Implementing Model Driven Architecture using Enterprise Architect

MDA in Practice

By Frank Truyen

frank.truyen@cephas.cc

for

Sparx Systems Pty Ltd

Version 1.1

Table of Contents

Model Driven Architecture and Enterprise Architect

Abstract

This document illustrates how Enterprise Architect can be leveraged to implement the MDA approach. It specifically covers:

Tool setup, including creating or customizing transformation templates.

PIM-to-PSM transformations.

PSM-to-PSM transformations.

Also reviewed is the support of the tool for UML profiles, MOF meta-modeling and OCL constraint specifications.

For background information on MDA concepts and their mapping to Enterprise Architect features please refer to the document – Mapping MDA Concepts to EA Features.

The Enterprise Architect solution for applying MDA

In **Mapping MDA Concepts to EA Features** we validated not only that *Enterprise Architect* satisfies all of the core requirements for being considered MDA compliant, but that it also provides many additional features to support this new development approach.

For a complete list of all of the product's features please visit the Sparx website[1]. Here our focus is primarily on those capabilities which are relevant to the implementation of MDA.

Product history

Enterprise Architect is a mature UML 2.0 based modeling tool for the Windows platform (with a version for Linux running under *Cross-Over Office*) which has been in development for over seven years, with a five year commercial history and record of fast evolution and impressive innovation.

Commitment to MDA and OMG standards in general

Sparx Systems has at numerous times stated their strong commitment to MDA and its underlying OMG standards such as UML, MOF and XMI. Its mission statement is quoted here:

> *To provide an affordable, high-quality, team based modeling environment founded on the UML 2.0 specification, with comprehensive support for model to model transformations as well as model driven generation of common development artifacts such as documentation, source code, test scripts, deployment descriptors, XML schemas, database schemas, etc.*

As will be illustrated below, *Enterprise Architect* offers a number of features targeted at MDA driven development, including a model to model transformation engine, allowing modelers to target multiple platform specific models from a single PIM – and to synchronize PIM changes into each PSM on demand. The built-in transformation templates include mappings to C#, DDL, EJB, Java, JUnit, NUnit, WSDL and XSD (XML Schema).

[1] http://www.sparxsystems.com/

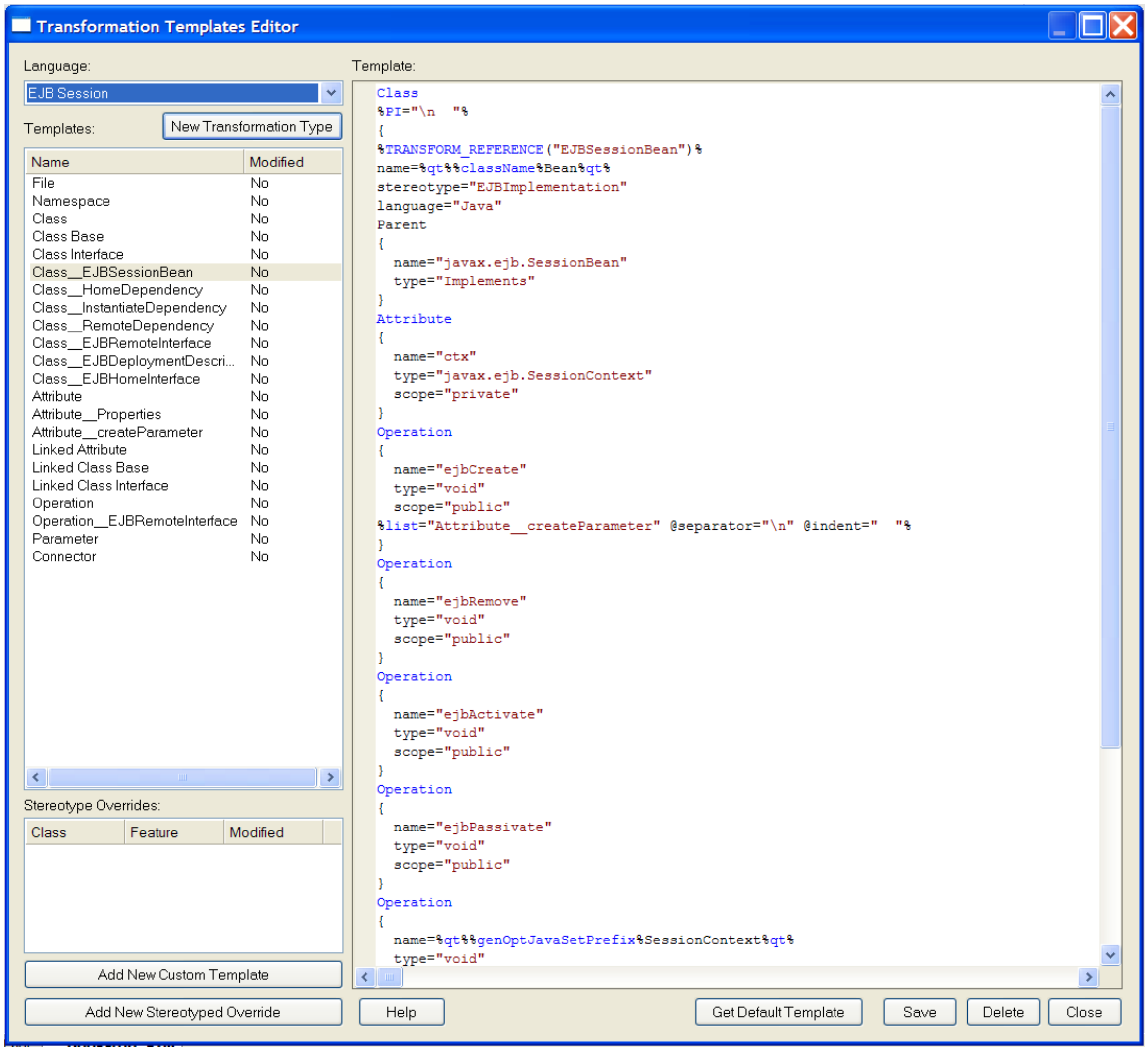
A look under the covers

Let us take a closer look at how *Enterprise Architect* implements some of its MDA features.

The core MDA component in EA is a template based transformation engine that generates PSM elements from a PIM source. Templates are one of the ways for expressing transformations endorsed by the official MDA guidelines (refer to **Mapping MDA Concepts to EA Features**). These customizable MDA transformation templates use an approach and syntax similar to the EA code generation templates.

Creating and Modifying Transformation Templates

The editor allows one to view and/or modify the predefined transformation templates (in this diagram an *EJB Session* template), as well as to create new ones.



Language: EJB Session

Templates:

Name	Modified
File	No
Namespace	No
Class	No
Class Base	No
Class Interface	No
Class__EJBSessionBean	No
Class__HomeDependency	No
Class__InstantiateDependency	No
Class__RemoteDependency	No
Class__EJBRemoteInterface	No
Class__EJBDeploymentDescri...	No
Class__EJBHomeInterface	No
Attribute	No
Attribute__Properties	No
Attribute__createParameter	No
Linked Attribute	No
Linked Class Base	No
Linked Class Interface	No
Operation	No
Operation__EJBRemoteInterface	No
Parameter	No
Connector	No

Stereotype Overrides:

Class	Feature	Modified

Template:

```
Class
%PI="\n  "%
{
%TRANSFORM_REFERENCE("EJBSessionBean")%
name=%qt%%className%Bean%qt%
stereotype="EJBImplementation"
language="Java"
Parent
{
  name="javax.ejb.SessionBean"
  type="Implements"
}
Attribute
{
  name="ctx"
  type="javax.ejb.SessionContext"
  scope="private"
}
Operation
{
  name="ejbCreate"
  type="void"
  scope="public"
%list="Attribute__createParameter" @separator="\n" @indent="  "%
}
Operation
{
  name="ejbRemove"
  type="void"
  scope="public"
}
Operation
{
  name="ejbActivate"
  type="void"
  scope="public"
}
Operation
{
  name="ejbPassivate"
  type="void"
  scope="public"
}
Operation
{
  name=%qt%%genOptJavaSetPrefix%SessionContext%qt%
  type="void"
```

The general principle is quite simple: EA provides access to the various elements (classes, attributes, operations, etc.) of the UML source model being processed. The scripting language defines the rules by which properties of each source element can be copied and/or transformed into a corresponding target model element. The mapping does not have to be one-to-one: new target elements can created as needed.

EA also constructs internal bindings between each target PSM and its source PIM. This is essential for allowing iterative forward synchronization from the PIM to the PSM, adding or deleting features as required. For example, adding a new attribute to a source class will result - during forward synchronization - in a new column being added to the corresponding *table* element in the target data model. Element features in the target model which were not originally generated by the transformation process are not deleted or affected in any way. Note that these references between source and target elements are not visible to the user (i.e. they are neither instances of UML associations nor instances of what EA terms *cross references*).

Transformation marks, typically in the form of stereotypes or tagged values, can be used inside the script to drive the transformation process as shown in the following examples.

Snippets from templates illustrating the use of marks to drive transformations:

```
%if classStereotype=="enumeration"%
  stereotype="enumeration"
%else%
  stereotype="XSDcomplexType"
%endIf%
```

And:

```
$qualifers = %opTag:"EATypeQualifiers"%
%if $qualifers != ""%
  %REPLACE($qualifers,";","::")%
%endIf%
```

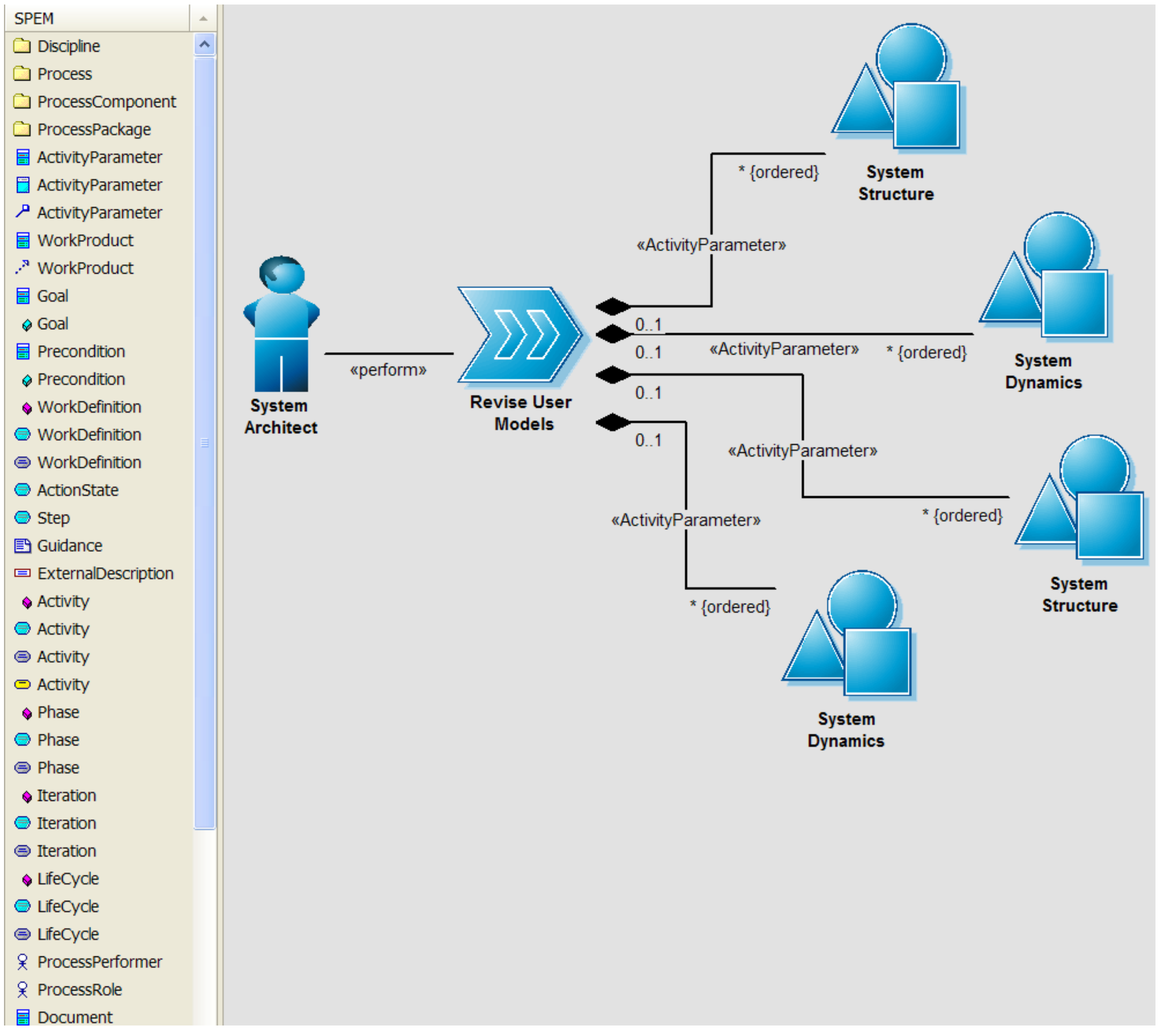
Please refer to the example further down in this document for a detailed description of the process required to take a sample platform independent model (PIM) and transform it into a number of different platform specific models (PSM).

Building and leveraging UML profiles

Sparx provides a list of commonly used UML profiles, freely available for download from their website (http://sparxsystems.com.au/resources/), in particular profiles for business, presentation, process and XML modeling.

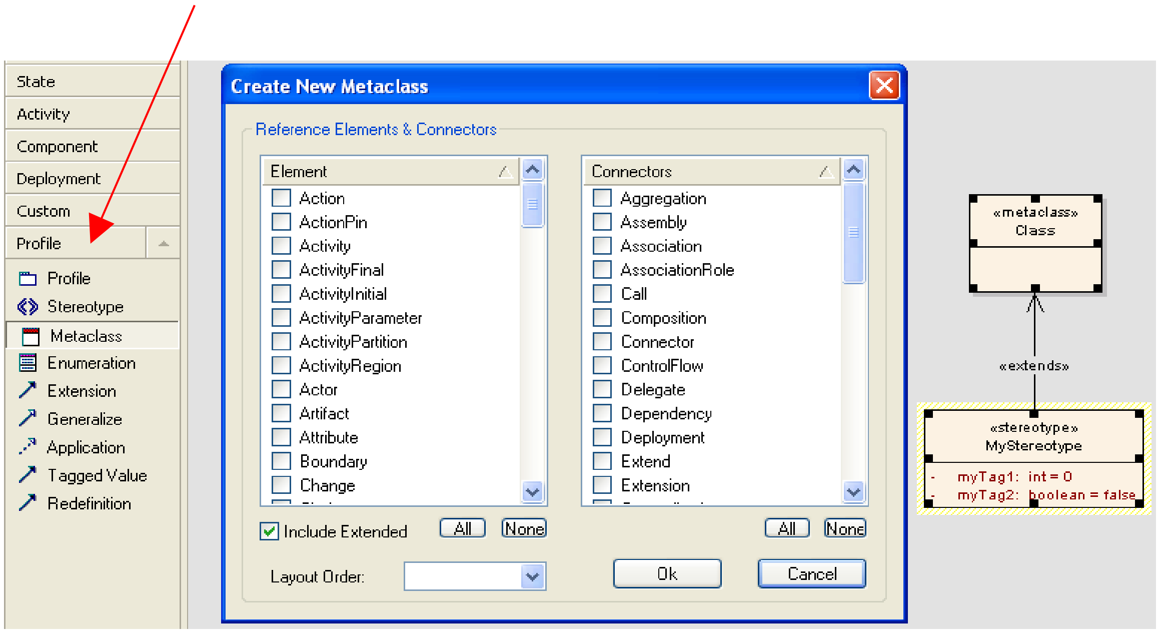
These profiles can be loaded into the *Resources* view of the tool and from there made part of the standard toolbox section.

