第 46 卷第 6 期
2014 年 12 月

南 京 航 空 航 天 大 学 学 报
Journal of Nanjing University of Aeronautics & Astronautics

Vol. 46 No. 6
Dec. 2014

一种基于 AADL 的 IMA 系统配置信息的正确性检测方法

胡 军[1,2] 马金晶[1] 袁 翔[1] 刘 雪[1]

(1. 南京航空航天大学计算机科学与技术学院,南京,210016;
2. 南京大学计算机软件新技术国家重点实验室,南京,210093)

摘要:综合模块化航空电子系统(Integrated modular avionics, IMA)中的系统配置信息的正确性是保证 IMA 系统可靠运行的重要保障。配置信息的重配置给系统的更新和移植提供了方便,同时也给重配置后的系统带来了不安全因素。本文针对满足 ARINC653 规范的 IMA 系统重配置信息的正确性检测方法,展开了基于架构分析和设计语言(AADL)模型转换与分析的研究。给出了一系列从 ARINC653 系统配置信息到 AADL 模型元素的映射规则,包括模块、分区、进程、健康监控、通信等核心概念,并设计了一个模型转换的方法,然后采用一个第三方的工具对所得到的 AADL 模型展开配置信息正确性的语义验证。最后本文还给出了一个实例分析。

关键词:软件可靠性测试;综合模块化航电系统;ARINC653 配置信息;AADL;REAL;OSATE

中图分类号:TP311 **文献标志码**:A **文章编号**:1005-2615(2014)06-0920-11

Correctness Verification for Integrated Modular Avionics System Configuration Based on AADL Model

Hu Jun[1,2], *Ma Jinjing*[1], *Yuan Xiang*[1], *Liu Xue*[1]

(1. College of Information Science and Technology, Nanjing University of Aeronautics & Astronautics, Nanjing, 210016, China; 2. State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210093, China)

Abstract: The configuration information correctness of the integrated modular avionics (IMA) system is an important guarantee of operational reliability. Reconfiguration of configuration information provides conveniences for system updating and transplantation, and brings unsafe factors to system. Concerning the detection method for the correctness of the IMA system which satisfies ARINC653 specification, the transformation and analysis of the architecture analysis and design language (AADL) model are researched. Elements mapping rules from ARINC653 system configuration information to the AADL model are proposed, including module, partition, process, health, monitoring, communication and other core concepts. A model transformation approach is given, and a formal semantic verification of the configuration information correctness of AADL model is presented based on a third-party tool. Finally, an example analysis is provided.

Key words: software reliability testing; integrated modular avionics; ARINC653 configuration information; AADL; REAL; OSATE

基金项目:国家重点基础研究发展计划("九七三"计划)(2014CB744904)资助项目;回国留学人员科研启动基金(2012)资助项目;611 航空科研基金(2012)资助项目;南京航空航天大学青年科技创新基金(NS2014098)资助项目。

收稿日期:2013-07-06;**修订日期**:2013-10-11

通信作者:胡军,男,副教授,E-mail:hujun@nuaa.edu.cn。

综合模块化航电系统(Integrated modular avionics, IMA)[1]是近些年来航空应用领域中出现的一类重要系统结构。ARINC653 标准[2]定义了一种 IMA 软件体系结构,制定了操作系统层和应用软件层之间的标准接口(APEX)。满足 ARINC653 标准的 IMA 系统称之为 ARINC653 系统。

ARINC653 系统的配置信息是 ARINC653 软件体系结构的重要组成部分,它包含了软件体系结构中所有层次的相关信息,用来对 IMA 系统中硬件接口、操作系统和应用程序进行参数配置。针对系统硬件和软件的改变,可以通过修改配置信息使得已有的系统能在新的环境下正确运行。如何确保修改后的配置信息的正确性是当前 ARINC653 系统领域的一个重要问题,也是本文的主要研究内容。

架构分析和设计语言(Architecture analysis and design language, AADL[3])是一种针对嵌入式系统建模描述语言,可应用于航空嵌入式系统的架构设计建模及分析[4]。本文提出了一种基于 AADL 模型的 ARINC653 配置信息验证和分析的方法,确保 ARINC653 系统重配置后的配置信息正确性。

1　相关工作

与本文相关的研究工作主要包括 AADL 和 ARINC653 系统两个方面,其中:AADL 由于出现较早,以及其在嵌入式系统中的应用,目前国内外已存在较多的研究工作[3,5-6],其相应的开源工具也有不少。如:本文使用的 OSATE 工具能有效编辑、调试 AADL 模型文件,并能图形化、实例化系统模型,为 AADL 的研究分析及建模提供了良好的操作平台。

关于 AADL 和 ARINC653 系统之间的转换工作并不多见,目前的研究主要关注在如何从 AADL 模型到 ARINC653 系统实现之间的转换。如:文献[7]提出了基于 AADL 的中间件生成技术,并详细介绍了 Ocarina 工具。Ocarina 支持从 AADL 模型生成运行在 PolyORB, PolyORB-HI (PolyORB-high integrity), POK(ARINC653 系统所采用的中间件)中间件之上的 Ada, C 分布式应用代码,即自动化的生成 ARINC653 系统进程运行代码。基于 AADL 各类软件构件,生成对应的源代码,如线程构件到 C 代码的转换。文献[6]研究了 AADL 到 RTSJ 的代码转换,RTSJ 被认为是未来航天应用系统的执行内核。

关于 ARINC653 配置信息的相关工作,目前主要的关注也是在如何从 AADL 模型到 ARINC653 配置信息生成的方法。如:文献[8]设计了配置文件自动生成工具,并将其定制成插件的形式集成到 OSATE 开发平台下,以辅助其用 AADL 语言为 ARINC653 标准的航空电子系统进行建模。而对于配置信息的逆向抽取建模并进行分析,目前尚缺乏这方面的工作。此外,关于配置信息的验证方法,文献[9]介绍了将 AADL 模型转换为 Petri 网进行形式化验证的方法,给出了一种与本文方法不同的模型验证的方法。对于 ARINC653 系统,研究基于形式化的方法进行可靠性验证也是今后研究的一个方向。

2　ARINC653 和 AADL

2.1　ARINC653 概述

ARINC653 是一种 IMA 软件体系结构,如图 1 所示。它包括如下的几个层次结构:应用软件层、ARINC653 系统结构层、实时操作系统层、硬件接口层和硬件层。系统的功能要求包括分区管理、进程管理、时间管理、存储器管理、分区内通信、分区间通信及健康监控等功能定义。

