TFSP:一种分布式实时系统的形式化描述工具*)

叶俊民[1,2,3] 王振宇[1,3] 陈 利[2,3] 赵 恒[1,3]

(哈尔滨工程大学计算机科学与技术学院 哈尔滨150001)[1] (华中师范大学计算机科学系 武汉430079)[2]

(武汉大学软件工程国家重点实验室 武汉430072)[3]

TFSP:A Formalized Description Tool of Distributed Real-Time System

YE Jun-Min[1,2,3] WANG Zhen-Yu[1,3] CHEN Li[2,3] ZHAO Heng[1,3]

(Computer Science and Tech. College, Harbin Engineering University, Harbin 150001)[1]

(Department of Computer Science, Central China Normal University, Wuhan 430079)[2]

(State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072)[3]

Abstract Distributed Real Time Systems (DRTS) have very broad applications in space navigation, nuclear reaction, military affairs and industry department where the security and reliability requirement of the DRTS is very high. Thus, how to develop correct DRTS application systems is of vital importance. In this paper, first a formalized notation system -- Timed Finite State Processes (TFSP) is proposed to describe the complex dynamic behaviors of DRTS, then we describe a distributed real-time medical treatment system by Darwin Architecture language and TFSP.

Keywords DRTS, TFSP, Software architecture

分布式实时系统在宇航、核反应、军事和工业等部门应用极为广泛,这些领域对系统本身的安全性和可靠性要求极高,因此如何开发出正确的分布式实时应用系统是一个关键问题。为了对分布式实时系统进行建模,我们提出了 TFSP (Timed Finite State Processes),即时间有限状态进程。TFSP 扩展了 Magee 等人提出的 FSP(Finite State Processes)[1]。

1 基本概念

1.1 软件体系结构

虽然 Garlan 和 Shaw[2],以及 Perry 和 Wolf[3]等人对什么是软件体系结构作出了明确的回答,但是软件体系结构目前尚无统一的定义。

一种可以接受的定义是,软件体系结构是程序/系统各构件的结构以及它们之间的内部关系和指导它们的设计并随时间而演变的原理和方针。从结构模式观点上看,软件体系结构包括软件构件、构件之间的联系和系统构造、方式、约束、语义、分析、属性、基本原理和系统需求。这一观点在体系结构描述语言(ADL)得到体现。

我们认为,对软件体系结构的完整描述应包括静态体系结构和动态体系结构两个方面。分布式实时系统体系结构描述也是如此。

1.2 Darwin 软件体系结构描述语言

Darwin 是一种分布式系统体系结构描述语言[3]。Darwin 将程序用层次化的构件来描述,其构件又分为原子构件和组合构件两类。构件之间通过访问服务(包括提供服务和需要服务)的绑定来进行交互。Darwin 描述了分布式实时系统体系结构的静态方面的信息,如构件、连接(绑定)、配置信息。

1.3 时间有限状态进程

TFSP 的非形式化定义由基本进程、组合进程、公共操作、特性和必要活动五个部分组成。基本进程包括活动前缀(→)、选择(|)、卫式活动(when)和字母表的扩展(+)。组合进程包括并行组合(‖)、重复符(forall)、进程标识(:)、优先级高(≪)和优先级低(≫)。公共操作包括条件(if then else)、再标识(/)、隐藏(\)和接口(@)。必要活动通过引入专用活动字母表的方式完成对以下内容的支持,其一是支持对时间超时(timeout)和延迟(delay)的描述。其二是引入活动 settime,以支持一个或者多个时钟变量的置零操作。其三是引入发送(send)和接收(receive)处理,以支持对分布式系统中的消息处理。TFSP 用于描述分布式实时系统体系结构的动态方面。

2 TFSP 基本语法的 BNF 形式

活动 a 的字母表表示 a 的合法活动名称,它与用户的应用有关。除了与用户相关的活动外,活动 a 至少包括以下基本活动{delay, timeout, send, receive, setime},即 talp(a)={delay, timeout, send, receive, setime}∪talp(与 a 的应用有关的活动)。Talp 表示求对应活动的函数。P* 表示有一个或者若干个 P。规定用首字母大写的方式来表示进程,用首字母小写的方式来表示活动。

定义1 TFSP 进程的 BNF 范式定义如下:

```
PROCESS::=P|Q
P::=
    (1)|STOP                                    //终止/死锁进程
    (2)|ERROR                                   //出错进程
    (3)|(a→p)                                   //活动前缀
    (4)|(when (B) a→P)                          //卫式活动
    (5)|(a1→P1|a2→P2)                           //选择活动
    (6)|P+A                                     //字母表扩展
    (7)|X                                       //进程标识符
    (8)|(delay (T) a→P|(delay(T))P              //延迟进程
    (9)|(do timeout (T) then a→Q else a→P)      //超时进程
```

