第 23 卷第 9 期 系统仿真学报© Vol. 23 No. 9
2011 年 9 月 Journal of System Simulation Sep., 2011

基于时间事件模型的实时系统仿真与时序分析

陈 艳[1]，徐晓峰[2]，李晓潮[1]，郭东辉[1]
（1.厦门大学信息科学与技术学院，厦门 361005；2.厦门大学物理系，厦门 361005）

摘 要：提出了一种基于*时间事件模型*的实时系统时序分析方法。该方法以仿真为基础对目标系统进行分析，不仅可以描述实时系统的周期和非周期任务，各种同步事件以及与时间有关的行为特征，*而且还能够刻画实时系统中多个任务间的通信与同步特性*；通过基于优先级的可抢占式的任务调度算法对目标模型进行仿真执行，*并基于仿真结果对其进行时序分析*，如任务响应时间和执行时间等，*避免了形式化分析方法存在的状态空间爆炸问题*。最后通过实例验证本模型与方法的有效性与实用性。

关键词：时间事件模型；仿真；时序分析；任务调度；任务间通信与同步
中图分类号：TP316 **文献标识码：**A **文章编号：**1004-731X (2011) 09- 1787-08

Simulation and Timing Analysis of Real-time System Based on Timed Event Model

CHEN Yan[1], *XU Xiao-feng*[2], *LI Xiao-chao*[1], *GUO Dong-hui*[1]
(1. School of Information Science and Technology, Xiamen University, Xiamen 361005, China;
2. Department of Physics, Xiamen University, Xiamen 361005, China)

Abstract: *A timed event model* and simulation based approach was proposed to analyze timing properties of real-time systems. *The timed event model was used to specify periodic or aperiodic tasks, the timing behaviors and the inter-tasks communication and synchronization behaviors of real-time systems*. The approach used a priority-based preemptive scheduling algorithm to simulate models, and then *got the timing properties, such as response time, execution time, etc., via analyzing the execution sequence generated*. Different from traditional formal methods, *it has no state-explosion problem*. Finally, a case study was proposed to validate the model and show the practicability of the approach.

Key words: timed event model; simulation; timing analysis; task scheduling; inter-task communication and synchronization

引 言

实时系统是一种带有时间约束的计算系统，这类系统的可靠性不仅要求其在逻辑上是正确的，而且要保证系统时序(Timing)的正确性[1]。近年来，随着实时系统在消费电子、工业控制、国防安全等领域的广泛应用，特别是在飞行控制和核电站等设备中的应用，实时系统的可靠性显得尤为重要。要确保这类关键设备运行的可靠性，在实时系统的软件设计过程中，需要进行系统运行的时序分析，以保证系统时序的正确性[2]。

实时系统的时序分析方法主要有两种：模型检测(Model Checking)方法和基于仿真的分析方法。模型检测方法是以可达性分析技术为基础的，它通过对系统状态的建模来描述实时系统与时间有关的行为特征，从而实现系统时序的正确性分析[3-5]。如以符号化和参数化分析方法[6-8]为基础的实时系统建模分析工具就是利用时间自动机[5]及其相关的系统建模方法[9-11]来描述实时系统的实际运行状态的。但是对于复杂系统的分析，他们存在状态空间爆炸问题[6-8]。与模型检测方法相比，基于仿真的分析方法不仅可以实现时序的精确分析，也不存在状态空间爆炸问题[14]。如文献[12]提出工具 STRESS，能够描述目标系统各个任务的行为特性，通过定义不同的资源共享和任务调度算法来仿真分析实时系统的时序特性；文献[13]提出工具 DRTSS，采用任务/资源图来描述系统的任务情况，并通过搜索匹配的模型参数和任务调度算法来仿真分析目标系统的时序特性；文献[14-17]提出工具 ART-FW，采用概率模型语言 ART-ML 来描述系统任务间消息传递的

收稿日期：2009-05-15 **修回日期：**2009-07-22
基金项目：国家自然科学基金(60753001);博士点基金(20090121110019)
作者简介：陈艳(1981-)，男，福建古田人，博士生，研究方向为实时系统分析与测试；**徐晓峰**(1983-)，男，福建泉州人，博士生，研究方向为软件测试；**李晓潮**(1970-)，男，福建福州人，博士，副教授，研究方向为实时嵌入式系统设计；**通讯作者:郭东辉**(1967-)，男，福建莆田人，博士，教授，博导，研究方向为人工智能、计算机网络通讯、集成电路设计自动化。

行为特性，并通过随机仿真方法获取目标系统时序特性的概率分布情况。然而，以上这些方法主要侧重于针对不同的资源分配和任务调度算法进行仿真分析，没有对实时系统中各种同步事件进行定义和描述，因此无法针对实时系统各种复杂的时间行为特性进行建模和仿真。考虑到系统时间行为是影响系统时序的重要因素，本文提出一种时间事件模型，该模型通过对系统同步事件、时间限制、状态迁移等进行形式化定义，可以方便地描述实时系统周期和非周期任务及其复杂的时间行为特性；同时，提出一种基于时间事件模型的系统仿真和时序分析方法，即通过系统任务调度算法、任务间通信与同步机制以及虚拟时钟等对系统模型进行仿真，并根据仿真所获取的执行信息对目标系统进行时序分析，获取系统任务响应时间和执行时间等信息，从而有效地了改进先前方法的不足之处。

