支持代码自动生成的行为建模①

Behavior Modeling for Code Generation

吴 同 （山东警察学院 治安系 山东 济南 250014）

摘 要: 本文根据模型驱动体系结构思想,针对目前代码自动生成工具对于行为代码生成支持不足的问题,提出了一种行为建模方法。通过对 UML 活动图进行扩展,进而对系统行为进行描述,使行为模型具有计算完备性。在一定程度上提高模型驱动代码生成的自动化程度,进一步减少应用系统开发的工作量,同时提高系统的可靠性和可维护性。

关键词: MDA 行为建模 扩展活动图 代码生成

1 引言

目前企业对软件系统的需求量不断增加,其质量要求也不断提高,应用系统的开发需要训练有素的程序员从事大量艰难琐碎的工作。但是仅依靠程序员手工编码不仅费时、费力,同时还存在着许多不稳定因素,造成程序的不可靠。OMG 于 2002 年正式提出 MDA(Model Driven Architecture)[1,2] 标准,将程序设计的抽象层次提高。设计时要进行严格的模型定义,最终自动生成全部或部分工程代码。这样做可以提高系统可靠性以及可维护性,同时也相当程度上减少了开发工作量。模型驱动的代码自动生成已经成为软件工程重要研究方向之一。

结合 MDA 方法,近几年也出现一些应用建模方法及其代码自动生成的工具。[3] 中提出了一种基于模型的 JAVA 代码自动生成方法。Compuware 公司的 OptimalJ、Interactive Objects 公司的 ArcStyle、IBM 的 Rational Rapid Developer 是目前市场上比较成熟的基于 MDA 的产品。应用程序的自动生成,其主要产品还是生成应用的大体框架,仍需要编程人员完成大量的手工编码工作。其中最主要的原因之一是对于系统的行为建模方法不完善。

本文针对以上情况对 UML 的活动图进行扩展,提出一种行为建模的方法,为模型驱动的代码自动生成提供支持。

2 基本思想

在面向对象分析设计中,可以认为应用系统任何行为都是由某个类所包含的方法体现[6]。因此对系统的行为建模实际这是对方法体进行模型描述。UML (Unified Modeling Language) 中活动图 (Activity Diagram) 提供了对行为流程建模的一定支持,但是对方法体建模支持不够完善,尤其是无法对代码的自动生成提供必要条件。本方法的主要思想是:对活动图进行表达方式以及语义上的扩展,采用 MOF 重型扩展与 UML profile 轻型扩展相结合的方法,使模型具备计算完整性。模型的计算完整性包括语法完整性(如规定所有属性和操作参数的类型必须预先定义)和语义完整性(如模型充分利用建模语言和生成器支持的语义结构),是代码自动生成的必要条件。

当然行为建模是基于结构建模基础之上的,行为模型中用到的许多概念实体是在结构模型中定义过的。例如类图所反映的系统各个概念实体及其它们之间的关系等。由于结构模型的建模方法目前已经相对成熟,并且到代码的转换规则比较直接,因此本文不做详细介绍。

3 活动图的扩展

UML 活动图表示从活动到活动的流。一个活动是一

① 基金项目:国家"863"计划 (2002AA4Z3240)

个状态机中进行的非原子的执行单元。活动最终导致一些动作,这些动作由可执行的原子计算组成,这些原子计算会导致系统状态的改变或一个值的返回。动作包括调用另一个操作、发送一个信号、创建或撤消一个对象、或者某些纯计算的操作,例如对一个表达式求值。

3.1 复合活动的扩展

在活动图中复合活动是指一个功能相对独立的活动流程。将复合活动节点展开就是另一个活动图。为支持行为建模和代码的自动生成,将复合活动扩充定义为三种形式:普通复合活动、本体类方法、异体类方法。采用扩展 UML profile 添加构造型的方法。

普通复合活动:指活动图中的子活动流程。此流程完全是为了模型表达清晰而设置。因此不必定义其输入与输出参数。在代码生成中,此子活动图转化而来的流程代码将直接插入方法体中。

本体类方法:指属于活动主体自身预定义的活动流程。父活动图所属类预定义的一个方法。它的定义是一幅完整活动图所描述的行为模型。为保证模型的一致性,在此处引用,必须保证已经存在它的定义。由于它属于类本身的方法,所以目标系统中此处不需要生成对类进行实例化的代码。

异体类方法:指属于其他类预定义的活动流程。它的定义是一幅完整活动图所描述的行为模型。为保证模型的一致性,在此处引用,同样必须保证已存在它的定义。由于它属于其它类的方法,因此目标系统中可能需要生成对类进行实例化的代码。

3.2 活动对象生命周期的定义

UML 活动图中定义了泳道(SwimLane)的概念,可以表达活动的主体。但是并没有定义主体的生命周期的表达方法,即生成代码时或者使用一个已经存在的对象,或者需要实例化一个类并不明确。生命周期确定了活动流程中协作对象的创建与消亡时机,为了提供对引用异体类方法的支持,因此定义被应用对象的生命周期。

