一种结合 MDA 的高阶模型转换方法

曾　一，许　林，黄兴砚，王翠钦
（重庆大学 计算机学院，重庆 400030）

摘　要：模型转换是 MDA 的关键技术，也是 MDA 的研究热点。目前，不同的 MDA 开发平台都有一套相对独立的开发技术和转换框架，这使平台之间缺乏兼容性，模型转换代码重用困难。究其原因是缺少一种与具体转换语言相对应，且与平台无关的转换规则模型。为了解决以上问题，将高阶模型转换的思想与模型驱动软件开发相结合，提出了一种构造模型转换规则的高阶转换元模型，并以 ATL 语言为例展示了高阶转换元模型的使用方法；最后通过一个实例验证了该方法的可行性和可用性。该方法提高了模型转换语言的抽象层次，降低了模型转换语言的重用难度，在一定程度上解决了模型转换技术不兼容的问题。

关键词：模型驱动架构；高阶模型转换；模型转换；ATL 元模型
中图分类号：TP311.5　　**文献标志码**：A　　**文章编号**：1001-3695(2012)12-4584-05
doi:10.3969/j.issn.1001-3695.2012.12.048

Method of higher-order model transformation combined with MDA

ZENG Yi, XU Lin, HUANG Xing-yan, WANG Cui-qin
(*College of Computer Science*, *Chongqing University*, *Chongqing* 400030, *China*)

Abstract: Model transformations are the core and research focus of MDA. Nowadays, the MDA development platforms have a set of relatively independent development techniques and transformation frameworks, which causes incompatibility between different platforms and difficulty in reuse of model transformation code. The reason lays on the lack of a platform-independent model of transformation rules corresponding to specific transformation language. To solve these shortcomings, this paper combined higher-order model transformation with model-driven software development, and proposed a kind of general higher-order transformation metamodel which described the model transformation. Furthermore the ATL language, for example, showed the use of higher-order transformation metamodel. At last, it provided a study case to verify its feasibility and availability. The approach improves the abstraction layer of model transformations, reduces the difficulty of the reuse of model transformation language, and solves the incompatible problem of model transformation technology in a way.

Key words: MDA (model driven architecture); higher-order model transformation (HOMT); model transformation; ATL metamodel

0　引言

模型驱动软件开发带动了模型转换技术的进一步发展。随着建模领域的复杂度越来越高，模型转换的过程和模型转换规则的编写变得越来越复杂。早期的模型转换语言并没有自己的建模元素和元模型支持，如何对模型转换进行有效地控制以及重复利用已存在的转换规则已成为目前亟待解决的问题。于是，将高阶模型转换的思想引入到 MDA 模型转换领域成为一种新的方法，它利用模型的方法来表示模型之间的转换规则，采用类似于对象建模的方式可视化描述模型转换规则。

文献[1,2]采用高阶模型转换的方法，设计了一种 AMW 平台用于在模型之间建立联系，但这种方法描述模型转换的能力不足，而且没有定义从高阶转换模型到具体模型转换语言的代码生成机制。文献[3]提出了将模型编织的思想用到模型转换过程当中，构造出了一个编织模型来表示模型之间的转换；但该方法没有对模型进行分层实现，不能与具体的 MDA 工具相兼容，同时造成代码与模型之间的联系是单向的，不能实现模型和转换规则代码的同步。而这些研究中都只运用了 MDA 标准中的模型转换框架，并没有把它与整个 MDA 理论相结合。针对以上问题，本文把高阶模型转换的思想与 MDA 的开发方法相结合，采用 EMF(eclipse model framework)[4]，在 M2 元模型层[5]构造一种通用的高阶转换元模型，并以 ATL 语言为例，设计了高阶转换元模型到 ATL 转换语言模型规则语义；最后采用 Acceleo 工具设计转换语言模板，实现了 ATL 转换语言模型到 ATL 代码的转换。

1　结合 MDA 的高阶模型转换过程

高阶模型转换(HOMT)是指以一种转换模型作为输入并输出另一种转换模型的方法[6,7]。HOMT 提升了模型转换技术的抽象层次，并将转换模型作为自己的输入/输出模型，通过建立两种转换语言元模型之间的映射关系，来完成两种模型转

收稿日期：2012-05-07；**修回日期**：2012-06-12

作者简介：曾一(1961-)，男，教授，主要研究方向为软件工程理论及应用、软件测试、面向服务的计算；许林(1986-)，男，硕士研究生，主要研究方向为软件工程理论及应用(546430850@qq.com)；黄兴砚(1987-)，男，硕士研究生，主要研究方向为软件工程理论及应用；王翠钦(1988-)，女，硕士研究生，主要研究方向为软件工程理论及应用.