下面第 1 节将描述实时系统的时间事件模型及其相关定义；第 2 节介绍模型的系统仿真与实现方法；第 3 节通过一个实例说明如何使用本模型和方法来对目标系统进行时序分析；最后是结论和未来工作。

1 实时系统建模

图 1 是本方法的整体框架图。主要包括三个部分：第一部分是指使用时间事件模型对目标系统进行描述和建模；第二部分是指仿真方法，主要包括任务调度算法，任务间通信与同步机制，系统时钟以及系统仿真执行算法等；第三部分是指对仿真结果进行分析。

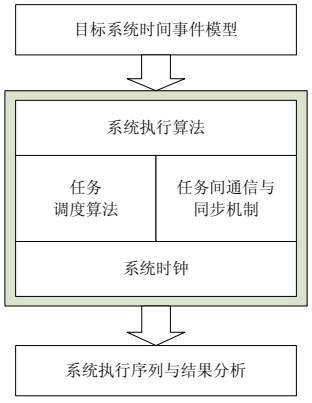


图 1 系统整体框图

实时系统通常是多任务并发执行，系统通过特定的调度算法实现资源的控制，任务间通过信号量、消息队列等方式实现通信与同步。本文提出的时间事件模型以系统的同步事件为实体来描述系统的时间行为特性。为此，本节首先给出同步事件的定义。

定义 1 同步事件：在实时系统中，同步事件 e 可表示为 $e =< p,\lambda,o,x,i >$，其中 p 表示该事件所在的任务；λ 表示事件类型；o 表示事件的操作对象；x 表示事件的相关数据或变量；i 表示事件的标识号。

同步事件的主要类型包括实际常用的获取信号量(*ST*)、释放信号量(*SG*)、发送消息(*MS*)、接收消息(*MR*)、任务延时(*TD*)等操作。各种类型的事件与操作对象以及相关数据的对应关系如表 1 所示。在实际应用中，还可以根据需要添加其他的同步事件。

表 1 各种同步事件

同步事件类型(λ)	操作对象(o)	相关数据(x)
MS	消息队列	发送的数据
MR	消息队列	接收到的数据
SG	信号量	——
ST	信号量	——
TD	——	延迟时间

定义 2 时间事件模型：实时系统的时间事件模型是一个多元组：$S =< P,X,E,A,\tau,\mu,\gamma >$，其中集合 P 表示系统中的任务集；$\forall p \in P$ 是一个周期或非周期的任务，包括任务名、任务 ID、任务的优先级、起始点和结束点等；集合 X 是一组全局变量，$\forall x \in X$，可以是整型变量(*int*)、消息队列(*mq*)、二进制信号量(*bs*)或计数信号量(*cs*)；E 是所有同步事件集合，$\forall\ e =< p,\lambda,o,x,i > \in E$ 为一个同步事件；$A \subseteq E \times E$ 是一组迁移，$\forall\ a(e_s,\ e_t) \in A$，包括起始事件 e_s 和目标事件 e_t，且有：

♦ $\tau(a)$是定义在迁移 a 上的一个时间限制，即表示从起始事件 e_s 到目标事件 e_t 所需要的执行时间 t 满足 $x \le t \le y$，其形式语法如下：

$$\tau(a)::= [x, y] \wedge \{x, y\} \in R^+$$

其中 R^+ 表示正实数。

♦ $\mu(a)$是定义在迁移 a 上的触发条件判定，当 $\tau(a) \wedge \mu(a) = true$ 的时候，a 发生，即系统从事件 e_s 迁移至事件 e_t，其形式语法如下:

$$\mu(a)::= true \mid (x \sim n) \mid (x \sim y) \mid \ \neg\mu \mid (\mu_1 \wedge \mu_2)$$

其中 $x, y \in X$ 是系统变量；$n \in R^+$；$\sim \in \{\le, \ge, >, <, =, !=\}$；若 $\mu(a)::= true$,则可以省略。

♦ $\gamma(a)$是一组变量赋值，$\tau(a) \wedge \mu(a) = true$ 的时候，$\gamma(a)$将开始执行。其形式语法如下所示：

$$\gamma(a) ::= L := R \mid (\gamma_1 \vee \gamma_2)$$
$$L ::= x$$
$$R ::= n \mid x \mid (x \sim y) \mid (x \sim n)$$

其中 $x, y \in X$ 是系统变量；$n \in R^+$; $\sim \in \{+, -, *, /\}$。

图 2 是一个时间事件模型例子：该模型包括 2 个任务 *T*1 和 *T*2，其中 *T*1 的优先等级大于 *T*2；该模型中包括一个二进制信号量 *s*1 以及一个整型变量 *x*；每个任务均包含三个事件：先是任务延时，然后获取信号量，在完成对 *x* 的赋值之后再释放信号量；所有事件之间迁移的时间限制

均为 1 到 10 个单位时间。

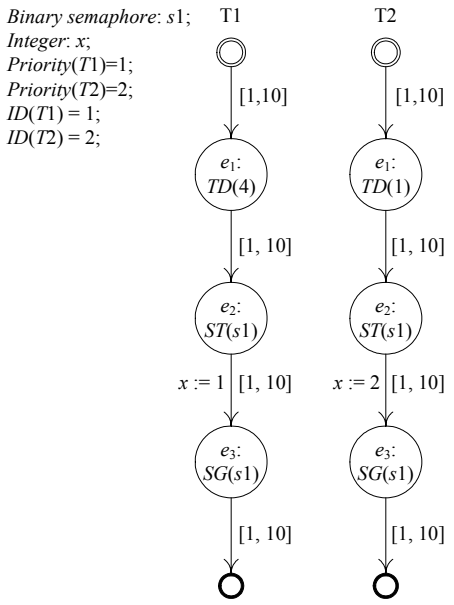


图 2 一个时间事件模型例子

定义 3 系统状态：令 $S =< P, X, E, A, \tau, \mu, \gamma >$ 是一个实时系统，一个系统状态 v 为 $P \cup X \cup E$ 的一个映射，且有：

