形式化与可视化结合的FDOOM软件开发方法*)

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 Transform between Fomal 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 combines the formal development method with the visual development method. This study is based on the transform method and rules between the UML model and the RAISE model . We developes a new software development Method FDOOM (Formal Development based on Object Oriented Modeling) that combine the UML with the RAISE together. And there is a demo in the end.

Keywords Formalize, Visualize, UML, RAISE, Model transform

1. 前言

随着软件开发技术和开发平台的不断发展,面向对象的软件设计技术和实现技术已经成为主流。UML作为一种完全支持面向对象的建模语言,提供了从软件开发过程模型、软件模型建立、模型精化、流程分析到软件生成的能力。但是,UML缺乏明确的形式语义基础,对UML建立的软件模型的验证不能利用已有的数学工具,只能通过比较需求、评审原型等方式完成,不能很好保证关键软件的高质量和高可靠性要求。

另一面,很多学者也致力于整个软件开发过程可采用数学方法和工具来严格定义,各阶段建立的模型可从前一个阶段推导产生,并且可以进行证明。人们广泛采用数理逻辑、进程代数等数学工具并发展出时序逻辑、时段演算、Petri网等新的方法,并产生了一些形式化语言如Z、VDM、CSP等。但是这些方法都存在局限性,形式化方法的数学特点对程序员来说是一种挑战;形式化语言不能很好地支持面向对象开发;数理逻辑和进程代数等数学方法又不能很好地描述时序、交互、容错;一般来说形式化语言没有好的代码生成系统。

针对以上问题,笔者将采用将面向对象软件开发中广泛适用的UML开发过程与具有面向对象特征的RAISE形式化方法结合起来,提出一种新的软件开发方法FDOOM(Formal Development based on Object Oriented Modeling),既保留现有面向对象软件开发的优点——可视化,同时能为面向对象方法建立较为严格的规范描述能力,并能够形式化地对软件模型的精化过程进行验证或证明,提高面向对象开发关键软件的可靠性,并在一定程度上减少关键软件需求分析和模型精化的周期[3]。

为了实现以上的目标,笔者在文[8]中探讨了RAISE模型和UML的对象模型之间的联系,建立RAISE模型和UML模型之间的"桥梁",使用RAISE的RSL语言表示UML的对象模型,使得将RAISE开发方法结合到UML的开发过程中成为可能。

2. FDOOM方法

2.1 方法概述

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,我们主要考虑在UML的总的开发框架下(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,我们还充分利用了现有的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:

- 问题定义和可行性研究。
- 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。
- 结合可视化模型产生形式化初始规范过程。
- 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。
- 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。
- 生成代码,根据需要添加用户界面和总控模块。
- 检查、完善代码。
- 建立系统配置模型,生成系统。
- 测试与完善。

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,则采用形式化方法获利甚少;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。整个开发过程采用UML的过程框架,可以

*)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。

采用螺旋模型控制整个开发过程，在每个螺旋过程中，又可分成多个阶段。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。而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。

