一种面向嵌入式实时软件的需求建模语言*)

舒风笛[1] 毋国庆[2]

(中国科学院软件研究所 北京100080)[1] (武汉大学计算机学院 武汉430072)[2]

摘 要 针对嵌入式实时系统复杂动态交互行为和严格实时的领域特征,提出了一种软件需求规约语言RTRSM*。该语言以扩充的层次并发有穷状态机HCA为核心,以支持合成的模板为基本组成单元,利用转换有效期和事件预定机制来描述时间限制,既具有较强的时间限制描述能力,又能自然而直接地支持交互行为的建模,可执行且具有良好的形式语义。给出了该语言的形式化语法,举例说明了其时间描述机制,并通过执行步算法和基于HCA项的结构化操作规则定义了该语言的形式化操作语义。

关键词 需求规约语言,层次并发有穷状态机,操作语义

A Requirements Modeling Language of Embedded Real-Time Software

SHU Feng-Di[1] WU Guo-Qing[2]

(Institute of Software, The Chinese Academy of Sciences, Beijing 100080)[1] (College of Computer, Wuhan University, Wuhan 430072)[2]

Abstract Aiming at the complicated dynamic interactive behaviors and rigid timing constraints of embedded real-time system, the paper presents a kind of software requirements specification language RTRSM*. It takes HCA, a kind of extended hierarchical and concurrent finite state machine as the kernel, and its basic unit is template, which supports composition. The language makes use of transitions with duration and events that can be scheduled and canceled to describe timing constraints. Therefore, it provides not only fairy powerful facilities for the description of timing constraints but natural and direct schema for the modeling of interactivity. It has formal semantics as well as executability. After introducing the syntax of RTRSM*, the paper illustrates how to describe timing constraints with this language. Subsequently, the paper defines its formal operational semantics in the form of execution step algorithm and structural operational rules based on HCA items.

Keywords Embedded real-time software, Requirements specification language, Hierarchical and concurrent finite state machine, Operational semantics

形式化需求规约语言有助于准确而无二义性地理解和描述目标系统,支持规约的严格形式化验证,但应又具有一定表达能力,能自然地对现实世界实体进行建模,支持领域特征描述。许多研究将形式语义和图形化方法相结合,得到易用性相对较强的可视化的形式化方法(Visual formalism)[1]。嵌入式实时软件(Embedded Real-time Software, ERS)作为嵌入式实时系统的软件部分,其需求建模语言应能很好地支持事件驱动、复杂动态交互行为和严格时间限制等领域特征的建模。操作性规约语言便于描述外部刺激、系统动作和状态等建模实体。现有与ERS的需求规约相关的基于状态转换的代表性工作各有其侧重点:Harel提出的Statecharts[1]面向复杂、交互式系统,为高度结构化和简洁的图形化描述语言,通过引入状态的层次、并发和广播机制对传统的状态转换图进行了扩充,具有良好的形式化语义并可执行,但Statecharts的时间限制描述能力较弱。为Statecharts转换加入上下时限所得到的timed Statecharts[2]虽然时间限制描述能力得到加强,但形式过于复杂。Modechart[3]强调的是控制和实时的描述,无转换操作机制,对数据计算支持不够。实时UML[4]是在基本UML基础上引入了起源于ROOM(Real-time Object Oriented Modeling)[5]的便于设计复杂嵌入式系统的结构,其描述单一对象行为的状态图与传统Statecharts无本质区别,主要通过交互图和活动图进行时间限制需求建模。此外,虽已有一些UML形式化语义和验证研究[6~8],但仍不成熟,尤其是时间限制方面。时间自动机(Timed Automata, TA)[9]和时间转换系统(Timed Transition System, TTS)[10]为较为成熟的实时模型,分别通过时钟变量和转换的上下时限来支持时间限制的描述,但两模型对反应性的建模不够自然和直接,且分析机制较为复杂。

我们曾提出的面向ERS的需求建模语言HRFSM[11]继承了Statecharts的层次并发状态和广播通讯机制,以模板为基本描述单位,以规则为状态图的形式化表示,能较好支持系统动态交互行为和数据计算的描述及处理。本文在其基础上,主要通过增加转换有效期、完善事件预定机制,得到新的具有更强时间限制描述能力并能同样直接自然支持交互行为建模的需求建模语言RTRSM*,它结构化、图形化、简单、无二义性、可合成,具有良好的形式化语义,可执行、可验证。本文首先介绍RTRSM*的语法,尤其是其核心,层次并发有穷状态机模型HCA,然后举例说明其时间限制描述机制,最后给出该语言的形式化操作语义。

1 需求规约语言RTRSM*

RTRSM*可看作是Statecharts的层次并发状态、HRF-

*)Supported by the National Natural Science Foundation of China under Grant No. 69873035, 60273026 (国家自然科学基金); the K. C. Wong Education Foundation, Hong Kong (王宽诚教育基金)。舒风笛 博士后,主要研究领域为需求工程,形式化理论和方法;毋国庆 教授,博士生导师,主要研究领域为软件理论,需求工程。

SM 的规则和模板[11]及其自身转换有效期和事件预定机制的结合。RTRSM* 通过模板来支持系统的分解，一个模板对应一个状态机，其具体表现形式为包括接口和数据定义的表格、状态转换图和与状态图等价的形式化的规则集。模板间通过事件和共享数据进行通讯。

ERS 做为反应式系统，目的是使计算机有效控制相应部件，而绝大多数计算机控制是离散的；且离散时间域便于分析，因此，我们将 RTRSM* 定义在离散时间域。在实践应用中则要求合理选择时间片。

1.1 层次并发有穷状态机 HCA

定义1 一个层次并发有穷状态机 HCA 为一7元组：$(S, T, E, C, A, V, Root)$，其中，S：状态集合，T：转换集合，E：事件集合，C：卫士条件集合，A：操作集合，V：HCA 中所定义和使用的变量和事件属性集合，$Root$：初始状态。S 继承了 Statecharts 的层次和并发状态[1]，即状态包括基本状态和实际为状态机的超状态，而后者又分为与状态和或状态。其中，与状态活跃，当且仅当它所有子状态都活跃；或状态活跃，当且仅当有且只有唯一子状态活跃。S 中的转换构成一树型结构，$Root$ 为根，每一节点的后代为其直接子状态，不存在祖先关系的状态称为正交状态。