♦ 对于任意一个整型变量 $int \in X$, $v(int) \in I$ (I 为整数集合);

♦ 对于任意一个消息队列 $mq \in X$, $v(mq) \in I$。若 $v(mq)>0$，则表示该消息队列中有 $v(mq)$个消息在等待被接收；若 $v(mq)<0$，则表示$|v(mq)|$个接收消息事件在等待消息；

♦ 对于任意一个二进制信号量 $bs \in X$，$v(bs) = true \vee false$，即表示该信号量可用还是不可用；

♦ 对于任意一个计数信号量 $cs \in X$，$v(cs) \in I^+$ (I^+为大于等于 0 的整数集合)，当 $v(cs) = 0$ 时，该信号量不可用；

♦ $\exists e \in E$, $v(e) = i$ 为 e 的标识号，表示当前系统状态下执行到哪个事件；

♦ $\forall p \in P$，$v(p) = ready \vee wait \vee run$，表示当前状态下各个任务的状态(将在下一节介绍)。

因此，两个系统状态 v_1 和 v_2 不一样，当且仅当：

$\exists u \in P \cup X \cup E$, $v_1(u) \neq v_2(u)$.

定义 4 系统步进：实时系统 S 从系统状态 v_1 迁移到系统状态 v_2 称为一个系统步进，且有：

♦ $\exists e \in E$,且 e 被执行，则发生一个系统步进；

♦ $\exists a(e_s, e_t) \in A$,且 $\tau(a) \wedge \mu(a) = true$,则系统从事件 e_s 迁移至事件 e_t，且 $\gamma(a)$发生，此时发生一个系统步进；

♦ $\exists p \in P$，且 p 的状态发生变化，则发生一个系统步进。

系统的执行就是以系统步进为单位，其结果是得到一个系统执行序列。

定义 5 执行序列：实时系统 S 的任意一次执行称为一个执行序列，可表示为：

$Q = \{<t_c, e>\} \cup \{<t_{start}, p>\} \cup \{<t_{end}, p>\}$.

其中序列$\{<t_c, e>\}$表示系统仿真执行中的所有同步事件及其完成时间；序列$\{<t_{start}, p>\}$记录了任务 p 每次开始执行的时间；序列$\{<t_{end}, p>\}$记录了任务 p 每次结束执行的时间。

因此，假设任务 p 被分成 n 次执行，则任务 p 的响应时间 $rt(p)$为：

$$rt(p) = t_{end}(n) - t_{start}(1) \quad (1)$$

而任务 p 的执行时间为：

$$et(p) = \sum_{i=1}^{n} t_{end}(i) - t_{start}(i) \quad (2)$$

2 系统仿真与实现方法

为了实现实时系统时间事件模型的系统仿真，需要设计任务调度算法、任务间通信与同步机制和系统时钟等模型。其中任务调度算法作为一个独立的系统级任务与目标模型并发执行，负责控制目标系统模型中的任务状态切换；任务间通信与同步机制完成目标系统模型中各种同步事件的执行过程；而系统时钟则用来控制目标系统模型的执行时间。

2.1 任务调度算法

本系统仿真采用基于优先级可抢占的任务调度算法[18]来控制时间事件模型中的任务执行。模型中的每个任务都存在三种状态：就绪(*ready*), 等待(*wait*)和运行(*run*)。这三种状态之间的迁移如图 3 所示：

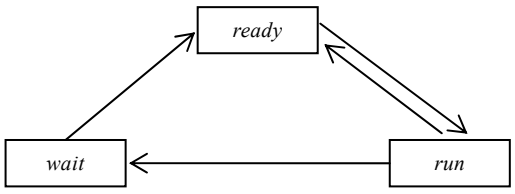


图 3 任务状态迁移图

设 $S =< P, X, E, A, \tau, \mu, \gamma >$ 是一个实时系统；R 为准备就绪的任务队列；p 为正在执行的任务；$Pri(p)$表示 p 的优先等级；函数 $MaxPri(R)$实现从 R 中提取优先等级最高的任务；函数 $SetState(p, s)$设置任务 p 的状态，其中 $s = ready \vee wait \vee run$。任务状态迁移可表达如下：

♦ $ready \rightarrow run$ 和 $run \rightarrow ready$：

当某个就绪任务的优先等级比其他就绪任务和正在运行的任务的优先级高，那么该任务将变成运行状态，同时正在运行的任务变成就绪状态，即：

$\exists p_r \in R \wedge (p_r = MaxPri(R)) \wedge (Pri(p_r) > Pri(p))$

$\rightarrow SetState(p_r, run) \wedge SetState(p, ready)$；

♦ $run \rightarrow wait$：

当正在运行的任务 p 试图访问一个不可获取的资源或进入延时的时候，任务 p 从 *run* 状态转变到 *wait* 状态。具体如下：

1) 当 p 欲从一个空的消息队列中接收信息时，其状态变成 *wait*：

$\exists MR(mq) \in E \wedge mq=null \rightarrow SetState(p, wait)$;

