

# 基于 UML 类图和顺序图的 C++ 代码自动生成方法的研究

王晓宇 钱红兵

(北京航空航天大学计算机学院 北京 100191)

**摘要** UML 是一种被广泛用于软件系统需求分析和详细设计的标准建模语言,研究将 UML 描述的软件详细设计自动生成代码的技术可以大大加速软件产品的开发进度,提高软件的质量。提出一种将 UML 类图和顺序图相结合生成具有静态结构和动态行为信息的 C++ 代码的方法,从而解决现在多数代码生成工具只能将静态图转换为 C++ 代码框架而不能处理动态行为模型转换的问题。该方法包括 UML 类图和顺序图的元模型以及相应的转换规则。最后通过一个采用 Velocity 技术实现的代码生成器生成代码的实例描述了代码生成的具体过程及结果。

**关键词** UML 代码自动生成 元模型

中图分类号 TP301 文献标识码 A DOI: 10.3969/j.issn.1000-386x.2013.01.046

## RESEARCH ON AUTOMATICALLY GENERATING C++ CODE FROM UML CLASS AND SEQUENCE DIAGRAMS

Wang Xiaoyu Qian Hongbing

(School of Computer Science and Engineering, Beihang University, Beijing 100191, China)

**Abstract** UML is a standard modelling language and is widely used in requirement analysis and high level design of software system. Research on the technology of generating C++ code automatically from high level software design depicted by UML can greatly accelerate the development process of software products and improve its quality. We propose an approach, which integrates UML class and sequence diagrams to form the C++ code containing both the static structure and dynamic behaviour information of the software system, therefore solves the problems of the current code generation tools that they are only able to transform static diagrams to C++ code frame other than dealing with the transformation of dynamic behaviour models. This approach consists of meta models of UML class and sequence diagrams as well as the corresponding transformation rules. A case of code generation by code generator, which is realized by Velocity, is used to present the specific process and result of code generation.

**Keywords** UML Automatic code generation Meta model

## 0 引言

在软件开发的过程中,开发人员总是重复编写一些简单的代码,而且每当新技术来临,又不得不一再地重复过去的工作。同时,需求的变化也从来没有停止过<sup>[1]</sup>。为了解决这些问题,人们提出了代码自动生成技术。代码自动生成技术根据模型驱动架构 MDA( Model Driven Architecture) 的思想,将由开发人员描述的软件系统模型转换为代码,使得模型成为软件开发的核心制品,提升了软件开发的抽象层次,从而提高软件开发效率和软件的可维护性<sup>[2]</sup>。

统一建模语言 UML 是一种以图形方式对系统进行分析、设计、的标准建模语言,使用 UML 建模可以清晰地表示系统的结构和行为信息。UML 模型中的类图显示了系统中各个类的静态结构,顺序图描述了对对象之间消息传递的时间顺序。因此,软件开发人员通常将二者结合描述软件系统的详细设计模型。代码自动生成就以类图和顺序图为输入,依据一定的转换规则生成具有静态和动态信息的代码。

目前关于将 UML 模型图生成代码的研究很多,文献[3-4]提出了一种将类图和顺序图生成具有结构和行为信息的 Java 代码的方法,文献[5]列出了多条顺序图到 Java 代码的转换规

则,文献[6]提出了一种将类图生成 C++ 代码的方法,文献[7]提出了静态模型到 C 代码的转换规则,文献[8]提出了一种将类图生成 .Net 组件代码的方法,文献[9]研究了代码自动生成技术中代码信息的来源。但对如何结合类图和顺序图生成与完整应用系统相接近的 C++ 代码的研究却比较少。

本文提出了一种将类图和顺序图相结合生成包括静态结构和动态行为信息的 C++ 代码的方法。

## 1 转换方法

图 1 描述了由 UML 模型转换到 C++ 代码的过程框架,包括 UML 模型、UML 元模型、代码生成器以及 C++ 代码。UML 模型包括语法正确的类图和顺序图,用来描述软件系统的静态结构和业务逻辑信息,由编程人员绘制。UML 元模型包括类图、顺序图的元模型,用来定义类图、顺序图的绘制规则。代码生成器的核心是代码转换规则,转换规则是根据 UML 模型元素的特点和 C++ 语言的代码结构建立的。自动生成代码时,首先输入符合 UML 元模型规则的 UML 模型,然后根据元模型的转

