

# LatticeFramework Studio for Code Generation

Li Xin

Lattice Business Software International, Inc.

li.xin@latticesoft.com

## Abstract

The heart of model-driven architecture is model. Model raises the level of abstraction beyond programming by specifying the solution directly using domain concepts. The most two popular models for code generation are UML and Domain-Specific Model (DSM). UML or DSM? There is a war brewing among modeling methodologists. Neither approach is perfect. Can we take the best of both sides? We advocate combining both approaches. In this paper, we introduce XEML (Entity Markup Language) for automating software development in domain-specific world using general-purpose modeling language.

D.3.3 [Programming Languages]: Processors Code Generation;

D.2.11 [Software Engineering]: Software Architecture Domain specific architectures

## General Terms

Design, Standardization, Languages

## Keywords

Code Generation, model language

## 1. Introduction

Business today is demanding shorter time-to-market, increased productivity and more cost-effective solutions, creating efficient well-run application is vital for success. Current method, namely handwork of skilled professionals performing manual tasks, can not keep pace with growing complexity and size of the software projects.

A new trend in software development is to use model driven techniques and code generation to development software systems. In recent years, a number of approaches have been proposed including model-driven architecture, software product line and software factory. Common to all these approaches is to treat model as first class citizen and use some kinds of code generation techniques to automate the software application development.

Model raises the level of abstraction beyond programming by specifying the solution directly using domain concepts. On the

other word, models are used to raise the level of abstraction and hide the complexity and implementation details.

Generating complete code from models has been an industry goal for many years. Model-driven architecture approach provides a set of modeling notations, most based on UML, for specifying different layers of a system namely user interface, business logic and database in a platform independent manner. A set of code generator then transforms these models into platform-specific implementations.

The two most popular models for code generation are:

- UML for program modeling, part of the OMG's Model Driven Architecture (MDA)
- Domain-Specific Languages (DSLs), little languages that are created specifically to model some problem domain.

A UML model normally serves as basis of a MDA based development approach. UML is designed and specified as a general-purpose modeling language for object-oriented software system. UML has several general advantages; It is most widely known object-oriented modeling notation, it has a graphical notations which is readily understand, and a rich set of standards for capturing key features of OO systems.

DSM (Domain-Specific Model) is an alternative approach to modeling and code generation. Domain-Specific modeling raises the level of abstraction beyond programming by specifying the solution directly using domain concepts. The final software product then generated from these high-level specifications. This is achieved through DSL (Domain-Specific Languages) that follows domain abstraction semantics, allowing developers to focus on the solution rather than the technical implementation of the solution.

United or Domain-Specific modeling languages? There is a war brewing among modeling methodologists. Neither approach is perfect. UML are based on code world and offer only modest possibilities to raise design abstraction and to achieve full code generation. UML focus on visualizing the code and therefore fail to produce a significant improvement in overall productivity when compared to code in C# or Java. With DSM, you need to create more complex DSLs and generators as well as the DSLs need to be constituted evolved. This creates a new kind of complexity and different DSLs and generators make the standardization of software development much difficult if not impossible. Using models in automation depends on high quality DSLs, if free or cheap standard DSLs for key problem domains do not exist, the model automation is not going to happen. Developing

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

all the needed DSLs in the organization is not appealing for companies with limited budgets.

Can we take the best of both sides? We advocate combining both approaches. In this paper, we introduce XEML (Entity Markup Language) for automating software development in domain-specific world using general-purpose modeling language. The remainder of the paper describes the XEML Entity Markup Language in detail (section 2), presents LatticeFramework Studio, a tool for creating and using XEML and code generators (section 3), and provides an example how to model an order processing system using XEML (section 4), and finally section 5 gives some final remark.