定义2 HCA 中的每条转换对应一条规则，形式为：

源状态，触发事件(事件属性)=>目的状态，WHEN 卫士条件，DO 操作 D

其中，非负整数 D 为转换有效期，缺省为0。此规则含义为：当源状态活跃、触发事件发生且卫士条件为真时，该转换可执行，而在成为可执行后 D 时间单元内(D 为0则立即执行)，该转换只要有效，可且必定实际执行，其有效定义为：源状态活跃且卫士条件满足。转换的实际执行过程是：退出源状态，执行操作，然后转到目的状态，具有原子性，瞬间的。转换 t 的转换名、源状态、触发事件、卫士条件、目的状态、操作和有效期分别用 $name(t)$，$source(t)$，$event(t)$，$cod(t)$，$target(t)$，$action(t)$，$duration(t)$表示。

不允许跨层转换，即源状态和目的状态必须为兄弟。默认转换的源状态为空，目的状态非空，D 为0，卫士条件缺省为真，无触发事件，一旦进入该转换所在超状态，则立即执行。对于其它转换：源状态、触发事件和目的状态非空，其它项可选。

定义3(转换的一致性) 两个转换是一致的，当它们是相同的，或其源状态正交且操作无变量写冲突，否则称它们冲突，不一致。对于任一转换集合 T，若其任两个转换都一致，则称 T 一致。

事件标记了对系统行为描述而言具有意义的时间点，描述了外部环境或系统某种瞬间变化，分别称为外部和内部事件。系统只能被动接受前者；后者则主要用于系统内部状态机的通讯或起定时器的作用。事件用唯一的事件名来标识，可带属性，其属性可当作输入数据变量来处理，一可将类似事件抽象为属性不同的相同事件，二可增强数据处理功能。

卫士条件标明转换执行的条件，是关于数据变量或事件属性的取值或状态机当前状态的一个条件表达式，可包含算术、关系和逻辑运算，以及判断某状态是否活跃的特殊运算：$In.In(s)$为真，当状态 s 活跃，否则为假。卫士条件为真，当该表达式非0。

定义4(HCA 转换表达式) HCA 的转换操作表达式集合 A 递归定义如下：

$A ::= Action \mid Action; A$

$Action ::= v=exp \mid SetEvent(e,n) \mid CancelEvent(e) \mid SendNotify(e) \mid SendControl(Comm)$

其中，exp 为算术表达式，可包含所定义的数据变量和事件属性，v 为所定义的可写数据变量，e 为内部事件，n 为正整数，有：

$SetEvent(e,n)$：定制事件 e，在该语句执行 n 个时间单元后，事件 e 产生并被广播；

$SendNotify(e)$：生成并广播事件 e；

$SendControl(Comm)$：处理同 $SendNotify$，表示控制软件向外部环境发送指令 $Comm$[12]。

$CancelEvent(e)$：取消预定的事件 e。

1.2 HCA 模型及其合成

定义5(HCA 模型) 一个 HCA 模型为一四元组(V, D, θ, T)。其中，V：变量集合，包括：

1) 模板中所定义的数据变量，包括接口变量集合 I 和私有变量集合 pr，I 又包括系统输入(由外部环境决定，系统只能读)、输出(由系统决定，发送给外部环境)和共享变量(外部环境和系统都可读写)；

2) 时钟变量 t，有：$type(t)=\overline{Z^-}$，其中 $type(v)$为变量 v 的类型；

3) 为状态机中每个状态和事件设置一对应的布尔变量，变量名为该状态名或事件名，称为状态变量、事件变量，为真则表示该状态活跃或该事件发生；

4) 记录列表 $FutureEvent$，其元素形式为(e,n)，e 为事件名，n 为自然数。

D：所有变量类型的并；θ：必须满足的初始条件；T：转换集合，包括特殊的时钟转换 $tick$。

定义6 HCA 模型 M 的系统状态为一映射 $gs: V \to D$，使得对于每个变量 $v \in V$，有 $gs(v) \in type(v)$，由此构成一模型的系统状态空间集合 GS。若 p 为 M 中变量的表达式，则 $gs(p)$表示 p 在 gs 中的取值。

定义7(转换关系) 如果存在一个转换 τ，其源系统状态为 gs_1，目的系统状态 gs_2，则定义 gs_1，gs_2 满足转换关系 ρ：$\rho(gs_1, gs_2)$，由 RTRSM* 的确定性，该转换唯一。

定义8 一模板 M 对应一 HCA 模型 hca，可表示为一三元组(I, hca, P)，分别表示模板的接口定义(记为 $I(M)$)、hca 和必须满足的性质。

定义9(HCA 的合成) 对于 $hca_1=(V_1, D_1, \theta_1, T_1)$，$hca_2=(V_2, D_2, \theta_2, T_2)$，它们并发合成和顺序合成分别得到对应于状态和或状态的 $hca = hca_1 \| hca_2 = (V, D, \theta, T)$ 和 $hca' = hca_1 \circ hca_2 = (V', D', \theta', T')$，其中顺序合成时先进入 hca_1，$\tau_1, \cdots, \tau'_m$ 为连接它们的转换，有：$V=V'=V_1 \cup V_2$；$D=D'=D_1 \cup D_2$；$\theta=\theta_1 \wedge \theta_2$；$T=T_1 \cup T_2$；$\theta'=\theta_1$；$T'=T_1 \cup T_2 \cup \{\tau'_j\}$，$j \in [1,m]$。

定义10 两个模板 $M_1=(I_1, hca_1, P_1)$，$M_2=(I_2, hca_2, P_2)$，其中 $hca_1=(V_1, D_1, \theta_1, T_1)$，$hca_2=(V_2, D_2, \theta_2, T_2)$是并发相容的，当：对于每个 $v \in I(M_1) \cup I(M_2)$在两模板中的类型匹配，且 $\theta_1 \wedge \theta_2$和 $P_1 \wedge P_2$可满足。

定义11 两模板 $M_1=(I_1, hca_1, P_1)$，$M_2=(I_2, hca_2, P_2)$的顺序合成体 $M=M_1 \circ M_2=(I, hca, P)$，其中 $\tau_1, \tau_2, \cdots, \tau_m$ 为从 hca_1到 hca_2的转换，$\tau'_1, \tau'_2, \cdots, \tau'_n$ 为从 hca_2到 hca_1的转换(HCA 中不允许出现跨层转换)，有 $I \subseteq I_1 \cup I_2$，$hca=hca_1 \circ hca_2$，$P = \bigwedge_{1 \le i \le m}(P_1 \wedge cond(\tau_i) \wedge event(\tau_i) \to P_2) \wedge \bigwedge_{1 \le j \le n}(P_2 \wedge cond(\tau'_j) \wedge event(\tau'_j) \to P_1) \wedge (P_1 \vee P_2)$。

