第37卷 第14期 计 算 机 工 程 2011年7月
Vol.37 No.14 Computer Engineering July 2011

·专栏· 文章编号：1000—3428(2011)14—0021—06 文献标识码：A 中图分类号：N945

一种 AADL 系统可靠性模型转换方法

高 磊，董云卫，张 凡，王广仁

(西北工业大学计算机学院，西安 710129)

摘 要：在原有构件依赖关系的基础上，提出一种架构分析与设计语言(AADL)系统可靠性模型的转换方法。该方法对 AADL 嵌入式系统体系结构进行可靠性建模，实现 AADL 可靠性模型到广义随机 Petri 网(GSPN)可靠性计算模型的转换。研究表明，该方法使 AADL 可靠性模型向 GSPN 模型的转换规则更加完备，能对嵌入式系统的可靠性进行准确与全面的分析与评估。

关键词：架构分析与设计语言；广义随机 Petri 网；可靠性模型；模型装换规则；可靠性评估

AADL System Reliability Model Transformation Method

GAO Lei, DONG Yun-wei, ZHANG Fan, WANG Guang-ren

(College of Computer, Northwest Polytechnical University, Xi'an 710129, China)

【Abstract】Based on the old component dependencies, this paper presents an Architecture Analysis and Design Language(AADL) system reliability model transformation method. It expands the dependencies to semantic connection, parameter connection and subcomponent calls and realizes the transformation to General Stochastic Petri Nets(GSPN). Therefore, it makes the model transformation rules from AADL reliability model to GSPN model more maturity and realizes exact and entire evaluation to reliability of embedded system.

【Key words】 Architecture Analysis and Design Language(AADL); General Stochastic Petri Nets(GSPN); reliability model; model transformation rules; reliability evaluation

DOI: 10.3969/j.issn.1000-3428.2011.14.006

1 概述

随着嵌入式系统的发展，嵌入式系统结构越来越复杂，规模越来越大，对系统的开发成本、开发周期及非功能属性(可调度性、可靠性、安全性)要求也越来越高，早期的嵌入式系统开发方法已经不能满足当前需求，为此，业界引入了模型驱动结构方法[1](Model Driven Architecture, MDA)，系统开发被提升到更高的级别——模型级，针对特定计算平台的编码工作由机器自动完成，模型成为开发过程中的核心。因此，新型嵌入式系统设计需要从模型级别对系统的非功能属性进行分析，从而可以在模型级别就对系统的可调度性、可靠性、安全性进行分析，如果系统的非功能属性不能满足需求，那么就可以从模型级别对系统架构进行修改，这样就能大大缩短系统的开发周期，节约开发成本[2]。

基于这个思想，美国自动化工程师协会(Society of Automotive Engineers, SAE)发布了航空标准 AS5506——架构分析与设计语言[3](Architecture Analysis and Design Language, AADL)。AADL 支持在非功能属性方面对嵌入式系统的软硬件体系进行分析，将硬件和软件抽象为不同的构件，构件间通过接口或调用实现数据的通信，它不关心构件的具体实现，通过软件构件、硬件构件和系统构件对实时嵌入式系统以及高可靠性系统进行了描述与分析。为了对嵌入式系统的可靠性进行描述，SAE 又发布了航空标准 AS5506/1——错误模型附录[4-5](Annex E: Error Model Annex)。此标准的提出完善了 AADL 建模语言，使其由原来单一的模型描述和验证增加了可靠性分析。

AADL 系统可靠性是一个静态的模型，为了对 AADL 系统架构模型进行可靠性分析和验证，需要采用形式化的方法对 AADL 系统架构可靠性模型进行动态描述，本文采用广义随机 Petri 网[6](GSPN)可靠性计算模型对 AADL 可靠性模型进行描述，然后通过对 GSPN 可靠性计算模型进行分析评估来验证 AADL 可靠性模型是否满足设计要求。

2 相关工作

前期工作中开展了一系列针对 AADL 软件架构模型的可靠性建模方法研究[7]，例如，利用 GSPN 模型、马尔科夫链、故障树模型等对 AADL 架构模型及 AADL 错误模型进行很好的描述，然而，基于 AADL 模型架构的可靠性模型转换方法工作量大，任务繁琐，需要将 AADL 语法中所有的语法元素转换到对应的 GSPN 模型，这样才可以利用 GSPN 模型计算工具在软件设计的早期对软件的可靠性需求进行分析和验证，从而提高软件的质量，节约开发成本，缩短开发周期。在文献[8]中描述了一部分 AADL 可靠性模型转换规则，这些模型转换规则包括 AADL 错误模型基本元素的转换、基本元素依赖规则 out propagation、in propagation、高级元素依赖规则 guard_in、guard_out 等，但这些规则不能涵盖所有的 AADL 语法元素，无法对 AADL 架构设计人员设计出来的系统模型进行充分并完备的转换，所以，本文在基本元素依赖规则和高级元素依赖规则的基础上，对其他 AADL 语法元素的转换规则进行描述，使得 AADL 模型与 GSPN 模型的模型转换理