换语言之间的转换(图 1)。

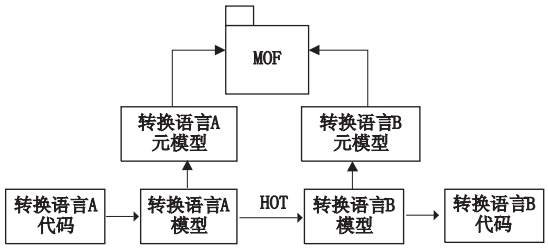

图1　高阶模型转换基本思想

根据这种思想,首先设计和构建高阶转换元模型和具体转换语言元模型并建立转换规则,把平台无关的转换规则模型转换为平台相关的转换语言模型;然后通过平台相关的转换语言模型生成的转换规则代码来执行 MDA 的模型转换。这样就能够把高阶转换过程与传统的 MDA 开发方法结合在一起。本文运用高阶转换模型的原理来实现一种平台无关的高阶转换模型到 ATL 转换语言之间的转换过程,这个过程需要与 MDA 过程相结合穿插进行。如图 2 所示,它包含五个基本步骤:

a)对复杂的单层编织模型进行分解,设计和实现高阶转换元模型。这样就解决了单层模型的复杂性和无法从 PSM 模型逆向转换到 PIM 模型的缺点,解决了无法与 MDA 架构相兼容的问题。

b)根据具体的源元模型和目标元模型提取它们的模型元素,并结合高阶转换元模型构建用于设计转换模型的建模元素。

c)步骤 b)建立的高阶转换模型不能直接被模型转换引擎执行,必须把高阶转换模型与相关模型转换语言联系在一起才能执行模型转换,因此,利用 Ecore 为 ATL 转换语言构建元模型,使得高阶转换模型与 ATL 转换语言模型的元元模型层统一。

d)设计高阶转换模型与 ATL 转换语言模型之间的映射关系,编写相关的转换规则,并在 EMF 转换引擎下实现模型之间的自动转换。

e)根据步骤 d)生成的 ATL 模型,编写具体实现 ATL 语言的模板规则,实现模型到代码的转换。

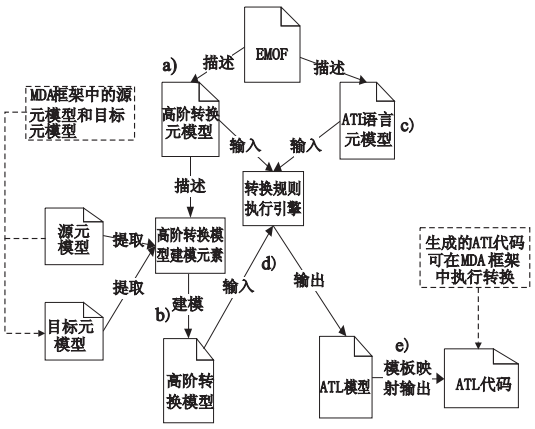

图2　高阶模型转换流程

2　高阶转换元模型的设计

根据 OMG 对 MDA 的规定[8],MDA 要解决不同抽象级别之间不同模型的转换问题 。MDA 指南并没有定义模型转换的具体规则,直到 2004 年才发布了模型转换的 RFP(request for proposal),但对模型转换的标准化过程目前仍在进行中。由于模型转换目前没有达到技术的统一,而且最初的转换由八个团队各自提交,这些方案之间存在很大的区别,没有统一的基础。本文中高阶转换元模型设计是根据这些 RFP 文档的共同点提取和抽象出来的。如图 3 所示。该元模型的每部分的具体含义如下:

HOTElement　所有元素的抽象元素,该元素包含了模型的一些公共部分,例如名字(name)和对模型的描述(description)。

Model　模型转换过程中的模型源元模型或目标元模型。

AbstractRef　是继承于 AbstrctElement 的一个抽象类。在模型转换过程中需要选取转换的模型和模型元素,需要使用 Reference 元素在模型转换过程中来引用它们。这时就需要抽取出模型引用和模型元素应用的公共结构,构成一个抽象类 AbstractRef 来表示。

