# UML Statechart 图的操作语义*

李留英[1], 王 戟[1,2] 齐治昌[1]

[1](国防科学技术大学 计算机学院,湖南 长沙 410073);

[2](武汉大学 软件工程国家重点实验室,湖北 武汉 430072)

E-mail: lyli@nudt.edu.cn

http://www.nudt.edu.cn

**摘要**: 面向对象标准建模语言 UML(unified modeling language)缺乏精确的动态语义.根据 UML1.1 语义文档,提出描述对象状态机的 UML Statechart 图的形式化操作语义.该语义覆盖了 UML Statechart 图的绝大部分特征,为 UML Statechart 图的代码产生、模拟和测试用例生成奠定了基础.根据上述语义,基于 Rose98 完成了 UML Statechart 图的测试用例生成和测试过程的模拟.

**关键词**: UML(unified modeling language);操作语义;测试

**中图法分类号**: TP311 **文献标识码**: A

UML(unified modeling language)[1]是对象管理组织 OMG 的第三代面向对象建模语言.它定义了多种图元,从不同视角和层次描述系统的静态结构和动态特性.自 UML 推出之后,计算机界广泛关注并纷纷推出对应的工具,如 Rational 公司的 Rose98 和 iLogix 公司的 Rhapsody[2].OMG 也成立专门小组来负责 UML 符号的标准化,并广泛征集 UML 的精确语义.

软件的 UML 模型可以作为测试该软件的依据,辅助测试数据的生成.一个直接的应用是用 UML Statechart 图所刻画的对象状态机作为类层测试模型,但是 UML 建模时通常不考虑 UML Statechart 图的状态和迁移的形式化定义.Statecharts 已经存在许多种语义解释[3~7].现有的 UML Statechart 图操作语义主要将 UML Statechart 图[8]转换为适合于验证工具 SPIN 的中间描述机制——层次自动机(EHA),以便进行形式化验证,这种转化摒弃了对象状态图本身的一些特性.

为此,我们根据 UML1.1 语义文档[1],比较全面地刻画了 UML Statechart 图的建模元素和特征,尤其是层次结构、复合迁移、伪状态.在定义语法和语义时,借鉴了文献[3,6,7]的一些基本思想,同时引入了多个"事件队列"描述 UML Statechart 图的单事件处理方式.该语义为代码产生、模拟、测试和测试用例的重用奠定了基础.

## 1 形式化语法

一个面向对象软件系统由一组类 $CN=\{c|c$ 是类$\}$组成.类 $c\in CN$ 的标记(signature)$\Sigma_c=(Attr_c, OP_c)$,其中 $Attr_c$ 和 $OP_c$ 分别表示类 $c$ 的属性集合和操作集合.每个类 $c$ 标记为$(\Sigma_c, SM_c)$,$\Sigma_c$ 和

* 收稿日期: 1999-11-22; 修改日期: 2000-07-10

基金项目: 国家自然科学基金资助项目(69973051);国家 863 高科技发展计划资助项目(863-306-ZT06-04-1);武汉大学软件工程国家重点实验室基金资助项目;霍英东青年教师基金资助项目(71064)

作者简介: 李留英(1972-),女,江苏泰兴人,博士,讲师,主要研究领域为 UML,面向对象测试;王戟(1969-),男,上海人,博士,副教授,主要研究领域为软件方法学,形式化方法,软件工程;齐治昌(1942-),男,北京人,教授,博士生导师,主要研究领域为软件工程.

© 中国科学院软件研究所 http://www.jos.org.cn

$SM_c$ 分别表示类的接口与类实例(对象)的状态机.设 $ID_c$ 表示类所创建的所有对象标识符集合,以 $ID_c$ 中元素 $id$ 为标识符的对象具有类 $c$ 的所有属性,并假定对象的状态机完全等同于类的状态机.在 UML 中,状态机也称为状态图(statechart diagram),它表现一个对象或一个交互在其整个生存期内接受外界激励时的状态变迁过程.它附属于一个类或者一个方法.UML Statechart 图是经典 Harel's Statecharts[3]的(面向对象)变体.经典 Statecharts 是一种扩展的有限状态机,是在传统有限状态机上增加层次、并发和广播通信机制(如图 1 所示)[3],是一种强有力的、灵活的状态迁移图.

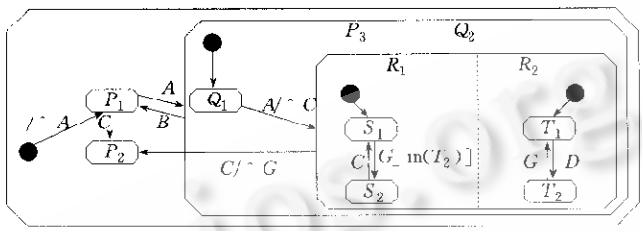


Fig.1 Running example of UML diagram
图1 UML 图实例

Statecharts=有限状态图+深度+正交性+广播通信机制.

与传统 Statecharts 类似,UML Statechart 图的基本元素有状态、迁移和动作.状态是对象在一定时期内的存在条件.它不仅包括状态名,还包括进入动作(entry action)、离开动作(exit action)和内部迁移等.迁移是对象对事件作出的响应.动作包括产生新事件、修改属性值和调用其他对象的操作.两者的差别在于 UML Statechart 图采用新的迁移优先级、禁止事件的合取、支持单事件处理机制.

在定义语法和语义时,假定系统当前的对象集为 OBJ,每个对象状态图的状态用抽象符号表示.若无特殊说明,通常都省略对象标识符下标 $id$.

