DOI:10.7544/issn1000-1239.2015.20130722
52(1):167-176,2015

AADL 分级调度模型的分析与验证

符 宁[1] 杜承烈[1] 李建良[2] 刘志强[3] 彭 寒[4]
[1](西北工业大学计算机学院 西安 710072)
[2](西北农林科技大学信息工程学院 陕西杨凌 712100)
[3](西北工业大学软件学院 西安 710072)
[4](西安航空学院计算机工程系 西安 710077)
(funingg@163.com)

Analysis and Verification of AADL Hierarchical Schedulers

Fu Ning[1], Du Chenglie[1], Li Jianliang[2], Liu Zhiqiang[3], and Peng Han[4]
[1](*School of Computer Science, Northwestern Polytechnical University, Xi'an* 710072)
[2](*College of Information Engineering, Northwest A&F University, Yangling, Shaanxi* 712100)
[3](*School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an* 710072)
[4](*School of Computer Engineering, Xi'an Aeronautical University, Xi'an* 710077)

Abstract In the system based on a hierarchical scheduler, the processor is shared between several collaborative schedulers. Such schedulers are becoming more and more investigated and proposed in reallife applications. For example, the ARINC 653 international standard which defines programming interface for avionic real time operating systems provides such a kind of collaborative schedulers. This article focuses on the modeling and the schedulability analysis of hierarchical schedulers. We investigate the modeling of hierarchical schedulers with architecture analysis and design language (AADL). A model checking based method for analyzing the schedulability of AADL hierarchical schedulers is proposed. The AADL thread components and hierarchical schedulers are modeled as network of timed automatons. The schedulability is described as a set of temporal logic formulas. Then we use a model checker Uppaal to analyze and verify the schedulability of hierarchical schedulers. Our work shows that analyzing schedulability of AADL hierarchical schedulers by model checking is feasible. The method uses an exhaustive method to automate analyze the properties of a system by a model checker. Compared with related works, the proposed method produces more precise results.

Key words complex embedded real-time system; architecture analysis and design language (AADL); UPPAAL; schedulability analysis; model checking

摘 要 针对嵌入式系统体系结构分析设计语言(architecture analysis and design language, AADL)分级调度模型的分析问题,提出了基于模型检验的可调度性分析和验证方法.基于时间自动机理论,将AADL分级调度模型转换为时间自动机网络,将待验证性质描述为时序逻辑公式,通过模型检验工具对可调度性进行分析和验证.研究结果表明,使用模型检验方法来分析AADL分级调度模型的可调度性是可行的.相对其他方法而言,该方法利用了形式化方法的穷举性来分析系统的性质,分析结果更加精确.

关键词 复杂嵌入式实时系统;体系结构分析设计语言;UPPAAL;可调度性;模型检测
中图法分类号 TP311

收稿日期:2013-05-22;**修回日期**:2014-04-08
基金项目:国家"八六三"高技术研究发展计划基金项目(2011AA010102);中国航空科学基金项目(20115553023)

复杂嵌入式实时系统广泛应用于航空电子、航天器、汽车控制等领域,这些系统具有资源受限、实时响应、容错、专用硬件等特点,对实时性、可靠性等性质有较高的要求.近年来,嵌入式实时系统体系结构分析与设计语言(architecture analysis & design language, AADL)[1-2]逐渐成为复杂嵌入式系统建模、设计与分析的标准语言,得到广泛的关注和应用.在嵌入式实时系统中,计算的正确性不仅取决于计算的逻辑结果,也取决于结果产生的时间.系统的实时可调度性是嵌入式系统能否满足应用需求的关键性质,因此 AADL 模型的可调度性分析成为 AADL 相关研究中的热点问题.

分级调度是嵌入式实时系统中一类典型的复杂调度模型.所谓分级调度是指系统处理器由多个相互协作的调度器进行共享,待调度任务被组织成不同级别的队列,采用不同的策略对各队列中的任务进行调度.本文采用模型检验的思路,以时间自动机为理论模型,将 AADL 模型中的进程、线程和分级调度器等转换为时间自动机模型,将系统的可调度性描述为一组待验证的时序逻辑公式,通过模型检验的方法对 AADL 模型的可调度性进行分析和验证.

1 问题描述

采用分级调度方法对多任务共享处理器进行调度是当前嵌入式系统典型的应用方式.例如在嵌入式实时多媒体应用中,可使用一个调度器用于处理视频和音频等关键任务,而引入另一调度器调度对响应时间不敏感的非关键任务.使用分级调度模型能够更加灵活地适用应用的需求,降低系统设计成本并提高资源利用率[3].

AADL 以构件的形式从软件组件和执行平台两个角度对嵌入式实时系统进行描述.其中软件组件主要包括共享数据、线程、进程、子程序等;执行平台组件主要包括处理器、存储器、总线、外设等.从可调度的角度我们主要关注系统的线程、进程和处理器组件.AADL 线程是嵌入式系统一个可并发调度的顺序执行的指令单元.AADL 进程则抽象描述了一段被保护的代码和数据的地址空间.处理器是线程的执行和调度平台.

在对 AADL 模型分级调度模型进行分析时存在两个主要问题:1)AADL 语言虽然提供了诸如调度周期、执行时间、截止期等标准属性来描述线程的实时特征,但缺乏对优先级、调度协议等复杂调度模型的描述能力;2)已有的分级调度器数量较多、类型各异[4-5],这些调度器采用各不相同的策略进行任务调度,缺乏一种能够对各种调度器进行描述的标准模型.为此,我们在与 AADL 核心标准保持语义一致的基础上对其进行扩展,引入自定义的扩展属性用于对分级调度进行描述.以时间自动机为标准模型,将线程、调度器等描述成相互同步的时间自动机网络,在此基础上分析 AADL 分级调度模型的可调度性.

以实例说明,图 1 是符合 ARINC653 标准的飞行控制系统 AADL 模型的部分定义.系统包括两个子系统:Nav_Autopilot_System 子系统和 HCI_System 子系统,通过总线进行交互,互相传递信息.

