jmi.reflect.RefPackage interface. This object is a JMI Package object that defines an outermost (or root-level) package containing all other objects representing meta data within MyEIS. Starting with this top-level instance of RefPackage, a client application may embark upon a process of discovery and access of the complete collection of meta data instances defined by MyEIS.

Copyright © 2002 John D. Poole. All rights reserved.

Metamodels and JMI Objects

As an example, let's assume that MyEIS has defined two simple metamodels of Relational and OLAP databases. This is illustrated in the diagram below[1]:

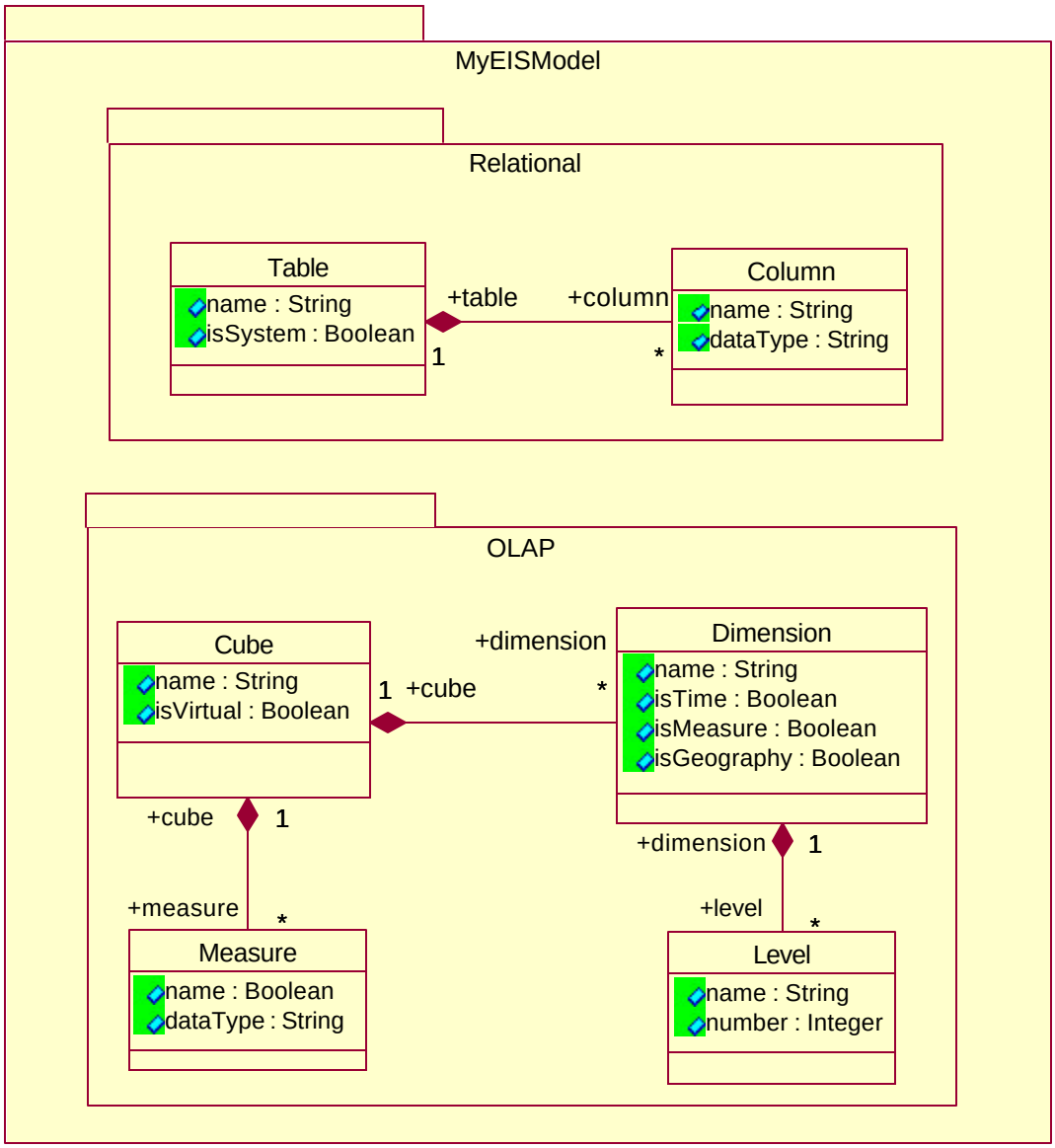

Figure 2: MyEIS Metamodels

[1] Note that we could have used the Relational and OLAP packages of the Common Warehouse Metamodel (CWM) in this example, but have chosen not to, so as to keep this example small and self-contained. The simplified Relational and OLAP metamodels defined above, however, still capture a number of essential aspects of their CWM counterparts.

Copyright © 2002 John D. Poole. All rights reserved.

The Relational metamodel consists of the (M2-level) Table and Column classes, a composition association relating Table and Column, and the single Relational Package containing the three relational modeling elements. The OLAP metamodel similarly consists of the (M2-level) Cube, Dimension, Level, and Measure classes, along with three associations defining their inter-relationships. These are likewise defined as modeling elements within a single OLAP Package.

