UML 状态机的形式语义*

蒋 慧[1], 林 东[2], 谢希仁[1]

[1](解放军理工大学 计算机系,江苏 南京 210007);
[2](国防大学 战略教研室,北京 100091)
E-mail: osman@263.net; donglin_cn@sina.com; xiexr@public1.ptt.js.cn

摘要: 许多大型系统在进行分析和设计时,均采用 UML 作为需求描述语言,尤其是一些对安全性要求较高的系统,更是广泛采用 UML 的动态行为描述机制——状态机来描述协议及控制机制.但是,由于 UML 没有形式化的动态语义,不利于对其所描述的需求进行形式化验证和证明.为了解决这一问题,采用以下方法为 UML 状态机构建形式语义.把 UML 状态机中的状态映射到一种项代数上,用归纳的状态项表示状态机的状态.然后,把状态项映射到一种加标记的变迁系统 LTS 上,LTS-状态是状态机的状态项,LTS-变迁是 UML 状态机的微步.最后,用 Plotkin 风格的结构操作语义 SOS(structural operational semantics)规则归纳地给出满足组合性的 UML 状态机语义.此方法既是对一些经典 Statechart 形式化方法的综合,又针对 UML 状态机的特点作了创新,使状态项能够动态地描述任意时刻 UML 状态机的配置树,简化 LTS 的标记,同时,结构化的语义规则更为形式化验证奠定了基础.

关 键 词: UML 状态机;形式语义;结构化操作语义(structural operational semantics,简称 SOS);状态图;加标记的变迁系统

中图法分类号: TP311 文献标识码: A

随着 OMG 组织采纳 UML 作为面向对象分析和设计建模语言的标准[1],UML 被广泛使用和推广.许多大型系统均采用 UML 作为需求描述语言进行分析和设计.但是,尽管 UML 的静态语义由元模型(meta-model)给出[2],但其动态语义却十分模糊,不利于对其所描述的需求进行形式化的验证和确认,这一点已经成为阻挠在安全性要求高的系统开发中使用 UML 的主要原因.为 UML 构造形式语义,人们做了大量的工作[3~8].本文将讨论为 UML 的一种重要动态行为描述机制——状态机(state machine)构造的形式化语义.状态机作为 UML 动态描述机制的重要组成部分,在描述系统及模型的动态行为时扮演着重要的角色.在一些对安全性要求较高的实时系统、嵌入式系统、分布式系统以及调度系统中,是描述协议、控制单元等组件的常用而重要的手段.由于系统往往对这些组件的正确性要求较高,因此,为 UML 状态机构造形式化的语义就显得尤为迫切.因为这不仅有利于准确而无二义性地理解其所表示对象的行为,有利于系统的代码生成及优化,而且更有助于对系统的正确性和安全性进行形式化验证和证明.

1 UML 状态机的状态项代数

UML 的状态包括简单状态、非并发复合状态和并发复合状态、初始状态、最终状态和子状态机状态、历史状态、同步状态.按如下状态分类:(1) 基本状态,包括简单状态、初始状态和最终状态.初始状态和最终状态

* 收稿日期: 2001-04-17; 修改日期: 2001-11-02
基金项目: 国家自然科学基金资助项目(69931040)

作者简介: 蒋慧(1973－),女,重庆人,博士,讲师,现在清华大学计算机科学与技术系博士后流动站工作.主要研究领域为面向对象,软件工程,形式化方法;**林东**(1966－),男,江苏南京人,博士,副教授,主要研究领域为运筹学,计算机仿真,人工智能,决策支持;**谢希仁**(1931－),男,福建晋江人,教授,博士生导师,主要研究领域为计算机网络.

没有具体的状态名,只是用来表明进入或退出某个状态.(2) 或状态,即非并发复合状态.在某一时刻只能有一个子状态处于活跃状态.(3) 与状态,即并发复合状态,有两个以上互不关联的(或正交的 orthogonal)的逻辑与组件,称为区域(region).

