第 35 卷 第 2 期 计 算 机 工 程 2009 年 1 月
Vol.35 No.2 Computer Engineering January 2009
·软件技术与数据库· 文章编号：1000—3428(2009)02—0021—03 文献标识码：A 中图分类号：TP311

基于 rCOS 的 UML 状态图语义研究

张晓蒙[1,2]，戎 玫[3]，张广泉[1,2]
(1. 苏州大学计算机科学与技术学院，苏州 215006；2. 重庆师范大学数学与计算机科学学院，重庆 400047；
3. 暨南大学深圳旅游学院，深圳 518053)

摘 要：统一建模语言(UML)中的状态图用于描述类的对象所有可能的状态及事件发生时状态的转移条件，从而进行系统动态分析。针对现有关于 UML 状态图形式化语义研究中存在的不足，该文提出基于统一程序设计理论的对象精化演算系统，用于描述 UML 状态图的形式化语义，给出与类图、序列图的一致性检验，为模型驱动开发提供了可行性。
关键词：统一建模语言；对象精化演算系统；状态图

Semantic Research on UML State-chart Based on rCOS

ZHANG Xiao-meng[1,2], RONG Mei[3], ZHANG Guang-quan[1,2]
(1. School of Computer Science & Technology, Soochow University, Suzhou 215006; 2. Dept. of Mathematics and Computer Science, Chongqing Normal University, Chongqing 400047; 3. Tourism College, Jinan University, Shenzhen 518053)

【Abstract】 The state-chart of Unified Modeling Language(UML) is used to describe all the possible states and the transmisson conditions of states when the events occurre to analyze the system dynamically. This paper proposes refinement Calculus of Object Systems(rCOS) based on Unifying Theories of Programming(UTP) to specify the formal semantic of UML state-chart, which overcomes the limitation in the semantic research. And it provides the consistency checking with class diagram and sequence diagram, and gives the possibility of MDA development.
【Key words】 Unified Modeling Language(UML); refinement Calculus of Object Systems(rCOS); state-chart

1 概述

统一建模语言(Unified Modeling Language, UML)是一种图形化的语言，用于明确规定和构建软件系统开发过程中形成的产品，使这些产品可视化并形成文档资料。UML 中不同的模型图被用在软件开发的各个阶段中，并且它们抽象的程度也不同。由于不同的图提供了不同的视角，因此出现了有争议的问题：(1)一致性(consistency)问题，各种观点下的模型在语法和语义上应该是协调的；(2)转化(transformation)和演化(evolution)问题，一个模型必须与它的精化在语义上是协调的；(3)可跟踪(traceability)问题，在某个观点下的模型中的一个变化应该可以造成其余观点下模型的协调性变化；(4)融合性(integration)问题，不同观点下的模型应该在软件成品之前被无痕地融合在一起。

要很好地处理以上问题，最好为 UML 模型提供形式化语义。通常 UML 模型的形式化方法被归于 2 大类：(1)直接方法，把 UML 图转换为现有的形式语言体系，如 Z, B, VDM, CSP, Petri-Nets, PSV。(2)直接给出 UML 图提供形式化语义。本文的研究属于第(2)种方法，在文献[1-3]基础上，通过直接给出状态机图的形式化语义并转化为对象精化演算系统(refinement Calculus of Object Systems, rCOS)规范形式，指出一个系统的模型由一个类图、一簇序列图和一簇状态机图构成，提供了序列图与状态机图的一致性检验。

2 rCOS

rCOS 是一种面向对象描述语言，它具有丰富的语言特征，包括子类(subtypes)、可见性(visibility)、继承(inheritance)、动态绑定(dynamic binding)和多态性(polymorphism)。它能够很好地描述面向对象设计和程序，此外，rCOS 是以文献[1]中的统一程序设计理论(Unifying Theories of Programming, UTP)为基础的。rCOS 的语义模型提供了一个演算用以支持面向对象设计中的结构和行为精化(refinement)，从而可以把文献[2]给出的 UML 形式化模型定义为需求模型和设计模型及其精化关系。

