UMLMDA - Modelling Guide

Dieter Moroff <moroff@user.sourceforge.net>

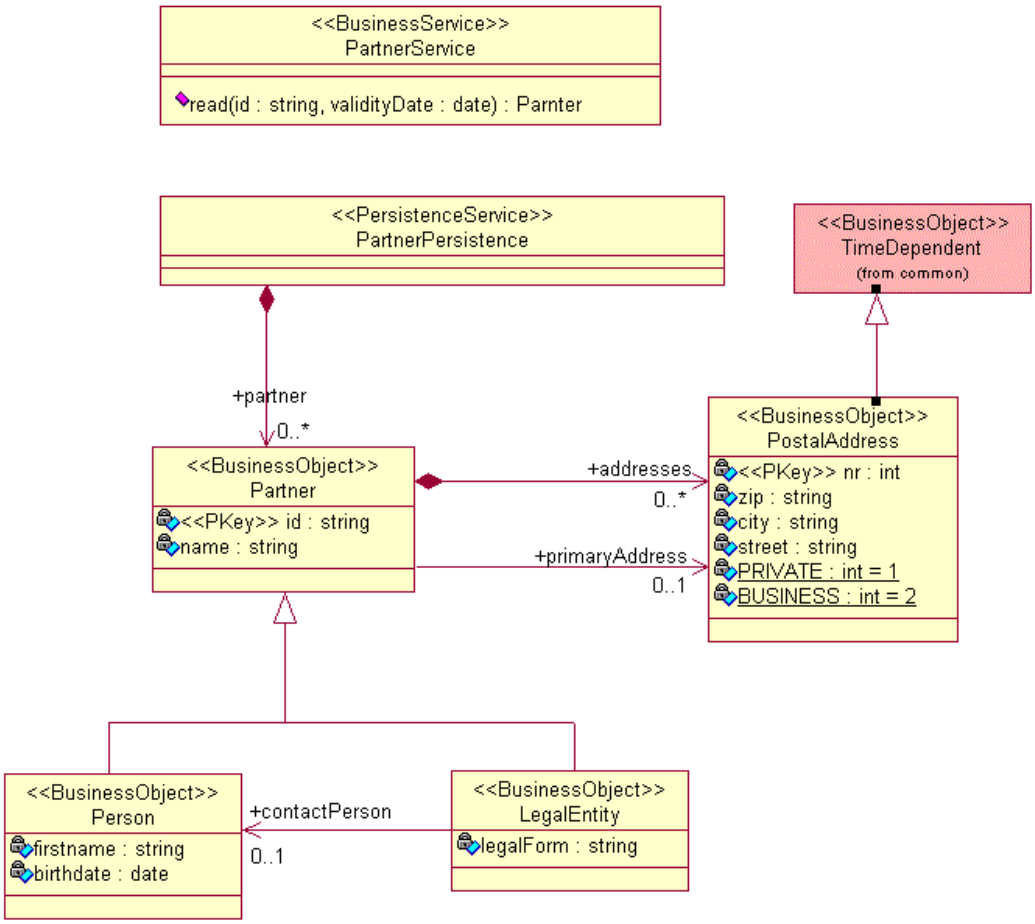
Table of Contents

1. Stereotypes 1
    1.1. BusinessComponent 2
    1.2. BusinessObject 2
    1.3. BusinessService 2
    1.4. PersistenceService 2
    1.5. PKey 2
2. Tagged Values 3
    2.1. Package 3
    2.2. Class 4
    2.3. Attribute 5
3. Generator Patterns 6
    3.1. Java Generator Patterns 6
    3.2. J2EE Generator Patterns 8

1. Stereotypes

The section describes the stereotypes used in models when using the out of the box generator from UMLMDA. If you use your own writers you may use different stereotypes and tagged values to define your own UML profile.

Figure 1. Sample using the different stereotypes on class and attribute level.



1.1. BusinessComponent

This stereotype is used for packages which contains objects and services which are packaged in a business component. Typically a business component is used to control the deployment unit.

1.2. BusinessObject

This stereotype is used for classes which primary function is to store data. Business objects are used as value objects with business services or used as structures using it with corba interfaces. This classes usually contains only attributes or constants.

1.3. BusinessService

This stereotype is used for classes which primary function is to provide business functionality or business services. This type of classes usually contains only methods or constants.