图1说明了将 RAISE 和 UML 结合的 FDOOM 开发方法的详细过程。

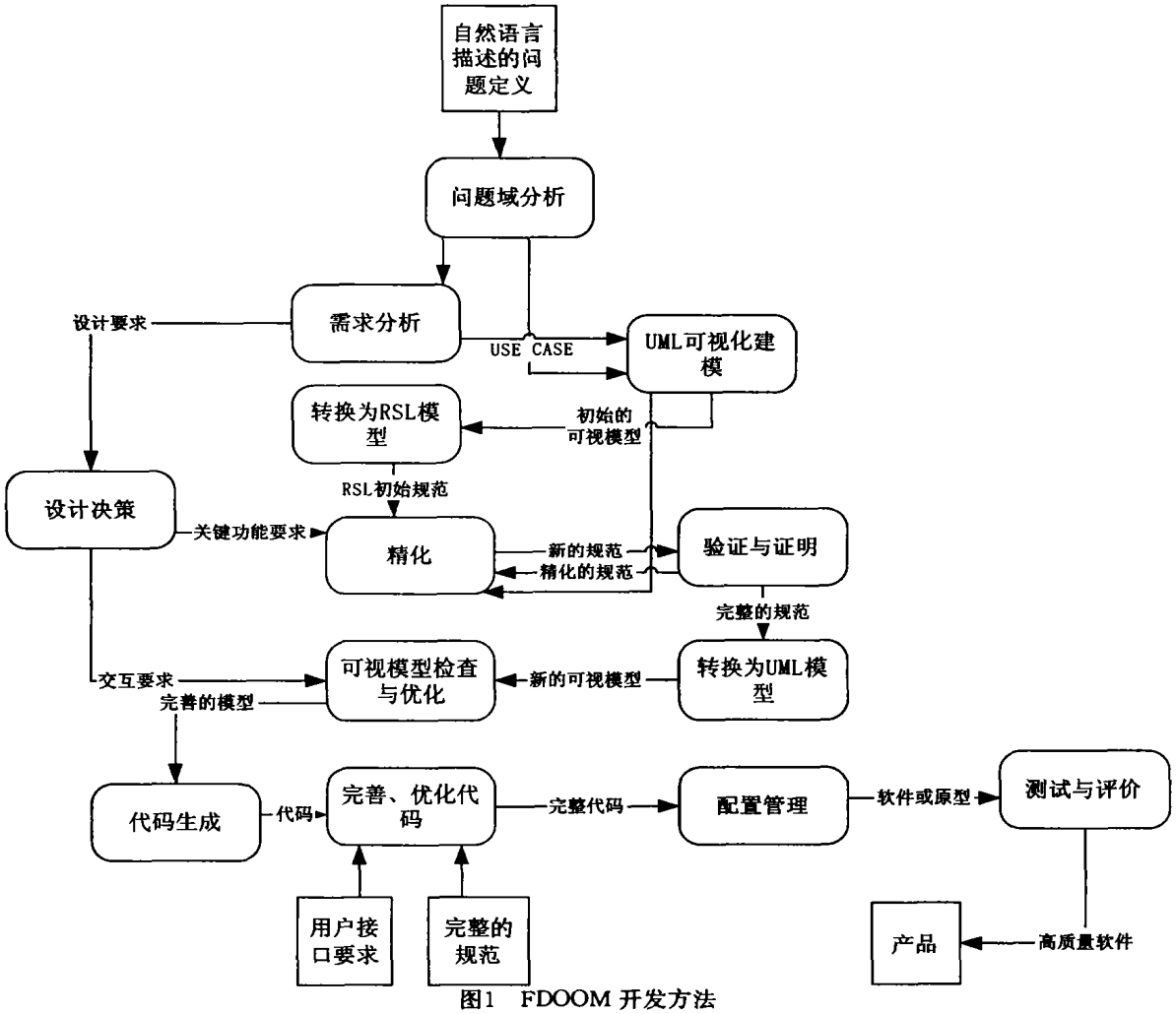

图1 FDOOM 开发方法

2.2 应用规则

在 UML 和 RAISE 结合的开发过程中，涉及到两次 UML 模型和 RAISE 模型的转换。在 UML 中，最重要的模型是对象类图，用 RSL 表示的初始规范也是从对象类图出发的，这两次转换是互逆的过程。显然，转换过程应该依据前面提到的使用 RSL 表示类关系所给出的基本结构，但是 RAISE 和 UML 在基本语法和语义上的差异，使得这种转换过程需要仔细分析。下面讨论转换中的几个典型的问题。

(1)Scheme *声明的处理* 在 UML 中对象类的所有元素都可以被 RSL 的 Scheme 所包含，同时，一个 Scheme 可以包含多个类的元素。在 RSL 中，Scheme 也只是抽象地表示一个方案，可以包括 class、object、type 的定义。这也就意味着 Scheme 本身没有必要进行转换，只需要对其中定义的元素进行转换。由于 RSL 的模块化特征，一个 Scheme 往往定义单一的一个或几个 class，可以将这些 class 分别转换成 UML 的对象类。如果某个 class 的形式化对象为其他 class 的参数，则应该变换成类包含其他类对象成员的关系。在下面的 RSL 定义中：

```
scheme
ELEM = class type Elem end,
DICTIONARY(KEY:ELEM,DESCR:ELEM) =
class
end
```

可见在一个 Scheme 中定义了两个 class，ELEM 的形式化对象 KEY 和 DESCR，因此应该产生两个 UML 的对象类 ELEM 和 DICTIONARY，其中，DICTIONARY 包含两个 ELEM 的对象成员。