基金项目：国家自然科学基金资助重点项目“高可靠构件化嵌入式软件设计与验证技术及其支撑环境研究”(60736017)；国家“863”计划基金资助项目“模型驱动的任务关键软件系统测试建模与支撑平台”(2007AA01Z147)

作者简介：高 磊(1984－)，男，硕士研究生，主研方向：嵌入式软件设计与验证，嵌入式计算；董云卫，教授；张 凡，讲师；王广仁，硕士研究生

收稿日期：2011-02-15 **E-mail：**gala_wlh@yahoo.com.cn

论更加完备,实现 AADL 软件架构模型的可靠性分析与验证。

3 AADL 可靠性模型研究

AADL 可靠性模型由 2 个部分组成：AADL 架构模型和 AADL 错误模型，AADL 架构模型以构件为基本单位，从软件和硬件 2 个方面综合描述一个系统，它主要描述了系统内各个构件的属性、构件间的连接以及软硬件构件的绑定关系，它侧重于描述构件实现怎样的功能，而并不关心怎样实现这些功能。构件的描述由类(type)和实现(implementation)构成，构件类中定义构件的属性，如端口、子程序、参数等，构件实现中定义构件实现的功能如包含的子构件、子程序调用、连接、流、模态等。AADL 错误模型主要描述了错误状态、错误事件、错误变迁以及相关参数等可靠性信息，错误模型与构件相对应，每一个构件都通过错误模型子附录与相应的错误模型联系，错误模型子附录通常在架构模型构件的实现中声明，由于错误是通过构件间的连接、软硬件的绑定或子程序调用进行传播的，需要在错误模型子附录中定义错误的过滤与屏蔽规则(guard_in、guard_out)和连接错误状态到模态机制 (guard_event 、 guard_transition 、 activate\deactivate transitions),这些语法规则对错误如何在构件间传播进行了很好的描述。

目前的 AADL 可靠性模型转换方法主要有 2 种，一种是将 AADL 可靠性模型转换为故障树(Fault Tree)模型，另外一种是转换为 GSPN 模型。故障树分析技术采用逻辑的方法，形象地进行风险分析工作，特点是直观、明了，思路清晰，逻辑性强，可以对系统做“从上到下”的定性定量的安全性和可靠性分析。基于故障树的模型转换理论是以 AADL 架构模型中的构件为基本事件，以构件间的连接关系确定基本事件以何种逻辑关系连接，故障树模型的因果关系清晰形象，对导致事故的各种原因及逻辑关系能做出全面、简洁、形象的描述，但若要对系统进行定量分析，必须事先确定各基本事件的发生概率，否则无法进行定量分析。另外，基于故障树的模型转换理论无法将 AADL 模型中的所有语法元素进行正确的描述，因此，不宜用这种方法实现 AADL 可靠性模型转换。

Petri 网是对离散并行系统的数学表示，适合于描述异步的、并发的计算机系统模型，它能够满足当前大多数系统所呈现出来的动态特性的描述要求。基于 GSPN 的可靠性模型转换理论是以 AADL 模型中构件的错误状态为基本元素，将错误模型中的错误状态对应转换为 GSPN 模型中的位置(place)，将错误事件转换为变迁(transition)，将错误状态的迁移转换为用弧(arcs)连接的从位置到变迁到位置的网络，初始错误状态在位置上标识有令牌(token)，令牌可以从一个位置移动到另一个位置，通过变迁的触发和令牌位置的改变来描述系统的动态行为。用 GSPN 作为可靠性模型转换的目标模型的优点是，首先可以根据每个子构件自身错误状态的变迁及变迁发生的概率分析出其可靠性状态，再根据系统内子构件间连接的方式确定错误传播对每个子构件和系统的影响，从而得出系统的可靠性数据。文献[9]已经将 AADL 可靠性模型向 GSPN 模型的转换形成了标准化形式化的转换理论，通过对这些理论的分析，实现了基本错误附录库、基本依赖关系元素和高级依赖关系元素向 GSPN 模型的转换，利用这些模型转换规则，可以实现从 AADL 系统架构模型到 GSPN 可靠性计算模型的模型转换，从而进行可靠性评估。