2) 当 p 欲获取一个为 *false* 的二进制信号量，其状态变成 *wait*：

$\exists ST(bs) \in E \wedge v(bs) = false \rightarrow SetState(p, wait)$;

3) 当 p 欲获取一个值为 0 的计数型信号量，其状态变成 *wait*：

$\exists ST(cs) \in E \wedge v(cs) = 0 \rightarrow SetState(p, wait)$;

4) 当 p 执行任务延时事件时，其状态变成 *wait*：

$\exists TD(t) \in E \rightarrow SetState(p, wait)$;

♦ $wait \rightarrow ready$：

对于任意一个处于等待状态的任务，若其等待的资源可获取，或延时结束，则该任务就从 *wait* 状态变成 *ready*。

2.2 任务间通信与同步机制

任务间通信与同步机制主要包括消息传递、信号量以及任务延时等。

2.2.1 消息传递

本系统仿真采用 FIFO 消息队列和异步消息传输机制[19]，主要包括消息发送(*MS*)和消息接收(*MR*)事件：当任务执行 *MS* 事件，不管该消息是否有被接收到，该任务都不会阻塞而继续往下执行；当任务执行 *MR* 事件，若消息队列中有消息，则接收消息且继续往下执行，否则，该任务将会等待直到从消息队列中接收到消息为止。因此，设 Q 表示某一消息队列，R 表示与 Q 关联的 *MR* 队列，S 表示与 Q 关联的 *MS* 队列，则 *MS* 和 *MR* 事件的执行过程可分别表示如下：

$$\begin{aligned}MS ::= &((R==null) \rightarrow Add(Q, x)) \wedge \\ &(\neg (R==null) \rightarrow (r=GetFirst(R) \\ &\qquad \wedge p=GetTask(r) \\ &\qquad \wedge SendMessage(r,x) \\ &\qquad \wedge SetState(p,ready))).\end{aligned}$$

$$\begin{aligned}MR ::= &\ p = GetTask(MR) \wedge \\ &((S==null) \rightarrow Add(R,MR) \wedge SetState(p,wait)) \wedge \\ &(\neg (S==null) \rightarrow (x=GetFirst(Q) \wedge SendMessage(r, x))).\end{aligned}$$

其中，函数 $Add(A, a)$是指把 a 加到队列 A 中，x 表示一则消息；函数 $GetTask(e)$是获取事件 e 所在的任务；函数 $GetFirst(A)$是指从已知队列 A 中提取第 1 个元素；函数 $SendMessage(r, x)$是指把消息 x 发给接收事件 r。

2.2.2 信号量

本模型主要有二进制信号量(*bs*)和计数信号量(*cs*)，其相关的操作事件为获取信号量事件(*ST*)和释放信号量事件(*SG*)。设 Q 是信号量的等待队列(FIFO)，则 *ST* 和 *SG* 的执行特性如下：

1) 当某个任务获取信号量的时候，如果该信号量是可获取的，那么信号量被获取，任务继续往下运行，同时该信号量变成不可获取状态；如果该信号量处于不可获取状态，那么该任务将进入等待队列当中。具体执行过程可表示如下：

$$\begin{aligned}ST(bs) ::= &\ (v(bs)==true) \rightarrow (v(bs)=false)) \wedge \\ &((v(bs)==false) \rightarrow (p=GetTask(ST) \wedge \\ &\quad Add(Q, ST) \wedge SetState(p,wait))).\end{aligned}$$

$$\begin{aligned}ST(cs) ::= &\ (v(cs)>0) \rightarrow (v(cs)= v(cs) - 1)) \wedge \\ &((v(cs)==0) \rightarrow (p=GetTask(ST) \wedge \\ &\quad Add(Q, ST) \wedge SetState(p,wait))).\end{aligned}$$

2) 当某个任务释放信号量的时候，如果该信号量处于可获取状态，那么信号量保持原有状态，任务继续执行；如果该信号量不可获取，那么将信号量变成可获取状态，同时如果等待队列中有任务在等待该信号量，那么最先进入等待队列的任务变成就绪状态。具体执行过程可表示如下：

$$\begin{aligned}SG(bs) ::= &\ (v(bs)==true) \rightarrow continue) \wedge \\ &((v(bs)==false) \wedge (\ Q ==null) \rightarrow (v(bs)=true)) \wedge \\ &((v(bs)==false) \wedge (\ Q\ !=null) \rightarrow (p=GetFirst(Q) \wedge \\ &\quad SetState(p,ready))).\end{aligned}$$

$$\begin{aligned}SG(cs) ::= &\ (v(cs)>0) \rightarrow (v(cs)= v(cs) + 1)) \wedge \\ &((v(cs)==0) \wedge (\ Q ==null) \rightarrow (v(cs)=1)) \wedge \\ &((v(bs)==0) \wedge (\ Q\ !=null) \rightarrow (p=GetFirst(Q) \wedge \\ &\quad SetState(p,ready))).\end{aligned}$$

2.2.3 任务延时

当一个正在运行的任务 p 执行任务延时事件 $TD(t)$时，任务 p 将从 *run* 状态转变到 *wait* 状态，直到延时结束才能从 *wait* 状态变成 *ready*。其执行过程可表达如下：

$$\begin{aligned}TD(t) ::= &\ c_0=c=t_0 \wedge SetState(p, wait) \wedge \\ &((\text{DO } c=t_n \text{ UNTIL } (c - c_0)>t) \rightarrow SetState(p, ready)).\end{aligned}$$

