

XSL Transformations for O-R Code Generation

Abstract:

Numerous organizations have tackled the problem of Object-Relational mapping. Indeed virtually all enterprise software systems interact with databases in some manner. Commercial products can often release the enterprise programmer from their duties writing (and maintaining) the necessary Object mapping API based on Java, SQL, blood, sweat and tears. These products have achieved varying levels of success. This paper presents a novel approach to this problem based on standard eXtensible Style Sheet Transformation (XSLT) technology. Though specifically focused on the problem of O-R mapping, it will become apparent that this approach to source code generation can be applied to a wide range of tasks facing Java programmers today.

It is assumed that the reader is familiar with the Java programming language, enterprise database systems, XML, and XSL. Knowledge of the general O-R mapping problem is helpful.

Paul Bemowski
J Enterprise Technologies, Inc.
<http://www.jetools.com>



Table of Contents

THE O-R MAPPING PROBLEM.....	3
Database Schema Classification	4
Available Solutions and their Merits	5
Hand Coding.....	5
O-R Mapping Tools	6
Code Generation.....	6
Comparisons	7
AN XSLT BASED APPROACH TO CODE GENERATION FOR O-R MAPPING... 7	7
XML and XSL	7
DB -> XML -> .java.....	8
A SIMPLE EXAMPLE – A BOOKSTORE APPLICATION	9
The Problem:.....	9
Analysis and Design:.....	9
A Three Step Iterative Process	11
Comments on the Example.....	13
OBSERVATIONS ON XSLT CODE GENERATION	13
Extended Mapping.....	13
Persistence Architecture	13
XSLT as a General Approach to Repetitive Software Development.....	14
APPENDIX A: REFERENCES.....	16
Databases.....	16
Object-Relational Mapping.....	16
O-R Mapping Products	16
XML and XSL	16
APPENDIX B: TYPEMAP.PROPERTIES (ORACLE SAMPLE).....	16
APPENDIX C: DBTOXML OUTPUT OF THE SAMPLE DATABASE SCHEMA. .	17
APPENDIX D: DATA TRANSFER OBJECT XSL STYLESHEET.....	18
APPENDIX E: FACTORY XSL STYLESHEET.....	20
APPENDIX F: ANT TASKS FOR GENERATION AND TRANSFORMATION.	24

As of early 2003, Sun Microsystems has estimated the number of Java developers worldwide at approximately 3 million.

One of the most repetitive tasks facing enterprise Java programmers in today is the development of data access layers sitting atop a relational database. Addressing this need to varying degrees of success are various Object-Relational (O-R) mapping tools, source code generators, integrated development environments, and application servers. Each provides a unique set of features providing trade offs between scalability, speed, functionality, configurability, etc. Fundamentally, however, each is used 'in lieu of' custom code written and optimized for the task at hand.

The O-R Mapping Problem

Object oriented programming is a way of thinking about components of computer software in terms of real world objects. OO designers discuss a piece of software in terms of the attributes and behaviors of a Person or a Game or an Account, rather than in terms of loops and algorithms and subroutines. This paradigm has proved extremely powerful for the development of many types of software systems.

The Relational Data Base Management System (RDBMS) has long been the preferred method of storing large volumes of related information. An RDBMS uses a number of two-dimensional tables to store data. Often these tables contain data that are related. Generally, relationships are of the form one to one, one to many, or many to many. Most if not all enterprise software developers will have an intimate familiarity with at least one database management system.

Access to an RDBMS is provided via the Structured Query Language or SQL. Most databases support some level of standard SQL, usually with a number of 'extensions' to standardized SQL. The most popular database as of this writing is Oracle, however other commercial (DB2, Informix, SQL Server) and freeware (PostgreSQL, MySQL) databases are also available.

For some time it was thought that the ultimate solution to the O-R problem would eventually be the migration away from relational database systems to object database systems. A programmer using an OO language would simply query an object database for necessary objects, modify them and send the objects back to the database. For better or worse, this has not happened, and current indications are that it never will.

The disconnect between the object and relational worlds is truly the definition of the Object-Relational mapping problem. In order to maintain the purity of objects defined in the problem domain a separate application layer is necessary to isolate

the problem domain from the plumbing required to read objects from and write them to a relational database. A task that is far more procedural than object oriented.

For the purposes of this paper, we are going to explicitly define the Object-Relational mapping problem as follows:

Object-Relational mapping is the bi-directional translation of data from a relational database into an object graph, and back.

Database Schema Classification

Database schemas come in many varieties. The layout of a schema depends on many factors: maturity, normalization, the quality of the data designer, and even the database product used. Here we are also going to split the RDBMS schema world into two subdomains:

- 1) Situations where the RDBMS schema does NOT map closely to objects. Meaning, the objects in your system definition do not map one-to-one with tables in your schema. This is a common situation when adapting new software systems on top of legacy databases – often created when procedural client-server systems were the norm. It would not be uncommon in this type of a situation to define a single object that could map in small parts to 5-10 tables in the database.
- 2) Situations where the tables in your database and the objects in your schema map closely. That is, there is a near one-to-one relationship between objects and tables. This is common in more modern software systems.

Here we argue that the only viable solution for O-R mapping in situation 1 is hand coding. There may or may not be value in using a simple mapping as an intermediate step to true object mapping. It could drastically reduce the coding necessary to achieve a valuable mapping, thus making it an option for RAD development; however it will almost certainly lead to performance degradation over hand coding.

We are also going to define two levels of complexity in our mapping. We will term these Simple mapping and Extended mapping.

Simple O-R mapping: Mapping software objects to database tables where relations are not considered. A simple mapping will not reflect one-one, one-many, or many-many relationships between tables in the Object representation. A simple mapping can be EXTREMELY valuable in its own right, as well as being a useful first step in coding an Extended mapping layer.

Extended O-R mapping: Mapping software objects to database tables where relations are considered. Here the O-R layer attempts to manage one-one, one-

many and many-many relationships in the generated object layer. These relations will manifest themselves in the Object world as associations and aggregations.

Note: if data and object inheritance is important, consideration of an object database is recommended over attempting to use a tool for O-R mapping. If a relational database is required, hand coding the O-R layer or investigation of 3rd party tools are the best options.