1) Nav_Autopilot_System 子系统主要功能是使 4 个作动器和 GPS 传感器交互,通过 GPS 获取当前地理位置信息,经过 P_NAV_Con 进程计算后,通过调整作动器参数值来控制飞行控制系统状态;

2) HCI_System 子系统主要功能是显示当前地理位置,帮助驾驶员了解当前情况,驾驶员可以此判断飞行情况是否满足要求;同时,驾驶员也可以输入自动驾驶参数和取消自动驾驶.

进程 P_NAV_Con 和 P_HCI 共享同一处理器.按照 ARINC653 标准分别称之为分区 1 和分区 2.每个进程中包含多个待调度线程,如 P_NAV_Con 中包含线程 T_AP_Computer, T_GPS_Reader 和 T_AP_Params.系统采用两级调度模型:

1) 第 1 级调度是分区调度,由调度器 processor_scheduler 实现.每个分区被轮转激活运行.分区被分配给一个固定的时间容量,调度发生时若分区的执行时间大于等于该时间容量,则分区被挂起,激活下一分区执行.时间容量的大小在系统运行前静态分配.

2) 第 2 级调度是线程调度.待调度线程在各自分区中组织成待调度队列.每个分区中采用不同的调度策略.P_NAV_Con 调度由 *Partition1_Scheduler* 实现.采用比率单调调度方法,为每一个周期任务指定一个固定的优先级,该优先级按照任务周期的长短顺序排列,任务周期越短其优先级越高.P_HCI 采用轮转调度,由 *Partition2_Scheduler* 实现.

对 AADL 语言进行扩展以支持对可调度性的分析,如图 2 所示代码.在线程描述代码中调度类型(*Dispatch_Protocol*)、调度周期(*Period*)、执行时间(*Compute_Execution_Time*)、截止期(*Deadline*)等为 AADL 标准属性,而 *RS_Properties*::*Fixed_Priority* 是为分析可调度性引入的扩展属性,为线程指定调度优先级.

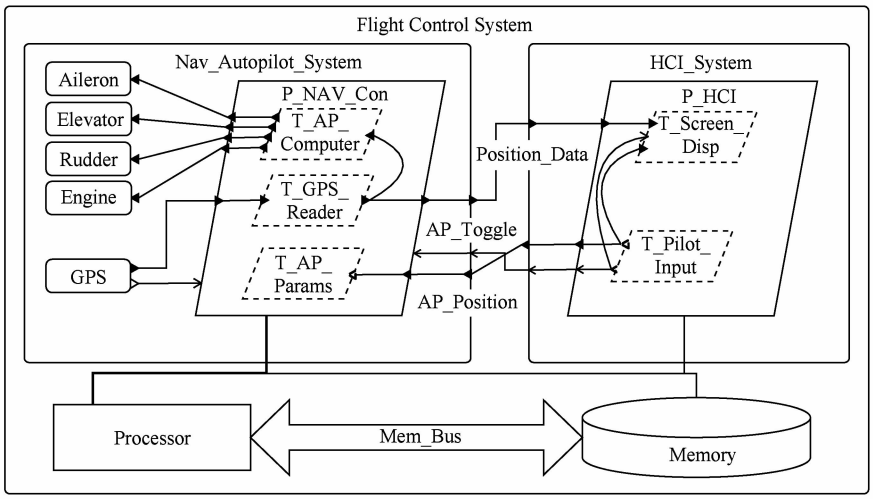


Fig. 1　AADL architecture of a part of flight control system.

图 1　部分飞行控制系统的 AADL 模型

在线程调度器代码中,对于进程 *P_HCI*,用属性 *RS_Properties*::*Scheduling_Protocol*=>*Automaton_User_Defined_Protocol* 指明,P_HCI 调度由用户自定义的调度器实现.引入时间自动机来描述 P_HCI 调度器的调度行为,将其记录在 XML 格式的文件 arinc_partition.sc 中.用属性 *RS_Properties*::*Source_Text*=>"arinc_partition.sc"指明,描述该调度的源文件名为 arinc_partition.sc.由于该源文件中可能包含多个调度器的自定义描述.进一步,用属性 *Automaton_Name*=>"*Partition*1_*Scheduler*"指明描述调度器的自动机名为 *Partition*1_*Scheduler*.分区调度的调度器描述方法与线程调度器类似.

```
thread implementation T_AP_Computer.Impl
properties
  Dispatch_Protocol=>Periodic;
  Compute_Execution_Time=>3 ms…3 ms;
  RS_Properties::Fixed_Priority=>2;
  Deadline=>10 ms;
  Period=>10 ms;
end T_AP_Computer.Impl;
⋮
process implementation P-HCI.Impl
subcomponents
  T-Screen-Disp: thread T-Screen.Impl;
  T-Pilot-Input: thread T-Pilot.Impl;
properties
  RS_Properties::Scheduling_Protocol=>Automaton_User_
  Defined_Protocol;
  RS_Properties::Source_Text=>"arinc_partition.sc";
  RS Properties::Automaton_Name=>"Partition1_Scheduler";
end P-HCI.Impl;
⋮
```

Fig. 2　A part of an AADL specification modeling an ARINC 653 avionic system.

图 2　ARINC 653 飞行控制系统的部分 AADL 代码

2　时间自动机理论

由 Alur 和 Dill[6] 提出的时间自动机是一种用于描述、分析实时系统行为的形式化模型,它在有限状态自动机的基础上扩充了时钟变量,用以刻画连续变化的时间.该理论的一个重要性质是:检测时间自动机中任意一个状态是否可达(reachable)的问题是可判定的.模型检测工具 UPPAAL[7] 和 KRONOS[8] 等为时间自动机的行为模拟、性质检测提供了强有力的支持.

本文使用的时间自动机中的符号如下:

Chan 为所有通道的集合,用小写字母 a,b 等表示其中元素;*Act* 是所有动作的集合,包含输入、输出和内部 3 类动作,$Act=\{a?\mid a\in Chan\}\cup\{a!\mid a\in Chan\}\cup\{\tau\}$,对于每一通道 a 有两个可见动作 $a?$ 和 $a!$;*Clock* 是所有时钟变量的集合.