### 1.1 状 态

UML Statechart 图的状态分为基本状态、OR-状态、AND-状态和伪状态.OR-状态和 AND-状态均是一种状态精化,使得一个状态包含多个子状态,实现状态层次结构和并发.伪状态又称连接子,是迁移路径上的暂时点.它包括初始(initial)伪状态、浅度(shallow)历史、深度(deep)历史、join、fork、条件分支(branch)和终止伪(final)状态.UML Statechart 图用方框表示状态,用有向箭头表示状态之间的迁移,方框内包容的方框是该状态的子状态,实心圆圈表示初始伪状态.

**定义 1.** UML Statechart 图的状态层次结构 $SH=(S,\rho,type,\gamma,\omega,\theta)$,其中:

(1) $S$ 是非空有穷状态集.

(2) $\rho:S\to 2^S$ 是状态精化函数,刻画状态间的层次(父/子)关系,$\rho(s)$定义状态 $s$ 的子状态集.每个 UML Statechart 图存在唯一根状态 $root\in S$,且$\forall\ s\in S:root\notin\rho(s)$.存在初始状态 $init\in S$ 且 $init\in\rho(root)\wedge\rho(init)=\varnothing$,$init$ 是 $root$ 的子状态.$\rho^*(S)=\bigcup_{i\geqslant 0}\rho^i(S)$和 $\rho^+(S)=\bigcup_{i\geqslant 1}\rho^i(S)$分别表示状态 $s$ 的包含自身的子孙和不包括自身的子孙;对 $X\subseteq S$,$\rho^*(X)=\bigcup_{s\in X}\rho^*(S)$.

(3) $type:S\to\{AND,OR,BASIC,PSEUDO\}$是状态类型函数,其中:

(a) 如果 $\rho(s)\neq\varnothing$且 $type(s)=OR$,称 $\rho(s)$是状态 $s$ 的 XOR 分解.当对象处于状态 $s$ 时,实际处于 $s$ 的一个子状态;

(b) 如果 $\rho(s)\neq\varnothing$且 $type(s)=AND$,称 $\rho(s)$是状态 $s$ 的 AND 分解.当对象处于状态 $s$ 时,对象实际同时处于 $s$ 的所有子状态上,

© 中国科学院软件研究所 http://www.jos.org.cn

$\forall s_1 \in S:(type(s_1)=AND \rightarrow \forall s_2 \in \rho(s_1):type(s_2)=OR)$;

(c) 如果 $\rho(s)=\varnothing$ 且 $type(s)=$BASIC,则 $s$ 是基本状态;

(d) 如果 $\rho(s)=\varnothing$ 且 $type(s)=PSEUDO$,则 $s$ 是伪状态.

(4) $\gamma:H \rightarrow 2^S$ 是历史函数,$H=H_s \cup H_d$ 是历史符号集,$H_s$ 和 $H_d$ 分别是浅度历史和深度历史.$\gamma$ 满足如下要求:$\forall x \in H, \gamma(x)$ 是 OR-状态集,即 $H$ 所隐含的只有 OR-类型状态. 函数 $\omega:S \cup H \rightarrow 2^S$ 定义为 $\omega(s)=\begin{cases}\{s\} & s \in S \\ \gamma(s) & s \in H\end{cases}$.

(5) $\theta:S \rightarrow 2^{S \cup H}$ 是缺省函数,定义 OR 状态 $s \in S$ 的缺省子状态和历史符号. 只有当 $\rho(s) \neq \varnothing \wedge type(s)=OR$ 时,$\theta(s)$ 有定义. 若 $x \in \theta(s)$,则对 $x \in S$,有 $x \in \rho^1(s)$,对 $x \in H$,有 $\gamma(x) \in \rho^*(s)$.

函数 $up:(S \backslash \{root\}) \rightarrow S$ 定义状态 $s \in S \backslash \{root\}$ 的父状态. 若 $up(s)=u$,则 $u$ 是 $s$ 的父状态. $root$ 是唯一的一个没有祖先的 $OR$-状态. $up^*$ 是 $up$ 的自反传递闭包,$up^*(s)$ 定义状态 $s$ 的所有祖先,且满足:

(a) $s \in up^*(s)$;

(b) $\forall s_1 \in S \backslash \{root\}: s_1 \in up^*(s_2) \Rightarrow up(s_1) \in up^*(s_2)$.

对 $X \subseteq S$,$up^*(X)=\bigcup_{s \in X} up^*(s)$.

为了精确地刻画状态间的关系,定义状态集的极小公共祖先 LCA(least common ancestor)和严格极小公共祖先 LCOA(least common OR ancestor).

**定义 2.** 非空集合 $X \subseteq S$ 的极小公共祖先表示为 $LCA(X)$,其中 $LCA(X)=x$ 当且仅当:

(1) $X \subseteq \rho^*(x)$;

(2) $\forall s \in S: X \subseteq \rho^*(s) \Rightarrow x \in \rho^*(s)$.

**定义 3.** $X \subseteq S$ 的严格极小公共 $OR$ 祖先表示为 $LCOA(X)$,其中 $LCOA(X)=x$ 当且仅当:

(1) $X \subseteq \rho^1(x)$;

(2) $type(x)=OR$;

(3) $\forall s \in S: type(s)=OR$,则 $X \subseteq \rho^*(s) \rightarrow x \in \rho^*(s)$.

如果状态 $s_1$ 和 $s_2$ 满足:$s_1=s_2$ 或者 $type(LCA(s_1,s_2))=AND$,则称 $s_1$ 和 $s_2$ 是正交状态,记为 $s_1 \perp s_2$. 如果 $X \subseteq S$ 且 $\forall s_1,s_2 \in X: s_1 \perp s_2$,则称 $X$ 是正交状态集. 如果 $s_1 \in \rho^*(s_2)$,则称 $s_1$ 和 $s_2$ 存在偏序关系,记为 $s_1 \leqslant s_2$,并将 $s_1 \leqslant s_2$ 且 $s_1 \neq s_2$ 记为 $s_1 < s_2$.