虽然前期的工作中已经实现了对 AADL 模型的可靠性评估，但是目前的可靠性模型转换理论存在一定的局限性和不完备性，当遇到 AADL 模型中构件间以语义连接相连时，便无法进行模型转换。因此，本文在研究这些转换理论的基础上，实现了其他 AADL 语法元素如子构件访问、子程序调用、子程序间的数据交互与共享、流[10]等向 GSPN 模型的转换，完善了 AADL 可靠性模型向 GSPN 模型的转换理论。

4 AADL 依赖规则的模型转换

4.1 基本依赖关系的模型转换

在 AADL 架构中，构件之间是通过端口和连接相互联系的，由于这种依赖关系的存在，造成了构件间的错误传播，有 2 种基本的基于连接的错误传播方式，即 out propagation 和 in propagation，本节讨论两者的联合 out-in propagation。

如图 1 所示，构件 1 产生的错误传播到构件 2，构件 1 是错误的发送者，它通过 Prop 发出一个错误，使得自身的状态从 Out_src 转移到 Out_dst，这个错误通过端口连接到达构件 2，构件 2 是错误的接收者，它通过 Prop 接收从端口连接上传送过来的错误，并从状态 In_src 转移到状态 In_dst。此处的错误传播是通过端口及连接实现的。

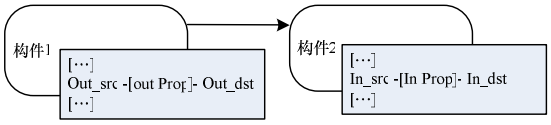


图 1 AADL 构件依赖关系的模型转换

4.2 高级依赖关系的模型转换

4.2.1 过滤与屏蔽规则 guard_in 和 guard_out

guard_in 允许用户对传播进入构件的错误进行控制，它只引入满足其过滤规则的错误，而其他情况下的错误全部屏蔽，其通过数据端口引入相应的错误。在这里，认定 Outprop 是一种传输状态，因为错误的传播是瞬间发生的，所以它是一种暂时状态。当 guard_in 控制错误传播时，发送错误的构件 1 由 Out_src 状态转换到 Out_dst，表示向外传播错误的节点 Outprop1 中便存在一个标识，这个标识配合另外一个发送构件 2 的 Outprop2 在满足语法规则的条件下，会促使接收错误的构件状态由 In_src 转换到 In_dst，如图 2 所示。

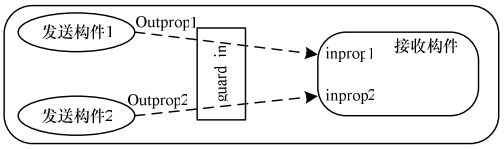


图 2 guard_in 图形描述

guard_out 用来描述当构件受到错误传播的影响时，如何向外传播错误。当满足错误模型子附录中错误状态的过滤条件时，构件自身才会发出一个错误 Outprop，每个 Outprop 对应的满足值为真的布尔表达式，而这个布尔表达式在 GSPN 模型中用瞬时迁移来描述。另外，错误处于传播状态是瞬时的，如果不能及时传播出去，则当前的错误会消失，因此，针对每一个 Outprop 应添加相应的吸收变迁。

4.2.2 错误状态与构件模式的连接机制

guard_event 与 guard_out 相似,两者的不同之处是 guard_event 不管构件内部状态如何变化，只激发一个对外事件，而 guard_out 可以产生多个错误输出。两者的 GSPN 模型也基本相同，不同的只是 guard_event 中构件激发的 Outprop 表示了一个事件。

guard_transition 是对状态转换的控制，促使状态转换的

端口在何种情况下发挥作用是由 guard_transition 中的控制条件制约的。构件模态在 GSPN 模型中用位置表示，在一定的条件下模态发生迁移，体现为 GSPN 模型中标识位置的变化。同样的，在 guard_event 中，每个瞬时迁移都表示了一个布尔表达式，这个瞬时迁移会促使构件模态发生变迁。

综合前面的模型转换规则可以看出，不论是基本依赖关系的模型转换还是高级依赖关系的模型转换，都是基于 AADL 架构模型中存在端口和连接实现的，这种转换规则的实现存在一定的局限性。在很多 AADL 架构模型中，不存在端口和连接，如子构件的访问、子程序调用等，但是这些模型的构件间存在数据和事件的交互，那么，当其中一个构件发生错误时，必然会有相应的错误传播去影响与它进行交互的构件，不能基于端口和连接实现这种类型的错误传播，所以，下文重点讨论在构件间没有连接存在的情况下，错误是如何传播的、相应的模型转换如何实现以及流(flows)对错误传播的影响。