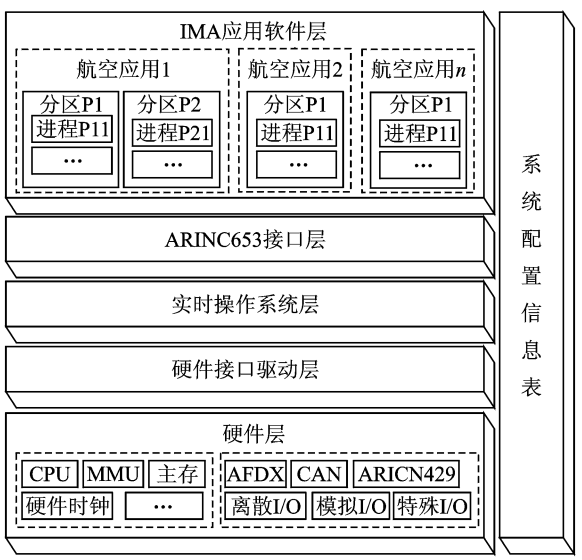

图 1　ARINC653 软件体系结构

Fig. 1　ARINC653 architecture

系统配置信息是 ARINC653 系统非常重要的组成部分,包括了以上所有层次的相关信息以及参

数配置。配置信息通常包括模块级和分区级两大类,分别描述分区间和各分区内的资源配置情况。文献[10]具体介绍了系统配置信息表,包括13个模块级和16个分区级配置表。表1给出了一个模块级通用配置信息表的例子,它指明了当前模块的总体信息。

表1 通用配置信息表

Tab. 1 Table of general configuration information

PARTION_NM	模块分区数目
SYSTEM_PARTION_NM	模块分区数目
MAF_DURATION	模块调度分配时间
CACHE_CONFIG	全局缓冲区配置数据
RAM_BEGIN	模块的内存起始地址
RAM_SIZE	模块的内存大小
CFG_AREA_BEGIN	模块配置信息区域的内存起始地址
CFG_AREA_SIZE	模块配置信息区域的内存大小
MAC_ADDRESS	模块MAC地址
MOUDLE_LOCATION	模块全局ID

配置信息可以有多种数据保存格式,ARINC653规范[11]使用的是XML数据格式,其他的数据格式有Excel,CSV等。配置信息内容与数据格式无关,本文在ARINC653标准中的XML数据格式的基础上进行研究。

使用配置信息可以提高系统的可移植性和可重用性,对已有配置信息的重配置,使系统能够适应新的设计需求(如硬件的改变,分区资源分配的改变,增加新的应用模块等)。但是对于重配置后的系统,需要确保系统的正确性和可靠性,比如给新添加的分区分配内存需求,而分区所在模块已经没有内存可以分配,这样新系统是不安全的。为了确保系统的安全性和可靠性,对重配置后的配置信息的正确性进行验证是必要的。

2.2 AADL概述

AADL是用来设计和分析系统的软硬件结构,包括独立的组件和它们的交互,尤其适用于性能关键的实时嵌入式系统。2009年1月发布了AADL V2.0版本[5],新版本对子程序建模、异步系统语义和动态加载语义、虚拟总线和协议建模等进行了改进和扩展。

AADL通过组件、连接等概念描述实时嵌入式系统的软、硬件体系结构;通过特征、属性描述系统功能与非功能性质;通过模式变换描述运行时体系结构演化;通过用户自定义属性和扩展附件库来对更加复杂的系统概念进行建模。AADL提供了3种建模方式:文本、XML以及图形化,其包含的组件和特征元素的图形化如图2所示。

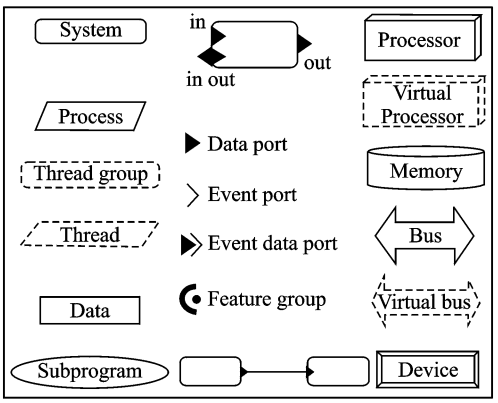

图2 AADL元素图形化形式

Fig. 2 Graphical representation of AADL elements

当现有的AADL组件和属性不能满足用户需求时,AADL引入了附件的概念。它拥有独立的语法和语义,但必须与AADL核心标准保持语义一致。如故障模型附件[12],支持构件、连接的故障事件、故障概率等属性建模;行为附件[13]增强了AADL对构件实际功能行为的详细描述能力,以更好地支持功能行为验证和自动代码生成。ARINC653附件[14]通过扩展属性集,将相应的组件和属性与ARINC653系统中的功能概念相对应,使得AADL具有对ARINC653系统进行建模的能力。本文就是基于这样的建模能力,将重配置后的系统配置信息转换成AADL模型。

3 ARINC653系统配置信息转换为AADL模型

3.1 转换规则

系统配置信息转换为AADL模型的核心内容,就是找出配置信息中的参数信息转换为AADL模型组件和属性等概念的规则。由于ARINC653配置信息包含的参数信息非常繁杂,本文的研究重点是7种核心概念(模块,分区,进程,分区间通信,分区内通信,内存,健康监控),对于其他的次要概念(硬件通道,AFDX虚拟连接等)的研究需要以后进一步研究。下面介绍这7种核心概念的转换规则和一个转换实例。

3.1.1　模块转换规则

考虑到 ARINC653 模块包含一个或多个航空电子应用软件,并保证这些应用软件之间相互独立运行。在配置表中是将运行在模块上的多个应用软件按功能划分为多个分区,同时指明每个分区分配的系统资源和分区内的调度信息。而 AADL 中的 processor 组件可以用来指明系统运行时的执行环境,包括 CPU 的调度分配、内存分配和通信连接总线等。

因此,可以将 ARINC653 模块转换为 AADL 的 processor 组件,这个组件提供了模块运行时的资源需求以及空间和时间隔离。具体而言一个 ARINC653 模块的模块名可以用组件属性 Deployment::ASN1_Module_Name 来定义,模块调度总时间片可以用扩展组件属性 ARINC653::Module_Major_Frame 来定义。