定义12 两并发相容的模板 $M_1=(I_1,hca_1,P_1)$，$M_2=(I_2,hca_2,P_2)$的并发合成体 $M=M_1\|M_2=(I,hca,P)$，有：$I\subseteq I_1\cup I_2$，$hca=hca_1\|hca_2$，$P=P_1\wedge P_2$。

2 RTRSM*的时间限制描述机制

嵌入式实时系统的时间限制反应在对外部刺激(包括时间流逝)及系统反应上，基于RTRSM*，前者可描述为能触发状态转换的事件，后者为相应由当前系统状态所决定的转换的实际执行，由于其瞬间性，即为其目的状态的进入。RTRSM*采用隐含时间信息，利用事件及其预定机制和转换有效期来描述。Statecharts的理想的同步假设[1](系统总比外部环境动作更快，在收到刺激后马上执行)能简化系统需求的分析，但一方面该假设并不总能被满足；另一方面，不便于充分完全描述实际中常见的以时间段形式给出的时间需求。RTRSM*转换带有有效期，能有效克服以上两点。本节首先通过对经典的铁轨交叉路口系统(Railroad Crossing System，RCS)[13]的特例：标准RCS稍加修改，以说明如何用RTRSM*来描述时间段形式的时间，然后将时间限制分为并非互斥的最大/小时间间隔和持续时间3类[14]，通过一个电话的例子[14]来系统说明RTRSM*的时间性质描述机制。

2.1 时间段形式的时间需求描述

用软件来控制一通常打开的铁路交叉路口的控制门。由探测器1检测火车的临近：当火车经过探测器1，该探测器发送通知near；火车在经过该探测器后，至少300个时间单元后才到达该路口。探测器2检测火车是否通过该路口，若通过则发送通知passed。火车通过路口与经过探测器1最多相隔500个时间单元。控制器在接收到near消息后100个时间单元时向门发送关闭的控制命令lower，并在接收到passed消息后100个时间单元时向门发送打开的控制命令raise。门的关闭需20～50个时间单元才能完成，其上升需要100～200个时间单元。根据火车调度方案，从一火车离开交叉路口到下一火车到达探测器1至少间隔100个时间单元。

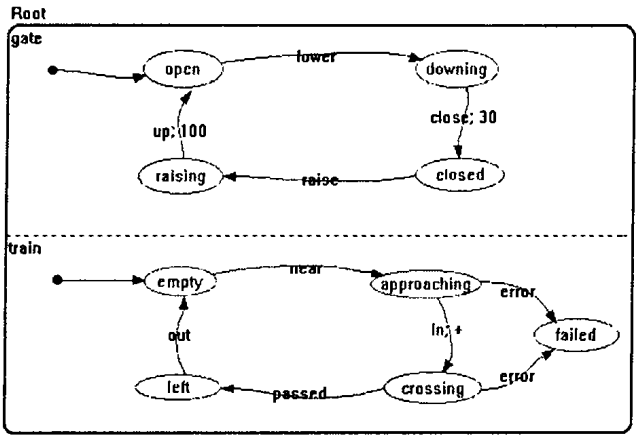

图1 标准铁轨交叉路口系统的状态转换图

```
模板 Root 的规则集为：
  Root_=>gate‖train
模板 gate 的规则集为：
  gate_=>gate.open 这是一条默认状态转换
  open,lower=>gate.downing Action SetEvent(close,20)
  downing,close=>gate,closed;30
  raising,up=>gate.open;100
  closed,raise=>gate.raising Action SetEvent(up,100)
模板 train 的规则集为：
  train,_=>train.empty 这是一条默认状态转换
  empty,near=?train.approaching Action SetEvent(lower,100);‖
      SetEvent(in,300);‖SetEvent(error,500)
  approaching,in=>train.crossing;+
  approaching,error=>train.failed
  left,out=>train.empty
  crossing,error=>train.failed
  crossing,passed=>train.left Action SetEvent(ot,100);‖SetEvent
      (raise,100);
```

图2 标准铁轨交叉路口系统的规则集

在该例中，门的向下关闭和向上打开都是持续动作，作为状态来描述，其时间需求以常见的时间段形式给出。若要求转换一旦满足则立即执行，虽然也能满足时间要求，在限制范围内完成门的关闭和打开，但没有完全地表达出用户的需求，从而可能隐藏需求中的错误，例如关于门关闭和打开的时间需求与火车调度方案不一致。转换有效期的引入则能较好地描述此类需求，从而支持通过相应的规约检查，发现需求中可能的错误。图1给出了用支持RTRSM*的面向嵌入式实时系统的需求工程环境SREE[12]所编辑得到的状态图。图中只显示了触发事件，其中"up;100"中的up为事件名，100即为有效期，"+"表示有效期是正无穷大。图2为由SREE自动生成的对应的规则集，其中，第一条规则表示的是状态Root的进入导致其并发子状态gate和train的进入，对应层次关系的分解规则[12]。规则的目的状态，如gate.open表示的是模板gate中的状态open，对于同一转换的操作中的多个表达式用"‖"相隔。限于篇幅，该系统相应的模板信息省略。

2.2 最大时间间隔

设时间间隔限制为 n，所涉及的前后刺激事件分别为 S_1，S_2，所对应的转换和所要进入的目的状态分别为 T_1、T_2，R_1、R_2(下同)。最大时间限制的描述主要通过设置新超时事件，超时则进入相应处理状态。有4种情况：

A. *S-S* 组合：前后两事件发生的最大时间间隔；

B. *S-R* 组合：某事件的产生和其所引发的系统反应间的最大时间间隔；如：听筒被提起后1秒内应发出待拨号声音；

C. *R-S* 组合：系统执行某一反应与其后接收到另一刺激事件的最大时间间隔；如：听筒发出待拨号声音后用户必须在10秒内按键。

D. *R-R* 组合：系统前后两次反应的最大时间间隔；如：连接建立后，呼叫方听筒中传出响铃声后1秒内被呼叫方电话铃响。

1) 限制A：包括事件预定和取消两步，且根据 T_1和 T_2有效期是否为0分别处理：

i) 预定事件：

· $duration(T_1)=0$：T_1的执行时间即 S_1的发生时间。由转换执行的瞬间性，在 T_1的操作中预定新事件 e_1：$SetEvent(e_1,n)$；增加由 e_1所触发的 R_1的迁出转换 T'，进行超时处理，其有效期为0，无卫士条件，使得若 S_1发生 n 时间内 S_2未发生，则进入超时处理。

