基于Uppaal的实时系统AADL数据流模型的转换与验证

沈宁敏 李 静 白海洋 庄 毅
(南京航空航天大学计算机科学与技术学院 南京 210016)

摘 要 体系结构分析设计语言AADL是一种可支持软硬件一体化建模及同一模型多元分析的形式化与图形化建模语言。采用时间自动机形式化模型检验方法对AADL模型中的数据流进行转换和验证。考虑到单一数据流与混合数据流的差异性,分别设计了数据流到时间自动机模型的转换规则,并通过时间自动机网络实现数据流的综合分析。设计开发了自动化模型转换的插件AADLToUppaal Plug-in,将其嵌入到OSTATE工具中,使用时间自动机建模与验证工具Uppaal对转换得到的时间自动机进行模拟和验证,等价地验证所设计的AADL模型数据流时延是否满足系统实时性要求。仿真实验结果表明,所设计的数据流模型转换方法能有效地将AADL模型转换到时间自动机模型,并能在Uppaal中正确地分析原模型的数据流时延特性。

关键词 AADL,时间自动机模型,数据流时延,Uppaal,软件验证
中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.1.047

Transformation and Verification Method of AADL Data Flows for Real-time System Using Uppaal

SHEN Ning-min LI Jing BAI Hai-yang ZHUANG Yi
(College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract Architecture analysis and design language (AADL) is a modeling language for formalization and graphics, which can support integrated modeling and multi-analysis. We used timed automata formal checking method to transform and verify data flow in AADL models. Considering the difference between single data flow and mixed data flows, we designed corresponding transformation rules from AADL models to timed automata, and analyzed and verified the data flow comprehensively by establishing timed automata network. An automated model conversion Eclipse plug-in was designed and integrated into the AADL modeling tool OSATE. Finally, timed automata modeling and Uppaal were used to simulate and verify the transformed timed automata while verifying whether the AADL model equivalently satisfies the real-time requirement of system. Experimental results show that the model transformation method is valid and flow latency qualities can be verified effectively with Uppaal.

Keywords AADL, Timed automata, Data flow latency, Uppaal, Software verification

1 引言

伴随微电子、通信等技术的飞速发展,嵌入式实时系统越来越广泛地应用于航空航天、工业控制等关键任务领域,而系统规模与复杂程度的不断提升,使系统的可靠性面临着严峻的挑战。实时系统[1]是指对任务响应和执行时间具有严格要求的系统,其可靠性不仅取决于正确的功能逻辑,同时与系统实时性要求紧密关联,当任务提前或超过规定的时间节点时,系统都将被认为是不可靠的。为了提升嵌入式实时系统的开发效率与系统可靠性,基于模型驱动架构(Model Driven Architecture, MDA)[2]的体系结构分析与设计语言AADL被应用于模型构建与系统开发。AADL(Architecture Analysis and Design Language)[3,4]可以支持系统软硬件一体化建模,并对系统的可靠性、实时性等非功能属性进行了详细描述,进而使模型驱动方法中的建模、分析、验证与代码生成等关键技术可融合在统一框架之下。在AADL模型中,数据流[5]是数据和事件信息的通道,一条完整的端到端数据流通常由一个采集器设备发出,传递给一个中间组件(如线程)处理后将处理结果传递给控制器做出响应。信号采集到控制响应的过程具有一定的时效性,若数据流时延过大,会导致系统中关键数据不能及时获取或关键任务不能及时触发,从而影响整个系统的实时性能[6]。

在AADL数据流时延特性的研究中,Peter Feiler[5]给出了AADL模型中影响数据流时延的因素及其分析方法,包括线程或设备的计算、不同组件之间的传输时延、数据采样速率和设备端口上数据队列的处理方式等,通过选取AADL属性集合中的Latency属性作为分析的依据,在其开发的OSATE插件中做简单模拟的实现数据流时延的计算。Su-Young Lee[6]等结合周期与非周期线程对数据流的影响,分析给出流经线程计算的数据流时延与线程计算时间、截止时间等关系,

到稿日期:2014-11-20 返修日期:2015-04-27 本文受中央高校基本科研业务费专项资金(NS2015092)资助。
沈宁敏(1991-),男,硕士生,主要研究方向为数据挖掘、并行计算,E-mail:ningminshen@163.com;李 静(1976-),女,博士,副教授,主要研究方向为数据挖掘、可信软件;白海洋(1988-),男,硕士生,主要研究方向为软件工程、软件可靠性;庄 毅(1956-),女,教授,博士生导师,主要研究方向为分布式计算。

得出了最优时延与最差时延的计算公式，但该方法没有考虑到实际的系统调度对数据流向的影响，其结果是很不精确的。谯婷婷[7]等人给出了基于飞行控制系统实例的数据流分析结果，但并没有给出通用的解决方案。综上所述，AADL 本身作为一种模型描述语言，其形式化程度并不高，现有方法难以直接对数据流时延进行验证[8]。而借助模型转换的思想将 AADL 模型转换为一种等价的形式化模型并利用已有的形式化方法和工具进行验证已成为目前的一个研究热点[9-12]。时间自动机[13]是实时系统验证领域比较成熟的一种形式化方法，自 1992 年被提出以来已经有了比较完善的理论和高效的建模与验证工具的支持，它借助时间自动机对 AADL 数据流进行等价验证，对系统实时性的约束具有很强的表达能力[14]。因此，本文使用时间自动机形式化方法对 AADL 中的单一数据流和混合数据流时延进行等价验证，建立了数据流到时间自动机模型的转换，通过设计自动模型转换插件和模型验证工具 Uppaal，结合调度模型对流经计算线程的数据流进行综合分析。