## 2. XEML – Entity Markup Language

The interchangeable parts are first step to mass produce manufactured goods and the standardization of software development is most important and difficult part of model-driven architecture. DSM raises the level of abstraction beyond programming by specifying the solution directly using domain concepts; however those complex ‘luxury’ models make the standardization of software development much difficult if not impossible. UML is designed and specified as a general-purpose modeling language for object-oriented software system; however UML failed because it focuses on visualize the code, not focuses on the final software products generated from UML models.

### 2.1 Standard, Standard, Standard!

We propose a general-purpose modeling language named XEML, an Entity Markup Language based on XML for business applications. XML is a meta-language that can be used to develop general purpose modeling languages. It is a reasonable language for representing models and achieves separation of concerns about model and implements business software. XML has been embraced enthusiastically by all the major IT suppliers and user groups. It standardization and rapid uptake have been the major development in IT over the past ten years. Industry rivals like IBM, Microsoft, SUN and Oracle all support the core XML standard, are developing major products based on it, and collaborate to develop related standards. XML help the case for declarative modeling in below several ways:

- It standardization and rapid uptake simplify the standards development process.
- There is a multitude of software tools and APIs specifically designed for handles XML
- XML is text-based (therefore platform-independent).
- XML is extensible allowing more meaningful description of the underlying data.

### 2.2 Object and Database Technology

Two perspectives in software development are distinguished: the data-oriented perspective and object-oriented perspective. The first perspective refers to declarative knowledge and is based on relation DATABASE technology; the second perspective refers to procedural knowledge and is based on OBJECT technology. Even they are totally different approaches for software development, they use similar approach to model business entity. ER (Entity-Relationship) diagram is used to model database system and UML is used to model object system.

When you analyze a business domain, you are creating a conceptual model of that domain. You identify the entities in that domain, the attributes and behavior of those entities, and their relationships. In fact, in almost all practical business applications, it has to be translated into two concrete implementations - one in term of programming entities (objects), and another one in term of relational database entities (tables).

### 2.3 Entity Markup Language

Entity Markup Language (XEML) is a general-purpose modeling language based on XML for the implementation of business applications. Like any other Markup languages, XML tags and attributes are used to define XEML models. The figure 1 shows portion of a XEML model described in XML.

```
<entity name="Customers" tableschema="dbo">
  <attributes>
    <attribute name="CustomerID" type="string" />
    <attribute name="CompanyName" type="string" />
    <attribute name="ContactName" type="string" />
    <attribute name="ContactTitle" type="string" />
  </attributes>
  <keys>
    <pkey name="PK_Customers">
      <key name="CustomerID" />
    </pkey>
  </keys>
  <relationships>
    <relationship source="Customers"
      target="Orders" type="1..*" />
  </relationships>
</entity>
```

Figure 1. A XEML model described in XML

In XEML, models are called **entities**, the components of an entity are called **attributes**, the references to other models are called **relationships**, and the **keys** (both primary and foreign keys) are defined for database system. This section provides an overview of the most commonly used XEML tags.

#### 2.3.1 Syntax

##### Schema Tag

A SCHEMA is the outer most container of an XEML file. A schema is the model generally created for a business application. It defines the general and global information for the models.

```
<schema name="Order System" owner="Sales">
```

##### Entity Tag

An ENTITY is an object in the business applications that encapsulate specified data. For example, an entity model can be used to depict a company’s customer base. A customer entity has attributes-such as the name, the contact and the address and relationships to other entities-such as the order. In theory, if the parts of system can be identified, the system can be expresses as an entity model. Essentially, entities defined in XML become classes in the object-oriented application system or tables in the database.

```
<entity name="Customers" tableschema="dbo">
```

### Attribute Tag

An ATTRIBUTE represents structures that contain data. The ATTRIBUTE tags define the database entity fields or application object properties. An attribute of an entity may be a simple value, such as a scalar (for example, a string, an integer or decimal), but can also be a BLOB (for example, binary), or date and time (such as date, or timestamp).

```
<attribute name="CompanyName" type="string" />
```