· $duration(T_1)\neq 0$：考虑到 S_1发生后 T_1不一定立即执行，故用新转换 t_1和 t_2替换转换 T_1，以获取 S_1实际发生时间；同样增加由 e_1的发生将立即导致执行的 R_1的迁出转换 T'进行超时处理，其中，e_1，e_2为新事件，R_1'为新状态，t_1的源状态、触发事件和卫士条件都与 T_1的相同，其它有：

$target(t_1)=R_1'$，$action(t_1)=\{SetEvent(e_1,n),SendNotify(e_2)\}$，$duration\ (t_1)=0$；

$source(t_2)=R_1'$，$event(t_2)=e_2$，$cond(t_2)=true$，$target(t_2)=R_1$，$action(t_2)=action(T_1)$，$duration(t_2)=duration(T_1)$

ii) 取消预定事件：防止系统下一次进入 R_1时，上次所设

置的事件 e_1 还未发生，从而破坏本次执行。

· $duration(T_2)=0$：在 T_2 的操作中取消预定事件 e_1；

· $duration(T_2)\neq 0$：采用与 i）中类似的方法，将 T_2 替换为新转换 t_1' 和 t_2'，其中 t_2' 的卫士条件、目的状态、操作和有效期都与 T_2 的相同，其它有：

$source(t_1')=source(T_2)=R_1$，$event(t_1')=event(T_2)=S_2$，$cond(t_1')=true$，$target(t_1')=R_2'$，$action(t_1')=\{CancelEvent(e_1),SendNotify(e_3)\}$；$duration(t_1')=0$，$source(t_2')=R_2'$，$event(t_2')=e_3$

2）限制 B：T_1 的有效期为 n，其它不变。

3）限制 C：与 A 中有效期为0的处理方法相同。

4）限制 D：是关于反应的限制，故不必考虑转换的有效期。在 T_1 的操作中加入 $SetEvent(e_1,n)$，在 T_2 的操作中加入 $CancelEvent(e_1)$，增加以 $source(T_2)$ 为源状态，e_1 为触发事件的无条件、零有效期的超时处理转换。

对于多个相邻事件或反应的最大时间限制，只需对 A 进行扩充：为相应路径上的所有中间状态增加触发事件为 e_1 的无卫士条件转换，由其进入相应超时处理，即该路径必须在指定时间内走完，否则，预定事件引发超时处理。

2.3 最小时间间隔

最小时间限制则是通过加入中间状态来强制系统停滞。相对于最大时间间隔限制，只需增加中间状态 A，不需取消预定事件。因为可执行转换的最早执行时间（相对于成为可执行那一刻而言）是0，所以此处不用考虑转换有效期。对于限制 A，同(1)的限制 A 中有效期为0时的事件预定操作。对于限制 B，增加 R_1'，在过 n 个时间单元后才能进入所对应转换的目的状态 R_1，将 S_1 所应引发的操作都放在从 R_1' 到 R_1 的转换的操作中，其卫士条件保留在由 S_1 引发的进入 R_1' 的转换中。对于限制 C 和 D，则在进入 R_1 后，先等待 n 时间单元后再进入可接受 S_2 的状态 R'。

对于多个刺激和多个反应间的最小化时间限制可以很简单地予以描述，如只需前2个刺激（反应）至少相隔指定的时间间隔[14]，但这样虽满足需求，但并不充分。另一种复杂但充分的方法是：①增加一初始为假的内部变量作为标志位；②在第一个相关转换的操作中预定指定间隔后发生的新事件 e，为相关路径上所有中间状态增加新事件 e 触发的无条件转换，其目的状态为源状态本身，操作是将标志位置真；③对于相关路径的最后一个状态：为其进入转换增加卫士条件，判断标志位是否为真，而对于为假的情况，则增加相应转换进行违反限制的处理，此外，再考虑到多事件可同时发生的情况，应再加上这些进入转换触发事件和 e 同时发生的情况，此时则不用判断卫士条件。虽然可能存在这些进入转换可延迟的情况，由于 RTRSM* 转换优先级定义（见3.1.2节），会先执行标志位置真的转换，而这些进入转换仍有效（卫士条件满足，源状态活跃），仍将执行且满足相应时间限制。对于以上由预定事件所触发的转换的有效期为0。

2.4 持续时间限制

对于最小持续时间限制，增加中间状态和转换，在源状态停留指定时间后才能通过预定事件的发生进入能响应刺激的状态。对于最大持续时间则是预定事件以触发相应处理。

3 RTRSM* 的操作语义

定义13（HCA 的初始轨迹） $\sigma=gs_0\lambda_1gs_1\lambda_2gs_2\cdots gs_i\in GS,i\in[0,+\infty)$，$\lambda_i$ 为连接 gs_{i-1} 和 gs_i 的全局转换（可能包括多个一致的正交转换），有：

· 初始化：$\lambda_0=initial$ 且除此外，$initial$ 不再发生，它作为系统的启动，进入稳定的系统初始状态 gs_1，其转换函数的作用包括系统根状态及相应默认状态的进入；

· 连续性：对于每个 i，$\rho(gs_i,gs_{i+1})$；

· 时间上限：对于任意的 $gs_i(e_\tau)=$true 的 $\tau\in T$，若 $u_\tau=0$ 且不存在这样的转换 τ'：$gs_i(e_{\tau'})=$true 且 τ' 优先级高于 τ，则必有 $\tau\in\lambda_i$；否则，必存在一 $j>i$，有 $(gs_j(c_\tau)=\text{false}\vee\tau\in\lambda_j)\wedge(gs_j(t)\leqslant gs_i(t)+u_\tau)$；