The following example shows a model leveraging the *Software Process Engineering Metamodel* (SPEM) profile.

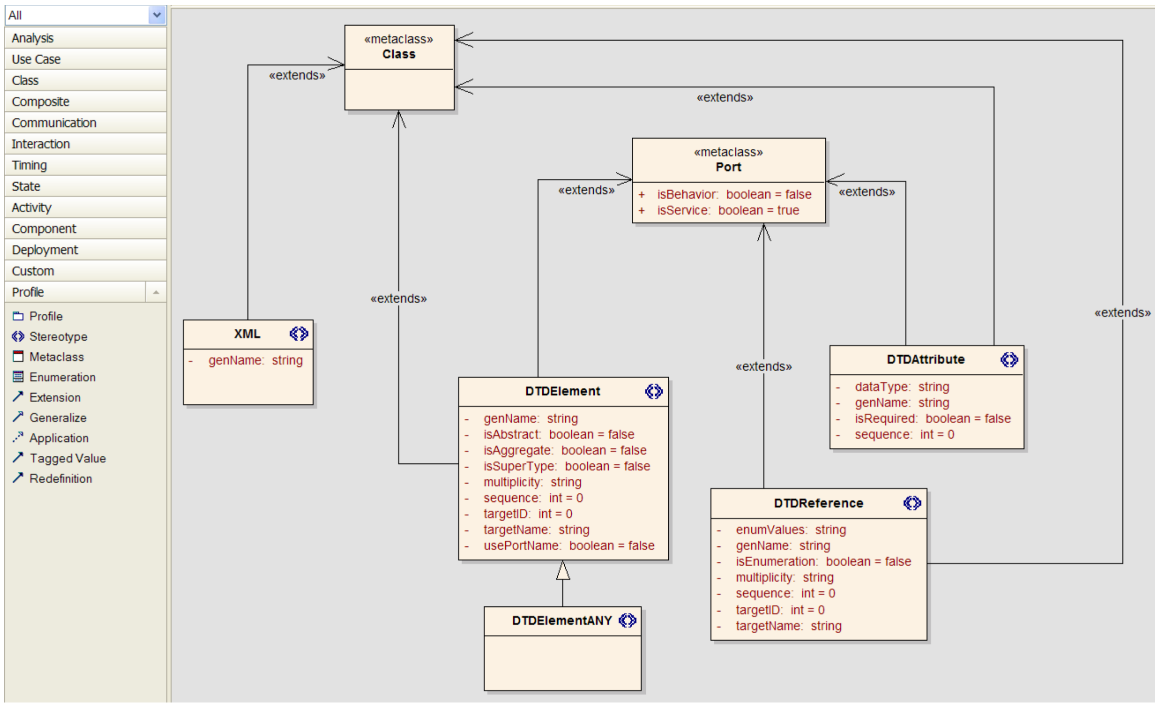


In the context of MDA, such profiles can be used as a basis for creating platform independent models with the necessary stereotypes and other marks to drive the later transformation into platform specific models.

The user can also create custom profiles by extending the UML metamodel with new stereotype definitions. Tagged values and other features can then be associated with these stereotypes. EA provides the necessary interfaces for this purpose in a dedicated toolbox section:



The following diagram shows a custom UML profile for creating XML DTD models.



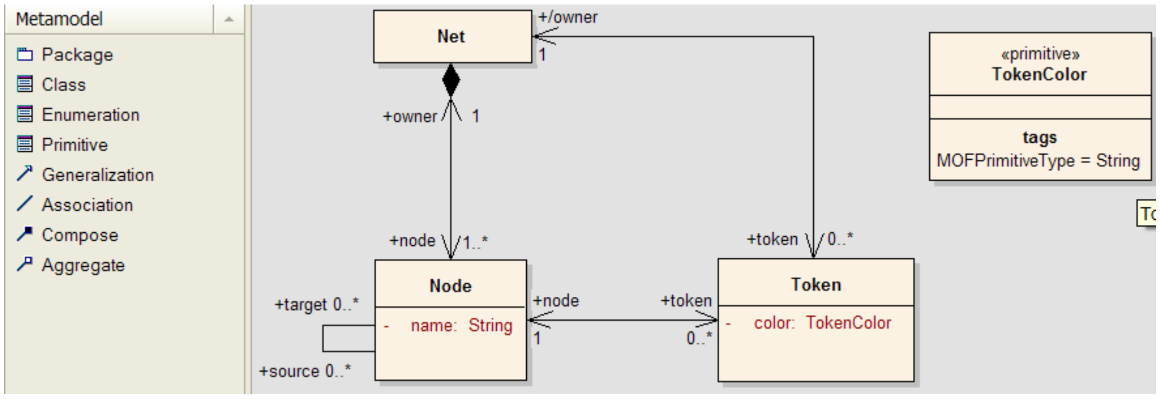
Creating MOF models

EA supports the ability to create MOF models[2] and export them for storage into an external Metadata Repository (MDR) tool such as NetBeans[3]. Supported MOF versions are 1.3 and 1.4.

MOF models are often used to create domain specific modeling languages (DSL) which can not be properly defined either in UML itself, or via the UML profile extension capability. Such user defined metamodels can then operate at the same meta level as UML.

The concepts of the DSL can be modeled in EA using the standard UML class diagram interface, but with a limited set of metamodel constructs defined by MOF (Package, Class, Enumeration and Primitive).

EA provides a dedicated area of its toolbox to build MOF-based metamodels, such as the network graph DSL example shown here.

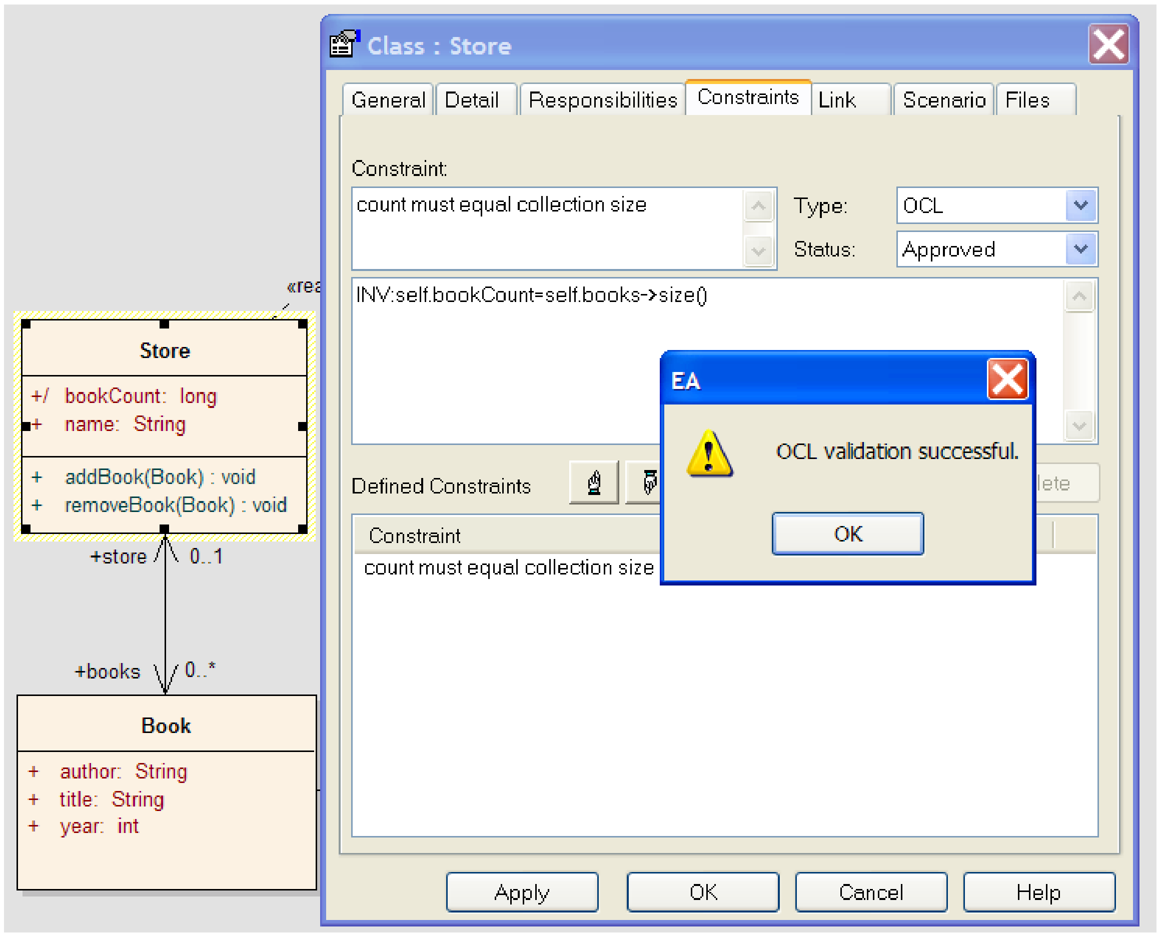


Note that the creation of an instance model based on this MOF metamodel is currently not supported by EA.

[2] http://www.omg.org/technology/documents/formal/mof.htm
[3] (http://mdr.netbeans.org/).

Defining constraints in OCL

OCL syntax can be used most everywhere UML constraints can be specified (class or attribute invariants, operation pre- or post-conditions, activity constraints, action constraints, etc – but currently not on messages or control flows). The syntax validation occurs automatically when saving the constraint.



Constraint: count must equal collection size
Type: OCL
Status: Approved

```
INV:self.bookCount=self.books->size()
```

OCL validation successful.

Defined Constraints

Constraint
count must equal collection size

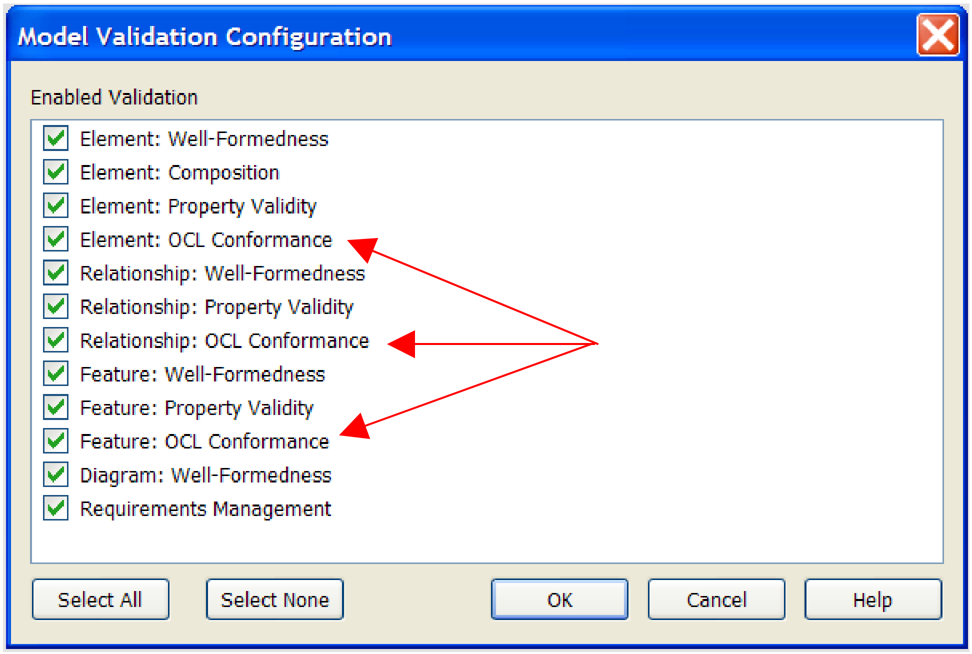
EA also supports the ability to validate the OCL statements against the model itself, meaning to verify beyond just the syntax that the OCL statement is expressed correctly in terms of actual model elements, and that the kind of validation syntax that it uses corresponds to the actual data types defined for these elements: numbers, strings, collections, etc.

In version 6.0 this validation is restricted to invariants only (i.e. OCL statements starting with the string 'INV:'). Upcoming releases will broaden the scope of validation to pre- and post-conditions also.

OCL validation needs to be enabled explicitly in the Model Validation Configuration dialog, accessible from the Main Menu by selecting:

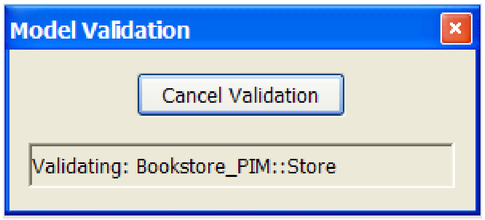
Project → Model Validation → Configure…

Note the three separate line items covering OCL conformance:

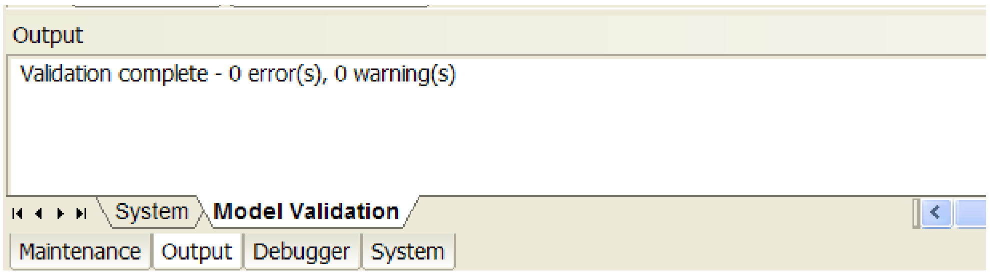


- ✓ Element: Well-Formedness
- ✓ Element: Composition
- ✓ Element: Property Validity
- ✓ Element: OCL Conformance
- ✓ Relationship: Well-Formedness
- ✓ Relationship: Property Validity
- ✓ Relationship: OCL Conformance
- ✓ Feature: Well-Formedness
- ✓ Feature: Property Validity
- ✓ Feature: OCL Conformance
- ✓ Diagram: Well-Formedness
- ✓ Requirements Management

Validation can be operated against a single UML element, a diagram, a package hierarchy, or the entire model. The results are displayed in the *Output* window.