*)本课题得到国防科技预研基金(41350601)和武汉大学软件工程国家重点实验室开放基金(SKL(4)020)资助。

```
(10)|(send (data|message_list|ack_signal)to P*)→P2        //发送进程
(11)|(receive (data|message_list|ack_signal) from P1)→P2  //接收进程
(12)|(setime 0 to P*)→P                                   //将零个或者若干个 P 的时钟之值置0
Q::=
(13)|P                                                    //基本进程
(14)|(Q1‖Q2)                                              //并行组合进程
(15)|(L::Q)                                               //进程标号
(16)|Q/R                                                  //进程再标号
(17)|Q\A                                                  //进程隐藏
(18)|‖C=(P||Q)≪{a1,…,an}                                 //高优先级
(19)|‖C=(P||Q)≫{a1,…,an}                                 //低优先级
```

对定义1的语义如下。在基本进程定义中,(1)和(2)表示两个常量进程,分别为终止进程和错误进程。注意:终止进程常常被用作死锁进程。(3)表示活动前缀。(4)表示卫式条件活动。(5)表示选择。(6)表示字母表扩展。(7)表示进程标识符。或者表示延迟一段时间片 T 后,后继行为由进程 P 表示。

(8)(delay(T)a→P|(delay(T))P 之意是,延迟时间片 T 之后,做动作 a,其后继行为由 P 描述。或者延迟一段时间 T 后,在 T 这一时间段片内,没有任何事件发生。

(9)(do timeout(T)then a→Q else a→P)之意是,表示如果进程在给定时间片 T 内,a 活动(事件)没有发生(即超时),则控制转给进程 Q;否则(动作完成而没有超时的话),其后继行为由 P 描述。

(10)(send(data |message_list | ack_signal)to P*)→P2 之意是,或者向进程 P*(*表示一个或者多个)发送数据或者消息(如果是向多个进程发送消息,这时的通信就是组通信),或者向 P* 发送应答消息,当动作完成之后,其后继行为由 P2 描述。

(11)(receive(data |message_list | ack_signal)from P1)→P2之意是,或者从进程 P1处接收数据或消息(可以是多个消息),或者从 P1处接收应答消息,当动作完成之后,其后继行为由 P2描述。依据 message_list | ack_signal 可以确定发送者 P1。

(12)(setime 0 to P*)→P 之意是,将一个或者若干个进程 P 的时钟之值置为0。

在组合进程定义中,(13)表示一个基本进程。

(14)表示进程之间的并行组合。(15)表示进程标号。(16)表示进程再标号。(17)表示进程隐藏。(18)表示高优先级。(19)表示低优先级。其中,(18)或(19)说明,进程 P||Q 中的活动{a1,…,an}的优先级要高于(低于)其它进程的活动。因此,进程 P||Q 在执行活动{a1,…,an}时,或者可以剥夺其它进程的执行权利,或者可被其它进程剥夺执行权利。

3 TRMCS 实例研究

3.1 TRMCS 问题描述

文[4]提出的一种基于分布式实时系统的远程医疗系统 TRMCS,如图1所示,该系统已经成为一种通用的形式化描述标准。下面,我们从体系结构的静态和动态两个方面来描述该问题。