2.1 rCOS 语法

rCOS 语法包括面向对象系统、类声明、命令和表达式，其主要部分与 Java 语法相似。

2.1.1 面向对象系统

一个面向对象系统 S 形如 cdecls.P，其中，cdecls 是一有限个类的声明；P 是主方法且形如(glb, c)，glb 是全局变量与其类型的集合，c 为一个命令。由于 P 无法直接获得类属性，因此 P 只能操作 glb 中的变量以及调用声明部分的类中的方法。

2.1.2 表达式

表达式由规则 e::=x|null|self|e.a|f(e)构造，其中，null 表示特殊类 NULL 的一个特殊对象，它可以是所有类的子类；self 用来表示当前时刻所有的活动对象；e.a 是 e 的一个属性 a；f(e)表示一个内嵌表达式。

2.1.3 类声明

一个声明形如 cdecls:=cdecls|cdecls，其中，cdecl 是一个如下形式的类声明：

基金项目：江苏省高校自然科学基金资助项目(05KJB520119)；重庆市自然科学基金资助项目(CSTC，2006BB2259)
作者简介：张晓蒙(1983－)，男，硕士研究生，主研方向：软件工程，形式化方法；戎 玫，副教授、博士；张广泉，教授、博士
收稿日期：2008-07-30 **E-mail：**doom1983713@126.com

Class N extends M{

private U_1 $u_1 = a_1$, U_2 $u_2 = a_2$,…,Um $u_m = a_m$;

protected V_1 $v_1 = b_1$, V_2 $v_2 = b_2$,…,Vn $v_n = b_n$;

public W_1 $w_1 = c_1$, W_2 $w_2 = c_2$,…,W_k $w_k = c_k$;

method m(paras){c}　}

其中，N 和 M 为类名，且 M 为 N 的直接超类；private 声明了类的私有属性、类型及初始值；pubilc 与 protected, private 相同；method 声明了方法，其中的 paras 是一列 m 的参数，c 为 m 的方法体。

2.1.4　命令

rCOS 语言支持典型的面向对象编程构造：

c::= skip | chaos | var T x = e　　//终止|忽略|本地变量声明

end x | c;c | c ◁ b ▷ c　　//解除声明|顺序|条件选择

| c Π c |b * c|le.m(e,v,u)　　//非确定性选择|迭代|方法调用

| le := e| C.new(x)　　//指派| 创建新对象

2.2　rCOS 语义

rCOS 语义是基于文献[1]提出的统一程序设计理论。

3　状态机形式化语义

3.1　形式化定义

UML 使用状态机描述类的对象的活动，它包含对象的所有状态以及在特定事件(例如方法调用)上的转移。本文只对简单状态图进行讨论，描述方法调用(如触发事件)和状态的变化而不关心方法的具体功能实现。具体的功能细节可以通过在转移上添加三元组 t=<Event, Action, Guard>来满足，它表示一个触发事件 Event 在满足 Guard 条件时进行 Action 活动。所有状态机都是被动的，对象的活动被触发只有通过方法调用，因此，Guard 与 Action 是可选的，Event 是必需的。

状态机可以定义为一个元组<S,L,s_0,T>，其中，S为状态或控制位置的集合；$s_0 \in S$为初始状态；L是一个有限集合，形如<Event, Action, Guard>，其中的Event是类中的一个方法，称为触发事件，Action是一个rCOS命令，Guard为一个布尔表达式，限定类的属性以及Action中的变量；T是一个关系，$T \subseteq S \times L \times S$，通常把T中每一个元素称为转移。一个转移t=<s,l,s′>可被表示成$s \xrightarrow{l} s'$，转移t可把状态s转移到s′。

假设 loc 是一个作为类私有属性引入的状态变量，标记 l 在状态机中表示触发事件，从而控制状态由转移 $s \xrightarrow{l} s'$的改变可以用 rCOS 描述为一个监护设计：