其中 c_0 和 c 是两个变量，t_0 表示初始时间，t_n 表示当前时间。

2.3 系统时钟

为了能够实现时间的精确仿真，在模型的事件迁移中包含事件的执行时间信息 τ，它表示从模型的一个事件到另一个事件所需的时间损耗。在实际系统运行过程中，由于系统软硬件方面的误差以及外界环境干扰影响，该时间损耗是个不确定的数值，因此 τ 在模型中也不是一个精确的数值，而是一个时间范围。该范围可通过系统的追踪技术[20]来获取，也可以使用最坏执行时间(WCET)分析技术[18]来确定。

迁移时间损耗的仿真方法如图 4 所示。在系统仿真过程中，我们通过虚拟时钟来控制模型的执行时间。当时间事件模型 *S* 开始仿真执行时，系统的全局时钟 *C* 便开始以计数器的方式递增，直到 *S* 仿真结束。对于 *S* 的任意一个迁移 *a*，系统通过 *pickupExecutionTime*()函数从 $\tau(a)$中随机抽取一个数值作为该迁移的时间损耗，然后通过 *getGlobalClockValue*()函数读取全局时钟 *C* 的值，并通过比较判断迁移时间是否结束。因此，本方法可根据系统的精度要求来调整系统仿真的时间精度。

```
Process timingSimulation(Transition a)                    (1)
  t = pickupExecutionTime( τ (a));                        (2)
  cc = getGlobalClockValue(C);                            (3)
  pc = cc;                                                (4)
  While (pc − cc) < t, do                                 (5)
      pc = getGlobalClockValue(C);                        (6)
  End while                                               (8)
End process                                               (9)
```

图 4 迁移时间损耗的仿真方法

2.4 系统仿真执行算法

基于任务调度算法，各种同步事件的执行特性以及系统时钟，系统便可对时间事件模型进行仿真执行。在模型的仿真执行过程中，所有的任务并发执行，且每个任务的执行过程如图 5 所示。其中，语句(2)表示任务从起始点开始执行；语句(5)完成获取事件 e_s 的下一个迁移 *a*，由于一个事件所引发的迁移可能有多个，因此在 *getNextTransition* 函数中需要用 $\mu(a)$来判定和选择一个适合的迁移；语句(6)模拟从起始事件 e_s 到下一个事件所需要的迁移时间损耗；当执行时间满足要求的时候，语句(7)完成变量赋值；然后，通过语句(8)获取到下一个同步事件 e_t，并用语句(9)的 *executeEvent* 函数进行执行。在执行同步事件的时候，需要根据事件的类型，按照第 2.2 节所描述的各种事件的执行特性来分别执行。

```
Process Task_execution                                    (1)
  e_s = "start";                                          (2)
  e_t = null;                                             (3)
  While not e_t = "end", do                               (4)
      a = getNextTransition(e_s);                         (5)
      timingSimulation(a);                                (6)
      executeAssignment( γ (a));                          (7)
      e_t = getNextEvent(a);                              (8)
      executeEvent(e_t);                                  (9)
      e_s = e_t;                                          (10)
  End while                                               (11)
End process                                               (12)
```

图 5 任务执行过程

另外，在任务执行的同时，系统的任务调度算法也在按照第 2.1 节描述的方法并行执行。因此，在任务执行的每个时刻，都有可能被更高优先级的任务抢占而暂停执行。

图 6 展示了图 2 所示模型的一次系统执行过程。其中，t_o 表示迁移时间损耗；t_d 表示任务延时时间。这次系统执行的过程如下：开始的时候，任务 *T*1 和 *T*2 都处于 *ready* 状态；由于 *T*1 的优先级高于 *T*2 的优先级，因此任务调度算法挑选 *T*1 先开始执行；当 *T*1 执行任务延时事件 *TD*(4)时，进入 *wait* 状态，等待 4 个单位时间；这时任务切换到 *T*2 开始执行；在 *T*2 相继执行完 *TD*(1)和 *ST*(*s*1)事件，并经过 1 个单位时间，此时 *T*1 任务延时结束，进入 *ready* 状态，因此系统又将任务调度到 *T*1 继续执行；当 *T*1 开始执行 *ST*(*s*1)的时候，由于信号量 *s*1 已被 *T*2 获取，因此 *T*1 被阻塞，系统任务又切换到 *T*2，直到 *T*2 对 *x* 进行赋值并释放了 *s*1 之后，*T*1 才又得以继续执行；最后，*T*1 先结束执行，然后 *T*2 再结束。

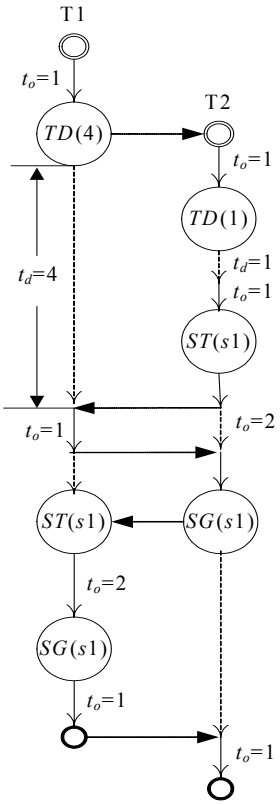


图 6 一个系统执行例子

由此可见，时间事件模型的仿真方法不存在状态空间爆炸的问题。另外，在系统执行过程中，对各个事件及其起始执行时间和任务的状态迁移时间进行记录，便可获取系统的执行序列，从而可以进行时序分析。