· 时钟滴答（前进）：σ 中有无穷多个 $\lambda_{>0}=tick$（时钟转换）。

定义14 一个 HCA 模型 hca 的合法轨迹集合 Σ_{hca} 由其所有初始轨迹及其后缀组成。

RTRSM* 所描述的系统行为是一组可能的由其环境所产生的外部刺激（包括时间的流逝）所导致的反应序列，即对应的 Σ_{hca}。没有引发状态迁移的事件发生的系统状态称为稳定状态，从一稳定状态出发，直至进入下一稳定系统状态的全局转换执行序列称为一宏步，每一全局转换的执行称为一微步[1]，规定每一微步的执行效果只在下一微步有效。事件的"真"只维持一微步。宏步的开始用显式全局时钟滴答来标识，所引发的系统状态迁移即为时钟转换 $tick$。下面首先用执行步算法定义了一宏步的执行过程。

3.1 执行步语义

3.1.1 执行步算法

（1）输入：

· 当前构图 C（各状态机当前状态所组成的集合），系统内部数据项的取值；

· 从上一步后外部环境所发生的变化，包括发生的外部事件集合 $Event$，外部数据项的修改等；

· 由该步之前执行的操作所预定的事件列表 $FutureEvent=\{(e,n)\mid e$：事件名，$n$：自然数$\}$；

· 该步以前即可执行，但尚未执行的转换列表 DELAY，其项 $item=(t,lefttime)$，其中 t 为转换名，$lefttime$ 为自然数，分别表示为 $item[1]$，$item[2]$；

（2）输出：新的系统状态；

（3）算法：

A 准备：

对于 FutureEvent 中的每一项 (e,n) 执行以下操作：

```
IF 1≥n THEN { 生成该(内部)事件;Event=Event∪{e};从列
   表中删除该项 } ELSE n=n-1;
DELAY=DELAY-{item | not(source(item[1])∈C∧cond
   (item[1]))}//删除不再有效的转换
对于 DELAY 中的每一项(t,lefttime):lefttime=lefttime-1;
flag=true;
```

B 计算将要执行的转换：

```
Exs= TO_EXE= EN1=EN2=∅;
EN1={t| event(t) ∈Event∧source(t)∈C∧cond(t)};
Must={t| t∈EN1∧duration=0};
IF flag=true //第一微步相对其它微步,需多处理被延迟的转换
THEN{EXs={item[1]| source(item[1])∈C∧cond(item[1])
     ∧item∈DELAY∧item[2]=0}∪{从 DELAY 中第二
     项非0的记录中选取将立即执行的转换,该选取或由系统
     具体运行情况决定,或无关紧要};
     DELAY=DELAY- EXs; }
EXs = EXs∪Must;
EN2={从EN1中有效期非0的转换中选取将立即执行的转换,该
     选取或由系统具体运行情况决定,或无关紧要};
EXs=EXs∪EN2; DELAY=DELAY∪(EN1-Must-EN2);
利用转换优先级消除 EXs 中冲突的转换,得到一致的同步转换
   集合 TO_EXE;
IF TO_EXE ≠∅ THEN 执行 C;ELSE 该执行步结束;
```

C 转换执行

Event =∅; flag=false; Sx={source(t)|t∈TO_EXE}; S_n={target(t)|t∈TO_EXE};
从 C 中删除 Sx;
CASE action(t)
{ 赋值语句:所有读变量都取该步执行前的值;写变量的新值从下步开始有效;若一转换操作中对同一变量有多次赋值,则以最后一次为准;
SetEvent(e,n):将(e,n)加入 FutureEvent;
CancelEvent(e):删除 FutureEvent 中第一个元素为 e 的项(可能为多项);
SendNotify(e):Event=Event∪{e} }
将 S_n 加入 C;
IF Event =∅ THEN 该执行步结束;ELSE 转至 B;

3.1.2 *算法说明* 为保证该算法终止,即微步序列有穷,要求一宏步内不能两次进入同一系统状态。Exs 中可能的转换冲突包括源状态不正交或转换操作中的赋值语句存在写冲突,RTRSM* 通过顺序执行以下优先级原则予以消除:高层转换优先级高;同层转换中,外部事件触发的转换优先级最高,被延迟的转换优先级最低。执行步算法中由此得到的 TO-EXE 若仍不一致,则说明存在 RTRSM* 所不允许的不确定性。允许实际中可能存在的不确定性能使建模更简单、直观,但出于以下考虑,RTRSM* 不允许不确定性。首先,不确定性可能隐藏错误,而如果多种情况都允许,则可将其抽象统一,而这正好体现规约的重要性质:能且正好能区分系统所期望的行为与其它。此外,从分析的角度来看,判断一个确定行为是否可接受,比证明所有可能的不确定行为都可接受要容易。

3.2 结构化合成语义

执行步算法并未说明宏步间如何连接,且是从系统整体角度定义,不是合成的。本节针对以上两问题,首先用支持层次和合成性的项形式重新定义 HCA,然后基于带标号的转换系统(labeled transition system,LTS),用结构化操作规则(structural operational rule,SOR)定义各种情况下项的变化,以说明宏步的开始、结束及组成,给出 RTRSM* 的合成的结构化操作语义(structural operational semantics,SOS)[15]。SOS 风格的语义定义除简单易懂外,也可使以后利用已有形式化分析、验证工具和关于 SOS 的元理论进行语义推导成为可能。SOR 的形式如下,意思是:该规则是可应用的,当前提和边条件都满足,在这种情况下,我们可以得到结论。

$$\text{规则名}\ \frac{\text{前提}}{\text{结论}}\ \text{边条件}$$

图3 SOR 的形式

将时钟的滴答做为一个外部事件,由它引发系统状态迁移,实际上也使异步与同步模型相统一。时钟转换包括两个子过程,首先是得到执行步算法的输入及确定被延迟转换是否实际执行,然后是该算法的"准备"。当算法结束,即一宏步执行完毕,再执行时钟转换,开始下一宏步。被延迟转换的实际执行可当作是由时钟滴答事件所触发,这样,所有转换都是事件触发,满足因果关系,不会出现文[16]中指出的:在步构造算法中,一旦被构造,关于该宏步是如何推演得到的所有信息都被丢失,使其语义不具备合成性。RTRSM* 禁止层间转换,这更易于定义合成语义。