Available Solutions and their Merits

The O-R mapping problem is not new. Solutions abound, and vary in quality, flexibility, complexity, performance, ease of use, etc. Decisions on which O-R mapping approach to use for a given system are based on trade offs between these various attributes.

As seen above, the state of your schema may dictate the use of hand coding. The need for rapid development may dictate the use of generation or a 3rd party tool. The high cost of 3rd party tools, and hand coding may drive you towards a code generation approach. Here we will attempt to compare and contrast the three primary classes of solution: hand coding, 3rd party tools, and code generation.

Hand Coding

The most straightforward solution to O-R mapping is to hand code an O-R layer. When done correctly, this involves writing a framework, and a series of classes conforming to that framework that are used to map from the database to a set of business objects, data transfer objects or EJBs.

This approach is by far the most flexible. All aspects of the mapping interactions are handled by the developer's own hands. Though best practices should always be followed (see Appendix A), there are virtually no restrictions regarding the layout of your data or the database you are using. There are no restrictions regarding the correlation of objects to tables, or attributes to columns. Done correctly, your O-R layer can be database portable.

Choices regarding the use of PreparedStatements vs. dynamic SQL, vs. stored procedures are up to the developer. In addition the choice of connection pool and object caching are also available. In the spirit of XP, the system can be designed simply, and extended or improved as necessary. Off the shelf components can be chosen individually, each on its own merits.

The key drawback of this solution is that it involves a significant amount of time to hand code the necessary mapping classes. For a complex schema this task will constitute a significant portion of your coding budget. In addition, as the database

schema changes and evolves, it is necessary to continually maintain the object mapping code. This can be a tedious process for developers.

O-R Mapping Tools

There are a number of O-R mapping tools available on the market today; references to several popular products are available in Appendix A.

The tools mentioned allow the user to define advanced relationships between database tables and attributes and Java objects. Each has its merits, however fundamentally all require the developer to release the coding of the O-R layer from his/her control.

Some of these tools operate in a GUI only mode, which is fine for a single developer project, or rapid prototyping. However, large, integrated, long-term projects require a GUI independent build process based on make or Ant which does not lend itself to any sort of GUI based tools. Indeed it often escapes me how product vendors often ignore this fact and invest heavily in flashy GUI based systems unsuitable for real enterprise development.

Another significant drawback of these systems is the occasional attempt to replace the SQL query language with a product specific, proprietary language. SQL may not be optimized for object based queries, and it has its limitations, however let's remember that it is supported by nearly every database, and virtually all developers have a background with SQL. Ultimately, all of these will be mapped back to SQL, only you won't have control over the mapping. SQL is not perfect, however access to it seems essential to a successful O-R layer.

There are several new query language standards proposed -- EJBQL, JDOQL, OSQL. If any of these gain traction there may be a reason to change, but for now it seems the benefits of re-inventing a query language don't justify migrating away from standard SQL.

Code Generation

Code generation comes in many flavors. Many developers may be familiar with the code generation functions available in UML modeling tools such as Together/J or Rational Rose. Generally, code generation involves the production of source code from some type of model or meta data. In terms of the O-R mapping problem, code generation would involve inspection of the database schema, and generation of code for an O-R mapping layer from it.

Several tools are available, and general conclusions regarding the attributes of these systems can be made, however a specific discussion is difficult. Later we

will focus on a specific approach to code generation that will make the characteristics of these systems more apparent.

Comparisons

The following chart compares the three fundamental approaches to Object-Relational mapping.

	Learning Curve	Performance	Control	Sensitivity to Change	Cost	Development Time
Hand Coding	Low	Varies	High	High	High	High
3rd Party Tools	High	High	Low	Low	High	Low
Code Generation	Low	Varies	High	Low	Low	Medium

Each of the above options has considerable merit. The power of mapping tools is compelling. Often the need for flexibility will dictate the use of a hand-coded solution.

What is needed is a combination of the benefits of hand coding and 3rd party tools that affords the developer complete control over the O-R mapping code, while adding the automation and mapping power of the off the shelf toolkits.

An XSLT Based Approach to Code Generation for O-R Mapping

This paper is not about the general problem of O-R mapping, rather a solution to a subset of the problem. Above, we presented an introduction to the O-R problem to lay the groundwork for the proposed solution. The problem of O-R mapping is not new, and is well defined elsewhere. There are several excellent white papers available describing this problem set and some solutions to it. References can be found in Appendix A.

XML and XSL



The eXtensible Markup Language or XML is a generalized language for defining extended lexicons for virtually any problem domain. XML is emerging as the defacto standard business to business communications language, however its usefulness does not end there.

The eXtensible Stylesheet Language or XSL is a language used for transforming XML documents into other types of documents. Transformations from XML to HTML, EDI, plain text, or another form of XML are common uses of XSL. The act of applying a style sheet to an XML document is called a transformation, hence XSL Transformations or XSLT.

These two technologies are incredibly powerful. Virtually all Java developers will have some exposure to these technologies in their 'toolbox.'

DB -> XML -> .java

One of the more interesting uses of XSLT is the possibility of using it to convert XML documents into java source code files. Plain text is a common result of transformation often to a CSV or EDI format. Noting that java source code is just another plain text file is central to our approach.

At this point, if we can describe the schema of the database in an XML document, an entirely new solution to the O-R mapping problem becomes evident. We can write a series of XSL transformations that operate upon XML describing the database to produce a generated Java data access layer.

Note that this solution to O-R mapping has two distinct benefits:

1. The power and complexity of the O-R layer is virtually limitless, easily rivaling current O-R mapping products.
2. The source generated is 100% your code. Customize it as necessary to suit the needs of your specific applications. Any number of designs can be implemented by varying the XSL stylesheets. Generation of DTOs and Factories, or EJB entity beans are two possible designs.

There are many other benefits to this approach; however the key point here is that we've combined the benefits of the two alternative approaches (hand coding or 3rd party tools).

Given the definitions of type 1 and 2 schemas, and simple and extended mappings above it becomes apparent that XSLT Code Generation is a viable option for simple type 1 mappings, and potentially (with modification) extended type 1 mappings.

