

文章编号:1001-9081(2005)02-0359-03

## UML 顺序图的结构化操作语义研究

黄 隲, 于洪敏, 陈致明

(解放军军械工程学院, 河北 石家庄 050003)

(huanglong789@sina.com)

**摘 要:** UML 顺序图侧重于展示对象之间的消息交互过程, 但其动态语义缺乏形式化的描述, 不利于对顺序图模型的准确理解和基于该模型的测试用例生成。为此, 依据 UML1.5 规范, 采用 BNF 定义顺序图的形式化语法, 提出了活动点的概念; 在此基础上, 讨论并给出了单个对象执行消息动作的结构化操作语义以及顺序图模型的整体结构化操作语义, 为模型检验和基于顺序图的测试用例生成提供了前提。

**关键词:** UML; 顺序图; 结构化操作语义

**中图分类号:** TP311 **文献标识码:** A

## Research on structured operational semantics of UML sequence diagram

HUANG Long, YU Hong-min, CHEN Zhi-ming

(Ordnance Engineering College, Shijiazhuang Hebei 050003, China)

**Abstract:** UML Sequence Diagram emphasizes particularly the process of message interaction of objects. But it lacks formal description of dynamic semantics, therefore goes against comprehension of the diagram and test case generation based on the diagram. Based on UML specification (version 1.5), this paper defined formal syntax of the diagram firstly and brought forward concept of Action-Point afterwards; thus discuss and put forward the structured operational semantics of message action of single object and the whole sequence diagram, provided the precondition of model checking and the test cases generation which based on sequence diagram.

**Key words:** UML; sequence diagram; structured operational semantics

### 0 引言

面向对象软件的运行机制是通过对象以及对象之间的动态交互实现的。正如 Ivar Jacobson 所说:“只有在所有的用例为所有事件进程建立了交互模型之后, 才可以确定已经发现系统所需的每个对象所扮演的角色, 以及它们的责任<sup>[1]</sup>。”因此, 交互建模在面向对象软件开发中越来越重要。UML 顺序图直观地展现了对象之间的消息动态交互情况, 不论对交互建模还是对交互测试都是很重要的依据和信息来源。而在 UML 规范中, 模型的语法是通过元模型以 UML 的类图的方式定义的, 静态语义采用 OCL(对象约束语言)进行描述, 模型的动态语义则是直接由英文的自然语言来表达的。采用这种方法描述的动态语义, 存在着不完全、不一致、模糊性等缺陷。现有的形式化工作主要是用于状态图, 针对顺序图进行语义分析的很少<sup>[2]</sup>。为此, 我们以 UML 规范(v1.5)为基础, 采用 BNF 范式定义其语法, 给出了它的结构化操作语义。这不仅准确地描述了系统的动态交互过程, 有利于建模人员之间的交流, 而且也能为基于顺序图的测试用例生成提供了形式规范。

### 1 UML 顺序图的形式化语法描述

定义 1 一个 UML 顺序图可以表示为一个五元组  $SD = \langle Obj, Msg, Evn, Loc, Time \rangle$ , 其中:

$Obj = \{ Object \mid Object \text{ 是顺序图中的对象} \};$

$Msg = \{ Message \mid Message \text{ 是顺序图中的消息} \};$

$Obj \cup Msg \neq \emptyset;$

$Obj \cap Msg = \emptyset;$

$Msg \subseteq Obj \times Obj;$

$Evn = \{ Event \mid Event \text{ 是一条消息所对应的事件} \};$

$Loc = \{ Location \mid Location \text{ 是一个事件所对应的活动点} \};$

$Time = \{ Time\_point \mid Time\_point \text{ 是一个事件所对应的时间点} \};$

定义函数映射  $F: (ObjectSet, MS) \rightarrow OperationSet;$

MS 为消息信号:  $MS = (Lifetime, Event);$

一个消息信号  $ms \in MS$  表示在对象激活期, 由于事件 Event 产生消息, 消息的结果是触发操作集 OperationSet 中的某个操作。

考虑顺序图中的交互对象, 可以分别定义对象的名称、对

收稿日期: 2004-07-21; 修订日期: 2004-10-09

作者简介: 黄隲(1975-), 男, 河北南和人, 博士研究生, 主要研究方向: 软件工程、装备指挥自动化; 于洪敏(1963-), 男, 山东栖霞人, 副教授, 博士研究生, 主要研究方向: 装备指挥自动化; 陈致明(1940-), 男, 吉林洮南人, 教授, 博士生导师, 主要研究方向: 计算机软件、指挥自动化。