3.1.2　分区转换规则

ARINC653 分区包含了一组实现相同功能的应用软件,它在时间和空间上是隔离的,不同分区的应用软件的运行是不受影响的。分区可以拥有不同的关键级别来保护分区间的数据的流通。在模块的总时间片内,分区被调度的周期和执行时间是固定的,同时,基于空间隔离的需求,分区被分配给不同的地址空间。

考虑到 AADL 中的 virtual processor 组件可以用来构建一个逻辑资源,概念上是 processor 组件的一个子组件,可以将 processor 组件指明的执行环境分成好几个互不影响的区域。每个逻辑资源可以指明调度信息以及内存分配情况等。此外,process 组件指明了具体执行条件、程序段和交互数据,它包含一个 thread 组件来表示执行的动作。

可以用 AADL 中的以上两种组件来表示 ARINC653 模块中分区概念,其中 virtual processor 组件指明系统运行时资源的分配(任务调度、分区资源等),而 process 组件指明了分区包含的内容(线程,数据等)。这两种组件之间的关联通过 AADL 属性 Actual_Processor_Binding 定义。processor 组件包含一组 virtual processor 组件,就像模块包含一组分区一样。具体的转换规则细节见表 2。

3.1.3　进程转换规则

ARINC653 进程是系统执行主体,它包含了执行代码、执行数据和堆栈区域等。一个分区可以包含多个进程来实现相应的应用功能。分区通过指明进程的调度策略、抢占策略、最大响应时间、内存分配情况等信息来控制进程的执行。

表 2　ARINC653 与 AADL 元素转换规则

Tab. 2　Transformation rules between ARINC653 and AADL

ARINC653	AADL
模块	处理器组件
—模块名	—处理器组件属性 Deployment::ASN1_Module_Name
—模块内存需求	—内存组件
—模块调度周期	—处理器组件属性 ARINC653::Module_Major_Frame
分区	虚拟处理器和进程组件
—分区名	—虚拟处理器组件属性 Deployment::ASN1_Module_Name
—分区内存需求	—处理器组件属性 Actual_Memory_Binding 绑定进程组件与子内存组件
—分区关键级	—虚拟处理器组件属性 ARINC653::DAL
—分区执行起始时间	—处理器组件属性 ARINC653::Partition_Slots
—分区执行时间	—处理器组件属性 ARINC653::Slots_Allocation
—分区调度协议	—进程组件属性 Thread_Properties::Scheduling_Protocol
—分区分配协议	—进程组件属性 Thread_Properties::Dispatch_Protocol
进程	线程组件
—进程名	—线程组件属性 Deployment::ASN1_Module_Name
—进程堆栈大小	—线程组件属性 Memory_Properties::Source_Stack_Size
内存	内存组件
—内存类型	—内存组件属性 ARINC653::Memory_Kind
—内存大小	—内存组件属性 Memory_Properties::Byte_Count
分区内通信	进程内连接
—缓冲区	—事件数据端口
—黑板	—数据端口
—信号量机制	—共享数据组件
分区间通信	进程间连接
—采样端口(数据流方向,采样频率)	—数据端口(端口类型,端口属性 ARINC653::Sampling_Refresh_Period)
—队列端口(数据流方向,队列大小)	—事件数据端口(端口类型,端口属性 Communication_Properties::Queue_Size)
健康监控	处理器,虚拟处理器和线程组件属性
—故障类型	—相应组件属性 ARINC653::HM_ERRORS
—故障恢复行为	—相应组件属性 ARINC653::HM_Actions
—故障级别	—属性所属组件类型

而 AADL 中的 thread 组件是系统最基本的调度执行单元,通过时间周期或者外部事件来执行线

程。线程间的通信可以通过端口连接、子程序调用和共享数据来实现。

将ARINC653进程装换为AADL的thread组件,因为它们有相同的概念:执行主体。这些执行主体所需要的特征可以用AADL属性来定义,包括执行周期、执行截止时间、执行时间、调度策略等等。ARINC653中的进程包含在分区中就可以用AADL中的thread组件包含在process组件中来表示。

3.1.4 内存分配的转换规则

ARINC653内存指明了模块的内存总需求,同时也指明了每个分区的内存需求信息,每个内存段的空间是相互独立且互不影响的。

AADL的memory组件指明了内存分配相关信息(大小,类别等)。processor组件通过属性Actual_Memory_Binding将process组件与memory组件绑定,指明process组件内存段的分配情况。

可以将ARINC653内存直接转换为AADL的memory组件,通过给每个process组件定义子memory组件,指明了每个分区在当前模块下的空间隔离情况。

3.1.5 健康监控的转换规则

ARINC653健康监控服务用来检测和报告系统硬件和软件发生的故障,当故障发生时进行故障隔离和执行相应的恢复动作。根据故障影响的区域,将故障分为3种级别:模块、分区、进程;根据故障发生的原因,将故障分为多种类型(如,浮点数错误、堆栈溢出、硬件错误等)。而故障的恢复工作根据不同的故障级别,恢复工作也是不同的(如,模块重启、分区冷启动、分区热启动等)。

可以直接采用ARINC653附件中的AADL组件processor,process,thread的相应属性ARINC653::HM_ERRORS和ARINC653::HM_Actions。ARINC653::HM_ERRORS定义故障类型,ARINC653::HM_Actions定义恢复工作。这些属性可以包含在不同组件中来表示ARINC653故障发生的不同级别。

3.1.6 分区内通信的转换规则

ARINC653分区内的通信是指属于同一个分区的进程之间的通信。分区内通信机制一般分为缓冲区、黑板、信号量和事件。缓冲区和黑板提供一般的进程间的通信和同步,信号量和事件提供了进程间的同步。所有的分区内通信信息必须确保原子访问的,即当写入信息的时候是无法读取信息。

AADL的port(端口)是组件内的一种特征,它用于线程之间信息的交换和传输。port的类型有3种:data port(数据端口)、event port(事件端口)、event data port(事件数据端口)。data port提供了数据的传递,但是不保留上次传递的数据,相反event data port维持了一个队列,保留了上次传递的数据。Data组件提供了在多个线程或子程序间共享数据的功能。