收稿日期:2012-08-07。2012 中国计算机大会论文。王晓宇,硕士生,主研领域:代码自动生成、模型驱动架构。钱红兵,副教授。

换规则生成 C++ 代码, 其中类图生成 C++ 中的类, 顺序图生成方法内部的具体实现。

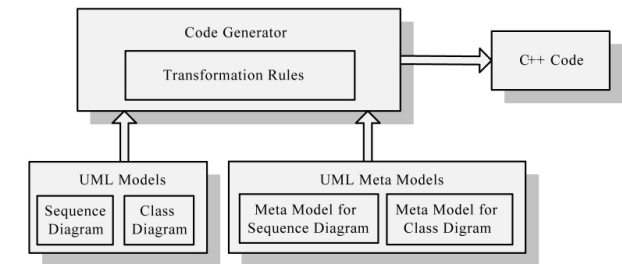


图 1 UML 到代码的转换方法

1.1 类图元模型

UML 的元模型定义了使用 UML 描述对象模型的完整语法规则<sup>[10]</sup>。图 2 为类图的元模型。UML 类图中的类与元模型中的 Class 对应, 属性和操作与 Attribute 和 Operation 对应。操作的参数对应 Parameter, 其中 kind 表示该参数的类型, 若 kind = in 表示该参数为调用操作时传递的参数, kind = out 则表示该参数为操作的返回值。各个类之间的关联关系用 Association 表示, AssociationEnd 记录关联端的名称和属性。属性和参数的类型用 Classifier 表示。

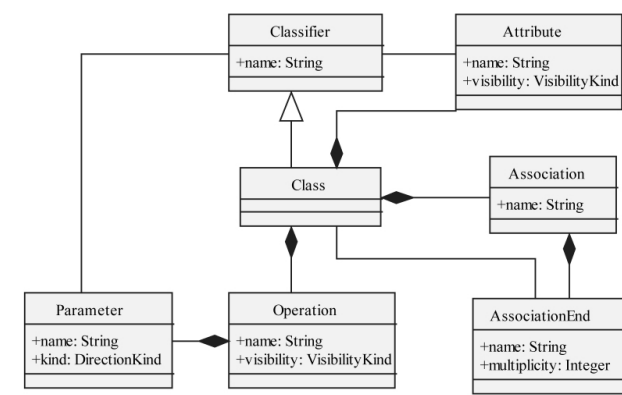


图 2 类图的元模型

1.2 顺序图元模型

图 3 为顺序图的元模型。一个顺序图用来描述类中的一个方法, 对应元模型中的 Interaction、Collaboration 和 Operation。其中 Operation 用来记录该顺序图所描述的方法的名称和可见性。顺序图中的对象对应元模型中的 ClassifierRole, 不同对象间通信的消息对应 Message, 消息中要执行的动作对应 Action, 该动作执行的条件用 recurrence 表示, 动作的内容用 Request 表示。动作分为调用( CallAction)、创建( CreateAction)、发送( SendAction)、返回( ReturnAction) 以及销毁( DestroyAction)。调用和创建操作的返回值用 ReturnVar 表示。参数的类型记录在 Classifier 中。

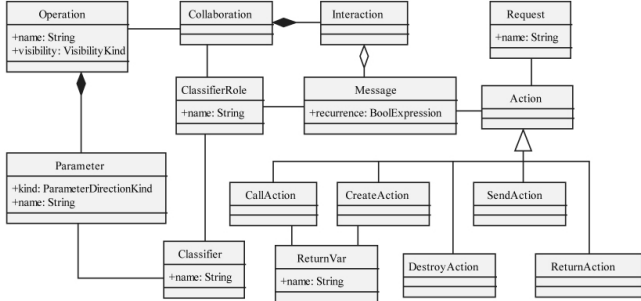


图 3 顺序图的元模型

2 转换规则

下面介绍 UML 类图和顺序图到 C++ 代码的转换规则。为了清楚准确地描述转换规则, UML 模型元素使用文献 [11] 中定义的元素, 转换规则中用到的标记如表 1 所示。转换规则使用表格描述, 其中第一列为待转换的模型元素, 第二列为该模型元素对应的元模型, 第三列为该元模型对应的转换规则。