ModelRef　抽象类 AbstractRef 的子类,可以看出具有模型和引用的共同属性。在模型转换过程中表示具体元模型的引用结构。一个 Model 可以与 ModelRef 相关联,用来表示模型转换中的输入模型和输出模型。

ElementRef　继承于 AbstractRef 元素。模型转换过程主要是实现模型元素之间的转换。ElementRef 用于表示对模型元素的引用,在转换中链接具体的模型元素。

LinkNode　模型转换是针对源元模型和目标元模型元素进行的,LinkNode 表示了链接模型两端的元素。

transformation　表示模型转换过程中由源模型到目标模型元素的一个转换,该转换运用模型元素之间的一个链接表示。它包含两个 LinkNode 元素,即 source 和 target。这两个 LinkNode 元素分别表示转换的两个端。

constraint　在本文的高阶转换元模型中,所有需要表示约束的对象都是依靠继承 constraint 对象来实现的,该对象在这里就表示了一个约束。鉴于 OCL 使用的广泛性,constraint 对象继承于 OCL 结构集。

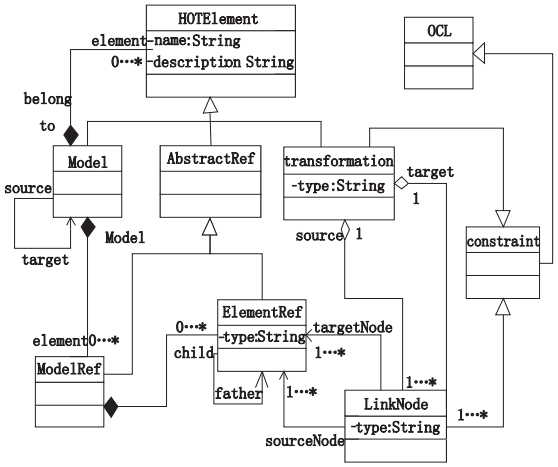

图3　高阶转换元模型

3　高阶转换元模型到 ATL 代码的实现过程

采用高阶转换元模型表示的高阶转换模型属于 PIM 层模

型,不能通过模型转换引擎来直接执行该转换模型,还需要把它与具体的模型转换语言关联起来。本章将对第 2 章所设计的高阶转换元模型进行具体实现,在 Eclipse 平台下,以 ATL 转换语言为例,采用 EMF 框架对高阶转换模型到 ATL 模型,以及 ATL 模型到 ATL 代码进行了具体实现,从而验证了该高阶转换元模型的可行性。

3.1 ATL 元模型的设计

ATL[9] 是符合 OMG 定义的 QVT 标准。ATL 元模型中包含了用于模型查询(query)、定义视图(view)和模型转换(transformation)的相关标准。由于高阶转换元模型是用来表示模型之间的转换规则,因此对于 ATL 中关于模型查找和视图部分的模型元素没有考虑在模型转换过程当中。本文从 ATL 完整的元模型中抽取了其中用于模型转换的核心部分,如图 4 所示,相关元素以及各对象之间的关系分析如下:

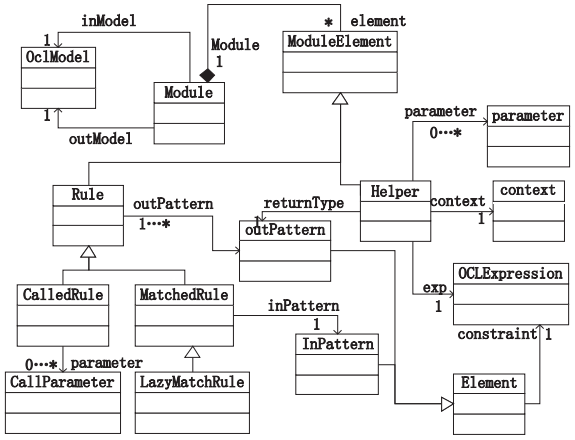

图4 ATL元模型

一个完整的模型转换被定义在 Module 对象当中。Module 关联了两个 OclModels,分别表示模型转换的输入元模型和输出元模型。一个 Module 对象也包含了多个 ModuleElement 对象。ModuleElement 表示模型转换中基本的模型转换块,其中包括 Helper 和 Rule 对象,这些对象共同构造了模型转换的核心。