### 1.2 迁 移

UML statechart 图的迁移包括如下元素:源状态、目标状态、激发事件、布尔条件和动作. 下面记激发事件、布尔条件和动作的相应集合分别为 $EE$,$CE$ 和 $AE$. 若 $e \in EE$,$c \in CE$,$a \in AE$,称 $L=e[c]/a$ 为一个迁移标记,所有迁移标记的集合记为 $Labs$. 布尔条件包括常用的布尔表达式和 $in(s)$,即当对象处于状态 $s$ 时,$in(s)$ 为真. 激发事件通常采用如下格式:$e|op()|\varepsilon|\neg e|e_1 \vee e_2|e_1 \wedge \neg e_2$,其中 $e_1$,$e_2$,$e$ 是基本事件名,$op() \in OP_c$,$\varepsilon$ 是空事件.

动作表达式 $AE$ 的语法 $G_a=(\{N_a,S_a\},T_a,P_a,S_a)$,$N_a$,$S_a$ 是非终结符,$S_a$ 是开始符号,$T_a$ 包含所有的终结符,产生式规则 $P_a$ 的 BNF 如下:

(1) $S_a ::= \varepsilon|N_a$

(2) $N_a ::= \hat{}e|\hat{}op()|\hat{}link.e|\hat{}link.op()|exp|\varepsilon|N_a;N_a|N_a||N_a$

$\wedge$ 是动作标记,$e$ 和 $op()$ 的定义同前,$\hat{}e$ 表示向自己发送事件 $e$,$link \in OBJ$ 是接受对象的名

© 中国科学院软件研究所 http://www.jos.org.cn

字,ε 表示空动作,*exp* 是基本赋值语句,$N_a$;$N_a$ 表示动作的顺序执行,$N_a \parallel N_a$ 表示并发动作.

该语法产生语言 $L(G_a)$,以表示动作集 $AE$.

由于迁移标记的各项都是可选的,所以 UML1.1 支持无激发事件的迁移,又称这种迁移为完成迁移(completion transition),它包含称为"完成事件"的隐含激发事件.当执行完所有迁移和当前活跃状态的进入动作以后,产生完成事件.与其他事件相比,完成事件具有更高的优先级,为此,假定每个完成迁移均标有完成事件.在定义语义时,设事件集包含完成事件集 $Ec$.对不能激活当前迁移的事件,可能会保留到下一状态,该事件称为延迟事件(deferred event).虽然 UML1.1 支持延迟事件,但是,根据状态覆盖和迁移覆盖需求[10],本文不考虑延迟事件.

**定义 4.** $\delta \subseteq 2^S \times Labs \times 2^S$ 是迁移关系,对 $(X,L,Y) \in \delta$,存在 $X,Y \subseteq S, X \neq \varnothing, Y \neq \varnothing, L \in Labs$.

下文中,假定用到如下函数:*source*,*target*,*label*,*trigger*,*guard* 和 *action*.对迁移 $t=(X,L,Y)$,设 $L=e[c]/a$,有 $source(t)=X$,$label(t)=L$,$target(t)=Y$,$trigger(t)=e$,$guard(t)=c$,$action(t)=a$.如果 $X$ 和 $Y$ 都是单元素集合,则将 $\{s\}$ 简记为 $s$.同时,用 $La$ 表示迁移标记只有动作的标记集,$Lc$ 表示迁移标记只有布尔条件的标记集,其中 $La \subseteq Labs$ 且 $Lc \subseteq Labs$.

迁移 $t=(X,L,Y)$ 的极小公共祖先 $LCA(t)=LCOA(X \cup \omega(Y))$.如果迁移 $t_1$ 和 $t_2$ 满足 $LCA(t_1) \perp LCA(t_2)$,则称 $t_1$ 和 $t_2$ 是结构化一致的,记为 $(t_1 \parallel t_2)$.两两结构化一致的迁移构成的集合称为结构化迁移集.给定迁移集 $T=\{t_1,\ldots,t_n\}$,$\forall\ 1 \leqslant i \leqslant n, t_i=(X_i,e_i[c_i]/a_i,Y_i)$,则 $a=a_1;\ldots;a_n$ 称为 $T$ 引发的动作,简称 $T$ 的动作.

简单迁移是指源状态和目标状态都是基本状态的迁移,复合(compound)迁移是指由伪状态 join,fork 或 branch 连接的简单迁移簇,它们存在多个源状态和多个目标状态.join 把从多个正交状态出发的迁移段集连接到 join 伪状态;fork 将迁移分割为多个到达不同正交状态的迁移段;branch 将单个迁移分割成多个条件迁移分支,它必定有标记为"else"的分支.只有当其他所有分支均为假时,"else"分支才是使能的[1].为定义方便,假定 join,fork,branch 各自所连接的简单迁移段构成集合 $T \subseteq \delta$.

**定义 5.** 设 $T=\{t_1,\ldots,t_n\}(n \geqslant 1)$ 是非空迁移集,$X \subseteq S$ 是非空状态集.

· 若 $X$ 是正交集且 $\#X=n \geqslant 2$,则当 $L_{e1},\ldots,L_{en} \in La, L \in Labs, s \in S$ 时,$join(X,L_{e1},\ldots,L_{en},L)=s$ 有定义,$\forall\ i \in \{1,\ldots,n\}$,$\exists\ s' \in X$ 使 $t_i=(s',L_{ei},join) \in T$ 且 $\forall\ i,j \in \{1,\ldots,n\}, i \neq j, t_i \perp t_j$,$t=(join,L,s) \in T$.