表 1 转换规则中的标记列表

标记类型	标记格式	标记说明	实例
非终结符号	大写字母	被相应的规则替换, 可以迭代使用	ATTRIBUTE
非终结符号	斜体小写字母	被 UML 模型中的数据替换	c.name( 符号“.”表示 name 是 c 的一个属性)
终结符号	带下划线的符号	规则中的不变部分, 直接生成代码	<u>private:</u>

2.1 类图的转换规则

依据 UML 模型和 C++ 代码的特性, 得出类图和 C++ 代码的关系为: UML 模型中的类、属性、操作分别对应 C++ 中的类、成员变量、成员函数。因此, UML 类图到 C++ 代码的转换规则包括类的转换规则、属性和操作的转换规则。

规则 1 类的转换规则如表 2 所示。待转换的模型元素是一个有名称的类, 元模型中对象 c 的属性 name 记录了该类的名称。转换的第一步是 .h 文件( HFILE) 和 .cpp 文件( CFILE) 的生成, 如表 2 转换规则第一行所示。 .h 文件包括头文件的引用 INCLUDE、成员变量 ATTRIBUTE 和成员函数 OPERATIONH 的声明, 被规则 2 中的内容替换。非终结符号 c.name 表示本规则所描述类的类名, 被第一列中的类名所替换, 语句 #ifndef 和 #define 后的 c.name 全部大写。在 C++ 中, 通常成员变量是私有的, 成员函数是公有的, 分别使用终结符号 private: 和 public: 描述, 如转换规则第二行所示。 .cpp 文件包括对应头文件的引用和成员函数 OPERATIONC 的具体实现, 如转换规则第三行所示。

表 2 类的转换规则

模型元素	元模型	转换规则
ClassName	c:Classifier +name=ClassName	CLASS-> HFILE CFILE
		HFILE-> #ifndef c.name _H_ #define c.name _H_ INCLUDE class c.name { <u>private:</u> ATTRIBUTE <u>public:</u> OPERATIONH }; #endif
		CFILE-> #include "c.name .h" OPERATIONC

规则 2 属性和操作的转换规则,如表 3 所示。待转换的是一个包含属性和操作的类,元模型中对象 o 和 a 分别记录操作和属性的名称,p1、p2 分别为操作的参数和返回值,c1、c2、c3 对应属性、参数、返回值的类型。转换规则的第一行定义了 ATTRIBUTE 的转换过程,c1.name 和 a.name 表示属性的类型和名称,二者组合完成了 C++ 中成员变量的声明。该转换规则中的符号“\_”是带下划线的空格,表示直接生成代码中的空格。OPERATIONH 的转换过程与此相同。OPERATIONC 的具体实现被顺序图生成的代码所替换,用非终结符号 SEQUENCE 表示。具体如规则 4 所示。

下面通过 Reader 类描述类转换到.h 文件的过程,如表 4 所示,Reader 类有一个 int 类型的属性 id 和一个返回值类型是 void 的操作 create()。根据其元模型,对应的转换规则分别包括属性、操作、和参数的转换规则。根据规则 1 生成的代码如表中转换过程第一行所示,ATTRIBUTE 和 OPERATION 还需要做进一步的转换,其他部分为最终生成的代码。根据规则 2 生成

的代码如表中第二到四行所示,属性和操作的声明分别为 int id 和 void create( int id)。最后经过合并,得到的代码如下所示。

```
#ifndef READER_H_
#define READER_H_
class Reader
{
private:
    int id;
public:
    void create( int id );
};
#endif
```

规则 3 关联关系的转规则,如表 5 所示。元模型中 c1、c2 表示两个相互关联的类,它们之间的关联关系用 a 记录。转换规则中定义了,在 ClassA 中添加对 ClassB 的引用,用符号 INCLUDE 表示。