5 AADL 可靠性模型转换规则的完善

5.1 子构件访问语法规则

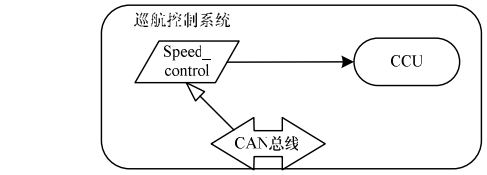
子构件访问的声明是通过关键字 provides(提供)和 requires(需要)在系统层级进行定义的，关键字 provides 描述了构件所能提供的访问权限，关键字 requires 描述了构件所需要的访问权限。在子构件访问中，构件间是基于语义连接进行通信的，所以错误传播可以发生在提供访问和需要访问的子构件之间。

5.1.1 总线访问

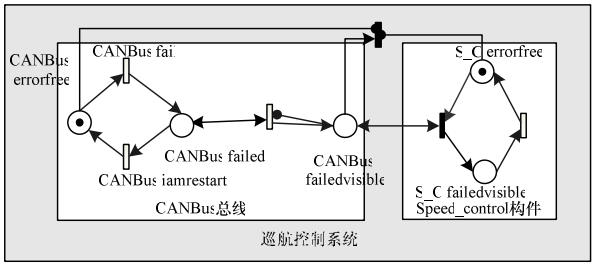
在总线访问中由总线构件提供访问权限，总线构件是硬件构件的一种，可根据 AADL 架构模型中提供访问的总线构件和相应的需要访问的构件来确定构件间的错误传播关系。通过扫描架构模型中的关键字 requires bus access 便可以确定错误传播的方向，即错误是从总线构件发出，传播到需要总线访问的构件，它们之间是一一对应的关系。利用 out-in propagation 规则就可以实现语义连接方式下错误传播的模型转换，如图 3 所示，对应 AADL 代码如下：

```
system implementation cruise_control_system.impl
connections
--data port connections not included
--bus access connections
bus_access : bus access CANBus -> Speed_control;
end   cruise_control_system.impl;
process Speed_control
features
speed_out : out data port;
bus_access : requires bus access CANBus.impl;
end Speed_control;
process implementation Speed_control.impl
annex Error_Model{**
  Model => errormodels::S_cEM.impl;
**};
end Speed_control.impl;
```

图 3(a)为一个巡航控制系统的 AADL 模型，在该系统中，进程 Speed_control 需要总线 CANBus 提供访问权限，失效事件从总线传播至速度控制构件，相应地为它们定义各自的错误模型。图 3(b)为该系统的 GSPN 模型，由于构件间的通信发生于语义连接上，构件 CANBus 发出一个错误 failedvisible，进程 Speed_control 接收到这个错误后自身状态由无故障状态跳转到错误状态，以此实现单一错误对系统内其他构件的影响。