· 若 $X$ 是正交集且 $\#X=n \geqslant 2$,则当 $L_{e1},\ldots,L_{en} \in La, s \in S$ 和 $L \in Labs$ 时,$fork(s,L_{e1},\ldots,L_{en},L)=X$ 有定义,且 $\forall\ i \in \{1,\ldots,n\}$,$\exists\ s' \in X$ 使 $t_i=(fork,L_{ei},s') \in T$ 且 $\forall\ i,j \in \{1,\ldots,n\}, i \neq j, t_i \perp t_j, t=(s,L,fork) \in T$.

· 若 $X$ 是正交集且 $\#X=n \geqslant 2$,,则当 $L_{e1},\ldots,L_{en} \in Lc, s \in S$ 和 $L \in Labs$ 时,$branch(s,L_{e1},\ldots,L_{en},L)=X$ 有定义,且 $\forall\ i \in \{1,\ldots,n\}$,$\exists\ s' \in X$ 使 $t_i=(branch,L_{ei},s') \in T$,$t=(s,L,branch) \in T$,有且仅有一个关键字 $else \in \{L_{e1},\ldots,L_{en}\}$.

UML Statechart 图的连接子分为 AND-连接子和 OR-连接子,其中 join 和 fork 是 AND-连接子,branch 是 OR-连接子.连接到 AND-连接子的迁移段属于同一复合迁移 CT,连接到 OR-连接子的迁移段属于不同 CT.设 join,fork,branch 各自所连接的迁移段构成集合 $T=\{t_1,\ldots,t_n\}(n \geqslant 2)$,对应的复合迁移是 CT.由于迁移只有一个激发事件,如果 CT 是由 join 构成的,$T$ 的前 $n-1$ 个元素指向 join,且激发事件为空.设迁移段集合 $T=\{t_1,\ldots,t_n\}(n \geqslant 2)$,由 join,fork 或 branch 连接

© 中国科学院软件研究所 http://www.jos.org.cn

的迁移满足:

(1) 如果连接子是 branch,必定存在 $n-1$ 个迁移.设每个迁移表示为 $CT=(CX,CL,CY)$,其中$\exists\ 2\leqslant i\leqslant n, t_i\in T, CX=source(t_i), event(CT)=event(t_1), guard(CT)=guard(t_1)\wedge guard(t_i)$, $action(CT)=action(t_1);action(t_i), CY-target(t_i)$.

(2) 如果连接子是 join,一定存在迁移 $CT=(CX,CL,XY)$,其中$\forall\ 1\leqslant i,j\leqslant n-1, t_i\perp t_j, CY=target(t_n), event(CT)=event(t_n), CX=\bigcup_{1\leqslant i\leqslant n-1} source(t_i), guard(CT)=guard(t_n), action(CT)=(action(t_1)\parallel\ldots\parallel action(t_{n-1}));action(t_n)$.

(3) 如果连接子是 fork,一定存在迁移 $CT=(CX,CL,CY)$,其中 $CX=source(t_1)$,$\forall\ 2\leqslant i,j\leqslant n, i\neq j, t_i\perp t_j, event(CT)=event(t_1), guard(CT)=guard(t_1), CY=\bigcup_{2\leqslant i\leqslant n} target(t_i), action(CT)=action(t_1);(action(t_2)\parallel\ldots\parallel action(t_n))$.

那么,称 CT 是由 join, fork 或 branch 构成的复合迁移.

所以,UML Statechart 图允许简单迁移和复合迁移.每个迁移的"辖域"就是它的极小公共祖先.图 1 的从 $Q_2$ 到 $P_2$ 迁移的辖域是状态 $SUD$,$S_2$ 到 $S_1$ 的迁移的辖域是 $R_1$,$T_2$ 到 $T_1$ 迁移的辖域是 $R_2$.

UML Statecharts 是一个八元组$(S,init,SH,Labs,\delta,EE,CE,AE)$,其中 $S,init,SH,Labs,\delta\subseteq 2^S\times Labs\times 2^S$,$EE,CE$ 和 $AE$ 分别是非空有限状态集、初始状态、层次结构、标记集、迁移集、事件集、条件集和动作集.

## 2 操作语义

本部分主要根据 UML1.1 规范文档,在规定新的优先级规则和事件处理机制的基础上定义 UML Statechart 图的形式化操作语义.

UML1.1 语义文档假定 UML Statechart 图所描述的对象状态机的执行语义由虚拟机定义.虚拟机的主要部件包括:事件队列,存放产生的事件实例序列直到被分派出去;事件派遣机制,从事件队列选择并分派事件实例;事件处理器,处理被派遣事件.事件处理器的主要任务是根据所获取的事件队列头元素,寻找相应的使能迁移集,这些同时执行的迁移构成一个 STEP.当 STEP 内的所有迁移均被执行完,STEP 终止,才考虑事件队列内的下一个元素.而每个对象实际执行的 RTC-STEP(run-to-complete STEP)可能包括多个 STEP.因为当状态机到达的目标状态可能涉及"完成迁移"时,必须在执行完该完成迁移之后才可能考虑外部事件,完成一个 RTC-STEP.

为了准确地刻画虚拟机的事件处理机制,设 $EE^*$ 是 $EE$ 所有可能事件列表的集合,事件队列为基于 $EE$ 的列表.它的基本操作包括取队列头元素 Dequeue、向队列尾插序列 Enqueue,队列是否为空 Empty 等.事件队列初始为空($\langle\rangle$).

设 $q=\langle e_1,\ldots,e_n\rangle(n\geqslant 0)$ 和 $q'=\langle e_1',\ldots,e_k'\rangle(k\geqslant 0)$ 为 $EE^*$ 的任意对象,$BOOL=\{\text{true},\text{false}\}$.