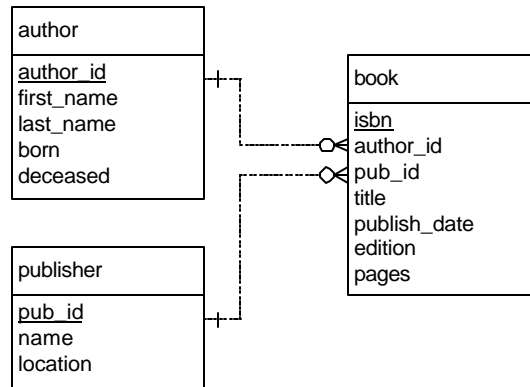
A Simple Example – A Bookstore Application

In the software development world, an example is worth a thousand words. Next, we will present an example of XSLT for code generation. Appendix B-F contain some of the detailed inputs and outputs from the example provided. The complete example including database, build scripts, transformations, required utilities, and source code for helper classes is available for download free of charge from: <http://www.jettools.com/products/jetgen>.

The Problem:

A bookseller has decided to create a web site allowing users to lookup books via an Internet web site. For some time, the bookseller has been using a RDBMS system to track information about each of the books in its collection. You, being a strong java/oo/web developer, have been hired to create this web based book searching system. In addition, they would also like you to code a Java Swing based GUI for use in-house for updating the data when new books are acquired, or removed from the collection.

The schema currently used by the RDBMS for storing the book information is shown below:

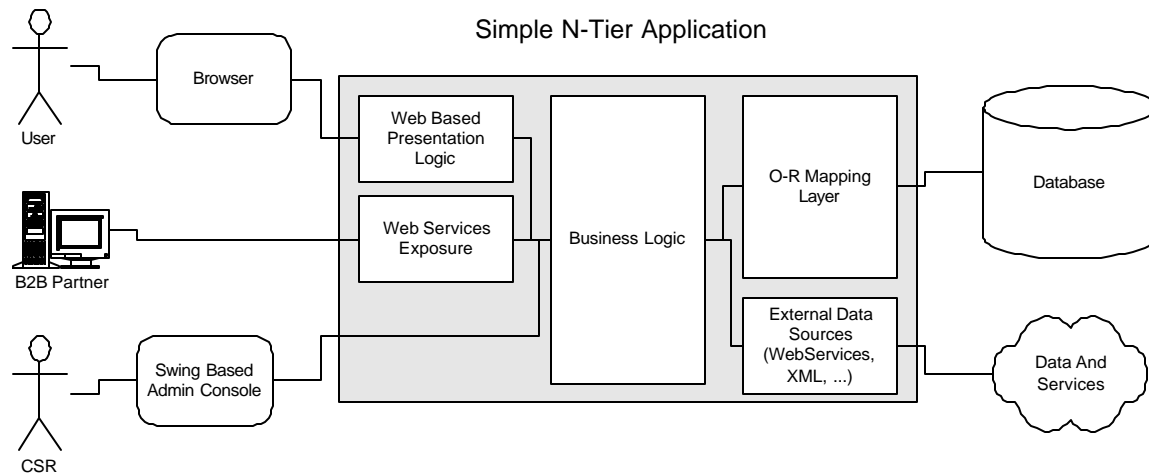


A brief analysis of the database schema shows us that it maps well to objects. In fact, the relationship is one to one, tables to objects. A type 2 simple mapping will be sufficient for our O-R mapping needs.

Analysis and Design:

Experience teaches us that the best approach to this problem will be a 3-tier design. Use of multi-layered middleware with business logic, SQL database access and presentation separated will prevent duplication and maintenance of multiple client-server systems. Initially, you will develop a middle data access layer and all of the necessary object mapping code. Then you will focus on writing the business logic and presentation tiers.

The following diagram illustrates the n-tiered design that we are working towards. Our focus here is a discussion of the O-R mapping layer. (We will see later that this code generation approach may have other applications in the system.)



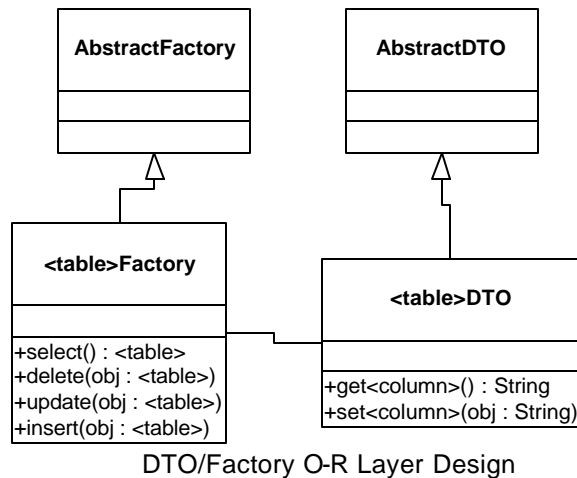
It is worth noting that the code in the O-R mapping layer has several characteristics:

- The code is voluminous. The sheer bulk of the O-R mapping code can lead us to look for a generation solution.
- The code is repetitive. Once a general framework and supporting classes are developed, hand coding of the O-R layer can be mind numbingly boring.
- The code is sensitive to changes in the schema. The frequency of changes to the schema will often depend on the maturity of the database. Young projects (often RAD projects) will have a highly dynamic schema that can drive developers crazy keeping the O-R layer in step.

The use of generation does not release us from the need to design the O-R layer before coding. An experienced developer or architect will have a good handle on how to design the O-R layer.

For this example, we will choose simple data transfer objects (DTOs) for data encapsulation and a factory pattern to marshal the DTOs to and from the database. The following diagram shows the concept for our design including abstract superclasses and concrete implementations.

EJB entity beans would be another valid option. To focus on the O-R mapping problem we have chosen a simpler design. It should be evident at the end of this exercise that the generation of EJB's would simply involve the modification of the transformations presented in this example.



With our overall system design in hand, and a more detailed O-R mapping layer design, we are ready to consider the task of code generation. Following the process outlined above, we will generate the XML meta-data for the schema, and code the XSL transformations.

A Three Step Iterative Process

It is assumed that the reader will follow the sample system available for download along with this paper. For complete understanding of the XSLT generation system, studying the example will be necessary. We will present an overview of the steps involved in this section.

1) Generation. First, we need a representation of the database schema as an XML document. We do this with a simple utility called DBtoXML in the example. DBtoXML uses the meta-data available in the JDBC API to generate the XML file.