The MyEISModel Package, as a modeling element, is effectively empty. It defines no classes itself, but instead, imports both the Relational and OLAP packages, thereby providing an outermost, *service-level* package for managing these technology-specific metamodels. The getTopLevelPackage() method always returns a single instance of the MyEISModel Package as a RefPackage object. Note that this behavior is strictly a convention of the underlying implementation, and is not required by JMI.

Now, we assume that the initial meta data for MyEIS consists of simple relational and OLAP model instances. The relational model consists of a single "Product" table consisting of columns representing the product attributes of "ID", "Description", and "Color". This is illustrated in the following (M1-level) object model:

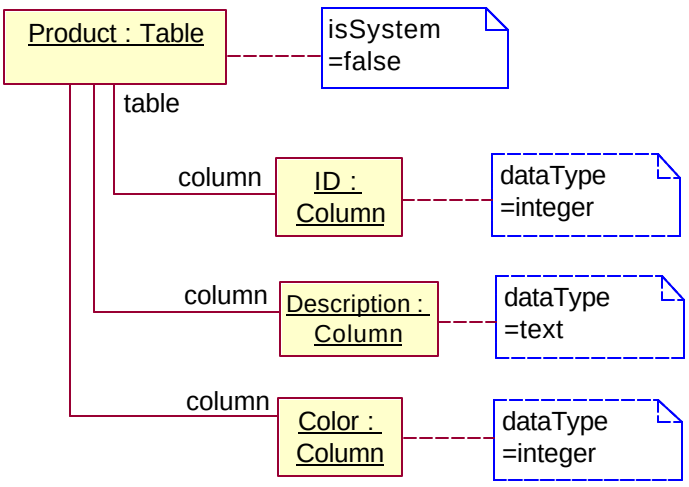

Figure 3: Relational Meta Data

The OLAP model consists of a definition of a "SalesAnalysis" Cube with three Dimensions: "Product", "Period", and "Location". The Cube has two Measures: "Sales" and "Cost". This is illustrated in the (M1-level) object model shown in Figure 4 below.

The object models shown in both diagrams are examples of an object-oriented representation of meta data. In JMI, the various meta data objects, associations, and packages are represented by Java language interfaces, as defined by the JMI interface mapping rules. Client applications can discover the overall structure and content of JMI-managed meta data by traversing these interfaces, once an initial connection to some top-level object has been established. In the particular case of MyEIS, this top-level or outermost object is, by convention, a single Java object that implements the JMI Reflective RefPackage interface.

Copyright © 2002 John D. Poole. All rights reserved.