1.4. PersistenceService

This stereotype is used for classes which provide persistence services for business object classes. Methods to control the persistence lifecycle of business object classes are provided.

1.5. PKey

This stereotype is used for attributes in business object classes, which contains the primary key attributes.

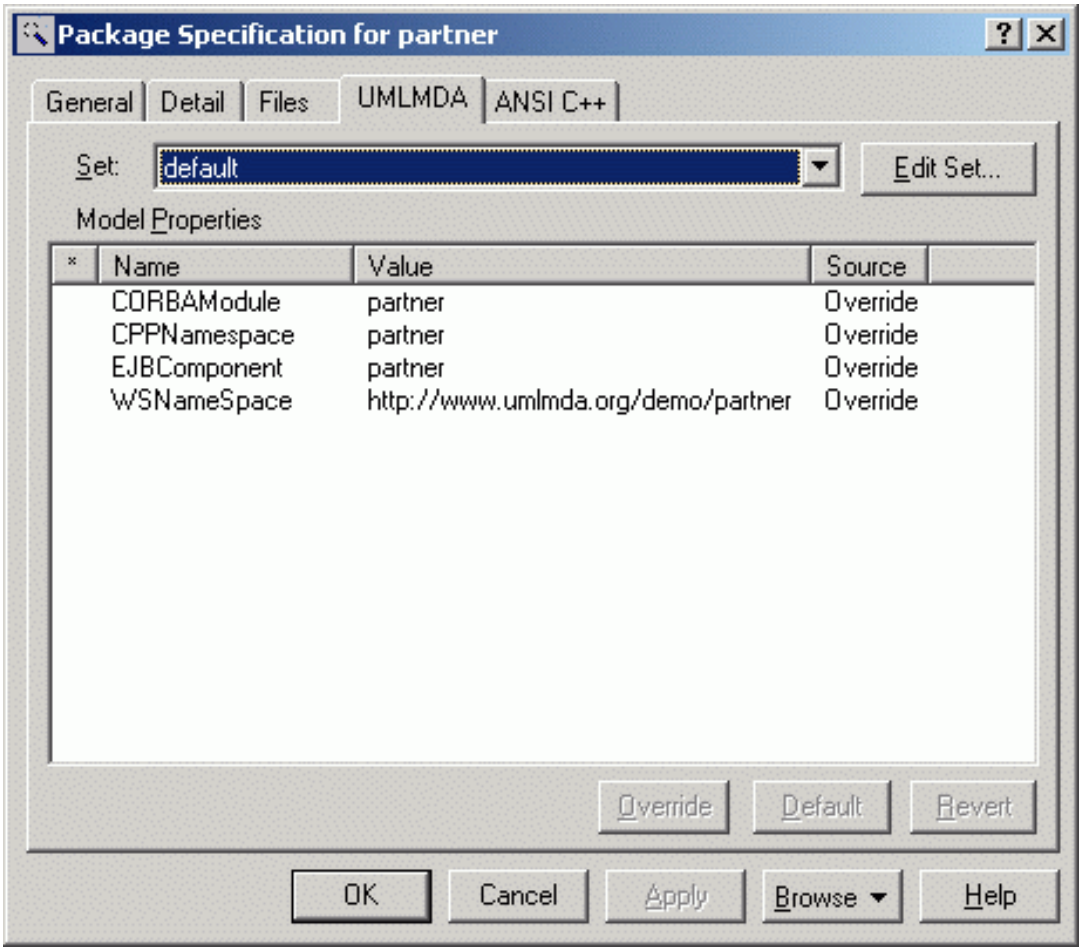
2. Tagged Values

$Revision$

This section describes the tagged values used used on the different model elements.

2.1. Package

Figure 2. Sample: tagged values for a package in Rational Rose



CORBAModule

Name of the CORBA module, used as name for the generated idl and as identifier for the CORBA naming service.

CPPNamespace

Name of the c++ namespace.

EJBComponent

Name of the EJB Component, this name is used as prefix in the jndi naming service.

