A Generation Approach of Transformation Code for Web Interchanging Documents

LE-LE TANG, FUYANG PENG
Beijing Institute of System Engineering
P.O.Box 9702-19, Beijing 100101
CHINA

Abstract: - This paper describes a generation approach of transformation code for web interchanging documents. This approach is based on model driven architecture and model transformation technology. We provide a model weaving tool named QMTW to define mapping rules between XML documents. All these rules are stored in a model named weaving model. A code generator is provided to read this weaving model and generate target XSLT code. When mapping rules change, developer only needs to modify the weaving model and run the code generator again. Our approach improves the quality of transformation code for web interchanging documents and reduces the work of maintenance.

Key-words: - Model Transformation, Web Application, UML, Model Driven Architecture, XSLT

1. Introduction

Nowadays, many web systems use XML as the interchanging format. In the process of data collecting, transforming and mining, many operations conforming to some semantics are always added on these XML-based documents. Under many circumstances, the results of these operations are temporary and need not be stored persistently. Therefore, web interchanging documents are not collected and transformed by traditional database technology. XSLT is always used to do these works instead.

Transforming these XML-based data by XSLT is an effective way, and supported by many tools. However, writing an effective XSLT code requires advanced programming skills and good understanding of XML's working mechanism. At the same time, writing XSLT code by hand is an elaborated, error-prone and hard-to-maintained work. Researchers have realized the importance of the XML technology and the need for automatic transformation approaches[1].

Inspired by MDA[2] and model transformation technology, we propose a model transformation based approach to generate XSLT code. This approach can define mapping rules between different XML documents conveniently, and generate executive XSLT code based on these rules. It improves the quality and maintainability of XSLT code, and makes the programmer's work easier.

The remainder of this paper is organized as follows. The next section introduces the related works. In section 3, we describe the process of our approach briefly. We provide a case study to demonstrate every steps of this approach in details in section 4. In the last section, we point out the future work and give the conclusions.

2. Related Works

Now that many of us depend on Web-based systems, they need to be reliable and perform well. To build these systems, Web developers need a sound methodology, a disciplined and repeatable process, better development tools, and a set of good guidelines. The emerging field of Web engineering fulfills these needs[3]. In recent years, many researchers in web engineering domain provide

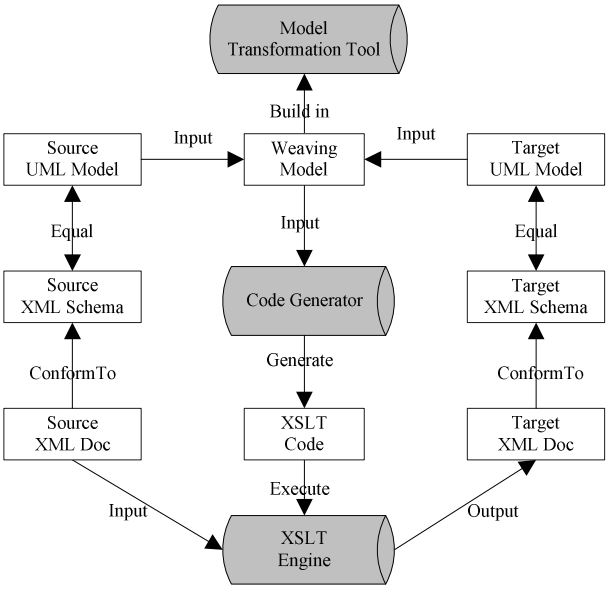
model driven development approaches for constructing web applications.

In [4], Piero Fraternali and Paolo Paolini describe a methodology for the development of WWW applications and a tool environment specifically tailored for the methodology. The methodology and the development environment are based upon models and techniques already used in the hypermedia, information systems, and software engineering fields. An approach to support the development of large-scale Web applications is described in [5]. Large development efforts have to be divided into a number of smaller tasks of different kinds that can be performed by multiple developers. They also implement a tool which provides a variety of code generators and a mechanism for checking whether view artifacts are compliant with the model.

The methods mentioned above demonstrate how to develop an entire web application, but pay little attention to web interchanging documents. Our approach solves the problem of how to generate transformation code for web interchanging documents.

3. Model Transformation Approach

We illustrate our model transformation approach for XSLT code generation in Figure 1.



