实时编程语言 RTS/Java 设计

桂先洲, 黄卫东
(国防科技大学航天学院, 湖南长沙 410073)

摘 要: 实时编程语言是有时间约束的系统描述语言. 已有语言在时态机制方面存在不同程度的缺点. 因此, 本文提出了基于 Java 的实时语言 RTS/Java, 用于解决 TGM 模型到 RT-Actor 模型的映射. 文章详细定义了 RTS/Java 类和方法及其约束, 设计了方法的激发机制. 此激发机制解决了 Allen 教授提出的时态约束. RTS/Java 语言支持多前端资源共享, 也成功地描述了三星实时侦察仿真系统.

关键词: 实时任务; 时态约束; Java; 建模语言
中图分类号: TP316 文献标识码: A 文章编号: 0372-2112 (2002) 02-0224-05

Design of Real-Time Programming Language RTS/Java

GUI Xiang-zhou, HUANG Wei-dong
(*School of Aerospace, National University of Defense & Technology, Changsha, Hunan 410073, China*)

Abstract: The Real-Time Programming language is considered as system description language with temporal constraints. There are shortcomings in the temporal mechanism of existed language. So, The paper presents the RTS/Java Real-Time language based on Java to solve the mapping from TGM model to RT-Actor model. The language defines the RTS/Java class, method and their constraints, and designs the method innovation mechanism. The mechanism has solved Allen's temporal constraints. RTS/Java language succeeds in describing the real-time simulation system of three surveillance satellites. RTS/Java features: (1) explicitly expressing temporal constraint, (2) temporal constructs illustrating task's period, (3) temporal logic independent of computer hardware, (4) the pure Object-Oriented style, multi-front-end sharing resources.

Key words: real-time tasks; temporal constrains; Java; modeling language

1 引言

迄今为止, 实时编程语言分四类. (1) 汇编语言. 其缺点是: 程序难编制, 难修改, 难重用, 难测试. (2) Ada、Modular-2 等. 其语言可保证计算逻辑结果的正确性. 但时态逻辑依赖于硬件配置. (3) Real-Time Euclid 语言[1] 等. 它提供了时间构造用于静态分析时间约束, 但时间构造不灵活[1,2]. (4) Esterel 语言等. 它显示提供时间约束, 对过程执行规定死限(deadline). 其问题是缺乏编译分析和预测性. 在程序设计风格上是属于模块化设计. RTS/Java 语言是 TGM 模型[3] 到 RT-Actor 模型的映射, 是基于 Real-Time Java™ 的实时描述语言. 它吸收 Real-Time Euclid 的构造, Esterel 时间约束表达式, 借鉴 RTM 语言[11] 的部分保留字.

TGM 模型是新的实时系统描述模型之一. TGM 模型的发展离不开文献[3~6] 作出的贡献. 论文[7, 8] 发展了和形式化定义了 TGM 模型. 文献[9] 给出了 TGM 的理论基础. Gul Agha 教授创建了 Actor 模型[10]. 目的是为了解决开放分布式系统中组件组态和推理. RT-Actor 模型是基于 Actor 模型的实时系统执行模型.

2 RTS/Java 语言的设计目的

2.1 RTS/Java 是支持 TGM 模型的编程设计语言

RTS/Java 是 TGM 模型到 RT-Actor 模型的桥梁, 也是两个模型的映射. 可见下图 1.

图 1 模型的映射关系

2.2 Real-Time Java 表达 RT-Actor 模型

1998 年 5 月 IDC 报告: 用 Java 研发项目比 C++ 平均节约 25% 的成本. Real-Time Jave™ 的迅速发展, 为 Java 语言在实时系统提供展示的机会. 先期的研究成果[12] 就是用 Java 描述 Actor 模型. 因此, RTS/Java 语言利用 Real-time Java 实现 RT-Actor 模型.