定义 1. 时钟约束.X 是一组时钟变量的集合,用 x,y 表示其中的元素.X 上的时钟约束 Φ 的集合 $\Phi(X)$,由如下文法定义:

$$\varphi ::= x\sim c\mid x-y\sim c\mid \varphi_1\wedge\varphi_2,$$

其中,$x,y\in X$,$c\in Q_{\geqslant 0}$ 是一个有理数,$\sim\in\{<,\leqslant,>,\geqslant\}$.

定义 2. 时间自动机.一个时间自动机可表示为六元组 $A=\langle L,B,X,I,E,l_{\text{ini}}\rangle$,其中:$L$ 是一个有穷位置的集合,用 l 表示其中元素;$B\subseteq Chan$ 是有限通道的集合,用 α,β 表示其中的元素;X 是有限时钟变量的集合,用 x,y 表示其中的元素;$I:L\to\Phi(X)$ 是一个映射,它给每一位置赋予一个时钟约束;$E\subseteq$

$L\times B_{?!}\times\Phi(X)\times\rho(X)\times L$ 是有向边的集合，元素 $(l,\alpha,\varphi,Y,l')\in E$ 描述了从位置 l 到 l' 的一个变迁. φ 是 X 上的一个时钟约束，它变迁发生时被满足. Y 是该变迁发生时复位置零值的时钟集合；$l_{\text{ini}}\in L$ 是初始位置.

定义 3. 时间自动机的操作语义. 时间自动机 $A=\langle L,B,X,I,E,l_{\text{ini}}\rangle$ 的操作语义可以定义为标记转移系统：$\zeta(A)=(Conf(A),Time\cup B_{?!},\{\xrightarrow{\lambda}\mid\in Time\cup B_{?!}\},C_{\text{ini}})$，其中，$Conf(A)=\{\langle l,v\rangle\mid l\in L\wedge v:X\rightarrow Time\wedge v$ 满足 $I(l)\}$ 是 A 的格局的集合；集合 $Time\cup B_{?!}$ 是变迁发生时所有可能的变迁标记的集合；对于每一个 $\lambda\in Time\cup B_{?!}$，变迁关系 $\xrightarrow{\lambda}\subseteq Conf(A)\times Conf(A)$ 可以是如下两种形式之一：

1) 延迟变迁，即对于 $t\in Time$ 的时间推移，自动机位置不发生改变. 记为 $\langle l,v\rangle\xrightarrow{t}\langle l,v+t\rangle$，该变迁可以发生当且仅当对于所有的 $t'\in[0,t]$，$v+t'$ 满足 $I(l)$.

2) 动作变迁，即时间不发生推移，动作 $\alpha(\alpha\in Time\cup B_{?!})$ 发生且某些时钟变量被重置为 0. 记为 $\langle l,v\rangle\xrightarrow{\alpha}\langle l',v'\rangle$，该变迁可以发生当且仅当存在 $(l,\alpha,\psi,Y,l')\in E$，使得 v 满足 ψ，且 $v'=v[Y::=0]$ 满足 $I(l')$.

多个时间自动机可以并发合成的方式构成时间自动机网络. 网络中的各时间自动机既可以并发执行各自的动作，又可以在共享的信道上由一方作出输出动作，一方作输入动作，从而实现同步通信. 这种通信实际上就对应着实体之间的交互.

3 分级调度的时间自动机模型

我们以时间自动机为理论模型，根据 AADL 模型结构和组件的语义建立分级调度过程的模型.

3.1 周期性线程的模型

周期性线程的分派策略是当线程的周期到来时该线程即被分派并等待执行，线程的周期默认与其截止时间相同. 周期性线程的相邻两次分派的时间间隔不能小于周期值，即一次触发分派事件到达时，上一次触发分派事件已经执行完成，否则线程出现超时错误. 线程包含 5 个属性：线程标识（ID）、初始化时间（$initTime$）、调度周期（$Period$）、执行时间（$exeTime$）、优先级（$TPriority$）. 如图 3 所示，在时间自动机中定义 5 个位置：初始位置（$ThreadInit$）、就绪位置（$Ready$）、执行位置（$Running$）、等待分派位置（$Waitdispatch$）和错误位置（$Error$）. 为记录线程调度过程中的已执行时间和已分派的时间，定义 2 个时钟：执行时钟 e 和截止时钟 t.

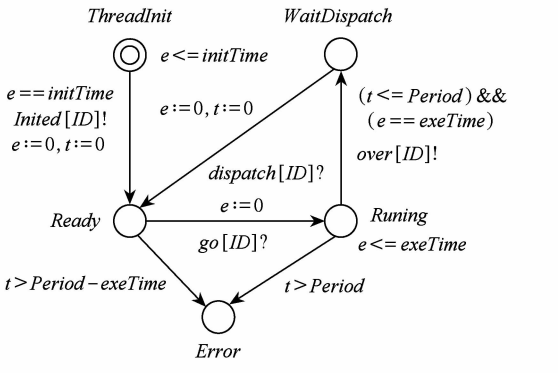


Fig. 3 Automaton modeling the periodic thread.

图 3 周期性线程模型

线程的初始状态在 $ThreadInit$ 位置. 线程初始化过程结束后，通过通道 $inited$ 发送事件给线程事件产生器，将执行时钟 e 和截止时钟 t 重置为 0 后迁移到 $Ready$ 位置，等待调度.

在 $Ready$ 位置上，如果接收到准予执行事件 go，则将执行时钟 e 重置为 0 后，迁移到 $Running$ 位置；如果截止时钟 t 大于线程的截止时间与执行时间之差（$Period$-$exeTime$），则发送超时事件 err，然后迁移到 $Error$ 位置.

在 $Running$ 位置上，如果执行时钟 e 等于当前线程的执行时间 $exeTime$，并且截止时钟 t 小于截止时间 $Period$，即线程在截止时间到达前执行完，则发送线程正常执行完毕事件 $over$，然后迁移到 $WaitDispatch$ 位置；如果执行时钟 e 小于当前线程的执行时间 $exeTime$，并且截止时钟 t 大于截止时间 $Period$，则说明线程在截止时间到达前没有执行完，发送超时事件 err，然后迁移到 $Error$ 位置.

在 $WaitDispatch$ 位置上，如果接收到线程分派事件 $dispatch[ID]$，则将执行时钟 e 和截止时钟 t 重置为 0 并迁移到 $Ready$ 位置，开始新的线程周期.