本文第 2 节介绍 AADL 数据流与时间自动机的基本原理；第 3 节分别设计了单一数据流与混合数据流的模型转换设计方法，并设计了自动模型转换插件；第 4 节通过建立的数据流模型转换实例对设计方法进行转换验证；最后对工作进行总结并指出进一步研究的方向。

2 AADL 与时间自动机模型

2.1 AADL 数据流

AADL 模型的组件是通过流(Flow)连接起来的，端到端的流(End to End Flow)描述了系统内部数据和事件的抽象信息路径。完整的 Flow 定义包含流声明(Flow Specification)与流实现(Flow Implementation)[14]两个部分。

(1)流声明。流声明在组件的声明中完成，它定义了组件的输入到输出的逻辑路径，包含 3 种标记类型：流的起源(Flow Source)、流的终点(Flow Sink)和流的路径(Flow Path)。

(2)流实现。流实现在组件的实现中完成，通过串联组件与子组件、子组件与子组件之间的连接，形成从输入端口到输出端口的串行序列，描述了组件中确切的数据流向。与流声明对应，流实现也有 3 种类型，每个实现都代表了一条路径。Flow Source 和 Flow Sink 代表了一个流中起点或终点的一个片段(而非一个端口)，Flow Path 的起点和终点为组件中声明的端口，中间路径可以包含子组件之间或组件内部的连接。

在 AADL 的相关属性集合[4]中，定义了描述端口连接和数据传输性质的属性元素，包含事件端口队列大小与处理策略、端口即时或延迟通讯方式、端口输入输出速率、端口输入输出时间、总线传输延迟等。这些属性描述了端口间连接或者总线传输的时延性质，数据流串联了这些元素，其总的传输时延也将受到这些属性定义的影响。其中与数据流延迟直接相关的是属性 Latency，它是一个时间属性值，代表数据或事件在数据流的两端口之间的最长运行时间，当数据流组件不是一个线程或没有它自己的派发协议、周期及截止时间时，其数值被用来分析数据流组件中端到端的时延分析。Latency 的标准解释如表 1 所列。

表 1 AADL 中 Latency 属性的说明

属性名	描述	声明及实例
Latency	用于规定在流或连接上所允许的时间延迟，描述的对象包括数据流(Flows)、连接(Connections)等；属性赋值为时间类型(整型加上时间单位)，支持的标准时间单位包括纳秒(ns)、微妙(μs)、毫秒(ms)、秒(s)、分(m)和小时(h)	声明：Latency：Time applies to(Flow, connection) 实例：Latency=>13μs

端到端的数据流作为流实现的特例，实现跨组件之间的数据流连接，起点为 Flow Source，终点为 Flow Sink，中间串联多个组件之间的流实现和连接。一个完整的端到端数据流的实例如图 1 所示。

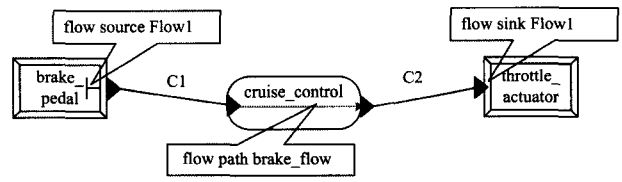


图 1 端到端的数据流示意图

一个端到端的数据流 Fp 可以描述为一个六元组 $Fp=\{F,f_s,f_d,Po,Co,La\}$，其中，

F：流声明中组件内部的数据流表示，用端口到端口以及在转移过程中的时延来定义，即 $F=\{Po\times Po,La\}$；

f_s：$f_s\in F$，即 flow source，是端到端数据流的起点；

f_d：$f_d\in F$，即 flow sink，是端到端数据流的终点；

Po：端口组件的集合，其中 $Po=\{$data port，event port，event data port$\}$，即数据端口、事件端口和数据事件端口的集合；

Co：$Co=\{Po\times Po,\phi'\}$，组件之间端口连接的集合，ϕ'为连接约束的条件，如禁止 event port 指向 data port 或者 event port，而其它指向的连接都是合法的；

La：Latency 时延属性的集合，它作用于 F 和 Co。

2.2 时间自动机

时间自动机[12,15,16]是在传统的有限状态自动机的基础上扩充了时钟、时钟约束和不变条件[17-19]。系统中定义的时钟在一定的取值范围内按照相同的速率同步连续增长，也可以在任意时刻复位为 0；系统中的时钟和整型变量形成的不等式成为状态转移的约束条件，在时钟满足一定的条件下，状态转移才会发生；不变条件是指对状态停留在一个位置的约束，当不变条件满足时，时钟才会在一个位置停留并持续增长。