Output from this utility for the sample database included with the example is shown in Appendix C. The genxml Ant task used to execute the utility is included in Appendix F.

2) Transformation. Next we need to apply XSL transformations to this XML to product java source code.

Here we make the (large) assumption that you have already authored the stylesheets. Given the utilities out-of-the-box, by far the largest task for XSLT code generation is a authoring the stylesheets. It is a highly iterative process, so automation of the transformation and compilation steps is encouraged. Ant is a good tool for this automation. It will also be extremely helpful to use the DTO and Factory stylesheets as examples.

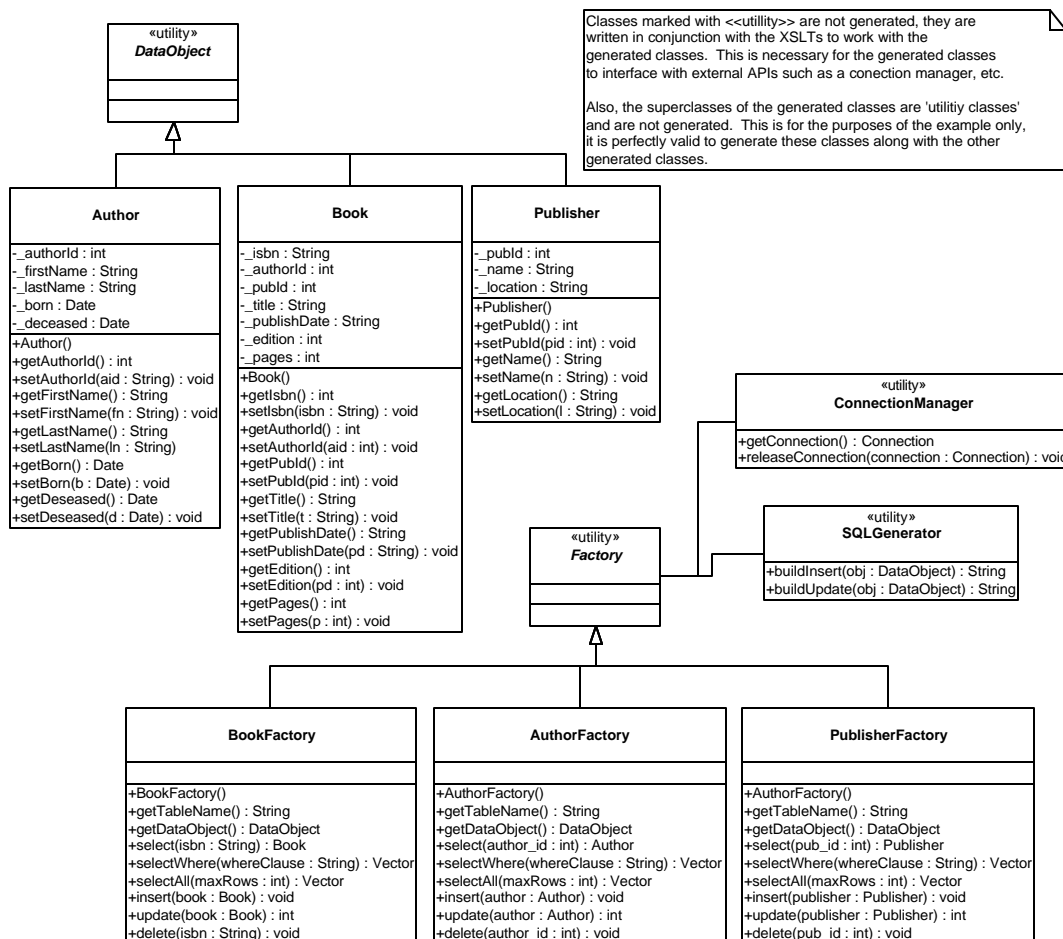
The DTO and Factory stylesheets are shown in Appendix D and E. The genjava Ant task used to apply the stylesheets to the XML is shown in Appendix F.

3) Compilation. Finally, we need to compile the generated source. When authoring the stylesheets, you will iterate many times through steps two and three.

The ant task for this step is omitted. Code compilation tasks in Ant should be well understood, and the Ant documentation has numerous examples.

Using these three components in concert yields the generated O-R mapping layer. Below we show the UML model of the classes resulting from our code generation tasks.

The diagram shows several of the support classes not generated via XSL. Details are omitted here, however source code is available in the downloaded example.



DTO/Factory classes generated by XSLT O-R Mapping

With this data access layer in hand, having only spent a few hours, rather than days or weeks, you're now ready to begin coding the other software layers. Choice of EJBs, JSPs, SOAP services, are all still in your hands, however the task of mapping your data to objects is complete in a matter of hours.

Comments on the Example

The example presented is contrived, a three table database will be rare in industry. It should be obvious however that the code generated is valuable. Consider a much larger schema that would no doubt be used in a real n-tiered system. As the size of the schema increases the generated code becomes more and more valuable.

The design presented is fairly typical, however it is by no means the only solution. This is a benefit of XSL code generation. The developer has complete control over the resulting O-R mapping layer, by coding the transformations however they see fit. Extension to using EJB entity beans is another potential design. This design would result in significant additional classes and interfaces, making the code generation even more valuable.

Observations on XSLT Code Generation

This paper has presented the O-R mapping problem, split it to subdomains, described a general solution and presented a concrete example of the solution. Finally, we would like to explore some implications and extrapolations resulting from the XSLT approach to O-R mapping.

Extended Mapping

Extended mapping of relations was not considered. (Extended mapping as defined in an earlier section). A mapping that includes the table relations could be extremely valuable. It is certainly possible to extend the concept to such a system. Here are the general steps:

1. Generate additional meta-data about table relations into the schema XML. This should be possible using JDBC via the exported and imported key methods in DatabaseMetaData.
2. Code XSL stylesheets that make use of the new meta-data. This would be a difficult task, however once completed, the transformations could be re-used on any number of projects and databases.

Persistence Architecture

XSLT code generation is not a persistence framework. It is a key component in a larger system that would include connection pooling, caching, active/lazy materialization support, and other runtime features of a complete persistence framework.