3.2.1 HCA 项及其扩充

定义15(HCA 项) N:可数的 HCA 状态名集合,Γ:可数的转换名集合,$N\cap\Gamma=\emptyset$,转换名和状态名都唯一,Π:可数的事件集合,对每个事件 $event$,对应有 $event$,表示事件 $event$ 不发生,有 $\neg\neg event=_{df} event$,$event\wedge\neg event=_{df} false$,$P$:关于系统数据变量的正命题集合,$\Pi=\Pi\cup P$,$\Sigma\subseteq\Pi\cup\{\neg event\mid event\in\Pi\}\cup P\cup\{(\neg p\mid p\in P\}=\Pi\cup\{(\neg x\mid x\in\Pi\}$,有:$\neg\Pi=\{\neg e\mid e\in\Pi\}$,$\neg\Pi=\{\neg v\mid v\in\Pi\}$。HCA 项递归定义如下:

(1) 基本状态:如果 $n\in N$,则$[n]$是一个 HCA 项;

(2) 或状态:$s=[n:\overset{*}{s};m;T;T']$是一个 HCA 项,其中,$n\in N$,$\overset{*}{s}=_{def}(s_1,s_2,\cdots,s_k)$,$k>0$,$s_1,s_2,\cdots,s_k$ 为 HCA 项且为 s 的直接子状态,s_1为其默认子状态,s_m 为 s 的当前活跃子状态,$\rho=_{def}\{1,\cdots,k\}$,$T\subseteq\Gamma\times\rho\times2^{\Sigma}\times2^{\Pi}\times\rho$,为 s 的所有本层转换(不包括默认转换),即连接的是 s 的直接子状态,$T'=\{(t,lefttime)\mid t\in\Gamma,lefttime$ 为正整数$\}$,用于记录可执行但未执行的转换,t 为转换名,$lefttime$ 为剩余的有效期。

(3) 与状态:$s=[n:\overset{*}{s}]$是一个 HCA 项,其中,$n\in N$,$\overset{*}{s}=_{def}(s_1,s_2,\cdots,s_k)$,$k>0$,$s_1,s_2,\cdots,s_k$ 为 HCA 项且为 s 的直接子状态。

(4) 所有的 HCA 项都由以上规则生成,此外无它。

定义16 对于或状态 $s=[n:\overset{*}{s};m;T;T']$,$\forall\ \bar{t}\in T$,$\bar{t}=(t,i,Tri,A,j,d)$,有:$\bar{t}$ 的名称 $name(\bar{t})=t\in\Gamma$;$\bar{t}$ 的源状态 $source(\bar{t})=_{df}s_i\in\overset{*}{s},i\in\rho$;$\bar{t}$ 的触发条件 $trigger(\bar{t})=_{df}Tri\subseteq\Sigma$,包括触发事件和卫士条件,$trigger^+(\bar{t})=_{df}trigger(\bar{t})\cap\Pi$,$trigger^-(\bar{t})=_{df}trigger(\bar{t})\cap\neg\Pi$;$\bar{t}$ 的操作 $act(\bar{t})=_{df}A\subseteq\Pi$;$\bar{t}$ 的目的状态 $target(\bar{t})=_{df}s_j\in\overset{*}{s},j\in\rho$;$\bar{t}$ 的有效期 $duration(\bar{t})=_{df}d$;将 $\bar{t}$ 换为 t,以上定义仍成立。

下面根据以上定义,有:

(1) 状态集合:函数 $states:HCA\to2^N$,有:

$states([n])=\{n\}$;

$states([n:(s_1,s_2,\cdots,s_k);m;T;T'])=\{n\}\cup\bigcup_{1\leqslant i\leqslant k}states(s_i)$;

$states([n:(s_1,s_2,\cdots,s_k)])=\{n\}\cup\bigcup_{1\leqslant i\leqslant k}states(s_i)$;

(2) 默认状态集合:对于一个项 s,其默认状态集合 $default(s)$递归定义如下:

$s=[n]$,则 $default(s)=\{n\}$;

$s=[n:(s_1,s_2,\cdots,s_k);m;T;T']$,则 $default(s)=\{n\}\cup default(s_m)$;

$s=[n:(s_1,s_2,\cdots,s_k)]$,则 $default(s)=\{n\}\cup\bigcup default(s_i)$,$1\leqslant i\leqslant k$

(3) 转换集合:函数 $trans:HCA\to2^{\Gamma}$,有:

$trans([n])=\emptyset$;

$trans([n:(s_1,s_2,\cdots,s_k);m;T;T'])=\{t\mid(t,i,E,A,j)\in T\}\cup\bigcup_{1\leqslant i\leqslant k}trans(s_i)$;

$trans([n:(s_1,s_2,\cdots,s_k)])=\bigcup_{1\leqslant i\leqslant k}trans(s_i)$;

对于转换 $\bar{t}=(t,i,E,A,j)$,有:$out(\bar{t})=states(s_i)$,$in(\bar{t})=default(s_j)$。

(4) 转换的一致性

对于一个项 s,转换 $t_1,t_2\in trans(s)$正交,记为:$t_1\perp t_2$,当且仅当存在一个项 $s'=[n:(s_1,s_2,\cdots,s_k)]$,$n\in states(s)$,$t_1\in trans(s_i)$,$t_2\in trans(s_j)$,$i\neq j$ 且 $action(t_1)\wedge action(t_2)$可满足。

为区分时钟转换和事件触发转换,我们需对定义11进行扩充。

定义17 扩充的 HCA 项:ex-HCA 项归纳定义如下:

(1) 所有 HCA 项都是 ex-HCA 项;

(2) 对于 HCA 项 $s_1=[n:\overset{*}{s};m;T;T']$和 $s_2=[n:\overset{*}{s}]$,$s_1'=[n::\overset{*}{s};m;T;T']$,$s_1^{\cdot}=[n:::\overset{*}{s};m;T;T']$;$s_2'=[n::\overset{*}{s}]$,$s_2^{\cdot}=[n:::\overset{*}{s}]$也是 HCA 项,其中 s_1',s_2'为由时钟转换得到的项,$s_1^{\cdot}$,$s_2^{\cdot}$则为执行事件触发转换所进入的项。

(3) 除事件触发转换:$\to\subseteq ex\text{-}HCA\times2^{\Pi}\times2^{\neg\Pi}\times ex\text{-}HCA$ 外,增加一类时钟转换:$\xrightarrow{\Delta}\subseteq ex\text{-}HCA\times ex\text{-}HCA$。