表 3 属性和操作的转换规则

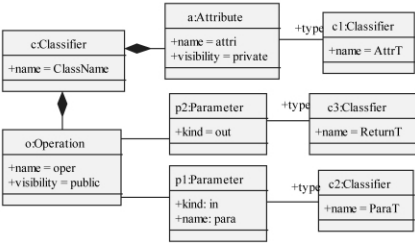
模型元素	元模型	转换规则
<div><div>ClassName</div><div>-attr: AttrT</div><div>+oper(para: ParaT): ReturnT</div></div>		ATTRIBUTE-> c1.name a.name _; OPERATIONH-> c3.name o.name (PARAMETER _); PARAMETER-> c2.name p1.name  OPERATIONC-> c3.name c.name :: o.name (PARAMETER _) { SEQUENCE }

表 4 实例分析表

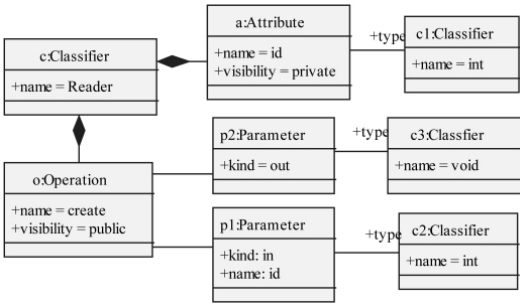


模型元素	元模型	转换过程
<div><div>Reader</div><div>-id: int</div><div>+create(id: int): void</div></div>		HFILE-> #ifndef READER_H_ #define READER_H_ class Reader { private: ATTRIBUTE public: OPERATIONH }; #endif ATTRIBUTE-> int id; OPERATIONH-> void create( PARAMETER _); PARAMETER-> int id

表 5 关联关系的转换规则

模型元素	元模型	转换规则
		INCLUDE-> #include "c2.name .h"

2.2 顺序图的转换规则

依据 UML 模型和 C++ 代码的特性,得出顺序图和 C++ 代码的关系为:UML 顺序图中的内容对应 C++ 成员函数的具体实现细节,顺序图中的分支、函数调用、对象创建分别对应 C++ 中的 if 语句、“.”运算符、new 关键字。因此,顺序图到 C++ 代

码的转换规则包括顺序图的转换规则、条件的转换规则、变量赋值的转换规则、对象创建的转换规则、方法调用的转换规则、消息发送的转换规则。消息返回和对象销毁操作不需要生成对应的代码。

规则 4 顺序图的转换规则,如表 6 所示。对象 o 表示该顺

序图是方法 `oper()` 的具体实现。非终结符号 `SEQUENCE` 被 `LOCALDATA` 和 `MESSAGE` 替换,说明一个方法的实现细节中包括局部变量的定义以及各对象之间相互传递消息的过程。

规则 5 条件的转换规则,如表 7 所示。对象 `a` 向对象 `b` 发送了一个包含条件的消息。元模型中对象 `m` 的属性 `recurrence` 记录了条件的内容,当该条件满足时,执行这个消息上的操作。`C++` 中使用 `if` 语句表示条件,如表中第三列所示。

规则 6 变量赋值的转换规则,如表 8 所示。将 `ClassB` 的方法 `oper()` 的返回值赋给变量 `x`,元模型中对象 `rv` 表示变量的名称 `r` 记录了方法的名称 `c` 表示函数的返回值类型即变量的类型。转换规则的第一行显示了变量 `x` 的定义,第二行中非终结符号 `ASIGNMENT` 表示该变量会被赋值,它可以被函数的返回值赋值,如规则 8 所示,也可以被常量或表达式赋值。

规则 7 对象创建的转换规则,如表 9 所示。“`<<create>>`”表示该消息是一个创建对象的操作。元模型中 `r` 表示待创建的对象的名称 `p` 和 `c2` 表示参数的名称和类型。`C++` 中有两种创建对象的方式,分别为 `ClassName object( param)` 和 `ClassName * object = new ClassName( param)`。为了避免用户使用完