3 实例分析

我们采用 JAVA 技术对本文的建模和仿真方法进行编

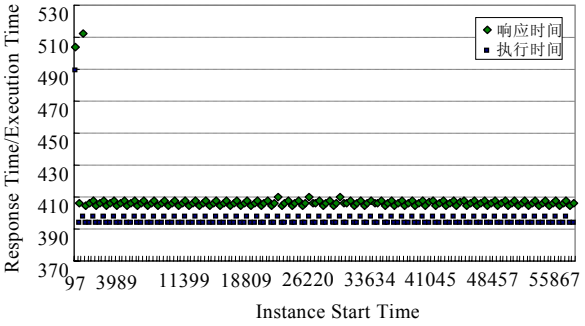
程实现，其中时间事件模型采用 XML 语言进行描述和存储。在本节，我们使用计算机科学领域中经典的铁路穿越问题(Railway Crossing Problem)[21]为实例来验证本文所阐述的建模和仿真方法的有效性和实用性。该问题的基本思想是：使用一个控制器自动控制多列火车穿越一个临界区域(比如一座桥)，而控制器的主要任务就是避免多辆火车同时穿越该临界区域。我们的验证思路是：首先，在实时嵌入式开发平台上采用消息传递机制来解决该问题，并在系统运行时收集系统的执行信息；然后，在普通 PC 机上使用本方法建立该实时系统的时间事件模型，并在系统仿真执行中收集系统的执行序列；最后，通过比较实际系统和仿真系统的任务响应时间和执行时间来验证模型的正确性。

第一步，在 Xilinx ML505 开发平台[22]上编程实现铁路穿越问题的解决方案。该开发平台含有一个 MicroBlaze 嵌入式软核，另外，我们还移植了一个 uC/OS II 实时操作系统[23]来对实际系统进行编程。实际系统主要包括两种任务：一个是表示火车的任务(取名为 Train)，另一个是表示控制器的任务(取名为 Controller)。任务之间采用消息传递机制进行通信和同步，其主要代码如图 7 所示。

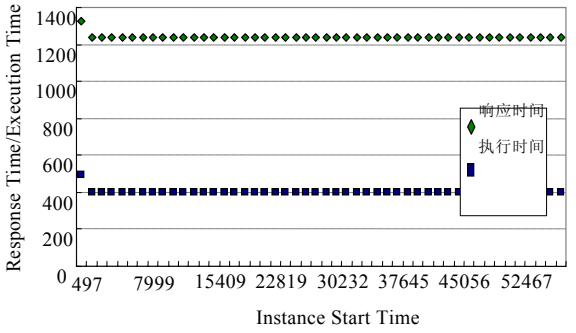
```
static void Train(void *p_arg){
    //Initialization;
    //mq and ms are two message queues.
    while(1){
        rr_OSQPost(mq, “request”);    //Send request message;
        /* I/O operations */
        rr_OSQPend(ms, 0, &err);      //Wait for allowance;
        /* I/O operations */
        rr_OSQPost(mq, “finish”);     //Finish crossing
        /* I/O operations */
        OSTimeDly(100);               //Task Delay
    }
}

static void Controller(void *p_arg){
    //Initialization;
    //mq and ms are two message queues.
    int bridge = 0;                   //The status of bridge
    int waiting = 0;                  //The length of waiting list
    while(1){
        //waiting for message
        msg = (char *)rr_OSQPend(mq, 0, &err);
        if(msg=="request"){
            if(bridge == 0){          //The bridge is available
                rr_OSQPost(ms, “allow”);
                bridge = 1;
            }else{                    //The bridge is unavailable
                waiting++;            //Update waiting list
            }
        }else if(msg=="finish"){
            if(waiting>0){            //Update waiting list
                msg = "allow";
                rr_OSQPost(ms, (void *)msg);
                waiting--;
            } else{
                bridge = 0;           //Update the status of bridge
            }
        }
    }
}
```

图 7　实际系统的主要代码

在此，我们假设系统有 2 列火车，其任务名称分别为 Train1 和 Train2，且与 Controller 之间的任务优先级关系为：Controller 高于 Train1；Train1 高于 Train2。我们通过实际系统运行，收集到系统的执行信息，包括任务切换信息以及同步事件执行信息等。通过分析系统的执行信息，获得系统任务的响应时间和执行时间分布情况，如图 8 所示(限于文章篇幅，本文主要讨论火车任务的时序情况)。其中，横坐标表示每次任务开始执行的时间，纵坐标表示每次任务的响应时间和执行时间，且时间单位为 tick (在本系统中，100 个 tick 为 1 秒)。