Figrue1 Model Transformation Approach

As illustrated in Figure 1, the approach has three steps. First, we must get the source and target conceptual model. Second, we define the mapping rules. At last, we use the code generator to generate the target XSLT code.

The first problem is how to construct the conceptual model for XML documents. There are three solutions for this problem: (i) adopt UML to design XML schema; (ii) use an extended ER model to design XML schema; (iii) use a special modeling approach for XML—AOM(Asset Oriented Modeling)[6]. Considering practicability, we choose the UML as the modeling technique.

If source XML documents or target XML documents have not XML schema, we can use UML tool to build the conceptual models and generate the XML schema. Otherwise, we can generate UML models from XML schemas. Some tool such as hyperModel[7] has the transformation ability between UML models and XML schemas. In this way, we can get the source and target model which are UML class diagrams. Thus the web interchanging XML documents conform to these UML models.

The second step is to define the mapping rules between source and target model by a model weaving tool--QMTW and to store these rules into a weaving model. Put the weaving model into a code generator, we can get the target XSLT code.

Pay attention, the weaving model containing the mapping rules can be stored as the artifact of our approach. When the mapping rules change, we only have to change the weaving model, and the XSLT code can be generated automatically. Next section, a case study is provided to demonstrate the details.

4. Case Study

4.1 Motivating Example

For example, a web bank system interchanges its data with a web payment center. The conceptual models of source and target data are illustrated in Figure 2. The mapping rules are also included in this figure.

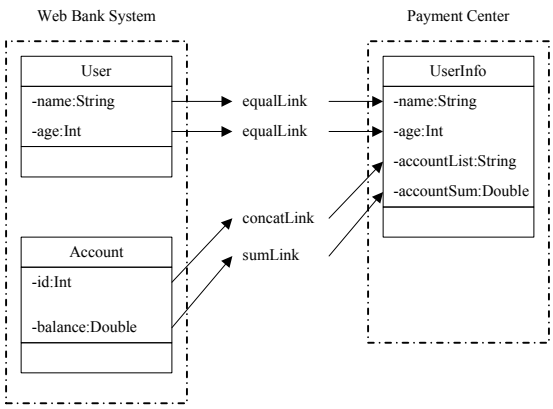


Figure 2 Source and Target Models

We can see from the figure that the bank system includes several User objects which contain a name and an age attribute. Every User object has at least one Account object which contains an id and a balance attribute. The payment center system includes several UserInfo objects which contain name, age, accountList(this attribute display all available accounts belong to this user) and accountSum(this attribute is the sum of all balances in available accounts) attributes.

4.2 Defining Mapping Rules

We have proposed a model weaving tool QMTW (QVT-Based Model Transformation Weaving Framework) to define model transformation rules. Model weaving is a special model transformation technique which defines the mapping rules as typed links and generates model transformation code for these rules. In another word, model weaving is a code generator of model transformation code.

The mapping rules in QMTW are represented by link, every link defines the relation between source model element and target model element. A link contains three parts: source model element, target model element and link type. Every link type has a corresponding code fragment. OMTW provides a link type library including many types which can be selected by user. User can also add customize link type into the library to fulfill new requirements.

By using QMTW, we define four rules shown in Figure 2. The description of these rules is listed below:

The value of name attribute in class User is equal to the value of name attribute in class UserInfo;

The value of age attribute in class User is equal to the value of age attribute in class UserInfo;

The value of accountList attribute in class UseInfo is equal to the concatenation of all the value of id attribute of Account in a User while removing repeat values (if exist) and adding a '/' between every two values.

The value of sum attribute in class UserInfo is equal to the addition of all the value of balance attribute of Account in a User.

These rules are represented by four links in QMTW. The link types are equalLink, concatLink and sumLink separately.

Source model, target model and all links are stored in a weaving model which is the input of code generator.