Helper 继承于 ModuleElement 对象,从图 4 中可以看出它是 Module 的重要组成部分。在编写 ATL 转换代码时,Helper的功能是为了把复杂的 ATL 转换代码模块化。根据具体的功能需求可以把相应的代码编写在不同的 Helper 中。在 ATL 中存在两种不同的 Helper,即 function helper 和 attribute helper。

Rule 继承于 ModuleElement 对象,同样也是聚合在 Module的重要组成部分。在 ATL 语法规则中,源元模型与目标元模型之间的转换都是在 Rule 代码块中进行的。在具体实现模型转换过程当中主要依靠继承于 Rule 的 MatchedRule、CalledRule 和 LazyRule 三个对象。

MatchedRule 在 MatchedRule 中可以直接定义转换源元模型和需要转换到的目标元模型。MatchedRule 对象所关联的 outPattern 和 inPattern 对象用来表述转换过程中的输出模型元素和输入模型元素。在转换过程中,需要编写相应的转换规则,转换规则通过与 MatchedRule 相关联的 OclExpress 对象表示。

LazyRule 继承于 MatchedRule 对象,除了与 MatchedRule 有相同的模型转换能力之外,还对 LazyRule 有相关的约束。它只能用在模型转换之中,被其他对象所调用,而不能单独进行模型转换。

CalledRule 直接继承于 Rule 对象。在执行 ATL 代码时默认情况下会依次执行每一条转换规则。如果需要自定义一个转换开始的入口点,就需要通过 CalledRule 来实现。CalledRule 可以包含多个输入 CallParameter 对象,表示 CalledRule 的输入对象。在 CalledRule 内部没有输入模型,只有输出模型。

3.2 高阶转换元模型到 ATL 元模型的转换规则

要完成高阶转换模型到 ATL 模型的转换,需要编写它们的元模型层之间的映射关系。本节将分析高阶转换元模型到 ATL 元模型具体的转换规则及其具体的实现,其中包括 ATL Module、ATL Helper 和 ATL Rule 这三部分的转换以及具体转换规则代码。

3.2.1 Module 头格式的转换

ATL 元模型包含 OCLModel 对象,通过 Module 对象的 inModel 和 outModel 属性可以关联到 OCLModel 对象,它表示 ATL 的输入元模型和输出元模型部分。在高阶转换元模型中,通过定义 ModelRef 表示对模型进行转换的输入元模型和输出元模型。根据以上描述可以建立如下规则:

MⅠ Module 节点为所有节点的根节点。转换为 ATL 模型时需要先构建全局 Module 节点,其他节点都应包含在其中,构成一个完整的 ATL 模型。

MⅡ 在 ATL 模型中需要描述输入元模型和输出元模型,这对应着高阶转换元模型中的 ModelRef 对象。ModelRef 对象的 type 属性值若为 sourceModel,则表示该模型为输入元模型;若 type 属性值为 targetModel,则表示该模型为输出元模型。在模型转换过程当中,根据 ModelRef 的 type 属性值转换到相应 Module 关联的 inModel 或 outModel 中,其中 inModel 和 outModel都属于 OclModel 对象。相关代码如下:

```
Belper def:mainModule:MMATL!Module=OclUndefined;
entrypoint rule firstRule(){
    to
    mainModule:MMATL!Module(name<-' HOT2ATL' )
    do{
        thisModule.mainModule<-ainModule;
    }
}
rule HotSourceModelRef2InModel{
    from
        s:MMHOT!ModelRef(s.type=' sourceModel' )
    to
        t:MMATL!OclModel(name<-s.name)
    do{
        thisModule. mainModule. inModel<-t;
    }
}
```

3.2.2 Helper 的转换

Helper 对象继承于 ModuleElement 对象。由继承关系可知,Helper 可以包含在 Module 对象之中。通过 Helper 可以将代码拆分到不同的代码块中,细分不同的处理过程。ATL 包含方法 Helper 和属性 Helper。在语法结构上方法 Helper 可以有参数,而属性 Helper 却没有参数。它们都是通过高阶转换元模型的 transformation 对象来表示的,可以通过 type 的属性来

区分它们的含义。其具体转换规则如下:

HⅠ 通过高阶转换元模型 transformation 对象的 type 属性判断,若其值为 Method_Helper,则该 transformation 对象表示方法 Helper;若其值为 Attribute_Helper,则表示该 transformation 对象为属性 Helper。transformation 的名字转换为 Helper 的名字。

HⅡ 根据 transformation 对象的 sourceLinkNode 属性所引用到的 LinkNode 节点的属性值判定:若 type 属性值为 parameter,则表示该 LinkNode 节点转换到 Helper 的 parameter 对象,LinkNode 连接的 constraint 对象保存为参数类型,LinkNode 名为参数名;若 LinkNode 的 type 属性为 context,则该 LinkNode 为 Helper 的 context 对象,LinkNode 的名字为所关联的 context 的类型。

HⅢ 与 transformation 关联的 Constraint 对象的属性 expression 转换为 Helper 的 exp 对象,该对象表示为 Helper 的表达式的方法体。Helper 的表达式结构基于 OCL 语法。

HⅣ transformation 对应的 targetLinkNode 对象转换为 Helper 的返回值对象 returnType,该 LinkNode 的 type 属性需要标明为 returnType。根据 LinkNode 的 type 属性可以区分出返回值为对象类型或是简单数据类型。若 type 值为 parameter,则表示返回值为简单数据类型;若为 element,则表示存储的具体对象类。类型名存储在 constraint 对象中,并转换为 Helper 对应的 OutPattern 对象。OutPattern 对象拥有 name 和 type 属性,分别用来存储返回值的名字和类型信息。

HⅤ 属性 Helper 没有输入参数 parameter 部分,其 transformation 对象对应的 sourceLinkNode 端的 type 属性为 element,表示 Helper 所关注的是 context 上下文部分,将该 LinkNode 的 name 属性映射为 context 的类型。

HⅥ 属性 Helper 的 transfromation 包含的 constraint 对象表示属性 Helper 的处理部分,用于转换到 OclExpression 对象。其 targetLinkNode 端表示返回值类型,转换为 outPattern 对象。

下列代码描述了 transformation 对象转换为 Method Helper 的过程,转换为 Attribute Helper 的过程与该过程类似,本文中不再赘述。

```
rule Hot2MethodHelper{
    from
        s:MMHOT!transformation(s. transType=' Method_Helper' )
    to
        t:MMATL!Helper(
        type<-' Method_Helper' ,
        name<-s. name,
        parameter<-s. sourceLinkNode->select(e|e. LinkType' P' )->
        collect(e|thisModule.getParameters(e)),
        HelperContext<-s. sourceLinkNode->select(e|e.LinkType' Context' )->
        collect(e|thisModule.getContext(e))->first(),
        exp<-thisModule. getExp(s),
        returnType<-s. targetLinkNode->select(e|e. LinkType' returnType' )->
        collect(e|thisModule. getOutPattern(e))->first()
        )
    do{
    thisModule. mainModule. element<thisModule. allModule. allModuleElement.
    append(t);
    }
}
```

3.2.3　Rule 的转换

根据 ATL 元模型分析,Matched Rule 和 Called Rule 继承于 Rule 对象,Lazy Rule 继承于 Matched Rule 对象。Lazy Rule 并不直接执行模型转换,需要 Matched Rule 和 Called Rule 调用才能执行。Called Rule 可以作为整个 ATL 转换规则的入口规则,所以 Called Rule 没有模型输入部分。可用高阶转换元模型中的 transformation 对象表示一个模型转换规则,通过 transformation 的 type 属性的不同取值来区分转换的具体 Rule。其相关的转换规则描述如下:

RⅠ 根据 transformation 的 type 属性的取值(Matched_Rule、Lazy_Rule 和 Called_Rule),可以将其转换为相应的 Rule 对象。它们都继承于 Rule 对象,拥有 outPattern,其转换的输出部分具有相同的映射关系,不同点在于输出部分和是否拥有输入参数。

RⅡ MatcheRule 中的 From 部分表示模型转换中元模型的输入部分,对应着 transformation 的 sourceLinkNode 端,其 LinkNode 的 type 属性为 element。LinkNode 节点连接的 ElementRef 对象表示用于转换的输入对象。element 的名字映射为 inPattern 的名字,ElementRef 所关联的元模型转换为 inPatternd 的类型。关联在 LinkNode 上的 constraint 对象的表达式部分转换为输入元对象的 inPattern 的约束属性。