这里提出的解决方法是在泳道中放置一个对象的引用,并且为这个应用添加一个标识位指示其是否需要实例化。如果标识为 TURE,遇到此异体类方法的调用则生成实例化代码。反之则直接引用。此处采用的是扩展 UML profile 添加标记值的方法。

3.3 活动的接口描述

由于本方法将普通复合活动以外的所有活动图都认为是某个类方法的模型描述,所以必须对方法的输入、输出参数以及可见性进行描述。因此需要对 UML 活动图增加接口描述元素,采用了对 MOF 进行扩展的方法。对活动接口描述的形式化定义:

INPARA = [DATATYPE PARANAME] *

OUTPARA = DATATYPE

ACCESS = Public | Private | Protect

在代码生成中可依据上述定义生成相应方法的签名(Signature)。另外需要指出的是在活动体内部定义的临时变量由于对外界是不可见的,所以在此处不需要考虑。

3.4 控制流程的扩展

UML 活动图中定义了以下几种活动控制流程:

转换:当一个状态的动作或活动结束时,控制流会马上传递给下一个动作或活动状态。转换也可以设置为有条件的。

分支:它描述了基于某个布尔表达式的可选择路径。其中各个监护表达式条件所覆盖范围不应该重叠。

分叉和汇合:分叉用来表示行为的并发执行,而汇合则表示两个或更多的并发行为的同步。

以上流程都可以在生成代码中体现。但是 UML 活动图并没有定义迭代控制流程。[5] 中提出为了获得迭代效果,可以用一个动作状态设置迭代器的值,用另一个动作状态增加该迭代器的值,并用一个分支来判断该迭代是否结束。但是使用此方法,不易于代码生成。因此需增加两种迭代活动节点:While 迭代活动节点和 For 迭代活动节点。这两中迭代活动节点实际是其他活动的容器,可以在其中放置需要迭代的复合活动节点或者简单活动节点。可以看出此处也使用对 MOF 进行扩展,增加迭带节点的方法。

While 迭代活动节点需要在建模时指定迭代的条件,而 For 迭代活动节点则需要指定其迭代范围。

3.5 服务活动原语

对 UML 活动图进行扩展的另一个重要方面是定义一系列服务活动原语。所谓服务活动原语是指几类创建系统应用时经常使用到的公共操作。例如对数据库的连接、对数据的各种查询操作等。定义服务活动

原语目的是在建立行为模型时,降低模型的复杂度,提高代码自动生成的效率。如果对行为各个细节都进行详细描述将会造成模型的复杂度提高,同时无法充分体现复用的优点。若所有活动流程都需要逐步建立,也会导致代码自动生成的效率下降。

为了充分体现设计模式的思想,可将服务活动原语按照其完成的功能分类,总结成为相关模式。系统设计人员在构建活动图时,如果使用到这些公共操作只需简单引入,无须再对其建立复合节点进行描述。当然活动服务原语已经定义了良好的接口供调用。

本文归纳总结了经常使用的原语,主要针对数据库进行操作的部分原语如表 1 所示。

表 1 部分服务活动原语

类型	名称	功能描述
添加	Add Not Match Any Field	检查输入参数是否为空后,根据输入参数向数据表中插入一条记录。当输入参数与数据表中的任何一条记录相对应的字段相等时,则不插入。
	Add Not Match All Fields	其功能与上个模式不同之处在于如果输入参数与数据表中的一条记录完全匹配则不插入。
	Add	向数据表中插入一条记录,并且返回数据表内容。
	Add And Replace	找到与输入参数相匹配的数据表记录后,用输入参数更新此记录。
匹配	Match Single Table	根据输入参数在数据表中查找相匹配的记录后,根据匹配情况做相应处理。
	Match Multi – Tables	根据输入参数在数据表中查找相匹配的记录后,根据条件参数返回与之相关联的表的信息。
更新	Update	根据输入参数,在数据表中找到匹配记录,更新相应字段,然后根据条件参数返回更新结果。
	Update Blank	根据输入参数,在数据表中找到匹配记录,如果相应字段为空则更新,然后根据条件参数返回更新结果。
	Delete	根据输入参数,在数据表中删除相应记录,然后根据条件参数显示更新结果。
	Clear	根据输入参数,在数据表中找到相应记录,清空特定字段。

4 扩展活动图对代码自动生成的支持

一个应用系统的整体模型应包括概念模型、行为模型、交互模型和部署模型等。扩展活动图所表示的行为模型只是应用系统整体模型的一部分,为了生成完整的目标系统必须要有其他模型的支持。扩展活动图中所引用的元素也必须在其他模型中预先定义,保证模型的一致性。