(*a*) Train1 的任务响应时间和执行时间分布情况

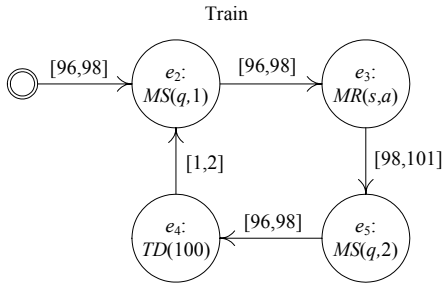


(*b*) Train2 的任务响应时间和执行时间分布情况

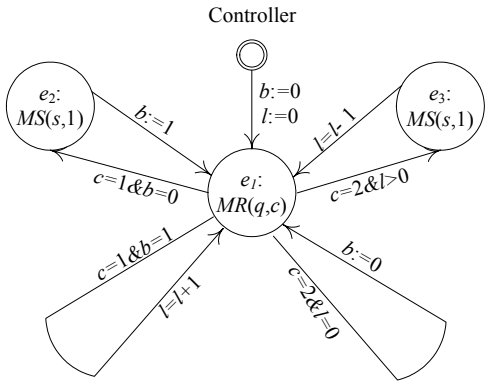
图 8　实际系统中火车任务的响应时间和执行时间分布情况

第二步，使用本方法对实际系统进行建模，得到的时间事件模型如图 9 所示。模型中的迁移时间信息是通过分析实际系统的执行信息获得的。在该模型中，q 和 s 为消息队列；a，b，c 和 l 都是整型变量。火车任务(Train)的执行过程如下：在开始运行 96 到 98 个 tick 后，火车靠近临界区域；于是它发送一个消息给控制器($MS(q,1)$)，请求获取穿越权限，如果控制器没有发送应答消息，则该列火车将停止运行，直到接收到来自控制器的许可方能开始穿越($MR(s,a)$)；每列火车穿越临界区域的时间为 98 到 101 个 tick，穿越成功后便发送一个消息给控制器，然后离开($MS(q,2)$)。火车任务是一个周期性的任务，在任务延时 100

个 tick 之后，又重新开始新一轮穿越(*TD*(100))。而控制器(Controller)任务为一个触发型任务：在开始经过 1 到 2 个 tick 之后(该任务所有的事件迁移时间都为 1 到 2 个 tick，为了图形简洁，在此省略)，便进入等待消息状态(*MR*(*q*,*c*))；如果收到来自火车的穿越请求且此时没有其他火车在使用该临界区域，则发送一个应答消息，若此时已有其他火车在使用该临界区域，则需等到那列火车成功穿越之后，才能发送应答消息(*MS*(*s*,1))。其中，变量 *b*=0 表示当前没有火车在穿越，*b*=1 表示当前有火车在穿越；变量 *l* 则用来记录当前有多少火车在等待应答。



(*a*) 火车模型



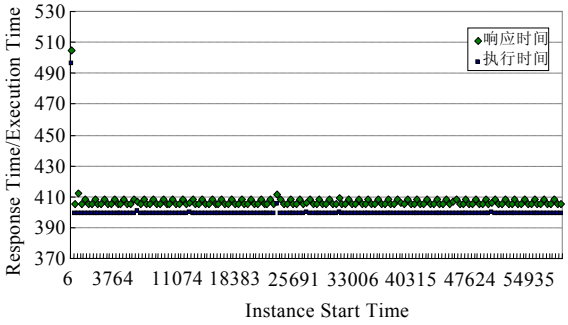
(*b*) 控制器模型

图 9 系统的时间事件模型

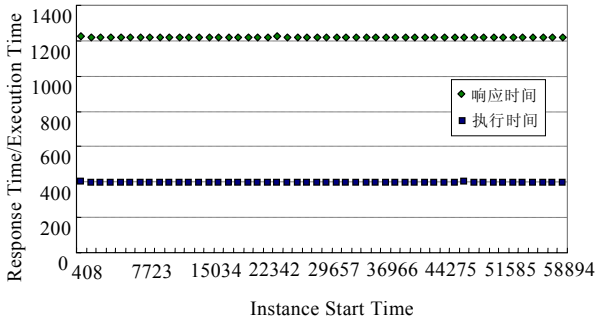
同样地，我们令系统有 2 个火车模型，分别 Train1 和 Train2，且与 Controller 之间的任务优先级关系与实际系统一致，即为：Controller 高于 Train1；Train1 高于 Train2。通过对系统时间事件模型的仿真执行，获取系统执行序列信息，再通过对执行序列进行分析，得到系统模型中两个火车任务的响应时间和执行时间分布情况，如图 10 所示。

第三步，比较实际系统和仿真系统的任务响应时间和执行时间。我们分别统计实际系统和仿真系统中火车任务的最大响应时间，最小响应时间，平均响应时间，最大执行时间，最小执行时间和平均执行时间，其结果如表 2 所示。与实际系统相比，仿真系统中 Train1 任务的平均响应时间误差约为 0.2%，平均执行时间误差约为 0.5%，；Train2 任务的平均响应时间误差约为 1.4%，平均执行时间误差约为 0.7%。由此可见，火车任务在实际系统和仿真系统中的响应时间和执行时间分布情况基本一致，从而验证了本方法的正确性。

另外，我们还可以通过增加任务，或修改任务的相对优先级等方式来进一步对目标系统进行各种影响分析。



(*a*) Train1 的响应时间和执行时间分布情况



(*b*) Train2 的响应时间和执行时间分布情况

图 10 仿真系统中火车任务的响应时间和执行时间分布情况

表 2 实际系统与仿真系统中火车任务的时序特性对比结果

对比项目	实际情况		仿真情况	
	Train1	Train2	Train1	Train2
最大响应时间	512	1237	505	1224
最小响应时间	405	1235	405	1218
平均响应时间	407	1235	406	1218
最大执行时间	398	402	399	405
最小执行时间	394	402	397	399
平均执行时间	395	402	397	399

4 结论

本文提出了一种以时间事件模型为基础的实时系统仿真和分析方法，能够支持信号量、消息传递以及任务延迟等各种同步事件，并且还可方便地扩展其它类型的同步事件。系统采用基于优先级可抢占的任务调度算法和各种通信与同步机制来实现目标模型的精确仿真，通过获取模型仿真的执行序列信息，实现对目标模型的时序分析，并可根据实际系统的精度要求方便地调整仿真的时间精度。本方法可用于实时系统的影响分析。在接下去的工作中，

我们将加入中断和系统环境的仿真模拟，同时，还将在更多的实际例子中进一步验证该模型与仿真分析方法的有效性和实用性。

参考文献:

[1] N Suri, M M Hugue, C J Walter. Synchronization issues in real-time Systems [J]. Proceedings of the IEEE (S0018-9219), 1994, 82(1): 41-52.

