软件形式化与可视化软件模型的转换*)

周彦晖　张为群
（西南师范大学计算机与信息科学学院　重庆400715）

The Transformation between Formal and Visual Software Model

ZHOU Yan-Hui　ZHANG Wei-Qun
(Faculty of Computer and Information Science, South West China Normal University, Chongqing 400715)

Abstract　It is an important issue in Software Engineering that combined the formal development method with the visual development method. This study is about the transform method and rules between the UML model and the RAISE model. At last try to put this technology and the common software develop process together.

Keywords　Formalize, Visualize, UML, RAISE, Model transformation

1. 前言

现有的面向对象的可视化方法和形式化方法都具有各自的优点，UML 是可以完全可视化的图形语言，使用简单，能够很好地体现面向对象软件开发的特点。现在已经有很多面向对象的软件 CASE 工具都支持 UML，其中包括 Rational 公司的 Rational Rose 系列，微软公司的 Visio，北大青鸟 CASE 工具，Visual UML 等。这些工具支持基于 UML 的软件开发过程模型 RUP(Rational Unify Process)下工作，能够很方便地使用 UML 建立软件模型，根据软件模型可以自动生成面向对象程序设计语言代码框架。开发者在这些代码框架下，利用现有的快速开发环境迅速产生原型、进行原型进化，直至产生最终软件系统。但是 UML 本身的图形语言缺乏形式化能力，软件开发过程中模型的建立和精化主要依靠软件开发人员的经验，而且由于面向对象方法的特点，决定了软件模型的精化过程没有固定方法，往往需要对模型进行多次重复的验证、修改、完善，最终才能产生较为完善的模型。例如，在 UML 中建立对象类图时，确定对象类的方法和属性是一个比较困难的问题，往往需要结合底层实现才能完成。UML 不是形式化语言，缺乏完善的模型检查能力，也不能很好地保证开发关键软件的可靠性。

形式化方法的最大目标是提高软件的可靠性。应用形式化语言能够为软件建立精确、无二义的模型—形式化规范，在初始规范的基础上，对初始规范进行检查和精化，并且可以采用数学方法证明这种精化符合初始规范的所有特性，从而在很大程度上保证软件开发过程的一致性和符合性。但是形式化方法的主要缺点也很突出，形式化语言在易用性上比图形语言差得多，形式化开发过程中要求开发小组中几乎所有的人员都要了解形式化方法，形式化语言往往没有非常有效的代码生成系统，这些问题限制了形式化方法在软件开发中的应用。

面向对象建模的可视化模型描述语言如 UML，直观、形象、易于掌握和使用。但是实践证明 UML 的图形模型缺乏严格的语义基础，容易产生歧义，因此为 UML 语言建立语义基础受到越来越多的重视。RAISE 作为一种形式化方法，具有较为完整的体系，提供了方法、语言和工具。同时 RSL 作为形式化模型描述语言具有对象、方案等特征，但是它也同样有和其他形式化方法类似的问题。

因此要将形式化方法和可视化方法结合起来形成一种新的软件开发方法和过程，可行的途径就是利用较为成熟的可视化方法 UML 和形式化方法 RAISE，取长补短。寻找 RAISE 和 UML 对象类模型之间的关系，需要使用 RASIE 来表示 UML 的类模型。

2. UML 与 RAISE 的模型关系

RAISE 规范语言——RSL (RAISE Specification Language)是一种广谱语言，它既可以用于书写非常抽象的、初级的规范，也可以用于书写易于甚至能自动转换成程序语言的更具体的规范，对这些规范可以进行精化并验证。如果能够使用 RSL 表示 UML 的基本模型，就能够建立 RSL 和 UML 的基本模型转换关系，进一步能够用 RSL 精化和验证这些 UML 模型，提高模型的有效性。在 RSL 中也提供了模块、方案、函数等概念，为表达 UML 模型提供了便利。

2.1　表示 UML 对象类

(1)模型转换的基础

定义1(类)　一个类是创建实例对象的蓝图，是构造一个新的类的代码重用基础。类(class)可以定义为一个五元组：$class::=\langle ID, INH, DD, OP, ITF\rangle$。其中，ID 是类的标识和名字；INH 是类的继承描述；DD 是数据结构描述；OP 是方法的实现；ITF 是统一的对外接口或协议。