### Key Tag

A PRIMARY KEY is used to uniquely identify each record within a database table. It consists one or more entity attributes.

```
<pkey name="PK_Customers">  
  <key name="CustomerID" />  
</pkey>
```

A FOREIGN KEY is a reference to a PRIMARY KEY in another database table. The foreign key consists of one or more entity attributes that are reference to fields in other tables.

```
<fkey name="FK_Orders_Customers">  
  <key name="CustomerID" parent="Customer" />  
</fkey>
```

### Relationship Tag

In object-oriented applications, not all properties of a model are attributes, your applications is typically modeled by multiple classes. At runtime, your object model is a collection of related objects that make up an object graph. The relationships between these model objects can be traversed at runtime to access the properties of the related objects.

```
<relationship source="Customer" target="Order" type="1..*" />
```

### 2.3.2 Extend the XEML Models

The beauty of XML is that it is text-based (therefore platform-independent) and extensible which allowing more tags be added if needed. For example, we can add a new ACTION tag to define the behavior of an entity model:

```
<entity name="Customers">  
  <attributes>...</attributes>  
  <actions>  
    <action name="makeOrder">  
      <input name="order" type="entity" />  
    </action>  
  </actions>  
</entity>
```

The following are the common tags to describe the OO and database system:

```
<entity name="Customer">  
  <attribute>...</attribute>  
  <key>...</key>  
  <relationship></relationship>  
  <action>...</action>  
  <event>...</event>  
  <message>...</message>  
  <rule>...</rule>  
</entity>
```

## 3. Tooling

The heart of model-driven development is modeling, but model is not just document or visualize the code, the most important is that final software product was automatically generated from these high-level specifications. Code generation is about writing programs that write programs. Tool support for creating and using models, languages and code generator is crucial for automating software development. On the other word, code generation is about tooling. The tools are not just helping you to model, but are helping you to speed your coding via systematic code generation.

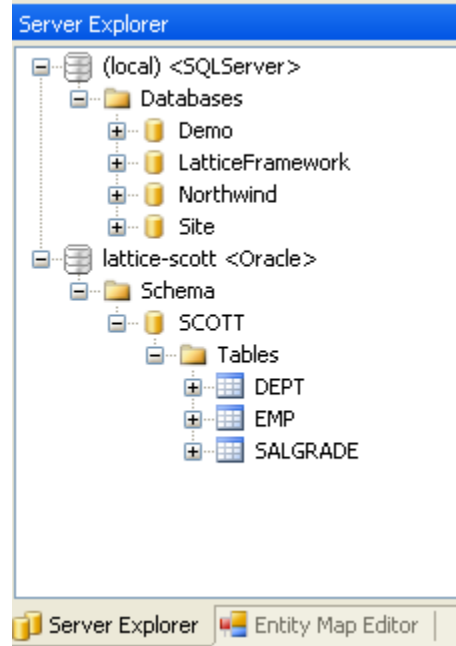
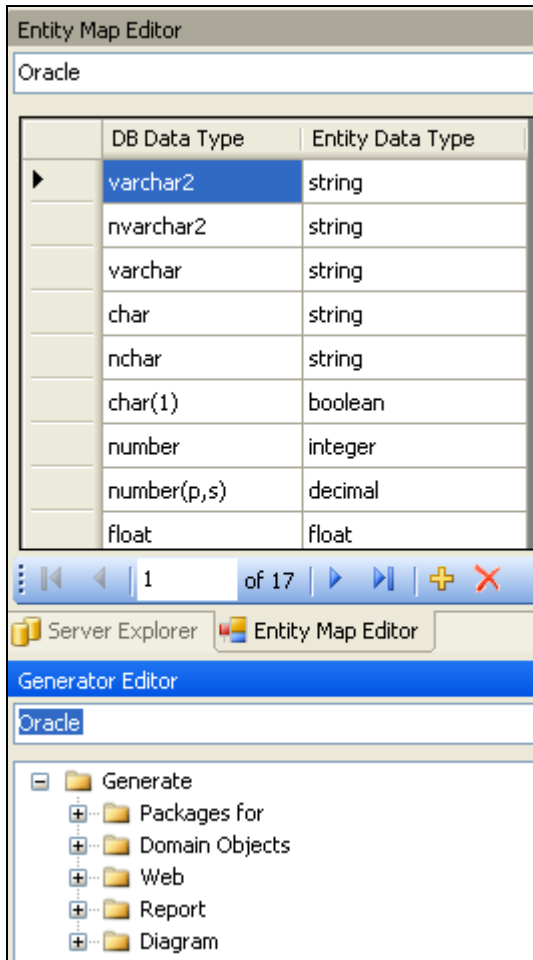