$cs(l) \overset{def}{=} (loc = s)^T; (true \vdash loc' = s')$

其中，监护g^T被定义为一个条件选择skip ◁ g ▷ false，通常情况下忽略在设计true ⊢ R中的前置条件true。

对于每一个触发事件m()，可以得到一个转移集合T(m()), m()为该集合中每一个转移的触发事件。则状态图的行为以及给定类的对象在调用m()后的动作可以通过方法定义被描述为：$m(){\{ \Pi_{l \in T(m())} cs(l)\}}$。

3.2　实例研究

本文以一个简化的战斗机导弹系统为例给出其中发射器(launcher)对象的状态机图，如图 1 所示。发射器在接收到锁定消息(LockSignal)后设定锁定目标的坐标位置，接收到开火消息(FireSignal)后预热导弹，等候中央控制器的发射命令。如果此时雷达报警有敌机导弹袭击，则转入发射等待状态，并设置敌机导弹的坐标位置，此时中央控制器发出发射消息，发射自杀式导弹，引爆来袭导弹。如果没有敌机导弹袭击，直接接收发射命令后发射攻击导弹摧毁原定目标。任务成功后重新进入待命状态并重装上导弹。

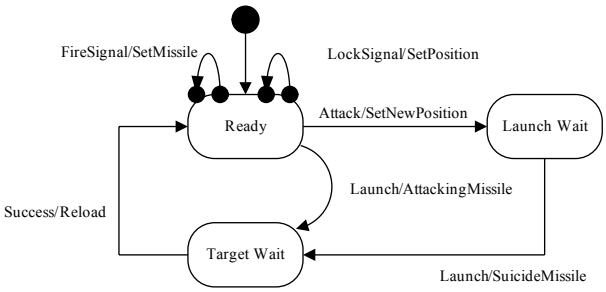

图 1　发射器对象的状态机图

由于初始伪态(initial pseudostate)并不是一个状态[3]，因此本文把Ready作为初始状态，则s_0 = Ready。所有转移上都有标签，且本文不考虑监护条件，则有

L={<LockSignal, SetPostion, >, <FireSignal, SetMissile, >, <Attack, SetNewPosition, >, <Launch, AttackingMissile, >, <Launch, SuicideMissile, >, <Success, Reload, > }

状态机图中状态 S 的集合为 S={Ready, LauchWait, TargetWait }，而且转移 T 的集合为

T = {<Ready, LockSignal, Ready>, <Ready, FireSignal, Ready>, <Ready, Launch, Ready>, <Ready, Attack, LaunchWait>, <LaunchWait, Launch, TargetWait>, <TargetWait, Success, Ready>}

在图 1 中，转移的动作已经在类 LaunchMissile 中由 rCOS 描述。本文只处理状态机图，rCOS 对于类图及序列图的描述见文献[2-3]。

LaunchMissile::SetPosition() { $(loc = Ready)^T$；loc′= Ready}

LaunchMissile::SetNewPosition(){

$(loc = Ready)^T$; loc′= LaunchWait}

其他动作同理可描述。

一个标记为l=<Event,Action,Guard>的转移$s \xrightarrow{l} s'$可描述为：$Spec \overset{def}{=} Guard^T;(cs(l)||Action)$，其中，$(p_1 \vdash R_1)||(p_2 \vdash R_2) \overset{def}{=} (p_1 \wedge p_2 \vdash R_1 \wedge R_2)$。设$l_i$=<m(), Action, $Guard_i$>, $i \in \{1,2,\cdots,k\}$为类C的状态机中的转移标签，其被事件m()触发，可以被描述为方法定义：$C::m(){\{Spec(l_1) \Pi Spec(l_2) \Pi \cdots \Pi Spec(l_k) \Pi (\neg(\bigvee_{i=1}^{k} Guard_i) \vdash chaos)\}}$。

为了更精确地描述状态机，可以引入一个 CHAOS 状态，任何事件都可以触发一个 CHAOS 到自身的转移。同样，对于一个状态上可执行的转移，若其执行的动作为 chaos，则会把该状态转移到 CHAOS。UML 语言中通常会在状态图中忽略混沌状态，因此，本文不做进一步讨论。