可以将ARINC653的缓冲区概念转换为AADL的event data port,因为缓冲区的数据不会被下次通信的数据替换掉。同理,由于ARINC653的黑板不保留上次的通信数据,将ARINC653的黑板概念转换为AADL的data port。AADL中的共享data组件表示ARINC653的信号量机制,可以通过AADL属性Concurrency_Control_Protocol定义信号量的并发机制。

3.1.7 分区间通信的转换规则

ARINC653分区间通信定义了两个或两个以上分区之间的通信,分区间通信使用通道。通道的两端定义了两个端口:源端口和目的端口。根据通道上数据发送和接受的方式,端口可以分为两种类型:采样端口和队列端口;采样端口根据一定的频率发送和接受数据,数据也是不保存的。队列端口发送和接受数据是没有规律的,不过数据是保存在一个接受队列里的。

可以将ARINC653的队列端口转换为AADL的event data port,event data port可以分时接受一组数据,并保存到队列中,这与ARINC653中队列端口的特征是一样的。将ARINC653的采样端口组件转换为data port,data port没有数据队列,这与ARINC653中采样端口的特征是一样的,上次接受的数据在下个数据到达时就会被丢弃。

表2详细描述了ARINC653配置信息中核心概念转换为AADL模型元素的规则。

3.1.8 分区间通信的转换实例

图3为一个系统分区间通信配置信息表(XML数据格式)中的一部分,它包含两部分内容:分区信息;通信信息。分区信息指明了每个分区的信息(分区名,分区ID),以及分区包含的端口。分区pr2包含一个名为pdatain的采样端口,端口的刷新频率为0.025 s,端口类型是目的端口。分区pr1包含一个名为pdataout的采样端口,端口的刷新频率为0.015 s,端口类型是源端口。通信信息指明了每个通道的信息,例子中只包含一个通道。通道的源端口是分区pr1的pdataout端口,目的端口是分区pr2的pdatain端口,这样就建立了

一条数据流通信连接。图 3 配置信息转换为 AADL 模型如图 4 所示,process 组件内的 data port 端口表示分区内的采样端口,属性 ARINC653:

```
〈Partition PartitionName="pr2" PartitionIdentifier="1"〉
    〈Sampling_Port Direction="DESTINATION"
    Name="pdatain" RefreshRateSeconds="0.0250"/〉
  〈/Partition〉
  〈Partition PartitionName="pr1" PartitionIdentifier="2"〉
    〈Sampling_Port Direction="SOURCE"
    Name="pdataout" RefreshRateSeconds="0.0150"/〉
  〈/Partition〉
〈Connection_Table〉
    〈Channel ChannelIdentifier="1"〉
      〈Source〉
        〈Standard_Partition PortName="pdataout"
        PartitionName="pr1" PartitionIdentifier="2"/〉
      〈/Source〉
      〈Destination〉
        〈Standard_Partition PortName="pdatain"
        PartitionName="pr2" PartitionIdentifier="1"/〉
      〈/Destination〉
    〈/Channel〉
  〈/Connection_Table〉
```

图 3　ARINC653 分区间通信配置信息

Fig. 3　Configurations of ARINC653 partition communication

```
process partition_process_1
  features
    pdatain: in data port;
  properties
    Deployment::ASN1_Module_Name => "pr2";
    ARINC653::Sampling_Refresh_Period => "0.25";
end partition_process_1;
virtual processor implementation partition_processor_1.i
end partition_processor_1.i;
process partition_process_2
  features
    pdataout: out data port;
  properties
    Deployment::ASN1_Module_Name => "pr1";
    ARINC653::Sampling_Refresh_Period => "0.15";
end partition_process_2;
process implementation partition_process_2.i
end partition_process_2.i;
virtual processor partition_processor_2
  properties
    Deployment::ASN1_Module_Name => "pr1";
end partition_processor_2;
virtual processor implementation partition_processor_2.i
end partition_processor_2.i;
processor ARINC653_Module
end ARINC653_Module;
processor implementation ARINC653_Module.i
  subcomponents
    partition_1: virtual processor partition_processor_1.i;
    partition_2: virtual processor partition_processor_2.i;
end ARINC653_Module.i;
system ARINC653_System
end ARINC653_System;
system implementation ARINC653_System.i
  subcomponents
    partition_1: process partition_process_1.i;
    partition_2: process partition_process_2.i;
    arinc653module: processor ARINC653_Module.i;
  connections
    con1: partition_1.pdataout -> partition_2.pdatain;
  properties
    Actual_Processor_Binding => reference(arinc653module.
partition_1) applies to partition_1;
    Actual_Processor_Binding => reference(arinc653module.
partition_2) applies to partition_2;
end ARINC653_System.i;
```

图 4　通信配置信息的 AADL 文本形式模型

Fig. 4　AADL textual model of communication configurations

Sampling_Refresh_Period 指明了端口的刷新频率。系统组件实现中的 connections 部分,声明了数据端口 pdataout 到 pdatain 的数据流来表示分区间的通信。相应的图形化模型如图 5 所示。

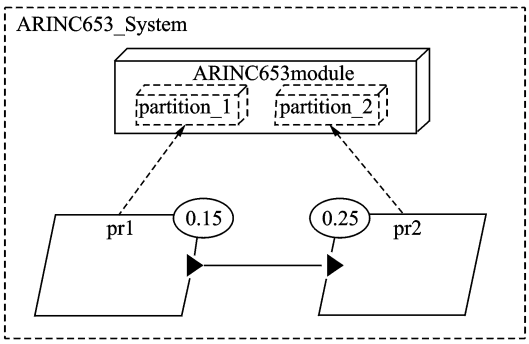

图 5　通信配置信息的 AADL 图形化模型

Fig. 5　AADL graphical model of communication configurations

3.2　转换方法

基于 3.1 节介绍的转换规则,本文给出了一个将 ARINC653 系统配置信息转换为 AADL 模型的方法。

转换方法的流程如图 6 所示,流程的起始输入是 ARINC653 配置信息文件(XML 数据格式),输出是表示 AADL 模型的模型文件。

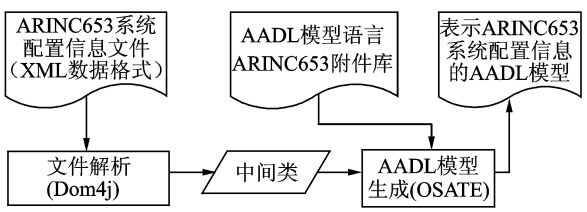

图 6　转换方法流程

Fig. 6　Process of model transformation

在这个处理过程中,转换方法采用了两个工具,一个是 Dom4j[15],主要用来读取基于 XML 数据格式的 ARINC653 配置信息文件,另一个工具是 OSATE。OSATE(Open source AADL tool environment)[16]是一个可用于 AADL 建模、编译和分析的开源工具。OSATE 根据模型的组件类以及子组件的包含关系和组件间的关联,将整个系统的所有组件类按层次组合成一个整体,OSATE 提供了模型生成接口,可以使用这个接口根据组件类生成 AADL 文本形式模型。