线程状态迁移到 $Error$ 位置后，将停止位置的迁移.

3.2 周期性线程的触发器模型

周期性线程的触发器用于产生周期性的线程调度请求. 如图 4 所示，在该自动机中定义初始位置（$TInit$）、周期性触发位置（$PTrigger$）和两个临时位置. 临时位置作为中间状态用于产生在同一时刻发生的不同事件，其时间延迟为 0. 定义时钟 x，初始值为 0.

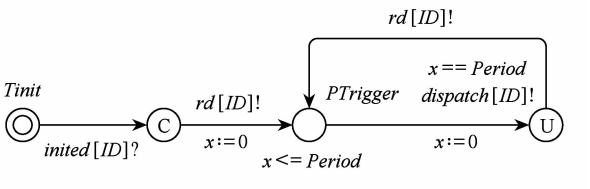


Fig. 4　Automaton modeling the periodic thread trigger.

图 4　周期性线程触发器

初始位置 *TInit* 首先等待线程初始化完毕事件 *inited*[*ID*]?,之后发送线程调度请求 *rd*[*ID*]?事件给调度器,然后迁移到 *PTrigger* 位置. 在 *PTrigger* 位置上,如果时钟 x 等于该线程的周期 *Period*,则发送线程调度请求事件 *rd*[*ID*]!给处理器调度器,并发送线程分配事件 *dispatch*[*ID*]!通知线程进入就绪状态. 之后重置时钟 x 为 0 后迁移回 *PTrigger* 位置. 这样就实现了让线程以 *Period* 为周期来循环被触发分派.

3.3　处理器调度模型

处理器组件根据属性 *Scheduling_Protocol* 的值来选择调度策略对进程进行调度,如图 5 所示,忽略处理器其他属性对调度过程的影响,在自动机中定义 4 个位置:重启动位置(*Restart*)、进程 1 调度位置(*SchedulePartiton*1)、临时位置(*Temp*)、进程 2 调度位置(*SchedulePartiton*2). *Restart* 和 *Temp* 位置的时间延迟为 0. 引入数组 *Partition*1_*Queue* 和 *Partition*2_*Queue* 分别用于存储分区 1 和分区 2 中待调度的任务队列.

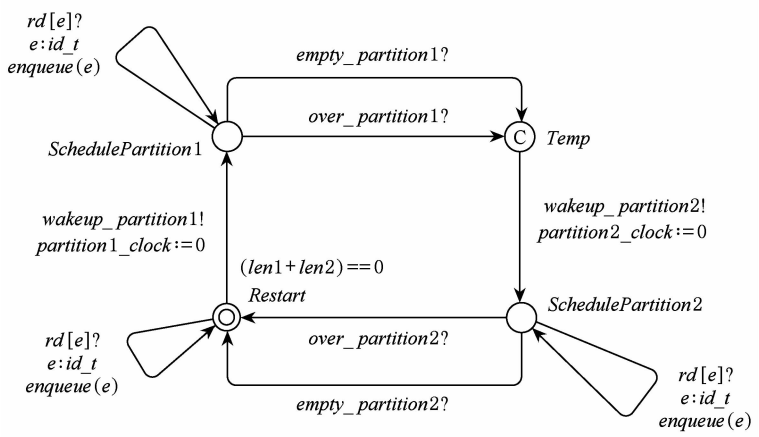


Fig. 5　Automaton modeling the processor scheduler.

图 5　处理器调度模型

在进程调度级别采用时间片轮转调度. 一次进程调度的过程是从 *Restart* 位置开始,首先进行分区 1 的调度,分区 1 处理完毕后迁移至 *Temp* 位置,然后进行分区 2 的调度. 分区 2 调度完成后迁移回 *Restart*,准备再次调度分区 1.

初始状态在 *Restart* 位置上,首先发送激活分区 1 事件 *wakeup_partition*1,将分区 1 时钟置 0,然后迁移至 *SchedulePartiton*1 位置. *wakeup_partition*1 事件激活分区 1 调度器对分区 1 中的线程进行调度.

在 *SchedulePartiton*1 位置上,若接收到线程调度请求事件 *rd*[*id*](*id* 为线程标识符),则将该事件触发分派的线程放入等待队列. 按优先级策略对等待队列中的线程按从高到低顺序进行排序,后迁移回自身. 若分区 1 任务队列为空(*empty_partition*1),或者分区执行时间已经达到调度时限的上限(*over_partition*1),则迁移至 *Temp* 位置.

Temp 位置的工作方式与 *Restart* 类同,发送激活分区 2 事件 *wakeup_partition*2,将分区 2 时钟置 0,然后迁移至 *SchedulePartiton*2 位置. *SchedulePartiton*2 位置的工作方式与 *SchedulePartiton*1 位置类同.

3.4　线程调度模型

线程调度器用于完成第 2 级调度,如图 6 所示. 即当处理器被分派给某一进程时,按照既定的调度策略完成进程内一组线程的调度. 同样,通过扩展

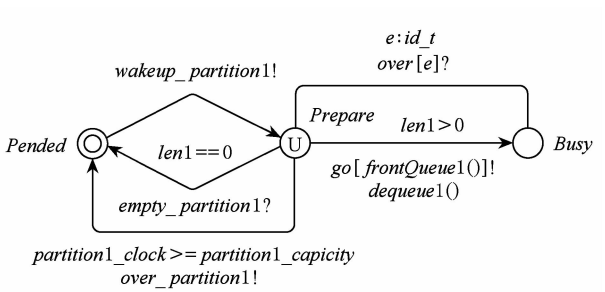


Fig. 6　Automaton modeling the thread scheduler of the partition 1.

图 6　线程调度模型

进程组件的属性 *Scheduling_Protocol*,引入自动机文件来描述调度器的调度行为.

以分区 1 的线程调度器为例. 分区 1 采用单调速率递减算法对进程内的线程进行调度. 在系统中引入数组 *Partition*1*_Queue* 作为待调度的线程的任务队列,按照优先级由大到小排列. 在自动机中定义 3 个位置:空闲位置(*Pended*)、准备位置(*Prepare*)和忙位置(*Busy*). *Prepare* 位置的时间延迟为 0.