象所属的类、对象的标识号、对象的激活期以及对象的状态等要素。

```
<Object> ::= [ <Obj_name> <Obj_class> [ <Obj_id> ]
              <Obj_live> <Obj_state>
<Obj_class> ::= cls_name{ cls_attr } + { cls_opr } +
<Obj_id> ::= <Digit1> { <Digit2> } * ;
<Digit1> ::= 1|2|3|4|5|6|7|8|9
<Digit2> ::= 0|1|2|3|4|5|6|7|8|9
<Obj_live> ::= Inactive|Active;
<Obj_state> ::= <Attr_val_list>
<Attr_val_list> ::= ( { cls_attr = 'value' } + )
```

对于顺序图中在对象之间传递的消息,可以定义消息的名称、消息的标识号、消息的类型、消息所引发的动作、消息的发送者与接收者以及消息所带有的参数等组成元素。

```
<Message> ::= Msg_name[ <Msg_id> ]! <Msg_type>
              <Msg_action>
              <Msg_sender> <Msg_receiver> [ <Msg_para_list> ]
              [ <Activator> ] [ <Predecessor> ];
<Msg_id> 的定义同 <Obj_id> 的定义;
<Msg_type> ::= Sync|Async|Ack
<Msg_action> ::= Call_opr|Send_sgl|Broadcast_sgl
               |Create_Obj|Destroy_Obj
<Msg_sender> ::= <Object>
<Msg_receiver> ::= <Object>
<Msg_para_list> ::= ( <Msg_para> { , <Msg_para> } * )
<Msg_para> ::= Integer|String|Float
<Activator> ::= { <Message> } *
<Predecessor> ::= { <Message> } *
```

消息所对应的事件可通过事件标识号和类别、事件发生的前置条件与后置条件来描述。

```
<Event> ::= [ <Event_id> ] <Event_type> pre_cond post_cond
<Event_id> 的定义同 <Obj_id> 的定义;
<Event_type> ::= send|receive
```

在顺序图中,每个事件对应着一个活动点(后面定义)。下面通过一个三元组来定义某个对象上的各个活动点及其所对应的活动。

```
<Location> ::= ' <' <Object> ' , ' <Loc_id> ' , '
              <Loc_action> ' > '
<Loc_id> 的定义同 <Obj_id> 的定义;
<Loc_action> ::= send|receive
```

每一个活动点和一个时间点相关联,根据所发生活动的不同,可将时间点分为发送时间点和接收时间点。

```
<Time_point> ::= T_send|T_receive
```

## 2 活动点的概念

文献[4]中提出了 Location 的概念,所谓 Location 是指对象生命线上可以发送或接收消息的点。该定义只说明了 Location 在对象生命线上的静态位置关系。为了体现对象之间消息交互的动态特性,我们在 Location 的基础上提出了活动点(ActionPoint)的概念。

定义2 对顺序图中的任一对象,其生命线上可以产生发送消息事件或接收消息事件两种行为所对应的点,称为具有相应行为的活动点。对于  $\forall O_i \in Obj$ ,采用标记  $\langle O_i, ap$ ,

$am \rangle$  表示该对象的活动点,且满足:  $dom(ap) = \{0\} \cup N$ ,  $am ::= !|?m.m$  和  $?m$  表示发送消息以及接收消息两种行为。