RTS/Java 在程序设计风格上遵循 Java 风格, RTS/Java = Real-Time Java + 保留字 + 实时构造. 实时构造可解决的问题有: (1) 监视超时的执行方法; (2) 定义方法体内出现的周期循环或循环; (3) 显式传递方法的时间约束; (4) 定义 RTS 类和方法.

收稿日期: 2000-12-07; 修回日期: 2001-07-13
基金项目: 国家自然科学基金(No. 60073003), 国防预研项目(No. 41320070104)

3 RTS/Java 类和方法分析

RTS/Java 的设计取决于 RTS 类、方法的设计. RTS 类要求: (1) 保证一个 RTS 类实现 RT-Actor 模型中的一类 Actor. (2) 保证类中某些方法共享对象公共状态.

RTS 类包含两种方法; (1) 公用 RTS 方法(简称为 RTS 方法). (2) 私用非 RTS 方法. 每个 RTS 方法实现 RT-Actor 模型中 Actor 方法. RTS 类中定义的方法与 TGM 模型中的任务一一对应. 从 TGM 模型的观点上看, 私用非 RTS 方法是代表任务中方法的一个辅助部分, 常用于小颗粒和可预测执行时间的操作, 例如, 数据从一种格式转换成为另一种格式.

非 RTS 对象中的方法不能调用 RTS 对象中的任何方法. 这种限制确保对象间所有消息是由于 RTS 方法产生. 标准库中的方法也被认为是非 RTS 方法.

RTS 对象的构造子(constructors)和 RTS 对象的去造子(destructors)被认为是非 RTS 方法. 它们可重载或重定义.

例 **1**　RTS 方法和非 RTS 方法. 对象 clock 时钟有下列 RTS 方法 Start、stop 和 read 方法. 每个方法有较短的死限. Start 启动表滴答; stop 停表并返回当前值; read 返回当前的值并允许 clock 时钟继续执行. stop 和 read 返回的浮点值是当前已过的时间(ms). Clock 必须有浮点类型的永久变量, 它纪录上次表启动时获得的时间积累.

Clock 必须定义非 RTS 方法 get_time 调用库方法, 并且转换收到的当前时间.

4 RTS 类的约束

为了确保 RTS/Java 语言支持不同地址空间的多用户共享, 我们必须对 RTS/Java 类进行约束. RTS 类与 Java 类有两方面的区别.

4.1 RTS 类的实例是一个独立的对象

不同对象可定位于不同的地址空间. 对象之间的通讯只能经过消息传递. 为了保证 RTS 对象间的地址空间的独立性、需要对类和类的使用施加三个限制. (1) 一个类应没有静态的成员变量. 由于静态成员变量在类所有实例中是全局的, 它们在类的实例之间共享内存. (2) 一个类应没有任何公共数据成员变量. (3) 一个类不能有永久友好类或非 RTS 友好类. 这是因为: 友好类的私用成员变量象公共的一样, 总是假定两个类共享同一地址空间. 非永久对象可以是友好的, 这是因为哪样的对象可以示例在同一主机上.

Java.io 包中的 InputStream、OutputStream 等类可作为友好类. 这是因为它在每个地址空间中都有一个独立版本.

4.2 每个 RTS 类实际上是由两个类组成

考虑到 TGM 模型可描述分布式实时系统. 一个实时任务可被不同的节点使用, 而一个 RTS 类用于表示一个或多个实时任务. 因此, 将 RTS 类分成两类: (1) 前端类(front end); (2) 服务类(server end).

定义 **1**　前端是系统中各种实体的接口. 它还负责 control_link 表的创建、管理和消息传递. 对象的前端使用普通 Java 构造子创建, 去造子消除.

定义 **2**　服务端实现 TGM 模型中任务功能. 对象的服务端使用 RTS/Java 语言的 create() 构造子创建、delete() 去造子删除.

定义 **3**　Control_link 列表由五元组形式递归定义:
(source, destination, ready_time, deadline, Control_link(s))

source 和 destination 给出了消息的发送者和接收者. Ready_time 就绪时间和 deadline 死限是约束消息接收者的就绪时间和死限. Control_link 控制链的取值可以是 none 和 passive. none 表示消息不携带任何信息. passive 表示信息出现在另一个消息中. Control_link 控制链可由多个 Control_link 控制链列表组成.