WSNamespace

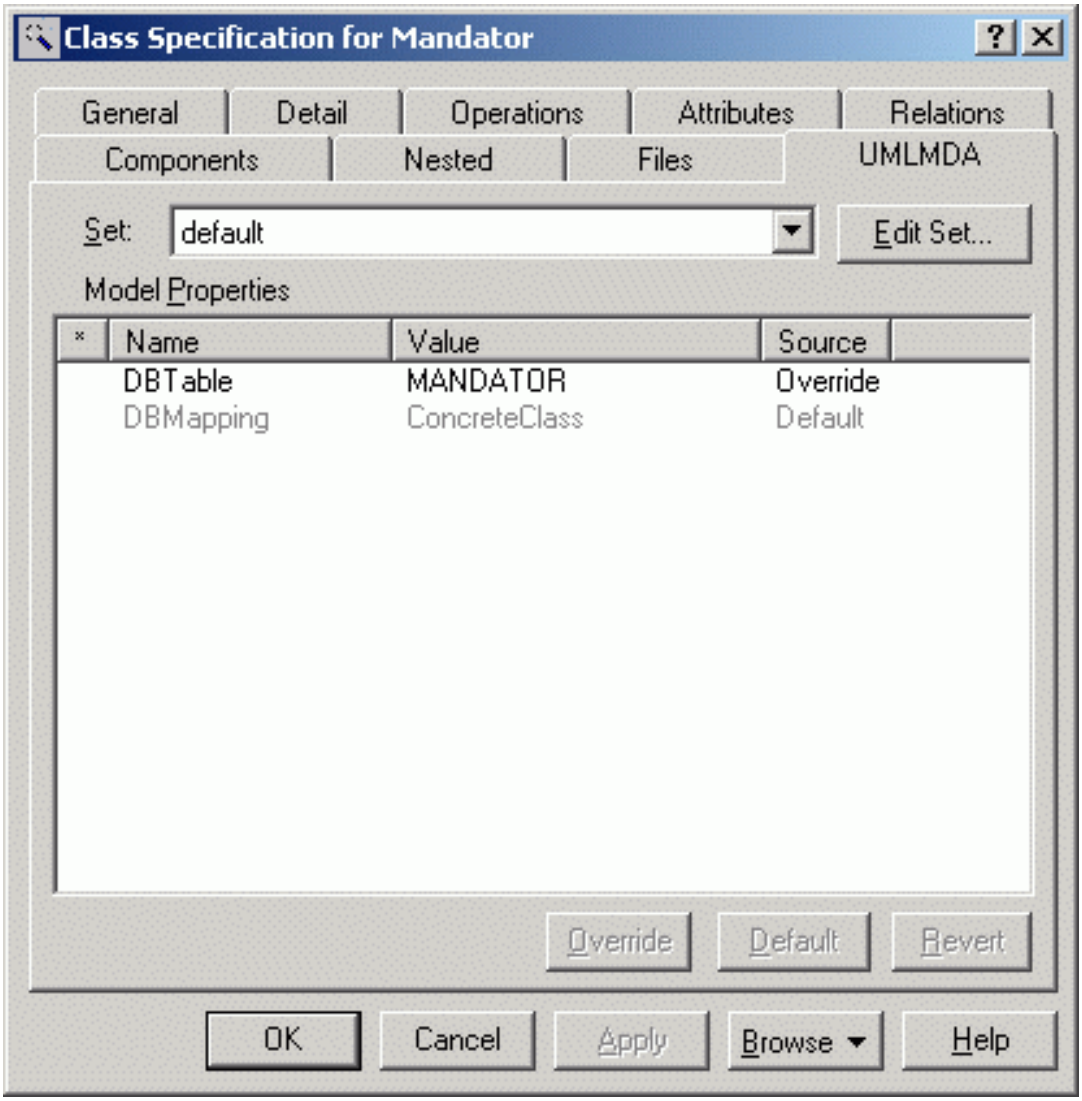
Namespace use in the gerneated wsdl files.

WSNamespacePrefix

Namespaceprefix use in the gerneated wsdl files, if empty default prefix tns1 is used.

2.2. Class

Figure 3. Sample: tagged values for a class in Rational Rose



DBTable

Name for the relational database table to with instances of the class are stored.