The example addresses some of these considerations with support classes. The SQLGenerator class can be used to optimize SQL. Connection pooling can be easily integrated with the adaptor interface already included. Other components should be integrated as the requirements demand.

Using XSLT O-R mapping, you can combine components to create a complete persistence architecture to suit your needs.

XSLT as a General Approach to Repetitive Software Development

The above example shows a solution to a very basic O-R mapping scenario. One should be so lucky in industry. The object of the exercise is to demonstrate the mechanics of using XML and XSL as a means of generating repetitive code based on some sort of meta data.

In many ways, this approach is a solution waiting for a problem. XML and Java together are being adopted at a dizzying rate... In the example, step 1 of our procedure was to generate an XML file containing our database meta data. Perhaps for some other applications the first step will not be necessary, as the meta-data may already be in an XML format.

If you've read this paper this far, you may already have a list of potential applications of this technology floating in your head. Virtually any external (or maybe even internal) system for which we can generate meta-data in XML is a candidate for code generation via XSL.

Here are a few potential applications:

- **Web Services.** Virtually all web service interaction is described in XML. The communication protocol, the deployment descriptors, and the service definition systems are all based on XML. There are numerous potential applications for code generation here.
- **XMI.** One of the applications of XMI is representation of UML models. If we can successfully represent the UML model in XMI, generation of source code could be accomplished via XSL. An XMI UML model could be translated into many languages.
- **JNDI.** Code generation for JNDI could follow a track similar to the one presented here. If we can use the JNDI API to generate meta-information about a datasource in XML, we could generate useful classes via XSL.

- **XML Schema.** If and when XML Schemas replace DTD's for XML definition, useful code could be generated from the very definition of the XML data format.

Extensions of this concept could lead to a generalized toolkit of utilities and stylesheets that would be applicable to a wide range of problems. Code generation is not a substitution for good process, methodology, analysis and design. However using code generation as a component in an integrated approach could greatly reduce repetitive tasks.

A developer well versed in code generation could increase quality code productivity by an order of magnitude with an XSLT code generation toolkit.

Appendix A: References

Databases

MySQL: <http://www.mysql.com>

PostgreSQL: <http://www.postgresql.org>

Object-Relational Mapping

Ambler, S.W. *Mapping Objects to Relational Databases*. 2000.
<http://www.ambysoft.com/mappingObjects.pdf>

Fussell, M.L. *Foundations of Object Relational Mapping*.
<http://www.chimu.com/publications/objectRelational/>

O-R Mapping Products

JET Gen – an O-R mapping tool based on XSLT. This product is the direct result of the thought process presented in this paper, and it was used in the examples.
<http://www.jettools.com/products/jetgen/>

Toplink: <http://www.webgain.com>

Hibernate: <http://hibernate.sourceforge.net>

XML and XSL

XML Definition: <http://www.w3c.org/XML/>

XSL Definition: <http://www.w3c.org/Style/XSL/>

The Apache XML Project, Home of the Xerces XML Parser, and the Xalan XSLT engine.
<http://xml.apache.org>

Appendix B: Typemap.properties (Oracle sample)

```
#####  
# This is a sample typemap for oracle.  
#  
# Because Oracle uses the same data type (NUMBER) for all ints, floats,  
# doubles, etc. this may pose a problem. This particular typemap  
# assumes that all NUMBERS are ints.
```



```
#####
CHAR:String
BIGINT:long
DECIMAL:double
DATE:Date
VARCHAR:String
VARCHAR2:String
FLOAT:float
BIT:boolean
SMALLINT:int
TIME:Date
LONGVARCHAR:String
CURRENCY:float
TIMESTAMP:Date
NUMERIC:double
REAL:float
TINYINT:int
DOUBLE:double
NUMBER:int
```

Appendix C: DBtoXML output of the sample database schema.

```
<database name="bookstore" >
  <table name="book" javaname="Book"
type="TABLE" >
    <primary_keys>
      <column>
        <javatype>String</javatype>
        <javaname>Isbn</ javaname>
        <rsgetter>String</rsgetter>
        <name>isbn</name>
        <type>CHAR</type>
      </column>
    </primary_keys>
    <column>
      <javatype>String</javatype>
      <javaname>Isbn</ javaname>
      <rsgetter>String</rsgetter>
      <name>isbn</name>
      <type>CHAR</type>
    </column>
    <column>
      <javatype>int</javatype>
      <javaname>AuthorId</ javaname>
      <rsgetter>Int</rsgetter>
      <name>author_id</name>
      <type>INTEGER</type>
    </column>
    <column>
      <javatype>String</javatype>
      <javaname>PubId</ javaname>
      <rsgetter>String</rsgetter>
      <name>pub_id</name>
      <type>CHAR</type>
    </column>
    <column>
      <javatype>String</javatype>
      <javaname>Title</ javaname>
      <rsgetter>String</rsgetter>
      <name>title</name>
      <type>CHAR</type>
    </column>
    <column>
      <javatype>String</javatype>
      <javaname>PublishDate</ javaname>
      <rsgetter>String</rsgetter>
      <name>publish_date</name>
      <type>CHAR</type>
    </column>
    <column>
      <javatype>int</javatype>
      <javaname>Edition</ javaname>
      <rsgetter>Int</rsgetter>
      <name>edition</name>
      <type>INTEGER</type>
    </column>
    <column>
      <javatype>int</javatype>
      <javaname>Page</ javaname>
      <rsgetter>Int</rsgetter>
      <name>pages</name>
      <type>INTEGER</type>
    </column>
  </table>
  <table name="author" javaname="Author"
type="TABLE" >
    <primary_keys>
      <column>
        <javatype>int</javatype>
        <javaname>AuthorId</ javaname>
        <rsgetter>Int</rsgetter>
        <name>author_id</name>
        <type>INTEGER</type>
      </column>
    </primary_keys>
    <column>
      <javatype>int</javatype>
      <javaname>AuthorId</ javaname>
      <rsgetter>Int</rsgetter>
      <name>author_id</name>
      <type>INTEGER</type>
    </column>
    <column>
      <javatype>String</javatype>
      <javaname>FirstName</ javaname>
      <rsgetter>String</rsgetter>
```

```

        <name>first_name</name>
        <type>CHAR</type>
    </column>
    <column>
        <javatype>String</javatype>
        <javaname>LastName</javaname>
        <rsgetter>String</rsgetter>
        <name>last_name</name>
        <type>CHAR</type>
    </column>
    <column>
        <javatype>Date</javatype>
        <javaname>Born</javaname>
        <rsgetter>Date</rsgetter>
        <name>born</name>
        <type>DATE</type>
    </column>
    <column>
        <javatype>Date</javatype>
        <javaname>Deceased</javaname>
        <rsgetter>Date</rsgetter>
        <name>deceased</name>
        <type>DATE</type>
    </column>
</table>
<table name="publisher"
javaname="Publisher" type="TABLE" >
    <primary_keys>
    <column>
        <javatype>int</javatype>
        <javaname>PubId</javaname>
        <rsgetter>Int</rsgetter>
        <name>pub_id</name>
        <type>INTEGER</type>
    </column>
    </primary_keys>
    <column>
        <javatype>int</javatype>
        <javaname>PubId</javaname>
        <rsgetter>Int</rsgetter>
        <name>pub_id</name>
        <type>INTEGER</type>
    </column>
    <column>
        <javatype>String</javatype>
        <javaname>Name</javaname>
        <rsgetter>String</rsgetter>
        <name>name</name>
        <type>CHAR</type>
    </column>
    <column>
        <javatype>String</javatype>
        <javaname>Location</javaname>
        <rsgetter>String</rsgetter>
        <name>location</name>
        <type>CHAR</type>
    </column>
</table>
</database>