此外,考虑到 ARINC653 标准中对配置信息的模块化的划分,本文还设计了相应模块的类结构,也即中间类。使用中间类来储存与配置信息相

对应模块的参数信息(比如,给出整体的描述信息)。同时,它作为模型转换时的输入信息,生成AADL模型。中间类信息如表3所示,其中一部分类关系UML图如图7所示。

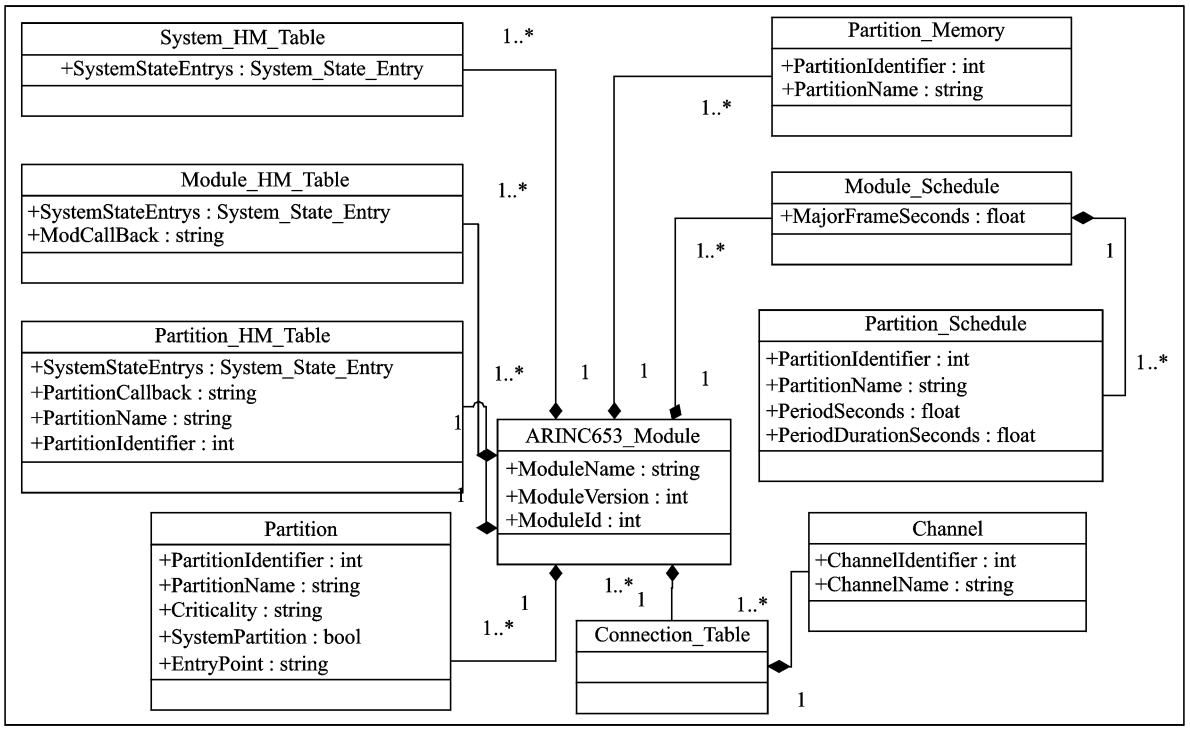

图7 一部分中间类的UML图

Fig. 7 UML diagrams of temporary class

表3 中间类概述

Tab. 3 Overview of temporary class

类名	类功能概述
ARINC653_Module	模块信息
Partition	分区信息
Process	进程信息
Module_Schedule	模块调度总时间框架
Patition_Schedule	分区调度信息
Window_Schedule	一个调度时间片信息
Patition_Memory	分区内存分配信息
Memory_Requirement	一个内存区域信息
System_HM_Table	系统健康监控表
Module_HM_Table	模块健康监控表
Patition_HM_Table	分区健康监控表
System_State_Entry	系统状态描述信息
Error_ID	故障状态描述信息
Queuing_Port	队列端口信息
Sampling_Port	采样端口信息
Connection_Table	系统通信连接表
Channel	一个通道信息
Channel_Port	通道源或目的端口信息

这个处理流程的主要步骤如下:

(1)使用Dom4j获取ARINC653系统配置信息,将获取到的参数信息保存到中间类。

(2)OSATE根据AADL模型语言ARINC653附件库,将中间类转换为AADL模型对象。转换过程是基于3.1节中所介绍的规则,转换时可以进行简单的配置信息的形式验证(比如:必要的配置信息是否已经被设置、数值类型是否正确、数值大小是否超过指定范围等),配置信息的语义验证将在下一节详细介绍。

(3)调用OSATE的模型文件生成接口,生成AADL模型文件。

4 ARINC653系统配置信息验证框架

对配置信息转换成的AADL模型进行语义验证的验证整体框架如图8所示。其中,模型转换功能就是基于上一节中的方法实现的,输入XML配置信息文件,输出转换成的AADL模型。在对所得到的AADL模型进行验证时,可以采用第三方工具Ocaina[17],这是一个可对AADL模型进行形式分析验证的软件。其中,所需验证的系统特征是基于需求执行分析语言(Requirement enforcement analysis language, REAL)[18]来进行描

述的。

REAL 是一种模型检测语言,用来分析模型架构描述之间的正确性。它是基于集合操作的语言,它允许建立集合的元素包括 AADL 实例(连接、组件或子程序调用),通过提供集合上的一阶逻辑定义和声明布尔表达式进行模型验证。使用 REAL 可以对表示重配置后的系统配置信息的 AADL 模型进行属性语义检测,来达到配置信息正确性检测的目的。

验证功能的输入分为两部分:配置信息转换成的 AADL 模型;根据配置信息的语义验证需求所设计的 REAL 定理。验证功能检测 AADL 模型中的属性语义是否满足定理描述的一阶逻辑表达式,满足输出 TRUE,否则输出 FALSE。

