文章编号:1001 - 2486(2005)05 - 0056 - 06

状态机嵌入 DEVS 的组合建模方法研究*

刘 晨,王维平,朱一凡
(国防科技大学 信息系统与管理学院,湖南 长沙 410073)

摘 要:DEVS 提供了模块化、层次化的系统建模和仿真执行框架,但是缺乏对于系统行为的抽象建模能力。状态图扩展了原有的 FSM,通过可视化的、灵活的状态迁移图描述系统的复杂行为。提出将状态图嵌入 DEVS 的组合建模方法,互为补充,以建立复杂的离散控制系统。DEVS 作为系统的建模框架并提供执行逻辑支持,状态图扩展 DEVS 的事件交互机制和时间推进机制,描述系统的行为逻辑。详细介绍了状态图和 DEVS 的语法语义,重点阐述状态图嵌入 DEVS 的实现机制。结合一个应用实例,说明嵌入状态图的 DEVS 用于建立离散控制系统模型的优势。

关键词:状态机;离散事件系统描述;行为建模;组合建模
中图分类号:TP391.9 文献标识码:A

Research on a Composable Modeling Approach of Embedding the State Machine into DEVS

LIU Chen, WANG Wei-ping, ZHU Yi-fan
(College of Information System and Management, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: DEVS provides a modular and hierarchical system modeling and simulation framework but lacks the ability of behavior modeling. Statecharts extend finite state machine and do outstanding jobs in modeling complicated behavior of systems by means of visual and flexible state - transition diagrams. Hence, a composable modeling approach, embedding the statecharts into DEVS to complement each other, is raised. It can be used to build complicated discrete control system. DEVS, as a system modeling framework, provides simulation logical support and statecharts expand the event interaction mechanism of DEVS which can be used to modeling system behavior logic. The paper details the syntax and semantics of statecharts and DEVS, and emphasizes on the mechanism of embedding statecharts into DEVS. The advantages of building discrete control system models by embedding statecharts into DEVS are illustrated by an application example.

Key words: state machine; discrete event system specification; behavioral modeling; composable modeling

为了验证并校验离散控制系统的设计,出现了多种形式化的建模方法,例如有限状态机、时间自动机、状态图等。目前十分流行的控制系统设计方法是基于仿真模型的设计,已经广泛应用于嵌入式系统的设计过程。例如,Ptolemy 和 Matlab/Simulink 建模仿真环境。基于仿真模型的设计思路是利用一些形式化的高层建模方法建立控制系统的抽象模型,并通过仿真来验证控制系统设计的正确性,然后将这些形式化的高层模型自动生成可执行的代码框架,以实现模型的校验。这种基于高层抽象模型的设计过程可以大大减少控制系统的设计时间,并通过仿真验证和代码生成,大大提高了控制系统的设计质量。

一般的仿真模型可以用 DEVS 框架表示。DEVS 形式具有简洁的操作语义,并且与真实系统存在简单的对应关系。DEVS 通过定义端口通信机制和抽象的系统行为函数来实现系统间的互操作和系统行为,可以方便建立具有时间概念的仿真模型。但 DEVS 是一种贫语义的系统描述,其优势在于对系统的组成结构、通信机制、时间概念的支持,其劣势在于缺乏对于系统行为的描述。而有限状态机、时间自动机等高层建模方法适于建立系统的行为模型,描述状态的转换规律,但又缺少 DEVS 所具备的优点。因

* 收稿日期:2005 - 06 - 10
基金项目:国家自然科学基金资助项目(60574056)
作者简介:刘 晨(1977—),男,博士生。

此,如何将控制系统的高层抽象模型与 DEVS 仿真模型互联是一个关键问题。

Norbert Giambiasi 实现了利用时间自动机来表示抽象的控制系统模型,并且通过将时间自动机映射到 DEVS 的方法实现了两种异构形式模型的互联[1]。时间自动机是一种扁平的状态迁移系统,状态不具有层次性,那么对于复杂的控制系统而言,必将产生状态爆炸的设计问题。而 Statecharts 是在有限状态机的基础上加上层次性、正交性和广播机制的复杂状态迁移系统,可以描述非常复杂的控制系统。