```

Appendix D: Data transfer object XSL stylesheet.

```

<xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="text"/>
<xsl:template match="table">
package domain;

import java.util.*;
import java.io.*;
import java.io.Serializable;

/**
 * This is a DataObject class wrapping the schema of the table <xsl:value-of
select="@name"/>.
 *
 * Generated by JETGen Database Code Generator -- http://www.jettools.com.
 */
public class <xsl:value-of select="@javaname"/> extends DataObject
    implements Serializable
{
    private static final String _tablename=&quot;<xsl:value-of select="@name"/>&quot;;
    private static final Hashtable _cam=new Hashtable();

    static String primaryKeyColumns[]=
    {<xsl:for-each select="primary_keys/column">&quot;<xsl:value-of
select="name"/>&quot;<xsl:if test="not(position())=last()">,</xsl:if></xsl:for-each>;

    static {
        <xsl:for-each select="column" >
            _cam.put (&quot;<xsl:value-of select="javaname"/>&quot;.toUpperCase(),
&quot;<xsl:value-of select="name"/>&quot;); </xsl:for-each>
        }

```

```

// Member variables
<xsl:for-each select="column">
private <xsl:value-of select="javatype"/> _<xsl:value-of select="javaname"/> // SQL
type:<xsl:value-of select="type"/> </xsl:for-each>

// constructors
public <xsl:value-of select="@javaname"/> () {}

public <xsl:value-of select="@javaname"/>(<xsl:call-template name="col_param_list"/>)
{
  <xsl:call-template name="col_assignment"/>
}

/** Column accessor map maps SQL column names to java accessor names.
 * it is used by the SQLGenerator class. */
public Hashtable columnAccessorMap() {return _cam;}

/** toString prints the type of object, and it's primary keys.
 */
public String toString(){
  String sname=this.getClass().getName();
  if (sname.indexOf(".") != -1)
    sname=sname.substring(sname.lastIndexOf(".")+1, sname.length());
  String key=<xsl:text>&quot;[&quot;+</xsl:text><xsl:call-template
name="pkey_string"/><xsl:text>&quot;]&quot;;</xsl:text>
  return sname+key;
}

/** Returns this object's primary key columns for db access. */
public String[] primaryKeyColumns() {return primaryKeyColumns;}

public String tableName() {return _tablename;}

public String[] primaryKeyValues()
{
  Vector v=new Vector();
  <xsl:for-each select="primary_keys/column">v.addElement(&quot;&quot;+_<xsl:value-of
select="javaname"/>);
  </xsl:for-each>
  String vals[]=new String[v.size()];
  v.copyInto(vals);
  return vals;
}

// Accessors
<xsl:for-each select="column">
<xsl:apply-templates select="." />
</xsl:for-each>
}
</xsl:template>

<xsl:template match="column">
public void set<xsl:value-of select="javaname"/>(<xsl:value-of select="javatype"/> x){
  _<xsl:value-of select="javaname"/>=x;
}
public <xsl:value-of select="javatype"/> get<xsl:value-of select="javaname"/>(){
  return _<xsl:value-of select="javaname"/>;
}
}
</xsl:template>

<xsl:template name="pkey_string">
<xsl:for-each select="primary_keys/column"><xsl:value-of select="javaname"/><xsl:if
test="not(position()=last())">+&quot;; &quot;+</xsl:if></xsl:for-each>
</xsl:template>

<xsl:template name="col_param_list">
<xsl:for-each select="column"><xsl:value-of select="javatype"/> parm<xsl:value-of
select="position()"/><xsl:if test="not(position()=last())">, </xsl:if></xsl:for-each>

```

```

</xsl:template>

<xsl:template name="col_assignment">
<xsl:for-each select="column">_<xsl:value-of select="javaname"/> = parm<xsl:value-of
select="position()"/>;
  </xsl:for-each></xsl:template>
</xsl:stylesheet>

```

Appendix E: Factory XSL stylesheet.

```

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text"/>

<xsl:template match="table">
package domain;

import java.util.*;
import java.sql.*;

/**
 * This class, generated by xsl, is used to build data objects that
 * 'wrap' RDBMS tables.
 *
 * Generated by JETGen Database Code Generator -- http://www.jettools.com.
 */
public class <xsl:value-of select="@javaname"/>Factory extends Factory
{
  private ConnectionManager cm=null;

  public static boolean sqlDebug=true;

  private static boolean verified=false;

  private static String tablename=&quot;<xsl:value-of select="@name"/>&quot;;

  private static String columnNames[]=
  {<xsl:for-each select="column">&quot;<xsl:value-of select="name"/>&quot;<xsl:if
test="not(position())=last()>,</xsl:if></xsl:for-each>;

  private static String columnTypes[]=
  {<xsl:for-each select="column">&quot;<xsl:value-of select="type"/>&quot;<xsl:if
test="not(position())=last()>,</xsl:if></xsl:for-each>;

  private static SQLGenerator sqlGenerator=null;

  static {
    try {
      sqlGenerator=new SQLGenerator(<xsl:value-of select="@javaname"/>.class);
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }

  public String getTableName() {return tablename;}

  public DataObject getDataObject() {
    return new <xsl:value-of select="@javaname"/>();
  }

  /** Constructor */
  public <xsl:value-of select="@javaname"/>Factory()