Validating: Bookstore_PIM::Store



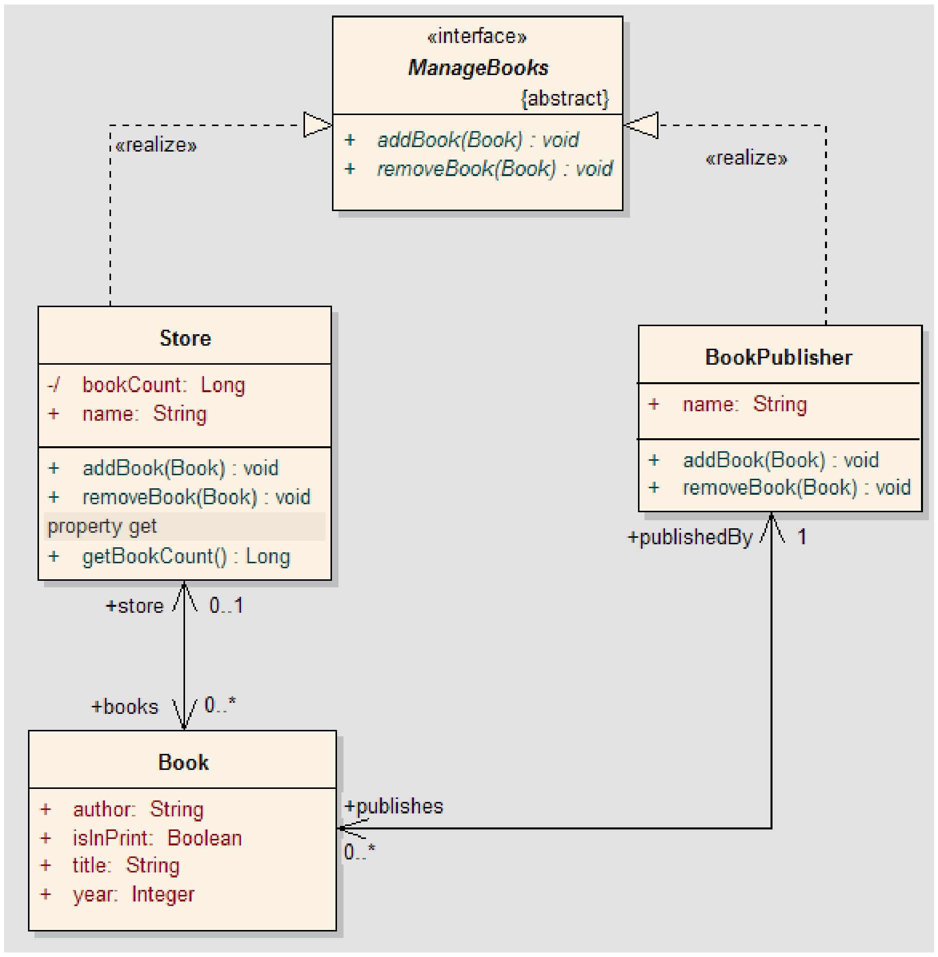
```
Validation complete - 0 error(s), 0 warning(s)
```

MDA by example

Starting from a simple platform independent model, the remaining pages illustrate a number of mappings to platform specific models using the default transformation templates defined in *Enterprise Architect*.

Base Platform Independent Model (PIM)

In this example the base PIM is void of any transformation marks (special stereotypes or tags): all of the transformation rules used are embedded inside the default templates.



Please note:

- The use of generic data types in this model : Integer, Long, Boolean and String. These data types will be mapped automatically by the transformation template using the CONVERT_TYPE macro (refer to the EA User Documentation for details).
- To automatically map more complex data types, such as a Date for example, a customization of the transformation template may be required. Likewise for attributes where the multiplicity is greater than 1 (or which are marked as collections) since these settings are often ignored in the default templates.

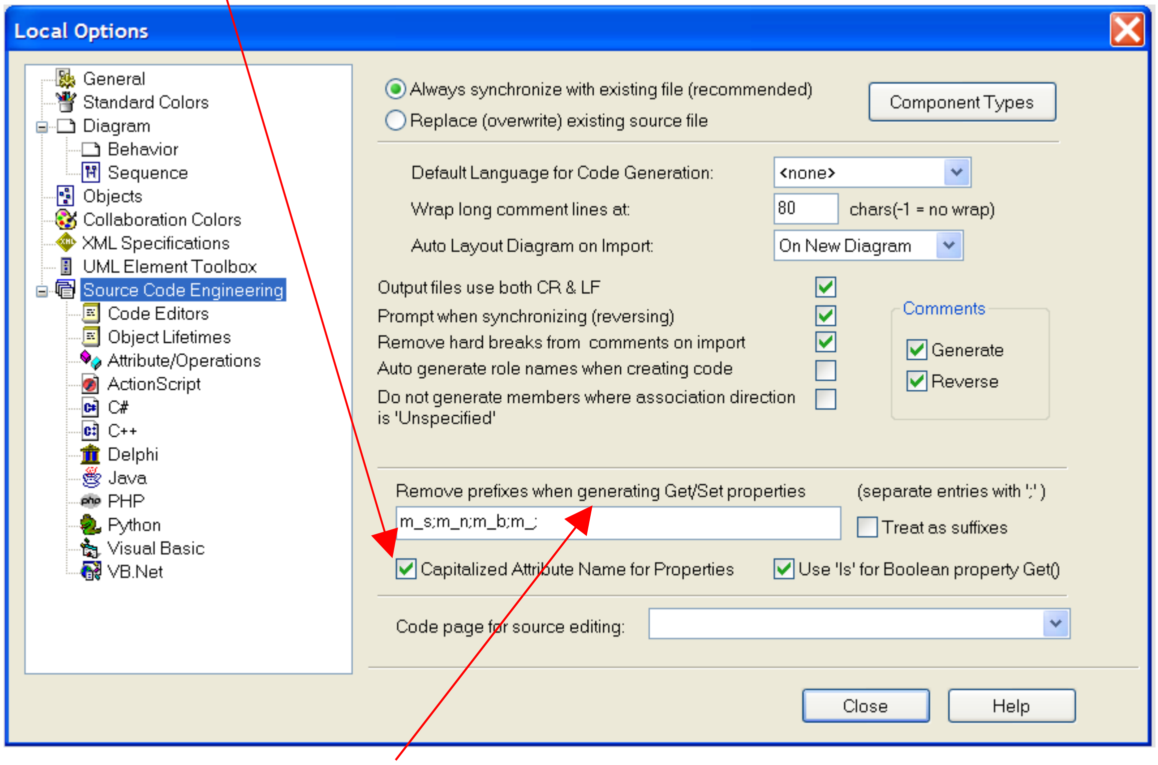
- The attributes for which accessor & mutator operations are desired as output results (also called *properties*), for example in Java and C#, should be scoped as *public* in the generic model. The transformation templates will automatically convert the attribute to *private* scope in the target model, as well as create the expected *get()* and *set()* operations. To avoid this behavior, mark the attribute as *private* in the PIM.
- The transformation templates typically ignore the marking of an attribute as read-only via the *derived* or *constant* flags - cases in which only an accessor operation should be generated, and not a mutator. To get around this issue, mark the attribute as *private* and either:
  - Create a manual accessor operation.
  - Mark it as a read-only *property* inside the PIM (see the *bookCount* derived attribute in the *Store* class above for an example).
  - Customize the template using the *attConst or attDerived* macros.

Configuration options

Additional configuration options (using the *Tools → Options* menu item) which can help with certain transformation processes later on are:

Capitalize attribute names for properties

This is especially important for transformations into the C# language, to avoid naming conflicts.



[✓ Always synchronize with existing file (recommended)] Replace (overwrite) existing source file

Default Language for Code Generation: <none>
Wrap long comment lines at: 80 chars(-1 = no wrap)
Auto Layout Diagram on Import: On New Diagram

Output files use both CR & LF [✓]
Prompt when synchronizing (reversing) [✓]
Remove hard breaks from comments on import [✓]
Auto generate role names when creating code [ ]
Do not generate members where association direction is 'Unspecified' [ ]

Comments: Generate [✓] Reverse [✓]

Remove prefixes when generating Get/Set properties (separate entries with ';')
m_s;m_n;m_b;m_;
Treat as suffixes [ ]
Capitalized Attribute Name for Properties [✓]
Use 'Is' for Boolean property Get() [✓]

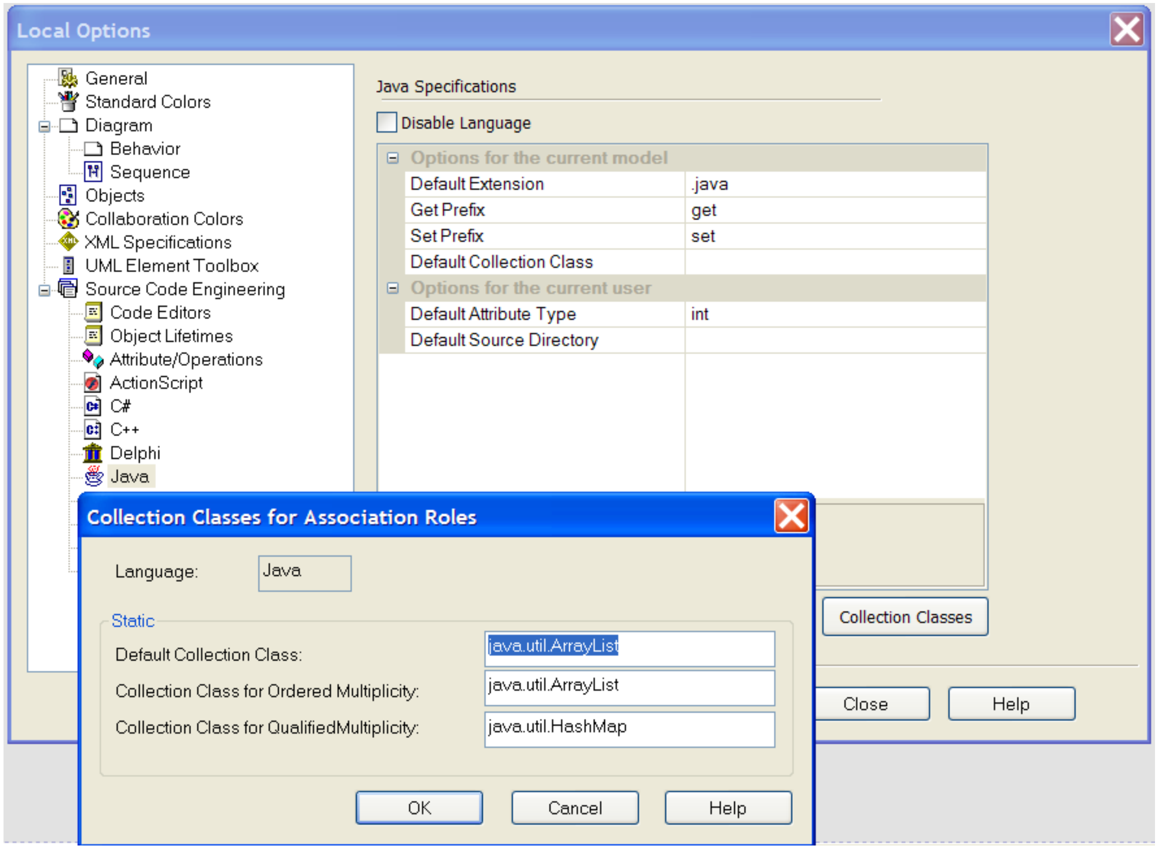
Code page for source editing:

Another option which can be of value is the automatic removal of certain attribute name prefixes when generating accessor/mutator operations.

Define target language collection mappings

Many target platforms (Java, C#, C++, Visual basic, etc.) define utility classes which can be used to hold different kinds of collections (e.g. lists, maps, bags). When properly configured in EA, these classes will be used by the transformation templates to automatically map UML association ends with a multiplicity > 1 to the appropriate collection type.

The example below shows a possible mapping for the Java target platform.



Options for the current model	
Default Extension	.java
Get Prefix	get
Set Prefix	set
Default Collection Class	
Options for the current user	
Default Attribute Type	int
Default Source Directory	

Collection Classes for Association Roles

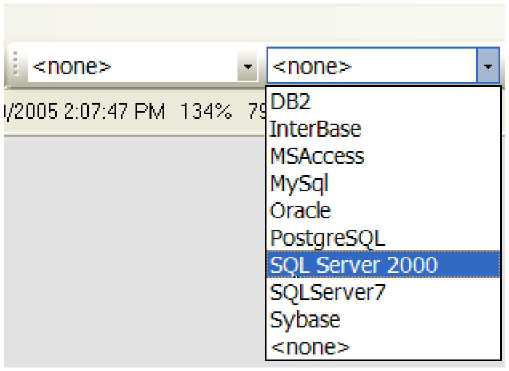
Language: Java

Static

Default Collection Class:	java.util.ArrayList
Collection Class for Ordered Multiplicity:	java.util.ArrayList
Collection Class for QualifiedMultiplicity:	java.util.HashMap

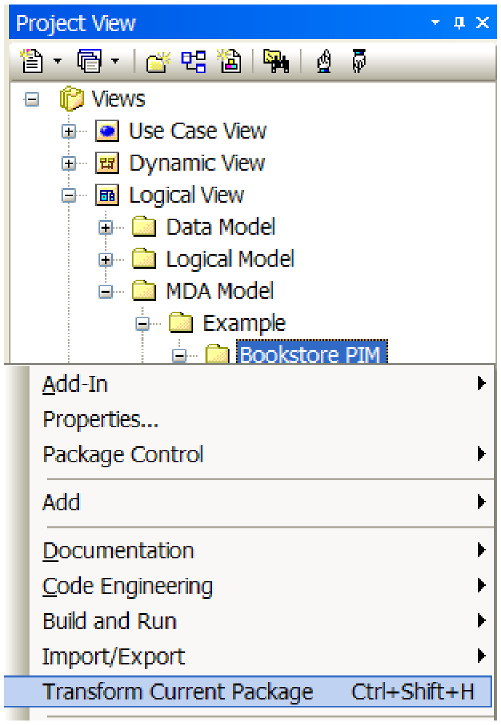
Define the default database

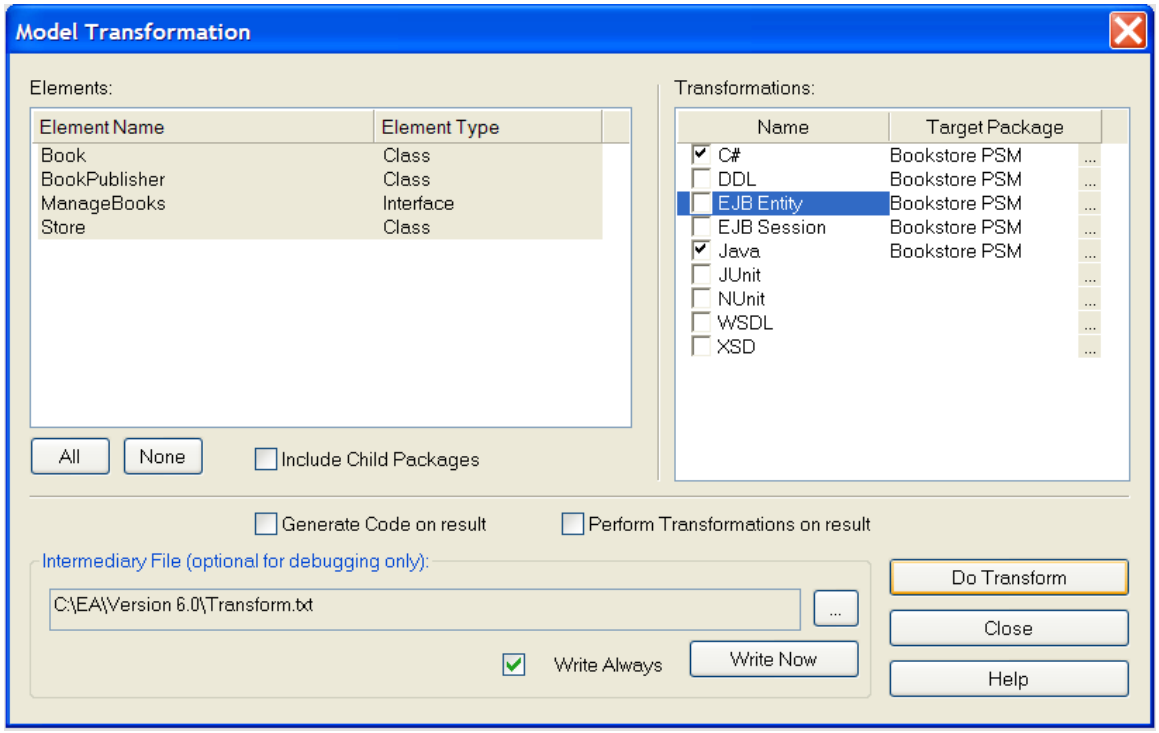
The database schema (DDL) transformation template uses the default database setting as its target environment, so it is important to initialize it properly via the toolbar or the options interface (*Tools* → *Options* → *Code Editors*):



Selecting the transformation template(s)

With the package holding the PIM classes selected in the Project View, the Model Transformation dialog can be invoked to designate the transformation targets:





Element Name	Element Type
Book	Class
BookPublisher	Class
ManageBooks	Interface
Store	Class

Name	Target Package
✓ C#	Bookstore PSM
DDL	Bookstore PSM
EJB Entity	Bookstore PSM
EJB Session	Bookstore PSM
✓ Java	Bookstore PSM
JUnit	
NUnit	
WSDL	
XSD	

☐ Include Child Packages

☐ Generate Code on result ☐ Perform Transformations on result

Intermediary File (optional for debugging only): C:\EA\Version 6.0\Transform.txt

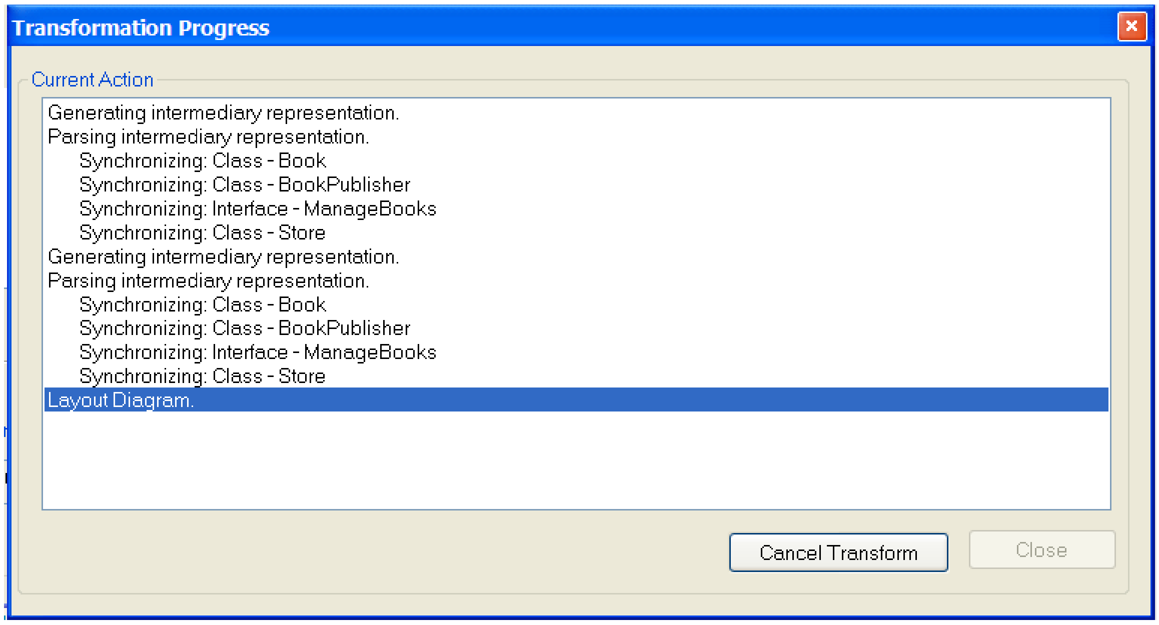
☑ Write Always

Note that elements can also individually transformed via their context menu (right click), by selecting the *Transform Selected Element(s)* option.

The major steps in using the above dialog interface are:

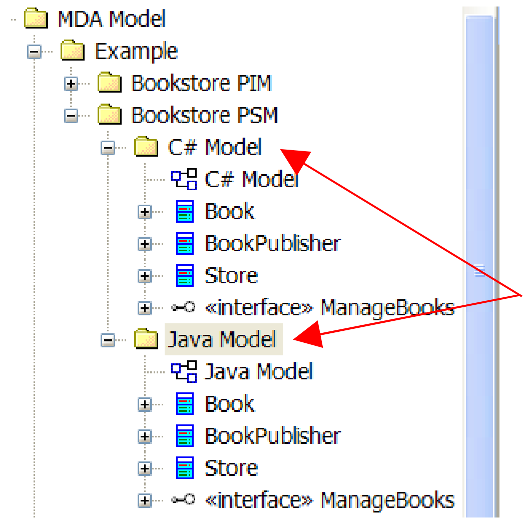
- Selecting one or more of the elements in the source package as appropriate for the intended transformation(s).
- Selecting a target destination PSM package for each transformation.
- Selecting the kind(s) of transformation. In this example Java and C# are chosen as two platforms which can be targeted from the same set of source elements.
- Setting *Generate code on result* : after the first transformation, when the target packages and namespaces have been created and the code generation target pathnames defined, subsequent transformations can use this option to automatically invoke the code generation feature for the target classes corresponding to the selected source classes.
- Setting *Perform transformations on result*: likewise, this option can be used to automatically invoke previously defined transformations on the target model, thus allowing a "chain of transformations" to occur from a single user command.
- Defining an *intermediate file* to capture the output of the transformation engine. This feature is invaluable when debugging transformation templates, or when creating custom transformations. This file holds the properties of each UML element before its actual creation in the target model.

Running the transformation(s)



```
Generating intermediary representation.
Parsing intermediary representation.
    Synchronizing: Class - Book
    Synchronizing: Class - BookPublisher
    Synchronizing: Interface - ManageBooks
    Synchronizing: Class - Store
Generating intermediary representation.
Parsing intermediary representation.
    Synchronizing: Class - Book
    Synchronizing: Class - BookPublisher
    Synchronizing: Interface - ManageBooks
    Synchronizing: Class - Store
Layout Diagram.
```

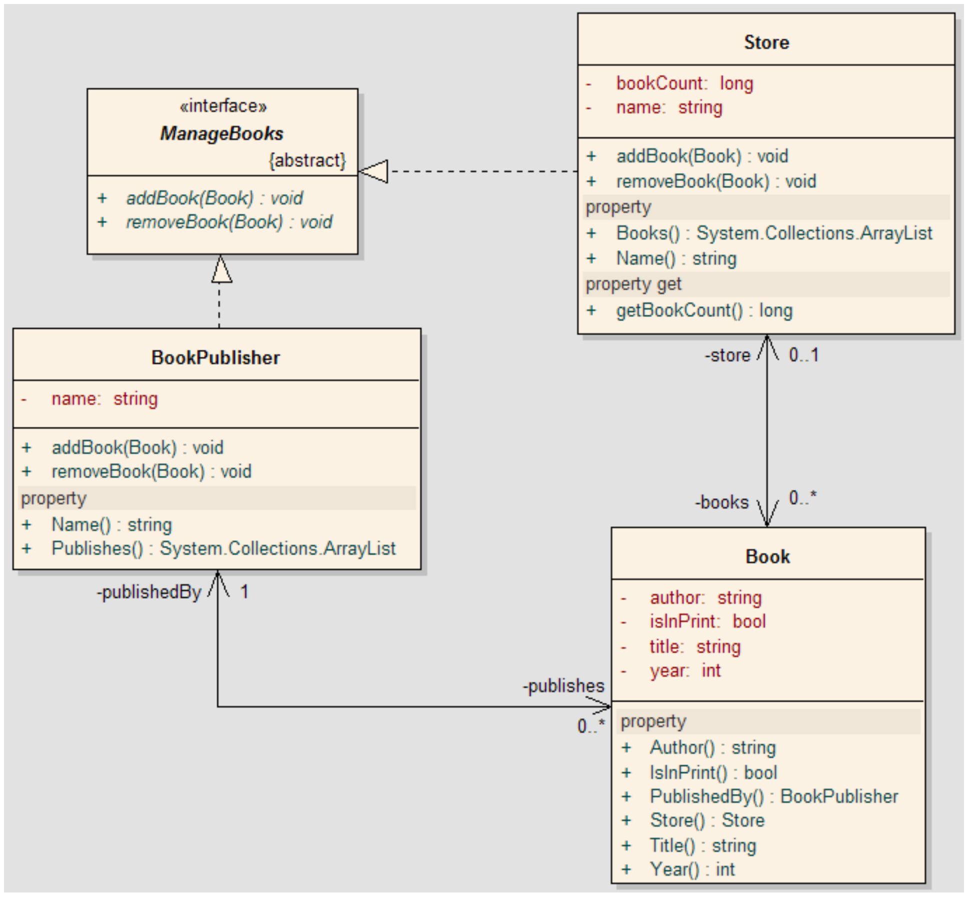
EA creates each target PSM package (if it does not exist already) with all of the generated elements under it. A class diagram showing these elements is constructed by default (or updated if created previously):


```
MDA Model
  Example
    Bookstore PIM
    Bookstore PSM
      C# Model
        C# Model
        Book
        BookPublisher
        Store
        «interface» ManageBooks
      Java Model
        Java Model
        Book
        BookPublisher
        Store
        «interface» ManageBooks
```

The package names “C# Model”, “Java Model”, and other PSM target package names are currently hard coded in each of the templates. By default EA marks these top level packages as *namespace roots*, so they are not part of the target namespace.

The target C# model

The following PSM is the result of the above transformation (diagram objects were re-arranged for clarity):

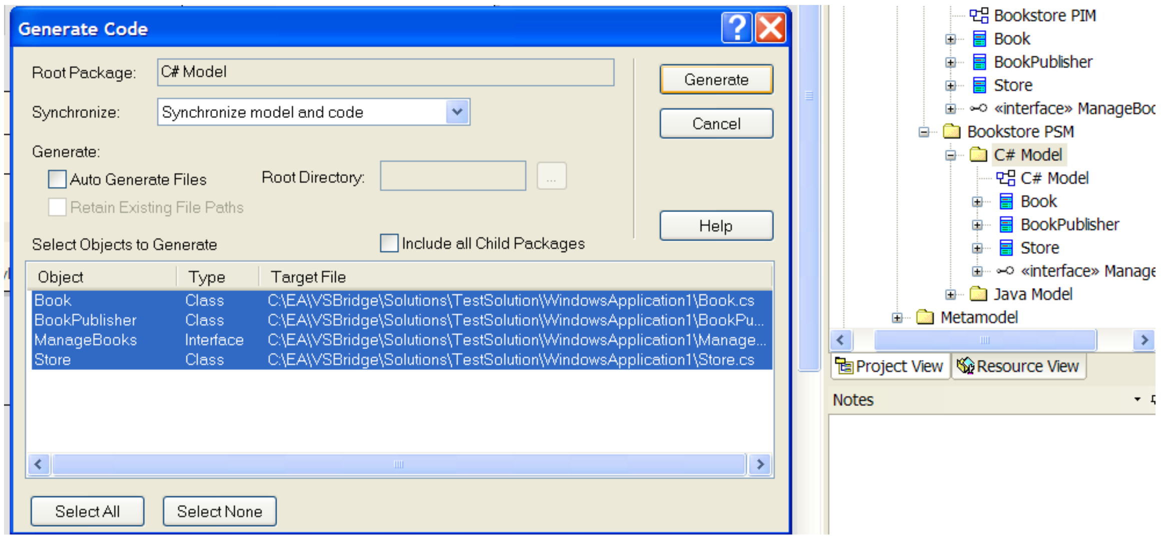


Note the following transformation results:

- Conversion of the generic data types to C# types.
- Conversion of the *public* attribute scopes back to *private*.
- Creation of the C# properties, including for the navigable UML association ends with a mapping to the appropriate collection class.

Generating the C# implementation code

The generated C# PSM can now be used as the source model for a transformation into actual implementation code, via the default code generation template of EA.



Root Package: C# Model

Synchronize: Synchronize model and code

Generate:

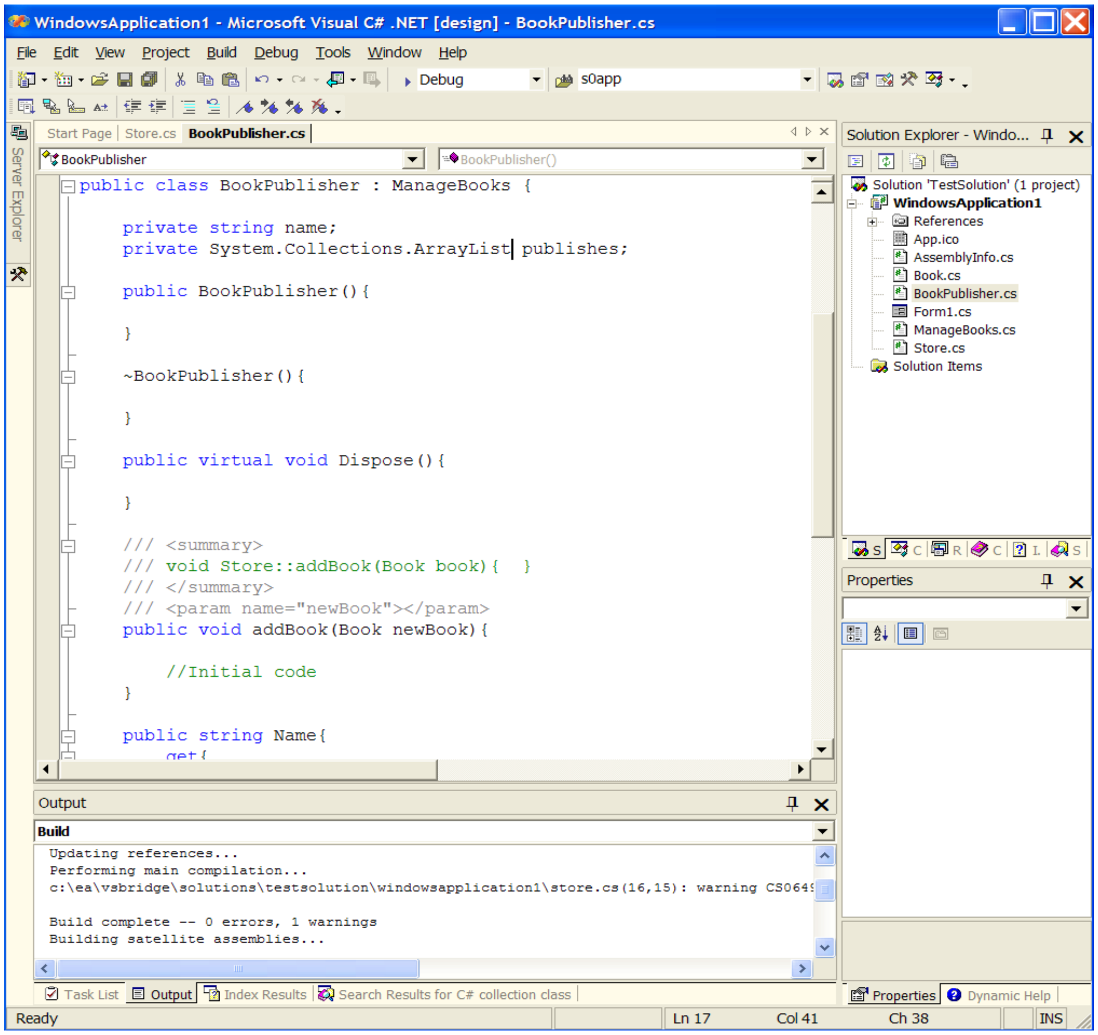
Auto Generate Files Root Directory:

Retain Existing File Paths

Select Objects to Generate Include all Child Packages

Object	Type	Target File
Book	Class	C:\EA\VSBridge\Solutions\TestSolution\WindowsApplication1\Book.cs
BookPublisher	Class	C:\EA\VSBridge\Solutions\TestSolution\WindowsApplication1\BookPu...
ManageBooks	Interface	C:\EA\VSBridge\Solutions\TestSolution\WindowsApplication1\Manage...
Store	Class	C:\EA\VSBridge\Solutions\TestSolution\WindowsApplication1\Store.cs

The resulting source code can now be further edited, compiled and built, either using the editor provided by EA, or by using an external IDE such as Visual Studio (Sparx provides an extension to EA for close integration with Visual Studio – for details please visit http://www.sparxsystems.com/products/mdg_vs.html).