论文提出将 Statecharts 嵌入 DEVS 的思想,即采用 DEVS 描述仿真模型框架,而将 Statecharts 嵌入 DEVS 来代替 DEVS 的状态转移行为描述,以实现 Statecharts 模型与 DEVS 仿真模型的互联。该方法的基本思想参见图 1,设计步骤:(1)利用 Statecharts 描述控制系统的高层抽象模型;(2)利用 DEVS 表示被控系统的仿真模型;(3)将 Statecharts 嵌入到 DEVS 框架中;(4)将嵌入 DEVS 框架的控制系统模型与 DEVS 被控系统模型互联,组成组合 DEVS 仿真模型;(5)进行仿真验证和校验。

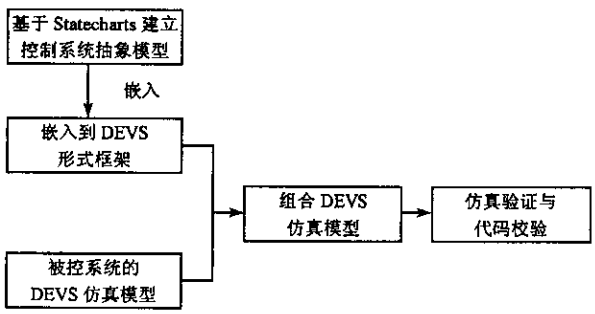

图 1　状态图嵌入 DEVS 的仿真方法

Fig.1　The method of embedding Statecharts into DEVS

1　Statecharts 与 DEVS

1.1　Statecharts

采用 Statemate 状态图的形式语义,考虑层间迁移、历史状态和进入退出状态动作,不考虑 join、fork、choice 等其它伪状态。定义 NS、NT、E、A、SA、T、G、π、H 可数集合,分别表示状态的命名空间、转移的命名空间、事件集合、动作集合、状态项集合、转移集合、守卫条件集合、索引集合、历史状态类型集合。Statecharts 的状态项集合 SA 是由以下状态项组成:

(1)空状态 $\emptyset$。(2)基本状态:若 $s=[n:s_p;entry;exit]$,$n\in N$,$s_p\in SA$,$entry$,$exit\in A$,则 s 是一个基本状态的状态项。n 是基本状态的状态名,s_p 是状态 n 的父状态的状态项。(3)与状态:若 $s=[n:s_p;(s_1,s_2,\cdots,s_k);entry;exit]$,$k>0$,$s_1,s_2,\cdots,s_k\in SA$,则 s 是一个与状态的状态项。$s_1,s_2,\cdots,s_k$ 是状态 n 的正交区域或并发子状态。(4)或状态:$s=[n:s_p;(s_1,s_2,\cdots,s_k);l;T_n;entry;exit]$,$l\in\pi$,$s_1,s_2,\cdots,s_k$ 是状态 n 的非并发子状态,s_1 表示状态 n 的当前活跃子状态,T_n 是状态 n 中连接各个子状态的变迁集合。对于任意状态项,有相应的函数:$\mathrm{Name}(s)=\hat{s}$,表示状态项的名称;$\mathrm{Trans}(s):SA\to 2^{T}$,表示源状态为 s 的迁移集合。当前状态配置函数 $\mathrm{Conf}:SA\to 2^{SA}$,可以根据任意当前活跃的状态项,计算出以当前活跃状态项为树根的状态配置。全部状态配置函数 $\mathrm{TotalConf}:SA\to 2^{2^{SA}}$,可以根据任意状态项,计算出以该状态项为树根的可能的所有状态配置。进入动作执行函数 $\mathrm{Entry}:SA\to 2^{A^*}$($A^*$ 表示 A 的元素序列集合),给出了当一个变迁进入到某个目标状态时,执行的入口动作序列。退出动作执行函数 $\mathrm{Exit}:SA\to 2^{A^*}$,给出了当一个变迁退出某个源状态时,执行的退出动作序列。状态更新函数 $\mathrm{Update}:H\times NS\times SA\to SA$,用于确认某个变迁发生后,变迁所对应的目标状态。函数的具体定义参见相关文献[2~4]。