(4) 所有的 *ex-HCA* 中的项都由以上规则生成，此外无它。

为区分内、外部事件所触发的转换，用 E 表示外部事件触发的转换的触发条件中的"正条件"，即要求发生的正事件和要求满足的正命题，而 N 为相应的"负条件"；定义12中的 $trigger^+(t)$ 和 $trigger^-(t)$ 则相应定义在内部事件所触发的转换 t 上。为方便起见，我们用 $s \xrightarrow[N]{E} s'$ 来代替 $(s,E,N,s') \in \rightarrow$。有若 $\vec{s}=(s_1,s_2,\cdots,s_m,\cdots,s_k)$，则 $\vec{s}[sm \rightarrow sm']=(s_1,s_2,\cdots,s_{m-1},s_m',s_{m+1},\cdots,s_k)$，$|\vec{s}|=k$。

3.2.2 基于 *ex-HCA* 的 SOR 宏步的开始，即时钟转换，称为第一微步；其后第一微步称为第二微步，有三种可能：为外部事件或到期的预定事件所触发，或执行的是过去成为可执行但未实际执行的转换，而其它微步则只能是由前一微步所生成的内部事件所触发。辅助谓词 *To-exe* 定义在可延迟转换集合上，$To\text{-}exe(t)$ 为真表示 t 实际执行。以下给出部分 SOR：

(1) 时钟转换(第一微步)

$$\text{Clock-bas}:\ \frac{-}{[n]\xrightarrow{\Delta}[n]}$$

$$\text{Clock-or}:\ \frac{-}{[n;\vec{s};m;T;\Phi]\xrightarrow{\Delta}[n;\vec{s}_{[sm\rightarrow default(sm)]};m;T;\Phi]}$$

$$\text{Clock-and}:\ \frac{-}{[n;\vec{s}]\xrightarrow{\Delta}[n;\vec{s}]}$$

$$\text{End-or1}:\ \frac{-}{[n;\vec{s};m;T;T']\xrightarrow{\Delta}[n;\vec{s}_{[sm\rightarrow default(sm)];m};T;T'[(t,leftime-1)/(t,lefttime)]]}$$

$$\text{End-or2}:\ \frac{Sm\xrightarrow{\Delta}Sm'}{[n;\vec{s};m;T;T']\Delta\rightarrow[n;\vec{s}_{[sm\Delta sm']};m;T;T']}[n;\vec{s};m;T;T']\rightarrow[n;\vec{s};m';T;T']$$

$$\text{End-and2}:\ \frac{\forall\ 1\leqslant m\leqslant|\vec{s}|\,Sm\xrightarrow{\Delta}Sm'}{[n;\vec{s}]\xrightarrow{\Delta}[n;\vec{s}[sm\rightarrow sm']]}$$

"Δ"表示时钟转换。规则 End-or1说明的是本层执行的本宏步结束(即下一宏步的开始)，只存在于或状态；End-or2和 End-and2则对应由下层引发的本宏步的结束，执行转换的条件分别是：当前子状态可执行时钟转换，且本层无事件触发转换可实际执行；所有子状态都执行时钟转换。

(2) 第二微步

A：本层外部事件引发：有：$name(S_m \xrightarrow[N]{E} s_n)=t$

$$\text{Fst-ex-or1}:\ \frac{S_m\xrightarrow[N]{E}S_n}{[n;\vec{s};m;T;T']\xrightarrow[N]{E}[n;\vec{s}_{[sm\rightarrow default(sm)]};n;T;T']}$$

$$\text{Fst-ex-or2}:\ \frac{S_m\xrightarrow[N]{E}S_n}{[n;\vec{s};m;T;T']\xrightarrow[N]{E}[n;\vec{s};m;T;T'\cup\{(t,duration(t))\}]}\ duration(t)\neq 0\wedge\neg To\text{-}exe(t)$$

$$\text{Fst-SetEvent-or1}:\ \frac{S_m\xrightarrow[trigger^-(t)]{trigger^+(t)}S_n}{[n;\vec{s};m;T;T']\xrightarrow[trigger^-(t)]{trigger^+(t)}[n;\vec{s}_{[Sn\rightarrow default(Sn)];n;T;T'}]}\ duration(t)=0$$

$$\text{Fst-SetEvent-or2}:\ \frac{S_m\xrightarrow[trigger^-(t)]{trigger^+(t)}S_n}{[n;\vec{s};m;T;T']\xrightarrow[trigger^-(t)]{trigger^+(t)}[n;\vec{s};m;T;T'\cup\{(t,duration(t))\}]}\ duration(t)\neq 0\wedge\neg To-exe(t)$$

$$\text{Fst-De-or}:\ \frac{-}{[n;\vec{s};m;T;T']\xrightarrow[trg^-(t)-event^-(t)]{trg^+(t)-event^+(t)}[n;\vec{s}_{[Sn\rightarrow default(Sn)]};n;T;T'-\{(t,lefttime)\}]}P$$

其中：$P=_{def} source(t)=s_l\wedge target(t)=s_n\wedge(t,lefttime)\in T'\wedge lefttime>0\wedge To\_Exe(t)$，$event^+(t)$ 和 $event^-(t)$ 分别表示 t 的正触发事件和非触发事件。

规则 Fst-ex-or1定义的是执行外部事件触发的转换，规则 Fst-ex-or2则是可执行的外部事件触发的转换被延迟；规则 Fst-SetEvent-or1说明的是执行预定事件的发生所触发的转换，而 Fst-SetEvent-or2定义的是预定事件触发的转换被延迟；规则 Fst-De-or 则对应执行本层有效的被延迟了的转换。

B：下一层

$$\text{Fst-ex-or3}:\ \frac{S_m\xrightarrow[N]{E}S_{m'}}{[n;\vec{s};T;T']\xrightarrow[N]{E}[n;S_{[Sm\rightarrow Sm'];m};T;T']}$$

$$\text{Fst-ex-and1}:\ \frac{S_m\xrightarrow[N]{E}S_{m'}}{[n;\vec{s}]\xrightarrow[N]{E}[n;\vec{s}_{[Sm\rightarrow Sm']}]}$$

对于由预定事件的发生和被延迟有效转换的实际执行所导致的子状态项的改变具有相应的规则。只用考虑实际执行了的转换，转换的延迟由本层处理。

(3) 第三及后续微步

A：本层：

In-or1：

$$\frac{-}{[n;\vec{s};m;T;T']\xrightarrow[trg^-(t)]{trg^+(t)}[n;\vec{s}_{[sm\rightarrow default(sm)]};n;T;T']}Q$$