前端和服务端不一定在相同的地址空间内. RTS 类可有多个前端与一个服务端对应. 每个前端分布在不同主机上. 通常, 在每个主机内的地址空间中创建前端. 在前端地址空间里, 前端可调用其对象中的方法. 前端类和服务端类的约束 BNF 范式可见文献[8].

例 **2**　多前端问题. 现有两个主机, 它们分别是: tweedledum 和 tweedledee. Clock 类的实例是 Timer. RTS 运行系统将 Timer 的服务端示例在 tweedledum 主机上. Timer 前端可分别示例在两个主机上. 因此, 在 tweedledum 和 tweedledee 上的对象都可以引用 Timer, 并且, 它们分别调用各自的主机上的前端. 若在 tweedledee 主机无对象使用 Timer 前端, 那么在 tweedledee 上的 Timer 前端可不再示例.

5 RTS 方法的约束

首先定义 RTS 方法的 BNF 范式:

```
RTS_method_definition ::=
    < Return_type> < Method_name> (< Parameter_list> )
      { < Method_body> retcon ( < Parameter> ); };
Return_type     ::= public < Type> | private < Type>
  Public 用于描述 RTS 方法, private 用于描述非 RTS 方法.
Method_name  ::=  < Identifier>
Parameter_list  ::=  < Name_type> {, < Name_type> }
Name_type: ::= < Type> < Identifier> | < Virt_type> | < Identifier> | ε
Type: ::= float| double| short| byte| char| boolean| < Class_name> | ε
Virt_type: ::= virt float| virt double| virt short| virt byte|
          virt char| virt boolean | virt < Class_name>
  vir 保留字用于描述新消息还未达到, 而使用上次消息值的概念.
Parameter     ::=  < Identifier> | < constant> | ε
Method_body  ::=  { < Local_definition> } { < RTS_statement> }
Local_definition  ::=  < Type> < Identifier>
RTS_statement  ::= < Java_statement> | < RTS_methodcall > |
          < Method_construct>
    RTS_methodcall: ::=  < Method_name> ( [< Real_parameter> ] )
          [< Temporal_keyword> < Integer> ]
  < Temporal_keyword> < Integer> 描述方法的时态约束.
Method_name  ::= < Identifier>
Real_parameter  ::= < RTS_term> {, < RTS_term> }
RTS_term        ::= ε| < Variable> | < constant> | < RTS_methodcall>
Method_construct ::= MAND_RETVAL < constant> |
```

```
                OPT_ RETVAL < Variable> |
            MAND_ RETVAL < Variable> | OPT_ RETVAL < constant> |
            BEGIN_ OPT_ PART | < Periodic_ part > |
        < Create_ member> | < Delete_ member> | < RTS_ directive>
RTS_ directive : : = SYN ([ < Variable> = ] < RTS_ methodcall> ,
                [ < Variable> = ] < RTS_ methodcall> ,
                { [ < Variable> = ] < RTS_ methodcall> })
```

RTS 方法中 parameter_ list 的参数可在前面使用 virt 关键字. RTS 方法的范式定义表明: RTS 方法不同于传统的 Java 方法. 最重要区别有以下几个方面:

(1) RTS 方法要显式地使用 create( ) 构造子建立服务端, 显式地使用 delete( ) 去造子删除服务端;

```
Create_ member : : = create ( < Parameter_ list> )
                [ < Location_ hints> ]
Parameter_ list : : = < Term> { , < Term> }
Term : : = ε | < variable> | < constant>
Location_ hints : : = CO_ LOCATE < Obj_ name> |
                DISJOINT < Obj_ name_ list> |
            CO_ LOCATE < Obj_ name> DISJOINT < Obj_ name_ list>
Obj_ name_ list : : = < Obj_ name> { , < Obj_ name> }
Obj_ name : : = < Identifier>
```

delete( ) 方法用于删除 RTS 类服务端的实例. 其 BNF 范式形式:

```
Delete_ member : : = delete ( < Parameter_ list> )
```

(2) 在 RTS 方法中, 最后必须有 retcon( ) 方法. RTS 方法的中间也可使用此方法, 它表示消息的发送;

(3) RTS 方法显式地定义时间约束;

(4) 时间约束可作为方法请求的一部分显式传送, 省缺时运行系统设置. 方法的时间约束对应于任务自身的约束.