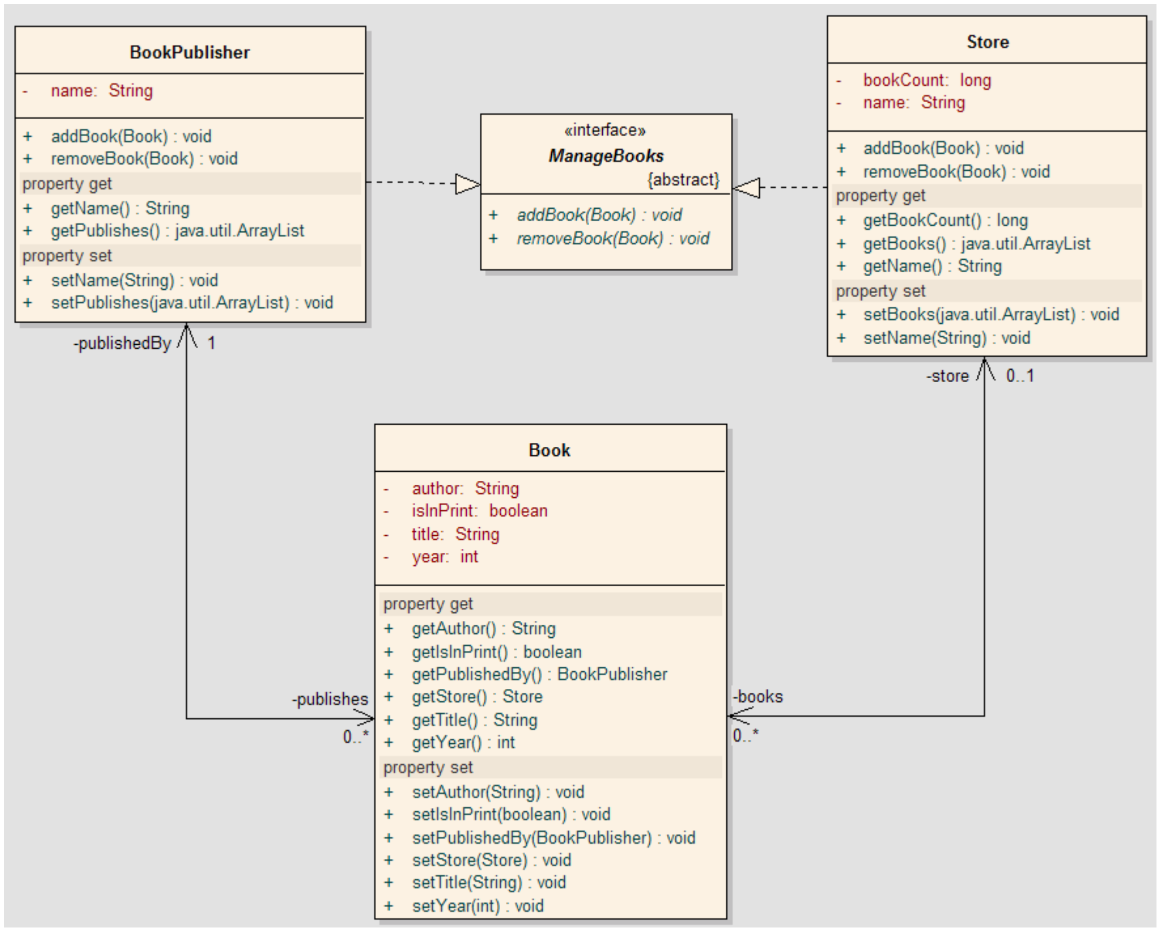


```
public class BookPublisher : ManageBooks {

    private string name;
    private System.Collections.ArrayList publishes;

    public BookPublisher(){

    }

    ~BookPublisher(){

    }

    public virtual void Dispose(){

    }

    /// <summary>
    /// void Store::addBook(Book book){   }
    /// </summary>
    /// <param name="newBook"></param>
    public void addBook(Book newBook){

        //Initial code
    }

    public string Name{
        get{
```

```
Updating references...
Performing main compilation...
c:\ea\vsbridge\solutions\testsolution\windowsapplication1\store.cs(16,15): warning CS064

Build complete -- 0 errors, 1 warnings
Building satellite assemblies...
```

The target Java model

This Java PSM is also the result of the above transformation (again diagram objects were re-arranged for clarity):

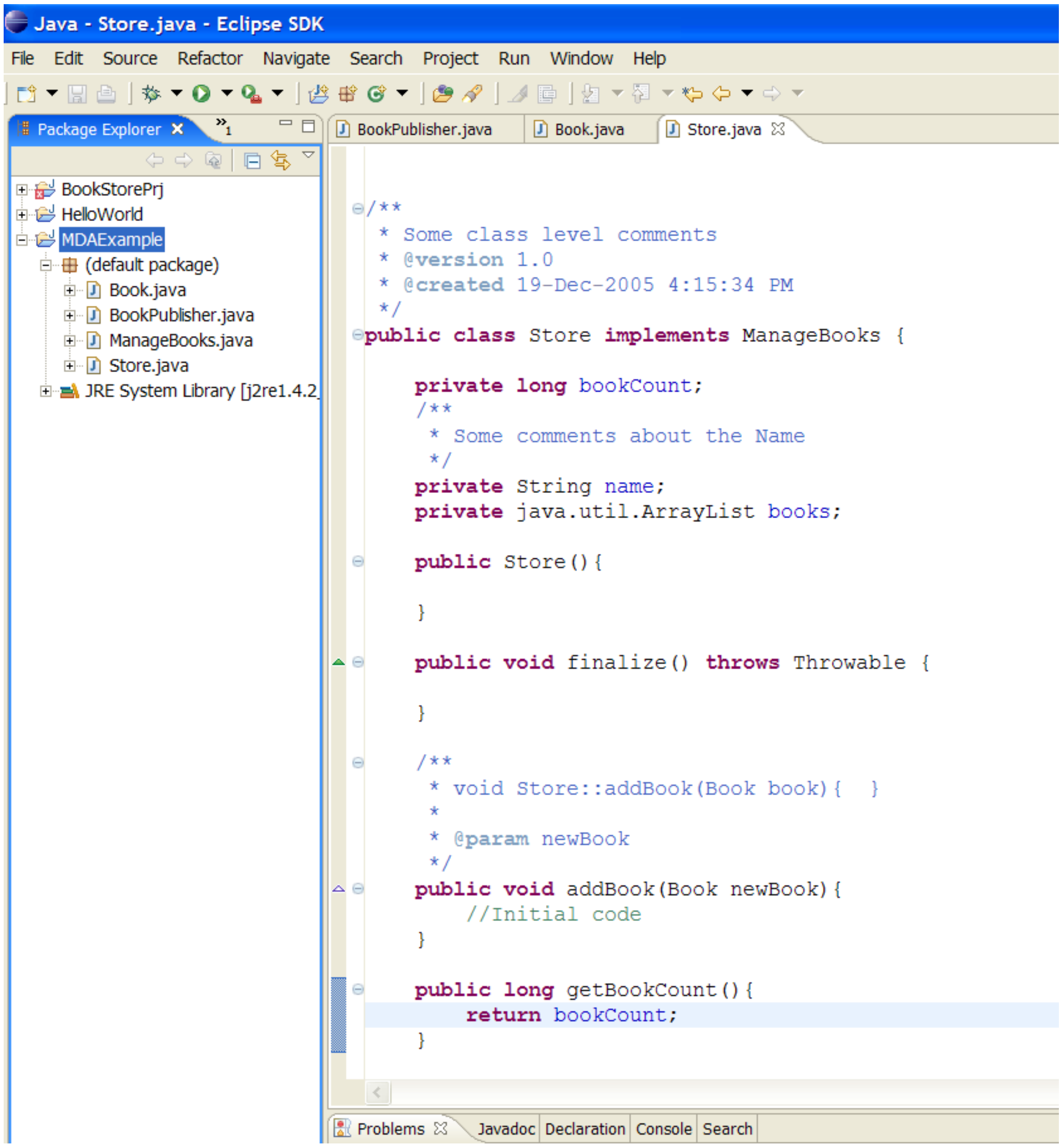


Note the following transformation results:

- Conversion of the generic data types to Java types.
- Conversion of the *public* attribute scopes back to *private*.
- Creation of the Java properties (accessors/mutators), including for the navigable UML association ends with mapping to the appropriate Java collection class.

Generating the Java implementation code

The generated PSM can now be used as the source model for a transformation into Java implementation code, via the default code generation template of EA. This code can then further be edited, compiled and built using an appropriate IDE such as Eclipse (Sparx provides another tool extension for close integration with Eclipse – for details please visit http://www.sparxsystems.com/products/mdg_eclipse.html).

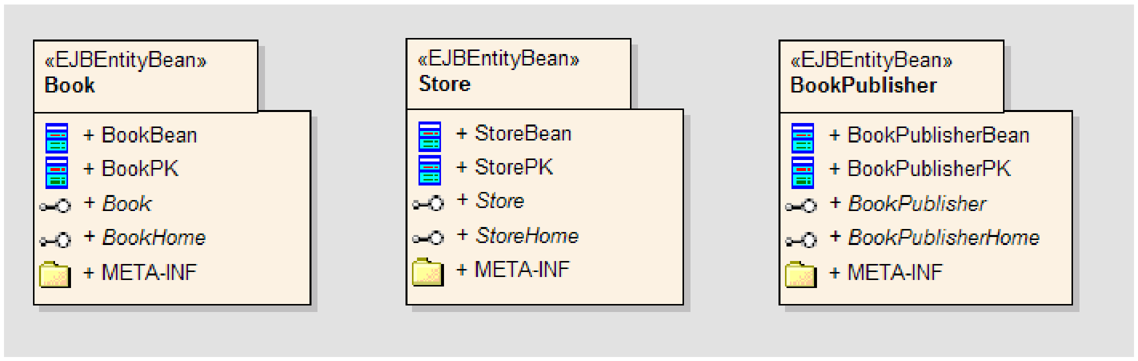


```
/**
 * Some class level comments
 * @version 1.0
 * @created 19-Dec-2005 4:15:34 PM
 */
public class Store implements ManageBooks {

    private long bookCount;
    /**
     * Some comments about the Name
     */
    private String name;
    private java.util.ArrayList books;

    public Store(){

    }

    public void finalize() throws Throwable {

    }

    /**
     * void Store::addBook(Book book){   }
     *
     * @param newBook
     */
    public void addBook(Book newBook){
        //Initial code
    }

    public long getBookCount(){
        return bookCount;
    }
```

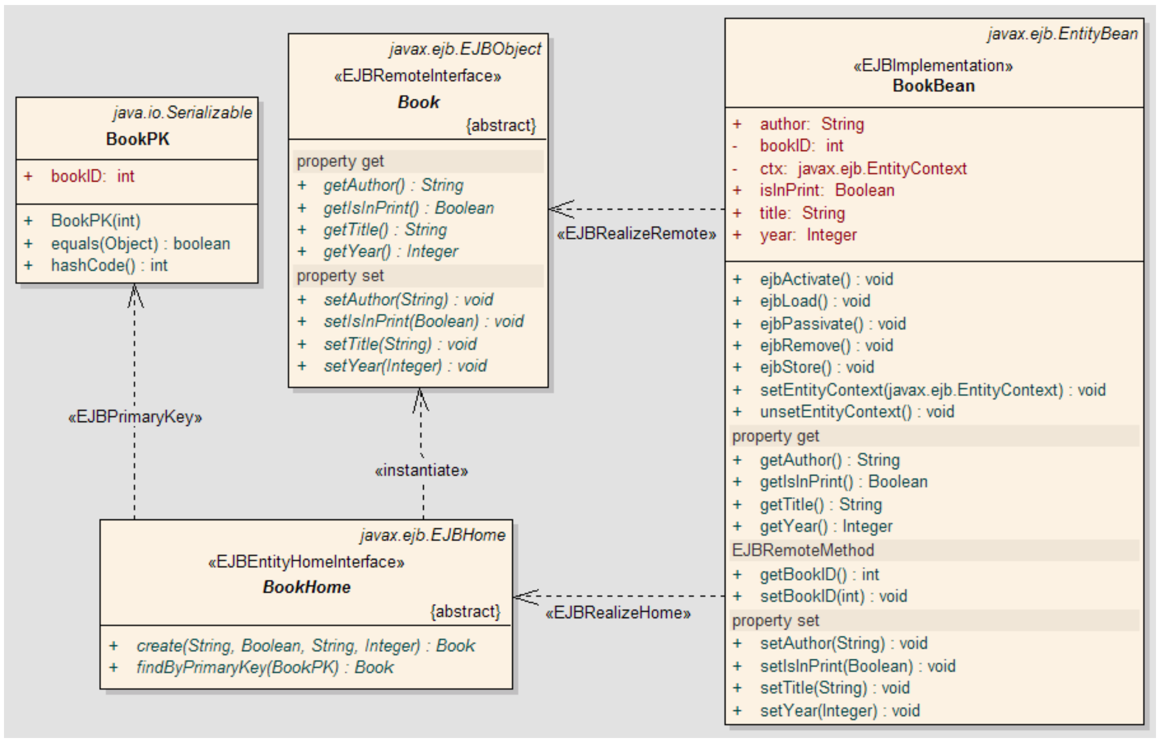
Generating an EJB entity model

Next, as part of the underlying middleware for this example, an EJB Entity PSM model is generated. In our example PIM there are no good candidate classes for generating EJB Session beans, but the output model is similar to the Entity bean PSM (without the primary key element).

The following diagram presents an overview of all the PSM model elements generated by this transformation.

