

代码生成技术在 MDA 中的实现^{*}

陈 翔, 王学斌, 吴泉源

(国防科学技术大学 计算机学院, 湖南 长沙 410073)

摘 要: 针对如何将具体的业务模型转换为应用程序代码的问题, 提出了一个采用 XSLT 技术的代码生成器设计方案, 用来实现 MDA 中的自动代码生成。从介绍代码生成器的输入文件入手, 讨论了代码生成器的处理原理, 最后举了一个采用 XSLT 技术的代码生成器生成代码的例子。

关键词: 代码生成器; XSLT; XML; XMI; XPath; 棱型驱动框架

中图法分类号: TP311 文献标识码: A 文章编号: 1001-3695(2006)01-0147-04

Code Generating Implementation in Model Driven Architecture

CHEN Xiang, WANG Xue-bin, WU Quan-yuan

(School of Computer, National University of Defense Technology, Changsha Hunan 410073, China)

Abstract: Aiming at the problem how to transform model to application code, the paper bring forward a project about code generator using XSLT, and the project can implement code generating in model driven architecture. In this paper, first introducing inputing documents of code generator, second discussing code generation processing phases, third illustrating transforming model to code using code generator.

Key words: Code Generator; XSLT; XML; XMI; XPath; MDA (Model Driven Architecture)

在软件的开发中, 开发人员往往要面对重复编写一些简单的、令人乏味的代码, 而且每当新技术来临的时候, 又不得不再重复过去的工作。开发的系统很少是用单一的技术构建的, 它总是需要同外界交流, 需求的变化也从来没有停止过。如果能够定义一个抽象的、独立于任何技术的模型, 再通过代码生成器生成相关的框架程序或者部分具体技术的业务代码, 那么程序员就可以把精力集中在创造性的开发上。

这就是 OMG (Object Management Group) 组织提出的 MDA^[1] (模型驱动框架) 思想的起源。OMG 提出, 一个 MDA 的应用, 是建立以平台无关的 PIM (Platform-Independent Model) 开始, 然后使用相应的 MDA 工具, 结合具体技术, 将 PIM 映射成为特定平台上的 PSM (Platform-Specific Model), 最后通过这个应用的特定 PSM, 使用代码自动生成器, 自动生成大部分的特定平台上的应用程序代码。MDA 的提出改变了软件设计、实现和维持的方法, 它提供了一种比中间件层次更高的、更抽象的解决方案, 使将来的互操作和移植变得更加容易和快速, 并且把从模型到代码的自动生成提高到更高的层次。

本文讨论了代码生成技术在 MDA 中的实现^[2]。如图 1 所示, 获取一个特定领域具体业务模型需求, 使用 UML 编辑器编辑特定领域的业务逻辑和数据结构, 生成用 UML 语言描述的具体业务模型, 再结合特定平台的附加信息和技术的要求, 通过 XMI 转换规则生成所需要的 XML 文件。而同时, 对于具体业务模型中的一些固有的程序代码框架和系统配置, 可以使用 XSLT 编辑器生成 XSLT 程序模板。最后再通过 XSLT 处理器, 将描述具体业务模型的 XML 与 XSLT 程序模板结合, 就可

以把这个具体业务模型映射成为特定平台上的应用程序代码, 从而实现快速地开发系统。一旦系统的需求或者技术有所改变, 只需要在元数据模型上进行修改, 再次使用代码生成器就可以生成不同的应用程序。这样就节约了重复的代码工作, 使得开发人员把精力集中到元模型的描述和代码生成器的使用上面^[3]。