在 RTS 中, 显式使用 create( ) 构造子的原因是为了避免 Java 的常规(normal) 规则.

6 RTS 方法激发机制

RTS 方法的激发(Innovation) 是为了解决 RTS 方法之间的消息传递问题. 它不同于传统的调用和返回机制.

定义 **3** RTS 方法激发是一个独立的单元: 激发方法. 此方法接受调用方法的消息(含有时态、数据和控制信息) 向另一个被调方法发送消息. 调用方法可自由地做任何事情, 当然也包括等待别人的消息. 而被调方法, 原则上与调用方法无关. 被调方法返回的执行结果不一定是发送给调用方法的消息.

为了传递 RTS 方法的消息, 我们参考文献[11] 的 Return to future, 借鉴 continuation 方式[13], 专门设计了激发方法机制(retcon 机制) 进行消息的传递. 它的目的是允许 RTS 方法返回某些值给后续的方法, 而且传递继承的时间约束. 可见图 2.

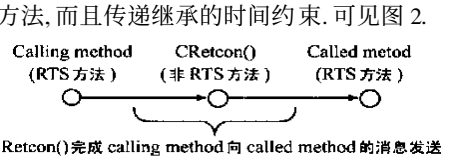

图 2

激发方法机制 retcon 解决了 J. F. Allen 教授在文献[14] 中提出的任务之间时态约束. 即: 任务执行的七大时态约束.

消息中的重要成员是 Control_ link, 它是消息传递过程中的控制策略. retcon 机制使用 retcon( ) 方法来实现, 此方法将 Control_ link 列表和返回值作为参数, 进行消息的发送.

传送到 retcon( ) 的 Control_ link 列表也可由运行系统创建. 当 retcon( ) 方法执行时, retcon( ) 机制将要检查收到的 Control_ link 参数, 它们是: (1) 消息中的 Control_ link 列表; (2) 消息中的源; (3) 目标; (4) 就绪时间; (5) 死限; (6) 运行系统给出的计算号;

例 **3** 方法 foo 的绝对死限为 100, 000, 它含有下列代码.

$$x = f(g(3), h(2));$$
$$j(k(4)) \quad \text{DEADLINE } 1000;$$
$$l();$$

g 和 h 发送它们的值到 f, 分别作为 f 的第一, 第二个输入参数, f 计算的结果放入变量 x 中. RTS 运行系统构造的 control_ link 列表. 可见图 3.

通常, 程序员定义的时间约束必须在编译时确定. 若程序员没有显式规定时间约束, 被调方法的时间约束将继承调用方法的时间约束.

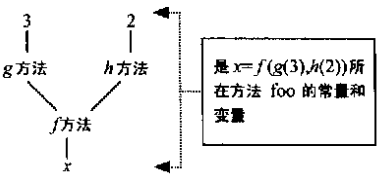

图 3

当第一行被执行, 简单的调度器将赋 f, g 和 h 的死限为 1000. 较复杂的调度器可考虑 f 花费的执行时间, 考虑 g 和 h 可能并行执行.

假定第二行在 30, 000 时执行, 则 j 的死限为 31, 000, 复杂调度器将给 k 的死限是: 31000– (j 的最大执行时间+ 上下文切换时间+ 运行系统的开销时间).

最后, l 不继承死限, 这是因为 l 的死限既没有规定, 也不给 foo 方法返回任何的值. 当 l 执行自己的 retcon( ) 时, 方法 foo 无事发生.