(a)AADL 模型



(b)GSPN 模型

图 3 总线访问示例

如果一个构件同时需要 2 个或多个总线访问，并且没有过滤与屏蔽规则对其进行约束，则每个总线构件传播出的错误都可以引发该构件错误状态的变迁，如图 4 所示。

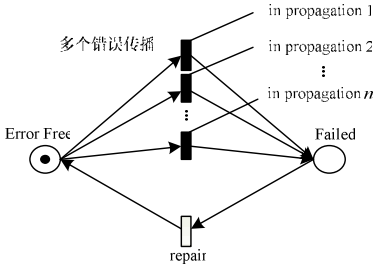


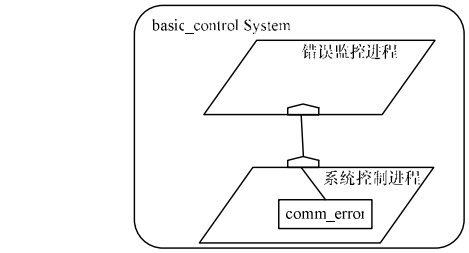
图 4 一对多的总线访问

5.1.2 数据访问

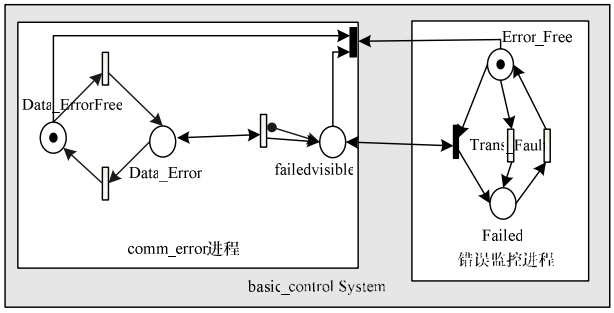
在数据访问中，提供访问的是数据构件，数据构件是软件构件的一种，在向外传播错误的同时，其自身错误状态也发生变迁。为数据构件定义 2 个最基本的错误状态，即 Data_ErrorFree、Data_Failed，分别代表数据构件是否发出错误数据的状态。在系统架构模型中，通过关键字 data access 确定错误传播的方向，为从提供数据访问的数据构件到需要数据访问的子构件。数据访问主要提供数据共享的功能，当一个数据构件为多个子构件提供数据访问时，将其拆分为一对一的数据访问来进行错误模型的转换，当同时有多个数据构件为一个子构件提供数据访问时，利用高级依赖关系控制和过滤随着数据访问传播的错误，如图 5 所示。对应的 AADL 代码如下：

```
system implementation basic_control.impl
subcomponents
control : process control.impl;
error_monitor : process error_monitor.impl;
connections
a_01:
data access control.error_log_data->error_monitor.error_data_in;
end basic_control.impl;
data implementation logs.error_logs
annex Error_Model{**
  Model => errormodels::Data_EM.impl;
  **};
end logs.error_logs;
…
```

```
error model Data_EM
  features
  Data_ErrorFree:initial error state;
  Data_Error:error state;
  Trans_Fault:error event{occurrence=>poisson 0.3};
  repair:error event{occurrence=>poisson 0.7};
  failedvisible:out error propagation{occurrence=>poisson 0.3};
end Data_EM;
…
```



(a)AADL 模型



(b)GSPN 模型

图 5 AADL 模型数据访问的模型转换

图 5(a)为一个底层控制系统的 AADL 模型，图 5(b)为该系统对应的 GSPN 描述。在该系统中，进程 error_monitor 需要来自数据构件 comm_error 的数据访问，而数据构件 comm_error 是嵌套在进程构件 control 中的，这是一种跨越了系统层次结构的访问调用。comm_error 与 control 之间是存在语义上的连接的，尽管这个连接经过进程构件 control 的端口，但是基于语义连接进行的通信是不会对构件 control 造成影响的，所以在语义连接上传播的错误不会对它的上层构件的错误状态变迁造成影响，这个发出的错误只会影响到需求数据访问的构件，即进程 control 的错误状态不会被数据构件 comm_error 发出的错误影响。

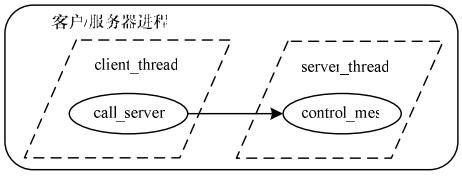
综合上述 2 种情况可以看出，在子构件访问这种语法规则中，错误是从提供访问的构件传播到需要访问的构件，提供访问的构件发出错误，需要访问的构件接收这个错误，并且其自身错误状态也发生变化，由于错误传播只发生在语义连接上，只要分别为语义连接两端的构件定义相应的错误模型，就可以对整个系统进行可靠性模型转换。另外还需要根据 AADL 模型描述，当有多个构件参与到子构件访问的过程中时，就需要利用高级依赖规则来处理构件间的错误传播，常用的高级依赖规则为 guard_in 和 guard_out。

5.2 子程序调用语法规则

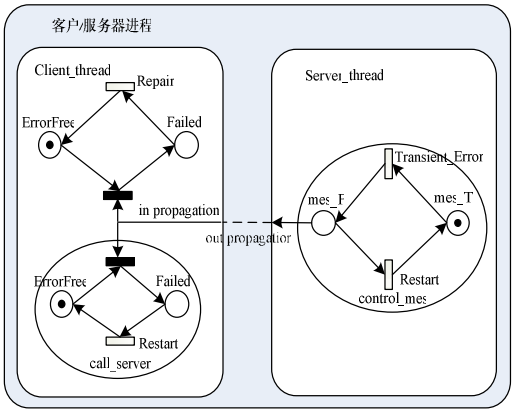
在 AADL 架构模型中，子程序构件抽象地描述了顺序执行的源程序段，它主要描述了数据操作的方法调用、基本的程序调用以及远程服务的调用，子程序是软件构件的一种。调用过程只能在线程实现或子程序实现中声明，其他类型的构件中不允许有子程序调用，调用可以发生在构件与子程序之间，也可以发生在 2 个子程序之间。子程序调用利用语义上的连接将线程与所要调用的子程序连接在一起，实现两者的交互及错误的单向传播。在子程序调用中，错误是由被调用的子程序传播到调用该子程序的线程或子程序。与数据访问不同之处在于，子程序调用中的错误传播不仅影响调用的子构件，该子构件的上层线程构件也会受到错误传播的影响。子程序构件的错误模型需要根据构件实现的功能进行具体定义，但应包括错误模型中的基本元素错误状态、错误事件、变迁，如图 6 所示。该系统的部分 AADL 架构代码描述如下：

```
  process implementation client_server.impl
  subcomponents
  client_thread: thread client_thread.impl;
  server_thread: thread server_thread.impl;
  properties
  Actual_Subprogram_Call => reference server_thread.control_mes
applies to client_thread.call_server;
  end client_server.impl;
  …
  thread implementation client_thread.impl
  calls    {      call_server: subprogram service_it;
           };
  end client_thread.impl;
  …
  thread server_thread
  features
  control_mes: server subprogram service_it;
  end server_thread;
  …
  error model control_EM
    features
    mes_T:initial error state;
    mes_F:error state;
    mes_Error:out error propagation;
    Transient_Error:error event{occurrence=> poisson 0.4};
    Restart:error event{occurrence=>poisson 0.6};
  end control_EM;
  …
```