状态图的操作语义可以通过标记系统 LTS 来描述,LTS 的状态是状态图的状态项。状态图操作语义由 LTS $\mathscr{A}$ 给出:$\mathscr{A}=\langle SA,Label,s_0,\to,F\rangle$,其中:$SA$ 为 LTS 状态集,对应状态图的状态项集合;$s_0\in S$ 为初始状态;$Label\subseteq E\times A\times\{0,1\}$ 是标记集合;$\to\subseteq S\times(E\times A)\times S$ 为变迁集合;$F\subseteq Q$ 为终态集,在

加标记变迁系统 LTS 上，以 Plotkin[5] 的结构化语义 SOS 形式给出状态图的操作语义[2]。

$$\text{BAS}\ \frac{-}{[n:s_p;entry;exit]\xrightarrow[\phi]{e}_0[n:s_p;entry;exit]} \qquad \text{AND}\ \frac{\forall i\in\{1,\cdots,k\},s_i\xrightarrow[a_i]{e}S'_i}{[n:s_p;(s_1,\cdots,s_k)]\xrightarrow[a_{v_{i=1}^k f_i}]{e}[n:s_p;(s'_1,\cdots,s'_k)]}\left(\begin{array}{l}a=a_{b(1)}\triangleright\cdots\triangleright a_{b(k)}\\ \exists\ \text{bijection}\ b:\{1\cdots k\}\to\{1\cdots k\}\end{array}\right)$$

$$\text{OR}-1\ \frac{\begin{array}{c}(\hat{t},l,sr,e,g,a,j,td,h)\in T_n,\\ g=true,sr\subseteq Conf(s_1)\end{array}}{[n:s_p;(s_1\cdots s_k);1;T_n]\xrightarrow[O]{e}_1[n:s_p;(s_1\cdots s_k);j;T_n]}\left(\begin{array}{l}exit=\text{Exit}(s_j)\\ entry=\text{Entry}(\text{Next}(h,td,s_j))\end{array}\right)$$

$$O=exit\triangleright a\triangleright entry,\text{Update}(h,td,s_j)\mapsto s_j$$

$$\text{OR}-2\ \frac{S_1\xrightarrow[O]{e}S'_1}{[n:s_p;(s_1\cdots s_k);1;T_n]\xrightarrow[O']{e}_1[n:s_p;(s_1\cdots s_k);1;T_n]}\qquad \text{OR}-3\ \frac{s_1\xrightarrow{e}0s_1,[n:s_p;(s_1\cdots s_k);1;T_n]\xrightarrow{e}1}{[n:s_p;(s_1\cdots s_k);1;T_n]\xrightarrow[\phi]{c}_0[n:s_p;(s_1\cdots s_k);1;T_n]}$$

$$O'=O,s'_1\to S_1$$

BAS 规则定义了基本状态的不变语义转移，即对于任何激发事件，基本状态不发生改变，且不产生任何动作，标记 0 代表基本状态不发生变化。OR－1 规则定义了或状态的一种前进语义转移，即或状态所含的某个变迁 $(\hat{t},l,sr,e,g,a,j,td,h)\in T_n$ 在事件 e 触发下，满足激发条件 $g=true,sr\subseteq Conf(s_1)$，从而导致或状态发生状态迁移，产生一序列动作 $O=exit\triangleright a\triangleright entry$，并更新到目标状态 $[n:s_p;(s_1\cdots s_k);j;T_n]$，$\text{Next}(h,td,s_j)\mapsto s_j$。OR－2 规则定义了或状态的一种前进传递语义转移。即或状态所含的某个子状态发生了语义转移，从而传递导致或状态发生语义转移。OR－3 规则定义了或状态的一种不变传递语义转移。即或状态所含的某个子状态发生了不变语义转移，从而传递导致或状态发生不变语义转移。或状态将不发生任何变化，也不执行任何动作。AND 规则定义了与状态的一种组合语义转移。即与状态的语义转移是由其子状态的语义转移按照优先级组合而成。