时间自动机作为一种形式化的方法有其严格的定义[20]。

定义 1(时钟约束) 对于一个时钟变量集 C，时钟约束 φ 的集合 $\Phi(C)$为如下形式：

$\varphi::=\{x\ \triangleright\!\triangleleft\ n\mid\varphi_1\wedge\varphi_2,x\in C,n\in N,\triangleright\!\triangleleft\in\{<,\leqslant,\geqslant,\neq\}\}$，$x$ 是一个时钟变量，n 是一个自然数集 N 中的常量，φ_1 和 φ_2 是两个时钟约束，时钟约束的结果为一个布尔值。

定义 2(时钟解释) 一个时钟解释是时钟集合 C 到自然数集的映射，$\nu:C\to N$。对于一个 $\delta\in R$，$\nu+R$ 表示的时钟解释是对时钟变量集合 C 中的任意时钟变量 x 的赋值为 $\nu(x)+\delta$；而 $\delta\cdot\nu$ 表示的时钟解释是对时钟变量 x 赋值为 $\delta\cdot\nu(x)$；对于 $X\subseteq C$，$\nu[X:=0]$表示对满足 $x\in X$ 的时钟 x 复位为 0，其余时钟保持增长。

定义 3(时间自动机) 一个时间自动机可以表示为一个

六元组，$TA=\langle L,l_0,C,Var,E,I\rangle$，其中

L 为有穷的位置(location)集合；

$l_0(l_0\in L)$为自动机初始状态；

C 为时钟集合，时钟默认从 0 开始，不断加 1，并可以在任意时刻被复位为 0；

Var 为一系列变量的集合；

E 为边(Edge)的集合，$E\subseteq L\times G\times Act\times U\times L$，其中 G(有可能为空)代表约束条件(guards)的集合 $\Phi(x)$，$Act=In\cup Out$，分别表示输入(记号?)和输出(记号!)的同步信号，构成触发转移的使能条件，U 是对时钟变量或者整型变量的更新操作，即在触发转移的同时完成变量的赋值操作，其形式为 $x:=expr,x\in C|Var\cup Var_G$，$Var_G$ 代表的是系统的全局变量，$expr$ 表示一个任意的实数；

I 为不变条件(invariant)，是状态转移的约束函数的集合。

一个时间自动机 TA 的语义可以用与时间系统相关的转移系统 S_{TA} 来定义。S_{TA} 的状态可用一个二元组$\langle l,\nu\rangle$来表示，$l\in L$ 是 TA 的一个状态，ν 是满足不变式 $I(l)$的一个时钟解释。若 $l=l_0$，即表示初始状态，在该位置上所有时钟变量的初始值为 0，即 $\nu(x)=0$。S_{TA} 有两种类型的迁移：

(1)延迟迁移。因时间流逝而发生的改变，$\langle l,\nu\rangle\xrightarrow{\delta}\langle l',\nu'\rangle$，其中 $\delta\in N,\nu\in I(l)$，且 $\nu+\delta\in I(l)$。在这种迁移中，时间自动机的位置并没有发生改变，只是在满足不变条件下的时钟值增长。

(2)位置迁移。因为满足约束条件而发生的改变，即满足 $l\xrightarrow{g,a,u}l'$的条件，发生由状态 l 到 l'的迁移，且 $\nu\in I(l)$，$\nu'\in I(l')$。

如图 2 所示，位置 l_1 经过边到达位置 l_2，guard 为守卫条件，action 为同步信号，update 为转移更新条件。图中还展示了一个实例，在时钟 cl_1 大于 5 的时候，当接收到灯光信号 light，变量 x 置 0，并把时钟 cl_1 复位。

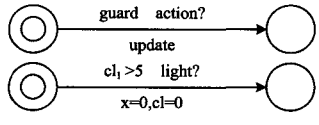


图 2 时间自动机状态转移实例

3 模型转换方法设计

本节设计了 AADL 数据流模型到时间自动机模型的转换方法。首先通过对 AADL 数据流的分析形成了数据流的形式化描述；继而建立这种形式化描述到时间自动机语义的映射关系以作为映射法则的定义，时间自动机的转换按单一和混合数据流两种类型分别给出转换法则和转换实例的说明；最后根据数据流性质验证语句和时间自动机模型检验方法对转换模型进行等价验证。本文设计的方法如图 3 所示。

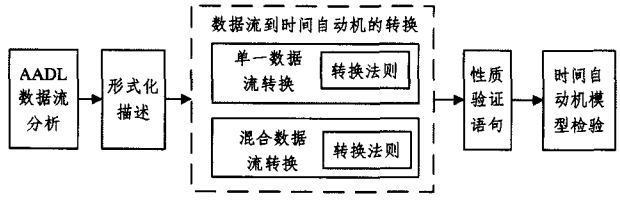


图 3 AADL 数据流的转换与验证方法流程

3.1 单一数据流

在单一数据流中，所有数据端口在同一层级的组件之间，数据不流向其子组件且不受线程计算和系统任务调度的影响，设备对数据以及事件信号的发送和接收都是即时的，数据流时延的产生仅考虑 Latency 属性。单一数据流到时间自动机的转换规则和转换实例如下。

(1)转换规则

定义转换规则 $\Gamma_{Fp},\Gamma_{Fp}(Fp)=TA$，即$\{F,f_s,f_d,Po,Co,La\}\rightarrow\langle L,l_0,C,Var,E,I\rangle$，即系统单条端到端数据流集合映射到单一时间自动机集合，具体的映射关系如下：