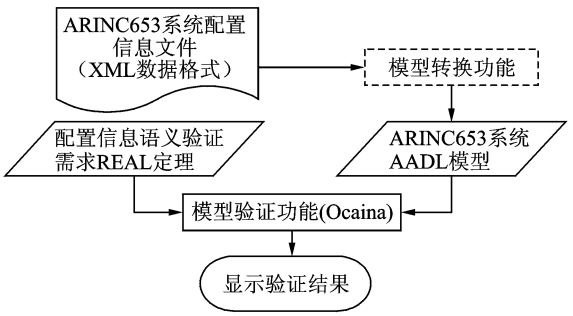

图 8　配置信息验证框架

Fig. 8　Configuration information validation framework

4.1　ARINC653 系统配置信息的语义验证需求示例

(1)时间约束。时间约束必须保证,在一个调度周期内,每个分区被调度至少一次。每个分区的时间分配总和与模块一个周期的时间是相等的。

(2)空间隔离。为每一个分区分配独立的内存段。

(3)健康监控。健康监控必须确保对于每个潜在的故障,都绑定到相关的恢复行为,每个级别的分层架构(模块、分区、过程)都有一个相关的恢复行为。

4.2　ARINC653 系统语义的 REAL 定理验证

(1)时间约束

时间约束需要验证两点约束:①一个 ARINC653 模块下的每个分区,在每一个调度周期内必须被调度一次;②分区调度时间需求的总和与模块的总时间需求必须相等。

图 9 表示了上面的第 2 点约束。AADL 属性 ARINC653::Module_Major_Frame 表示模块级调度周期,ARINC653::Partition_Slots 是一个数值列表,表示模块下每个分区的时间分配。需要检查每一个 processor 组件(ARINC653 模块)的 ARINC653::Partition_Slots 值总和是否与 ARINC653::Module_Major_Frame 值相等。

```
theorem partitions_execution
 foreach cpu in processor_set do
  vp := {x in Virtual_Processor_Set
       | Is_subcomponent_of(x, cpu)};
  check(float(property(cpu,
       "ARINC653::Module_Major_Frame"))
       =sum(property(cpu,
       "ARINC653::Partition_Slots"))
       and Cardinal(vp)>0);
end;
```

图 9　分区时间约束定理

Fig. 9　Theorem of partition time constraints

(2)空间隔离

空间隔离需要验证的约束:一个内存段只能被分配给一个分区。

图 10 描述的 REAL 定理描述了上述的约束。首先找出系统的主 memory 组件(图中变量 mainmem,ARINC653 模块级内存),再获取它的所有子 memory 组件(图中变量 partmen,ARINC653 分区级内存),最后获取每一个子 memory 组件分配给的 process 组件。验证最后得到的 process 组件集里的元素个数是否等于 1,也即验证一片内存区域分配给的 ARINC653 分区的个数必须是一个。

```
theorem memory_bound
  foreach s in System_Set do
  mainmem := {y in Memory_Set
  | Is_subcomponent_of(y, s)};
  partmem := {x in Memory_Set
  | Is_subcomponent_of(x, mainmem)};
  partitions := {x in Process_Set
  | Is_Bound_to(x, partmem)};
  check(Cardinal(partitions)=1);
end;
```

图 10　空间隔离验证定理

Fig. 10　Theorem of spatial isolation validation

(3)健康监控

健康监控需要验证的两点主要约束如下:

①当一个分区发生故障时,通过分区间的通信,它有可能影响到其他分区。为使发生故障的分区不影响其他分区,处于通信接收端的分区应处在较低关键级。

②在 ARINC653 系统架构下,故障被分为 3 个不同的级别:模块,分区,进程。ARINC653 进程发生的故障至少在这 3 个级别里中的一处被检测覆盖。

图11描述的定理描述了上述的第一点约束。首先获取系统中使用通信功能的virtual processor组件(ARINC653分区执行环境),再判断目的分区的ARINC653::Criticality(表示分区关键级)是否小于发送分区,即目的分区的关键级必须小于发送分区。

```
theorem check_partition_level
  foreach src in Process_Set do
      thr := {x in Thread_Set
          | Is_subcomponent_of(x, thr)};
      spart := {x in Virtual_Processor_Set
          | Is_Bound_to(src,x)};
      dpart := {x in Virtual_Processor_Set
          | Is_Connected_to(src,x)};
check(cardinal(src)>0 and cardinal(dst)>=0
        and (max(property(dpart,
            "ARINC653::Criticality")) <
            max(property(spart,
            "ARINC653::Criticality"))));
end;
```

图11 分区通信关键级验证定理

Fig. 11 Theorem of partition level communication validation

图12描述的定理验证了上述的第二点约束。首先获取包含待分析thread组件(ARINC653进程)的process组件(ARINC653分区),再得到与这个process组件绑定的virtual processor组件(ARINC653分区执行环境)。然后获取包含virtual processor组件的processor组件(ARINC653模块)。获取thread, virtual processor, processor这3个组件(ARINC653进程,分区,模块)的故障列表(AADL属性ARINC653::HM_ERRORS),判断是否与默认的已知故障列表(变量errors)相等,即所有的故障列表是否在系统中都有定义。

```
theorem check_error_coverage
  foreach thr in Thread_Set do
      prs := {x in Process_Set
          | Is_subcomponent_of(thr, x)};
      vp := {x in Virtual_Processor_Set
          | Is_Bound_to(prs, x)};
      cpu := {x in Processor_Set
          | Is_subcomponent_of(vp,x)};
      var errors := List("Power_Fail",
    "Module_Config","Module_Init",
"Module_Scheduling","Partition_Scheduling",
"Partition_Config","Partition_Handler",
  "Partition_Init","Deadline_Miss",
"Application_Error","Numeric_Error",
  "Illegal_Request","Stack_Overflow",
  "Memory_Violation","Hardware_Fault");
      var actual_errors := List(
      property(cpu,"ARINC653::HM_Errors")+
      property(thr,"ARINC653::HM_Errors")+
      property(vp,"ARINC653::HM_Errors"));
    check(Is_In(errors,actual_errors)
     and Is_In(actual_errors,errors));
end;
```