4.3 Code Generation

There are two steps in code generation process. First, XSLT code framework including XSL element and XSL attribute is generated according to the target model. This work is completed by framework code generator. The values of XSL attributes are computed by calling special XSLT templates which will be generated in next step. The algorithm of framework code generator is as follows. Considering simplicity, we only give the algorithm for one package.

```
Void GenerateClassCode(Class class)
{
For every Class = classA in Package{
  Write <xsl:element> into target code;
  For every Attribute = attributeA in classA{
        If attributeA is a primitive type {
            Write <xsl:attrivute> into target code;
            Add other params into code;
            If a rule = linkA is linked with attributeA
                Add <xsl:call-template name=
    "$classA_attributeA_linkA"/>
            Write </xsl:attribute> into target code;
        }
        If attributeA is a object of classB
            GenerateClassCode(classB);
  }
  For every associated Class = classC{
     If classC is aggregated by classA
        GeneratedClassCode(classC);
     If classC is navigated by classA
            Write a <xsl:ref> into target code;
            Add other params into code;
            Write a </xsl:ref> into target code;
  }
  Write </xsl:element> into target code;
}
}
```

From the framework code we can see that the XSL element and XSL attribute define the objects and attributes of the target model. But to get the value of attributes, we need to call some XSL templates such as $classA_attributeA_linkA template. The bodies of these templates will be generated and added in the second step.

The second step is to scan all mapping rules in the weaving model and to construct the bodies of XSL templates. Every link type has a code fragment stored in the link type library. Code generator picks out these code fragments and configures their parameters to build entire XSL templates. For example, the code fragment of sumLink is listed as follows:

```
<xsl:template name="$sumLink_name ">
<xsl:value-of
select="sum($srcClass/@$srcAttribute)"/>
</xsl:template>
```

By inputting concrete parameters into $sumLink_name, $srcClass and $srcAttribute, we can get the XSL template for sumLink. After all these templates are completed and added into the code framework generated in the first step, then we can get the final XSLT code.

4.4 Executing Transformation

When the code generation work is completed, the XSLT code then can be put in any place user wanted to transform source XML documents into target XML documents.

We construct a source XML document to verify whether the generated XSLT code can work or not. The source document is as follows:

```
<User name="TangLeLe" age="28">
    <Account id="1001" balance="122.0"/>
    <Account id="1002" balance="123.0"/>
    <Account id="1003" balance="144.0"/>
</User>
```

After the transformation work, we get the document listed below:

```
<UserInfo name="28" age="TangLeLe"
accountList="1001/1002/1003"
sum="389">
</UserInfo>
```

By checking the relations between source and target document, we can see that they conform to the mapping rules defined in section 4.2.

5. Conclusions and Future Works

The contributions of this paper are: (i) we provide a special model transformation tool which can define mapping rules between web interchanging documents conveniently. (ii) XSLT code can be generated automatically according to mapping rules. It reduces the burden of programmer and improves the quality of code. (iii) When mapping rules change, user only needs to modify the weaving model, and then new XSLT code could be generated automatically. In this way, the maintainability of code is improved.

But there are still some limitations in our approach. (i) Some web interchanging documents can not be represented by XML schema. In this case, our approach can not be applied on these documents. (ii) Since the description ability of UML is a little weak, some sophisticated XML schema can not be modeled with UML.

In the future work, we will extend UML to fulfill more sophisticated XML schema. We will find the common data structures used in web interchanging documents and summarize some patterns to facilitate the work of modeling and code generating.

References:

[1] G.L. Song, K. Zhang, J. Kong. *Automatic Generation of Transformation Rules for Model Management*. In: *VL/HCC'05 Workshop on Visual Modeling for Software Intensive Systems*. Dallas, USA 2005

[2] OMG. *MDA Guide Version 1.0.1*. 12th June; omg/2003-06-01] 2003. Available from: http://www.omg.org/cgi-bin/apps/doc?formal/03-06-01.pdf.

[3] Murugesan, Athula Ginige and San, *Web Engineering: An Introduction*. Multimedia, IEEE,Jan-Mar 2001. Vol.**8**(1): p. 14-18.

[4] Piero Fraternali, Paolo Paolini, *Model-driven development of Web applications: the AutoWeb system*. ACM Transactions on Information Systems (TOIS) October 2000. Vol.**18**(4): p. 323 - 382.

[5] H. Tai , K. Mitsui , T. Nerome , M. Abe , K. Ono , M. Hori, *Model-driven development of large-scale web applications*. IBM Journal of Research and Development,September/November 2004. Vol.48(5/6): p. 797 - 809.

[6]*Asset Oriented Modeling*. Available from: http://www.aomodeling.org/.

[7] *hyperModel*. Available from: http://www.xmlmodeling.com/hyperModel/.