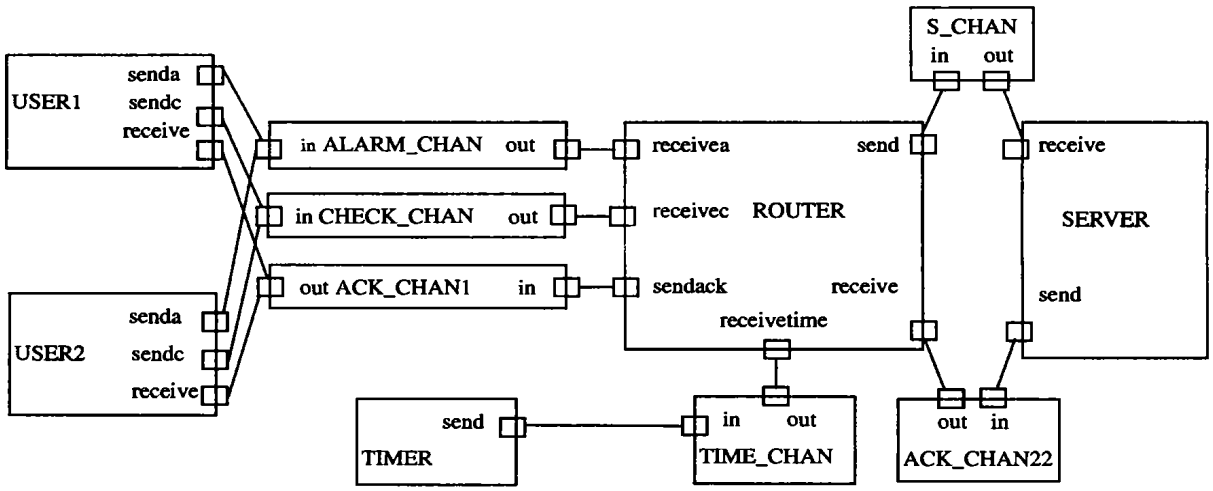

图1 TRMCS 问题的软件体系结构

3.2 TRMCS 的静态体系结构描述

利用 Darwin 体系结构描述语言[1],图1的 TRMCS 问题的软件体系结构描述如下:

```
component TRMCS{
partal
USER1.senda,USER2.senda:ALARMMESG;
USER1.sendc,USER2.sendc:ALARMMESG;
USER1.receive,USER2.receive:ACKMESG;
inst
USER1,USER2:USER;
ALARM_CHAN, CHECK_CHAN, ACK_CHAN1, S_CHAN,
ACK_CHAN2:CHANNEL;
TIME_CHAN:CHANNEL;
TIMER_INST:TIMER;
ROUTER_INST:ROUTER;
SERVER_INST:SEVER;
bind
USER1.senda--ALARM_CHAN.in;
USER1.sendc--CHECK_CHAN.in;
USER1.receive--ACK_CHAN1.out;
USER2.senda--ALARM_CHAN.in;
USER2.sendc--CHECK_CHAN.in;
USER2.receive--ACK_CHAN1.out;
ALARM_CHAN.out--ROUTER.receivea;
CHECH_CHAN.out--ROUTER.receivec;
ACK_CHAN1.in--ROUTER.sendack;
ROUTER.send--S_CHAN.in;
```

```
ROUTER. receive--ACK_CHAN2: out;
ROUTER. receivetime--TIME_CHAN. out;
TIMER. sen--TIME_CHAN. in;
S_CHAN. out--SERVER. receive;
ACK_CHAN. in--SERVER. send;
}
```

3.3 TRMCS 的动态体系结构描述

设 $T_i(i=1,\cdots,n)$是相关时间片。USER_ALARM 进程行为是发送一个报警消息，该消息通过通道 ALARM_CHAN 传递到 ROUTER 中，接着进程 USER_ALARM 等着接收通过通道 ALARM_CHAN 传来的 ROUTER 的应答消息。

如果等待应答消息超时，则给出错误信息，其后继行为由进程 USER_ALARM 确定；如果不超时则可成功接收应答消息，其后继行为由进程 USER_ALARM 确定。USER_ALARM 进程的具体表示如下：

```
USER_ALARM=
(send (alarm_message) to ALARM_CHAN → receive (alarm_ack)
    from ACK_CHAN→
    do timeout (T1) then receive (alarm_ack) → error → USER_
     ALARM
     elas receive (alarm_ack)→USER_ALARM).          (1)
```

如(1)式，超时部分的写法比较繁琐，故在此引入 this 指针的表示方法。this 表示“当前运行指针”，它与最近的一个 do 操作中的 then 和 else 相匹配的动作部分有关。在上下文明确的前提下，用 this 指针来简化超时部分繁琐的写法。因此，(1)式可以写成：

```
1)USER_ALARM=
  (send (alarm_message) to ALARM_CHAN → receive (alarm_
  ack) from ACK_CHAN→
    do timeout(T1) then this→error→USER_ALARM else this→
       USER_ALARM).                                 (2)
```