1.1 状态项代数

假设 UML 状态机的状态名空间为 N,其中 $initial_s \in N$, $finial_s \in N$,分别表示状态项 s 的初始状态和最终状态(如果有的话).整个状态机中变迁的集合用 T 表示.UML 状态机的状态项代数定义如下:

定义 **1**. UML 状态机的状态项集 SA 是由以下规则所定义的最小集合:

(1) 空状态∅:是一个状态项,它只表示状态机初始状态的父状态;

(2) 基本状态:对每一 $n \in N, s_p \in SA$,是状态 n 的父状态的状态项,$s=\lfloor n:s_p;entryAction;DoActivity;ExitAction \rfloor$则是一个状态项.$n$ 是该状态的状态名 $\hat{s}$,$EntryAction$,$DoActivity$,$ExitAction$ 分别表示该状态的入口动作、活动和出口动作,由函数 $Entry: SA \to Action$, $Do: SA \to Action$ 和 $Exit: SA \to Action$ 可分别得到 3 个动作,即对 $s \in SA$, $Entry(s)=EntryAction$, $Do(s)=DoActivity$,$Exit(s)=ExitAction$;函数 s 是表示状态 n 的状态项;

(3) 与状态:如果 $n \in N$, $s_p \in SA$,对 $k>0$,SA 中的项 $s_1,s_2,\ldots,s_k$ 是状态 n 的正交区域,是状态 n 的并发子状态,s_p 是状态 n 的父状态的状态项,则 $s=\lfloor n:s_p;(s_1,s_2,\ldots,s_k);entryAction;DoActivity;ExitAction \rfloor$是状态 n 的一个状态项;

(4) 或状态:如果 $n \in N$, $s_p \in SA$,s_p 是状态 n 的父状态的状态项,则 $s=\lfloor n: s_p;(s_1,s_2,\ldots,s_k);l;T_n;h;D_h;EntryAction;DoActivity;ExitAction \rfloor$是状态 n 的状态项.其中,对 $k>0$,SA 中的项 $s_1,s_2,\ldots,s_k$ 是状态 n 的子状态; $l \in \{1,2,\ldots,k\}$,表示 s_l 是状态 n 的当前活跃子状态;T 是状态 n 中连接各子状态的所有变迁的集合; $h \in \{none, shallow, deep\}$ 是一个标记,表示该或状态是否拥有历史状态,$none$ 表示没有,$shallow$ 表示有一个浅历史状态,$deep$ 表示有一个深历史状态; $D_h \in \{1,2,\ldots,k\}$,表示 S_{Dh} 是默认的历史状态.

关于 UML 状态机的状态项有如下函数:

(1) 函数 $Type: SA \to \{\text{basic, or, and}\}$,给出状态项的类型,只有 basic,or 和 and 这 3 种;

(2) 函数 $parent: SA \to SA$,求出给定状态的父状态:$parent(s)=s_p$.每个状态都有且只有一个父状态,整个状态机中有一个状态没有父状态,这就是状态机的初始状态,称它的父状态为空,用∅表示.

UML 状态机的变迁包括 5 个部分:源状态、目标状态、触发器、卫士和效果.在状态图中是一个箭头,箭头两端分别是源状态和目标状态,箭头上是“触发器[卫士条件]/效果”(event-signature ‘[’ guard-condition ‘]’ ‘/’ action-expression)表示其他 3 个部分.

定义 **2**. 在或状态 $s=\lfloor n:s_p;(s_1,s_2,\ldots,s_k);l;T_n;h;D_h;entryAction;DoActivity;ExitAction \rfloor$中,变迁集合 T_n 是连接该或状态的各个子状态的变迁的集合,它的元素变迁 t 是一个 6 元组$\langle \hat{t}, i,e,g,a,j \rangle$.其中, $\hat{t} \in TN$ 是变迁 t 的名字, $i, j \in \{1,2,\ldots,k\}$ 分别表示变迁 t 的源状态和目标状态在或状态的子状态集中的索引, $e \in E$ 是触发该变迁的事件,称为该变迁的触发器,$g \in Guard$ 是变迁的卫士条件,$a \in A$ 是该变迁的效果.则有:

(a) $name(t)=\hat{t}$;

(b) $source: T \to SA$, $source(t)=s_i$,是 t 的源状态;

(c) $target: T \to SA$, $target(t)=s_j$,是 t 的目标状态;

(d) $trigger: T \to E$, $trigger(t)=e$,是 t 的触发器;

(e) $guard: T \to \{\text{true, false}\}$ 是布尔函数,取得 t 的卫士条件的真值;

(f) $effect: T \to A$, $effect(t)=a$,是 t 的效果.