调度器初始处于 *Pended* 位置,当接收到唤醒进程的事件 *wakeup_partiton*1 时迁移至 *Prepare* 位置.

在 *Prepare* 位置,若队列中存在等待执行的线程,则从队列头部取得优先级最高的线程 *ID*,向该线程发送执行事件 *go*[*ID*],将该线程从等待队列中删除后迁移至 *Busy* 位置. 若待执行任务队列为空,则向处理器调度器发送 *empty_partition*1 事件,并迁移回 *Pended* 位置. 即当前所有在等待的线程都执行完毕,分区 1 调度完毕. 若进程的执行时间已经大于预先分派给该进程的时间容量,则挂起该进程,并迁移至 *Pended* 位置.

在 *Busy* 位置,若接收到线程执行完毕事件 *over*[*e*],则迁移至 *Prepare* 位置. 即一个线程执行完,调度器继续选择等待队列中优先级最高的线程开始执行.

进程 2 采用轮转调度,实现机制与分区 1 调度器类似. 定义数组 *Partition*2*_Queue* 作为待调度的线程的任务队列,每次则从队列头部取得待线程 *ID*,向该线程发送执行事件 *go*[*ID*].

3.5 调度系统的时间自动机模型

在对线程、触发器、进程调度器和线程调度器形式化描述的基础上,可将分级调度模型描述为多个时间自动机的并发组合. 分级调度模型的时间自动机模型可表述为

$$T_0 \parallel T_1 \parallel \cdots \parallel T_n \parallel Trigger_0 \parallel Trigger_1 \parallel \cdots \parallel Trigger_n \parallel ProcesorSchedular \parallel ThreadSchedular1 \parallel ThreadSchedular2,$$

其中,$T_0, T_1, \cdots, T_n$ 为系统中的线程,$Trigger_0$, $Trigger_1, \cdots, Trigger_n$ 为线程触发器,*Procesor-Schedular*, *ThreadSchedular*1, *ThreadSchedular*2 分别为分区调度器和线程调度器. 这些自动机共享分区任务队列、线程优先级数组等存储空间,共享优先级排列、取队列头线程和任务删除等系统函数. 描述系统各部分的自动机以并发的方式运行,通过共享通道的动作发送/接收匹配来进行状态迁移的同步和等待.

4 待验证性质的时序逻辑描述

进一步可将待验证性质描述成时序逻辑公式[7],并基于模型检验工具对 AADL 分级调度模型进行模拟分析和验证. 对于分级调度模型,我们关心如下待验证性质:

1) 状态可达性(reachability). 表示期望调度模型能够到达的某个状态,即存在从初始状态开始到该状态的一条运行轨迹,例如:$\exists \Diamond T_i.Runing$ 表示线程 T_i 能够得到执行;$\exists \Diamond Schedular1.Busy$ 表示处理器能够被线程调度器 *Schedular*1 使用.

2) 系统安全性(safety). 期望分级调度器在运行过程中不会进入错误的状态. 例如希望整个分级调度模型运行过程中不会出现死锁,表示为 $\forall \Box$Not Deadlock.

$\forall \Box(T_1.Runing+T_2.Runing+\cdots+T_n.Runing<=1)$表示在同一时刻处理器最多被一个线程使用;$\forall \Box(ThreadSchedular1.Busy+ThreadSchedular2.Busy<=1)$表示调度器 1 和调度器 2 总是互斥使用处理器.

3) 系统活性(liveness). 表示期望的事件最终能够发生. 可用于描述当线程处于某一状态时,在调度器的控制下最终能计入一个期望的状态. 例如,$T_i.Ready \rightarrow T_i.Runing$ 表示当线程的调度请求发送并处于等待执行状态时,线程最终能够得到执行;$T_i.Runing \rightarrow T_i.WaitDispatch$ 表示如果线程被调度执行,则能够成功终止;$T_i.WaitDispatch \rightarrow T_i.Ready$ 表示周期性线程总是能够被反复激活运行.

4) 时间约束与可调度性. 这类性质描述中包含有时间变量,用于强调对实体行为的时间约束是否被满足. 例如线程 T_i 在等待 s 个时间单位后一定能够获准被运行,该性质可以表示为 $T_i.Ready \rightarrow (T_i.Runing \wedge t<=s)$,其中 t 为时钟变量,它在进程处于 *ready* 状态时被置 0;$\forall \Box(\neg T_i.Error)$表示线程不会进入 *Error* 状态,即线程在其时间约束内总是能够被调度执行;$\forall \Box(T_1.Error+T_2.Error+\cdots+T_n.Error=0)$表示所有的线程都不会进入 *Error* 状态. 即所有的线程在其时间约束范围内都能够被调度执行,即该组线程在其时间约束范围内是可调度的.

5　系统实现与可调度性验证

根据本文所提出的模型结构及方法,我们基于 Eclipse 平台开发了一个模型转换器来实现从 AADL 模型到 UPPAAL 可识别的时间自动机模型的转换,并把它以插件形式集成在由美国卡内基梅隆大学软件工程研究所开发的 AADL 的开源开发工具环境(open source AADL tool environment, OSATE)[9]中.在 OSATE 环境下可以进行 AADL 模型代码的编辑、语法检查及模型系统实例化等操作.

模型转换器利用 OSATE 开发环境中提供的 API 接口来对 AADL 模型进行遍历,从 AADL 模型中提取必要的线程和处理器属性信息,按照第 3 节所设计的模板结构,生成时间自动机模板,输出包含所有模型信息的 XML 文件.

实现对等待队列中的线程按不同优先级策略进行排序,可以在调度器模板的局部声明中定义不同的函数,通过调用函数来实现等待队列中的优先级排序、取最高优先级线程和删除一个线程等操作.

以 UPPAAL 为验证工具,以 3.5 节所示的分级调度模型为验证对象,随机取一组待调度线程及其时间约束数据如表 1 所示.表 1 中"—"所示为轮转调度部分,无优先级之分.