- $Dequeue: EE^*\to EE\times EE^*$ 满足 $Dequeue=\begin{cases}\langle e_1,\langle e_2,\ldots,e_n\rangle\rangle & n>0\\ \text{无定义} & n\leqslant 0\end{cases}$.
- $Enqueue: EE^*\times EE^*\to EE^*$ 满足 $Enqueue(q,q')=\langle e_1,\ldots,e_n,\langle e_1',\ldots,e_k'\rangle\rangle$.
- $Empty: EE^*\to BOOL$ 满足 $Empty(q)=\begin{cases}\text{true} & n=0\\ \text{false} & n>0\end{cases}$.
- 连接运算符 $\wedge: EE\times EE^*\to EE^*$ 满足 $e\wedge q=\langle e,e_1,\ldots,e_n\rangle e\in EE$.

© 中国科学院软件研究所 http://www.jos.org.cn

UML 对象在创建之初,初始状态是活跃状态.所有活跃状态构成当前格局,表示整个 UML Statechart 图的全局状态.给定根状态 $root \in S$,$root$ 的格局 $Conf \subseteq S$ 满足如下规则:① $root$ 属于 $Conf$;② 若 $Conf$ 包含 $OR$-状态 $A$,则 $Conf$ 必定包含且仅包含 $A$ 的一个直接子状态;③ 若 $Conf$ 包含 $AND$-状态 $A$,则 $\rho(A) \subseteq Conf$;④ $Conf$ 是满足上述规则的有限状态集合中的最小集合.图 1 的 $\{SUD, p_3, Q_2, S_1, T_2\}$ 是一个格局,而 $\{SUD, p_3, p_2, Q_2, S_1, T_1\}$ 却不是格局.假设所有格局的集合是 $Conf_{all}$.

设 $K \subseteq S$ 的最小超集表示为 $\Psi(K)$,其中 $K \subseteq \Psi(K)$,且满足:

(1) $\forall\, d \in \Psi(K): type(d) = AND \Rightarrow \rho(d) \subseteq \Psi(K)$;

(2) $\forall\, d \in \Psi(K): type(d) = OR \Rightarrow \theta(d) \in \Psi(K) \wedge (\rho(d) \cap K = \varnothing)$.

每个迁移可能导致对象离开当前活跃状态,进入其他状态,这两个状态分别称为该迁移的主源(main source)和主目标(main target),分别由函数 $MSource: \delta \to S$ 和 $MTar: \delta \to S$ 决定.迁移 $CT = (X, L, Y)$ 的主源 $MSource(CT) \in S$ 是 $LCA(CT)$ 的子状态,它包含所有源状态,且满足 $MSource(CT) \in \rho(LCA(CT)) \wedge LCA(X) \leqslant MSource(CT)$;主目标是 $LCA(CT)$ 的子状态,它包含所有目标状态,即 $MTar(CT) \in \rho(LCA(CT)) \wedge LCA(\omega(Y)) \leqslant MTar(CT)$.一旦迁移被执行,将离开主源,执行与迁移相关的动作序列,进入主目标.迁移进入和离开的状态集,分别由函数 $exit$ 和 $enter$ 确定:

- $exit: \delta \times Conf_{all} \to 2^S$ 满足 $\forall\, t \in \delta, Conf \in Conf_{all}$.
  $exit(t, Conf) = \rho^*(MSource(t)) \cap Conf$.
- $enter: \delta \times Conf_{all} \to 2^S$ 满足 $\forall\, t \in \delta, t = (X, L, Y), Conf \in Conf_{all}$.
  $enter(t, Conf) = \Psi(\rho^*(MTar(t)) \cap up^*(Y))$.

**定理 1.** 设 $Conf \in Conf_{all}, t = (X, L, Y) \in \delta$,则 $X \subseteq Conf \Rightarrow (Conf \setminus exit(t, Conf) \cup enter(t, Conf)) \in Conf_{all}$.

在图 1 中,从 $Q_2$ 到 $P_2$ 的层间迁移 $t' = (Q_2, C/\neg G, P_2)$,则 $LCA(t') = SUD$,主源是 $Q_2$,主目标是 $P_2$.设当前格局 $Conf = \{SUD, p_3, Q_2, S_1, T_1\}$,$t-D$,当前激发事件为 $D$,则对象离开状态 $T_1$ 到达 $T_2$,故 $exit(t, Conf) = \{T_1\}$,$enter(t, Conf) = \{T_2\}$,新格局为 $\{SUD, p_3, Q_2, S_1, T_2\}$.

如果一个迁移的源状态属于当前格局,当前激发事件与该迁移的激发事件一致,且迁移的布尔条件为真,则称该迁移是使能的.因此定义状况 $Status = (Conf, E, C) \in Cnof_{all} \times 2^{EE} \times 2^{CE}$,其中 $Conf \in Conf_{all}$,$E \subseteq EE$ 和 $C \subseteq CE$ 分别是对象所处格局、当前激发事件的集合、条件变量值的集合,且 $\#E = 1$.设 $Status_{all}$ 是对象所有状况的集合.初始状况 $(Conf, E, C) \in Status_{all}$ 满足 $Conf = \{root, \omega(init)\} \wedge E = \Phi \wedge C = \varnothing$.同时引入事件计值 $EvalEV_{(id, Status)}$、条件计值 $EvalCV_{(id, Status)}$ 和 $EvalAV_{(id, Status)}$,分别计算迁移标记各个选项的值.

$EvalEV: EE \times 2^{EE} \to BOOL$ 满足

$$EvalEV(x, E) = \begin{cases} \text{true} & x \in E \\ \text{false} & x \notin E \\ \text{true} & x = \epsilon \\ \neg EvalEV(e, E) & x = \neg e, e \in E \\ EvalEV(e_1, E) \vee EvalEV(e_2, E) & x = e_1 \vee e_2, e_1, e_2 \in EE \end{cases}$$

$E \subseteq EE$.