$Po\rightarrow L$，端口映射为基本状态，按照数据的流向通过自动机中的边串联起来，加入一个 Start 表示流的开始，并在最后加入终止状态 End 作为统一可达性验证的参考状态。

$La\rightarrow\langle C,Var\rangle$，时延对应于时钟变量和整型变量，一个端到端的流中设置了几个分段的时延(或整个流的总时延)，通过时钟变量和整型变量记录约定的约束值。

$\langle Co,La\rangle\rightarrow\langle E,I\rangle$，每个 Connection 的时延转换有时钟赋值、转移条件和不变条件 3 个部分。时钟赋值在端口对应状态的进入条件中设置，其时钟设置为 0；转移条件在连接起点端口状态转移到终点端口状态的边上设置，判断条件为 clock==latency，表示数据的传递在 Latency 设置的时间点上完成；不变条件为 clock<=Latency，是对转移前状态的约束。

流声明中 F 代表组件内部的数据流，这部分的转换方法与 Connection 的转换方法$\langle Co,La\rangle\rightarrow\langle E,I\rangle$相同。在流实现跨组件的 Flow Path 转换中，Latency 起到验证的作用，检查 Flow Path 内部流的累加是否超过设置的 Latency 值。为了实现这种转换，设置了相应的时钟和整型变量，时钟从 Flow Path 的开始状态之前开始计时，同时在 Flow Path 终点端口状态之后加入 Committed 状态，转移条件为所设置时钟值小于或等于 Latency 值，Committed 的后继状态为下一个端口状态或 End 状态。

(2)转换实例

以自动化管焊设备的焊枪转速控制部分 AADL 模型为例，其主要功能是将一个传感器采集到的焊枪头电机的转速信息传递给计算线程，将计算线程处理后产生的控制信号传递给控制设备，从而形成一个速度调节反馈系统。该模型的 AADL 文本格式代码如图 4 所示。

```
system WeldingSystem
end WeldingSystem;
system implementation WeldingSystem. SystemImpl1
  subcomponents
    Sensor1:device EncoderSensor. impl;
    Processor1:processor Processor1;
    SpeedActuator:device SpeedActuator. impl;
    Process0:process Process0. ProcessImpl1;
  connections
    C1:data port Sensor1. Encode_Out -> Processor1. Encode_In;
    C2:data port Processor1. SpeedControl_Out -> SpeedActua-
        tor. SpeedControl_In;
  flows
    SpeedControl_Flow:end to end flow Sensor1. SensorOut_Flow ->
    C1->Processor1. InProcessor_Flow-> C2 -> SpeedActua-
```

```
    tor. ActuatorIn_Flow
      {
        Latency => 8ms;
      };
  properties
    Allowed_Processor_Binding => reference processor1 applies to
    Process0;
    Actual_Processor_Binding => reference processor1 applies to
    Process0;
end WeldingSystem. SystemImpl1;
device EncoderSensor
  features
    Encode_Out:out data port;
  flows
    SensorOut_Flow:flow source Encode_Out {
      Latency => 2ms;
    };
end EncoderSensor;
device implementation EncoderSensor. impl
end EncoderSensor. impl;
processor Processor1
  features
    Encode_In:in data port;
    SpeedControl_Out:out data port;
  flows
    InProcessor_Flow:flow path Encode_In -> SpeedControl_Out {
      Latency => 4ms;
      };
end Processor1;
processor implementation Processor1. impl
end Processor1. impl;
device SpeedActuator
  features
    SpeedControl_In:in data port;
  flows
ActuatorIn_Flow:flow sink SpeedControl_In {
  Latency => 1ms;
  };
end SpeedActuator;
device implementation SpeedActuator. impl
end SpeedActuator. impl;
thread CalThread
  features
    Cal_In:in data port;
    Cal_Out:out data port;
end CalThread;
thread implementation CalThread. ThreadImpl1
  properties
    Compute_Execution_Time => 4ms,…,4ms;
    Period => 10ms;
    Deadline => 10ms;
    SEI::Priority => 1;
    Dispatch_Protocol => Sporadic;
end CalThread. ThreadImpl1;
thread Thread1
end Thread1;
thread implementation Thread1. ThreadImpl1
  properties
    Dispatch_Protocol => Periodic;
    Period => 10ms;
    Deactivate_Execution_Time => 3ms,…,3ms;
    Deadline => 10ms;
    SEI::Priority => 0;
end Thread1. ThreadImpl1;
thread Thread2
end Thread2;
thread implementation Thread2. ThreadImpl1
  properties
    Dispatch_Protocol => Periodic;
    Period => 10 Ms;
    Deactivate_Execution_Time => 2ms,…,2ms;
    Deadline => 10ms;
    SEI::Priority => 2;
end Thread2. ThreadImpl1;
process Process0
  features
    DataPort1:in data port;
    DataPort2:out data port;
end Process0;
process implementation Process0. ProcessImpl1
  subcomponents
    Th0:thread CalThread. ThreadImpl1;
    Th1:thread Thread1. ThreadImpl1;
    Th2:thread Thread2. ThreadImpl1;
end Process0. ProcessImpl1;
```