将待验证的调度模型输入验证工具 UPPAAL,可生成如图 7 所示的时间自动机网络.这些自动机以并发的方式执行,通过通过共享通道的动作发送/接收匹配来进行状态迁移的同步和等待.模型演化过程中的时间推移可通过观察时钟变量的值,或者模拟器中执行 trace 的事件发生时间获得.

Table 1　Data of AADL Threads to be Scheduled

表 1　待分级调度的 AADL 线程数据表

Thread ID	Initialization Time	Scheduling Period	Execution Time	Priority	Partition
0	91	95	9	452	Partition1
1	36	106	11	815	Partition 1
2	223	197	20	729	Partition 1
3	87	112	7	335	Partition 1
4	132	318	31	113	Partition 1
5	153	112	12	—	Partition 2
6	149	302	29	—	Partition 2
7	47	50	4	—	Partition 2
8	51	164	15	—	Partition 2
9	78	203	18	—	Partition 2

进一步,可通过模拟器对该模型的演化过程进行模拟.模拟过程描述了网络中每个自动机的自身位置、交互情况、网络演化过程和时钟变量值.通过步进的方式对自动的演化进行模拟能够使用户跟踪

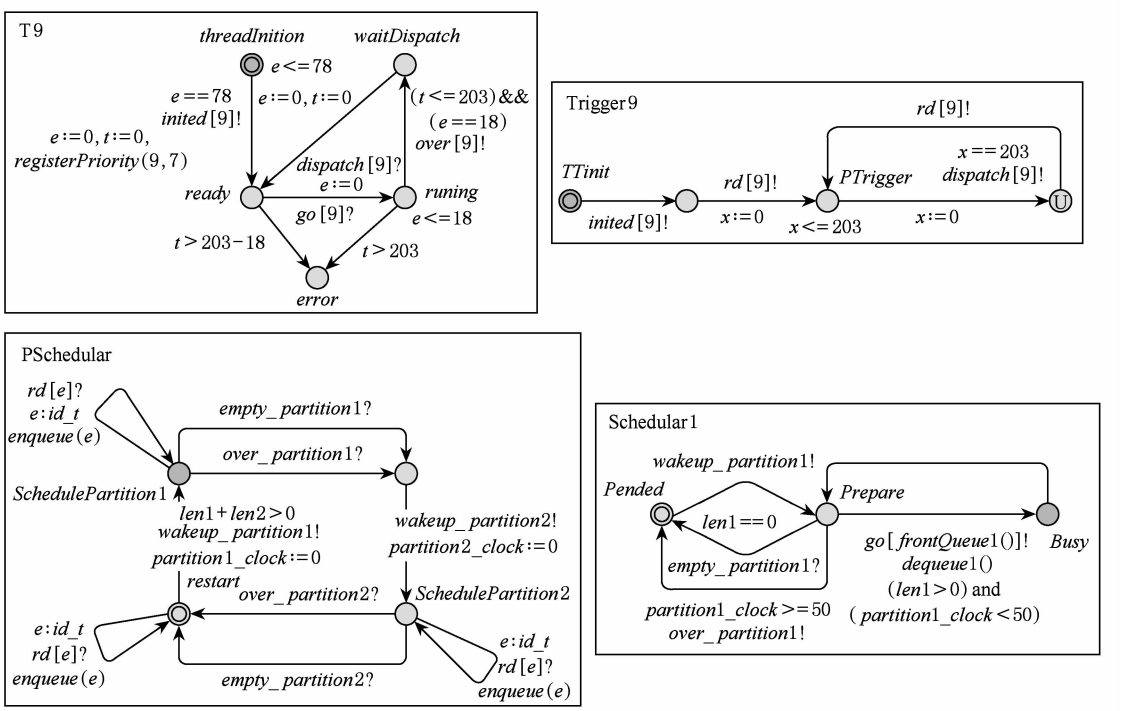


Fig. 7　A part of automaton network modeling the system scheduling.

图 7　调度过程中的部分自动机状态图

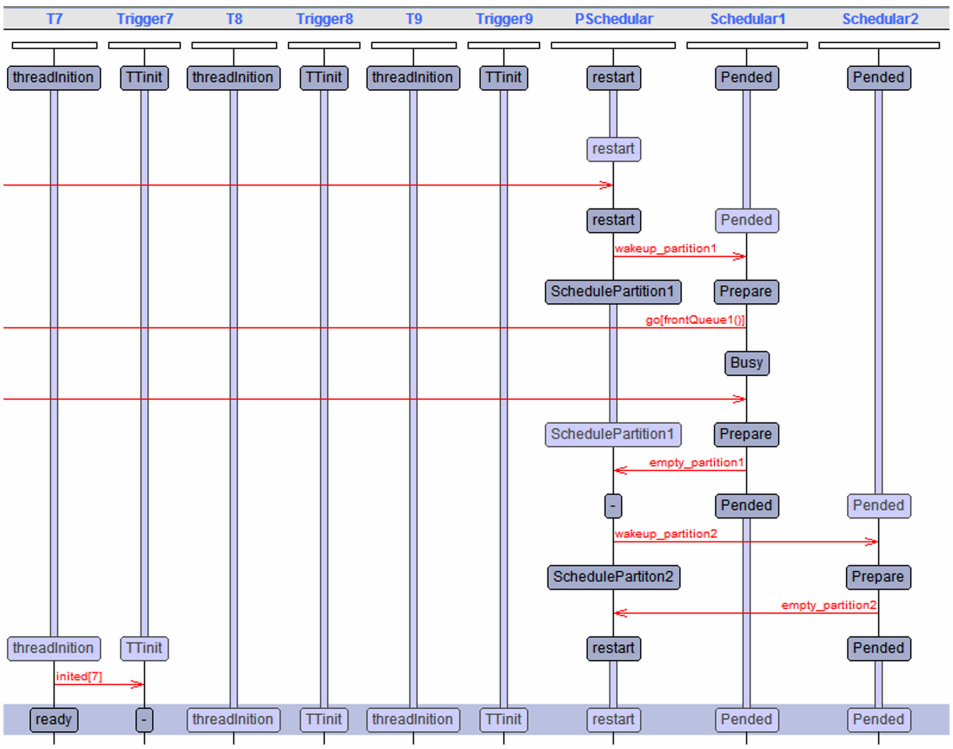


Fig. 8 The simulation of AADL hierarchical scheduling model.