3.3　模型一致性讨论

文献[4]指出，一个软件系统的开发周期总是从需求模型的构建开始。在UML中，需求模型由一个类模型和若干用例组成，概念类没有方法，而且概念类之间的关联是无向的。由此需求模型RM可定义为RM=(Γ_r,U_r,Δr)，其中，Γ_r为一个概念类模型；U_r为一用例图(被定义为一用例控制类)；Δr为一簇系统序列图，其中每一个对应一个用例。

本文把状态机图添加到上述定义的需求模型中，从而可以把需求模型重新定义为RM=<Γ_r,U_r, Δr, Ω_r>，其中，Ω_r为一簇状态机图。需求模型中的状态机图Ω_r只包含用例控制类，通过状态机图来描述用例控制类对象的活动。序列图中消息的顺序与用例控制类的状态机图决定了用例控制类中方法的定义。对于每个用例U，用例控制类声明的操作出现在序列图中并且用例控制类中每个方法的方法体被定义为在状态图

中被方法调用所触发的动作。对于上述定义的需求模型RM，用rCOS可以描述为

$$\text{Spec(RM)} \overset{\text{def}}{=} \text{cdecls}_{\Gamma_r}$$

U_1-Controller; U_2-Controller;…;U_n-Controller

其中，$U_i(i=1,2,\cdots,n)$是RM的用例。

需求模型的一致性需要保证序列图和状态机图的一致性，而序列图和状态机图的一致要求状态机图中事件发生的顺序和序列图中消息传送的秩序相同，图 2 为战斗机导弹系统的一个序列图。可以看出，在发生敌机导弹袭击时，仍然先摧毁锁定的目标，然后摧毁敌机导弹，这明显与上文的状态机图不一致。

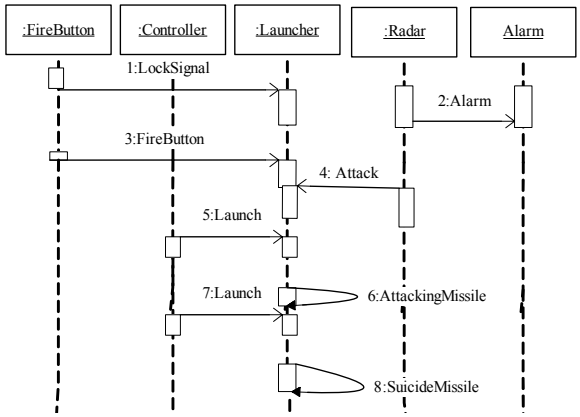

图 2　用例控制类 LaunchMissile 的序列图

本文通过如下方法进行状态机图与序列图的一致性检查：首先在序列图中搜集由状态机图中描述的对象所接收和发送的消息，按消息先后顺序形成一个消息链: $msg_1, msg_2,\cdots, msg_n$，若在序列图中包含并发情况，则可能有多条消息链，必须检查每一条链。

如果序列图中某条消息的接收对象就是状态机图的描述对象，则此消息对应状态机图中转移的事件 Event，如果消息的发送者是状态机图的描述对象，则此消息对应状态机图中转移的动作 Action。形式化的可通过消息链得到序列图产生的标签<Event, Action, Guard>，其中，Event 必须是相应消息中的方法名；Guard 为消息的监护条件；若下一条信息的接收者是此对象，则 Action 为这条消息的方法体，否则，如果下一条信息的发送者为状态机图描述的对象，则 Action 为下一条信息中的方法体。通过交替遍历消息链得到的标签和状态图，可以比较这簇标签是否与状态机图相同，从而判定模型是否满足一致性。判定方法如下：

步骤 1 对第 1 个标签，从状态机图初始状态s_0开始遍历状态机图，如果状态机图中从s_0出发的所有转移中正好有一标签与消息链产生的第 1 个标签相同，那么可以得到后继状态s′，并把此标签附到s′上，进入步骤 2。如果不止一个标签与消息链产生的第 1 个标签相同，则进入步骤 3。如果没有标签与消息链产生的第 1 个标签相同，则进入步骤 4。