其中：$Q=_{def} source(t)=s_m\wedge target(t)=s_n\wedge To\_Exe(t)$，描述的是转换执行；若转换被延迟，则对应规则 *In-or2*，相对规则 *In-or1*，边条件改为 $source(t)=s_m\wedge\neg To\_Exe(t)$，目标项中的 T' 增加记录 $(t,duration(t))$，其它不变。

B：下层：

In-or3：

$$\frac{S_m\xrightarrow[trg^-(t)]{trg^+(t)}S_{m'}}{[n;\vec{s};m;T;T']\xrightarrow[trg^-(t)]{trg^+(t)}[n;\vec{s}_{[sm\rightarrow sm']};m;T;T']}$$

对于与状态，若为单个子状态执行转换，对应规则 In-

and1，目标项的子状态集合中对应的子状态所对应的项换为该转换的目标项，其它不变。此情况下多个子状态中的转换只要一致，可同步执行。

以上规则中，同时执行的转换要求一致。除本层转换优于下层转换外，以上规则存在以下优先级关系：

* 非时钟转换所对应的规则优先级高于时钟转换所对应的规则；

* 第二微步对应规则中，Fst-ex-or1优先级最高，且排斥其它转换的执行；若执行的是 Fst-ex-or2，则可执行 Fst-SetEvent-or1；若执行规则 Fst-SetEvent-or2，则可执行规则 Fst-De-or。

结论 模板和规则的使用是 HRFSM 和 RTRSM* 区别其它相关工作的主要特点之一：模板保证了系统内部数据的封闭性和可继承性，具有合成性；规则增强了模型的严密性，且是规约原型化验证的直接基础[13]。而 RTRSM* 的转换有效期和事件预定机制使得其相对同样起源于 Statecharts 的其它规约语言具有更强的时间性质的描述能力，同时又保持了较 TA 等实时模型对事件驱动、交互性更为直接和自然的建模支持。

RTRSM* 图形化，具有较好的可读性和易用性，无二义性、结构化、可合成、可执行且具有良好的形式化语义，支持相应的规约的原型化和形式化验证。武汉大学计算机科学系需求工程小组实现了需求工程环境 SREE[12]，支持基于 HRFSM（文献中称为 RTRSM）的需求规约的编辑、原型化验证及部分形式化验证，并已实现了室温自动控制系统、机器人自动装配系统和火电厂锅炉自动防垢除垢控制系统等实验。此外，作者还提出了基于区间限制的命题时序逻辑 RITL，以描述 RTRSM* 模型性质，并对相应需求规约的形式化验证方法进行了探讨，包括：构造系统状态空间可达图进行模型检测；通过语义编码的方法，将其嵌入形式化验证工具 PVS[17]，在利用该工具进行主要是基于定理证明的规约验证的同时，也对两语言的语义进行了检查和验证[18]。

进一步工作主要包括两方面，一是使用更多大而复杂的实际应用来检验所做工作，二是 RTRSM* 合成语义的进一步研究，如利用 SOS 的元理论及相应已有形式化工具进行合成语义推导，以及相关形式化验证工作的完善和工具实现等。

致谢 在此，我们向阅读过本文并提出修改意见的中科院软件所王永吉研究员表示感谢。

参 考 文 献

1 Harel D. Statecharts: A Visual Approach to Complex Systems. Science of Computer Programming, 1987, 8(3): 231～274

2 Kesten Y, et al. Timed and Hybrid Statecharts and Their Textual Representation. Formal Techniques in Real-Time and Fault Tolerant Systems: Springer-Verlag, 1991. 591～620

3 Mok A, et al. Specification and analysis of real-time systems: Modechart language and toolset. In C. Heitmeyer and D. Mandrioli, eds. Formal Methods for Real-Time Computing, J. Wiley and Sons, 1996. 33～54

4 Douglass B P. Real-Time UML: Developing Efficient Objects for Embedded Systems (2nd edition). Berkeley, California: Addison Wesley Longman, 1998

5 Selic B, Gullekson G, Paul T. Ward: Real-Time Object-Oriented Modeling. New York: John Wiley&Sons, 1994

6 Cheng B H C, Campbell L A, et al. Automatically Detecting and Visualizing Errors in UML Diagrams. Requirements Engineering Journal, Springer-Verlag, 2002, 7(4): 264～287

7 董威，王戟，等. Statecharts 的模型检验方法. 软件学报，2003，14(4)：750～756

8 Alur R, Dill D L. Automata for modeling real-time systems. In: Proc. of 17th ICALP, LNCS 443: Springer-Verlag, 1990

9 Litvak B, Tyszberowicz S. Behavioral Consistency Validation of UML Diagrams. In: SEFM'03, Brisbane, Australia, 2003

10 Henzinger T A, Manna Z, Pnueli A. Timed transition systems. In: Bakker J W de et al, eds. Real-Time. Theory in Practice, LNCS 600: Springer-Verlag, 1992. 226～251

11 Wu Guo-qing, Xiao Hai-feng, et al. Specifying requirements of real-time system with rules and templates. Wuhan University Journal of Natural Sciences, 2000, 5(3): 278～284

12 舒风笛，毋国庆，王敏，等. 面向嵌入式实时软件系统需求工程环境—SREE. 计算机科学，2002，29(4)：4～8

13 Heitmeyer C L, Jeffords R D, et al. A Benchmark for Comparing Different Approaches for Specifying and Verifying Real-time Systems. In: Proc., Tenth Intern. Workshop on Real-Time Operating Systems and Software. New York, May, 1993

14 Dasarathy B. Timing Constraints of Real-time systems: Constraints for Expressing Them, Methods of Validating Them. IEEE Trans. on Software Engineering, 1985, 11(1): 80～86

15 Aceto L, Fokkink W J, Verhoef C. Structural operational semantics. In: Bergstra J A, Ponse A, Smolka S A, eds. Handbook of Process Algebra: Elsevier, 2001. 197～292

16 Mendler M, Luettgen G. Statecharts From Visual Syntax to Model-Theoretic Semantics. In: Bauknecht K, Brauer W, Mück T, eds. IDFST 2001, Vienna, Austria, Sep. 2001. 615～621

17 Shankar N. PVS: Combining specification, proof checking, and model checking. In: Srivas M, Camilleri A, eds. FMCAD '96, LNCS 1166: Springer-Verlag, Nov. 1996. 257～264

18 舒风笛. 嵌入式实时软件系统的需求规约与验证：[博士学位论文]. 武汉：武汉大学，2003