Figure 2. Database Server Explorer

LatticeFramework Studio is an integrated and rapid application development tool based on XEML entity models. LatticeFramework allows you to create and edit XEML entity models, or directly export XEML models from different database system, such as MySQL, Oracle, SQL Server or PostgreSQL, (see Figure 2) it allows you to easily creating new code generators using Generator Editor and map RDBMS data types to XEML data type using Entity Map Editor.

Figure 3 shows the Generator Editor and Entity Map Editor of LatticeFramework Studio.



**Figure 3.** Generator Editor and Entity Map Editor

LatticeFramework Studio has been developed using C# 2.0. LatticeFramework Studio takes XEML entity models as input and it uses different XSL templates to translate XML specifications into multiple artifacts in the area of UI, Business Logic, Data Access, documentation, and diagrams. One specification and multiple source files then generated. Some examples of artifacts that can be generated from these XEML entity models are:

- UI Components
- Domain Object in C#, java, PHP or whatever you prefer
- Database DDL and procedures
- Schema Reports in Word or HTML format
- Entity-Relationship and UML class diagrams

### Application Development Process

The application development process of using LatticeFramework Studio is essentially a three-step process - involving Application Modeling, Template Development and Customization - followed by automatic code generation:

1. Develop XEML entity model of the application

2. Develop code generation templates to generate an application from the model
3. Generate the code
4. Iterate and increment

The developer will follow a path of iteration through these three steps using an agile development process to achieve the desired application. Step 4 is an important step in this process. A continuous involvement of domain experts, architects and technical experts are needed to make model-driven development an agile process that can compete with any other agile method in productivity and efficiency.

### 4. Sample

To demonstrate the strength of XEML, we present a brief example project: an order-processing system. We focus on the XEML entity models definition and generator creation.

#### Place Order Use Case:

- 1) The use case starts when the customer selects Place Order
- 2) The customer enters his or her name and address.
- 3) While the customer enters product codes
  - a) The system will supply a product description and price
  - b) The system will add the price of the item to the total
- 4) The customer will enter credit card payment information.
- 5) The customer will select submit.
- 6) The system will verify the information, save the order as pending, and forward payment information to accounting system.
- 7) When payment is confirmed, the order is marked confirmed, an order ID is returned to the customer, and the use case ends.

The application will be implemented following typical layer architecture: UI layer, Business layer and Data layer. By using XEML and LatticeFramework Studio, it becomes possible to rapidly and reliably implement the application without having to code each layer manually.

#### 4.1.1 Defining the XEML Entity Models

There are two ways to define the XEML models: create it from the scratch using your favorite XML editors or Notepad or export the XEML models from existing database system.

The following steps show you how to define XEML models:

1. Define the entity and its attributes
2. Define the relationship among the models
3. Define the primary and foreign keys
4. Define the actions
5. Define the events

Figure 4 shows a portion of XEML models for order-processing system.

```

<schema name="Order-Processing">
  <entities>
    <entity name="Customer">
      <attributes>
      <keys>
      <relationships>
    </entity>
    <entity name="Order">
      <attributes>
      <keys>
      <relationships>
    </entity>
    <entity name="OrderLine">
      <attributes>
      <keys>
    </entity>
    <entity name="Product">
      <attributes>
      <keys>
      <relationships>
    </entity>
    <entity name="Catalog">
      <attributes>
      <keys>
      <relationships>
    </entity>
  </entities>
</schema>

```

6.