AADL 子程序调用的模型转换如图 6 所示。



(a)AADL 模型



(b)错误传播示意图

图 6 AADL 子程序调用的模型转换

图 6(a)为一个客户/服务器进程。在该进程中，2 个子构件分别为客户线程 client_thread 和服务器线程 server_thread，客户线程的子程序 call_server 调用服务器线程的子程序 control_mes，即通过子程序调用，服务器将控制信息传递给客户，实现客户与服务器的交互。在该系统中，接收错误的子程序为 call_server，同时客户端线程 client_thread 也同样会受到该错误的影响而导致自身状态的变迁。图 6(b)为该进程中错误传播的示意图，分别为子程序 call_server 和 control_mes 定义了不同的错误状态及错误事件，可以看出，control_mes 发出的错误同时被 call_server 和线程 client_ thread 接收，这种错误传播类似于 out_in_propagation。

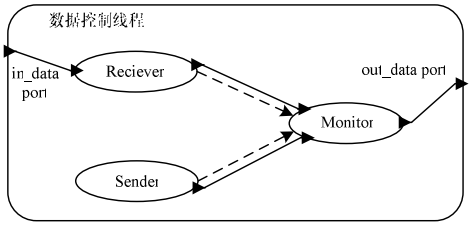
5.3 数据交互与共享语法规则

在 5.2 节的子程序调用中，由于构件及子程序上没有端口的存在，因此错误传播的处理方式比较简单。而在子程序间的数据交换与共享中，数据的交互是通过参数和连接实现的，需要根据相应 AADL 架构模型将基本依赖关系与高级依赖关系结合来处理其中的错误传播。

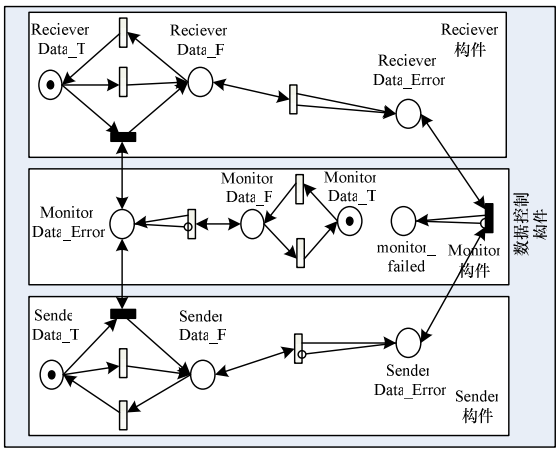
子程序中的 parameter(参数)类似于构件中的数据接口，它描述了传入或传出子程序的数据值，参数是在子程序的类中声明的。参数可以与数据接口相连，也可以与事件数据接口相连。子程序间参数连接的方式决定了如何处理错误传播，若子程序间是通过参数一对一的进行连接，便利用基本依赖规则处理错误传播；若子程序间是通过参数多对一的连接，则利用高级依赖规则处理错误传播，如图 7 所示。对应的 AADL 代码如下：

```
thread data_control
features
in_data: in data port;
out_data: out data port;
monitor: server subprogram service_it;
end data_control;
thread    implementation data_control.impl
calls    {
                reciever: subprogram service_it;
                sender: subprogram service_it;
        };
--connections
--parameter in_data -> reciever.in_parameter;
--parameter reciever.out_parameter -> service_it.in_mes1;
--parameter sender.out_parameter -> service_it.in_mes2;
--parameter service_it.out_mes -> out_data;
end data_control.impl;
…
subprogram implementation service_it.impl
annex Error_Model{**
  --
  guard_event =>
  in_mes1[data_error] and in_mes2[data_error]
  applies to monitor_failed;
  **};
end service_it.impl;
```