定义2(对象)　对象(object)是一个类的实例，通过实例化操作产生，可以定义为一个三元组：$obj::=\langle OID, BODY, CID\rangle$。其中 OID 是对象的名字和标识；BODY 是该对象的实例存储结构；CID 是该对象所依赖的标识。

一般地，经常使用 C. CID 表示类 C 的标识，O. OID 表示对象 O 的标识。

由于对象是类的实例化，可以将对象扩展为一个六元组：$obj::=\langle OID, BODY, ODD, OOP, OITF, CID\rangle$。其中，ODD

*)教育部和重庆市"软件工程可视形式化"项目的部分工作。周彦晖　讲师，硕士，主要从事软件工程方向研究。张为群　教授，从事软件工程、人工智能方向研究。

是对象由类实例化之后的数据结构;OOP 是实例方法;OITF 是对象的接口。

RSL 中的基本语言单位是模块,模块和类的关系可首先考虑模块的定义。

定义3(RSL 模块) 可以定义为一个六元组:$SCHEME ::= \langle id, type, value, axiom, hide, extend \rangle$。其中 id 为模块标识;type 为模块内数据类型定义;value 为模块内数据和操作符号;axiom 是模块内定义的公理(操作规则);hide 为模块内符号在模块内外的可见性,extend 是 RSL 的模块扩展性——包括子类型和模块公理重新定义能力。

从定义3可知,RSL 模块和定义1的元组对应关系如下:

$id \rightarrow ID$

$type \cup value \rightarrow DD$

$axiom \rightarrow OP$

$hide \rightarrow ITF$

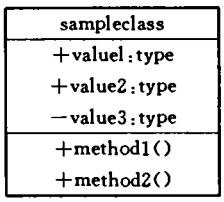
$extend \rightarrow INH$

因此,使用 RSL 的模块表示类是可行的。在 UML 中并未规定任何基本数据类型(实际上是面向具体语言的),RSL 中类型可分为固有类型和其他自描述的类型,考虑到形式化规范的抽象特征,如果具体使用某种程序设计语言实现这种规范时,RSL 所有的固有类型都是这些语言的基本类型,因此可以不使用 RSL 的固有类型,而全部采用自描述的其他类型。

(2)模型转换 RSL 描述类的基本定义形式如下:

```
id =
  class
    declarations
  end
```

对于任意 UML 对象类:



可描述为:

```
sampleclass=
hide value3 in
  class
    type
      type1,
      type2,
      type3
      value
        value1,
        value2,
        value3,
        ……,
        method1:type1→type2,
        method2:type2→type3
    axiom
      ∀ t1:type1·method1(t1)≡dosomething type1/2,
      ∀ t2:type2·method2(t2)≡dosomething type2/3
end
```

2.2 表示 UML 对象类的泛化与特化关系

泛化/特化是现实世界中一般性实体与特殊性实体之间的关系,一般性实体是特殊性实体的泛化,特殊性实体是一般性实体的特化。泛化也称为"a-kindof"联系。在对象模型中,表示一般性实体的对象类称为超型的类(Supertype),简称超类(基类),表示特殊性实体的对象类称为子型的类(Subtype),简称子类(派生类)。子类继承超类的特性(属性、操作、关联等),同时可以有自己的特性。

(1)转换的基础

定义4(继承) 继承(INH)定义为一个二元偏序关系:

$INH ::= \langle C, > \rangle$;

$> = \{\langle c_i, c_j \rangle \mid c_i, c_j \in C\}$

其中 C 为所有类的集合。">"表示继承。

由继承的定义,可以把具有继承关系的类重新定义:

定义5类(继承)

类 $c_n ::= \langle c_n.ID, \mathop{Y}_{i=1}^{n} c_i.DD, \mathop{Y}_{i=1}^{n} c_i.OP, \mathop{Y}_{i=1}^{n} c_i.ITF \cap c_n.ITF \rangle$

其中,类 $c_{i+1} > c_i$,$(i=1,2,\cdots,n-1)$。

在 RAISE 中,泛化/特化关系可以使用子类型来表达。如果类型 T1的值都包含在类型 T2中,则 T1是 T2的子类型。T2中可能含有不属于 T1的值。例如 RAISE 中的固有类型自然数 Nat 就是整数 Int 的子类型。子类型一般使用子类型表达式来描述,子类型表达式分为受限断言和不受限断言两种。例如:

{|t:Text·len t>0|}

此受限断言定义了 Text 的子类型,只包含非空文本。受限断言可以用于部分私有继承或重载。

{|t:Text|}

就定义了一个不受限断言。不受限断言可以用于公有继承。

可见,使用 RSL 的子类型和 extent 可以表示定义5的类的继承关系。

(2)转换 在 UML 中泛化/特化关系一般如图1所示。

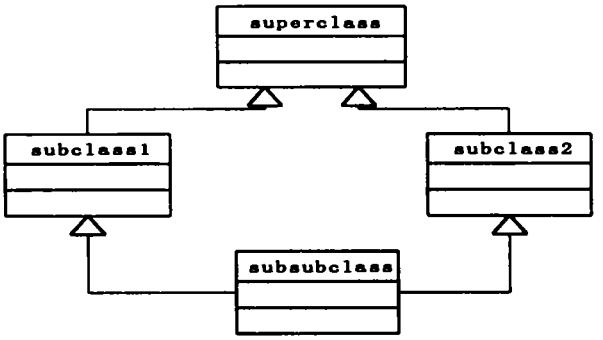


图1 UML 泛化/特化关系

利用以上的转换对应关系,可以得到 RSL 表示的派生类(限篇幅关系,只给出最底层派生类):

```
subclass1,subclass2/ * use two subclass * /
scheme subsubclass=
  extend subclass1,subclass2 with
  class
    object
      P:superclass
      sub1:subclass1(P)
      sub2:subclass2(P)
    type
      subsub_type = {| i: P.super_type · is_sub1_type (i) ∧ is_
      sub2_type(i)|}
    value
      somevalues
    axiom
      someaxioms
    end
```

2.3 表示 UML 对象类的关联关系

(1)转换基础 UML 中的关联关系可以分为自反关联、二元关联和 N 元关联。其中,自反关联可以归入二元关联,而 N 元关联将会造成实现复杂,因此 N 元关联应该避免使用。下面讨论二元关联。

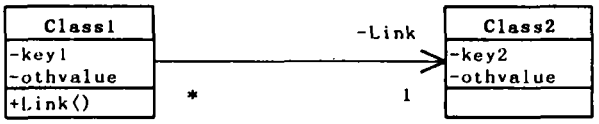


图2 UML 中类的一对多关联关系

二元关联又存在多种对应关系，包括一对关联、一对多关联、多对多关联。一般情况下，多对多关联将被分解为两个一对多关联来实现，如图3所示。因此，下面直接讨论一对多关联。

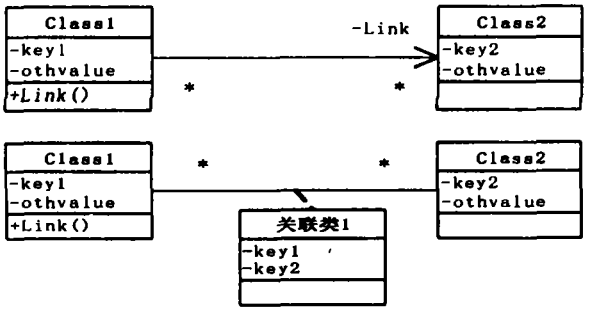


图3 将多对多关联转换成两个一对多关联

定义6 对象关系 ORL 是定义在对象集合 OSET 上的一个二元关系：

$ORL::=\{\langle O_1,O_2\rangle \mid (O_1\in OSET)\wedge(O_2\in OSET)\}$

定义7 关联关系 RTN：

$RTN::=\{\langle O_1,O_2\rangle \mid (O_1\in OSET)\wedge(O_2\in OSET)\wedge(O_2\in O_1.ODD)\wedge\exists\ M\cdot s(O_1,O_2,M)\wedge(M\in O_2.OITF)\}$

其中，$O_1.ODD$ 是 O_1 对象的数据成员；谓词 s(x,y,z)表示 x 对象向 y 对象发送消息 z。

(2)RSL 表示关联 对于一对多的关联，在 RAISE 中可以定义一个类型来表示被关联类，并使用该类型建立全局对象(OBJECT)，同时在关联类中建立一个链接属性(实质上是个函数)，该属性是从关联类到被关联类关键属性的映射，完成定义7中谓词 s(x,y,z)的功能。