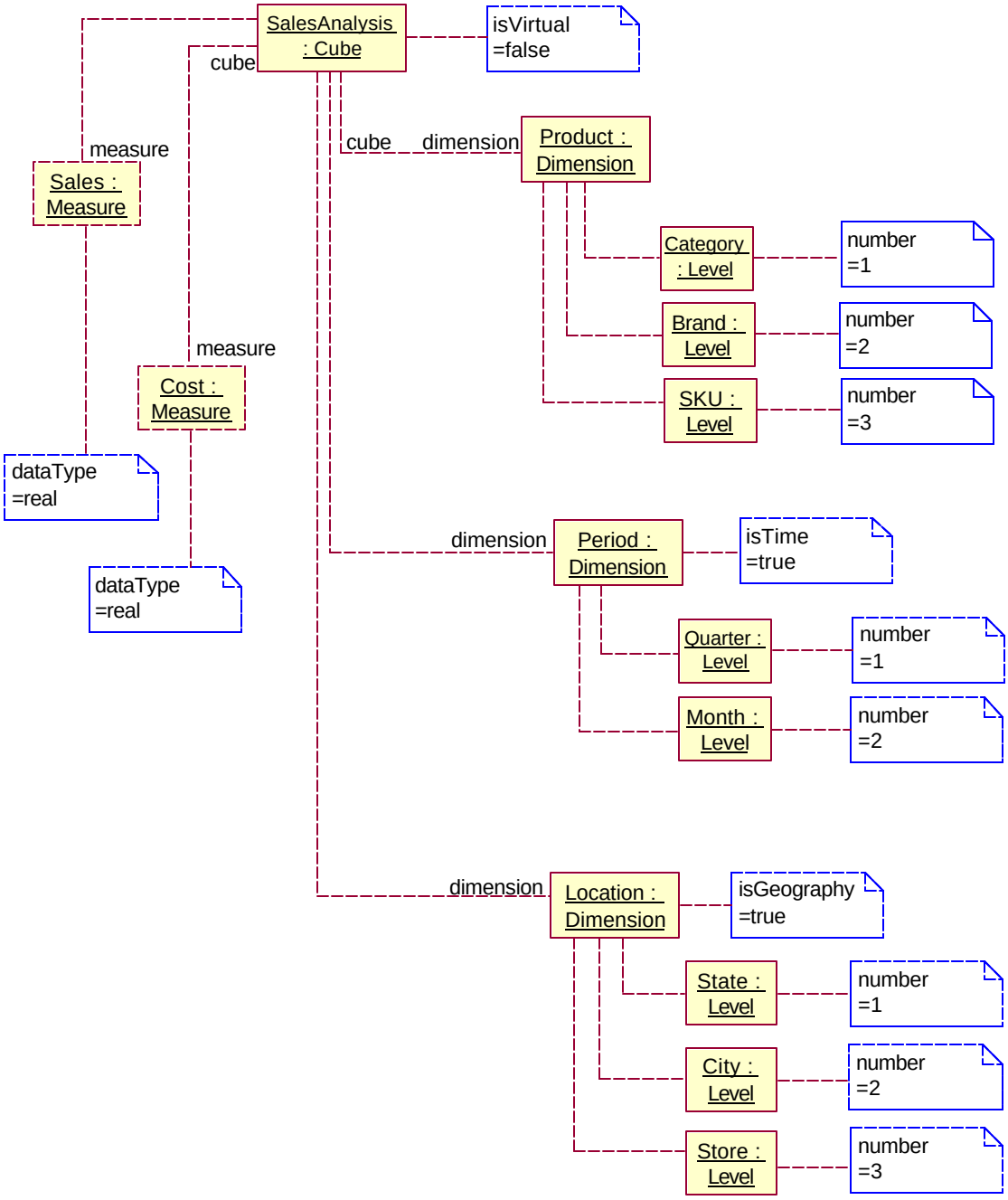

Figure 4: OLAP Meta Data

Copyright © 2002 John D. Poole. All rights reserved.

Meta Data Access Via JMI

As mentioned previously, a client gains access to the top-level "MyEIS" RefPackage object by calling the getTopLevelPackage() method on an instance of the Connection interface. From that point on, the client may access the rest of the meta data via either the tailored (metamodel-specific) JMI interfaces, or through JMI Reflective methods.