根据定义,活动点是对象生命线上离散的、有限的和抽象的点,按照其所具有行为的不同,可以划分为发送活动点和接收活动点两类。

## 3 UML 顺序图的结构化操作语义描述

对语义进行形式化描述的方法目前主要有操作语义、指称语义、公理语义和代数语义。操作语义的基本思想是用抽象的方法描述语言中每一成分的执行效果,以免所描述的语义依赖于该语言实现时所用的具体计算机,其突出特点在于具有直观的表述方式。Plotkin 提出的结构化操作语义(Structured Operational Semantics, SOS)促进了操作语义学的新发展。其基本思想是:复合成分的操作语义应该可以归结为它的各个组成部分的操作语义。结构化操作语义由三部分组成<sup>[3]</sup>:一是语法范畴;二是语法规则;三是动态语义,由一组推理规则组成。选用结构化操作语义作为描述顺序图语义的工具具有以下几个优点:一是结构化操作语义侧重于描述系统操作的实现过程,因此非常适合于体现顺序图中对象之间消息的动态交互过程;二是结构化操作语义是语法制导的,这样,在前面定义顺序图形式语法的基础上,有利于保持与静态语义定义的一致性;三是结构化操作语义具有结构化的特征,使得可以在定义单个对象动态语义的基础上,通过复合关系得到整个顺序图的动态语义。

### 3.1 单个对象执行消息动作的结构化操作语义

定义函数  $\theta$  表示属性集到属性值的映射,即  $\theta: Attr \rightarrow Value$ 。则  $\forall O_i \in Obj$ , 其状态集可以表示为  $\delta_i = \{(A_j, V_j) \mid \forall A_j \in Attr, V_j = \theta(A_j), j \in [1, k], k \in N\}$ ,  $k$  为属性个数。相应的,整个顺序图的全局状态为  $Conf = \bigcup_{i=1}^n \delta_i$ ,  $n$  为对象个数。 $\forall O_i \in Obj, \sigma_j \in \delta_i$ , 称  $\langle O_i, \sigma_j \rangle$  为对象  $O_i$  的一个格局。令  $C$  为  $O_i$  的所有格局的集合,定义转换关系  $T \subseteq C \times A \times C$ , 其中  $A = \{!(O_i, m_1, O_s), ?(O_i, m_2, O_i), \varepsilon\}$ 。

1) 当  $O_i$  不执行任何消息收发动作时:

$$\langle O_i, \sigma_j \rangle \xrightarrow{Null} \sigma_j$$

2)  $O_i$  执行一条消息  $m_1$  的发送动作时:

$$\langle O_i, \sigma_j \rangle \xrightarrow{!(O_i, m_1, O_s)} \langle O_i, \sigma'_j \rangle$$

$\sigma'_j$  的确定是通过  $O_i$  自身的 FSM 执行得到的。即  $\sigma_j \xrightarrow{E[CA]} \sigma'_j$ 。由于 SendEvent 事件的发生,导致内部状态发生迁移,并产生相应的动作 Action。

3)  $O_i$  执行一条消息  $m_1$  的接收动作时:

$$\langle O_i, \sigma_j \rangle \xrightarrow{?(O_i, m_1, O_i)} \langle O_i, \sigma'_j \rangle$$

在同步时,直接调用方法执行,异步时,  $O_i$  发送  $m_1$  引发动作 Action,当其满足  $O_i$  使能迁移事件 E 时,触发其 FSM 执行,得到新状态  $\sigma'_j$ 。

4)  $O_i$  顺序执行消息  $m_1$  和  $m_2$  的发送动作时:

$$\frac{\frac{\langle O_i, \sigma_j \rangle \xrightarrow{!(O_i, m_1, O_s)} \langle O_i, \sigma'_j \rangle, \langle O_i, \sigma_j \rangle \xrightarrow{!(O_i, m_1, O_t)} \langle O_i, \sigma''_j \rangle}{\langle O_i, \sigma_j \rangle \xrightarrow{!(O_i, m_1, O_s); !(O_i, m_1, O_t)} \langle O_i, \sigma'_j \rangle}}{\langle O_i, \sigma_j \rangle \xrightarrow{!(O_i, m_1, O_s)} \langle O_i, \sigma'_j \rangle}$$