使用 RSL 表示的关联关系为：

```
scheme Class2=
  class
    type
      key2_type,other_type
    value
      key2:key2_type
      othervalues
    axiom
      someaxioms
  end
Class2/ * use class2 type * /
scheme TYPES=
  class
    type class2=Class2-set
  end
object OBJ:TYPES
scheme Chass1=
  class
    type
      class1,key1_type,other_type,Link_type
    value
      key1,
      Link:chass1 ——m→ OBJ.class2,
      othervalues
    axiom
      someaxioms
  end
```

2.4 表示 UML 聚集关系

聚集关系是关联关系的一个特例，聚集关系的定义是定义7的特例。

定义8 聚集：

$AGN::=\{\langle O_1,O_2\rangle \mid (O_1\in OSET)\wedge(O_2\in OSET)\wedge(O_2\in O_1.ODD)\}$

其中，$O_1.ODD$ 是 O_1 对象的数据成员。

它表示了对象间整体与部分的关系。作为整体的类叫做整体类，作为部分的类叫做部分类。在面向对象理论中，具有较强的组成关系的聚集也被称为组合关系，但是在实现上，区别在于组合关系中的部分类的创建和消亡依赖于整体类。在本身所讨论的问题范围内来看，UML 和 RAISE 都没有描述类的实例化过程和消亡过程的必要，因此，这里只讨论聚集。

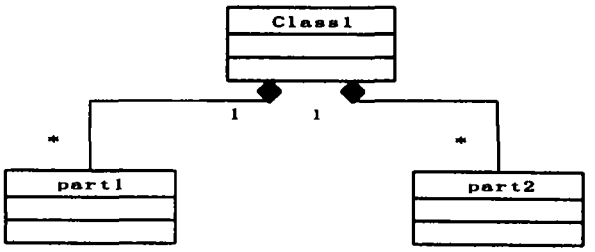


图4 UML 类的聚集关系

使用 RSL 可以把聚集关系描述为 scheme 中包含多个 Object 的形式，完全符合 RSL 的基本定义，基本形式如下：

```
Scheme Class1=
  class
    object
      part1_obj:part1
      part2_obj:part2
……
```

至此，已经完成了将 UML 的基本类模型转换成 RAISE 模型的基本工作，UML 的对象类图可以利用以上的转换方式转换为 RAISE 的初始规范，也可以把 RAISE 形式化规范转换为 UML 对象类图。

3. 可视化与形式化结合的开发方法基本思路

为了充分利用 UML 的可视化建模过程，同时提高 UML 的模型验证能力，因此主要考虑在 UML 总的开发框架下(RUP 过程)，使用 RAISE 的形式化模型来代替 UML 开发过程中的多次循环重复进行模型精化和模型检查的过程。而 UML 开发过程中具有优势的可视化方法如：USE CASE 分析、交互分析、协同分析、部署模型分析、组件模型分析都能够弥补形式化方法的不足之处，另外还应该充分利用现有的 UML CASE 工具的代码生成能力。基于这样的考虑，UML 和 RAISE 结合起来的总体过程如下：

a)问题定义和可行性研究。

b)需求分析和可视化建模过程，产生基本 USE CASE 图，对象类图，状态图等。

c)结合可视化模型产生形式化初始规范过程。

d)结合可视化模型对形式化初始规范进行精化、验证，产生新的规范。

e)根据形式化规范细化并改进对象类图、组件图、协调图，完成模型检查。

f)生成代码，根据需要添加用户界面和总控模块。

g)检查、完善代码。

h)建立系统配置模型，生成系统。

i)测试与完善。

基于业务构件的快速可重构信息系统的框架研究

李绪蓉　凌兴宏　丁秋林
（南京航空航天大学信息科学与技术学院　南京210016）

Research on the Framework of RRIS Based on Business Components

LI Xu-Rong　LING Xing-Hong　DING Qiu-Lin
(College of Information Science & Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016)

Abstract　Rapidly Reconfigurable Information System (RRIS)and its framework have been hotspots of research circles. The inherent characteristics of a framework offer a direct support for RRIS. Different software technologies will lead to different framework solutions and their reconfiguration mechanisms are very different. This paper gives a proper summary to the framework design of RRIS, and then, analyzes the strength and weakness between Object-Oriented enterprise framework and component-based enterprise framework. Moreover, the framework of RRIS based on business components is put forward and its building, functions, design and implementation are discussed in details.
Keywords　Reconfiguration mechanisms, RRIS, Business component, Framework, Software architecture, Enterprise framework