© 中国科学院软件研究所 http://www.jos.org.cn

$EvalCV:CE\times 2^{CE}\rightarrow BOOL$ 满足

$$EvalCV(x,C)=\begin{cases}\text{true} & x=\varepsilon\\ \text{true} & x=in(s),s\in Conf\\ \text{false} & x=in(s),s\notin Conf\\ x & x\in C\\ \neg EvalCV(c,C) & x=\neg c,c\in C\\ EvalCV(c_1,C)\vee EvalCV(c_2,C) & x=c_1\vee c_2,c_1,c_2\in CE\\ EvalCV(c_1,C)\wedge EvalCV(c_2,C) & x=c_1\wedge c_2,c_1,c_2\in CE\end{cases}$$

$C\subseteq CE,Conf\in Conf_{all}$;

$EvalAc:AE\times 2^{CE}\rightarrow 2^{EE^*\times 2^{CE}}$ 满足:

(1) $EvalAc(\varepsilon,C)=\{(\langle\rangle,C)\}$;

(2) $EvalAc(^{\wedge}e,C)=\{(\langle e\rangle,C)\}$;

(3) $EvalAc(^{\wedge}0.e,C)=EvalEv_{(id',status')}(^{\wedge}e,C)$,$id'$ 是 $o$ 的标识符,$e$ 是 $o$ 的事件或者操作,$status'$ 是对象 $id'$ 所处的状况;

(4) $EvalAc(c_1=c_2,C)=EvalAc(\langle\rangle,C\cup\{c_1\}),c_1,c_2\subset CE$;

(5) $EvalAc(a_1;A,C)=\{e_1{}^{\wedge}\pi_1(EvalAc(A,C_1)),C_1\cup\pi_2(EvalAc(A,C_1))\}(e_1,C_1)\in EvalAc(a,C)$;

(6) $EvalAc(a_1\parallel A,C)=EvalAc(a_1;A,C)\cup EvalAc(A;a_1,C)$,其中 $EvalAc(A;a_1,C)$的定义类似于(5).

$\pi_1,\pi_2$ 是投影函数,$\pi_1:EE^*\times 2^{CE}\rightarrow EE^*$ 满足 $\pi_1(E,C)=E,E\in EE^*,C\subseteq CE$,$\pi_2:EE^*\times 2^{CE}\rightarrow 2^{CE}$满足$\pi_2(E,C)=C,E,C$同上.将它们延拓为 $\pi_1{}^*:2^{EE^*\times 2^{CE}}\rightarrow EE^*$ 和 $\pi_2{}^*:2^{EE^*\times 2^{CE}}\rightarrow 2^{CE}$,其中设 $Y\in 2^{EE^*\times 2^{CE}}$,$\pi_1{}^*(Y)=\{E\mid\{(E,C)\}\in Y,E\in EE^*,C\subset CE\}$,$\pi_2{}^*(Y)=\{C\mid\{(E,C)\}\in Y,E\in EE^*,C\subseteq CE\}$.若无特殊说明,则投影函数及其延拓为同一函数.

**定义 6.** 迁移使能函数 $Enabled:\delta\times Status_{all}\rightarrow BOOL$ 满足$\forall t=(X,e[c]/a,Y)\in\delta$,$Status=(Conf,E,C)\in Status_{all}$,$Enabled(t,Status)=$true 当且仅当 $X\subseteq Conf$,$EvalEV(e,E)$和 $EvalCV(c,C)$为 true.

一旦获取事件队列的头元素,有可能使多个迁移变为使能.如果没有使能迁移,事件将被丢弃,获取事件队列的下一个元素.如果存在多个使能迁移,且某些迁移离开同一个状态时会发生冲突,为此要选择最大无冲突迁移集.在格局 $Conf\in Conf_{all}$的情况下,迁移之间的冲突由 $Conflict_{Conf}:\delta\times\delta\rightarrow BOOL$ 决定,当$\forall t_1,t_2\in\delta,t_1\neq t_2\wedge(exit(t_1,Conf)\cap exit(t_2,Conf)\neq\varnothing)$时,$Conflict_{Conf}(t_1,t_2)=$true,称迁移 $t_1$ 和 $t_2$ 关于格局 $Conf$ 相互冲突.如果迁移源状态之间存在父子关系,规定迁移的优先级将解决部分冲突.迁移 $t_1$ 的优先级高于 $t_2$(记为 $t_1>t_2$)当且仅当 $source(t_1)\subset\rho^{-}(source(t_2))$.图 2 的状态 $A$ 和 $G$ 之间不存在父子关系,因此从状态 $A$ 到 $B$ 的迁移和从 $G$ 到 $D$ 的迁移无法确定优先级,只能任选一个以避免冲突.