图 7(a)为一个数据控制线程，图 7(b)为该线程对应的 GSPN 模型。它通过自身的接口 in_data 接收数据，经过内部 3 个子程序的处理，由子程序 monitor 生成数据并通过接口 out_data 向外发送数据。在该线程中不仅存在参数连接(图中实线部分)，还存在子程序调用(图中虚线部分)。在参数连接中，几个子程序间的错误传播用高级依赖规则 guard_event 来处理；同时在子程序调用中，子程序 reciever 和 sender 分别接收来自 monitor 的错误传播。为这 3 个子程序定义相同的错误模型，错误状态为 Data_T、Data_F，错误事件为 Transient_Error、Restart，向外传播错误 Data_Error，另外，子程序构件 monitor 还有个由高级依赖规则 guard_event 生成的错误事件 monitor_failed，通过参数连接和子构件调用来确定系统中的错误传播。



(a)AADL 模型



(b)GSPN 模型

图 7 AADL 数据构件模型转换

5.4 AADL 流语法规则

在 AADL 架构模型中，流就是一个逻辑存在的数据和控制信息流，它由一个顺序序列的线程、处理器、设备和端口连接或数据访问连接构成，流的 3 个基本要素为流源(flow source)、流目的(flow sink)、流路径(flow path)，其中，流源和流的目的为系统内子构件的某个端口，流路径是从源端开始经过端口和构件到目的端的一条路径。定义端到端的流的目的就是提供一种特别的方式来分析可靠性及错误数据的传播，由于流路径是经过多个构件最终到达流目的的，因此重点处理错误传播经过流路径以后对整个系统的可靠性造成怎样的影响。如果流路径上源构件传播出一个错误，那么处在流路径上的所有子构件都会受到错误传播的影响，即从源端构件传出一个错误，构件 1 接收这个错误并发出另外一个错误由构件 2 接收，以这种方式错误传播到目的端构件，如图 8 所示。

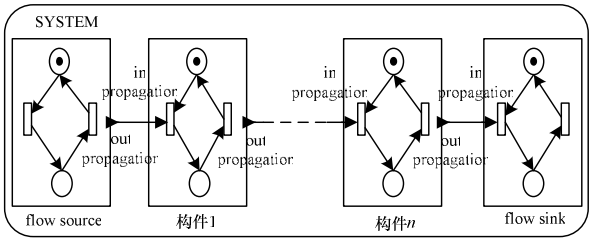


图 8 AADL 流中的错误传播

端到端的流不仅可以按照存在于线程间、处理器间、设

备间语义上的连接进行，还可以按照子程序间的参数连接进行，这使得系统内用来描述逻辑信息的端到端的流可以将通过某个构件的流信息输入改变为不同形式的输出，例如，数据、事件数据端口，或数据访问、数据类型都可以发生改变。同样的，对于错误模型中的错误传播来说，流源构件发出的错误，经过流路径上的构件，最终目的构件接收到的错误已经不是源构件发出的错误，而是目的构件之前的子构件发出的错误传播，即在图 8 中，每个子构件接收到的 in propagation 都是不相同的，它们发出的 out propagation 也是不相同的，这种错误传播规律符合前面提到的流信息改变的规律。图 8 是单一流路径的错误传播，如果一个子构件处于 2 条或多条流路径上，在其错误模型子附录中就要相应地定义高级依赖规则来控制通过该构件的错误传入和传出。

6 结束语

本文介绍了 AADL 架构模型向 GSPN 可靠性模型的模型转换，对 AADL 依赖规则的模型转换进行介绍，基于这些基本的依赖关系转换规则，对 AADL 其他语法规则的模型转换如子构件访问、子构件调用、子程序间的数据交互和共享等进行了讨论和研究，完善了 AADL 可靠性模型转换规则，并在已有的 AADL 可靠性评估工具中添加这些语法元素的转换规则，使可靠性评估工具更加完备。虽然本文对 AADL 可靠性模型转换规则进行了扩充和完善，但是 AADL 语言还在不断更新其语法标准，添加新的语法元素和规则，后期的工作和研究重点在于随着 AADL 标准的更新扩充相应的 AADL 可靠性评估模型到 GSPN 可靠性计算模型的模型转换规则，完善可靠性评估工具使其满足可靠性评估的要求。

参考文献

[1] OMG, Inc.. Model Driven Architecture(MDA)[EB/OL]. (2009-01-03). http://www.omg.org/mda.