RⅢ transformation 关联的 targetLinkNode 对象表示模型转换中的输出部分,可以包含多个输出 LinkNode,对应着 Rule 可以输出多个目标对象元素。其每一个 targetLinkNode 对象转换为 Rule 中的一个 outPattern。outPattern 的名字映射为 targetLinkNode 的名字,关联在 targetLinkNode 端的 constraint 对象映射为 outPattern 的约束部分。

RⅣ CalledRule 没有输入模型,只有输入参数。若 transformation 的 type 属性值为 Called_Rule,则该 transformation 转换 CalledRule。transformation 关联的 sourceLinkNode 则表示 Called Rule 的输入参数。LinkNode 的 name 属性映射为 CallParameter 的 name 属性,LinkNode 关联的 ElementRef 对象转换为输出模型的所属类型,关联在 LinkNode 上的 constraint 对象的属性值表示 CallParameter 的类型。

RⅤ LazyRule 是不能直接用于模型转换的 Rule,它只能被 MatchedRule 或 CalledRule 间接调用。若 transformation 的 type 属性值为 Lazy_Rule,则转换为 ATL 中的 LazyRule。其映射规则与 Matched Rule 一致。其不同之处体现在 ATL 执行引擎的解释过程中,结构上没有不同。transformation 对象转换为 Matched Rule 的过程如下:

```
rule Hot2MatchedRule{
    from
        s:MMHOT!transformation(s. transType=' Method Rule' )
    to
        t:MMATL!MatchedRule(
        name<-s. name,
        outPattern<-targetLinkNode->select(e|e. LinkType)=
        ' Element' )->collect(e|thisModule. getRuleOutPattern(e)),
        inPattern<-s. sourceLinkNode->select(e|e. LinkType=
        ' Element' )->collect(e|thisModule. getRuleInPattern(e))->
        first()
    do{
        thisModule. mainModule. element<-
        thisModule. allModuleElement. append(t);
    }
}
```

3.3　ATL 模型到 ATL 代码的转换

在 Eclipse 的 EMF 平台下,为实现完整的模型到代码转换过程,专门设立了模型到文本(model to text,M2T)[10]项目。在这个项目之下实现了 JET、Acceleo 和 Xpand 三个 M2T 的开发工具。本节笔者采用 Acceleo 工具实现了 ATL 模型到代码的

自动转换,主要设计 ATL 的 Module 头文件、Helper 和 Rule 的模板文件,这些模板都是比较简单和直观的,本文以 Helper 模型到代码转换为例进行分析。

从 ATL 元模型的定义可以看出,一个 ATL 模型中可以包含多个 Helper,在对 ATL 模型转换为 ATL 代码过程中,需要对每一个 Helper 对象进行迭代,生成独立的 Helper 代码段。并且在生成 Helper 代码段时需要通过其 type 属性判断该 Helper 为属性 Helper 或方法 Helper,属性 Helper 没有参数部分。其具体实现代码如下:

```
[for(helper:Helper|aModule.eContents(Helper))]
helper[' ['/]context[helper. name/][']']?def:
[helper. HelperContext. name/]
[if(helper. type. equalsIgnoreCase(' Method_Helper' ))]=
([for(para:Parameter|helper. parameter) separator(' , ')]
  [para. name/]:[para. type/]
[/for])
[/if]:[helper. returnType. name/]=[helper. exp. expression/];
[/for]
```

4 高阶转换模型到 ATL 代码的转换实例

本章介绍了采用高阶转换元模型对源模型 Families 到目标模型 Persons 进行模型转换的实例。图 5 是用 Ecore 建立的 Families 和 Persons 用例元模型,其中 Family 拥有 lastName 属性,并且包含一个父亲、一个母亲、多个(零到多个)儿子和女儿,它们属于 Member 对象的实例,拥有 firstName 属性。现需要把 Family 的成员全部转换到 Person 对象的表示方式。每一个 Person 有 fullName 属性,Male 和 Female 继承于 Person 对象。