定义 **3**. 变迁函数: $trans: SA \to 2^T$.给出一个状态的所有变迁:

$$trans(s)=\begin{cases}\varnothing & s\text{是基本状态}\\ T_s \cup \bigcup\{trans(s_i) \mid 1 \le i \le k\} & s=[\hat{s}: s_p;(s_1,\ldots,s_k);l;T_s;h;D_h]\text{是或状态}\\ \bigcup\{trans(s_i) \mid 1 \le i \le k\} & s=[\hat{s}: s_p;(s_1,\ldots,s_k)]\text{是与状态}\end{cases}.$$

1.2 状态机的配置函数

如果状态机当前处于某个简单状态,则它是活跃的,并且所有直接或传递地包含该状态的复合状态都是活跃的,而在同一层的复合状态还可能是并发的.所以,状态机当前活跃的"状态"可以用一个状态树来表示,状态机的初始状态位于树的根,叶子结点是基本状态,中间结点是复合状态.称状态树为一个状态配置(state configuration).

定义 **4**. 用 $C=SA\cup\{(\}\cup\{)\}\cup\{,\}$ 上的串表示状态机的状态配置 C^+,C 上的配置只能由以下规则产生:

(1) 对任意 $s\in SA$,则 s 是一个状态配置,$s\in C^+$;

(2) 对任意状态配置 $s,s_1,\ldots,s_k$,则 $s(s_i)\in C^+(i\in\{1,2,...,k\})$,$s(s_1,\ldots,s_k)\in C^+$是状态配置.若 $s(s_2),s_2(s_{21},\ldots,s_{2k})$是状态配置,则 $s(s_2)[s_2\leftarrow s_2(s_{21},...,s_{2k})]$也是状态配置,表示用 $s_2(s_{21},\ldots,s_{2k})$置换 $s(s_2)$中的 s_2,得到 $s(s_2(s_{21},...,s_{2k}))$.若 $s(s_1),s_2(s_{21},\ldots,s_{2k})$是状态配置,则 $s(s_1)\circ s_2(s_{21},...,s_{2k})$也是状态配置,表示 $s(s_1)$和 $s_2(s_{21},\ldots,s_{2k})$毗连,得到 $s(s_1)(s_2(s_{21},...,s_{2k}))$.

状态项代数最重要的性质是每个状态项都能够惟一地确定状态机当前的状态配置.下面给出配置函数.

定义 **5**. "现在以前"函数: $backward:SA\to C^+$.给定状态项 $s\in SA$,s_p=*parent*(s)是 s 的父亲,$backward(s)$ 给出所有当 s 活跃时而处于活跃的状态,即给出"从过去到现在"的部分配置.

$$backward(s)=\begin{cases}\hat{s} & s_p=\varnothing\\ backward(s_p)[\hat{s}_p\leftarrow\hat{s}_p(\hat{s})] & s_p=[\hat{s}_p:s'_p;\vec{s};l;T_{sp};h;D_h]\text{或状态}.\\ backward(s_p)[\hat{s}_p\leftarrow\hat{s}_p(\hat{s}_1,...,\hat{s}_k)] & s_p=[\hat{s}_p:s'_p;(s_1,...,s_k)]\text{与状态}\end{cases}$$

定义 **6**. "现在以后"函数: $forward:SA\to C^+$.给定状态项 $s\in SA$, $forward(s)$ 给出当 s 活跃时,把其中的或状态用它的当前活跃子状态代替,即给出"现在以后"的配置子串.

$$forward(s)=\begin{cases}\hat{s} & s\text{是基本状态}\\ \hat{s}(forward(s_l)) & s=[\hat{s}:s_p;\vec{s};l;T_s;h;D_h]\text{是或状态}.\\ \hat{s}(forward(s_1),...,forward(s_k)) & s=[\hat{s}:s_p;(s_1,...,s_k)]\text{是与状态}\end{cases}$$

状态 s 的状态配置可通过如下方式得到:用"现在以前"函数 *backward*()求出所有因 s 活跃而处于活跃的 s 的祖先,然后用"现在以后"函数 *forward*()求出所有或状态的当前活跃子状态.这就是完整的当前配置:函数 *configuration*().该函数把"现在以前"配置中所有处于底层的或状态,用它的"现在以后"替换以后,就得到当前配置.