(2)Type *声明的处理* RSL 中的 type 声明往往表示对某些类型的使用，这些类型是已经定义的。因此，这完全符合各种编程语言中类型的含义，但是 RSL 中的 type 声明只有一个，往往利用这个 type 来产生一个全局 object，包含了多个对象类的定义。所以 RSL 中的 type 声明可以变换成 UML 中的类成员说明，但是必须注意：type 定义中的类型基本上都对应与一个独立的 UML 对象类。type 定义中的 axiom 应该有针对性地作为对象类的方法。通过对 type 的变换，可以检查 UML 对象类是否完整。

RSL 中还存在一种可变类型声明，用来列举某些常量：type Color == red | green | blue。对于 C++，相当于：enum Color { red,green,blue}。

这样的 type 声明的含义对应于一般程序设计语言的枚举类型，因此可以变换 UML 模型中的抽象枚举类型。至于该枚举类型的详细定义可以在生成代码后参考 RSL 的规范再详细给出。

(3)Object 声明　Object 声明在 RSL 中同样作为模块声明，可以理解为直接定义某个 class 的实例。例如：

```
Object
    LIST_APPLY:
        class
            value
            ......
            axiom
      ......
        end
```

等价于：

```
scheme
    LIST =
        class
            value
            ......
            axiom
            ......
        end
```

因此在 UML 中没有必要进行特别的处理，可以根据实际情况转换成类的对象成员或是全局对象(代码生成后)。

(4)模块嵌套　在 RSL 中允许模块嵌套定义：

```
object
    INTEGER_LIST:
        class
            object
                INTEGER:class type Elem = Int end,
                LIST:PARAM_LIST(INTERGER)
            ......
                end
```

这样的定义表示 INTEGER 和 LIST 是 INTEGER_LIST 的成员，而且同样可以视为已经实例化的类。

这样的定义在一般的程序设计语言中都是不允许的，转换时需要将每个嵌套模块转换为一个独立的对象类，嵌套在内部的模块还应该转换为外部对象类的类对象成员。

3. FDOOM 方法实际应用参考实例

下面通过一个简单的管理信息系统的例子，应用前面给出的 FDOOM 方法，概要说明其开发过程。

3.1　初始需求描述

需要编写一个港口进行船只停泊管理的系统，当船只到达港口后，只能停泊到空闲而且与船只大小合适的泊位，如果没有合适泊位，船只将会在等待区等待泊位。这个系统能够让港口管理员控制船只进出港口，登记船只的到达、停泊和离开的情况。

3.2　使用 UML 建立基本模型

在上面的基本问题描述中，可以分析出如图2所示的基本 USECASE 图。

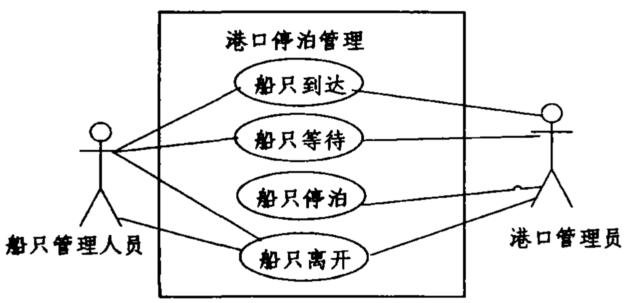

图2　USECASE 图

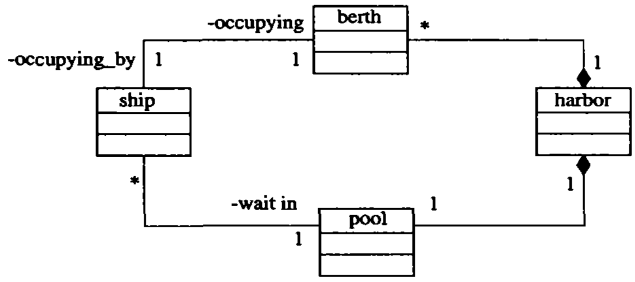

图3　初始对象类图

通过对问题的初步分析，在系统中存在四种基本对象：船只(ship)，港口(harbor)，泊位(berth)，等待区(pool)。

可以假设所有船只到达港口后都应该在等待区等待(即使是概念性的)，然后才能进入港口。接着产生初始的对象类图如图3，这时对象类基本不包括具体的方法，留到后面形式化规范和进化过程中产生。