DBMapping

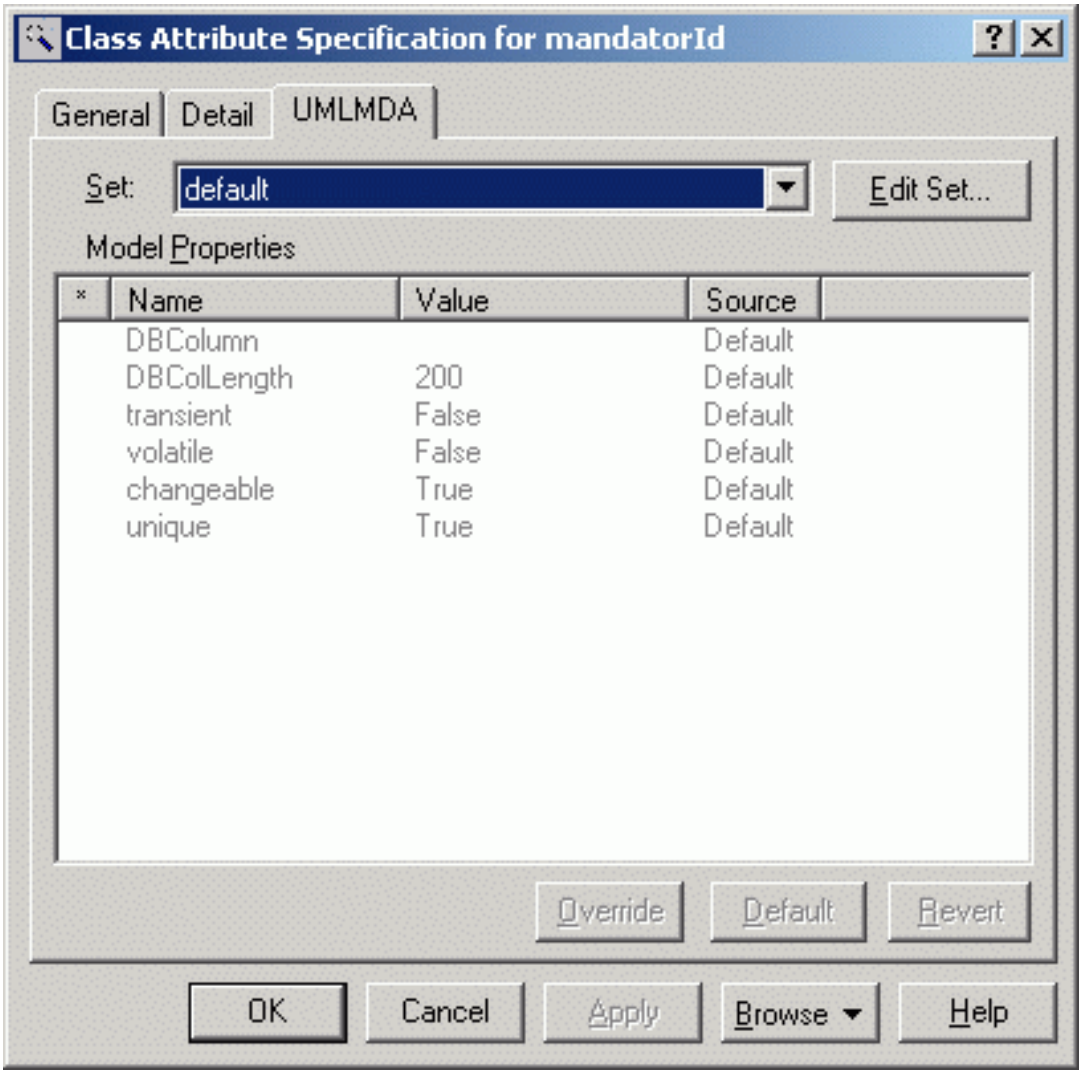
Table 1. Possible values for tagged value DBMapping

ClassHierarchie	The attributes from all classes in the same class

	hierarchie are mapped in the same database table. This mapping is fast, but using not null columns or foreign keys must be used carefully, because depending on the concrete class instance, some columns are not filled because they don't exist in the class. The name of the database table must be specified only in the base class it's ignored in the subclasses.
ConcreteClass	Every concrete class is mapped in a different database table. This mapping can't be used if classes with the same base class are used in a polymorph collection. The name of the database table must be specified in every class.
SubClass	The attributes from the base class and the derived classes arev stored in different database tables. Ready an instance of a subclass requires database reads in multiple tables. The name of the database table must be specified in every class.