1　引言

系统可重构的思想蕴含着"适者生存"的"应变"的哲理。快速可重构性已成为智能系统的基本特性[13]，是衡量系统响应变化能力的重要指标，是企业赢得全球化市场竞争的关键因素。快速体现为尽量缩短新产品的上市时间，而系统重构体现为变化来临时系统中的子系统会重新组合而产生新功能、新过程或更高层的整体性[12]。快速可重构信息系统(Rapid Reconfigurable Information System, RRIS)及其框架研究已经成为业界的重要研究课题[15]。框架是半成品的应用程序，设计良好的框架应支持从系统分析、设计到代码级的重用。基于框架的开发可避免重复劳动，降低开发成本，提高软件生产率。重构应以重用为基础，框架的固有特性(模块化、可重用、可定制与可扩展)将对信息系统的快速重构提供直接的支持。不同的软件技术导致不同的框架解决方案，其重构机制也大不相同：OO框架技术利用对象的继承、组合(很少使用)以及设计模式等机制支持系统的重构，而构件化框架技术则通过构件在框架上的即插即用的方式来组建应用，即通过构件的演化、替换及重组等方式来支持重构。二者各有利弊，OO框架容易实现一些，而构件化框架比OO框架更灵活、演化性更好，代表未来的发展趋势。

业务构件技术是构件技术的最新发展，具有下面4个优点：促进了软件生产工业化；可取得生产率的突破；可减少业务技术隔阂；可有效地支持信息系统快速重构。业务构件可以很好地模拟业务过程并将其平滑地转变成软件实现，设计良好的业务构件可通过即插即用的方式，随业务变化而灵活重组。基于业务构件的RRIS框架研究(具有理论价值和现实意义)必将成为未来软件开发的重点。本文首先概要介绍框架有关概念，然后对RRIS框架的设计方案进行详细的比较分析，最后提出基于业务构件的RRIS的框架的建立过程。

2　框架概述

2.1　框架定义

框架的定义较多，反映出人们对框架内涵的不同理解。

李绪蓉　博士研究生，主要研究领域为分布式系统、系统集成与系统重构。凌兴宏　主要研究领域为Agent技术与系统集成。丁秋林　教授，博导，主要研究领域为CIMS技术与系统集成技术。

如果软件类型适合采用原型法和螺旋模型进行开发，以上过程也适用于演化型原型方法。但是如果软件本身的重点在用户界面和用户交互过程上，或者软件没有明显的关键功能，典型的如小型Web应用程序，多媒体教学系统等，在采用形式化方法上获利甚少；如果在快速开发平台上能够比较容易地产生部分功能的原型，那么这部分功能的开发也没有必要采用形式化方法。

在以上的开发过程中，对开发人员掌握形式化开发方法的要求相应降低了。由于整个开发过程采用了UML的过程框架，既可以采用螺旋模型控制整个开发过程，在每个螺旋过程中，又分成多个阶段。形式化方法在这些不同阶段中的应用层次完全不同，在问题定义与分析、可视化建模、代码生成、完善编码和测试等阶段的开发人员，只需要使用UML、面向对象语言及编程、RAD开发环境等开发技术，完全可以不涉及形式化技术。因此，软件开发过程中最重要的模型精化和模型检查过程采用形式化方法，加强了软件模型的精确程度，这也将提高软件的可靠性。

该方法的详细过程与规则和开发实例笔者将在后文中详述。

参考文献

1　周彦晖. 基于面向对象模型的形式化规约和FDOOM开发方法：[硕士论文]. 2002
2　The RAISE Method Group, George C, et al. The RAISE Development Method. TERMA Elecktronik AS, Denmark, 1999
3　Andrews J D, Groote J F, Middelburg C A. Semantics of Specification Languages. Workshops in Computing. Spring-Verlag, 1993
4　Bruun P M, et al. RAISE Tools Refernce Manual: [Technical Report LACOS/CRI/DOC/17]. CRI: Computer Resources International, 1995
5　Fowler M. UML Distilled (Second Edition). Addison-Wesley, 2000
6　Albir S. UML in a nutshell. O'Reilly, 1998
7　Rumbaugh B G, Jacobson J. The Unified Modeling Language User Guide. Addison-Wesley, 1999