图 8 AADL 分级调度模型的模拟过程

模型的演化过程,观察时钟变量的推进,分析自动机设计中的错误,查看其工作方式是否满足用户的预期. 模拟过程如图 8 所示.

进一步的可验证模型满足如下相关性质:

$E<>Ti.Runing, i\in\{0,1,\cdots,n\}$;

$E<>ThreadSchedular_i.Busy, i\in\{1,2\}$;

$Ti.Ready\rightarrow Ti.Runing, i\in\{0,1,\cdots,n\}$;

$Ti.Runing\rightarrow Ti.WaitDispatch, i\in\{0,1,\cdots,n\}$;

$Ti.WaitDispatch\rightarrow Ti.Ready, i\in\{0,1,\cdots,n\}$;

$A[](ThreadSchedular1.Busy+ThreadSchedular2.Busy)<=1$;

$A[](T_0.Runing+T_1.Runing+,\cdots,+T_n.Runing)<=1$;

$A[]Not\ Deadlock$;

$A[](T_0.Error+T_1.Error+\cdots+T_n.Error)==0$.

以上性质在 Uppaal 4.0.13 中验证通过,该验证结果说明,表 1 给出的输入线程在所示的分级调度模型中可调度性能够得到满足.

模型检验的方法能够精确地穷尽系统演化中的每个可能状态,因此给我们精确分析 AADL 分级调度模型中各实体的运行状态和时间特征提供了可能. 如我们还可以通过输入以下逻辑表达式验证 $E<>T_0.Runing$ 和 $T_0.t<=0$. 该性质表明,存在一种可能的执行路径使得 T_0 不需要等待就得到立即执行.

模型检验的方法以穷举的方式分析系统中所有可能存在的状态. 对于状态空间的增长问题,可引入符号模型检测方法、On the Fly 算法、抽象归纳、偏序方法和对称归约、局部状态空间等[10]. 这些方法在验证工具 Uppaal 中得到了充分实现,为分级调度模型的分析提供了很好的支持.

6 相关工作

文献[11-13]提出一种可用于描述嵌入式系统实时特征的领域描述语言 Cheddar,用时间自动机对 AADL 调度模型进行建模,并实现了模型可调度性仿真与分析工具. 与我们采用模型检测的思路不同,其原理是基于处理器利用率,依据 EDF 和 EMS 等经典公式进行计算,根据计算结果对可调度性进行判定.

文献[14]提出了利用模型检测工具 UPPAAL 对 AADL 线程组件的可调度性进行分析和验证的方法. 与本文的工作比较,该文仅研究了线程级的调度问题,而我们则重点讨论包含多个调度级别的层

级调度模型的建模与分析.

文献[15]对时间进程代数 ACSR 进行扩展,提出 ACSR-VP. ACSR-VP 继承了 ACSR 对时间、通信和资源需求等方面的描述能力,并支持传值通信和动态优先级. 进一步基于 ACSR-VP 对实时系统可调度性进行建模和分析,并基于模型检查工具 VERSA 对可调度性进行验证. 文献[16]提出工具 Furness 基于 ACSR 对 AADL 可调度性进行分析,支持对 AADL 模型调度过程的仿真,但目前只支持简单的调度模型.

文献[17]采用时间自动机理论,对符合 OSEK/VDX 标准的实时多任务系统进行建模,用模型检测工具对系统时序逻辑进行分析. 文献[18]使用 Uppaal 对嵌入式系统的可调度性进行分析和验证,重点研究多处理器任务调度场景的分析问题. 这些研究的并未结合 AADL 语言的特性,因此不能直接用于 AADL 调度模型的分析.

文献[19-20]采用动作时序逻辑描述语言 TLA+(temporal logic of actions plus)对 AADL 执行模型的部分语义作了初步研究,包括端口通信、共享变量的构件间通信、抢占式调度策略以及模式的语义,但 TLA+的模型检测工具分析能力不足.

法国 Verimag 实验室提出 AADL 到 BIP(behavior interaction priority)的语义转换[21],如 AADL 线程转换到 BIP 原子构件、数据端口连接转换到 BIP 的连接器,仅描述了 AADL 线程、进程、处理器构件以及 Behavior Annex 的部分语义,采用 Aldebaran 工具进行死锁、安全性的验证.

英国 Leicester 大学研究 AADL 到重写逻辑 Maude 语言的模型转换[22],支持行为验证,并实现了 MOMENT2-AADL 工具. 文献[23]研究了 AADL 到实时演算(real time calculus, RTC)[24]的语义转换,以支持端到端的时间延迟分析. TIMES[25]是另一种基于任务自动机的调度分析工具,能够精确分析系统的调度性. 但是其对非周期线程的定义和 AADL 中非周期线程的语义不一致,因此不能直接用在 AADL 的调度分析过程中.

与相关研究比较,本文的工作关注于 AADL 分级调度模型,我们用时间自动机对分级调度过程进行建模,通过模型检验的方法对可调度性进行分析. 本文方法利用形式化方法的穷举性来分析系统的可调度性,因而分析结果更加精确. 并且能够通过模拟器给出调度过程的调度轨迹,对于不能调度的情况则能够给出调度不成功的反例.

7　总　　结

本文探讨了使用形式化方法和验证工具对 AADL 分级调度模型的分析和验证方法. 以时间自动机为理论,对 AADL 模型中的进程、线程和分级调度器等进行建模,将系统的可调度性描述为一组待验证的时序逻辑公式,通过模型检验的方法对可调度性进行分析. 讨论了一个符合 ARINC653 标准的 AADL 分级调度模型的分析实例. 本文的工作说明使用 UPPAAL 等验证工具来分析 AADL 分级调度模型中的任务可调度性是可行的. 相对其他方法而言,本文方法利用形式化方法的穷举性来分析系统的可调度性,精确性较高.

下一步的工作包括针对 AADL2.0 标准研究分级调度模型的表述和分析方法,完善从 AADL 到时间自动机自动模型转换方法,提高分析验证工具的易用性等.