步骤 2 从 s′开始，遍历状态机图，如果有转移标签与消息链产生的下一个标签相同，则把下一个标签附在 s′的后继状态上，得到新的 s′。如果不止一个标签与消息链产生的标签相同，则进入步骤 3。如果没有标签与消息链产生的标签相同，则进入步骤 4。否则，重复执行步骤 2。

步骤 3 状态图具有二义性。

步骤 4 序列图与状态图不一致。

步骤 5 如果能按照消息链产生的所有标签成功遍历状态图，则状态图与序列图满足一致性。

图 2 中的发射器对象的消息链产生的标签如下：

Message	Generating Label	Attached State
1.LockSignal	LockSignal/SetPosition	Ready
3.FireSignal	FireSignal/SetMissile	Ready
4.Attack	Attack/SetNewPosition	Ready
5.Launch	Launch/AttackingMissile	LaunchWait
7.Launch	Launch/SuicideMissile	

通过产生的标签遍历上文中的状态机图的过程如下：

Current Object State	Transition Label	Post-Object State
Ready	LockSignal/SetPosition	Ready
Ready	FireSignal/SetMissile	Ready
Ready	Attack/SetNewPosition	LaunchWait
LaunchWait	Launch/SuicideMissile	ERROR

当对于消息链产生的标签 Launch/AttackingMissile 遍历状态机图时，从状态 LaunchWait 开始没有标签与 Launch/AttackingMissile 相同，说明序列图与状态图并不一致。

4　结束语

目前对 UML 的形式化工作主要致力于单个图的形式化，并且仅仅处理 1 种观点或 2 种观点下的模型的协调性，不同的组织都意欲强调不同的符号系统，使用全部甚至扩展的 UML 序列图或状态机的表达力，这无疑使 UML 失去了在多重观点下建模的优势，同时增加了某类 UML 模型的复杂性、削弱了其他 UML 模型应起的作用。在此类工作中，文献[5]以 XYZ/E 形式化语言构建 UML 活动图，并给出其语义；文献[6]通过 PSV 形式化 UML 模型，给出了 UML 状态图到 PVS 规范的转换模型与规则。而对 UML 完整模型的一致性演化方面的研究都只集中在处理类图与序列图，例如文献[2]给出了 UML 类图的形式化语义；文献[7]分别从静态和动态 2 个方面给出了 UML 序列图的形式化语义；文献[4]提出了融合 UML 模型的思想，通过模型的融合检验模型间的一致性，并给出了初步的模型演化规则。本文把基于 rCOS 描述的状态机图引入模型中，进一步完善了需求模型的描述，并对一致性条件进行了研究。下一步是对需求模型向设计模型演化过程中类图、序列图与状态机之间的一致性进行研究。

参考文献

[1] Hoare C A R, He Jifeng. Unifying Theories of Programming[M]. [S. l.]: Prentice-Hall, 1998.

[2] 杨　静, 张明义, 刘志明. 精化 UML 模型[J]. 计算机科学, 2007, 34(3): 250-253

[3] Fowler M. UML 精粹[M]. 3 版. 徐家福, 译. 北京: 清华大学出版社, 2005.

[4] Liu Zhiming, Li Xiaoshan, Liu Jing, et al. Integrating and Refining UML Models[R]. Macao, China: The United Nations University, Tech. Report: No.295, 2004.

[5] 朱雪阳, 唐稚松. UML 活动图的时序逻辑语义[J]. 计算机研究与发展, 2005, 42(9): 1478-1484

[6] 赖明志, 尤晋元. 从 UML 状态图到 PVS 规范的自动转换、验证[J]. 电子学报, 2002, 30(12A): 2122-2125.

[7] Li Xiaoshan, Liu Zhiming, He Jifeng. A Formal Semantics of UML Sequence Diagrams[R]. Macao, China: The United Nations University, Tech. Report: No.292, 2004.