对象后忘记删除而造成内存泄露,本文采用第一种对象创建的方法。转换规则第一行定义了创建对象时需要使用的参数,第二行 `MESSAGE` 生成对象创建的代码,第三行中非终结符号 `PARAMETER` 转换为待传递的参数(`PARAMETER` 包含参数的类型和名称,`PARAMETER` 仅包含参数的名称)。

规则 8 方法调用的转换规则,如表 10 所示。消息的内容为调用对象 `objectB` 中的方法 `oper()`。元模型中 `cr` 记录对象的名称 `r` 表示方法的名称 `p` 和 `c2` 表示参数的名称及类型。方法调用包括其他对象方法的调用和对象自身方法的调用。`C++` 中使用符号“`.`”访问成员函数,转换规则的第二行显示了调用其他对象的方法并将结果赋值给一个变量的规则,当调用自身方法时,去掉 `cr.name` 即可。无返回值的方法的转换规则如表中第四行所示。

规则 9 消息发送的转换规则,如表 11 所示。消息中不包含任何方法调用或者对象创建的内容,元模型中 `r` 记录了消息的内容。消息发送包括发送到其他对象的消息和发送给自身的反身消息。二者都只需直接将消息中的内容直接生成代码,如表中第三列所示。

表 6 顺序图的转换规则

模型元素	元模型	转换规则
<div><div>ClassName</div><div>-attr: AttrT</div><div>+oper(para: ParaT): ReturnT</div></div>	<div><div>Collaboration</div><div><div>oOperation</div><div>+name = oper</div><div>+visibility = public</div></div><div><div>cClassifier</div><div>+name = ClassName</div></div></div>	SEQUENCE-> LOCALDATA MESSAGE

表 7 条件的转换规则

模型元素	元模型	转换规则
<div><div>a: ClassA</div><div>b: ClassB</div><div>1 [con]</div></div>	<div><div>mMessage</div><div>+recurrence = con</div><div><div>Collaboration</div><div><div>oOperation</div></div></div></div>	<div>MESSAGE -&gt; if( m. recurrence )</div> <div>{</div> <div>MESSAGE</div> <div>}</div>

表 8 变量赋值的转换规则

模型元素	元模型	转换规则
<div><div>ClassA</div><div>ClassB</div><div>1: x=oper()</div><div>ClassB</div><div>+oper(): ReturnT</div></div>	<div><div>mMessage</div><div><div>Collaboration</div><div><div>oOperation</div></div></div><div><div>rRequest</div><div>+name = oper</div><div><div>Action</div><div><div>rvReturnVar</div><div>+name = x</div><div><div>cClassifier</div><div>+type</div><div>+name = ReturnT</div></div></div></div></div></div>	<div>LOCALDATA-&gt; c. name rv. name ;</div> <div>ASSIGNMENT-&gt; rv. name =</div>

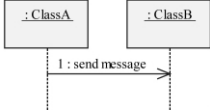
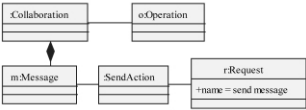
表 9 对象创建的转换规则

模型元素	元模型	转换规则
<div><div>ClassA</div><div>ClassB</div><div>1: &lt;&lt;create&gt;&gt;</div><div>1: objectB ( x )</div></div>	<div><div>mMessage</div><div><div>Collaboration</div><div><div>oOperation</div><div><div>c1Classifier</div><div>+name = ClassB</div></div></div></div><div><div>caCreateAction</div><div><div>rRequest</div><div>+name = objectB</div><div><div>pParameter</div><div>+kind = in</div><div>+name = x</div><div><div>c2Classifier</div><div>+name = ParaT</div></div></div></div></div></div>	<div>LOCALDATA-&gt; c2. name p. name ;</div> <div>MESSAGE-&gt; c1. name r. name ( PARA );</div> <div>PARAMETER-&gt; p. name</div>