2.3. Attribute

Figure 4. Sample: tagged values for an attribute in Rational Rose



3. Generator Patterns

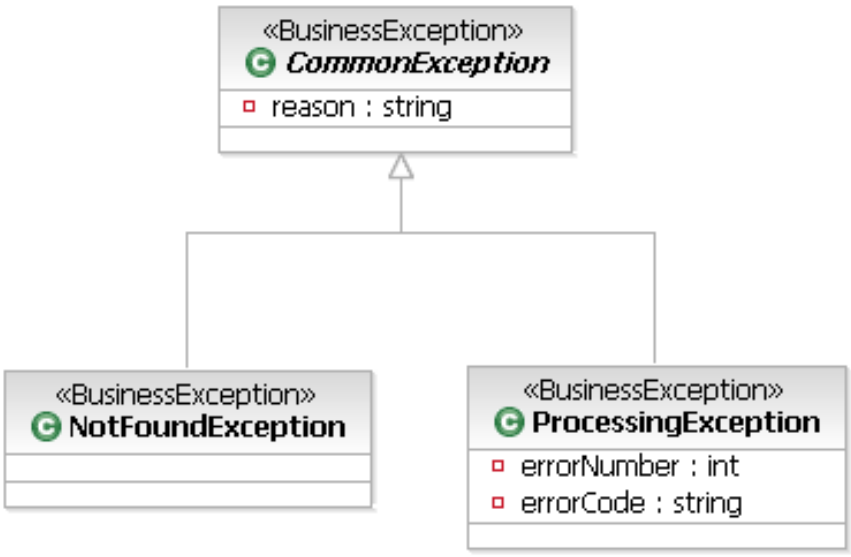
This section describes the generator patterns used by the UMLMDA out of the box writers.

3.1. Java Generator Patterns

This section describes how models using the UMLMDA profiles are mapped to Java code using the Java writers.

3.1.1. <<BusinessException>>

From a model class with the stereotype <<BusinessException>> a Java class extending java.lang.Exception is generated. If the exception have a modelled superclass it's extending this class instead.



For every attribute a getter and a setter method is generated. Addition code could be added to user sections.

```
package org.umlmda.demo.common;

//@@UserSection-Begin CommonException.Imports
//@@UserSection-End CommonException.Imports

/**
 * Base of all exceptions.
 */
public class CommonException extends Exception
{
   public CommonException(String reason)
   {
      this.reason = reason;
   }

   private String reason = "";

   /**
    * Default Ctor
    */
   public CommonException() {
      super();
   }

   /**
   * Get reason
   */
   public String getReason() {
      return reason;
   }
   /**
   * Set reason
   */
   public void setReason(String reason) {
      this.reason = reason;
   }

   //@@UserSection-Begin CommonException.NonGeneratedMembers
   //@@UserSection-End CommonException.NonGeneratedMembers
};
```

3.1.2. <<BusinessObject>>

TBD.

3.1.3. <<BusinessService>>

TBD.

3.1.4. <<BusinessComponent>>

TBD.

3.2. J2EE Generator Patterns

This section describes how models using the UMLMDA profiles are mapped to J2EE code and deployment descriptors using the J2EE writers.

3.2.1. <<BusinessException>>

Nothing is generated for this stereotype. The Code generated from the Java generator profile is used.

3.2.2. <<BusinessObject>>

Nothing is generated for this stereotype. The Code generated from the Java generator profile is used.

3.2.3. <<BusinessService>>

For this stereotype the typical J2EE patterns are generated in a j2ee subpackage from the original package, that is:

- Local- and Remote-Interface
- Home- and LocalHome-Interface
- SessionBean implementation delegating to the business implementation
- Utils (Servicelocator, ...)
- J2EE Client Adapter, delegating from the business interface to the J2EE client interfaces

3.2.4. <<BusinessComponent>>

For this stereotype the following artefacts are generated:

- J2EE Client Adapter factory, implementing the factory interface generated with the Java generator profile.
- `ejb-jar.xml` deployment descriptor
- `jboss.xml` JBoss specific deployment descriptor containing JNDI bindings.
- `ibm-ejb-jar-ext.xmi`, `ibm-ejb-jar-bnd.xmi` IBM WebSpere specific deployment descriptor containing JNDI bindings.
- more vendor specific deployment describtors coming ...

To be done, describe the other patterns (WS, XML, SQL, Hibernate, ...)