参　考　文　献

[1] Feiler P H, Lewis B A, Vestal S. The SAE architecture analysis & design language (AADL) a standard for engineering performance critical systems [C] //Proc of the 1st IEEE Int Conf on Control Applications. Piscataway, NJ: IEEE, 2006: 1206-1211

[2] Yang Zhibin, Pi Lei, Hu Kai, et al. AADL: An architecture design and analysis language for complex embedded real-time systems [J]. Journal of Software, 2010, 21(5): 899-915 (in Chinese)
(杨志斌, 皮磊, 胡凯, 等. 复杂嵌入式实时系统体系结构设计与分析语言: AADL [J]. 软件学报, 2010, 21(5): 899-915)

[3] Kay J, Lauder P. A fair share scheduler [J]. Communications of the ACM, 1988, 31(1): 44-45

[4] Lawall J L, Muller G, Duchesne H. Language design for implementing process scheduling hierarchies [C] //Proc of the 14th ACM SIGPLAN Symp on Partial Evaluation and Semantics-based Program Manipulation. New York: ACM, 2004: 80-90

[5] Regehr J, Stankovic J A. HLS: A framework for composing soft real-time schedulers [C] //Proc of the 22nd IEEE Int Real-Time Systems Symp. Piscataway, NJ: IEEE, 2001: 3-14

[6] Alur R, Dill D L. A theory of timed automata [J]. Theoretical Computer Science, 1994, 126(2): 183-235

[7] Behrmann G, David A, Larsen K G. A tutorial on uppaal [C] //Proc of the 17th Formal Methods for the Design of Real-time Systems. Berlin: Springer, 2004: 200-236

[8] Yovine S. KRONOS: A verification tool for real-time systems [J]. Int Journal on Software Tools for Technology Transfer, 1997, 1(1): 123-133

[9] Open source AADL tool environment (OSATE). [2013-03-20]. http://www.aadl.info/addl/currentsite/tool/osate.html

[10] Baier C, Katoen J P. Principles of Model Checking [M]. Cambridge: MIT Press, 2008

[11] Singhoff F, Plantec A. AADL modeling and analysis of hierarchical schedulers [C] //Proc of the 13th ACM Int Conf on SIGADA. New York: ACM, 2007: 41-50

[12] Singhoff F, Legrand J, Nana L, et al. Cheddar: A Flexible Real Time Scheduling Framework [C] //Proc of the 10th Int ACM SIGADA Conf. New York: ACM, 2004: 1-8

[13] Singhoff F, Legrand J, Nana L, et al. Scheduling and memory requirements analysis with AADL [C] //Proc of the 11th Int ACM SIGADA New York: ACM, 2005: 1-10

[14] Liu Qian, Gui Shenglin, Li Yun, et al. Schedulability verification of AADL model based on UPPAAL [J]. Journal of Computer Applications, 2009, 29(7): 1820-1824 (in Chinese)
(刘倩, 桂盛霖, 李允, 等. 基于 UPPAAL 的 AADL 模型可调度性验证[J]. 计算机应用, 2009, 29(7): 1820-1824)

[15] Sokolsky O, Lee I, Clarke D. Schedulability analysis of AADL models [C] //Proc of the 16th Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2006

[16] Gui S, Luo L, Li Y, et al. Formal schedulability analysis and simulation for AADL [C] //Proc of the 6th Int Conf on Embedded Software and Systems. Piscataway, NJ: IEEE, 2008: 429-435

[17] Waszniowski L, Hanzálek Z. Formal verification of multitasking applications based on timed automata model [J]. Real-Time Systems, 2008, 38(1): 39-65

[18] David A, Illum J, Larsen K G, et al. Model-based framework for schedulability analysis using uppaal 4.1 [J]. Model-Based Design for Embedded Systems, 2009, 1(1): 93-119

[19] Rolland J F, Bodeveix J P, Chemouil D, et al. Towards a formal semantics for AADL execution model [C] //Proc of the 4th European Congress on Embedded Real-Time Software. Piscataway, NJ: IEEE, 2008: 1-8

[20] Rolland J F, Bodeveix J P, Filali M, et al. Modes in asynchronous systems [C] //Proc of the IEEE Int Conf on Engineering Complex Computer Systems. Piscataway, NJ: IEEE, 2008: 282-287

[21] Chkouri M Y, Robert A, Bozga M, et al. Translating AADL into BIP - Application to the verification of real-time systems [C] //Proc of the 5th ACESMB Workshop Conjunction with MODELS. Berlin: Springer, 2008: 5-19

[22] Benammar M, Belala F, Latreche F. AADL behavioral annex based on generalized rewriting logic [C] //Proc of the 7th Research Challenges in Information Science. Piscataway, NJ: IEEE, 2008: 1-8

[23] Sokolsky O, Chernoguzov A. Analysis of AADL models using real-time calculus with applications to wireless architectures, MS-CIS-08-25 [R]. Philly: University of Pennsylvania, 2008

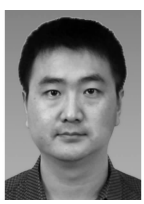
[24] Thiele L, Chakraborty S, Naedele M. Real-Time calculus for scheduling hard real-time systems [C] //Proc of the 41st Int Symp on Circuit s and Systems ISCAS. Piscataway, NJ: IEEE, 2000: 101-104

[25] Amnell T, Fersman E, Mokrushin L, et al. TIMES—A tool for modelling and implementation of embedded systems [C] //Proc of the 8th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer, 2002: 460-464

Fu Ning, born in 1976. PhD, assistant professor in the School of Computer, Northwestern Polytechnical University. His current research interests include modeling and verification embedded system by formal methods.

Du Chenglie, born in 1970. PhD. Professor and PhD supervisor in the School of Computer, Northwestern Polytechnical University. His main research interests include cyber-physical system modeling & analysis and trustworthy software architecture.

Li Jianliang, born in 1971. PhD candidate, associate professor in the College of Information Engineering, Northwest A&F University. His current research interests include modeling and verification embedded system by formal methods.



Liu Zhiqiang, born in 1975. PhD, associate professor. His current research interests include modeling and verification embedded system by formal methods.

Peng Han, born in 1976. PhD candidate, lecturer in the School of Computer Engineering, Xi'an Aeronautical University. His current research interests include modeling and verification Cyber-Physical system by formal methods.