表 10 方法调用的转换规则

模型元素	元模型	转换规则
<div><div>ClassA</div><div>objectB: ClassB</div><div>1: oper ( x )</div><div>ClassB</div><div>+oper(para: ParaT): ReturnT</div></div>	<div><div>Collaboration</div><div><div>oOperation</div><div><div>crClassifierRole</div><div>+name = objectB</div></div></div><div><div>mMessage</div><div><div>c1Classifier</div><div>+name = ClassB</div><div><div>c2Classifier</div><div>+name = ParaT</div></div></div><div><div>caCallAction</div><div><div>rRequest</div><div>+name = oper</div><div><div>pParameter</div><div>+kind = in</div><div>+name = x</div></div></div></div></div></div>	<div>LOCALVAR-&gt; c2. name p. name ;</div> <div>MESSAGE -&gt;</div> <div>ASSIGNMENTcr. name _r. name ( PARA );</div> <div>PARAMETER-&gt; p. name</div> <div>MESSAGE -&gt; cr. name _r. name ( PARA );</div>

表 11 消息发送的转换规则

模型元素	元模型	转换规则
		MESSAGE -> r. name ;

3 实例分析

本文以图书管理系统为例,描述如何结合类图和顺序图生成包含结构和行为信息的 C++ 代码。为了便于理解,对实例进行了简化,只给出了一个概念性的模型。该模型包括图书 Book、读者 Reader、借书记录 Record 以及系统交互界面 SysInterface 四个类,如图 4 所示。读者可以使用该系统借书、还书、查看书籍的详细信息。

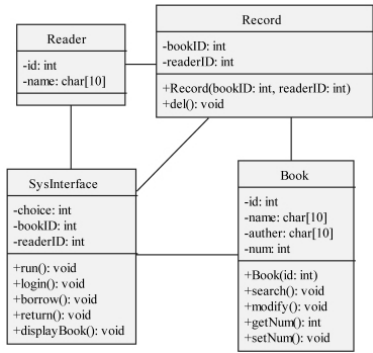


图 4 图书管理系统类图

图 5 是 SysInterface 类中方法 return() 对应的顺序图。还书时,系统首先提示用户输入图书编号 bookID,并根据该编号创建一个 Book 类的对象 b,调用 search() 方法将该书的信息从数据库读入 b 中。然后将该书的数量加 1,调用 modify() 方法将更新后的结果保存至数据库中。最后创建一个图书编号为 bookID,读者编号为 readerID 的 Record 对象 rc,调用该方法 del() 在数据库中删掉该条借阅记录。

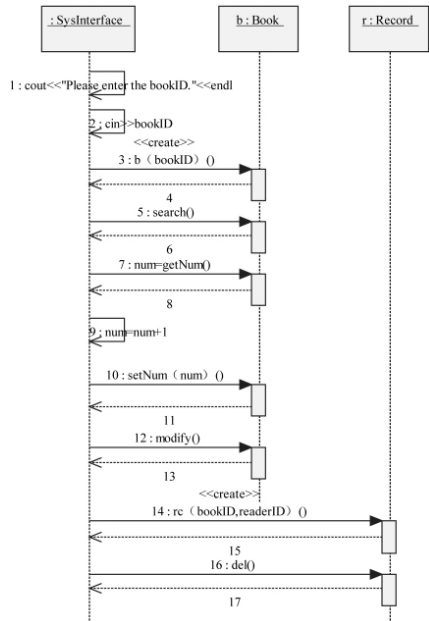


图 5 方法 return() 的顺序图

类 SysInterface 生成代码的过程为:

- (1) 应用规则 1,生成 SysInterface. h 和 SysInterface. cpp 文件;
- (2) 应用规则 2,生成 SysInterface. h 文件中的成员变量 choice、bookID、readerID 和成员函数 run()、login()、borrow()、return()、displayBook() 的声明,SysInterface. cpp 文件中成员函数 run()、login()、borrow()、return()、displayBook() 的框架;
- (3) 应用规则 3,生成 SysInterface. h 文件中对 Book. h、Record. h、Reader. h 文件的引用;
- (4) 应用规则 4,生成 return() 函数的具体实现结构,先声明变量,然后顺序显示对象间传递的消息;
- (5) 应用规则 9,生成提示用户输入图书编号,并存储该编号到变量 bookID 的语句;
- (6) 应用规则 7,生成创建 Book 类的对象 b 的语句;
- (7) 应用规则 8,生成调用对象 b 的方法 search() 的语句;
- (8) 应用规则 6 和规则 8,生成变量 num 的声明和赋值语句;
- (9) 应用规则 9,生成变量 num 加 1 操作的语句;
- (10) 应用规则 8,生成调用对象 b 的方法 setNum() 和 modify() 的语句;
- (11) 应用规则 7,生成类 Record 的对象 rc 的创建语句;
- (12) 应用规则 8,生成调用对象 rc 的方法 del() 的语句;

经过以上步骤后自动生成的代码如表 12 所示。由于篇幅有限,SysInterface. cpp 中没有列出 run()、login()、displayBook() 的框架。结果显示,在 search()、modify()、getNum()、setNum()、del() 已定义的前提下,自动生成的 return() 方法清楚地描述了还书操作的执行流程,是可执行的 C++ 代码。由此可见,可以使用细粒度的顺序图描述各个方法的动态行为信息,应用本文提出的转换规则生成与图书管理系统相接近的 C++ 代码。与多数文献中提出的只能生成代码框架的方法相比,本文提出的方法可以生成内容更加完整的代码。

根据以上研究成果,本文实现了一个基于 PSM 的代码自动生成工具。该工具采用 Eclipse 插件开发的方式,元模型使用基于 XML 的元数据表示方法,转换规则使用 VTL 编写的模板实现,代码生成器使用 JDom 技术解析 XML 文件内容,然后使用 Velocity 引擎根据模板的访问请求返回相应的模型数据并与代码模板合并生成目标代码。如图 6 所示,其中包括 SysInterface 类对应的 XML 文件、类图到 C++ 代码的转换模板、自动生成的 C++ 代码。由于时间有限,顺序图到 C++ 代码的转换模板还未完成。Velocity 技术的使用提高了代码的生成效率与代码模板的灵活性,从而为生成可执行的代码提供了技术支持。

表 12 生成的代码表

SysInterface.h 文件	SysInterface.cpp 文件
<pre>#ifndef SYSINTERFACE_ #define SYSINTERFACE_ H_ #include "Book.h" #include "Record.h" #include "Reader.h" class SysInterface { private:     int choice;     int bookID;     int readerID; public:     void run();     void login();     void borrow();     void return();     void displayBook(); }; #endif</pre>	<pre>#include "SysInterface.h" void SysInterface::borrow() { } void SysInterface::return() {     cout &lt;&lt; "Please enter the bookID. " &lt;     &lt; endl;     cin &gt;&gt; bookID;     Book b( bookID );     b. search();     int num;     num = b. getNum();     num = num + 1;     b. setNum( num );     b. modify();     Record rc( bookID ,readerID );     rc. del(); }</pre>

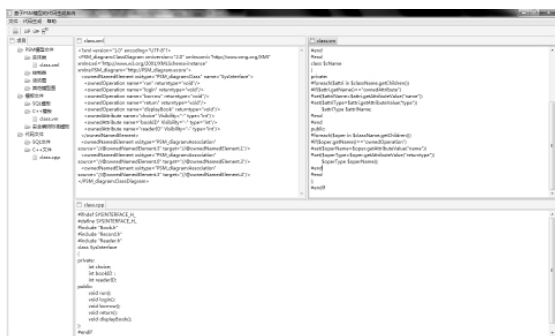


图 6 SysInterface.h 文件的生成界面

## 4 结 语

本文描述了一种结合类图和顺序图生成具有结构和行为信息的 C++ 代码的方法。先将 UML 模型中的各个元素映射到相应的元模型,然后根据元模型的转换规则逐步生成 C++ 代码。与多数文献中提出的只能生成 C++ 代码框架的方法相比,本文提出的方法可以生成内容更加完整的代码,减少了编程人员手动添加代码的工作,从而提高了软件的开发效率和软件产品的质量。由于顺序图不易于描述代码中复杂的逻辑信息,所以部分逻辑复杂的代码需要编程人员手动添加或者参考活动图的信息。此外,本文只关注了类图和顺序图中的主要模型元素,类图中的关联类、受限关联以及顺序图中的片段都还没有相应的转换规则。所以,下一步将继续完善类图和顺序图的转换规则并提出活动图的转换规则。