定义 **7**. 当前配置函数: $configurarion:SA\to C^+$.在给定一个状态项 $s\in SA$ 时,*configuration*(s) 给出当状态机处于 s 时的状态配置:

$$configuration(s)=backward(s)[s_i\leftarrow forward(s_i)](s_i\in backward(s)\wedge bottom(s_i)=true\wedge Type(s_i)\neq basic).$$

2 从 UML 状态机的项代数到加标记的变迁系统 LTS

用微步来定义 UML 状态机的操作语义,从而使语义是组合的:任意一个 UML 状态机的宏步都可以用微步的序列表示.与文献[9~11]中一样,采用加标记的变迁系统(labelled transition system,简称 LTS)作为语义的论域.其中,LTS-状态是 UML 状态项(也即 UML 状态机的一个状态配置),LTS-变迁是 UML 状态机的微步.

定义 **8**. UML 状态项 $s\in SA$ 的语义 〖s〗 由 $(SA,Label,\to,s)\in$ LTS 给出,其中

(1) SA 是状态集合;

(2) $Label$: $E\times G\times A\times 2^E\times TN$,是标记的集合;

(3) $\to\subseteq SA\times(E\times G\times A\times 2^E\times TN)\times SA$ 是变迁的集合;

(4) s 是初始状态.

如果有 $(s,(trigger,guard,effect,out,\hat{t}),s')\in\to$,则用 $s\xrightarrow[out]{(trigger,guard,effect)}\hat{t}\ s'$ 表示,称 UML 状态项 s 在标记 $(trigger,guard,effect,out,\hat{t})$ 下变迁到状态项 s'.其中,*trigger*,*guard*,*effect*,*out* 和 $\hat{t}$ 分别称为这个变迁的触发器、卫

士条件、效果动作、产生事件和名.

2.1 UML状态机的组合操作语义

在加标记的变迁系统 LTS 上,以 Plotkin 的结构化操作语义 SOS(structural operational semantics)风格[12]给出 UML 状态机的组合操作语义.每条 SOS 规则的形式为:

$$\text{规则名}\ \frac{\text{前提}}{\text{结论}}.$$

(边缘结果)

其中,边缘结果会给出整个变迁所产生的新事件集 O 以及 $s \mapsto s'$,表示状态 s 在变迁后改变到形式 s'.

下面是 UML 状态机的 SOS 规则.设当前状态为 s_i,当前事件为 e,当前要完成的变迁 t=*next_tranx*(),g 是 t 的卫士条件,$\hat{t}$ 是 t 的名.

2.1.1 进入或状态

一旦进入或状态,将会区分以下 3 种情况:

(1) 默认入口 ImEnterOr:表示时,进入变迁终止于复合状态的外边.这时完成默认的变迁,如果该变迁有卫士,则必须为真.在执行与初始变迁相关的动作之前,执行状态的入口动作.

$$\text{ImEnterOr:}\ \frac{target(t)=s_j=\lfloor \hat{s}_j : s_{jp};\vec{s};l;T_j;h;D_h \rfloor}{s_i \xrightarrow[O]{(e,g)} \hat{t}\ s_j}.$$

$$\begin{pmatrix} s_i \mapsto update(s_i) \\ O = out(s_i) \triangleright OE(effect(t)) \triangleright OE(enter(s_j)) \\ \quad \triangleright OE(Activity(s_j)) \triangleright OE(enter(s_{jl})) \triangleright OE(Activity(s_{jl})) \end{pmatrix}$$

(2) 明确入口 ExEnterOr(explicit entry):变迁明确地进入或状态的某个子状态,则该子状态变成活跃的,并且在执行完复合状态的入口代码后执行子状态的入口代码.如果变迁终止于一个传递嵌入的子状态,则递归地应用这一规则.