Figure 4. XEML models for Order Processing System

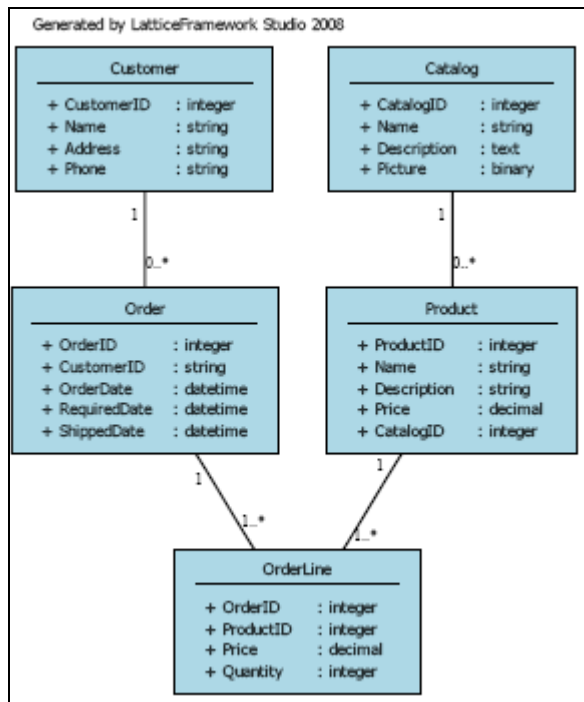


Figure 5. Generated Class Diagram

Based on XEML models, the ER diagram and Class diagrams can be generated. Also documentations in Microsoft Word and HTML format can be generated too.

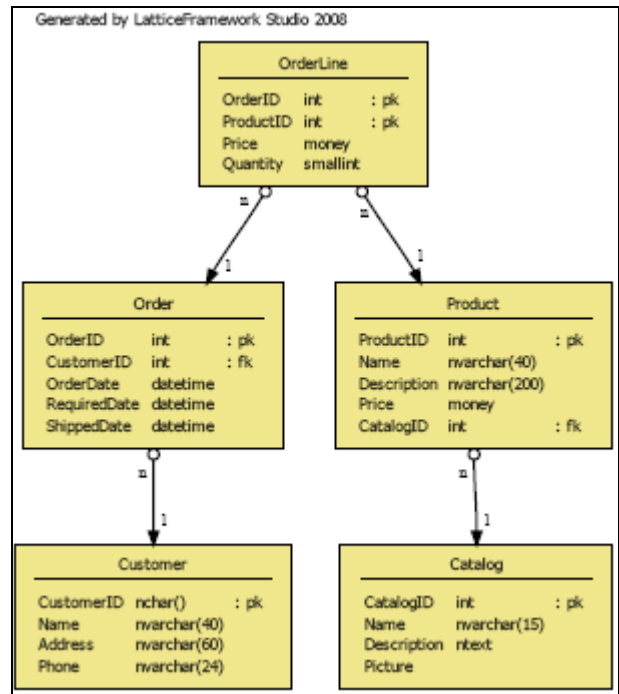


Figure 6. Generated Entity-Relationship Diagram

#### 4.1.2 Implementing the Code Generator

Since the XEML model is actually XML documentation, the XSLT templates are used to create different code generators.

With XSLT you have scripted access to the XML DOM, the XSLT processor parses the XML document to construct the DOM, and XSLT scripts manipulate the DOM input to construct the DOM output. It's possible to construct plaintext source code files from the DOM output. The plaintext output forms one to many source code files.