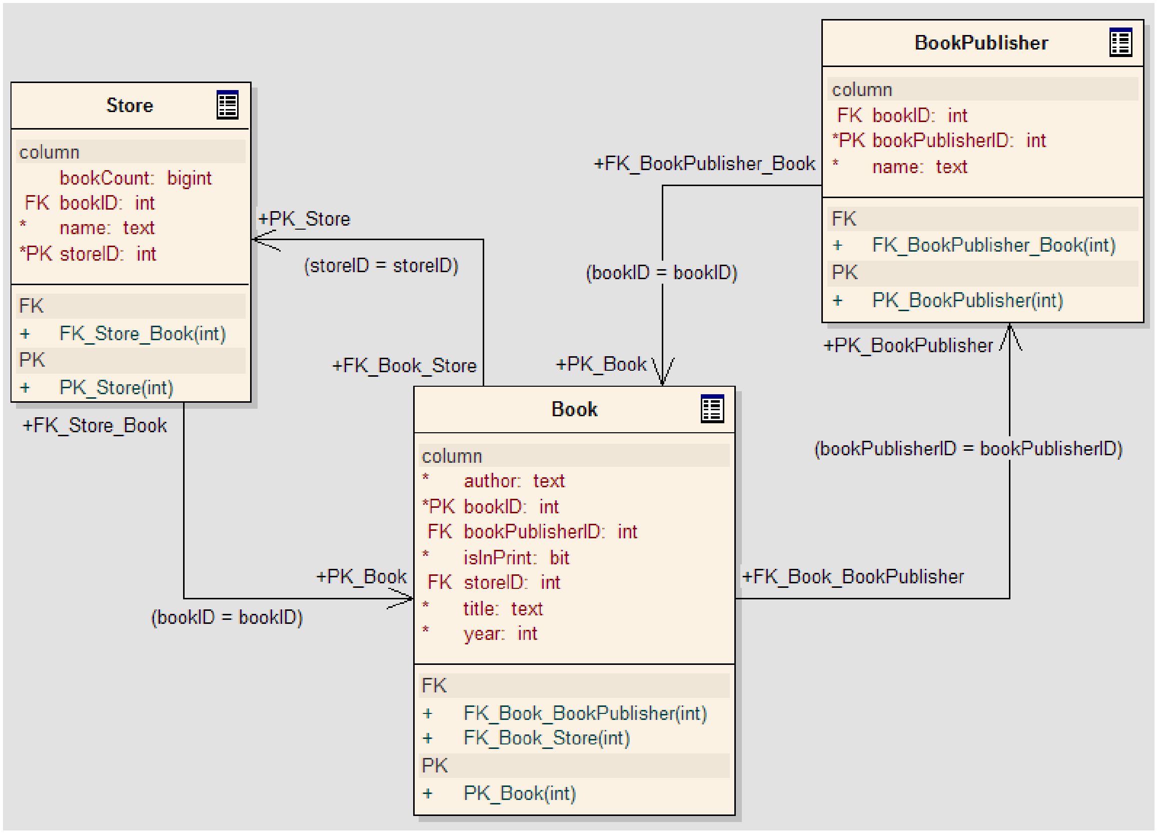


Here we view in more detail the elements created for one of the source PIM classes, in accordance with the EJB specification:



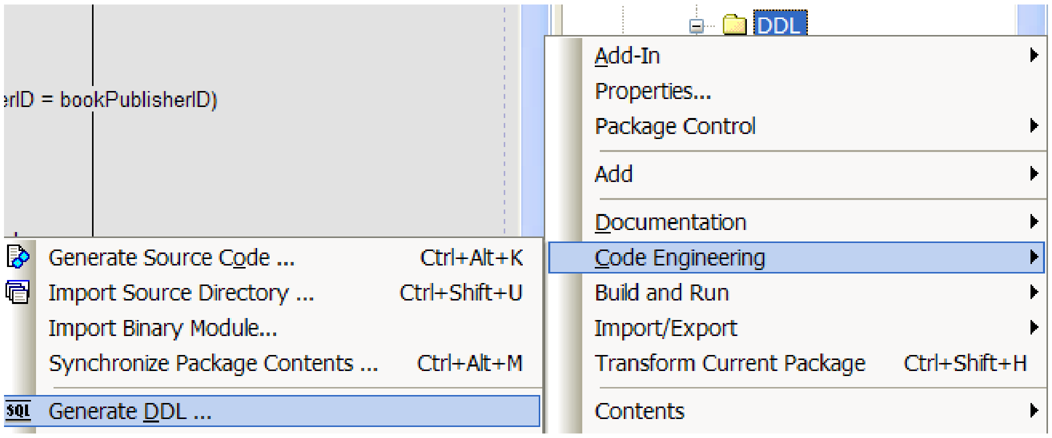
Generating a data model

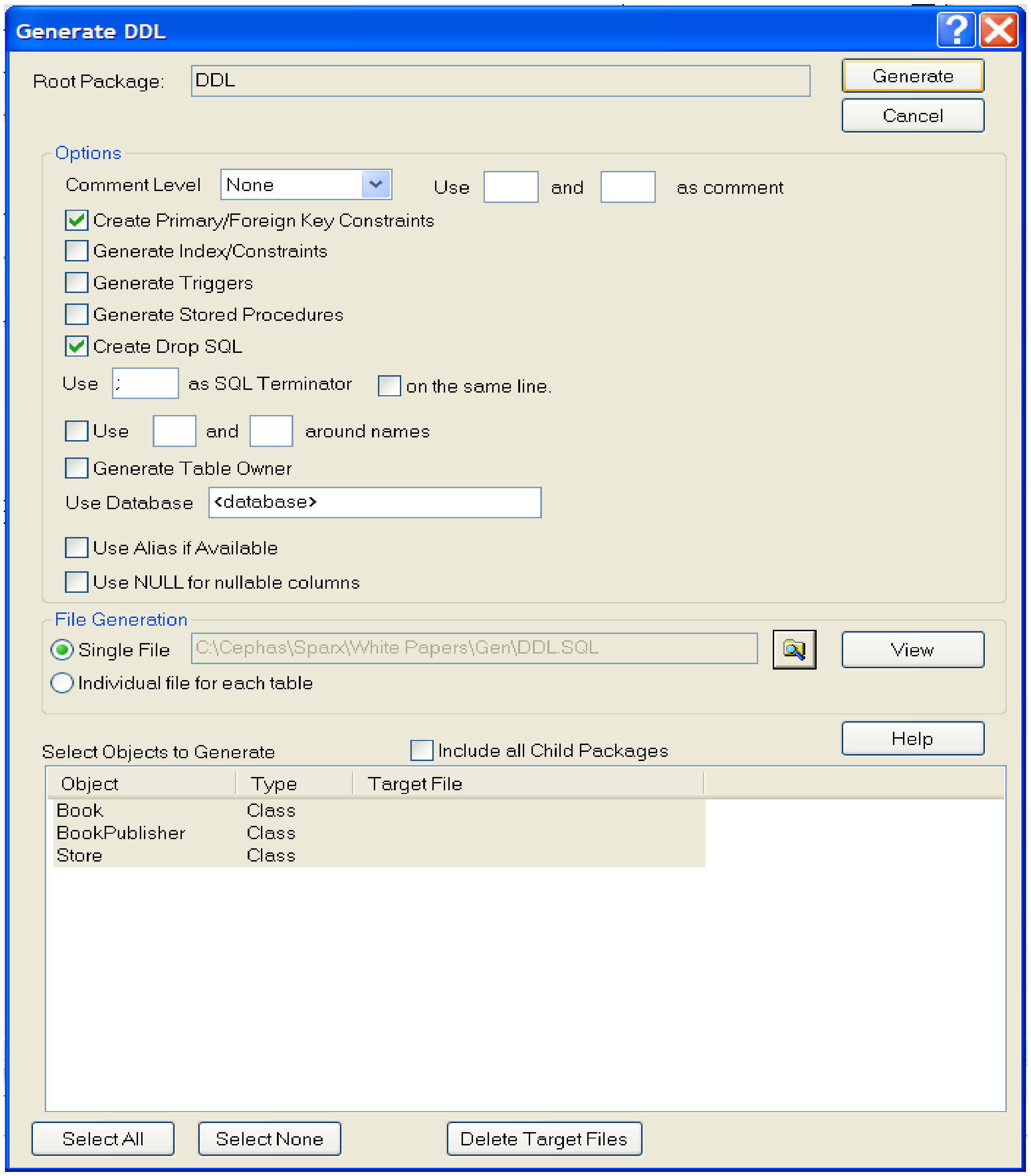
In the information viewpoint of the system one may wish to create a PSM model of the data elements defined by the PIM, as a basis for creating a database schema. In this transformation EA leverages its built-in UML profile for data modeling (for background information refer to : http://sparxsystems.com.au/resources/uml_datamodel.html).



Note the mapping of the data types - in this example the default database was set to *SQL Server 2000* – as well as the addition to the target model of both primary and foreign keys.

From that target model EA can generate the actual DDL, as show in this sample:





Root Package: DDL

Options

Comment Level: None Use [ ] and [ ] as comment

- [✓] Create Primary/Foreign Key Constraints
- [ ] Generate Index/Constraints
- [ ] Generate Triggers
- [ ] Generate Stored Procedures
- [✓] Create Drop SQL

Use ; as SQL Terminator [ ] on the same line.

[ ] Use [ ] and [ ] around names

[ ] Generate Table Owner

Use Database: <database>

- [ ] Use Alias if Available
- [ ] Use NULL for nullable columns

File Generation

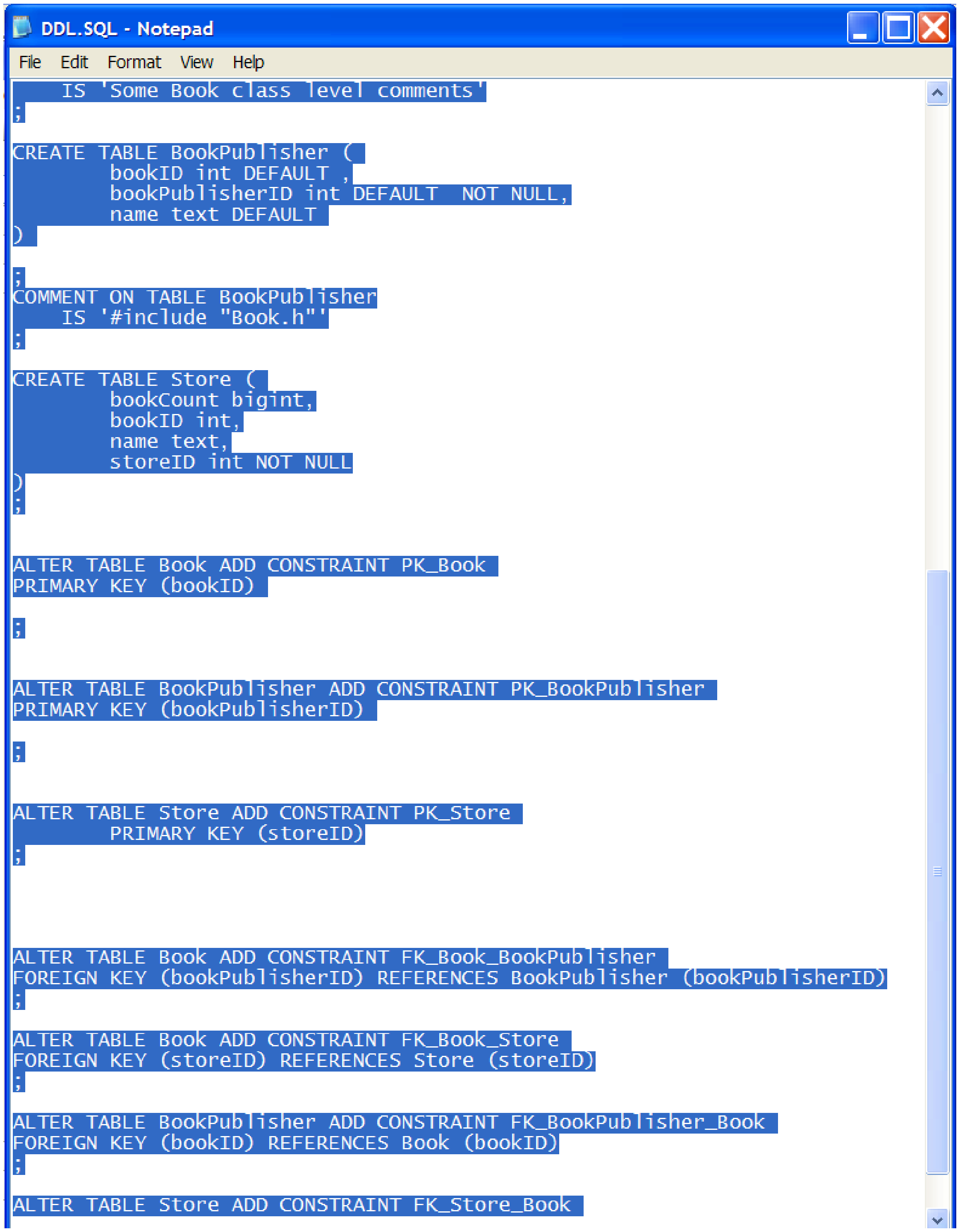
[● Single File] C:\Cephas\Sparx\White Papers\Gen\DDL.SQL

○ Individual file for each table

Select Objects to Generate [ ] Include all Child Packages

Object	Type	Target File
Book	Class	
BookPublisher	Class	
Store	Class	

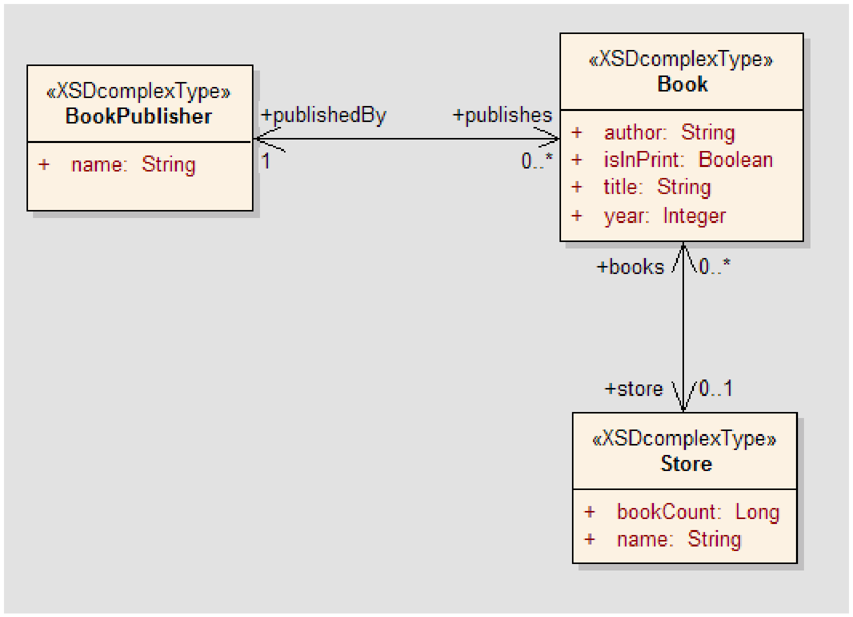
What follows is a snapshot of the resulting DDL script.



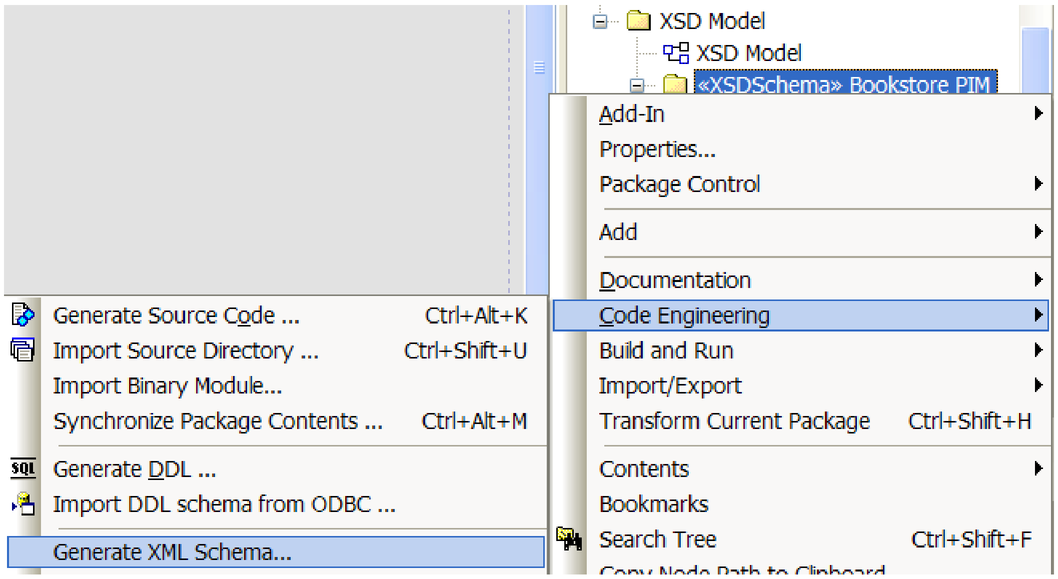
```
    IS 'Some Book class level comments'
;

CREATE TABLE BookPublisher (
        bookID int DEFAULT ,
        bookPublisherID int DEFAULT  NOT NULL,
        name text DEFAULT
)

;
COMMENT ON TABLE BookPublisher
    IS '#include "Book.h"'
;

CREATE TABLE Store (
        bookCount bigint,
        bookID int,
        name text,
        storeID int NOT NULL
)
;


ALTER TABLE Book ADD CONSTRAINT PK_Book
PRIMARY KEY (bookID)

;


ALTER TABLE BookPublisher ADD CONSTRAINT PK_BookPublisher
PRIMARY KEY (bookPublisherID)

;


ALTER TABLE Store ADD CONSTRAINT PK_Store
        PRIMARY KEY (storeID)
;




ALTER TABLE Book ADD CONSTRAINT FK_Book_BookPublisher
FOREIGN KEY (bookPublisherID) REFERENCES BookPublisher (bookPublisherID)
;

ALTER TABLE Book ADD CONSTRAINT FK_Book_Store
FOREIGN KEY (storeID) REFERENCES Store (storeID)
;

ALTER TABLE BookPublisher ADD CONSTRAINT FK_BookPublisher_Book
FOREIGN KEY (bookID) REFERENCES Book (bookID)
;

ALTER TABLE Store ADD CONSTRAINT FK_Store_Book
```