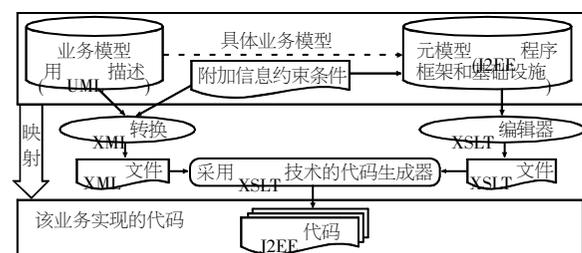


图 1 具体业务模型到应用代码的映射

1 代码生成器的输入

要实现将一个模型通过代码生成器映射为所需要的代码, 首先要将某个具体业务模型转换为代码生成器可以读取的文件, 然后代码生成器从这些输入文件中提取出模型中的信息并生成系统应用代码。通过比较和研究, 提供给代码生成器作为输入文件的最好方法就是使用一种统一的、容易验证的而且能够完整描述模型信息的数据格式, XML 文件和 XSLT 模板程序正好提供了这一切。为此, 我们把具体的业务模型分为与技术无关的业务模型和与技术相关的元模型两部分。业务模型采用 UML 类图表示, 以 XML 文件形式存储模型信息, 元模型是抽取具体业务处理过程共性的结构, 它以 XSLT 程序模板作为存储方式。

2 技术无关的业务模型——XML 文件

采用 UML 表示业务模型,是因为 UML 的最大长处在于对系统的结构方面的建模,它可以将业务模型以可视化的形式直观地描述在开发者面前,这在很大程度上是通过使用类模型来完成的。尽管 UML 的类模型能够很好地描述一个系统的具体业务模型,但是它没法作为代码生成器所能够读取的输入文件,所以最终还是要以 XML 的模式向代码生成器提供输入的数据。XML 模式是文档的模式,它和良好的 UML 模型在复杂程度上并没有显著的区别。为此可以把 XML 模式看作是从高层业务模型(UML)到底层应用代码生成的连续统一体中的中间部分,由它来作为代码生成器的输入文件,存储业务模型的信息。

XMI 试图通过 XML 语言为程序员和其他用户提供一种交换元数据信息的标准途径^[4]。它希望能够帮助使用各种语言和开发工具的 UML 开发人员自由地交换数据模型,可以把它当作一种将 UML 映射到 XML 模式的标准方法。XML 元数据交换旨在能够在建模工具之间方便地交换元数据,而且可以进一步用来根据一组转换规则从 UML 图生成 XML,如图 2 所示。

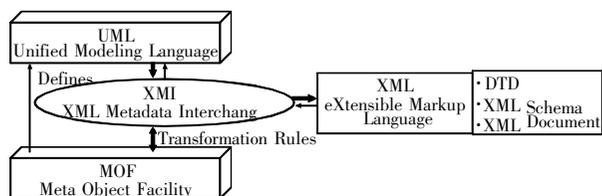


图 2 采用 XMI 实现从 UML 到 XML 的转换过程

把 UML 结构映射到 XML 可以有多种不同的选择,只要在 UML 结构和 XML 模式语句间建立一对一的映射,这个过程很容易实现自动化,而且可以找到用于从模型自动派生模式的许多工具。在实际代码生成技术实现的项目中,我们是采用了 Eclipse 建模框架(EMF)^[5],用它来实现从 UML 模型派生成 XML 模型。

3 技术相关的元模型——XSLT 模板

对于具体业务模型中元模型部分的抽取,我们分析一个业务功能的开发所要涉及到的各种层次,各个对象之间的接口关系以及业务的处理中所有共性的东西:业务层次结构、业务处理流程、业务对象模型、界面构件等,将这些共性信息抽取出来作为元模型,再编辑成代码生成器可以处理的 XSLT 模板。整个元模型抽取的结果为两个方面:

(1) 程序模板。常用的业务处理的程序框架,可编辑成代码生成器可处理的 XSLT 模板程序。

(2) 基础设施。存储业务模型的扩展信息和包含程序模板运行的特定支撑环境。如基于 J2EE 环境,分为三部分:Java 类库、Web 运行支持(包括配置文件、JavaScript 脚本、XML 支撑文件等)、业务运行配置(数据库方式,或者文件方式)。也可编辑成代码生成器可处理的 XSLT 模板程序。

4 代码生成器的原理

一个典型的代码生成的处理原理为:输入文件通过语法分

析转换为输入文件的语法分析树,语法分析树再通过代码生成器的分析,转换成生成代码的语法分析树,这样代码书写器就可以通过生成代码的语法分析树获取输入文件的信息,生成所需要的代码。图 3 是一个代码自动生成的处理原理^[6]。