3.3　初始形式化规范

在确定了对象之后，接下来需要确定各个对象的属性。这些属性的表示都应该采用 RSL 的形式化描述。

船只在本系统中的主要属性就是是否适合停靠某个泊位，可能因素是大小、吃水深度、装载的货物类型等。由于这些特性和具体需求有关，应该由相关的专业人员来界定，因此这里可以只给出一个抽象的函数：

fits:*Ship*×*Berth*→*Bool*

这个 fits 函数是待定义的，可以根据具体情况在实际软件开发中确定。

泊位的主要属性是泊位是空闲还是停泊有船只，因此可以设定一个占用(occupancy)属性，取值为空闲(vacant)和被某个船只占用(occupied_by(occupant:ship))。

港口(harbour)是泊位和等待区的聚集。所有基本属性应该包含一个泊位的数组和列表。等待区可以假设总是能够满足要求的，因此等待区没有特别属性，只是需要在港口中包含一个等待区属性，可以认为是等待区的对象成员。

对应以上属性的处理有很多选择，但是通过对需求的分析，这些属性至少应该满足以下的一致性条件：①一条船只不能同时停靠在两个地方；②任何一个泊位都只能停靠一条船；③ 船只只能停靠在合适的泊位。这三个条件形式化的描述是：

```
∀ s:Ship•
~(waiting(s) ∧ is_docked(s)) ∧
(∀ b1,b2:Berth•
occupancy(b1)=occupied_by(s) ∧ occupancy(b2)=occupied_by(s)⇒
b1=b2) ∧
(∀ b:Berth•occupancy(b)=occupied_by(s)⇒fist(s,b))
```

而且，所有对象的行为或方法都应该满足以上的约束条件，在 RSL 中就是每个相关的函数应该有相应的公理限制。例如船只到达这个行为，必须满足的公理是：

```
[arrive_consistent]
∀ s:Ship•
arrive(s)post consistent() pre consistent() ∧ can_arrive(s)
```

由于 RSL 的特点决定了主模块只有一个，可以采用主模块来描述港口对象，而港口对象是泊位和等待区的聚集，因此需要在港口的模块中使用 ship 和 berth 类型，所以首先定义：

```
scheme TYPES =
    class
        type
            Ship,Berth,
```

```
        Occupancy == vacant | occupied_by(occupant:Dhip)
    value
        fits:Ship × Berth → Bool
end
object T:TYPES
```

下面定义系统的抽象规范，其中可以利用前面已经得到的一致性条件，使用已经定义的类型。具体的方法可以依据RSL中所规定的步骤[2]。

于是可以得到初始规范A_HARBOUR0(由于篇幅关系，此处略)。

3.4 精化初始化规范并形成精化后的UML模型

对应初始化规范，可以从以下几个方面验证是否符合初始需求的要求：(1) 船只到达能够被正确记录；(2) 存在合适的空闲泊位时船只能够停泊；(3) 停靠的船只可以离开；(4) 船只只能停靠在合适的泊位。

在初始规范A_HARBOUR0规范中，并没有考虑pool和berth的具体性质，因此状态改变函数是抽象定义，下面通过定义pool和berth对象，精化初始规范得到新的形式化初始规范A_HARBOUR1[1]。

要保证精化的有效性，需要证明A_HARBOUR1是A_HARBOUR0的精化，也就是要证明A_HARBOUR0的所有axiom在A_HARBOUR1的函数中都满足。

为了说明该证明过程，从A_HARBOUR0中抽取出船只是否能够停靠的形式化初始需求如下：

```
[waiting_docks]
∀ h:Harbour, s1, s2:T.Ship, b:T.Berth·
    waiting(s2, docks(s1,b,h)) ≡ occupancy(b,h)
        pre can_arrive(s,h)                          (1)
```

在A_HARBOUR1中waiting函数被精化为：

```
waiting: T.Ship × Harbour → Bool
waiting(s,(ws, bs)) ≡ P.Isin(s, ws)                  (2)
```

