从UML顺序图生成状态图的一个方法

袁 海 李宣东 郑国梁

(南京大学计算机软件新技术国家重点实验室,南京大学计算机科学与技术系 南京210093)

A Method for the Transformation from Sequence Diagram to Statechart Diagram

YUAN Hai LI Xuan-Dong ZHENG Guo-Liang

(State Key Lab for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nangjing 210093)

Abstract UML (Unified Modeling Language) is a visual modeling language used for specifying, visualizing, constructing, and documenting the artifacts of software systems by various diagrams. It has been widely accepted as a standard modeling language in both academic and industrial areas. UML sequence diagrams are mostly used in specifying system requirements. By representing interactions, which are arranged in time sequence, between the objects in a system, sequence diagrams can construct scenarios indicating the system's functions. A UML statechart diagram is a graph shows the sequences of states that an object or an interaction goes through during its life in response to received stimuli, together with its responses and actions. It's useful in the design stage of system development. This essay discusses the computer-aided transformation from sequence diagrams to statechart diagrams, which can offer strong support for the transfering from requirement analysis to system design in the software development process. With OCL (Object Control Language) semantic constrain, a transform algorithm is provided in the paper. And the differences with the related works are also mentioned.

Keywords UML, Sequence diagram, Statechart Diagram

1. 引言

UML是由OMG及其合作伙伴共同制定的一种可视化的面向对象软件建模语言。面向对象建模语言起源于20世纪70年代中期,此后不断有新的面向对象的建模工具被开发出来,到90年代中期,共有近50种软件建模语言。这些语言各有优缺点,然而却没有一种能完全满足面向对象软件开发的需求。90年代中期,Booch、Rumbaugh和Jacobson开始致力于开发一种新的集合其他软件建模语言优点的新型建模工具。在OMG,Rational Software公司及其他软件公司的参与下,于1997年1月推出UML 1.0版。UML集合了Booch、OMT、OOSE/Jacobson方法及其他面向对象软件建模方法,采用多种图形对一个软件系统的需求说明、软件构建等进行多角度的描述。UML使用的图形有[1]:

- 用例图(use case diagram)
- 类图(class diagram)
- 行为图(behavior diagrams):
  - 状态图(statechart diagram)
  - 活动图(activity diagram)
  - 交互图(interaction diagram)
    - 顺序图(sequence diagram)
    - 协作图(collaboration diagram)
- 实现图(implementation diagrams):
  - 构件图(component diagram)
  - 配置图(deployment diagram)

本文主要讨论了UML顺序图到状态图的转换。顺序图以时间顺序显示对象在其生命周期内的交互活动,而状态图刻画了对象在其生命周期内接受外部事件触发后经历的状态转换[2]。前者适用于软件的需求说明阶段,而后者在软件设计时使用。在计算机辅助下实现两者的转换,可以对软件开发过程中需求分析到设计这一阶段提供强有力的支持。

2. UML顺序图

UML顺序图展示时间顺序上多个对象在自己的生命周期内与其它对象间通过消息传递的交互行为,它并不展示对象间的关系。UML顺序图有两个维度:纵向是时间线,定义对象在交互活动中的生命周期;横向是不同的对象。消息在不同对象之间按时间顺序进行传递。图1、图2是两个顺序图的例子。