The figure below illustrates a complete Java program that obtains a Connection to MyEIS, obtains the top-level package object, uses the tailored JMI interfaces to find all instances of both Relational Table and OLAP Dimension, and then prints the name of each instance:

```
import java.util.*;
import javax.naming.*;
import com.myeis.*;

public class EISTest
{
	public static void main (String[] args)
	{
		EISTest eisTest = new EISTest();

		// connect to a EIS resource instance
		eisTest.connectToEISResource();

	} // main

	public void connectToEISResource()
	{
		try
		{
			// Obtain the EIS ConnectionFactory from the JNDI tree

			Context initCtx = new InitialContext();
			com.myeis.ConnectionFactory cxf
				= (com.myeis.ConnectionFactory)initCtx.lookup(
					"java:comp/env/eis/MyEIS" );

			javax.resource.cci.ConnectionSpec cxs
			= (javax.resource.cci.ConnectionSpec)cxf.createConnectionSpec();

			// Set the user name and password values

			((com.myeis.ConnectionSpecImpl)cxs).setUserName( "guest" );
			((com.myeis.ConnectionSpecImpl)cxs).setPassword( "password" );

			// Now, establish a connection to the EIS resource

			com.myeis.Connection cx
				= (com.myeis.Connection)cxf.getConnection( cxs );

			// Obtain the top-level EIS meta data package extent from
			// the Connection and get the relational and OLAP package
```

Copyright © 2002 John D. Poole. All rights reserved.

```
                // interfaces

                javax.jmi.reflect.RefPackage tlp
                        = (javax.jmi.reflect.RefPackage)cx.getTopLevelPackage();

                com.myeis.MyEISModel.RelationalPackage relPkg
                        = ((com.myeis.MyEISModel)tlp).getRelationalPackage();

                com.myeis.MyEISModel.OLAPPackage olapPkg
                        = ((com.myeis.MyEISModel)tlp).getOlapPackage();

                // Get the collection of all Tables in the relational meta data
and
                // print their names

                com.myeis.MyEISModel.Relational.TableClass tableProxy
                        = relPkg.getTable();
                Collection tableCol = tableProxy.refAllOfClass();
                Iterator tableIter = tableCol.iterator();
                while ( tableIter.hasNext() )
                {
                        Object obj = tableIter.next();
                        if ( obj instanceof com.myeis.MyEISModel.Relational.Table )
                                System.out.println( "Table name is " +
                        ((com.myeis.MyEISModel.Relational.Table)obj).getName() );
                }

                // Get the collection of all Dimension in the OLAP meta data and
                // print their names

                com.myeis.MyEISModel.OLAP.DimensionClass dimProxy
                        = olapPkg.getDimension();
                Collection dimCol = dimProxy.refAllOfClass();
                Iterator dimIter = dimCol.iterator();
                while (dimIter.hasNext() )
                {
                        Object obj = dimIter.next();
                        if ( obj instanceof com.myeis.MyEISModel.OLAP.Dimension )
                                System.out.println( "Dimension name is " +
                        ((com.myeis.MyEISModel.OLAP.Dimension)obj).getName() );
                }

                // Now close the connection to the EIS resource

                cx.close();

                System.out.println( "Done");

        } // try
        catch ( javax.naming.NamingException e )
        {
                e.printStackTrace();
        }
        catch ( com.myeis.ResourceException e )
```

Copyright © 2002 John D. Poole. All rights reserved.

```
            {
                  e.printStackTrace();
            }
      } // connectToEISResource
}
```

Figure 5: Simple JMI Client Application

Running this program against the meta data described in the previous section would produce the following output:

Table name is Product
Dimension name is Product
Dimension name is Period
Dimension name is Location
Done

Summary

In this paper, we illustrated one of but many possible ways to establish a connection to a JMI-enabled meta data resource and then peruse its meta data content. In this particular case, we relied on the fact that a custom implementation of the J2EE™ Connector Architecture's Common Client Interface's Connection class provided a method whereby a single JMI RefPackage instance could be obtained. This object (purely by convention) represents an outermost or top-level JMI Package object that contains the rest of the standard JMI metaobject instances, as defined by the JMI Specification. We also provided a simple Java application client that uses the JMI tailored interfaces derived from two example metamodels to obtain meta data instances from a JMI-enabled meta data resource.

Copyright © 2002 John D. Poole. All rights reserved.