设当前 $Status=(Conf,E,C)$下所有使能迁移的集合为 $ET$,采用下述方法确定最大无冲突迁移集 $FireTr$:

(1) 计算 $Status$ 状况下最大使能迁移集:$ET=\{t\in\delta\mid Enabled(t,Status)=\text{true}\}$;

(2) 从 $ET$ 内移去与优先级较高的迁移相互冲突的迁移:

© 中国科学院软件研究所 http://www.jos.org.cn

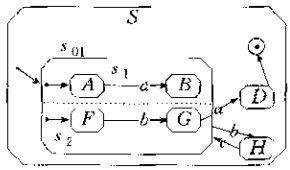


Fig. 2 A UML Statechart diagram including AND-state
图2 包含AND-状态的UML Statechart图

$$ET=ET\backslash\{t_1\in ET\mid \exists\ t\in ET: t>t_1\wedge Conflict_{Conf}(t_1,t)\};$$

(3) 从 $ET$ 内获取最多无冲突迁移集 $FireTr$,$FireTr$ 必须满足如下条件:

(a) $\forall\ t_1,t_2\in FireTr:(t_1=t_2\vee\neg Conflict_{Conf}(t_1,t_2))$;

(b) $\forall\ t_c\in ET:t_c\in FireTr\vee(\exists\ t\in FireTr:(t_c\neq t\wedge Conflict_{Conf}(t_c,t)))$;

UML1.1 规定每个状态可以包括进入动作、离开动作、内部活动和内部迁移等动作状态.在执行 STEP 的过程中,执行状态动作和迁移的动作序列所产生的事件序列都存放在事件队列 $Event$,完成事件存放在队列 $E_{com}$,其中 $Event\in EE^*$ 且 $E_{com}\in EE^*$.

**定义 7.** 步骤 $STEP\subseteq Status_{all}\times 2^{\hat{}}\times Status_{all}$ 是状况之间的一种关系,$((Conf,E,C),Tr,(Conf',E',C'))\in STEP$ 当且仅当下述 3 个条件同时成立:

(1) $Tr\subseteq FireTr$;

(2) 如果 $Empty(E_{com})=$true,则 $E_0=\{e\in EE\mid(e,Event)=Deque(Event)\}$,否则 $E_0=\{e\in EE\mid(e,E_{com})=Dequeue(E_{com})\}$;

(3) 下述任意条件之一成立:

(a) 如果没有使能迁移,即 $\#Tr=0$,则步骤为空,$(Conf',E',C')=(Conf,\Phi,C)$;

(b) 否则,设 $\#Tr=n,Tr=\{t_1,\ldots,t_n\},\forall\ 1\leqslant i,j\leqslant n,i\neq j,t_i\parallel t_j$,

$$Conf'=\bigcup_{1\leqslant i\leqslant n}(Conf\backslash exit(t_i,Conf))\cup enter(t_i,Conf),\forall\ 1\leqslant i\leqslant n,$$

$$EvalAc(action(t_i),C)=(E_i,C_i)\wedge E_i\in EE^*\Rightarrow Event=Enqueue(Event,E_i),$$

$$C'=\bigcup_{1\leqslant i\leqslant n}C_i,E'=\varnothing,E_{com}=Enqueue(E_{com},\langle C\varepsilon\rangle),C\varepsilon\in E_{com}.$$

在图 1 中,设当前状况 $Status=(Conf,E,C)$,其中 $Conf=\{SUD,P_3,Q_2,S_2,T_2\},E=\{C\},C=\varnothing$,则从 $S_2$ 到 $S_1$ 的迁移 $t_1$ 和 $Q_2$ 到 $P_2$ 的迁移 $t_2$ 均是使能的.由于 $t_1$ 的优先级高于 $t_2$,则最大无冲突迁移集 $FireTr=\{t_1\}$.执行完 $t_1$ 到达新状况 $Status'=(Conf',E',C')$,其中 $Conf'=\{SUD,P_3,Q_2,S_1,T_2\},E'=\varnothing,C'=\varnothing$.从状况 $Status$ 到 $Status'$ 的转换构成一个 $STEP$.当对象处于 $Status'$ 时,$E_{com}$ 包含完成事件.如果 $Status'$ 满足 $\neg\exists\ t\in TR:Enabled(t,Status')$,则状况 $Status'$ 是稳定的,此时对象处于稳定状态.

由于 UML 假定每个对象状态机只有在完成一个 $RTC\text{-}STEP$ 之后才能处理其他对象发送的事件,为此引入事件队列 $E_{env}\in EE^*$ 存放环境和其他对象发送来的事件.

**定义 8.** $RTC\text{-}STEP\subseteq Status_{all}\times 2^{\delta}\times Status_{all}$ 是状况之间的一种关系,$((S,E,C),T,(S',E',C'))\in RTC\text{-}STEP$ 当且仅当:

(1) 若 $Empty(E_{env})=true$,则 $T-\varnothing$,$RTC\text{-}STEP$ 为空,$(S,E,C)=(S',E',C')$.

(2) 若 $Empty(E_{env})=false$,则 $(e,E_{env})=Dequeue(E_{env})\wedge e\in EE$ 且 $Event-Enqueue(e,Event)$.

(3) 否则存在状况序列 $(Status_0,\ldots,Status_{m+1})$ 和 $T=(tr_0,\ldots,tr_m)$,其中 $m\geqslant 0$ 且 $\forall\ 0\leqslant i\leqslant$

© 中国科学院软件研究所 http://www.jos.org.cn

$m:tr_i \subseteq \delta \wedge Status_i \in Status_{all}$, $Status_0=(S,E,C)$, $Status_{m+1}=(S',E',C') \in Status_{all}$, 若 $m=0$ 且 $\#tr_0=0$, 则 $STEP$ 为空, $RTC\text{-}STEP$ 也为空, $(S,E,C)=(S',E',C')$, 若 $m>0$, 则 $\forall\ 0 \leqslant i \leqslant m: \#tr_i \geqslant 1 \wedge (Status_i, tr_i, Status_{i+1}) \in STEP$ 且 $Status_{m+1}$ 是稳定的.

**定义 9**. UML Statechart 图描述的对象状态机的操作语义是 Kripke 结构, $K=(Status_{all}, \Theta_{Init}, \xrightarrow{RTC\text{-}STEP})$.

- $\Theta_{Init} \in Status_{all}$ 是对象初始状况, $\Theta_{Init}=(\{root, \omega(init)\}, \varnothing, \varnothing)$, $Event$ 和 $E_{com}$ 为空.
- $\xrightarrow{RTC\text{-}STEP}$ 是 $K$ 的迁移关系.
- 当 $status=(\{root, final\}, \varnothing, \varnothing)$ 时, 状态机停止执行.