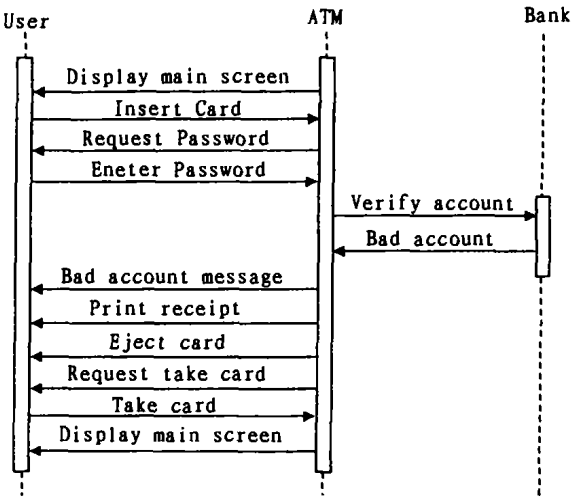

图1

顺序图使得系统需求更容易被客户、需求分析员及软件开发人员所理解。然而,由于顺序图本身没有包含足够的语义信息,给顺序图模型的自动分析和验证带来了困难。因此,我

们在进行顺序图到状态图的转换之前，必须对顺序图进行扩展，添加必要的语义信息。UML 提供了一种方便的语言——对象约束语言(Object Constraint Language，OCL)，使用户能在图上表示约束，提供一定的语义信息。我们使用 OCL 为每个对象加上语义约束，这个约束包含每个对象的状态向量〈state vector〉，如图3、图4、图5所示。

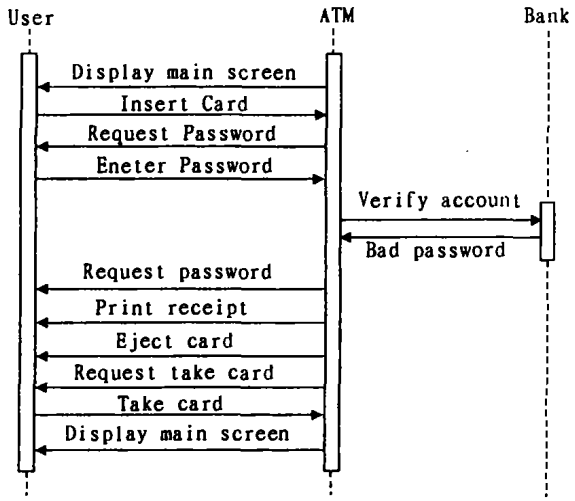

图2

CardIn，CardHalfway，PasswdGiven：Boolen
Insert card(c：card)
Pre：CardIn＝false
post：CardIn＝true
Enter password(p：Sequence)
Pre：passwdGiven＝false
Post：passwdGive＝true
Bad password()
Pre：passdGiven＝true
Post：passwdGive＝false
Eject card()
Pre：cardIn＝true
Post：cardIn＝false and cardHalfway＝true and passwdGiven＝false
Take card()
Pre：cardIn＝false and cardHalfway＝true
Post：

图3　User 类的 OCL 约束

card：Card，passwd：Sequence
Verify account(c：Card，p：Sequence)
Pre：
Post：card＝c and passwd＝p
Bad account()
Pre：
Post：card＝null and passwd＝null
Bad password()
Pre：
Post：card＝null and password＝null

图4　Bank 类的 OCL 约束

CardIn，CardHalfway，PasswdGiven，AccountRight，PasswdRight，Cancel：Boolen
card：Card，passwd：Sequence
Display main screen()
Pre：cardIn＝false and cardHalfway＝false
Post：
Insert card(c：card)
Pre：cardIn＝false
Post：CardIn＝true and card＝c
Request password()
Pre：passwdGiven＝false
Post：
Enter password(p：Sequence)
Pre：passwdGiven＝false and p→for All(p→include(d) ◊ digit(d))
Post：passwdGive＝true and passwd＝p
Verify account(c：Card. p：Sequence)
Pre：c≠null and p ≠null
Post：
Bad account()
Pre：
Post：AccountRight＝false and card＝null
Bad password()
Pre：
Post：PasswordRight＝false and PasswdGiven＝false and passwd＝null
Eject card()
Pre：cardIn＝true
Post：cardIn＝false and cardHalfway＝true and passwdGiven＝false and card＝null and passwd＝null
Request take card()
Pre：cardHalfway＝true
Post：
Cancel()
Pre：
Post：＝Cancel＝true
Cancel message()
Pre：Cancel＝true and cardIn＝true
Post：