图 4　自动化管焊设备焊枪速度控制系统 AADL 模型代码

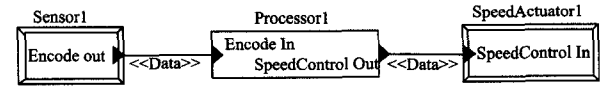


图 5　自动化管焊设备焊枪转速控制数据流模型

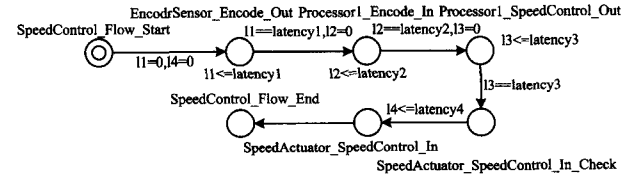


图 6　自动化管焊设备焊枪转速控制数据流时间自动机

自动化管焊设备焊枪转速控制数据流模型如图 5 所示。按照上述定义的转换规则转换得到的时间自动机模型如图 6 所示。转换实例说明如下：

1)端口 EncoderSensor. Encode_Out, Processor1. Ecode_In, Processor1. SpeedControl_Out, SpeedActua-tor_SpeedActuator_SpeedControl 分别转换为对应的状态 EncoderSensor_Encode_Out, Processor1_Ecode_In, Processor1_SpeedControl_Out, SpeedActuator_SpeedControl，并加入初始状态 SpeedControl_Flow_Start 和结束状态 SpeedControl_End。

2)AADL 模型定义了 l_1 至 l_4 4 个时钟，以及与其时延值相对应的整型变量 latency1 至 latency4。

3)数据流 SensorOut_Flow、ActuatorIn_Flow 和 Proces-

sor1 中的数据流 InProcessor_Flow 分别转换成相应的时钟赋值和判断条件，例如 InProcessor_Flow 连接的时延值为 4ms，在进入其源端口状态 Processor1_Encode_In 前，将相应的时钟 l_2 置 0，经过该状态后，到达目标端口状态 Processor1_SpeedControl_Out 的转移条件为 $l_2==$latency2，表示在延迟 4ms 时该转移发生。

4)在判断端到端的流 SpeedControl_Flow 时，在其源端口对应的状态 EncodrSensor_Encode_Out 前将 l_4 置 0，在加入的状态 SpeedActuator_In_Check 的转出条件中加入判断 $l_4<=$latency，表示满足约束条件，允许达到下一个端口状态。

3.2 混合数据流

混合数据流指数据由设备采集，流经计算线程或触发周期时间而产生。数据流时延对线程计算、系统调度进行了详细的描述。可调度性是指系统的调度策略和资源状况能否满足所有任务的执行要求，在任务截止时间到来之前执行并不发生死锁，是嵌入式实时系统正确执行的基本保证。混合数据流到时间自动机的转换规则和转换实例如下。

(1)转换规则

混合数据流到时间自动机的转换法则扩充为 $\Gamma_{Fp}(Fp)=\langle\overline{TA},Var_G,\phi',Ch\rangle$，即转换成一个时间自动机网络，具体的映射关系的扩充说明如下：

$Po_0\rightarrow TA$，根据源端口 Po_0 所在设备的派发形式和派发周期设计一个数据产生时间自动机，通过同步通道通知数据流时间自动机数据的产生。

$F_T\rightarrow\langle Ch,Var\rangle$，在指向线程输入端口对应状态的边上，设置数据流 F_T 时间自动机与非周期线程模板通信的同步通道，通知相应线程派发，在从线程输入端口状态转移到输出端口状态的边上设置线程执行完成的接收同步信号。

(2)转换实例

如图 7 所示，该速度控制系统是在图 5 的基础上做了扩充，数据流源头的数据采集设备 Sensor1 是周期工作的，数据信号的发送也是周期的。当数据流经了一个计算线程，在设定的计算时间后，由输出端口将结果返回数据流继续传递给控制器。这种结构下的数据流的时延必须考虑线程计算时间，而线程从触发到执行结束的时间不是一个确定的值，它与系统所处的调度状态有关。周期信号采集设备 Sensor1 发送数据的时间自动机如图 8 所示，当局部时钟 cl 每增长到设备的采样周期值时，就会产生一次数据的派发。混合数据流经转换规则生成的时间自动机如图 9 所示，进入线程输入端口状态的边上设置了通知线程派发的同步信号 dispatched[TId]!，TId 为对应线程的标号，计算完线程后，完成信号 complete[TId]?通知其进入线程输出端口的状态 CalThread_Cal_Out。