[2] 刘晓娟, 温冠华, 李建军, 等. 软件可靠性度量方法[J]. 计算机工程, 2009, 35(3): 57-59.

[3] Feiler P H, Gluch D P, Hudak J J. The Architecture Analysis & Design Language(AADL): An Introduction[EB/OL]. (2006-02-06). http://repository.cmu.edu/sei/406/.

[4] Feiler P, Rugina A. Dependability Modeling with the Architecture Analysis & Design Language(AADL)[EB/OL]. (2007-07-08). http://www.sei.cmu.edu/library/abstracts/reports/07tn043.cfm.

[5] International Society of Automotive Engineers. SAEAS5506/1-2006 SAE Architecture Analysis and Design Language(AADL) Annex Volume 1, Annex E: Error Model Annex[S]. 2006.

[6] Bause F, Kritzinger P S. Stochastic Petri Nets: An Introduction to the Theory[M]. Berlin, Germany: Friedrich Vieweg & Sohn Verlag, 2002.

[7] NASA Office of Safety and Mission Assurance. Fault Tree Handbook with Aerospace Applications[EB/OL]. (2002-08-01). http://www.skybrary.aero/bookshelf/books/394.pdf.

[8] Zhang Chenyu, Yang Zhiyi, Dong Yunwei. Research and Assessment of the Reliability of a Fault Tolerant Model Using AADL[C]//Proc. of ASEA'08. Sanya, China: [s. n.], 2008: 45-52.

[9] Rugina A E. Dependability Modeling and Evaluation——From AADL to Stochastic Petri Nets[D]. [S. l.]: Institut National Polytechnique de Toulouse, 2007.

[10] Feiler P, Hansson J. Flow Latency Analysis with the Architecture Analysis and Design Language(AADL)[R]. [S. l.]: Software Engineering Institute, Tech. Rep.: CMU/SEI-2007-TN-010, 2007.

编辑 任吉慧

(上接第 20 页)

表 4 表明在同一数据集选择不同的类作为异类时，算法的识别率有不同程度的提高。这也说明了对于正类异类数据分布情况不同的数据集，对于算法的异常检测结果会有不同的影响，本文算法虽然也表现出随数据集特征变化，识别率有所不同的特点，但相比 2 种参考算法表现出更好的泛化性和稳定性。

表 5 表明在训练数据和测试数据均较少的情况下，本文的算法仍然提高了算法的稳定性使得识别率得到了提高。

本文提出的算法，在合适的 t 和 b 的选择下，可以达到较好的效果，不过根据数据集特点的不同提高的幅度略有不同。因此，模糊决策法的提出在一定程度上解决了算法中的一些问题，使异常检测算法提高了稳定性和适应度。

5 结束语

本文将在核化空间深度的异常检测算法和最小包围核的算法中引入了一种较为准确的模糊决策方法进行异常检测。实现了算法的融合，达到扬长避短、提高算法泛化能力和算法稳定性的目的。

参考文献

[1] 罗 敏, 阴晓光. 基于核函数的入侵检测方法研究[J]. 计算机应用研究, 2007, 24(12): 162-164.

[2] 任勋益, 王汝传. 基于主元分析和支持向量机的异常检测[J]. 计算机应用研究, 2009, 26(7): 2719-2722.

[3] 张 剑, 龚 俭. 异常检测方法综述[J]. 计算机科学, 2003, 30(2): 312-314.

[4] 孙美凤, 黄 飞, 陈云菁, 等. 基于特征模式的马尔可夫链异常检测模型[J]. 计算机工程, 2008, 34(24): 155-156.

[5] Chen Yixin, Dang Xin, Peng Hanxiang, et al. Outlier Detection with the Kernelized Spatial Depth Function[J]. Pattern Analysis and Machine Intelligence, 2009, 31(2): 288-305.

[6] Shawe-Taylor J, Cristianini N. Kernel Methods for Pattern Analysis[D]. Cambridge, UK: Cambridge Univ. Press, 2004.

[7] Wu Mingrui, Ye Jieping. A Small Sphere and Large Margin Approach for Novelty Detection Using Training Data with Outliers[J]. Pattern Analysis and Machine Intelligence, 2009, 31(11): 2088-2092.

[8] Tax D M J, Duin R P W. Support Vector Data Description[J]. Machine Learning, 2004, 54(1): 45-66.

[9] 李本海, 张序君. 分类问题的模糊决策分析[J]. 模糊系统与数学, 1992, 6(1): 81-86.

编辑 索书志