UML Statechart 图的操作语义是一个 Kripke 结构, 它是一组由(迁移)关系连接的状态集, 通常称这种状态为状况, 迁移为 RTC-STEP. 当对象处于某个稳定状况时, 将外部环境事件队列 $E_{env}$ 的头元素插入事件队列 Event, 根据 Event 头元素选择使能迁移, 执行 RTC-STEP 算法, 当对象到达新的稳定状况时, Event 为空. 当无外部事件且根状态不产生任何事件时, UML Statechart 图所描述的对象停止执行. 这种结构完整地刻画了状态机状态行为的动态变化过程. 迁移优先级的提出不仅解决了迁移之间的冲突, 而且保证了执行结果的一致性.

## 3 总 结

本文根据 UML1.1 语义文档定义的 UML Statechart 图操作语义, 可用于面向对象软件类层测试用例的生成. 为了自动地构造类层测试数据, 不仅考虑传统 Statecharts 的状态层次结构、并发和同步通信机制, 而且定义了伪状态、复合迁移、STEP 和 RTC-STEP 的形式语法和语义. 在定义 STEP 和 RTC-STEP 时, 增加内部事件队列 Event、完成事件队列 $E_{com}$ 和外部事件队列 $E_{env}$, 以形象地刻画虚拟机的事件处理机制和完成事件的优先级, 使 UML Statechart 图的语义更简洁直观.

目前, 我们已经在 Rose98 平台上建立了电梯系统[9]、收录机[10]和铁路调度系统[4]的 UML 模型. 采用 W-方法和 Wp-方法和所定义的测试用例生成规则[10]构造整个对象状态机的测试用例集, 每个测试用例是激发事件和布尔条件构成的输入序列. 根据报告判断实现是否与设计一致, 或者发现如下错误: 丢失状态、额外状态、额外迁移、丢失迁移、不正确动作等. 同时依据语义和模拟技术, 形象地演示测试全过程, 以便用户了解整个测试过程和测试结果.

本文的操作语义只考虑对象的控制结构和简单数据类型, 假定迁移执行时间为零, 没有考虑对象的实时特性. 而实时问题和复杂数据类型将是以后研究的重点. UML1.3 只对状态类型进行重新划分, 增加了同步状态, 没有其他改进. 上述语义完全可以适用于 UML1.3.

**References:**


[1] Unified Modeling Language Document Set. Version1.1, Rational Software Corporation, 1997.

[2] Ilogix Rhapsody, iLogix Corporation. http://www.ilogix.com/product.html.

[3] Harel, D. Statecharts: a visual formalism for complex system. Science of Computer Programming, 1987,8(3):231~274.

[4] Harel, D., Pnueli, A., Schmidt, J.P., *et al*. On the formal semantics of statecharts. In: Jones, D.M., ed. Proceedings of the 2nd IEEE Symposium on Logic in Computer Science. Ithaca, New York, IEEE Press, 1987. 54~64.

[5] Harel, D. Amnon Naamad. The STATEMATE semantics of statecharts. ACM Transactions on Software Engineering and Methodology, 1996,5(4):293~333.

[6] Mikk, E., Lakhnech, Y., Petersohn, C. On formal semantics of statecharts as supported by STATEMATE. In: The 2nd BCS-FACS Northern Formal Methods Workshop. Spring-Verlag, 1997.


© 中国科学院软件研究所 http://www.jos.org.cn

[7] Petersohn, C. Data and control flow diagrams, statecharts and Z: their formalization, integration and real-time extension [Ph. D. Thesis]. Germany, 1997.

[8] Latella, D., Majzik, I., Massink, M. Towards a formal operational semantics of UML statechart diagrams. In: Ciancarini, P., Gorrieri, R., eds. IFIP TC6/WG6.1 3rd International Conference on Formal Methods for Open Object-Oriented Distributed Systems. Florence, Italy: Kluwer academic Publisher, 1999. 331~347.

[9] Harel, D., Gery, E. Executable object modeling with statecharts. IEEE Computer, 1997,30(7):31~42.

[10] Li, Liu-ying, Qi, Zhi-chang. Test Selection from UML Statechart. TOOLS31, Nanjing, 1999. 273~279.

## An Operational Semantics for UML Statechart Diagrams*

LI Liu-ying[1], WANG Ji[1,2], QI Zhi-chang[1]

[1](*School of Computer Science, National University of Defence Technology, Changsha* 410073, *China*);

[2](*State Key Laboratory for Software Engineering, Wuhan University, Wuhan* 430072, *China*)

E-mail: lyli@nudt.edu.cn

http://www.nudt.edu.cn

**Abstract**: As a standard object-oriented modeling language, UKL (unified modeling language) is lack of formal dynamic semantics. In this paper, an operational semantics for UML statechart diagrams is proposed according to UML1.1 specification documents. The formalization is able to deal with most of the features of UML statechart diagrams, which sets the basis for code generation, simulation and test cases generation. It has been used to generate test cases and to simulate the test process on platform of Rose98.

**Key words**: UML (unified modeling language); operational semantics; test

* Received November 22, 1999; accepted July 10, 2000

Supported by the National Natural Science Foundation of China under Grant No. 69973051; the National High Technology Development 863 Program of China under Grant No. 863-306-ZT06-04-1; the Foundation of State Key Laboratory for software Engineering in Wuhan University of China; the Huo Ying Dong Yongerteather Foundation of China

© 中国科学院软件研究所 http://www.jos.org.cn