扩展活动图可以被认为是 MDA 中定义的 PIM(平台无关模型),不考虑任何平台相关的信息。在代码生成中使用基于模板的思想具有较强的可维护性[4],本方法根据不同开发平台的特性预定义模板,然后根据模型信息进行代码生成。例如可以对前面定义的服务活动原语建立 JAVA、C + + 平台下的模板,以满足不同平台的需求。

5 扩展活动图实例

如图 1 所示:定义了 Transaction 类的 Purchase 方法行为描述。可以看出其主要流程是首先传入商品 ID 和订购数量,检查库存。若库存充足则查询运送计划,若可以安排送货则形成订单。(这里只是举例说明,不一定符合实际业务流程。)

}

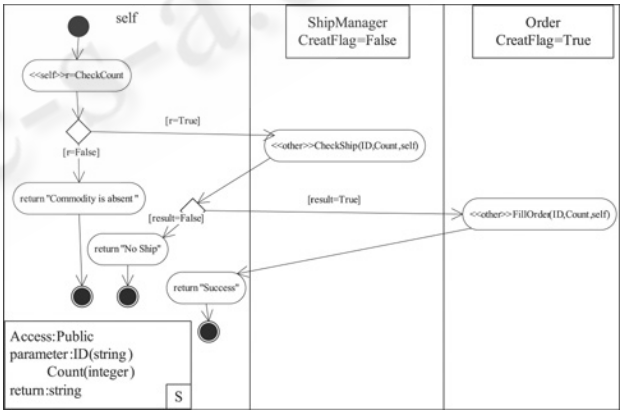

图 1 扩展活动图例 1

图中含有 < < self > > 构造型的活动为本体类方法,含有 < < other > > 构造型的活动为异体类方法。含有‘S’的标签定义了此方法的签名。而 CreateFlag 表示是否进行实例化。

根据模型信息,可以得到如下方法体,由于此模型为 PSM,所以只给出伪码,生成代码时使用特定于某个平台的模板,则可以生成实际目标代码。

```
Public Purchase(string ID, integer Count)
{
    r = CheckCount(ID,Count);
    if (r)
    {
        result =
ShipManager.arrangeship(ID,Count,self);
        if (result)
        {
          //实例化订单对象 order
          order.fillorder(ID,Count,self);
          ruturn" Success "
        }
        else
          return" No Ship "
    }
    else
        return" Commodity is absent. "
```

图 2 所表示的扩展活动图模型中,使用了 While 迭代控制流程。同时使用 < < include > > 标志循环中含有预定义的 < < Add > > 服务活动原语。这里不再给出其实现代码。

6 总结

本文提出了一种对 UML 活动图进行扩展,对应用系统行为进行建模的方法。该方法主要目的是为代码自动生成提供支持,在一定程度上解决目前大多数应用自动生成工具在行为描述方面的不足。可以进一步减少系统的手工代码量。提高系统的可维护性及可靠性。

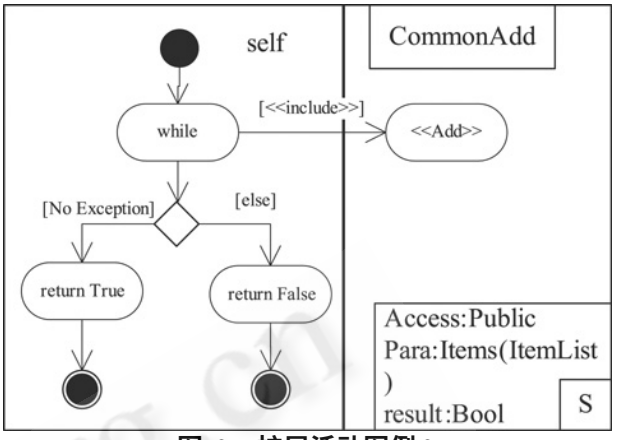

图 2　扩展活动图例 2

下一步工作主要是对模型进行进一步完善,使其表达方式更为简练,表达语义更加完备。其次是对服务原语进行扩充,使其能够表达更复杂的业务逻辑。同时模型的一致性约束问题也是主要研究内容之一。

参考文献

1 J. Miller, and J. Mukerji. MDA Guide Version 1.0.1. Document number omg/2003 – 06 – 01. Retrieved from: http://www.omg.com/mda, 2003.

2 David S. Frankel: Model Driven Architecture: Applying MDA to Enterprise Computing. 2003 John Wiley & Sons.

3 CHOW KO, Weijia Jia, CHAN VITO CP and CAO JN, "Model – Based Generation of Java Code", Proc. International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, USA, 26 – 29 June 2000.

4 Template – based Code Generation Technical White Paper. Copyright 1999 – 2000 iMatix Corporation.

5 Grady Booch, James Rumbaugh, Ivar Jacobson: The Unified Modeling Language User Guide. 1999, Addison Wesley Longman, Inc.

6 Anneke Kleppe , Jos Warmer, Making UML Activity Diagrams Object – Oriented, Proceedings of the Technology of Object – Oriented Languages and Systems (TOOLS 33), p.288, June 05 – 08, 2000