图5　ATM 类的 OCL 约束

根据以上的工作，我们可以定义 UML 顺序图 SD：

定义1　UML 顺序图 SD 是一个多元组，SD ＝ (M，O，λ)，

其中，M 是一个消息的一个偏序集。定义 M 上的操作“＜”：$m_i < m_j$，$i<j$ 表示消息 m_i 不晚于消息 m_j 被发送。O 是对象集合；

这里对象定义为二元组〈ID，S〉，其中 ID 是对象名，S 为对象状态向量集合。$\lambda \subseteq O \times M \times O$ 表示消息传递。m^{source} 和 m^{dest} 分别表示消息 m 的发送和接受对象，$m^{source} \in O$，$m^{dest} \in O$。vs_i 和 vs_i' 分别表示消息 m_i 的源对象 m_i^{source} 发送消息前后的状态；vd_i 和 vd_i' 分别表示消息 m_i 的目的对象 m_i^{dest} 接受消息前后的状态。

3. UML 状态图

UML 状态图是状态机的图形表示，主要用来描述对象生命周期内状态(state)和动作(action)的序列。这个序列是对特定的事件(event)反应的结果，事件可以是信号(signal)，或者是操作调用(opereation invocation)等。一个状态图主要由状态和转换(transition)构成。状态指的是一个对象的生命周期中的某个条件，或者在满足某个条件、执行某个活动、等待某个事件的过程中的一次交互，用圆角的矩形表示状态。状态包括两个区域：名字区域和内部转换(internal transition)区域。内部转换区域是可选的，它指的是在此状态下将要执行的内部动作或活动(activity)的列表。

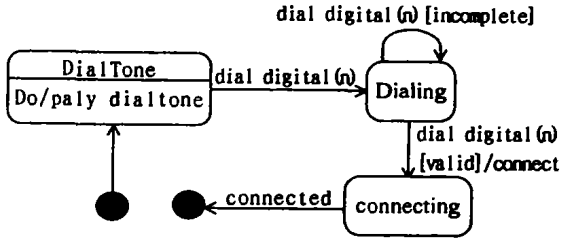

图6　UML 状态图

状态图中状态之间的箭头被称为转换，表明一个对象在某个条件满足并且某个事件发生的时候，有一个状态进入另一个状态。转换通常带有形如 e[c]/a 的标记，其中 e 代表事件，c 代表卫式条件(guard-condition)，a 表示活动。整个标记表示当条件 c 满足的时候，事件 e 的发生将引起转换的击发(fire)，对象将执行活动 a 并进入下一个状态。一个典型的状态图如图6所示。

我们可以用转换系统(Transition System)定义 UML 的状态图。

定义2 UML 状态图 SC 是一个转换系统，SC = (S,T,α,β,s_0,λ,V,μ)。

其中 S 是状态的集合。T 是转换的集合。t:s ⊢a ⊢s' 表示状态 s 经过转换 t 到达状态 s'，转换 t 的标签 λ(t)=a。ε 是空事件，它不做任何事情，也不改变状态图中的状态，即 t:s ⊢ε ⊢s。α 和 β 是两个从 T 到 S 的映射，α(t)和 β(t)分别表示转换 t 的源状态和目标状态。s_0是初始状态。λ 是转换 T 到字符串的映射，是转换 t 的标签。如果消息 m 是状态图中的事件，则其相应的转换标签为 λ(t)=m；如果消息 m 是状态图中的动作，则其相应的转换标签为 λ(t)=/m。V 是状态向量的集合。μ 是 S 到 V 的映射，表示状态对应的状态向量的值。

4. 从 UML 顺序图创建 UML 状态图

对于一个复杂的系统，通常使用多个顺序图来描述系统内部各对象之间的消息传递关系。本文给出了一种从多个相关的顺序图中提取状态图的方法。相关的顺序图是指这些顺序图描述的都是系统的同一个功能侧面，比如图1、图2的 ATM 顺序图。这个算法分为三步：