7 RTS/ Java 中方法构造设计

方法体内的构造是: BEGIN_ OPT_ PART、OPT_ RETVAL、MAND_ RETVAL、周期部分、SYN 等. 更为详细的设计可见文[8].

7.1 方法中选择部分开始的构造 BEGIN_ OPT_ PART

其功能表示方法中选择部分的开始位置. 选择部分的开始不能在任何类型的循环或条件分枝内.

7.2 选择部分返回构造 OPT_ RETVAL retval

retval 是一个返回表达式. 其功能是: 在方法选择部分的任何点上可使用关键字 OPT_ RETVAL. 当选择部分违反时间约束而不能完成执行时, 它返回一个值. 此值的类型必须匹配方法定义的返回类型.

7.3 必要部分返回构造 MAND_ RETVAL retval.

retval 是一个返回表达式. 此构造功能是: 方法一旦执行就被认为是进入必要部分. 因此, 无须有必要部分开始的构

造.MAND_RETVAL 的作用类似于 retcon( ). 当必要部分违反时间约束而不能完成执行时,它返回一个值.

例 4　选择部分. 下面的代码表明上面关键字的使用方法:

```
MAND_RETVAL- 1 // Return standard Java Sign for failure
    ……      // bunch of code
int calc = 0;            // used for calculating results
BEGIN_OPT_PART
for (int i= 0; i< 10; i + + =
{     ……      // a bunch of code which determines
              //a new value for calc
  OPT_RETVAL calc   // update value to be returned
}
retcon (calc);
```

方法一开始就设置必要部分的初始返回值为出错值“-1”,以备方法执行失败时使用. 当方法的必要部分出错,接收方法可做相应的处理. 方法中的主循环是选择部分,循环次数为十次. 每次 *calc* 的值重新计算.

通常,RTS 方法的执行次序是运行系统确定的. 为了满足所有时间约束,运行系统的调度算法至关重要. 永久对象的方法必须执行. 它的结果可能不被使用. 但它们潜在修改了永久对象的状态.

7.4　周期部分构造

周期部分的 BNF 范式:

```
Periodic_part  ::=   < Loop_construct>  PERIOD time_1
              [PHASE time_2 ][  VARIATION time_3 ]
            < RTS_statement>
Loop_construct  ::=   while (< expression> ) |
          for ( < init_statement> , < expr_1> , < expr_2>  )
    time_x          ::=   以 ms 为单位的时间常数.
    Expression      ::=   Java 语言中表达式.
    expr_1, expr_2  ::=   Java 的表达式.
    init_statement  ::=   Java 中 for 循环的初始语句.
```

周期部分用于定义方法内的周期循环. PERIOD、PHASE 和 VARIATION 是时间约束,它们的值在编译时必须确定. 周期部分的相位相对于系统起动. 给定的方法可含有多个周期部分,但周期部分之间不可嵌套,也不能包含在另一结构中.

例 5　基于循环的周期部分的例子:

```
for (  int i= 0;  i < = 100;  i+ + =
        PERIOD 1000 VARIATION 500
{   foo. bar( );
        x= foo. zip ( i);
    }
```

循环每 1000±500ms 执行一次. 每次调用 foo. *bar* 和 foo. *zip*. 注意到前者无死限. 由于后者要返回一个值,它必须在循环中当前重复结束前完成. 假定当前开始时间为 100000,则 foo. *zip* 必须在 101500 之前完成执行(当前周期结束的上界). 在循环的迭代中,这两个方法被调用的最早时刻是时间 100500(当前周期结束的下界).

7.5　同步伪方法构造

为了给方法调用提供同步,RTS 提供了一个同步伪方法(pseudo-method) SYN( ). 同步伪方法的哑元可以是任何其它的方法名.

例 6　同步伪方法.

```
SYN ( f( )  DEADLINE 4000, g( ) DEADLINE 8000 )
        READY_TIME 1000;
```

假定这个语句在 10,000 时间到达,则 f 有死限 14000, g 有死限为 18000, f 和 g 两者的就绪时间为 11000.