5)  $O_i$  顺序执行消息  $m_1$  和  $m_2$  的接收动作时的语义描述  
与4)类似,限于篇幅,不再赘述。

义可由4)、5)综合而得。

7)  $O_i$  对消息  $m_1$  和  $m_2$  的选择执行过程:当条件表达式满

6)  $O_i$  顺序执行消息  $m_1$  和  $m_2$ ,或接收  $m_1$  发送  $m_2$  时的语

足时,发送  $m_1$ ,否则发送  $m_2$

$$\frac{\langle b, \sigma_j \rangle \longrightarrow true \quad \langle O_i, \sigma_j \rangle \xrightarrow{!(O_i, m_1, O_s)} \langle O_i, \sigma'_j \rangle}{\langle \text{If } b \text{ then } !(O_i, m_1, O_s) \text{ else } !(O_i, m_2, O_t), \sigma_j \rangle \longrightarrow \langle O_i, \sigma'_j \rangle}$$

8)  $O_i$  循环发送消息  $m_1$  时,可将其表示为  $*[u = 1 \cdots n]m_1$  (或  $+ [u = 1 \cdots n]m_1$ )<sup>[4]</sup>。

a) 对于  $+ [u = 1 \cdots n]m_1$  的情形,有( $b$  为循环入口条件):

$$\frac{\frac{\langle b, \sigma_j \rangle \longrightarrow false}{\langle \text{While } b \text{ do } !(O_i, m_1, O_s), \sigma_j \rangle \longrightarrow \sigma_j}}{\langle b, \sigma_j \rangle \longrightarrow true \quad \langle !(O_i, m_1, O_s), \sigma_j \rangle \longrightarrow \sigma'_j \quad \langle \text{While } b \text{ do } !(O_i, m_1, O_s), \sigma'_j \rangle \longrightarrow \sigma'_j}{\langle \text{While } b \text{ do } !(O_i, m_1, O_s), \sigma_j \rangle \longrightarrow \langle \sigma'_j \rangle}$$

b) 对于  $*[u = 1 \cdots n]m_1$  的情形,有( $b$  为循环出口条件):

$$\frac{\frac{\langle O_i, \sigma_j \rangle \xrightarrow{!(O_i, m_1, O_s)} \langle O_i, \sigma'_j \rangle}{\langle b', \sigma'_j \rangle \longrightarrow true}{\langle \text{Do } !(O_i, m_1, O_s) \text{ While } b', \sigma'_j \rangle \longrightarrow \sigma'_j}}{\langle b', \sigma'_j \rangle \longrightarrow false \quad \langle !(O_i, m_1, O_s), \sigma'_j \rangle \longrightarrow \sigma'_j \quad \langle \text{Do } !(O_i, m_1, O_s) \text{ While } b', \sigma'_j \rangle \longrightarrow \sigma'_j}{\langle \text{Do } !(O_i, m_1, O_s) \text{ While } b', \sigma'_j \rangle \longrightarrow \langle \sigma'_j \rangle}$$

9)  $O_i$  发送消息  $m_1$  创建对象  $O_j$  时:

$$\frac{\frac{\langle O_i, \sigma_j \rangle \xrightarrow{!(O_i, m_1, O_j)} \langle O_i, \sigma'_j \rangle}{\langle O_i, \sigma_j \rangle \xrightarrow{\text{create}(O_i, O_j)} \langle O_i, \sigma'_j \rangle}}{\langle O_j, \varepsilon \rangle \xrightarrow{\text{create}(O_i, O_j)} \langle O_j, \sigma_{exist} \rangle, \varepsilon \text{ 为空状态}, \sigma_{exist} \text{ 表示 } O_j \text{ 处于存在状态。}}$$