[2] Y Chen, Y H Lee, W E Wong, D H Guo. A race condition graph for concurrent program behavior [C]// Proceeding of the Intelligent System and Knowledge Engineering. Xiamen, Fujian, China. USA: IEEE, 2008: 662-667.

[3] E M Clarke, J M Wing. Formal methods: state of art and future directions [J]. ACM Computing Surveys (S1557-7341), 1996, 28(4): 626-643.

[4] Y Chen, Y H Lee, X F Xu, W E Wong, D H Guo. A Genetic Algorithm Based Approach for Event Synchronization Analysis in Real-time Embedded Systems [C]// Proceedings of The 6th International Conference on Embedded Software and Systems. Hangzhou, Zhejiang, China. USA: IEEE, 2009: 201-208.

[5] R Alur, D L Dill. Automata-theoretic verification of real-time systems [M]// Formal Methods for Real-Time Computing, Trends in Software Series. USA: John Wiley & Sons Publishers, 1996: 55-82.

[6] F Wang. Efficient verification of timed automata with BDD-like data-structures [J]. Software Tools for Technology Transfer (S1433-2787), 2004, 6(1): 189-205.

[7] F Wang. Symbolic parametric safety analysis of linear hybrid systems with BDD-like data-structures [J]. IEEE Transactions on Software Engineering (S0098-5589), 2005, 31(1): 38-51.

[8] S Umeno. Event order abstraction for parametric real-time system verification [C]// Proceedings of the 7th ACM International Conference on Embedded Software, Atlanta, GA, United States, Oct. 2008. USA: ACM, 2008: 1-10.

[9] D K Kaynar, N Lynch, R Segala, F Vaandrager. Timed I/O automata: a mathematical framework for modeling and analyzing real-time systems [C]// Proceeding of the 24th IEEE Int'l Real-Time Systems Symposium, Cancun, Mexico, Dec. 2003. USA: IEEE, 2003: 166-177.

[10] T A Henzinger, Z Manna, A Pnueli. Timed transition systems [M]// Real Time: Theory in Practice, Lecture Notes in Computer Science 600. Germany: Springer-Verlag, 1992: 226-251.

[11] 张广胜, 蒋昌俊, 沙静, 孙萍. 基于代价时间 Petri 网的合同网模型研究[J]. 系统仿真学报, 20(20): 5438-5445, 2008.

[12] N C Audsley, A Burns, M F Richardson, A J Wellings. STRESS: a simulator for hard real-time systems [J]. Software-Practice and Experience (S1097-024X), 1994, 24(6): 534-564.

[13] M F Storch, J S Liu. DRTSS: a simulation framework for complex real-time systems [C]// Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium, Boston, MA, Unite States, Jun. 1996. USA: IEEE, 1996: 160-189.

[14] A Wall, J Andersson, C Norström. Probabilistic simulation-based analysis of complex real-time systems [C]// Proceedings of the 6th IEEE International Symposium on Object-oriented Real-time distributed Computing, Hakodate, Hokkaido, Japan, May 2003. USA: IEEE, 2003: 257-266.

[15] J G Huselius, J Andersson. Model synthesis for realtime systems [C]// Proceedings of the 9th European Conference on Software Maintenance and Reengineering. Manchester, UK. USA: IEEE, 2005: 52-60.

[16] J Andersson, J Huselius, C Norström, Anders Wall. Extracting simulation models from complex embedded real-time systems [C]// Proceedings of the International Conference on Software Engineering Advances. Tahiti, French Polynesia. USA: IEEE, 2006: 7-17.

[17] J Andersson, A Wall, C Norström. A framework for analysis of timing and resource utilization targeting complex embedded systems [M]// ARTES - A Network for Real-Time research and graduate Education in Sweden, Editor: Hans Hansson. Sweden: Uppsala University, 2006: 297-329.

[18] C L Liu, J W Layland. Scheduling algorithms for multiprogramming in a hard real-time environment [J]. Journal of ACM (S0004-5411), 1973, 20(1): 46-61.

[19] Y Lei, R H Carver. Reachability testing of concurrent programs [J]. IEEE Transactions on Software Engineering (S0098-5589), 2006, 32(6): 382-403.

[20] P Montesinos, L Ceze, J Torrellas. DeLorean: recording and deterministically replaying shared-Memory multiprocessor execution efficiently [C]// Proceedings of the 35th International Symposium on Computer Architecture. Beijing, China. USA: ACM, 2008: 289-300.

[21] G Behrmann, J Hakansson, M Hendriks. UPPAAL 4.0 [C]// Proceedings of Third International Conference on Quantitative Evaluation of Systems. Riverside, CA, USA: IEEE, 2006: 125-126.

[22] Xilinx ML505/ML506/ML507 Evaluation Platform User Guide [R]. USA: Xilinx, Inc. Nov., 2008.

[23] J J Labrosse. MicroC/OS-II The Real-time kernel (Second Edition) [M]. China: CMP Books, 2002.