8　Main 方法的构造设计

RTS 程序可有多个 main( ) 主方法. 每个 main( ) 主方法对应分离的地址空间,通常一个 main( ) 主方法对应着一个主机或虚拟主机. main( ) 主方法还需使用 RTS 方法的构造,如:时间约束等. RTS main( ) 主方法含有自己的构造,它们用于处理系统时钟的起动、停止和同步. 这些构造详细的定义可见文[8].

8.1　启动时钟构造 **START_CLOCK[time]**

time 表示某个时刻. 此构造功能是:当程序到达 START_CLOCK 时,RTS 运行系统挂起 main( ) 主方法的执行,启动 RTS 内部时钟. 当规定的 time 时间到达时,主机再恢复 main( ) 主方法的执行. 若 time 省缺,RTS 系统不起动内部的时钟,继续 main( ) 主方法的执行.

8.2　暂停时钟构造 **SUSPEND_CLOCK:**

8.3　恢复时钟构造 **RESUME CLOCK time**

time 表示某个时刻. 此构造功能是:RESUME_CLOCK 恢复 SUSPEND_CLOCK 构造挂起的系统时钟.

8.4　终止同步时钟构造 **FINISH_CLOCK time**

time 表示时刻. 此构造功能是:当程序到达 FINISH_CLOCK 时,RTS 运行系统将在 time 时间点上停止系统时钟. 在此构造指定的系统时钟还没有到达时,方法可继续执行. FINISH_CLOCK 构造要求运行系统忽视任何请求,忽视此点后的任何方法的时间约束.

8.5　同步启动时钟构造 **START_SYN_CLOCK time**

time 表示某个时刻. 此构造功能是:START_SYN_CLOCK 是解决不同 main( ) 主方法的同步问题. 当 main( ) 主方法的所有实例到达自己的 START_SYN_CLOCK 构造语句时. 常驻在每个主机上的运行系统及时地、同步地起动它们内部的时钟,同时恢复各自局部 main( ) 主方法的执行.

8.6　暂停同步时钟构造 **SUSPEND_SYN_CLOCK time**

8.7 恢复同步时钟构造 **RESUME_SYN_CLOCK [time]**

此构造的功能是:RESUME_SYN_CLOCK 在同一时间恢复多个 main( ) 主方法执行. RESUME_SYN_CLOCK 同步启动已经停止运行的、含有 RESUME_SYN_CLOCK 的所有 main( ) 主方法. 若 time 省缺,则它同步已经停止运行的、含有 SUSPEND_CLOCK 的所有 main( ) 主方法,然后重新同步启动它们执行.

例 7:使用 START_CLOCK 和 FINISH_CLOCK 的 main( ) 主方法,它可以在任何在主机上运行.

```
class1_name      obj1 ;
```

```
    //creates object of class 1, including front-end
class2_name        obj2 ;
    boolean stop_cond        //stop condition
{
  stop_cond = FALSE ;
  obj1.create () ;          //creates server- end of obj1
  START_CLOCK               //starts the system
  obj2.create () ;          //creates server- end of obj2
  int          x, y;
  while (stop_cond) PERIOD 120000
        //starts a loop with a period of 120 seconds
  {
  SYN ( x = obj1.method1(x); y = obj2.method1 (y) ) ;
    System.out.println("x=" + x ) ;
    System.out.println("y=" + y) ;
    //some code conditionally modifies stop_cond.
  };
  obj2.delete() ;           //deletes server- end of obj2
  FINISH_CLOCK              //stops the system clock
  obj1.method2 () ;         //prints out object's state
  obj1.delete() ;           //deletes server end of obj1
}
```

注意到:在 *obj*1 的被创建和消除时,系统时钟没有运行.在 *obj*2 的创建和消除时,系统时钟正在运行.还注意到:两个永久对象作为变量说明.这种说明实际上是调用了前端类的构造子.