图12 故障覆盖验证定理

Fig. 12 Theorem of fault coverage verification

5 验证实例分析

基于前两节描述的模型转换和验证方法,现在具体的给出一个例子来说明配置信息正确性检测的过程。

表4描述了一个ARINC653模块下的分区调度信息,图13是包含表4所表示配置信息的一部分XML配置文件。当前模块的总调度周期是0.2 s,它包含5个分区。表4和图13详细地指明了每个分区的执行起始时间和执行时间,即相应时间窗口的起始时间和窗口大小。

表4 分区调度配置信息

Tab. 4 Partition scheduling configurations

Window ID	1.1	4.1	2.1	3.1	4.2	1.2
Patition	P1	P4	P2	P3	P4	P1
Window offset	0.00	0.02	0.03	0.04	0.07	0.1
Window duration	0.02	0.01	0.01	0.03	0.01	0.02
Window ID	4.3	2.2	3.2	4.4	5.1	
Patition	P4	P2	P3	P4	P5	
Window offset	0.12	0.13	0.14	0.17	0.18	
Window duration	0.01	0.01	0.03	0.01	0.02	

```
〈Module_Schedule MajorFrameSeconds="0.200"〉
  〈Partition_Schedule PartitionIdentifier="1"
                PartitionName="system management"
                PeriodSeconds="0.100"
                PeriodDurationSeconds="0.020"〉
      〈Window_Schedule WindowIdentifier="101"
                WindowStartSeconds="0.0"
                WindowDurationSeconds="0.020"
                PartitionPeriodStart="true"/〉
      〈Window_Schedule WindowIdentifier="102"
                WindowStartSeconds="0.1"
                WindowDurationSeconds="0.020"
                PartitionPeriodStart="true"/〉
  〈/Partition_Schedule〉
  ……
  〈Partition_Schedule PartitionIdentifier="5"
                PartitionName="IHVM"
                PeriodSeconds="0.200"
                PeriodDurationSeconds="0.020"〉
      〈Window_Schedule WindowIdentifier="501"
                WindowStartSeconds="0.180"
                WindowDurationSeconds="0.020"
                PartitionPeriodStart="true"/〉
  〈/Partition_Schedule〉
〈/Module_Schedule〉
```

图13 XML格式分区调度配置信息

Fig. 13 XML partition scheduling configurations

由于例子包含的是时间调度方面的信息,需要对配置信息进行时间约束验证,验证步骤如下:

(1)根据本文介绍的模型转换方法,将图13描述的ARINC653系统配置信息转换为AADL模型。图14描述了转换成的AADL模型的一部分。分区P1,P2,P3,P4,P5分别转换成virtual proces-

sor 组件 partition_1,partition_2,partition_3,partition_4,partition_5,分区时间片的分配使用属性 ARINC653::Slots_Allocation 和 ARINC653::Partiton_Slots 表示,前者指明分区占用系统时钟的顺序,后者指明顺序时间片的大小。针对系统时钟空闲的情况,即没有任何一个分区占用系统时钟。先另创建一个 virtual processor 组件(partition_idle),再将相应的空闲时间片的大小赋予这个组件。

```
processor implementation ARINC653_Module.i
  subcomponents
   partition_idle: virtual processor partition_processor_0.i;
    partition_1: virtual processor partition_processor_1.i;
    partition_2: virtual processor partition_processor_2.i;
    partition_3: virtual processor partition_processor_3.i;
    partition_4: virtual processor partition_processor_4.i;
    partition_5: virtual processor partition_processor_5.i;
  properties
    ARINC653::Module_Major_Frame => 0.2s;
    ARINC653::Slots_Allocation => (reference (partition_1),
     reference (partition_4),reference (partition_2),
     reference (partition_3),reference (partition_4),
     reference (partition_idle),reference (partition_1),
     reference (partition_4),reference (partition_2),
     reference (partition_3),reference (partition_4),
     reference (partition_5));
    ARINC653::Partition_Slots => (0.02s,0.01s,0.01s,0.
03s,
          0.01s,0.02s,0.02s,0.01s,0.01s,0.03s0.01s,0.
02s);
end ARINC653_Module.i;
system ARINC653_System
end ARINC653_System;
system implementation ARINC653_System.i
  subcomponents
    partition_1: process partition_process_1.i;
    partition_2: process partition_process_2.i;
    partition_3: process partition_process_1.i;
    partition_4: process partition_process_2.i;
    partition_5: process partition_process_1.i;
  properties
    Actual_Processor_Binding =>
reference(arinc653module.partition_1) applies to partition_1;
    Actual_Processor_Binding =>
reference(arinc653module.partition_2) applies to partition_2;
  Actual_Processor_Binding =>
reference(arinc653module.partition_3) applies to partition_3;
    Actual_Processor_Binding =>
reference(arinc653module.partition_4) applies to partition_4;
  Actual_Processor_Binding =>
reference(arinc653module.partition_5) applies to partition_5;
end ARINC653_System.i;
```

图 14　分区调度的 AADL 模型

Fig. 14　AADL model of partition scheduling

(2)根据要验证的需求约束,编写相应的 REAL 定理。由于表 4 表示的配置信息包含的是时间需求方面的信息,需要验证系统分区时间约束。这里使用 4.2 节中图 9 描述的定理,验证模块总时间片与分区的时间片的总和是否相等,并将上述编写的验证定理保存成以 real 为扩展名的文件。

(3)根据第 1 步转换生成的 AADL 模型和第 2 步生成的 REAL 定理,使用 Ocaina 工具进行模型验证,输出验证结果如图 15 所示。arinc653.real 是 REAL 定理所在文件的名字,ARINC653_System 是转换生成的 AADL 模型文件的名字,ARINC653_System.i 是图 14 描述的 AADL 模型中的 system 组件实例名。通过验证 system 组件实例下的 processor 组件属性来达到对系统配置信息进行时间约束验证的目的。

```
bin\ocarina.exe-real_continue_eval-real_lib arinc653.real-
aadlv2-g real_theorem-r ARINC653_System.i ARINC653_
System.aadl
resources execution
requirement:partitions_execution
theorem partitions_execution is: TRUE
theorem resources is: TRUE
```

图 15　验证结果

Fig. 15　Validation results

由于表 4 包含的配置信息是满足时间需求约束的,即一个模块调度周期内,分区时间片的总和是小于等于模块总时间片,验证结果返回 TRUE(图 15 中 theorem resources is: TURE,表明定理都验证通过)。