1.2 DEVS

DEVS[6,7]（Discrete Event system Specification）离散事件系统描述是 B.P. Zeigler 在研究一般系统论的基础上创建的一种离散事件系统仿真理论。它把每个子系统都看作是一个具有独立内部结构、行为和明确 I/O 接口的模型，若干个模型可以通过一定的连接关系组成组合模型，组合模型可以作为更大的组合模型的元素使用，从而形成对模型的层次、模块化描述。

原子 DEVS 的结构如下：$M=<X,S,Y,\delta_{\text{int}},\delta_{\text{ext}},\lambda,ta>$，$X$：输入集合；$S$：状态集合；$Y$：输出集合；$\delta_{\text{int}}$：$S\to S$，内部转移函数；$\delta_{\text{ext}}$：$QX\to S$，外部转移函数；$Q=\{(s,e)\mid s\in S,0\leqslant e\leqslant ta(s)\}$：全部状态集合；$e$：系统自上次转移后消逝的时间；$\lambda$：$S\to Y$，输出函数；$ta$：$S\to R^+_{0,\infty}$，时间推进函数。组合 DEVS 的结构如下：$N=<X,Y,D,\{M_d\},\{I_d\},\{Z_{id}\},Select>$，$X$：输入集合；$Y$：输出集合；$D$：组合 DEVS 的所有模型组件集合；$\{M_d\}$：$\forall d\in D$，$M_d$ 为经典 DEVS 模型；$\{I_d\}$：$\forall d\in D\cup\{N\}$，I_d 为模型 d 的输入模型集合，$I_d\subseteq D\cup\{N\}$，$d\notin I_d$；$\{Z_{id}\}$：$\forall i\in I_d$，Z_{id} 为模型 i 到模型 d 的输入输出关系映射函数，且有 Z_{id}：$X\to X_d$，if $i=N$，表示组合模型的输入到子模型的输入的映射；Z_{id}：$Y_i\to Y$，if $d=N$，表示子模型的输出到组合模型的输出的映射；Z_{id}：$Y_i\to X_d$，if $d\neq N\wedge i\neq N$，表示子模型的输入到子模型的输出的映射。DEVS 的执行机制参见文献[6,7]。

2 嵌入状态机到 DEVS

2.1 状态的映射和属性引用

状态图嵌入 DEVS，首先需要将状态图的状态映射到 DEVS 的状态，同时状态图可以拥有 DEVS 定义的属性所有权，可以在动作中对 DEVS 属性进行访问或赋值。状态图的状态是离散状态，可以用布尔值属性定义，属性名称即状态名称。定义映射 k：$SA\to\{1,\cdots,n\}$，表示状态图的状态项到其对应的 DEVS 属性的索引映射。DEVS 的状态 s 是一个所有属性值构成的向量 $s=(a_1,\cdots,a_n)$，状态图某个状态 s_i 活跃，可以表示为 $s=(\cdots,a_{k(sj)}=1,\cdots)$，否则表示为 $s=(\cdots,a_{k(sj)}=0,\cdots)$。

2.2 事件的接收和发送

为了嵌入 DEVS，状态图的接收事件分为两大类，分别是外部转移事件和内部转移事件。外部转移

事件由 DEVS 的输入接口接收,表示 DEVS 接收到外部系统发送的事件;内部转移事件由 DEVS 仿真器自动派遣,表明仿真时间发生了推进。对于嵌入 DEVS 的状态图而言,这两类事件都会激发当前状态对应的迁移,如果迁移的守卫条件满足,则激发该迁移。根据激发迁移的事件类型,可以将状态图的迁移分为外部转移和内部转移,进而可以构造 DEVS 的转移函数。因而,状态图的事件集合扩展为 $E = E_{\text{ext}} \cup E_{\text{int}}, e \in E_{\text{ext}}, \sigma \in E_{\text{int}}$。状态图的变迁集合扩展为 $T_n \subseteq NT \times \pi \times 2^N \times E \times I \times G \times A \times 2^N \times \pi \times H$,其中 I 表示 DEVS 的输入接口集合,当状态图由外部转移事件激发时,需要明确事件的输入接口,因为不同的输入接口可以接收同一类型的输入事件。$E \times I$ 即 DEVS 的输入向量集合,当状态图由内部转移事件激发时,不需要考虑接口信息,即 $I = \phi$。状态图的事件发送需要遵循 DEVS 事件交互机制,即通过输入接口发送事件。事件发送动作可以在状态图的执行动作中定义。