9 结束语

本文分析了现阶段不同种类的实时语言,指出了各自存在的问题.文章充分吸取不同种类实时语言的优点,提出了 RTS/Java 编程语言规范.RTS/Java 用于解决 TGM 模型和 RT-Actor 模型之间的映射问题,而且有显著的优点:(1)提供时态保留字,保证时态约束的直接表达;(2)提供功能/时态构造,更能刻画实时任务的周期性;(3)时态逻辑正确性不直接依赖于硬件设施;(4)纯面向对象设计风格;(5)支持多前端机共享资源.它的实现依赖于虚拟机结构.

文章将类分成前端类和服务类,更体现多机实时性能.给出了类和方法的约束.RTS/Java 成功地描述了分布式三星实时侦察仿真系统[15].此语言为实时系统的设计、仿真和分析提供了新的手段.

参考文献:

[1] Eugene Kligeman, Alexader D Stoyenko. Real-time euclid: A language for reliable real-time systems [J]. IEEE Trans. On Software Engineering, 1986, SE-12(9): 941- 949.

[2] Kevin B Kenny, Kwei-Jay Lin. Building flexible real-time systems using the flex language [J]. IEEE Computer, May 1991, 24(5): 70- 78.

[3] Silberman A, T Marlowe. A task graph model for design and implemerrtation of real-time system [A]. Proceedings of the 1996 IEEE on Engineering Complex System [C], NEW YORK: IEEE Press, Montreal, 1996, 432- 441.

[4] A K Mok, S Sutanthavibul. Modeling and scheduling of dataflow real-time systems [A]. Proceedings of the 1985 Real-Time Systems Symporsium [C], San Diego, 1985: 178- 187.

[5] R Gupta, M Spezialetti. A compact task graph representation for real-time scheduling [A]. Real-Time Systems [C], 1998, 13(7): 13- 21.

[6] A Grimshaw. Mentat: An Object-oriented macro Data Flow System [D]. Ph. D. dissertation, University of Illinois, Urbana, Illinois, 1988.

[7] Gui X Z. Design of the model and programming language for complex real-time systems [D]. National University of Defense Technology, 1999.

[8] 桂先洲, 等. TGM 模型的 RTL 形式化定义 [J]. 计算机仿真, 2001, 18(2): 37- 40.

[9] 桂先洲, 等. TGM 任务图模型和其理论基础 [J]. 计算机工程与科学, 2001, 23(1): 61- 64.

[10] Gul A Agha, Ian A Mason, Scott F Smith, Carolyn L Talcott. A foundrtion for actor computation [J]. J Functional Programming 1, Cambridge University Press. Jan. 1993: 1- 15.

[11] Ami Abraham Silberman. RTM-Design and Implementation [D]. Ph. D. thesis, University of Illinois at Urbanar Champaign, 1997.

[12] Brian Nielson, Gul Agha. Semantics for an actor-based real-time larrguage [R]. Department of Computer Science, 1304 W. Spring field Avrenue, Urbana, IL 61801, USA.

[13] Abelson H, G J Sussman, J Sussman. Structure and Interpretation of Computer Programs [M]. The MIT Press, Cambridge Mass., 1985.

[14] J F Allen. Maintaining knowledge about temporal intervals [J]. Comm. ACM, 1983, 26(11): 832- 843.

[15] 桂先洲. 实时卫星侦察的TGM 模型和验证 [R]. 技术报告, 2000 年.

作者简介:

桂先洲 男. 1961 年 12 月出生于湖北麻城.国防科大航天学院副教授.毕业于国防科大计算机专业和自动控制专业,博士.参加两代银河仿真实时计算机的设计和研制工作,此两项工程都获得国家科技进步一等奖.发表的文章曾被 EI、ISTP、SA 检索.研究领域:实时系统,计算机建模,CSCW,仿真理论基础,仿真语言等. Email: guixz@263.net

黄卫东 男. 1969 年出生于山东海阳.国防科大博士研究生,总装备部航天局工作.研究领域:图像处理,实时系统,飞行器总体设计等.