```

```

    {
    }

    /** Select by primary key. */
    public <xsl:value-of select="@javaname"/> select <xsl:text>( </xsl:text><xsl:apply-
templates select="primary_keys"/><xsl:text>)</xsl:text>
    throws SQLException{
        Connection con=null;
        Statement state=null;
        <xsl:value-of select="@javaname" /> x=null;
        try {
            con=getConnection();

            String sql=&quot;select * from <xsl:value-of select="@name"/> where <xsl:call-
template name="pkey_sql"/>;
            if (sqlDebug)
                System.out.println(sql);
            state=con.createStatement();

            ResultSet rs=state.executeQuery(sql);
            if (rs.next()) {
                // build the object
                x=new <xsl:value-of select="@javaname"/><xsl:text>( </xsl:text>;
                <xsl:for-each select="column">
                    x.set<xsl:value-of select="javaname"/><xsl:text>( </xsl:text>rs.get<xsl:value-of
select="rsgetter"/><xsl:text>( </xsl:text>&quot;<xsl:value-of
select="name"/>&quot;<xsl:text>);</xsl:text>
                    </xsl:for-each>
                }
            }
            else {
                return null;
            }
        }
        catch (Exception ex) {
            ex.printStackTrace();
            throw new RuntimeException("Unable to build objects:"+ex);
        }
        finally {
            if (state != null)
                try{state.close();}
                catch (Exception ex) {}
            if (con != null)
                try{releaseConnection(con);}
                catch (Exception ex) {}
        }
        return x;
    }
}

/** Select all */
public Vector selectAll <xsl:text>(int max)</xsl:text>
throws SQLException, DataException{
    Connection con=null;
    Statement state=null;
    Vector v=null;
    <xsl:value-of select="@javaname"/> x=null;
    try {
        con=getConnection();

        <xsl:variable name="tname"><xsl:value-of select="@name"/></xsl:variable>
        <xsl:choose>
            <xsl:when test="(contains($tname, string('SITE_DB'))=true())">
                String sql=&quot;select * from <xsl:value-of select="@name"/> where STATUS !=
'D'&quot;;
            </xsl:when>
            <xsl:when test="(contains($tname, string('PROMOTION_DB'))=true())">
                String sql=&quot;select * from <xsl:value-of select="@name"/> where STATUS !=
'D'&quot;;
            </xsl:when>
        }
    }
}

```

```

<xsl:otherwise>
String sql="select * from <xsl:value-of select="@name"/>";
</xsl:otherwise>
</xsl:choose>
if (sqlDebug)
    System.out.println(sql);
state=con.createStatement();
try {state.setMaxRows(max);} catch (Exception ex) {}
v=new Vector();

ResultSet rs=state.executeQuery(sql);
while (rs.next()) {
    // build the object
    x=new <xsl:value-of select="@javaname"/><xsl:text>(</xsl:text>;
    <xsl:for-each select="column">
        x.set<xsl:value-of select="javaname"/><xsl:text>(</xsl:text>rs.get<xsl:value-of
select="rsgetter"/><xsl:text>(</xsl:text>&quot;<xsl:value-of
select="name"/>&quot;<xsl:text>);</xsl:text>
        </xsl:for-each>
        v.addElement(x);
    }
}
finally {
    if (state != null)
        try{state.close();}
        catch (Exception ex) {}
    try{releaseConnection(con);}
    catch (Exception ex) {}
}
return v;
}
}

/** Select where. Allows application programmer to add their
 * own where clause to 'select * from <xsl:value-of select="@name"/>
 * where xxx' User replaces xxx only, NOT where. */
public Vector selectWhere <xsl:text>(String where)</xsl:text>
throws SQLException,DataException {
    Connection con=null;
    Statement state=null;
    Vector v=null;
    <xsl:value-of select="@javaname"/> x=null;
    try {
        con=getConnection();
        String sql="select * from <xsl:value-of select="@name"/> where &quot;+where;

        if (sqlDebug)
            System.out.println(sql);
        state=con.createStatement();

        v=new Vector();

        ResultSet rs=state.executeQuery(sql);
        while (rs.next()) {
            // build the object
            x=new <xsl:value-of select="@javaname"/><xsl:text>(</xsl:text>;
            <xsl:for-each select="column">
                x.set<xsl:value-of select="javaname"/><xsl:text>(</xsl:text>rs.get<xsl:value-of
select="rsgetter"/><xsl:text>(</xsl:text>&quot;<xsl:value-of
select="name"/>&quot;<xsl:text>);</xsl:text>
                </xsl:for-each>
                v.addElement(x);
            }
        }
    }
    finally {
        if (state != null)
            try{state.close();}
            catch (Exception ex) {}
        try{releaseConnection(con);}
    }
}
}

```

```

        catch (Exception ex) {}
    }
    return v;
}

/** Insert */
public int insert <xsl:text>(</xsl:text><xsl:value-of select="@javaname"/>
x<xsl:text></xsl:text>
throws SQLException, DataException {
    Connection con=null;
    Statement state=null;
    int rows=-1;
    try {
        con=getConnection();
        String sql=sqlGenerator.buildInsert(x);

        if (sqlDebug)
            System.out.println (sql);

        state=con.createStatement();
        rows=state.executeUpdate(sql);
    }
    finally {
        if (state != null)
            try{state.close();}catch (Exception ex) {}
        try{releaseConnection(con);}
        catch (Exception ex) {}
    }
    return rows;
}

/** Update */
public int update <xsl:text>(</xsl:text><xsl:value-of select="@javaname"/>
x<xsl:text></xsl:text>
throws SQLException, DataException {
    Connection con=null;
    Statement state=null;
    int rows=-1;
    try {
        con=getConnection();
        String sql=sqlGenerator.buildUpdate(x);

        if (sqlDebug)
            System.out.println (sql);

        state=con.createStatement();
        rows=state.executeUpdate(sql);
    }
    finally {
        if (state != null)
            try{state.close();}catch (Exception ex) {}
        try{releaseConnection(con);}
        catch (Exception ex) {}
    }
    return rows;
}

/** Delete by primary key. */
public int delete <xsl:text>(</xsl:text><xsl:apply-templates
select="primary_keys"/><xsl:text></xsl:text>
throws SQLException, DataException{
    Connection con=null;
    Statement state=null;
    int rows=-1;
    try {
        con=getConnection();