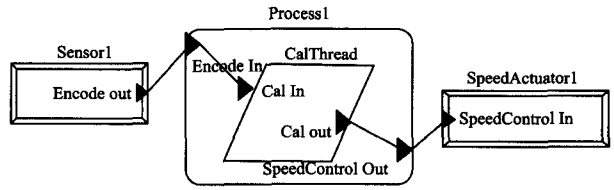


图 7 自动化管焊设备焊枪转速控制数据流模型

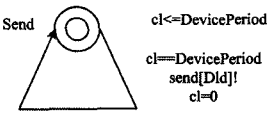


图 8 周期设备数据派发时间自动机

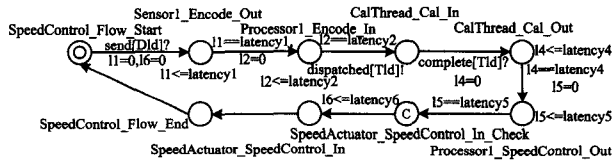


图 9 自动化管焊设备焊枪转速控制混合数据流时间自动机

对于由数据流连接起来的多个线程且线程间具有级联关系的较为复杂的情况，线程 1 产生的结果作为触发线程 2 派发的信号，从而导致数据流时延和对调度模型的影响具有更大的不确定性。这种 AADL 模型的转换仍然遵照以上的转换规则，线程转换为单独的时间自动机模板[21]，数据流按照基本的单一数据流转换并加入同步通道与线程通信。AADL 模型示例图和转换后的结果分别如图 10 和图 11 所示。混合数据流转换到的时间自动机对调度模型自动机的扩展并不影响处理器的工作，而只是通过时间自动机网络中的广播通道通信的方式，告知处理器和线程事件派发的到来。由这种扩展组成的整个 AADL 执行系统的时间自动机，将更好地模拟实际的调度，并得到更准确的数据流时延验证方法。

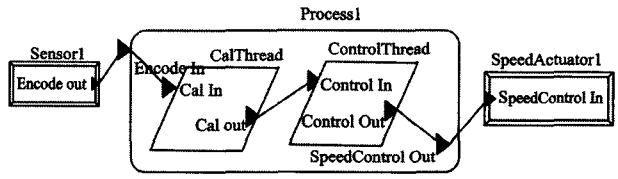


图 10 带有级联线程的混合数据流模型

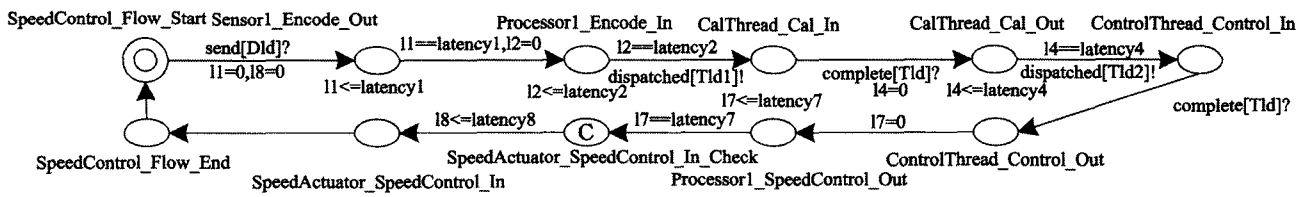


图 11 带有级联线程的混合数据流时间自动机

3.3 自动转换插件

AADL 建模工具 OSATE[22]是基于 Eclipse 平台的一套可视化工具，具有比较灵活的插件扩展功能。基于上节 AADL 数据流模型的转换方法，本文设计开发了自动化模型转换的插件 AADLToUPPAAL Plug-in 并将其嵌入到 OSTATE 工具中。该模型转换插件首先解析 AADL 模型文件，获取调度模型的相关信息，建立程序中 AADL 对象模型，按照转换法则将其映射为 Uppaal 工具支持的时间自动机模型并保存为文件，进而通过 Uppaal 工具打开模型并进行性质验

证。其输入输出关系如图 12 所示，该模型转换插件实现了 AADL 模型解析、模型转换和时间自动机模型生成 3 大功能模块。

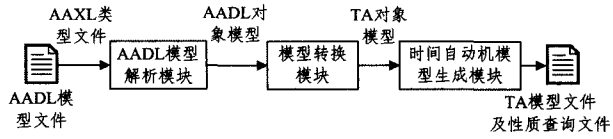


图 12　自动化模型转换插件模块输入输出流程

(1)AADL 模型解析。该模块负责读取建模工具中建立好的 AADL 模型，读入模型文件，按照语义信息和格式规范解析出 AADL 模型在转换中所需的基本信息，构建在插件程序中 AADL 的对象模型。

(2)模型转换。该模块实现了插件核心的转换功能，根据文本建立的模型转换规则，将读入的 AADL 模型信息对应的转换规则映射为时间自动的基本元素，构建程序中时间自动机的对象模型。转换模块还提供基本的模型检查功能，当所建立的模型缺失转换所需的基本信息时，它会将错误信息提示给用户。

(3)时间自动机模型生成。目标模型是指 UPPAAL 工具支持的时间自动机模型，它具有自己的模型文件格式，该模块解析了文件的格式，将模型转换模块得到的时间自动机对象模型转换为模型文件，实现了模型自动构建的功能。由于时间自动机在 Uppaal 工具中以状态图的形式展示，文件中也保存了图形的基本信息，因此目标文件生成模块还需要调整模型文件中状态和边的坐标信息。