$$\text{ExEnterOr:}\ \frac{\begin{array}{c} target(t)=s_{jk} \\ parent(s_{jk})=s_j=[\hat{s}_j : s_{jp};\vec{s};l;T_j;h;D_h] \end{array}}{s_i \xrightarrow[O]{(e,g)} \hat{t}\ s_l}.$$

$$\begin{pmatrix} s_i \mapsto update(s_i) \\ s_j \mapsto [\hat{s}_j : s_{jp};\vec{s};k;T_j;h;D_h] \\ O = out(s_i) \triangleright OE(effect(t)) \triangleright OE(enter(s_j)) \\ \quad \triangleright OE(Activity(s_j)) \triangleright OE(enter(s_k)) \triangleright OE(Activity(s_k)) \end{pmatrix}$$

(3) 历史入口 HisEnterOr(shallow history entry):若该子状态是一个历史伪状态,如果变迁终止于或状态的一个历史伪状态,则在进入历史入口之前,活跃子状态成为最近活跃子状态,除非最近的活跃子状态是一个最终状态或者这是第 1 次进入该状态.在 *update*()函数中,已经在每次退出某个状态时,根据历史标签,更新当前活跃子状态指针,因此,对于历史入口,则认为它是进入该状态的当前活跃子状态.

$$\text{HisEnterOr:}\ \frac{\begin{array}{c} target(t)=s_{D_h} \\ parent(s_{D_h})=s_j=[\hat{s}_j : s_{jp};\vec{s};l;T_{sj};h;D_h] \\ h \in \{shallow, deep\} \end{array}}{s_i \xrightarrow[O]{(e,g)} \hat{t}\ s_l}.$$

$$\begin{pmatrix} s_i \mapsto update(s_i) \\ O = out(s_i) \triangleright OE(effect(t)) \triangleright OE(enter(s_j)) \\ \quad \triangleright OE(Activity(s_j)) \triangleright OE(enter(s_l)) \triangleright OE(Activity(s_l)) \end{pmatrix}$$

2.1.2 或状态内子状态之间的转移:WithinOr

若变迁是在某或状态的两个子状态之间进行,则需要修改该或状态的当前活跃状态指针,并且根据历史标签递归地更新所有子状态.

$$\text{WithinOr: } \frac{source(t)=s_i \quad target(t)=s_j \quad parent(s_i)=parent(s_j)=s=\left[\hat{s}:s_p;\vec{s};i;T_s;h;D_h\right]}{s_i \xrightarrow[O]{(e,g)} \hat{t}\, s_j}.$$

$$\begin{pmatrix} s_i \mapsto update(s_i) \\ s \mapsto \left[\hat{s}:s_p;\vec{s}\left[s_j \leftarrow update\left(s_j\right)\right];j;T_s;h;D_h\right] \\ O=out(s_i)\triangleright OE(effect(t))\triangleright OE(enter(s_j))\triangleright OE(Activity(s_j)) \end{pmatrix}$$

2.1.3 进入与状态

一旦进入与状态,则默认或明确地进入它的所有并发子状态(区域).将会区分以下两种情况:

(1) ImEnterAnd 如果变迁终止于复合状态的边,则默认地进入所有区域.

$$\text{ImEnterAnd: } \frac{target(t)=s_j=\left[\hat{s}_j:s_p;(s_1,...,s_k)\right]}{s_i \xrightarrow[O]{(e,g)} \hat{t}\, s_j}.$$

$$\begin{pmatrix} s_i \mapsto update(s_i) \\ s_j \mapsto update(s_j) \\ O=out(s_i)\triangleright OE(effect(t))\triangleright OE(enter(s_j)) \\ \quad \triangleright OE(Activity(s_j))\triangleright \bigcup_{m=1}^{k}(OE(enter(s_m))\triangleright OE(Activity(s_m))) \end{pmatrix}$$

(2) ExEnterAnd 如果变迁明确地进入一个或多个区域(如派生),则明确地进入这些区域,默认地进入其他区域.

$$\text{ExEnterAnd: } \frac{target(t)=s_m \quad parent(s_m)=s_j=\left[\hat{s}_j:s_p;(s_1,...,s_k)\right]}{s_i \xrightarrow[O]{(e,g)} \hat{t}\, s_j}.$$

$$\begin{pmatrix} s_i \mapsto update(s_i) \\ s_j \mapsto update(s_j) \\ O=out(s_i)\triangleright OE(effect(t))\triangleright OE(enter(s_j)) \\ \quad \triangleright OE(Activity(s_j))\triangleright \bigcup_{q=1}^{k}(OE(enter(s_q))\triangleright OE(Activity(s_q))) \end{pmatrix}$$