则有  $\langle SD, \delta_{\Sigma}^{n-1} \rangle \xrightarrow{m_i} \langle SD, \delta_{\Sigma}^n \rangle$  且满足  $\delta_{\Sigma}^n = (\delta_{\Sigma}^{n-1} / (\sigma_p \cup \sigma_q)) \cup \sigma'_p \cup \sigma'_q$

其中,  $\langle SD, \delta_{\Sigma}^0 \rangle$  称为初试格局,  $\delta_{\Sigma}^0 = \prod_{i=1}^k \sigma_i^{init}, \sigma_i^{init}$  为对象的初始状态。按照上述动作执行过程,可由初始格局递推至  $\langle SD, \delta_{\Sigma}^n \rangle$ ,称为交互模型的终止格局。

表示  $O_j$  处于存在状态。

10) 对象  $O_j$  的撤消操作语义:

$$\begin{aligned} \text{a) 自撤消: } & \langle O_j, \sigma_j \rangle \xrightarrow{\text{stop}} \langle O_j, \varepsilon \rangle \\ \text{b) 消息撤消: 令布尔表达式 } & e: m_1 = \text{Destroy}(O_i, O_j) \\ & \frac{\langle e, \sigma_j \rangle \longrightarrow true \quad \langle O_j, \sigma_j \rangle \xrightarrow{\text{Destroy}(O_i, O_j)} \varepsilon}{\langle O_j, \sigma_j \rangle \xrightarrow{!(O_i, m_1, O_j)} \langle O_j, \varepsilon \rangle} \end{aligned}$$

### 3.2 顺序图的整体结构化操作语义

$\forall O_i \in Obj, \sigma_i$  表示在某时刻  $O_i$  所处的状态,  $\delta_{\Sigma} = \prod_{i=1}^k \sigma_i$ ,  $k = \#(Obj)$ , 称  $\langle SD, \delta_{\Sigma} \rangle$  为顺序图的一个格局。考虑当经过一个交互场景之后,整个顺序图的状态变化情况。

$$\langle SD, \delta_{\Sigma}^0 \rangle \xrightarrow{m_1} \langle SD, \delta_{\Sigma}^1 \rangle \xrightarrow{m_2} \langle SD, \delta_{\Sigma}^2 \rangle \longrightarrow \cdots \xrightarrow{m_n} \langle SD, \delta_{\Sigma}^n \rangle, n = \#(Msg)$$

$m_1 m_2 \cdots m_n$  组成了一个有穷消息序列  $Meseq$ , 表示一个交互场景中所有的交互消息。

对  $\forall m_i \in Meseq$ , 存在  $S(m_i) = O_p, R(m_i) = O_q$

$$\langle O_p, \sigma_p \rangle \xrightarrow{!m_i} \langle O_p, \sigma'_p \rangle, \langle O_q, \sigma_q \rangle \xrightarrow{?m_i} \langle O_q, \sigma'_q \rangle$$

### 4 结语

UML 顺序图模型广泛应用于面向对象软件的交互建模阶段,同时,它也可以作为生成交互测试用例的依据。本文针对顺序图在形式化语义方面的不足,在定义顺序图形式化语法的基础上,给出了它的结构化操作语义,为模型的一致性检验和测试用例的生成提供了前提。

#### 参考文献:

- [1] 徐锋. 实战 OO: 交互建模[J]. 程序员, 2004, 05.
- [2] WHITTLE J. Formal Approaches to Systems analysis Using UML: An Overview[J]. Journal of Database Management, 2000, 11(4): 4-13.
- [3] 陆汝. 计算机语言的形式语义[M]. 北京: 科学出版社, 1994.
- [4] OVERGAARD G. A Formal Approach to Collaborations in the Unified modeling Language[A]. In Proceedings of the Second IEEE International Conference on the Unified Modeling Language (UML99) [C]. IEEE Computer Society Press, 1999. 99-111.
- [5] TSIOLAKIS A. Semantic Analysis and Consistency Checking of UML Sequence Diagrams[R]. Diplomarbeit, TU - Berlin: [TR2001-06], April 2001.