若A_HARBOUR1是A_HARBOUR0的精化，则(2)式应使(1)式成立，证明如下：

```
[waiting_docks]
waiting(s2,docks(s1,b,h) ≡ s1 ≠ s2 ∧ waiting(s2,h) =>
P.Isin(s2,docks(s1,b,h) ≡s1 ≠ s2 ∧ waiting(s2,h) =>
P.Isin(s2,(P.remove(s1,ws),B.change(T.indx(h),T.occupied_by
(s1),bs) =>
P.Isin(s2,(p.remove(s1,ws),B.change(T.indx(h),T.occupied_by
    (s1),bs))) ≡
    s1 ≠ s2 ∧ P.Isin(s2,h) =>
P.Isin(s2,(ws \ s2, bs † s1) ≡ s1≠s2 ∧ P.Isin(s2,h) =>
若 s1=s2 P.Isin(s2,ws \ s1) ≡ false
          s1≠s2 ∧ P.Isin(s2,h) ≡ false
若 s2≠s2 (ws \ s1, bs † s1) ≡ h
          => P.Isin(s2,h) ≡ s1≠s2 ∧ P.Isin(s2,h)
```

□

其余证明都可以按类似方法完成。

最后，利用UML模型和RSL模型的对应关系进行模型转换，这里需要注意转换过程中对scheme、type、object的处理。然后可以根据形式化规范建立顺序图、协同图，生成代码后，利用前面的规范具体编写完整代码[1]。

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: C. George, A. E. Haxthausen, S. Hughes, R. Milne, S. Prehn, J. S. Pedersen. The RAISE Development Method. TERMA Eleckronik AS, Denmark, 1999

3 Andrews D J, Groote J F, Middelburg C A, et al. 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 Booch G. Rumbaugh, J.Jacobson. The Unified Modeling Language User Guide. Addison-Wesley,1999

8 周彦晖，张为群. 软件形式化与可视化软件模型的转换. 计算机科学,2003,30(7)

(上接第66页)

的存在，按顺序求取的F-S鉴别矢量集整体性能并非"最优"。在提出的综合反映单个鉴别矢量优劣与鉴别矢量间彼此相关性大小的评价函数$f(\xi)$的基础上，设计出两种F-S鉴别矢量的优选方案，并在Concordia大学CENPARMI手写数据库上进行了实验。实验表明，优选出的鉴别矢量集中位置靠前的鉴别矢量由于彼此间相关性较小，基于这些矢量作特征抽取的识别结果优于相同数目的按顺序排列的F-S鉴别矢量集的性能。实验验证了鉴别矢量间相关性大小对特征抽取的重要性。

进一步，我们认为对于F-S方法求解第i个鉴别矢量的一个优化模型应为如下多目标规划问题模型

$$\begin{cases} \max R(\xi) \\ \min g(\xi) \\ \xi^T\xi_j=0 \end{cases} \tag{22}$$

其中，

$$g(\xi)=\sum_j|\rho(\xi,\xi_j)| \quad j=1,2,\cdots,i-1 \tag{23}$$

依据最优化理论，模型(4)所得的最优解只是模型(22)的弱有效解。

参考文献

1 Sammon J W. An optimal discriminant plane. IEEE Trans Comput,1970,19(9):826～829

2 Foley D H,Sammon J W. An optimal set of discriminant vectors. IEEE Trans Comput,1975,24(3):281～289

3 Duchene J,Leclercq S. An optimal transformation for discriminant and principal component analysis. IEEE Trans on pattern analysis and machine intelligence,1988,10(6):978～983

4 Hamamoto Y,Matsuura Y,Kanaoka T,et al. A note on the orthonormal discriminant vector method for feature extraction. Pattern Recognition,1991,24(7):681～684

5 Liu K,Cheng Y Q,Yang J Y. A generalized optimal set of discriminant vectors. Pattern Recognition,1921,25(7):731～739

6 Guo Y F,Shu T T,Yang J Y,et al. Feature extraction method based on the generalized Fisher Discriminant criterion and face recognition. Pattern Analysis & Application, 2001, 4(1): 61～66

7 Jin Z,Yang J Y,Hu Z S,et al. Face recognition based on the uncorrelated discriminant transformation. Pattern Recognition, 2001,34:1405～1416

8 丁学仁，蔡高厅. 工程中的矩阵理论. 天津：天津大学出版社，1985

9 Jin Z,Yang J Y,Tang Z M,et al. A theorem on the uncorrelated optimal discriminant vectors. Pattern Recognition,2001,34:2041～2047

10 Yoshihiko H,et al. Recognition of handwritten numerals using Gabor features. In: Proc. of the Thirteenth ICPR. 250～253

11 Liao S X,Pawlak M. On image analysis by moments. IEEE Trans Pattern Analysis and Machine Intelligence,1996,18(3):254～266