当对配置信息进行重配置时,比如当前模块需要添加一个新分区 P6,对 P6 进行调度配置,重配后的信息如表 5 所示,添加了对分区 P6 的时间片的分配。对于重配置后的配置文件,需要再次进行时间约束验证。由于表 5 包含的配置信息不满足时间需求约束,即一个模块调度周期内,分区时间片的总和是大于模块总时间片(window duration 一行所有数据的和大于 0.2 s),模型验证功能将返回 FALSE。

表 5　重配置后的分区调度配置信息

Tab. 5　Reconfiguration information of partition scheduling

Window ID	1.1	4.1	2.1	3.1	4.2	1.2
Patition	P1	P4	P2	P3	P4	P1
Window offset	0.00	0.02	0.03	0.04	0.07	0.1
Window duration	0.02	0.01	0.01	0.03	0.01	0.02
Window ID	4.4	2.2	3.2	4.5	5.1	4.3
Patition	P4	P2	P3	P4	P5	P6
Window offset	0.12	0.13	0.14	0.17	0.18	0.08
Window duration	0.01	0.01	0.03	0.01	0.02	0.03

通过本文介绍的配置信息正确性检测方法,可以检查配置信息进行重配置后(例子中添加新分区的时间调度信息)可能存在的错误,避免新系统运行时造成的灾难性后果,确保系统的可靠和安全性。

6　结束语

本文提出了一种对 ARINC653 系统进行配置

信息验证方法，方法的核心是模型转换及模型验证。选用AADL作为模型语言是因为它的语义很适合于分区实时关键系统的概念。首先，根据扩展的ARINC653附件库，将配置信息转换为AADL模型；其次，根据配置信息的语义验证需求，设计相应的REAL定理。最后基于REAL定理调用，对生成的模型文件进行需求约束验证。通过模型验证方法找出配置信息中的错误，及时调整相应模块的配置，提高系统的安全性和可靠性。

随着对ARINC653系统配置信息和AADL的深入研究，下一步的工作主要集中在以下几个方面：

(1)在进行模型转换的过程中，本文介绍了配置信息核心概念转换为AADL模型元素的规则。对于次要概念如何转换为AADL模型元素，需要进一步的研究。

(2)本文提出的配置信息正确性检测方法可以进行一些语义验证，但是这些检测是不全面的，对比如任务时间可达性方面的验证无能为力。所以，需要进一步研究系统配置信息非功能属性的正确性检测问题。

(3)深入研究配置信息中分区调度、分区通信、分区抢占等难点问题，进一步研究分区级可调度性判定等问题。

参考文献：

[1] 易建平，韩庆. 飞机综合模块化航电系统总体设计研究[J]. 科学技术与工程，2010，10(19)：4709-4714.
Yi Jianpin, Han Qin. Design research of aircraft Integrated modular avionics system[J]. Science Technology and Engineering, 2010,10(19):4709-4714.

[2] Aeronautical Radio Inc. ARINC specification 653 avionics application software standard interface[S]. Annapolis: Aeronautical Radio Inc, 1997.

[3] Feiler P H, Gluch D P, Hudak J J. The architecture analysis & design language (AADL): An introduction[R]. Technical Report, Software Engineering Institute of Carnegie Mellon University. Huntsville: SEI AADL Team, 2006:45-68.

[4] Delange J, Hugues J, Pautet L, et al. Code generation strategies from aadl architectural descriptions targeting the high integrity domain[C]//Proceedings of the 4th European Congress ERTS. Toulouse: SafeTRANS, 2008:23-46.

[5] SAE Aerospace. Architecture analysis and design language[R]. AS5506. Warrendale: SAE Corporate Communications, 2008:35-146.

[6] Jean-Paul B, Raphaël C, David C, et al. A mapping from AADL to Java-RTSJ[C]//Proceedings of the 5th International Workshop on Java Technologies for Real-time and Embedded Systems. Oslo, Norway: ACM, 2007:165-174.

[7] Zalila B, Hugues J, Pautet L. Ocarina user guide[R]. Paris: TELECOM Pairs Tech, 2005:67-79.

[8] 徐建华. 基于AADL的ARINC653配置工具的研究与实现[D]. 成都：西南交通大学，2011.
Xu Jianhua. Research and implementation of ARINC653 configuration Tool Based on AADL[D]. Chengdu: Southwest Jiaotong University, 2011.

[9] 张慧，经小川，谢伟华. 基于Petri网的AADL模型正确性验证研究[J]. 计算机技术与发展，2012，22(9)：91-94.
Zhang Hui, Jing Xiaochuan, Xie Weihua. Verification research on correctness of AADL model based on petri net[J]. Computer Technology and Development, 2012,22(9):91-94.

[10] Jan P, Bernd K. System testing in avionics domain[C]//Proceedings of the Digital Avionics Systems Conference. Dallas, USA: IEEE, 2006:18-29.

[11] Committee A E E. Avionics application software standard interface[M]. Washington: Aeronautical Radio, 1997:20-30.

[12] SAE Aerospace. Architecture analysis and design language (AADL) annex volume 1[R]. Warrendale: SAE Corporate Communications, 2006:67-93.

[13] Lasnier G, Wrage L, Pautet L, et al. An implementation of the behavior annex in the AADL tool-set osate2[C]//Proceedings of the 6th IEEE International Workshop UML and AADL. Las Vegas, USA: IEEE, 2011:134-149.

[14] Delange J, Pautet L. ARINC653 annex overview[R]. Paris: TELECOM Pairs Tech, 2008:79-92.

[15] 李纲，王晓东，岑雄鹰. XML文档分解技术及文档存取模型[J]. 计算机应用研究，2001，18(3)：127-130.
Li Gang, Wang Xiaodong, Cen Xiongying. The XML document decomposition technique and access model[J]. Application Research of Computers, 2001, 18(3):127-130.

[16] SEI AADL Team. Open source AADL tool environment[R]. Technical Report, Software Engineering Institute of Carnegie Mellon University. Huntsville: SEI AADL Team, 2006:12-34.

[17] Zalila B, Pautet L, Hugues J. Towards automatic middleware generation[C]//Proceedings of the Object Oriented Real-Time Distributed Computing (ISORC), 11th IEEE International Symposium. Miami, USA: IEEE, 2008:221-228.

[18] Gilles O, Hugues J. Validating requirements at model-level[C]//Proceedings of the 4th Workshop on Model-Oriented Engineering. Grenoble, France: ACM, 2008:211-218.