2.3　状态图的动作

状态图的动作是状态变迁的附属行为,一般包括离开状态、进入状态、变迁动作等。动作执行将进一步细化当前离散状态下的其它系统状态。状态图在动作执行中,可以访问 DEVS 的属性,并对属性赋值,以细化 DEVS 的状态变化。动作执行需要语义丰富的动作语言描述,如能够进行各种运算,能够完成控制、逻辑语句等。动作执行可以由一些脚本语言完成,同时需要提供一些仿真接口,用于仿真管理。DEVS 的输出函数和时间推进函数是在状态发生变化时执行的,对于嵌入 DEVS 的状态图而言,需要在动作中对其进行定义。

2.4　构造 DEVS 转移函数

嵌入 DEVS 的状态图用于描述 DEVS 的行为逻辑,最终需要将状态图的行为逻辑映射到 DEVS 的行为函数上。DEVS 的输入函数和时间推进函数已经包含在状态图的动作中,因而,只需要将状态图表达的行为逻辑映射到 DEVS 的外部转移函数和内部转移函数即可。这种行为映射需要严格保证状态图的执行语义和因果逻辑关系。DEVS 的转移函数构造算法如下:

```
For each c ∈ TotalConf(root)
    δ(q,e,i) = δ(q,e,i)⊕ ‖ ifcurState = = c then ‖
    ε(s) = ε(s)⊕ ‖ ifcurState = = c then ‖
    Consturct(root)
    δ(q,e,i) = δ(q,e,i)⊕ ‖ break ‖
    ε(s) = ε(s)⊕ ‖ break ‖
endFor
```

```
Construct([n :s_p ;entry ;exit]) =
For each t ∈ Trans(n)
    if(Type(Trigger(t)) ∈ E_ext)
    then δ(q,e,i) = δ(q,e,i)⊕
    ‖ if(Guard(t) = = true)
    ‖ then Exit([n :s_p ;entry ;exit]) ▷ Effect(t)
    ‖ ▷ Entry(Nexit(History(t),TD(t),Target(t))) ▷ break ‖
    else if(Type(Event(t)) ∈ E_int)
    then ε(s) = ε(s)⊕
    ‖ if(Guard(t) = = ture)
    ‖ then Exit([n :s_p ;entry ;exit]) ▷ Effect(t) ▷
    ‖ Entry(Next(History(t),TD(t),Target(t))) ▷ break ‖
endFor
```

循环判断所有可能的状态配置 c,构造外部转移函数和内部转移函数,将外部转移函数和内部转移函数并入伪代码 ‖ if curState = = c then ‖,$\oplus$表示并入运算符,‖ *code* ‖ 表示 code 是映射到转移函数的目标执行代码。取出当前状态配置的根状态,执行构造函数 Construct(root),并入伪代码 ‖ *break* ‖。上述算法的思想:DEVS 事件发生将触发外部转移函数或内部转移函数,而任意时刻 DEVS 只能处于其嵌入的状态图的某个状态配置,根据当前状态配置判断可以激发的迁移,并执行相应动作,完成转移函数。因为采用 STATEMATE 的执行语义,对于可能冲突的迁移,上层状态优先于其子状态,所以从根状态开始构造转移函数。构造函数 Construct(root)是一个递归算法,将从根状态开始向下递归判断所有当前状态配置状态对应的变迁,生成 DEVS 的转移函数。下面给出构造 DEVS 转移函数 Construct()的算法。