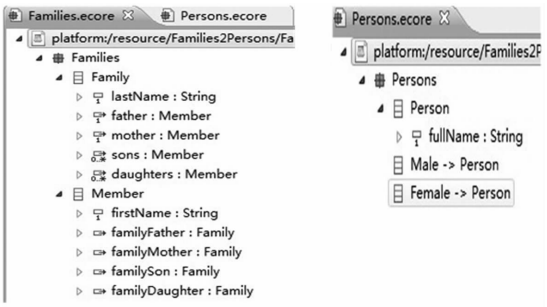

图5 Families和Persons元模型

根据转换流程,采用高阶转换元模型对 Families 模型到 Persons 模型的转换关系进行建模,如图 6 所示。

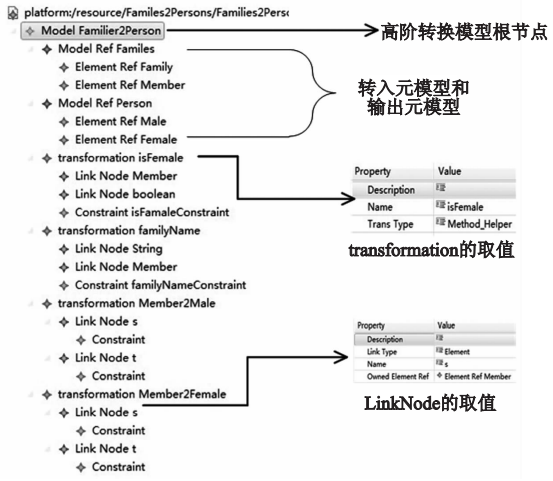

图6 模型转换过程

通过上述建模过程,可以自动生成高阶转换模型的 XML 文件。将该 XML 文件作为模型转换过程的输入模型,在 ATL 转换引擎自动处理过程下,可以生成需要的 ATL 模型文件。然后运行上章模板文件,可以得到具体的 ATL 代码文件(.atl 后缀)。

5 结束语

本文从领域模型建模、模型之间的转换和具体实现技术三个方面研究了高阶转换元模型的设计、高阶转换模型到 ATL 模型的转换以及 ATL 模型生成 ATL 代码的过程;最后用一个实例在 Eclipse 平台下进行了具体实现。

与传统的模型转换语言相比,结合了 MDA 开发过程的高阶模型转换方法能够使高阶转换元模型转换为多种模型转换语言,在一定程度上提高了模型转换语言的重用性,降低了模型转换技术的使用难度。同时,该方法遵循了 MDA 标准开发过程,提高了模型转换语言的可管理性与后期的维护效率,有利于在不同 MDA 开发平台之间扩展。在下一步研究中,将逐渐完善高阶转换元模型的设计以及具体语法的构造,同时实现基于高阶模型转换的建模工具。

参考文献:

[1] FABRO M D D, JOUAULT F. Model transformation and weaving in the AMMA platform[C]//Proc of Generative and Transformational Techniques in Software Engineering. 2005:71-77.

[2] FABRO M D D, JEAN B, FRÉDÉRIC J, *et al*. AMW: a generic model weaver[C]//Proc of 1ères Journées surl'Ingénierie Dirigée par les Modèles. 2005:105-114.

[3] 王学斌,王怀民,吴泉源,等. 一种模型转换的编制框[J]. 软件学报, 2006,17(6):1423-1435.

[4] PATERNOSTRO M. EMF: eclipse modeling framework[M]. [S. l.]: Addison-Wesley Professional,2010.

[5] OMG. Meta object facility(MOF) specification[EB/OL]. (2002)[2011-04]. http://www. omg. org.

[6] OLDEVIK J, HAUGEN O. Higher-order transformations for product lines[C]//Proc of the 11th International Software Product Line Conference. Washington DC: IEEE Computer Society,2007:243-254.

[7] TISI M, JOUAULT F, FRATERNALI P, *et al*. On the use of higher-order model transformations[C]//Proc of the 5th European Conference on Model Driven Architecture-Foundations and Applications. Berlin: Springer-Verlag,2009:18-33.

[8] OMG. Overview and guide to OMG's architecture (minor editorial corrections of omg/03-05-01)[EB/OL]. (2007). http://www. omg. org/cgi-bin/doc? omg/03-06-01. pdf.

[9] PIERS W. M2M/Atlas transformation language(ATL)[EB/OL]. (2010)[2011-04]. http://wiki. eclipse. org/ATL.

[10] ELDER P. IBM rational project lead. M2T: eclipse model to text[EB/OL]. (2012). http://www. eclipse. org/project-slides/EMFT-JET_Europa_Review. pdf.