1. 从一个顺序图中得到单个对象的状态图。
2. 从多个相关顺序图得到单个对象在整个系统中的状态图。
3. 合并对象的状态图，得到系统状态图。

假定存在 n 个相关顺序图 SD_1,SD_2,…,SD_n，系统中有 k 个对象，即 #(∪O_i) = k，前两步算法如下：

1. 为每一个对象创建一个状态图，状态图只有一个初始节点(initial node)，且为当前状态；

2. 对于一个顺序图 SD，包含了 j 个消息 m_1,m_2,…,m_j

(1)所有的对象 O_i，如果 O_i 在 SD 中出现，则其对应的状态图中 Curr:=initial node；

(2)对于消息 m_1，在 m_1^{source}对应的状态图中，

a)如果∃ t∈T，满足 λ(t)=m_1，α(t)=Curr，vs_1'=μ(β(t))，转到3；

b)如果∃ s∈S，满足 vs_1'=μ(s)，vs_1≠vs'_1，则增加一个转换 t，使得 λ(t)=/m_1，α(t)=Curr，β(t)=s，Curr:=s，转到3；

c)增加一个状态 s，使得 μ(s)=vs_1'，并增加一个转换 t，使得 λ(t)=/m_1，α(t)=Curr，β(t)=s，Curr:=s；

(3)在 m_1^{dest}对应的状态图中，做同样的操作，转换时将其中的 vs 和 vs' 分别换成 vd 和 vd'；λ(t)=/m_1 替换为 λ(t)=m_1。