3 结 论

UML 状态机对经典状态机做了很多扩展.本文在以下几个方面有所创新:(1) 与文献[9]相比,本文定义状态项代数的方法优点在于,所定义的状态项包含父状态信息,可以向前递归地找出“现在以前”的历史状态配置,又可以向后找出“现在以后”的活跃状态配置,从而使每个状态项都确实地代表状态机的当前配置;(2) 引入配置串概念明确表示 UML 状态树,有助于精确地表示 UML 状态机的动态语义;(3) 引入 3 个配置函数可以求出在任一时刻 UML 状态机的状态配置,体现出 UML 状态机的不确定性;(4) 与其他有关 UML 状态机的语义的工作相比[13~15],本文给出的 UML 结构操作语义满足组合性和因果性,适于进行形式化验证.本文今后的工作是尝试在一些形式化验证工具中,利用本文给出的形式化语义进行具体的验证,从而进一步完善.

References:

[1] UML version 1.3. http://www.omg.org.

[2] Booch, G., Rumbaugh, J., Jacobson, I. The Unified Modeling Language User Guide. Boston: Addison-Wesley, 1999.

[3] Jézéquel, J.M., Le Guennec, A., Pennaneach, F. Validating distributed software modeled with UML. In: Bézivin, J., Muller, P.A., eds. Proceedings of the 1st International Workshop on the Unified Modeling Language, UML'98—Beyond the Notation. Vol. 1618 of LNCS, Springer-Verlag, 1998. 331~340. http://www.essaim.univ-mulhouse.fr/uml/evenements/.

[4] Evans, A., France, R., Lano, K., *et al*. Developing the UML as a formal modelling notation. In: Bézivin, J., Muller, P.A., eds. Proceedings of the 1st International Workshop on the Unified Modeling Language, UML'98—Beyond the Notation. Vol. 1618 of LNCS, Springer-Verlag, 1998. 293~307. http://www.essaim.univ-mulhouse.fr/uml/evenements/.

[5] Offutt, J., Abdurazik, A. Generating tests from UML specifications. In: France, R., Rumpe, B., eds. Proceedings of the 2nd International Conference on UML'99, the United Modeling Language, Beyond the Standard. Vol. 1723 of LNCS, Springer-Verlag, 1999. http://www.cs.colostate.edu/UML99/.

[6] Araújo, J. Formalizing sequence diagrams. In: Andrade, L., Moreira, A., Deshpande, A., eds. Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Why? How? 1998. http://www.acm.org/sigplan/oopsla/oopsla98/.

[7] Geisler , R. Precise UML semantics through formal metamodeling. In: Andrade, L., Moreira, A., Deshpande, A., eds. Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Why? How? 1998. http://www.acm.org/sigplan/oopsla/oopsla98/.

[8] Kim, S.K., Carrington, D. Formalizing the UML class diagram using object_z. In: Rumpe, B., France, R.B., eds. Proceedings of the 2nd International Conference on the Unified Modeling Language. Vol. 1723 of LNCS. 1999. http://www.cs.colostate.edu/UML99/.

[9] von der Beeck, M. A concise compositional statecharts semantics definition. In: Proceedings of the FORTE/PSTV 2000. Kluwer, 2000. http://forte-pstv-2000.cpr.it/

[10] Uselton, A., Smolka, S. A process-algebraic semantics for statecharts via state refinement. In: Olderog, E-R., ed. Proceedings of the IFIP TC2/WG2.1/WG2.2/WG2.3 working Conference on Programming Concepts, Methods and Calculi (PROCOMET'94), North Holland/Elsevier, 1994.

[11] Maggiolo-Schettini, A., Peron, A., Tini, S. Equivalences of statecharts. In: Montanari, U., Sassone, V., eds. Proceedings of the CONCUR'96 (Concurrency Theory). Vol. 1119 of LNCS, Springer-Verlag. 1996. 687~702.