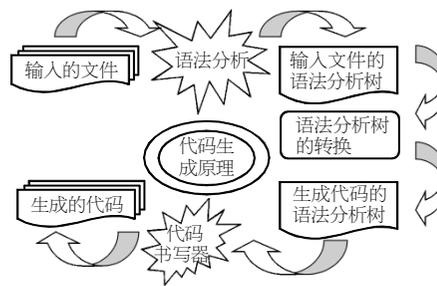


图 3 代码生成器的原理

在这个处理过程中,采用 XSLT/XML 技术与采用传统技术的代码自动生成器相比,前者能够更好地实现从模型到代码的转换并且促使代码生成器技术的发展。

(1) 语法分析器。在这儿输入文件是描述业务模型 XML 文件。XML 文件的语法分析可以直接采用 XSLT 处理器中的语法分析器。而对于采用传统技术的语法分析器,我们则需要使用专门的语法分析器,或者懂得如何使用语法分析工具如 LEX/YACC 来创建语法分析的框架。

(2) 语法分析树。XSLT 处理器创建自己的语法分析树结构,并且利于 XPath 语言提供脚本程序访问的接口。作为使用者,不需要担忧要由自己来创建语法分析树结构。对于使用传统技术的语法分析器,开发者就必须先创建自定义的语法分析树结构,再对输入文件进行语法分析,把结构存入到自定义语法分析树结构中。

(3) 语法分析树的转换。XSLT 处理器通过 XPath 语言提供获取语法分析树节点上值的接口,它直接执行语法分析树的转换过程,而不需要开发者进行设计,开发者唯一要做的事情就是懂得使用 XPath 的语言。换句话说,XSLT 的使用者书写 XSLT 和 XPath 模板来执行对语法分析树的转换。另一方面,对于使用 XSLT 程序设计转换程序,是在一个较高的高度,即明白需要转换成什么样的语法分析树。尽管这样,程序设计者绝对不用关注语法分析树的数据结构实现的详细信息,降低了开发难度。

(4) 代码书写器。将生成的代码以各种文件形式输出。XSLT 处理器也很好实现了这个功能。

总之,与传统的代码生成器相比,采用 XSLT 技术的代码生成器,对于代码生成器自身的系统结构就不需要过多关注,可以把精力集中到书写代码生成的逻辑上,实现更完善的代码生成功能。

采用 XSLT 实现代码生成器,还具有以下的优点:采用 XSLT 和 XPath 写代码生成器的模板是非常简单的,XPath 是一种 XSLT 应用程序所特有的 XML 路径语言,是 W3C 用于确认和选择部分 XML 文档的通用语言标准。XPath 是作为 XSLT 的对 XML 文档各部分进行定位的语言,它给 XSLT 提供一个整合的定位语法,用来定位 XML 文件中各个部位。XPath 除了提供一套定位语法之外,还包括一些函数,它们提供基本的数值运算、布尔运算和字符串处理功能。

XSLT 是针对转换的扩展样式表语言^[8],它的转换标准本

质上来讲是一个转换机制,它让你指定将一个 XML 标记转换成什么并用来显示。根据提供的一个或多个程序模板,代码生成器可以连续地结合输入的 XML 文件,生成所需要的代码。而且在反复生成代码的过程中,可以不断获取反馈信息,纠正代码生成器的输入文件和模板,从而保证了代码生成器的正确性。XSLT 描述的模板甚至可以根据一些先前声明好的语句,确认输入的文件的有效性。

5 代码生成器的实现

在从具体业务模型映射到实际代码的过程中,采用 XSLT 技术的代码生成器读取对应具体业务模型的 XSLT 程序模板文件,读取描述具体业务模型信息 XML 文档,通过 XSLT 处理器,生成我们所需要的 Java 源代码^[7]。XSLT 规范文档本身是一个 XML 文档,它用来描述所生成程序的模板,在需要替换的地方以 XSLT 特有的元素形式标志出来,通过 XSLT 处理器可以读取 XML 文档信息进行替换。这样即保持了将要生成程序的结构上的美观,又可以实现所需要的代码生成功能。当然对于较为复杂的替换,就需要在 XSLT 文档中引入复杂的 XPath 语言,它可以从 XML 文档中确认选择出所需要替换的信息。图 4 就是一个简单的通过代码自动生成器读取 XML 文件结合 XSLT 文档转换为 Java 代码的过程图^[9]。