(4)重复2和3处理下一条消息，直到顺序图中所有的消息都被处理完。

3. 重复上一步，处理下一个顺序图。直到所有的顺序图都处理完毕。

根据上面的算法，得到的图1、图2中各个对象的状态图如下面图7、图8、图9所示。

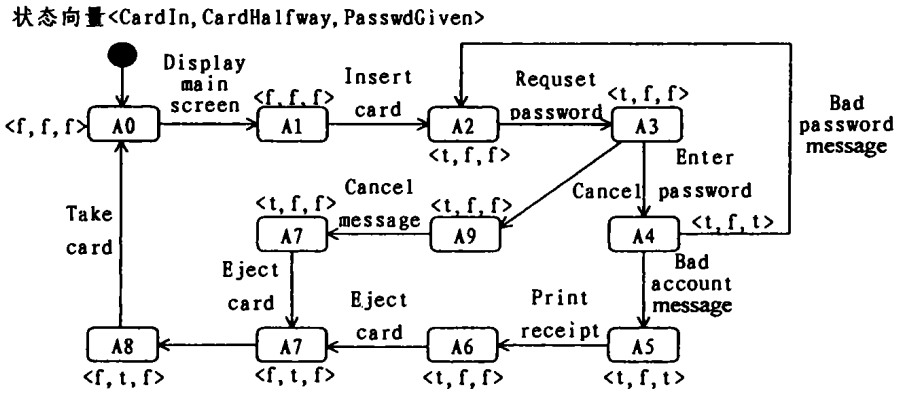

图7 User 对象的状态图

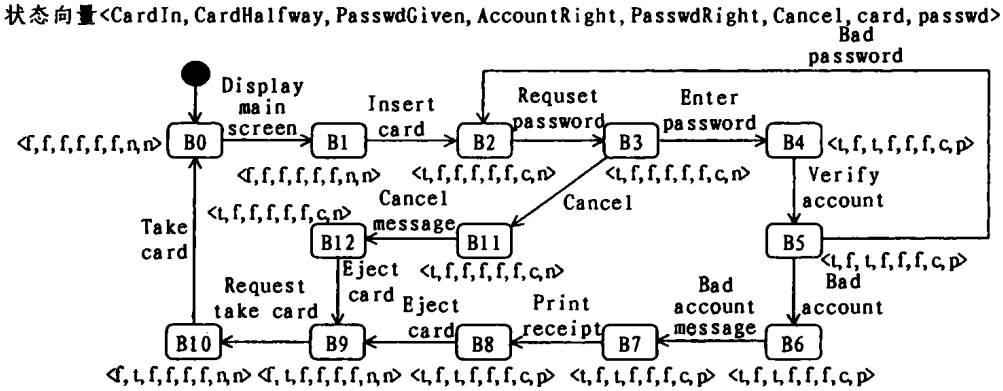

图8 ATM 对象的状态图

4. 最后我们将根据上面算法得到的 k 个状态图 SC_1,SC_2,…,SC_k 进行合并，得到系统的状态图 SC=(S,T,α,β,s_0,λ,V,μ)。SC 是各个状态图在通讯和时间约束下的一个同步积。

其中：$S \subseteq S_1 \times S_2 \times \cdots \times S_k$；$T \subseteq T_1 \times T_2 \times \cdots \times T_k$；α($t_1$,$t_2$,…,$t_n$)=(α($t_1$),α($t_2$),…,α($t_n$))；β($t_1$,$t_2$,…,$t_n$)=(β($t_1$),β($t_2$),…,β($t_n$))；$s_0$={($s_{1.0}$,$s_{2.0}$,…,$s_{k.0}$)}；λ($t_1$,$t_2$,…,$t_n$)=(λ($t_1$),λ($t_2$),…,λ($t_n$))；V=($V_1$,$V_2$,…,$V_k$)；μ=($\mu_1$,$\mu_2$,…,$\mu_k$)；

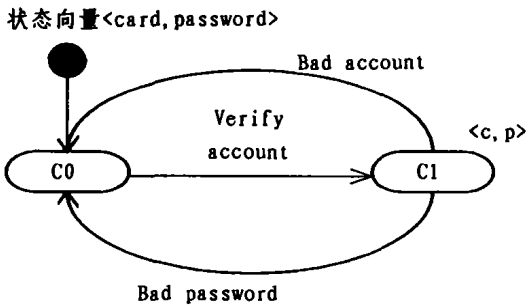

图9 Bank 对象的状态图

如上所述，系统的状态图 SC 是各个子状态图在通讯和时间约束下的一个同步积。而我们可以根据顺序图的特点来定义同步约束的：

1. 一个转换 t 涉及且仅与一个事件对应。该事件涉及到两个对象 m^{source} 和 m^{dest}。也即转换的前后状态 S 只有两个分量发生变化。

2. 在任一时刻仅有一个转换发生。

由此我们可以使用一个递增的构造算法来构造这个全局状态图。

在描述算法之前，我们首先作如下的定义：

定义3 集合 $Seed \subseteq S_1 \times S_2 \times \cdots \times S_n$，$Seed$ 的初始值 $Seed = \{\langle S_{1.0}, S_{2.0}, \cdots, S_{k.0}\rangle\}$，如果 $s \in Seed$，则 $s[i]$ 表示 s 的第 i 个分量。

算法为：

1. 状态图初始状态 $S = Seed$，$T = \phi$。

2. 从 $Seed$ 中任取一个状态 $Sa \in Seed$。

3. 如果存在某个对象 O_i 的转换 t 使得 $\alpha(t) = Sa[s]$，且 $\lambda(t) = m$，则查找是否∃ $S_{i,j}$ 使得 $S_{i,j} = \alpha(t')$，t' 为另一对象 O_d 的一个转换，且 $\lambda(t') = /m$。转到5。

4. 如果存在某个对象 O_d 的转换 t 使得 $\alpha(t) = Sa[d]$，且 $\lambda(t) = /m$，则查找是否∃ $S_{i,j}$ 使得 $S_{i,j} = \alpha(t')$，t' 为另一对象 O_i 的一个转换，且 $\lambda(t') = m$。转到5。

5. 如果满足3、4中的任意一种情况，查找状态图中是否存在与 $Sa = \langle \cdots, \alpha(t), \cdots, \alpha(t'), \cdots\rangle$ 对应的状态 $Sb = \langle \cdots, \beta(t), \cdots \beta(t'), \cdots\rangle$，如果存在则检查两个状态间是否存在转换 $Tab = \langle \varepsilon, \cdots, t, \cdots, t', \cdots \varepsilon\rangle$，没有这样转换的话则进行添加。