Construct([n :s_p ;(s_1,s_2,…,s_k) ;l ;T_n ;*entry* ;*exit*]) =

For each $t \in$ Trans(n)
  if (Type(Trigger(t)) $\in E_{ext}$)
  then $\delta(q,e,i) = \delta(q,e,i) \oplus$
    if (Guard(t) = *true*)
    then Exit([n :s_p ;(s_1,s_2,…,s_k) ;l ;T_n ;*entry* ;*exit*]) ▷ Effect(t)
    ▷ Entry(Nexit(History(t),TD(t),Target(t))) ▷ break
  else if (Type(Event(t)) $\in E_{int}$)
  then $\varepsilon(s) = \varepsilon(s) \oplus$
    if (Guard(t) = *true*)
    then Exit([n :s_p ;(s_1,s_2,…,s_k) ;l ;T_n ;*entry* ;*exit*]) ▷ Effect(t) ▷ Effect(t)
    ▷ Entry(Next(History(t),TD(t),Target(t))) ▷ break
end For
Construct(s_1)

Construct([n :s_p ;(s_1,s_2,…,s_k) ;*entry* ;*exit*]) =

For each $t \in$ Trans(n)
  if (Type(Trigger(t)) $\in E_{ext}$)
  then δ(q,e,i) = δ(q,e,i) ⊕
    if (Guard(t) = = true)
    then Exit([n :s_p ;(s_1,s_2,…,s_k) ;*entry* ;*exit*]) ▷ Effect(t)
    ▷ Entry(Next(History(t),TD(t),Target(t))) ▷ break
  else if (TYPE(Event(t)) $\in E_{int}$)
  then ε(s) = ε(s) ⊕
    if (Guard(t) = = *true*)
    then Exit([n :s_p ;(s_1,s_2,…,s_k) ;*entry* ;*exit*]) ▷ Effect(t)
    ▷ Entry(Next(History(t),TD(t),Targer(t))) ▷ break
end For
∃ *bijectionb* : {1,…,k} → {1,…,k}
For all $i \in$ {1,…,k}
  Construct($S_{b(i)}$)
end For

Construct([n :s_p ;*entry* ;*exit*])算法含义 如果当前状态为基本状态,则确定该状态项作为源状态所关联的变迁集合,从中任意取出一个变迁(任意时刻只能发生一个变迁是基本假设,否则不合法),如果是外部事件激发,则构造外部转移函数,将守卫条件判断和执行动作伪代码并入外部转移函数 如果是内部事件激发,则构造内部转移函数。

Construct([n :s_p ;(s_1,s_2,…,s_k) ;l ;T_n ;*entry* ;*exit*])算法含义 如果当前状态为或状态,则确定该状态项作为源状态所关联的变迁集合,从中任意取出一个变迁,构造转移函数,过程与基本状态相同。然后,递归调用当前活跃子状态的 Construct()算法。

Construct([n :s_p ;(s_1,s_2,…,s_k) ;*entry* ;*exit*])算法含义 如果当前状态为与状态,则确定该状态项作为源状态所关联的变迁集合,从中任意取出一个变迁,构造转移函数,过程与基本状态相同。然后,根据优先级,递归调用所有子状态的 Construct()算法。

3 应用实例

以一个自动加油系统为例,说明嵌入状态图的 DEVS 的建模能力。自动加油系统分为控制部分、传送机、加油机和油罐。可以将该系统设计为控制系统、感应系统、传送系统和加油系统。这些分系统可以用 DEVS 进行描述,并组成系统模型,其组成关系如图 2 所示。控制系统与其它系统间交互信号包

括 感应系统发送给控制系统的传送带的位置信息 location∈R 和油罐的油量信息 level∈R,控制系统发送给传送系统的控制命令 com1∈ {start,stop},以及控制系统发送给加油系统的控制命令 com2∈ {open, close}。