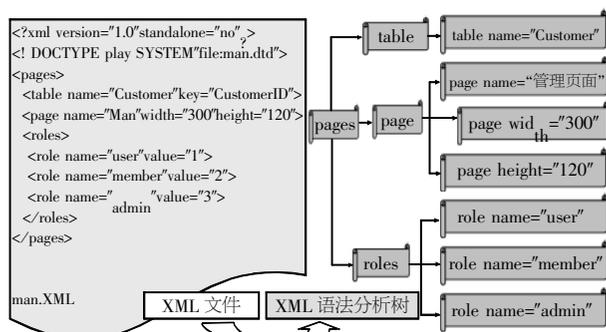


图 4 XML 文件和转换生成的 XML 语法分析树

图 4 中的 XML 文件 man.XML 是以元素和属性描述了一个客户管理页面信息,XML 的命名空间、元素、属性、设置构成了完整的 XML 特定应用域的信息。这些 XML 文件描述的约束语言是 Java 语言。

- (1) <! DOCTYPE play SYSTEM " file: man. dtd" > 用于指定验证数据的有效性的 DTD 文件名。
- (2) < /pages > 元素属性名,表示这个是页面。
- (3) < table > 的元素属性表示这个页面所读取的数据库表名和主键。 < page > 的元素属性表示页面的名称、宽度和长度。
- (4) < role > 的元素属性定义的客户角色。

图 4 还描述了 XSLT 处理这样一个 XML 文件的结果,通过语法分析器,XML 文件信息被转换为 XML 语法分析树的结构。有了这样一个形象表示 XML 文件的语法分析树,模板程序中的 XPath 语言就可以非常容易在其中定位所需要的 XML 元素,方便 XSLT 处理器进行转换。

下面介绍其中 XSLT 脚本的部分语法:

- (1) 前两行是 XSLT 模板脚本的声明,与生成的代码没有联系。

- (2) < xsl: output method = " text" / > 行表示生成的代码是以文本形式保存的。

- (3) < xsl: template match = " /" > 行指示 XSLT 处理器从语法分析树的根节点开始处理,并且将这个规则应用到所有的该 "template" 标志的脚本中。

- (4) < xsl: value-of select = " //table/@ name" / > 指示 XSLT 处理器寻找语法分析树中 < table > 元素下 name 的命名属性值,将其返回替换该行,并且输出到 . Java 中。

- (5) < xsl: for-each select = " //table/roles/role" > < /xsl: for-each > 是用来重复选择语法分析树中 role 节点的所有属性。

图 5 中 XSLT 文件 Page.XSL 描述了采用 XSLT 描写的 Java 固定代码的模板。

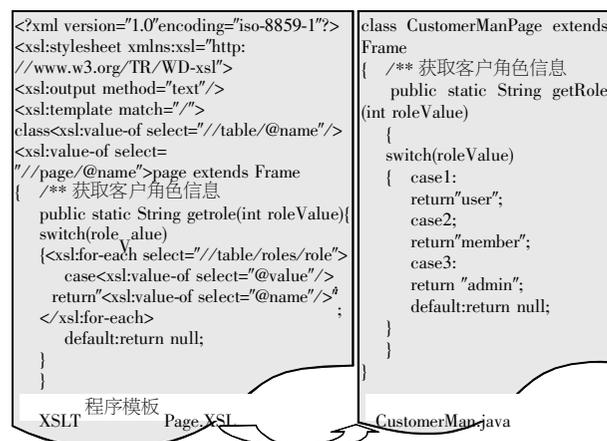


图 5 XSLT 程序模板和相应生成的 Java 代码程序

图 6 在此共有三个 role 节点被找到。图 6 描述的是一个使用 XSLT 技术的代码生成过程。

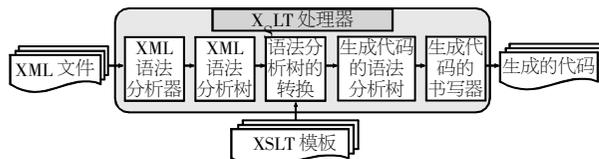


图 6 采用 XSLT 技术的代码生成处理过程

它的具体步骤如下:

- (1) 分析具体业务模型中与技术无关的业务模型,获取完整的描述模型的 XML 文件信息。所获取的一个 XML 文件的例子如图 4 所示。
- (2) 分析具体业务模型中技术相关的元模型部分,通过 XSLT 编辑器编辑成为包含业务模型技术实现的 XSLT 模板程序。所获取的 XSLT 模板程序例子如图 5 所示。
- (3) 通过 XSLT 处理器创建一个 XML 的语法树结构,如图 4 所示。XML 语法分析树使得 XSLT 和 XPath 很容易地定位语法分析树,获取所需要的 XML 元素。
- (4) XSLT 处理器获取 man.XML 文件的语法分析树,根据 Page.XSL 模板的脚本,就可以快速生成 CustomerPage.Java 文件,如图 5 所示。生成的 Java 文件片段是一个描述客户管理页面的类,它继承了 Frame 基类的方法和属性,记载管理页面所需要的页面属性如高度(Height)、宽度(Width)等,并且提供该页面的动作语义。该类中有个获取客户角色的方法。在添加客户管理信息时,如果需要填写客户角色的信息,可在下拉可选框的点击事件中调用该方法获取客户角色。

这样一来,只要输入 XML 语言文件, XSLT 模板就可以结合输入的文件,共同生成各种形式的输出文件。支持 XSLT 语言标准的处理器将 XML 文件转换为 XML 文件或者其他文本文件。这里顺带介绍几个开源的 XSLT 处理器,如 SAXON XSLT 处理器、Microsoft XML Parser³,在这里笔者所采用的是 Apache Xalan XSL 处理器^[10]。在配置好 Xalan 中 xalan.jar, xml-apis.jar, xercesImpl.jar 这些 jar 包的环境参数 classpath 后,在 DOS 窗口下键入如下的命令就可以运行 XSLT 脚本生成所需要的代码:

```
Java org.apache.xalan.xslt.Process -IN foo.xml -XSL foo.xsl -OUT
foo.Java
```

该语句的意思是:执行 XSLT 处理器 org.apache.xalan.xslt.Process,读取 man.XML 文件和 Page.XSL 文件,生成 CustomerPage.Java 文件。

6 结束语

本文主要是讨论了代码生成技术在 MDA 中的实现。采用 XSLT 技术的代码生成器,可以方便地将描述业务模型的 UML 转换为 XML 文件,并且能够对这些 XML 文件进行准确的语义检查,同时读取根据业务模型实现实例抽取出的共性的 XSLT 程序模板,通过 XSLT 处理器生成简洁完整的源代码。如果所提供的业务模型信息足够完整,各个业务处理逻辑正确,通过这个代码生成器映射出来的代码可直接编译运行,实现基本的增,删,改等业务操作。当然,这样设计的代码生成器也有它的不足,首先是这样的基于 MDA 的代码生成器,需要有能够完整详细描述系统业务逻辑和数据的元数据模型,这就需要有大量的开发经验和优秀的设计师支持。其次, XSLT 的

模板过于冗长,学习起来起点较高,能够熟练撰写 XSLT 模板的开发者较少。但是不可否认,采用 XSLT 技术的代码生成器使得模型到代码的转换变成可能,并且生成的程序不再只是代码框架,而是对应于模型的完整的应用程序代码,这也极大地扩展了代码生成技术的应用领域。

参考文献:

- [1] oaquin Miller, Jishnu Mukerji. MDA Guide version 1. 0. 1 [EB/OL] . <http://www.omg.org/docs/omg/03-06-01.pdf>, 2003-06-12.
- [2] OMG. Model Driven Architecture (MDA) [EB/OL] . Available at Java and XSLT Eric M. Burke O'Reilly & Associates Inc. , 2003.
- [3] David S Frankel. 应用 MDA [M] . 鲍志云. 北京:人民邮电出版社, 2003.
- [4] Birgit Demuth Heinrich Hussmann, Sven Obermaier. Experiments with XMI Based Transformations of Software Models[R] . 2003.
- [5] OMG. Meta Object Facility (MOF) Specification version 1. 4 [EB/OL] . <http://www.omg.org/docs/fomal/00-04-03.pdf>, 2002-04.
- [6] Soumen Sarkar. Model Driven Programming Using XSLT[R] . 2002.
- [7] O'Reilly. Java & XSLT[M] . 2003.
- [8] The Latest XSLT Specification. 国际互联网联盟的最新 XSLT 规范 [EB/OL] . 2004-05.
- [9] J Craig Cleaveland. 用 XML 与 Java 创建程序生成器[M] . 胡俊. 美科海电脑技术出版社, 2003.
- [10] Apache Xalan XSL 处理器的命令集 [EB/OL] . <http://xml.apache.org/xalan-j/commandline.html>, 2004-05.