如果不存在状态，则添加该状态及转换 Tab。

6. 如果不存在3、4的任何一个状况，转到10。

7. 修改 $Seed$，如果 $Sb \notin S$，$Seed = Seed \cup \{Sb\}$。

8. 修改状态图：$S = S \cup \{Sb\}$，

$T = T \cup \{Tab\}$，

$\alpha(Tab) = Sa$，

$\beta(Tab) = Sb$，

$\lambda(Tab) = m$

9. 重复以上3至8。

10. 修改 $Seed$，$Seed = Seed - \{Sa\}$。

11. 重复以上2至10，直到 $Seed = \phi$。

根据上面算法生成的 ATM 系统的状态图如图10所示。

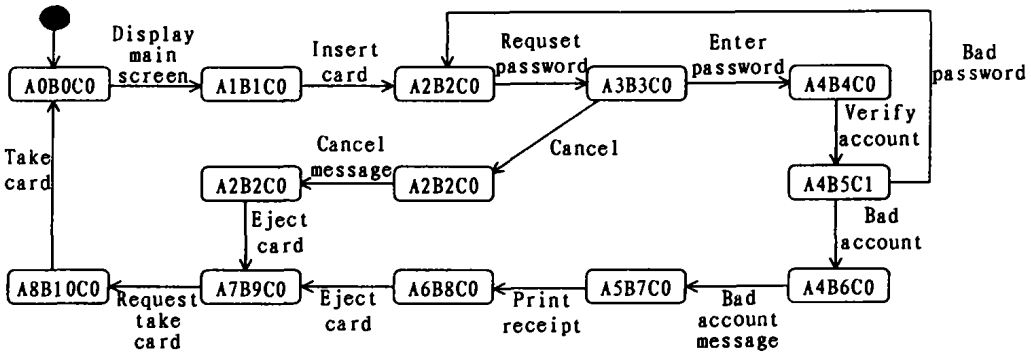

图10 ATM 系统的状态图

根据上面的构造过程，不难证明，对于子状态图 SC_i 中的任意转换 $t: s \vdash m \vdash s'$，在全局状态图中都存在一个转换 $T: S \vdash m \vdash S'$，其中 $S[i] = s$，$S'[i] = s'$，$T[i] = t$。而对于子状态图 SC_i 中的任意一条路径 p，都在全局状态图中至少存在一条对应的路径 P，其中 P 中的转换序列对应于子状态图的部分 P[i]是在 p 中添加了若干 ε 转换构成。

反过来，对于全局状态图中的任意转换 $T: S \vdash m \vdash S'$，其第 i 个分量 T[i]或者等于空转换 ε，或者为某一非空转换 t。若分量为非空转换，根据上面的构造过程，则在第 i 个子状态图中存在转换 $t: s \vdash m \vdash s'$，其中 $s = S[i]$，$s' = S'[i]$，$\lambda(t) = m$ 或者 $\lambda(t) = /m$。因此对于全局状态图中的一条路径 P，它的第 i 个分量 $P[i] = s_0 t_1 s_1 t_2 s_2 \cdots t_n s_n, t_k$ 或者为空转换 ε，或者对应于第 i 个子状态图中的某个转换。对于空转换 ε，由于转换前后状态分量 S[i]不发生变化，我们可以合并转换前后的状态。由此得到新的路径分量 P'[i]，它对应于子状态图 SCi 中的一条路径 p。两者除 λ(t)不同外，其他表示都等同。