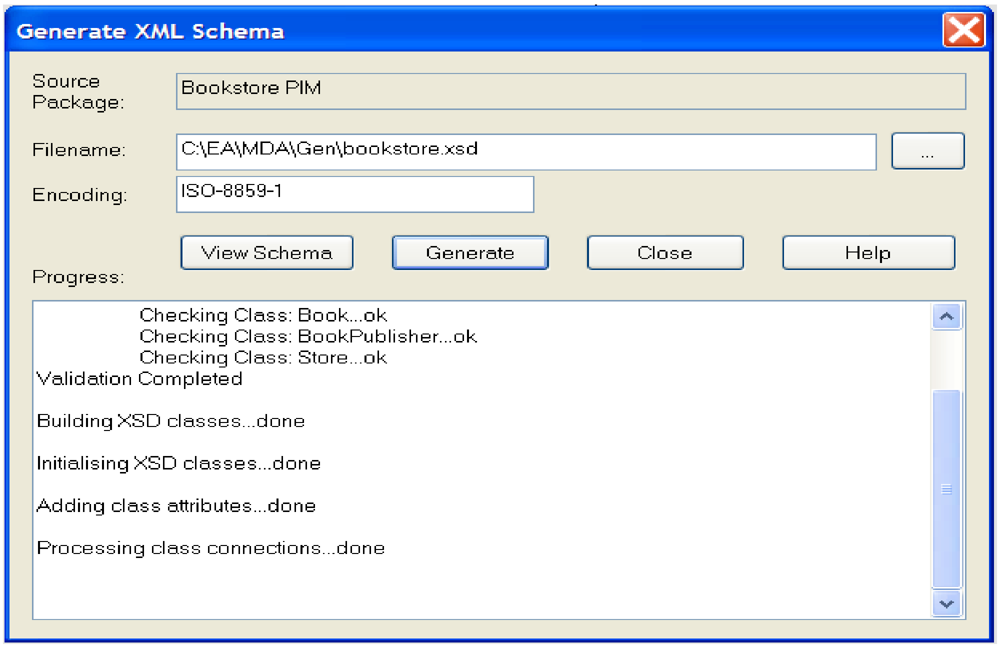
Generating an XML schema model

Yet another powerful transformation, especially in the context of system integrations, is a PSM model containing the XML Schema representation of the PIM data elements:



The current transformation rules simply mark all classes (other than enumerations) as *XSDComplexType.* Also note that mapping of the data types does not occur at this stage, but rather during the generation of the XML schema itself.

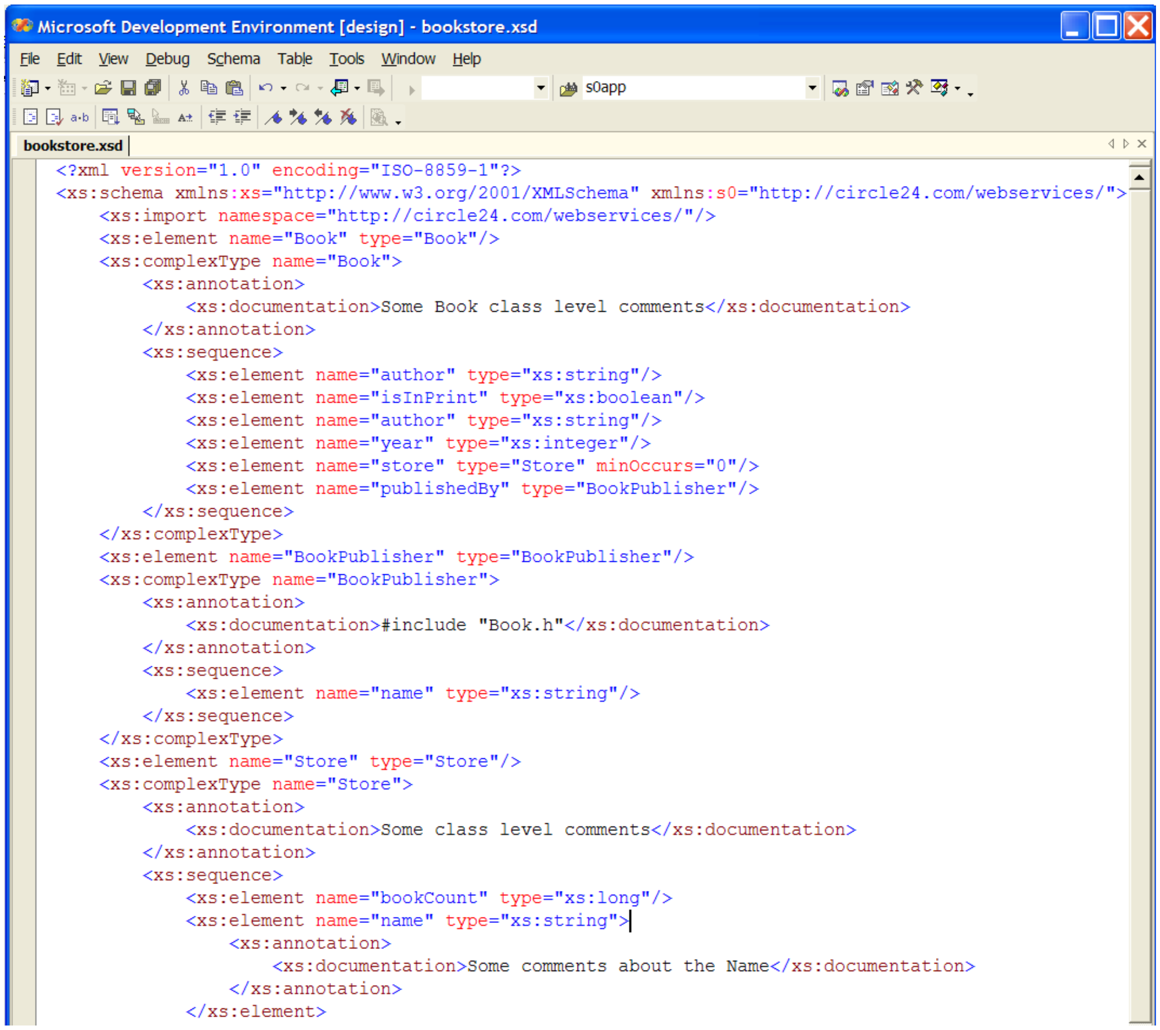

XSD Model
XSD Model
«XSDSchema» Bookstore PIM
Add-In
Properties...
Package Control
Add
Documentation
Code Engineering
Build and Run
Import/Export
Transform Current Package Ctrl+Shift+H
Contents
Bookmarks
Search Tree Ctrl+Shift+F
Generate Source Code ... Ctrl+Alt+K
Import Source Directory ... Ctrl+Shift+U
Import Binary Module...
Synchronize Package Contents ... Ctrl+Alt+M
Generate DDL ...
Import DDL schema from ODBC ...
Generate XML Schema...



Source Package: Bookstore PIM
Filename: C:\EA\MDA\Gen\bookstore.xsd
Encoding: ISO-8859-1
Progress:

```
        Checking Class: Book...ok
        Checking Class: BookPublisher...ok
        Checking Class: Store...ok
Validation Completed

Building XSD classes...done

Initialising XSD classes...done

Adding class attributes...done

Processing class connections...done
```

Which produces the actual XML Schema file:

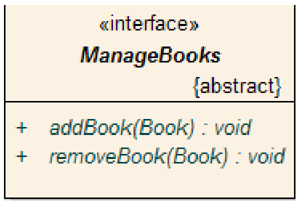

Microsoft Development Environment [design] - bookstore.xsd
File Edit View Debug Schema Table Tools Window Help
s0app
bookstore.xsd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://circle24.com/webservices/">
    <xs:import namespace="http://circle24.com/webservices/"/>
    <xs:element name="Book" type="Book"/>
    <xs:complexType name="Book">
        <xs:annotation>
            <xs:documentation>Some Book class level comments</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="author" type="xs:string"/>
            <xs:element name="isInPrint" type="xs:boolean"/>
            <xs:element name="author" type="xs:string"/>
            <xs:element name="year" type="xs:integer"/>
            <xs:element name="store" type="Store" minOccurs="0"/>
            <xs:element name="publishedBy" type="BookPublisher"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="BookPublisher" type="BookPublisher"/>
    <xs:complexType name="BookPublisher">
        <xs:annotation>
            <xs:documentation>#include "Book.h"</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Store" type="Store"/>
    <xs:complexType name="Store">
        <xs:annotation>
            <xs:documentation>Some class level comments</xs:documentation>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="bookCount" type="xs:long"/>
            <xs:element name="name" type="xs:string">
                <xs:annotation>
                    <xs:documentation>Some comments about the Name</xs:documentation>
                </xs:annotation>
            </xs:element>
```

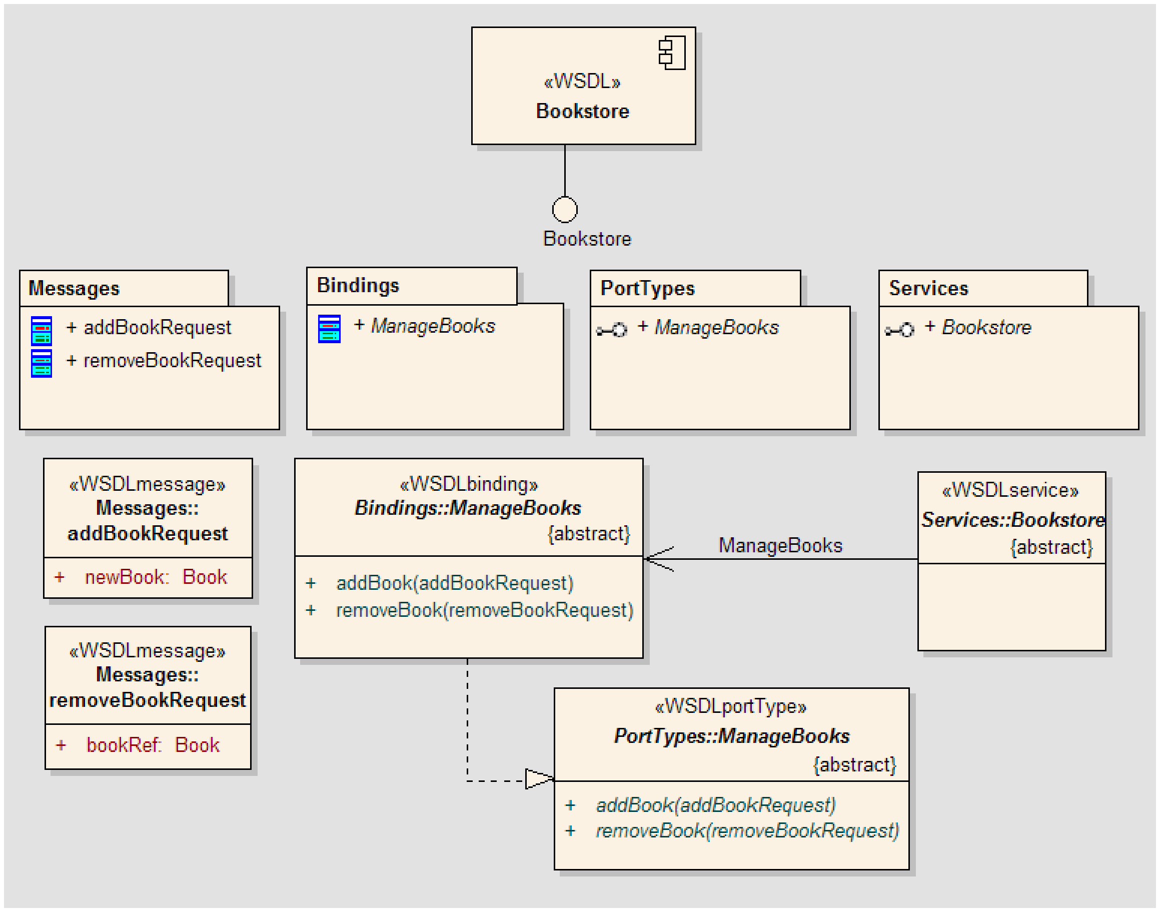
Generating a WSDL model

The transformation into WSDL provides designers and architects with a mechanism for quickly transforming abstract interfaces into a complete Web Service specification.

The starting point is typically an *interface* class such as *ManageBooks* in our example.



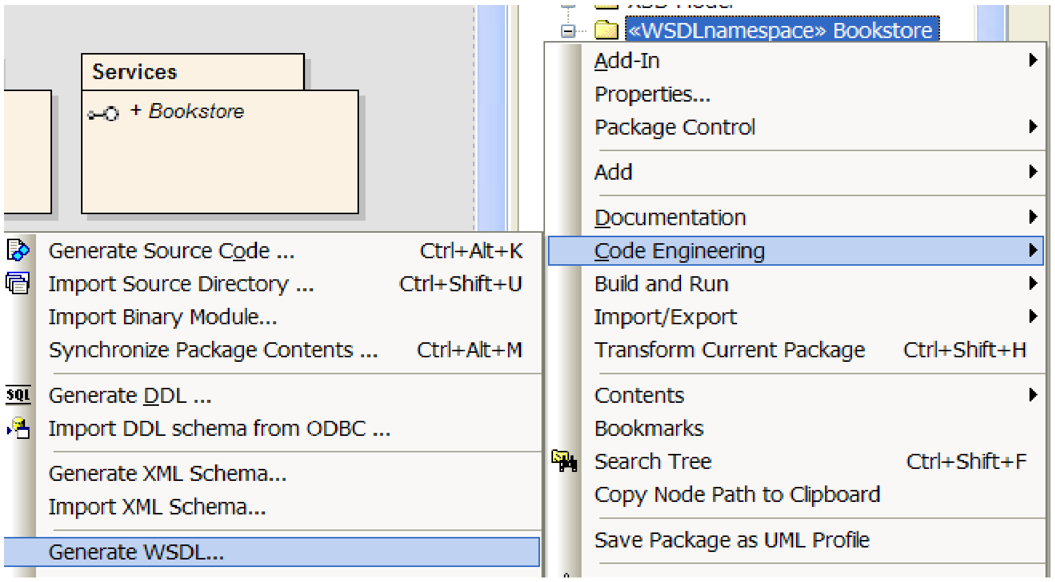
The following diagram illustrates the artifacts produced:

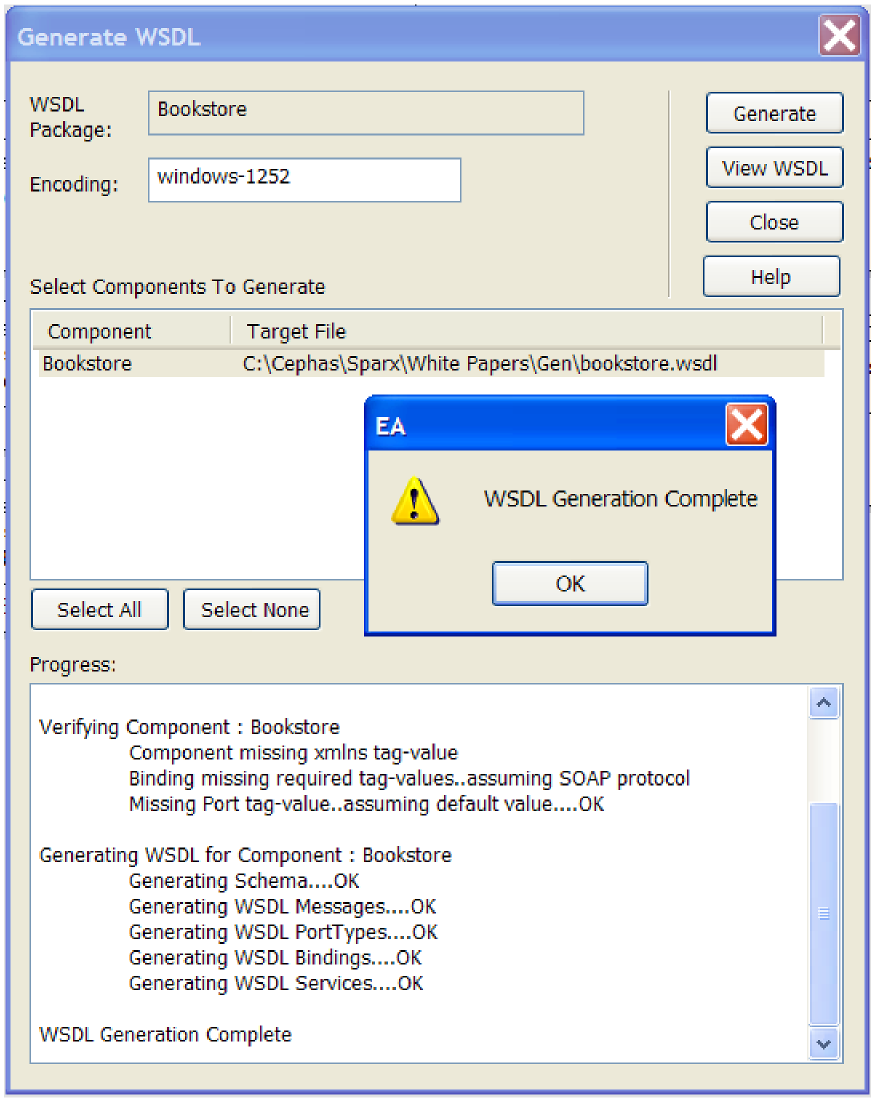


Note the transformation results in terms of :

- The operation input arguments mapped into WSDL *request messages* (there are no output or result arguments in this example).
- The operations themselves mapped into a WSDL *port type* and its corresponding *binding* using the *request messages*.
- The creation of an abstract *service* representing the interface.

From the root package, the actual WSDL file can now be generated:


Services
+ Bookstore
Generate Source Code ... Ctrl+Alt+K
Import Source Directory ... Ctrl+Shift+U
Import Binary Module...
Synchronize Package Contents ... Ctrl+Alt+M
Generate DDL ...
Import DDL schema from ODBC ...
Generate XML Schema...
Import XML Schema...
Generate WSDL...
«WSDLnamespace» Bookstore
Add-In
Properties...
Package Control
Add
Documentation
Code Engineering
Build and Run
Import/Export
Transform Current Package Ctrl+Shift+H
Contents
Bookmarks
Search Tree Ctrl+Shift+F
Copy Node Path to Clipboard
Save Package as UML Profile



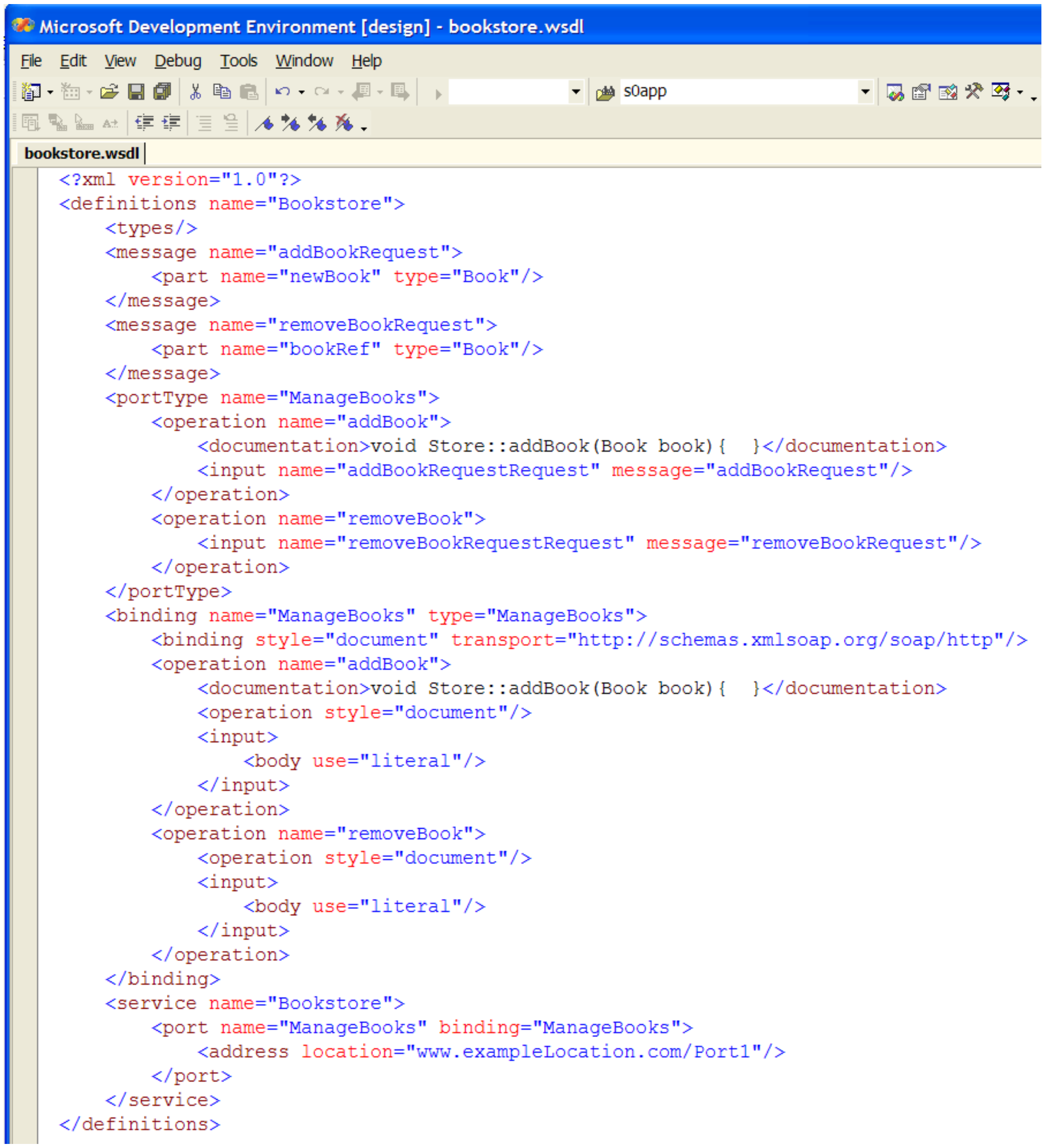
WSDL Package: Bookstore
Encoding: windows-1252

Component	Target File
Bookstore	C:\Cephas\Sparx\White Papers\Gen\bookstore.wsdl

Progress:

```
Verifying Component : Bookstore
          Component missing xmlns tag-value
          Binding missing required tag-values..assuming SOAP protocol
          Missing Port tag-value..assuming default value....OK

Generating WSDL for Component : Bookstore
          Generating Schema....OK
          Generating WSDL Messages....OK
          Generating WSDL PortTypes....OK
          Generating WSDL Bindings....OK
          Generating WSDL Services....OK

WSDL Generation Complete
```

… and the resulting file edited for validation and further processing:

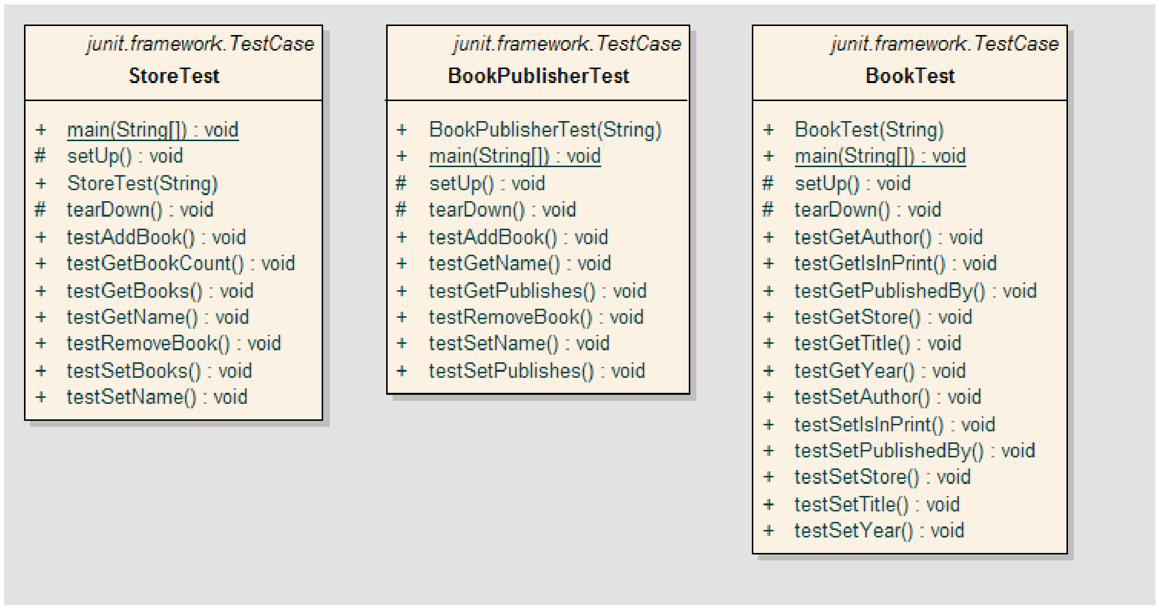

Microsoft Development Environment [design] - bookstore.wsdl

bookstore.wsdl

```
<?xml version="1.0"?>
<definitions name="Bookstore">
    <types/>
    <message name="addBookRequest">
        <part name="newBook" type="Book"/>
    </message>
    <message name="removeBookRequest">
        <part name="bookRef" type="Book"/>
    </message>
    <portType name="ManageBooks">
        <operation name="addBook">
            <documentation>void Store::addBook(Book book){  }</documentation>
            <input name="addBookRequestRequest" message="addBookRequest"/>
        </operation>
        <operation name="removeBook">
            <input name="removeBookRequestRequest" message="removeBookRequest"/>
        </operation>
    </portType>
    <binding name="ManageBooks" type="ManageBooks">
        <binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="addBook">
            <documentation>void Store::addBook(Book book){  }</documentation>
            <operation style="document"/>
            <input>
                <body use="literal"/>
            </input>
        </operation>
        <operation name="removeBook">
            <operation style="document"/>
            <input>
                <body use="literal"/>
            </input>
        </operation>
    </binding>
    <service name="Bookstore">
        <port name="ManageBooks" binding="ManageBooks">
            <address location="www.exampleLocation.com/Port1"/>
        </port>
    </service>
</definitions>
```

Generating a Unit Test model

So far all of the above transformations have started from a platform independent source model to produce a platform specific one. Next we see an example where a PSM takes on the role of a PIM with regards to a further downstream transformation. In particular we take the Java model we created earlier and run it through the MDA transformer to obtain a JUnit test model, effectively setting up a chain of transformations.



The test code skeletons can now be generated from the resulting classes, edited in an appropriate IDE to add the test logic, and executed after loading them into the JUnit framework.

A similar transformation capability is available to generate NUnit test classes from C# source models.

Additional information

For a summary of MDA resources and MDA style transformations in EA see:

http://sparxsystems.com.au/resources/mda/index.html

For an overview of writing transformations see:

http://sparxsystems.com.au/resources/mda/writing_transformations.html

About SPARX Systems

Company Background

Established in 1996 by Geoffrey Sparks, Sparx Systems is an Australian company based at Creswick, near Ballarat, Victoria. With over seven years' investment in the development of Enterprise Architect, the company's motivated team of employees are dedicated to the ongoing development and support of software tools, object-oriented methodologies and CASE tools.

Company Vision

Sparx Systems aims to satisfy the growing needs of the software and business development industry by providing immediate delivery and ongoing support of affordable, productive and user-friendly business/system design software.

Sparx Systems believes that a complete modeling and design tool should be used throughout the full process/software lifecycle. Our subscription plan reflects this, and our belief that "life-cycle" software should be as dynamic and modern as the systems you design and maintain.

Sparx software is intended for use by analysts, designers, architects, developers, testers, project managers and maintenance staff - almost everyone involved in a software development project and in business analysis. It is Sparx Systems' belief that highly priced CASE tools severely limit their usefulness in a team, and ultimately to an organization, by narrowing the effective user base and restricting easy access to the model and the development tool. To this end, Sparx Systems are committed to both maintaining an accessible pricing model and to distributing a 'Read Only' (EA *Lite*) version of EA for use by those who only need to view modeling information.

User Base

Sparx software is utilized by a wide variety of companies ranging from large, well-known, multinational organizations to many smaller independent companies and consultants. The Sparx discussion forum confirms a solid and active user base.

Sparx software is used for the development of various kinds of software systems for a wide range of industries, including: aerospace, banking, web development, engineering, finance, medicine, military, research, academia, transport, retail, utilities (gas, electricity etc.), electrical engineering and many more. It is also used effectively for UML and business architecture training purposes in many prominent colleges, education facilities and universities around the world.

Contact Details

Website : http://www.sparxsystems.com

Sparx Systems can be contacted at the following email addresses:

General Enquiries: sparks@sparxsystems.com.au

Support Enquiries: support@sparxsystems.com.au

Sales and Purchase Enquiries: **sales@sparxsystems.com.au**

About Cephas Consulting Corp.

Company Background

Since 2001, Cephas Consulting Corp. has been active helping its corporate clients introduce state of the art information technologies. We offer expertise in the areas of:

- Modeling business applications using object oriented techniques.
- Building distributed component infrastructures.
- Introducing formal software development processes.
- Migrating development organizations into Model Driven Architecture (MDA).
- Providing advanced UML/MDA training and mentoring.

Company Focus

Cephas specializes in introducing UML™ modeling into organizations via training and mentoring techniques, using *Enterprise Architect* as the primary tool for capturing both business and system domain information.

The team of software engineers and architects at Cephas combines many years of experience, allowing it to offer a one-stop solution addressing all aspects of managing the enterprise meta-data using *Enterprise Architect*:

- Training & mentoring from beginner to expert level.
- Migrating meta-data out of legacy tools.
- Establishing onsite guardianship of the tool, including configuration and replication management.
- Customizing the tool in order to respond to unique client requirements.
- Providing expert level support and maintenance.

Commitment to the OMG and MDA

Cephas Consulting has the required expertise to lead organizations into the use of Model Driven Architecture. As early adopters we have successfully implemented MDA and are pleased to be among the first participants to the MDA *FastStart* program put in place by the OMG. We are also thrilled to work as OMG members on expanding the mind share of MDA in the marketplace, because we believe it is ideally suited to deal with the challenges of managing complex software development in times of rapid technology obsolescence.

Our highest commitment is in achieving success through quality, and we take pride in the accomplishments of our clients.

Contact Details

Website : http://www.cephas.cc

Cephas Consulting can be contacted at the following email addresses:

General enquiries: cephas.contact@cephas.cc

Author enquiries: frank.truyen@cephas.cc