作者简介:

陈翔(1979-),男,硕士研究生,主要研究领域为分布式计算、软件工程;王学斌(1978-),男,博士研究生,主要研究领域为分布式计算、软件工程;吴泉源(1942-),男,教授,博士生导师,主要研究方向为智能软件与分布计算。

(上接第 53 页)另外,如果某一关键词出现在文档的各级标题中,那么该关键词的权值将比出现在正文中的关键词权值大,并且权值将按标题的不同层次从高到低依次减少,即出现在一级标题的关键词模式的权值最大,出现在二级标题中的关键词模式的权值相对较小,以此类推,层次越深的标题中出现的关键词模式的权值依次减少,而正文中出现的关键词模式权值最小。

文档的知识描述是实施有效知识管理的关键。提出基于知识模式的文档描述构建方法。为知识管理提供了一种有效的方法与技术。

参考文献:

- [1] teffen Staab, Rudi Studer, Hans-Peter Schurr, et al. Knowledge Processes and Ontologies [J] . IEEE Intelligent Systems, 2001, 16 (1) : 26-34.
- [2] Gerhard Fischer, Jonathan Ostwald. Knowledge Management: Problems, Promises, Realities, and Challenges[J] . IEEE Intelligent Systems, 2001, 16(1) : 60-72.
- [3] Dieter Fensel. Ontology-based Knowledge Management [J] . IEEE Computer, 2002, 35(11) : 56-63.
- [4] Jay Liebowits. Knowledge Management and Its Link to Artificial Intelligence[J] . Expert Systems with Application, 2001, 20: 1-6.
- [5] Yogesh Malhotra. Expert Systems for Knowledge Management: Crossing the Chasm between Information Processing and Sense Making [J] . Expert Systems with Application, 2001, 20: 7-16.
- [6] David G Schwartz, Dov Te'eni. Tying Knowledge to Action with kMail[J] . IEEE Intelligent Systems, 2000, 15(3) : 33-39.

作者简介:

苏健(1974-),男,讲师,博士,主要从事知识管理、数据挖掘、决策支持等领域的研究;钟晴江(1965-),男,讲师,硕士,主要从事软件工程、系统集成等领域的研究。

标题模式定义如下:

```
< 标题 > :: = ( < space > | < newline > ) * < preTitle > < space > *
{ < titleString > } < newline > *
```

其中, < titleString > 是需要返回的具体标题字符串, < preTitle > 是标志标题级别的特征字符串,以下是相关定义:

```
< preTitle > :: = ( 第 < titleNumber > ( 章 | 节 | 点 ) < dot > ? )
| ( < titleNumber > < dot > ? ) | ( < leftBracket > < titleNumber > < rightBracket > )
< titleNumber > :: = < number > | < chineseNumber > | < englishNumber >
< number > :: = ( [ 0 - 9 ] < dot > ? ) | ( [ 1 - 9 ] + [ 0 - 9 ] * ) |
( [ 0 - 9 ] < dot > [ 0 - 9 ] * ) | ( [ 1 - 9 ] + [ 0 - 9 ] * < dot > [ 0 - 9 ] * )
< englishNumber > :: = ( [ a - zA - Z ] * < dot > ? ) | ( [ a - zA - Z ] * < dot > [ a - zA - Z ] * )
< chineseNumber > :: = < CN > | ( < CN > + < CN > ) | ( < CN > 百 < CN > + < CN > )
< CN > :: = [ 一 | 二 | 三 | 四 | 五 | 六 | 七 | 八 | 九 | 零 ] ;
```

5 结束语

知识管理的对象大多数是弱结构文档,而建立这些弱结构