本算法的复杂度为 $O(n^2 * m)$，其中 n 为对象数目，m 为转换数目即 $m = \#T$。

结论及相关工作 开发高质量的软件需要在整个生命周期中都有形式化技术的支持。UML 提供了各种丰富的图来支持面向对象的软件开发的全过程。软件开发的一个发展趋势就是自动化程度越来越高，现在已经可以从 UML 图自动实现程序编码，但对于编码前的阶段，还缺少相应的自动化工具。我们所做的工作有利于降低需求分析和概要设计、详细设计阶段的复杂性。S. Some 和 R. Dssouli 提出了一个从多个场景合成时间自动机的算法[4]，该算法以渐进的方式从多个场景构造对应的时间自动机；Jon Whittle 和 Johann Schumann 则给出了一个从 UML 顺序图合成状态图的算法[5]。本文在这些工作的基础上，通过给予 UML 顺序图一定的语义限制，给出了一个在计算机辅助下生成 UML 状态图的一个渐进的构造式算法，与 Whittle 的算法相比，大大降低了算法的复杂度。同时由于每个对象只管理自身的状态，降低了存贮开销。

我们今后的工作将是如何实现其它 UML 图的自动生

（下转第161页）

不难找出其中仿真实体的文法描述同下推自动机的对应关系[3,8]。

仿真实体实现引擎是由下推自动机实现，将图形仿真界面的形式化描述为下推自动机 $A=(Q,\Sigma,\Gamma,\delta,q_0,Z_0)$，其中：

(1)在仿真界面上的各个图元的静止状态构成仿真引擎的状态集合 Q；

(2)仿真实体特征参数构成实体引擎的输入符号集 Σ；

(3)仿真实体模型属性值构成下推符号表 Γ；

(4)初始状态 q_0是仿真环境运行时进入的第一个界面状态；

(5)不同的图元具有不同的仿真实体的模型参数设置以及不同的图元输出函数。根据自动机的当前状态 s，栈顶符号 Z，输入符号 a，构造状态转换函数 δ 以及图元输出函数 μ 见算法1。

算法1 状态转换函数 δ；

① 若 $s=q_0$，$Z=Z_0$时，图元转入执行状态，自动机将产生一个新的图元标识，控制器进行抽象实体图元处理；该图元是仿真实体的超类对象。

② 当 $s=q_i$ 时，若 $a\in\Sigma$ and a ∈{qi 的特征参数}，则接受该输入符号 a，并有$(q_i,a,Z)\vdash-(p,Z_1Z_2\cdots Z_m)$，$m\geqslant 0$。表示 q_i 为嵌套实体，p 为 q_i 的一个子实体，Zi 是 p 的属性标识；即当 q_i 是嵌套属性状态，输入字符为 a 时，下一个状态变为 p，下推栈弹出栈顶符号 Z，压入 $Z_1Z_2\cdots Z_m$，输入指针右移一位。

③ 当 $s=q_i$ 时，若 $a\in\Sigma$ and a ∈{qi 的特征参数}，则接受该输入符号，有$(q_i,a,Z)\vdash-(p,\varepsilon)$，其中 q_i 为原子仿真实体，Z 是 qi 的属性描述。表示完成 Zi 的设置，弹出栈顶属性符号 Zi，转入下一个状态变为 p，控制器指针右移一位。

④ 若 $s=q_i$，$Z=Z_i$，输入为 a_i 时 δ 无定义，则停机并拒绝接受输入符号串。

⑤ 若 $s=q_i$，堆栈为空，输入符号串为空，则停机，接受输入符号串。

输出映射函数 $\mu:Q\times\Gamma\times\Sigma\rightarrow Y$；Y 为仿真实体输出数据集合，$y=\mu(s,Z,a)$。对于具体的图元，μ 取图元的输出函数，即 μ=func_name。

4.3 仿真控制器

算法2给出了下推自动机控制器根据特征参数字符串 $\omega=\omega_1\omega_2\cdots\omega_n$ 以及下推自动机模型，实现仿真实体设置过程。

算法2

1)根据仿真输入获得仿真实体的标识号，由该标识查找相应的下推栈的栈底符号 Z_0，并用 Z_0初始化下推栈。自动机的起始状态 q=Entityinit 为仿真环境起始界面状态。