## 参 考 文 献

- [1] Anneke Kleppe, Jos Warner, Wim Bast. 解析 MDA [M]. 鲍志云,译. 北京: 人民邮电出版社, 2004.
- [2] 刘辉, 麻志毅, 邵维忠. 元建模技术研究进展 [J]. 软件学报, 2008, 19(6): 1317-1327.

- [3] Abilio G Parada, Eliane Siegert, Lisane B de Brisolara. Generation java code from UML class and sequence diagrams [C] // 2011 Brazilian Symposium on Computing System Engineering (SBESC). Pelotas, Brazil, 2011.
- [4] Usman M, Nadeem A. Automatic generation of java code from UML diagrams using UJECTOR [C] // International Journal of Software Engineering and its applications (IJSEIA), 2009, 3.
- [5] Mathupays Thongmak, Pornsiri Muenchaisri. Design of rules for transforming UML sequence diagrams into java code [C] // Proceedings of Ninth Asia-Pacific Software Engineering Conference. Gold Coast, Australia, 2002.
- [6] Dan Regep, Fabrice Kordon. Using metascibe to prototype an UML to C++/Ada code generator [C] // 11th International Workshop on Rapid System Prototyping. Paris, France, 2000.
- [7] 由志远. 基于 MDA 的嵌入式软件代码生成器设计与实现 [D]. 西安: 西安电子科技大学, 2010.
- [8] Deuk Kyu Kum, Soo Dong Kim. A systematic method to generate . Net components from MDA/PSM for pervasive service [C] // Fourth International Conference on Software Engineering Research, Management and Applications. Washington, USA, 2006.
- [9] Jichen Fu, Wei Hao, Farikh B Bastani. Model-Driven Development: Where Does the Code Come from? [C] // Fifth IEEE International Conference on Semantic Computing. Palo Alto, USA, 2011.
- [10] OMG. UML Semantics. Version 1.1 [R]. The Object Management Group, Document ad/97-08-05, Framingham MA, 1997.
- [11] OMG. OMG Unified Modeling Language™ (OMG UML) Superstructure. Version 2.2 [R]. The Object Management Group, Document, 2009.

(上接第 131 页)

## 3 结 语

在对小区宜居性评价的过程中,我们得到了以下几点心得体会:

- (1) 本文的层次分析,判断矩阵的建立采用了三个指标度量法,比 9 标度法容易操作很多,作为专家就可以很好地把握评价标准。
- (2) 本文的层次分析法,不需要进行传统方法的一致性检验,而是通过最优传递矩阵将比较矩阵转换为一致性矩阵。此方法可快速排列权重值,而且矩阵的计算推导过程明确简单,使用 MATLAB 很容易实现。
- (3) 本文的层次分析法与传统的分析法在结果上是一致的,使计算简单层次法更加有效。
- (4) 本文只是对此城市的三个区进行计算,但是对于更多小区来说计算方法是一样的,只是获得数据有些繁琐。

## 参 考 文 献

- [1] 李金明, 李润. 基于 AHP-FCE 法的煤矿安全管理系统研究 [J]. 计算机应用与软件, 2010, 27(4): 227-229.
- [2] 朱天志, 刘晓, 李政. 现代住宅功能设计中的定量与模糊概念 [J]. 河北科技师范学院学报, 2006, 18(3): 24-31.
- [3] 黄虎解, 耿永常, 赵晓红, 等. 住宅建筑方案优化设计研究与实现 [J]. 计算机辅助设计与图形学报, 2006, 12(6): 423-427.
- [4] 刘素平. 普通住宅户型分析 [J]. 山西建筑, 2006, 32(5): 21-22.
- [5] 唐舜. 选房使用指南 [M]. 北京: 机械工业出版社, 2008.