[12] Plotkin, G. A structural approach to operational semantics. Technical Report, DAIMI-FN-19, Computer Science Department, Aarhus University, 1981.

[13] Paltor, I., Lilius, J. Formalising UML state machines for model checking. In: France, R., Rumpe, B., eds. Proceedings of the Unified Modeling Language (UML'99). Vol. 1723 of LNCS. Springer-Verlag, 1999. http://www.cs.colostate.edu/UML99/.

[14] Lilius, J., Paltor, I. vUML: a tool for verifying UML models. Technical Report No. 272, Turku Centre for Computer Science, 1999.

[15] Wang, Y, Talpin, J.P. Pre-Order semantics of UML state-machine. Research Report, No. 3958, INRIA, 2000.

The Formal Semantics of UML State Machine*

JIANG Hui[1], LIN Dong[2], XIE Xi-ren[1]

[1](*Department of Computer, PLA University of Science and Technology, Nanjing* 210007, *China*);

[2](*Division of Strategy, Defense University, Beijing* 100091, *China*)

E-mail: osman@263.net; donglin_cn@sina.com; xiexr@public1.ptt.js.cn

Abstract: More and more large systems are taking UML as requirements description language for system analysis and design, especially in those safety-critical systems. One of the most important dynamic behavior specifying mechanism of UML——the UML state machine, is widely used for specification of communication protocols and control units. Unfortunately, UML has no strictly defined formal dynamic semantics. It is difficult to do formal verification and proof on the requirements. In this paper, a formal semantics of UML state machine is built. The UML state is firstly represented by inductive state term from some kind of term algebra. Secondly, a labeled transition system (LTS) is introduced, in which an LTS-state is a UML state term, an LTS-transition is a

micro step of UML state machine. In the end, a set of Plotkin-style structural operational semantics (SOS) rules inductively defines a compositional formal semantics for UML state machine. This method not only synthesizes those formal methods for classical Statecharts, but also makes innovation addressed to UML state machine. At any time, the configuration of the machine can be inferred from the state term. The simplified LTS-label and structuralized operational semantics rules will play a fundamental role in formal verification.

Key words: UML state machine; formal semantics; SOS(structural operational semantics); statechart; labeled transition system

* Received April 17, 2001; accepted November 2, 2001
Supported by the National Natural Science Foundation of China under Grant No.69931040

IEEE/WIC2003 网上智能与智能代理技术联合国际会议(WI2003/IAT2003)

征 文 通 知

IEEE/WIC2003 网上智能与智能代理技术联合国际会议(WI2003/IAT2003)由 IEEE Computer Society 和国际网上智能协会(Web Intelligence Consortium),国家自然科学基金委员会,中国计算机学会和北京工业大学联合协办,定于 2003 年 10 月 13 日~17 日在北京举行.会议由 IEEE Press 出版正式的论文集,优秀论文将收录于 IOS Press 出版的 Web Intelligence and Agent System: An International Journal 和 World Scientific Publishing 出版的 Annual Review of Intelligent Informatics.

一、征文范围

WI2003 会议征文范围包括(但不限于)

新的 Web 技术;网格计算;智能人网交互;基于智能 web 的商务;知识网络和管理;语义 web;web 智能体 web 信息管理;Web 信息检索;web 挖掘;知识网络与管理;基于智能网络的商务.

IAT2003 会议征文范围包括(但不限于)

分布式智能;可学习和自适应 Agent;数据和知识管理 Agent;面向自组织的计算范例;计算模型、体系结构及其基础;智能 Agent 的应用.

二、征文要求

论文必须未公开发表,并用英文书写.本次会议只接受电子文档(LaTex、MSWord、PostScript、PDF 文件均可).论文格式和投稿要求,详见会议网址.

三、重要日期

征文截至日期: 2003 年 3 月 20 日

录用通知日期: 2003 年 5 月

四、联系方式

联系地址:Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong
(香港浸会大学计算机科学系)

联系人:Dr. Jiming Liu

电话:(852)3411-7088 传真:(852)3411-7892

E-mail: jiming@comp.hkbu.edu.hk

会议网址: http://www.comp.hkbu.edu.hk/WI03/ 和 http://www.comp.hkbu.edu.hk/IAT03/