2)置 j=1。

3)根据状态转换函数 $\delta(q_i,\omega_j,z_k\alpha)=\{(q_{i+1},\gamma\alpha)\}$，其中 $\gamma,\alpha\in\Gamma^*$，$z_k\in\Gamma$ 进行状态转换：$(q_i,\omega_j\omega_{j+1},\omega_{j+2}\cdots\omega_n,z_k\alpha)\vdash(q_{i+1},\omega_{j+1}\omega_{j+2}\cdots\omega_n,\gamma\alpha)$。$j=j+1$。

4)重复3)直到$(q_i,\varepsilon,\varepsilon)$为止，则 $\omega=\omega_1\omega_2\cdots\omega_n$ 构成了仿真实体 Z_0的属性描述。$(\mu(\omega_1),\cdots,\mu(\omega_n))$为仿真输出集合。

结束语 从以上讨论可知，用下推自动机可以实现对图形仿真实体的形式化描述。在仿真描述阶段，可以基于模型设计方法，利用仿真实体的设计流程、设计模型进行仿真平台的设计。由于仿真系统的复杂性，设计模型中存在大量的经验系数，在数据库中存储了以往的设计场景结构数据。自动机的状态转换矩阵能准确地表示平台仿真数据的嵌套关系，还有助于抽取设计经验系数的选取规律。由于该仿真方法是基于规范的体系描述结构进行的，有利于模型扩充和变更。

参 考 文 献

1 Calvin J O. Data Subscription. In: 12th Workshop on Standards for the Interoperability of Distributed Simulation, March 1995. 12

2 Senay H, Ignatius E. A Knowledge-Base System for Visualization Design, 1994, 14(6): 36～47

3 陈崇昕. 形式语言与自动机. 北京邮电出版社, 1988

4 康耀红. 数据融合理论与应用. 西安电子科技大学出版社, 1997

5 王锦，卿杜政，欧阳伶俐，等. 分布式交互仿真技术综述. 系统仿真学报, 1996, 8(3): 1～5

6 DIS Steering Committee, The Dis Vision: A Map to the Future of DIS. http: // ftp. sc. ist . ucf. edu/SISO/dis/library/vision. doc, 1995

7 Johnson K, Puckett J. Using Finite State Machines for Behavior Representation with Composable Object Models in NASM[A]. 1999 Spring Simulation Interoperability Workshop

8 Hopcroft J E, Ullman J D. Introduction to Automata Theory Langues and Computation. Addison Wesley Publishing Company, 1979

(上接第158页)

成，如何在自动合成的过程中发现和消除不一致性，如何利用 UML 对实时系统自动建模，以及实现一个基于 UML 的计算机辅助程序设计工具。

参 考 文 献

1 Rational Software, Microsoft, Hewlett-Packard, Oracle, Sterling Software, MCI Systemhouse, Unisys, ICON Computing IntelliCorp, i-Logix, IBM, ObjecTime, Platinum Technology, Ptech Tskon, Reich Technologies, Softeam, "UML Summary, Version 1.1", 1997.7

2 王云，周伯生. 标准建模语言 UML 简介. 计算机应用研究, 1999, (12): 44～49

3 张幸儿. 计算机编译理论. 科学出版社

4 Some S, Dssouli R. From Scenarios to Timed Automata: building specifications from users requirements

5 Whittle J, Schumann J. Generating Statechart Designs From Scenarios

6 Arnold A. Finite Transition Systems. Prentice Hall, 1992. 5～45

7 Glinz M. An integrated formal model of scenarios based on statecharts. In: 5th European Software Engineering Conf. (ESEC), Sitges, Spain, 1995. 254～271

8 Harel D. Statecharts: A visual formalism for complex systems. Science of Computer Programming, 1987, 8: 231～274

9 Gehrke T, Firley T. Generateive sequence diagrams with textual annotations. In Formal Beschreibungstechniken für verteilte Systeme (FBT99), 1999. 65～72