3.4　性质验证语句

数据流验证所关心的性质主要是流逻辑的正确性和设计要求的时延特性。通过数据流到时间自动机转换规则及自动转换插件的设计，这两点性质可以通过时间自动机的建模和验证工具 Uppaal 进行模拟和验证。在模拟器中可以单步或者连续检查时间自动机状态的转移，观察状态序列的变化来模拟系统的运行，在验证器中利用时序逻辑 TCTL 可以验证一些关键性的性质。性质查询语言包含状态公式和路径公式，一个状态公式可以直接对模型中的某个状态的存在性进行验证。例如，一个简单的状态公式 $i=3$ 在任何一个 i 等于 3 的状态上为真。一个状态公式和状态转移中的守卫条件都是无副作用的表达式，与守卫条件不同的是它支持限制条件的析取。死锁作为一个特殊的状态公式仅仅包含一个关键字 deadlock，实际上它并不代表某个具体的状态，而是作为一个对状态转移路径的说明，若一个状态的后继状态不存在，那么对于时间自动机而言，它就进入一个死锁状态。在 Uppaal 中，死锁状态是用路径公式 $E\langle\rangle deadlock$ 来表示的。

Uppaal 为性质验证提供了一种 BNF 语法，$prop::=E\langle\rangle p|E[]p|A\langle\rangle p|A[]p$，其中 $E\langle\rangle p$ 用来验证满足状态公式 p 的状态从初始化位置是否可达，常常在可行校验中使用；$E[]p$ 和 $A[]p$ 为安全性公式，在 Uppaal 中，安全属性是以肯定的形式出现，记 p 为状态公式，用路径公式 $A[]p$ 表示在所有可达的状态中 p 都满足，而 $E[]p$ 表示存在一条路径，这条路径上所有的状态都满足 p；$A\langle\rangle p$ 为存活性公式，意味着某种情形最终总会发生。数据流模型验证所关注的性质如表 2 所列。

例如，E⟨⟩SpeedControlFlow. Speedcontrol_Flow_End 性质验证语句，若满足则说明整个数据流在模型转换的过程中已经正确地连接起来，能够形成逻辑上的通路，同时也证明了在该数据流上每个状态对应的时延约束是可以满足的，单独验证某端的时延特性也可以用 E⟨⟩SpeedActuator_SpeedControl_In_Check。

表 2　数据流性质验证语句

性质验证语句	验证性质说明
E⟨⟩SpeedContronFlow. Speedcontrol_Flow_End	数据流状态连接验证
E⟨⟩SpeedActuator_SpeedControl_In_Check	数据流上的状态时延验证
E⟨⟩SpeedActuator_SpeedControl_In_Check	单独验证某端的时延
E⟨⟩SpeedControl_Flow_start	数据流到始端时延验证
E⟨⟩SpeedControl_Flow_End	数据流到终端时延验证
E⟨⟩Processor. MissedDeadline	处理器超时状态检查

4　模型转换验证

为了对所设计的数据流转换方法和性质验证语句进行验证，设计了两组 AADL 模型的测试用例，利用 Uppaal 工具对其转换后的时间自动机模型进行验证。数据流验证的主要性质为流逻辑设计的正确性与时延要求，通过时间自动机的结束状态与端口检查状态的可达性可以实现这两个性质的验证。E⟨⟩SpeedControlFlow. Speedcontrol_Flow_End 表示整个数据流在模型转换的过程中已正确连接，可以形成逻辑上的通路；E⟨⟩SpeedActuator_SpeedControl_In_Check 表示点对应的时延约束是可以得到满足的。测试环境为 Inter(R) Core(TM) i3-3220 CPU@3. 30GHz，Window7(32 位)，Uppaal4. 1. 17。

测试 1：定义图 5 系统模型中 Encode_Out 到 Encode_In、Encode_In 到 SpeedControl_Out、SpeedControl_Out 到 SpeedControl_In 及 Encode_Out 到 SpeedControl_In 的时延，利用 E⟨⟩SpeedControlFlow. Speedcontrol_Flow_End 和 E⟨⟩SpeedActuator_SpeedControl_In_Check 性质验证语句分别对 SpeedControl_In 端口和 Encode_Out 的时延进行验证，验证结果显示两条性质均可以满足。

测试 2：定义图 7 所示系统中不同端口之间的混合数据流，分别设定各端的时延和线程的周期与执行时间，每个线程的执行优先级各不一样，运行调度的策略支持可抢占执行。计算分析可得，转换得到的时间自动机验证结果显示 E⟨⟩SpeedControl_Flow_End 的性质不满足，与实际情况相符。该用例转换得到的时间自动机模型在 Uppaal 模拟器 1 中单步运行或在模拟器 2 中连续运行，观察每个线程和处理器的状态转移和模板之间的数据通信，最后在验证器中验证 E⟨⟩Processor. MissedDeadline 性质，实验结果如图 13 所示。

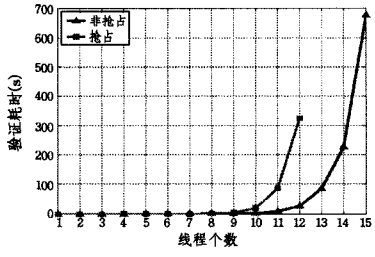
E⟨⟩ Processor. MissedDeadline
验证费时/kernel 费时/总费时：0. 031s/0. 047s/0. 075s
常驻内存/虚拟内存的使用峰值：6416kB/30960kB
不满足该性质

图 13　Uppaal 中验证器验证超时性质

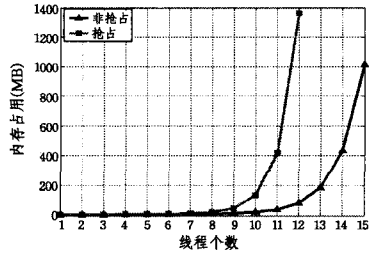
以上两个用例分别对单一数据流和混合数据流转换得到的时间自动机进行了验证，实验结果表明时间自动机可以正常运转，而混合数据流的时间自动机可以与调度模型时间自