下面用 TFSP 来描述 TRMCS 的体系结构动态行为。结论如下：

```
2)USER_CHECK=
  (send(check_message)to CHECK_CHAN→receive (check_ack)
  from ACK_CHAN1→
    do time(T2)then this→error→USER_CHECK else this→US-
       ER_CHECK).
3)‖USER=(USER_ALARM‖USER_CHECK).
4)ROUTER_RECEIVEALARM=
  (receive(alarm_message)from ALARM_CHAN→do timeout(T3)
  then this→error→ROUTER_RECEIVEALARM
  else this→send (alarm_message) to S_CHAN→recevie (alarm_
  ack)from ACK_CHAN2→
  do timeout (T4) then this→error→ROUTER_RECEIVEALARM
  else this→send(alarm_ack) to ACK_CHAN1→ROUTER_RE-
  CEIVEALARM).
5)ROUTER_RECEIVECHECK=
  (receive(check_mssage)from CHECK_CHAN→do timeout (T5)
  then this→error→ROUTER_RECEIVECHECK
    else this→send(check_message)to S_CHAN→receive(check_
         ack)from ACK_CHAN2→
      do timeout (T6) then this → error → ROUTER → RE-
           CEIVECHECK else this → send (check_ack) to ACK_
           CHAN1→ROUTER_RECEIVECHECK).
6)‖ROUTER=([0…1]:ROUTER_RECEIVEALARM‖[0…1]:
  ROUTER_RECEIVECHECK).
7)SERVER_RECEIVEALARM=
  (receive(alarm_message)from S_CHAN→do time(T7) then this
  →error→SERVER_RECEIVEALARM
    else this→send (alarm_ack)to ACK_CHAN2→SERVER_RE-
         CEIVEALARM).
8)SERVER_RECEIVECHECK=
  (receive(check_message)from ROUTER→do timeout(T8)then
  this→error→SERVER_RECEIVECHECK
    else this→send(check_ack)to ACK_CHAN2→SERVER_RE-
         CEIVECH.
9) ‖ SERVER = (SERVER_RECEIVEALARM ‖ SERVER_RE-
   CEIVECHECK).
10) CHANNEL = (in→lose→CHANNEL | in→out→CHANNEL)
    @{in,out}.
11)TIMER=(send time to ROUTER→TIMER).
12)‖USER_ROUTER=(u[0…1]:USER‖r:ROUTER
   ‖sa[0…1]:SERVER_RECEIVEALARM‖
   sc[0…1]:SERVER_RECEIVECHECK‖t:TIMER)/
{
   u[0]. send(alarm_message)to ROUTER/r[0]. receive(alarm_
   message)from USER_ALARM,
   u[1]. send(alarm_message)to ROUTER/r[1]. receive(alarm_
   message)from USER_ALARM,
   r[0]. send(alarm_message)to SERVER/sa[0]. receive(alarm_
   message)from ROUTER,
   r[1]. send(alarm_mesage)to SERVER/sa[1]. receive(alarm_
   message)from ROUTER,
   sa[0]. send(alarm_ack)to ROUTER/r[0]. receive(alarm_ack)
   from SERVER,
   sa[1]. send(alarm_ack)to ROUTER/r[1]. receive(alarm_ack)
   from SERVER,
   r[0]. send(alarm_ack)to USER/u[0]. receive(alarm_ack)from
   ROUTER,
   r[1]. send(alarm_ack)to USER/u[1]. receive(alarm_ack)from
   ROUTER,
   u[0]. send(check_message)to ROUTER/r[0]receive(check_
   message)from USER_CHECK,
   u[1]. send(check_message)to ROUTER/r[1]. receive(check_
   message)from USER_CHECK,
   r[0]. send(check_message)to SERVER/sc[0]. receive(check_
   message)from ROUTER,
   r[1]. send(check_message)to SERVER/sc[1]. receive(check_
   message)from ROUTER,
   sc[0]. send(check_ack)to ROUTER/r[0]. receive(check_ack)
   from SERVER,
   sc[1]. send(check_ack)to ROUTER/r[1]. receive(check_ack)
   from SERVER,
   t. send time to ROUTER/r. recevie(time)from TIMER
}.
```

讨论

1)在 USER2向 CHECK_CHANNEL 发送 check 时，如果发现 CHECK_CHANNEL 正被 USER1所占用时，USER2就需要延迟一端时间后重发。这一行为可以描述如下：

```
USER2_CHECK=send(check_message)to CHECK_CHAN→when
(is_occupied) setime 0 to USER2_CHECK → delay (T) USER2_
CHECK.
```

2)对于 Darwin 语言而言，进一步的工作是将时间表示扩展到构件描述上。

3)提出 TFSP 不是目的，我们的目的是从基于 TFSP 模型的规格说明中导出测试计划和测试用例。这些内容将是我们进一步研究的方向。

参 考 文 献

1 Magee J, Kramer J. Concurrency -- State models & Java Programs. Wiley. 1999. 317～341

2 Shaw M, Garlan D. Software architecture: perspectives on an emerging discipline. Beijing: Tsinghua University Press, 1998, 5: 19～32

3 Perry D E, Wolf A L. Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes, 1992, 17 (4): 40～52

4 Bertolino A, Corradini F, Inverardi P, Muccini H. Deriving Test Plans from Architectural Descriptions. In: ACM Proc. Int. Conf. on Software Engineering (ICSE2001), June 2000. 220～229

5 Magee J, Kramer J, Giannakopoulou D. Behaviour Analysis of Software Architechure. In: Proc. First Working IFIP Conf. on Software Architecture (WICSA1), San Antonio, Texas, Feb. 1999. 35～49