Another important advantage of using an XSL style sheet for translating the XEML model into code is that XSL is a declarative language. XSL is declarative (as compared to procedural) in the sense that it describes results of the translation rather than the steps to perform it. This pervades in a higher-level of abstraction.

These facts make the source code generation task extremely easy since the tasks of parsing, validating, and generating code can all be performed with freely available tools in the manner of rapid scripting-based development.

Figure 7 shows a portion of XSL template for generate domain objects in C#.

```

//
// Name: <xsl:value-of select="$entity"/>.cs
// Author: <xsl:value-of select="@currentuser"/>
// Date: <xsl:value-of select="@currentdate"/>
//
// The code was generated by LatticeFramework Studio
// Copyright © 2008 Lattice Business Software.
// All Rights Reserved.
//
//-----
#endregion
using System;
using System.Collections.Generic;

namespace LatticeFramework.Domain
{
    public partial class <xsl:value-of select="$entity"/>
    {
        #region private fields
        <xsl:for-each select="attributes//attribute">
        private <xsl:choose>
            <xsl:when test="@type="integer">int</xsl:when>
            <xsl:when test="@type="text">string</xsl:when>
            <xsl:when test="@type="boolean">bool</xsl:when>
            <xsl:when test="@type="binary">byte[]</xsl:when>
            <xsl:when test="@type="datetime">DateTime</xsl:when>
            <xsl:when test="@type="timestamp">DateTime</xsl:when>
            <xsl:when test="@type="guid">System.Guid</xsl:when>
            <xsl:otherwise><xsl:value-of select="@type"/></xsl:otherwise>
        </xsl:choose> _<xsl:call-template name="replace-string">
            <xsl:with-param name="text"
                select="@name"/>
            <xsl:with-param name="from" select="'"'/>
            <xsl:with-param name="to" select="''"/>
        </xsl:call-template>;</xsl:for-each>

        #endregion
    }
}

```

Figure 7 Template for generating domain object in C#

The screenshot shows a web browser window displaying a form titled "Customer form". The form has a header "Edit Form" and four input fields labeled "CustomerID:", "Name:", "Address:", and "Phone:". Below the fields are two buttons labeled "Save" and "Cancel".

Figure 8 Shows generated web 2.0 form (ExtJS) and the following shows a portion of generated domain object in C#.

```

//
// Name: Customer.cs
// Author: Julie
// Date: 7/31/2008 1:53:55 AM
//
// The code was generated by LatticeFramework Studio
// Copyright © 2008 Lattice Business Software.
// All Rights Reserved.
//
//-----
#endregion
using System;
using System.Collections.Generic;

namespace LatticeFramework.Domain
{
    public partial class Customer
    {
        #region private fields

        private int _CustomerID;
        private string _Name;
        private string _Address;
        private string _Phone;

        #endregion

        public Customer(){}

        #region public properties

        public int CustomerID
        {
            get { return _CustomerID; }
            set { _CustomerID = value; }
        }

    }
}

```

## 5. Conclusion

In this paper, we have described LatticeFramework Studio and XEML, a general-purpose modeling language for the specification and generation of artifacts for database and OO systems. All the features are implemented and available in LatticeFramework Studio.

## References

- [1] LatticeFramework Studio <http://www.latticesoft.com>
- [2] Amber, S.W. United or Domain-Specific Modeling Languages? Software Development's Agile Modeling Newsletter 2006
- [3] Greenfield, J., Short, K., Software Factories, Wiley, 2004.
- [4] Schneider G., Winters J., Applying Use Case, Addison-Wesley 1998
- [5] Czarnecki K., Eisenecker U., Generative Programming Addison-Wesley 2000
- [6] Chen, P.S., The Entity-Relationship Model—Toward a Unified View of Data ACM Transaction on Database System, Vol. 1, No. 1, March 1976
- [7] Tolvanen, J.P., Modeling for Full Code Generation Embedded Computing Design August 2007
- [8] Sarkar, S., Model-Driven Programming Using XSLT XML Journal August 2002