动机构成的时间自动机网络进行综合验证。

为了分析时间自动机模型在 Uppaal 中的时间与空间性能，本文设计了一组包含 15 个线程的测试用例，每个线程的周期与截止时间均为 20ms，执行时间均为 1ms，优先级依次设为 0 到 14，性能测试中考虑了线程执行的非抢占与抢占两种情况。验证计算过程中 Uppaal 服务进程的 CPU 利用率稳定在 25%左右，如图 14 所示，耗时与内存消耗随着线程个数的增加而不断增长，这是由于 Uppaal 中的状态空间表示默认采用差值有界矩阵的数据结构，时间自动机的规模因线程个数呈几何式增长。另外，可抢占的时间自动机比非可抢占的更为复杂，故在时间与空间上消耗较高。



(a)验证耗时随周期线程个数的变化趋势



(b)内存随周期线程个数的变化趋势

图 14

结束语 本文建立了 AADL 数据流到时间自动机模型的映射，利用模型转换得到的时间自动机分析了单一数据流和混合数据流的时延性质，结合时间自动机构成时间自动机网络，来更好地模拟 AADL 模型所描述系统的运转情况，从而更准确地验证数据流时延的特性。最后给出了数据流时延性质查询语句的验证，实验结果表明模型转换方法是可行的。未来的研究中还需进一步扩展所设计的时间自动机以及模型转换法则，使其支持更复杂的 AADL 模型描述。

参 考 文 献

[1] Krishna C M. Real-Time Systems[M]. John Wiley & Sons, Inc. ,1999

[2] Kleppe A G, Warmer J B, Bast W. MDA explained, the model driven architecture: Practice and promise[M]. Addison-Wesley Professional, 2003

[3] Feiler P H, Gluch D P, Hudak J J. The architecture analysis & design language (AADL): An introduction[R]. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2006

[4] Feiler P H, Lewis B A, Vestal S. The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems[C]// IEEE International Symposium on Computer Aided Control System Design. Washington: IEEE Computer Society Press, 2006: 1206-1211

[5] Feiler P H, Hansson J. Flow latency analysis with the architecture analysis and design language (aadl): Technical Note CMU/SEI-2007-IN-010[R]. Software Engineering, Institute, 2007

[6] Lee S Y, Mallet F, De Simone R. Dealing with AADL End-to-end Flow Latency with UML MARTE[C]// 13th IEEE International Conference on Engineering of Complex Computer Systems, 2008(ICECCS 2008). IEEE, 2008: 228-233

[7] Jiao Ting-ting, Wang Le, Ye Guo-dong. Software Reliability Verification based on AADL[J]. Journal of Computer Application, 2012, 32(A02): 92-95

[8] Peled D. Software reliability methods[M]. Springer, 2001

[9] Amnell T, Fersman E, Mokrushin L, et al. TIMES: a tool for schedulability analysis and code generation of real-time systems [M]. Springer Berlin Heidelberg, 2004

[10] Fersman E, Mokrushin L, Pettersson P, et al. Schedulability analysis of fixed-priority systems using timed automata[J]. Theoretical Computer Science, 2006, 354(2): 301-317

[11] Zhang Y, Dong Y, Zhang Y, et al. A study of the AADL mode based on timed automata[C]// 2011 IEEE 2nd International Conference on Software Engineering and Service Science (ICSESS). IEEE, 2011: 224-227

[12] Wei L, Shuyu L. Research on the Formalization of AADL Model [C]// 2013 Fifth International Conference on Computational and Information Sciences (ICCIS). IEEE, 2013: 72-75

[13] Alur R, Dill D. The theory of timed automata[C]// Real-Time: Theory in Practice. Springer Berlin Heidelberg, 1992: 45-73

[14] Johnsen A. Architecture-Based Verification of Dependable Embedded Systems[D]. Mälardalen University, 2013

[15] Yovine S. Model checking timed automata[M]. Springer Berlin Heidelberg, 1998

[16] Alur R. Timed automata[M]// Verification of Digital and Hybrid Systems. Springer Berlin Heidelberg, 2000: 233-264

[17] Björnander S, Seceleanu C, Lundqvist K, et al. A Formal Analysis Framework for AADL[J]. The Journal of Science and Technology, 2011, 49(5): 105-118

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

[19] Alur R. Timed automata[M]// Computer Aided Verification. Springer Berlin Heidelberg, 1999: 8-22

[20] Gui S, Luo L, Li Y, et al. Formal schedulability analysis and simulation for AADL[C]// International Conference on Embedded Software and Systems, 2008(ICESS'08). IEEE, 2008: 429-435

[21] Abdeddaim Y, Maler O. Job-Shop Scheduling Using Timed Automata? [M]// Computer Aided Verification. Springer Berlin Heidelberg, 2001: 478-492

[22] Abdoul T, Champeau J, Dhaussy P T, et al. AADL execution semantics transformation for formal verification[C]// 13th IEEE International Conference on Engineering of Complex Computer Systems, 2008(ICECCS 2008). IEEE, 2008: 263-268