```

```

        String sql=&quot;update <xsl:value-of select="@name"/> set STATUS='D' where
<xsl:call-template name="pkey_sql"/>;
        if (sqlDebug)
            System.out.println(sql);
        state=con.createStatement();

        rows= state.executeUpdate(sql);
    }
    finally {
        if (state != null)
            try{state.close();}
            catch (Exception ex) {}
        try{releaseConnection(con);}
        catch (Exception ex) {}
    }
    return rows;
}
}
</xsl:template>

<xsl:template name="cols_as_parms">
<xsl:for-each select="column"><xsl:value-of select="javatype"/> in_<xsl:value-of
select="javaname"/><xsl:if
test="not(position()=last())"><xsl:text>,</xsl:text></xsl:if></xsl:for-each>
</xsl:template>

<xsl:template match="primary_keys"><xsl:for-each select="column"><xsl:value-of
select="javatype"/> in_<xsl:value-of select="javaname"/><xsl:if
test="not(position()=last())"><xsl:text>,</xsl:text>
</xsl:if></xsl:for-each>
</xsl:template>

<xsl:template name="pkey_sql">
<xsl:for-each select="primary_keys/column"><xsl:value-of
select="name"/><xsl:text>='</xsl:text>&quot;+in_<xsl:value-of
select="javaname"/><xsl:text>+&quot;';</xsl:text><xsl:if
test="not(position()=last())"><xsl:text> AND </xsl:text></xsl:if><xsl:if
test="position()=last()">&quot; </xsl:if></xsl:for-each>
</xsl:template>

<xsl:template name="col_list">
<xsl:for-each select="column"><xsl:value-of select="name"/><xsl:if
test="not(position()=last())">,</xsl:if></xsl:for-each>
</xsl:template>

<xsl:template name="first_col">
<xsl:for-each select="column"><xsl:if test="position()=1"><xsl:value-of
select="name"/></xsl:if></xsl:for-each>
</xsl:template>

</xsl:stylesheet>

```

Appendix F: Ant tasks for generation and transformation.

```

<!-- ++++++ -->
<!-- This is a sample ANT build file showing a typical usage of the -->
<!-- JETGenerator XSLT based code generator. -->
<!-- -->
<!-- ANT can be obtained from http://jakarta.apache.org/ant -->
<!-- -->
<!-- The JETGenerator can be obtained from www.jettools.com -->
<!-- ++++++ -->
<project name="build" default="all" basedir=".">
  <!-- do some setup work -->
  <target name="init">
    <property name="JAR_DIR" value="../lib" />

```



```

</target>

<!-- The equivalent of 'make all' -->
<target name="all" depends="init,genxml,genjava,compile" />

<!-- Generate DBXML from a database connection -->
<target name="genxml" depends="init" >
  <mkdir dir="generated" />
  <java classname="DBtoXML" fork="true" >
    <arg value="-url jdbc:ids:sampled/ids/sample.prp" />
    <arg value="-driver org.enhydra.instantdb.jdbc.idbDriver" />
    <arg value="-out generated/db.xml" />
    <arg value="-tables book,author,publisher" />
    <arg value="-user " />
    <arg value="-pass " />
    <!-- The typemap.properties specification is optional -->
    <!-- This may be required if your JDBC driver does not support -->
    <!-- some of the extended features of the JDBC spec. -->
    <!-- several samples of typemap.properties are given in the -->
    <!-- xsl directory -->
    <!-- arg value="-typemap ${XSL_DIR}/typemap.properties" -->

    <!-- This will include the required gen.jar, xalan.jar, -->
    <!-- and xerces.jar -->
    <classpath>
      <fileset dir="${JAR_DIR}">
        <include name="**/*.jar" />
        <include name="**/*.zip" />
      </fileset>
    </classpath>
  </java>
</target>

<!-- Generate Java Code from DBXML using XSLT -->
<target name="genjava" depends="init">
  <mkdir dir="generated" />
  <java classname="XMLTableProcessor" fork="true">
    <arg value="-input generated/db.xml" />
    <arg value="-style xsl/DataObject.xsl" />
    <arg value="-outdir generated/domain" />

    <classpath>
      <fileset dir="${JAR_DIR}">
        <include name="**/*.jar" />
        <include name="**/*.zip" />
      </fileset>
    </classpath>
  </java>

  <java classname="XMLTableProcessor" fork="true">
    <arg value="-input generated/db.xml"/>
    <arg value="-style xsl/Factory.xsl"/>
    <arg value="-outdir generated/domain" />
    <arg value="-suffix Factory" />
    <classpath>
      <fileset dir="${JAR_DIR}">
        <include name="**/*.jar" />
        <include name="**/*.zip" />
      </fileset>
    </classpath>
  </java>

  <java classname="XMLDBProcessor" fork="true">
    <arg value="-input generated/db.xml"/>
    <arg value="-style xsl/FactoryManager.xsl"/>
    <arg value="-outdir generated/domain" />
    <classpath>
      <fileset dir="${JAR_DIR}">

```

```
        <include name="**/*.jar" />
        <include name="**/*.zip" />
    </fileset>
</classpath>
</java>
</target>

<!-- Compile the static and the generated classes to a single lib dir -->
<target name="compile" depends="init">
    <mkdir dir="lib" />
    <javac destdir="lib">
        <classpath>
            <fileset dir="${JAR_DIR}">
                <include name="**/*.jar" />
                <include name="**/*.zip" />
            </fileset>
        </classpath>

        <src>
            <pathelement location="src" />
            <pathelement location="generated" />
        </src>
    </javac>
</target>
</project>
```