控制系统的设计可以由嵌入状态图的 DEVS 描述,如图 3 所示。控制系统的静态结构如属性定义、输入输出接口定义,按照 DEVS 规范进行描述,而控制系统的行为逻辑则由状态图进行描述。控制系统的根状态是一个与状态,包括 updateLocation、updateLevel 和 control 三个正交的子状态。updateLocation 状态对应的变迁是一个循环变迁,用于接收 location 端口输入的传送带位置信息;同样,updateLevel 状态用于接收 level 端口输入的油罐油量信息;control 状态负责根据接收的传感系统信息,向传送系统和加油系统发送指令。如果 location 未到指定位置,则处于 transmitting 状态,并更新时间推进函数,到下一时刻,内部事件将触发 transmitting 状态对应的迁移,判断迁移条件 location 是否到达指定位置,如果是,则激发迁移到状态 wait0,并向输出端口 com1 发出 com1.stop 信号,更新时间推进函数;否则激发循环迁移,transmitting 状态不变,更新时间推进函数,直到条件满足。处于 wait0 状态,且内部事件发生时,状态将转移到 filling,并向输出端口 com2 发出 com2.open 信号,更新时间推进函数;依此类推,控制系统将按照状态图的逻辑保证输出正确的控制信号,实现自动加油系统的功能。

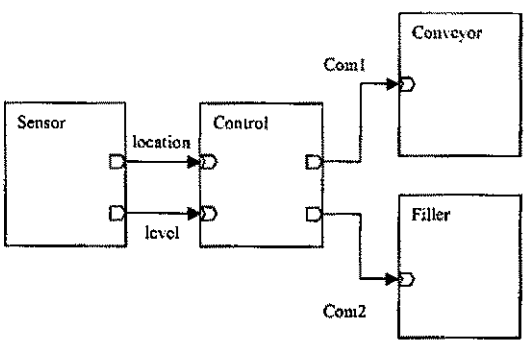

图 2　自动加油系统模型构成

Fig.2　The filling system composite model

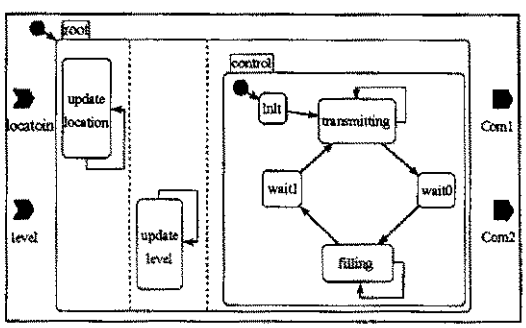

图 3　控制系统模型

Fig.3　The control system model

4　结束语

将状态图嵌入 DEVS 是利用状态图灵活的行为建模能力补充 DEVS 对于行为描述的不足,同时利用 DEVS 描述仿真模型结构和仿真运行逻辑的强大能力补充状态图相应的缺陷。两者的结合使得复杂控制系统的建模仿真更加简捷、直观,通过仿真验证可以有效地提高复杂控制系统的设计效率,完成高层建模到底层代码实现的自动映射。

参 考 文 献:

[1] Giambiasi N, Paillet J L. From Timed Automa to DEVS Models[A]. Proceedings of the 2003 Winter Simulation Conference, 2003.

[2] Beeck M V D. A Structured Operational Semantics for UML-statecharts[J]. Journal of Software System Model, 2002.

[3] Luttgen G, Beeck V D, Cleaveand R. Statechartws via Process Algebra[J]. Lecture Notes in Computer Science, 1664, 1999.

[4] Luttgen G, Beeck V D, Cleaveand R A Compositional Approach to Statecharts Semantics[A]. In Proc of ACM Sigsoft Eighth Int. Symp. on the Foundations of Software Enginnering[C], ACM,2000.

[5] Plotkin G. A Structural Approach to Operational Semantics[R]. Technical Reports, 1981.

[6] Zeigler B P. Theory of System Modeling and Simulation[M]. New York: Academic Press, 2000.

[7] Ki Jung Hong, Tag Gon Kim, Devsif: Relational Algebraic Devs Intermediate Format[A]. In: Sup Kwon,2002.