ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.11, November 2010, pp.2738−2751
doi: 10.3724/SP.J.1001.2010.03701

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于形式化规约生成软件体系结构模型的方法*

祝 义[1,2+], 黄志球[1], 曹子宁[1], 周 航[1], 刘亚萍[1]

[1](南京航空航天大学 信息科学与技术学院,江苏 南京 210016)

[2](徐州师范大学 计算机科学与技术学院,江苏 徐州 221116)

Method for Generating Software Architecture Models from Formal Specifications

ZHU Yi[1,2+], HUANG Zhi-Qiu[1], CAO Zi-Ning[1], ZHOU Hang[1], LIU Ya-Ping[1]

[1](College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

[2](School of Computer Science and Technology, Xuzhou Normal University, Xuzhou 221116, China)

+ Corresponding author: E-mail: zhuyi@nuaa.edu.cn, http://www.nuaa.edu.cn

Zhu Y, Huang ZQ, Cao ZN, Zhou H, Liu YP. Method for generating software architecture models from formal specifications. *Journal of Software*, 2010,21(11):2738−2751. http://www.jos.org.cn/1000-9825/3701.htm

Abstract: This paper uses LOTOS to describe the requirement specification of real-time systems and proposes a method for generating software model from formal specifications by establishing a mechanism that translates LOTOS specifications into UML-RT models. Finally, this paper illustrates how to use the method when modeling real-time software. The UML-RT models generated by this method can increase the reliability for designing the software for real-time systems.

Key words: formal specification; software architecture; LOTOS; UML-RT; real-time system

摘 要: 使用 LOTOS 描述实时系统需求规约,通过建立 LOTOS 规约到 UML-RT 模型的模型转换,提出一种基于形式化规约生成软件体系结构模型的方法.最后,通过一个实例来说明如何将该方法应用于实时软件建模.利用这种方法建立的 UML-RT 模型,能够从整体上提高实时系统软件体系结构设计的可信性.

关键词: 形式化规约;软件体系结构;LOTOS;UML-RT;实时系统

中图法分类号: TP311 **文献标识码**: A

形式化方法源于Dijkstra和Hoare的程序验证,其主要优点是具有精确性,可以验证,并且便于机器支撑和自动处理等.这些特点对克服目前软件生产中软件的可靠性差、难以实现自动化的困境具有明显的作用.进程代数是一种用来解决并发系统通信问题的形式化方法,可以描述和分析并发、异步、非确定等系统行为,具有良好的语义与可扩展性,使得它非常适合于实时软件的建模与分析.经典进程代数如 CSP[1],CCS[2],LOTOS[3]. LOTOS(language of temporal ordering specification)是国际化标准组织 ISO(Internationl Standards Organization)于1989年发布的一个针对形式描述技术(formal descriptional techniques)的国际标准(标准号:ISO 8807).LOTOS

* Supported by the National Natural Science Foundation of China under Grant No.60873025 (国家自然科学基金); the Jiangsu Provincial Natural Science Foundation of China under Grant No.BK2008389 (江苏省自然科学基金); the Natural Science Foundation for Colleges and Universities in Jiangsu Province of China under Grant No.08KJB520010 (江苏省高校自然科学基金)

Received 2008-12-20; Revised 2009-04-10; Accepted 2009-07-23

中关于行为方面的概念是基于进程代数方法定义的,因此,LOTOS 在并发行为的描述方面集成了 CCS 与 CSP 的优点.此外,LOTOS 还引入了 ACT-ONE 中的方法来定义数据结构和值表达式.

UML-RT(unified modeling language for realtime)[4]是 UML 在实时方面的扩展,主要用于复杂实时系统软件体系结构(software architecture,简称 SA)的建模与分析,描述实时软件系统的结构模型和行为模型.与标准的 UML 相比,UML-RT 利用 UML 的扩展机制引入了 ROOM(real-time object oriented modeling language)[5]的语义和 ObjecTime 的基于角色建模方法,使得它能够在 SA 层描述实时系统的行为.

UML-RT 虽然引入了 ROOM 语义,但是它仍旧属于半形式化建模语言范畴,难以对系统指定的关键性质进行检验.研究人员一般通过建立 UML-RT 到形式化方法的转换机制将 UML-RT 模型转换成形式化模型,例如将 UML-RT 模型转换成进程代数[6]、状态机[7]等,然后在形式化模型中进行模型检验.但是,这类方法不能从源头上解决 UML-RT 模型本身具有的问题.因为 UML-RT 模型设计中需求规约一般直接采用自然语言,相对于形式化规约而言,自然语言描述的需求往往被认为是不够精确的,存在二义性.然而,如果我们将自然语言描述的需求规约转化为形式化规约,然后将形式化规约转换为保留形式化语义的 UML-RT 模型,就可以从根本上解决 UML-RT 模型的问题.将这种通过形式化规约生成的 UML-RT 模型作为实时系统的 SA 模型,能够从本质上提高 SA 设计的可信性.自然语言规约到 SA 模型的转换框架如图 1 所示.

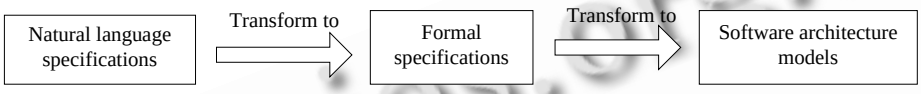


Fig.1 Transformation framework from natural language specifications to SA model

图 1 自然语言规约到 SA 模型的转换框架

自然语言规约到形式化规约的转换目前已有大量的相关研究工作.文献[8]将自然语言描述的规约按照规约类型分类,提出一种将自然语言规约转换为形式化规约的方法.文献[9,10]提出不同的精化系统规约的方法,将非形式化规约转换为形式化规约.文献[11]使用 LOTOS 提出一种增量式规约产生的方法,主要用于描述复杂系统的需求规约.基于以上研究工作,本文假设自然语言规约到形式化规约的转换已经完成,因此后文主要讨论形式化规约到 SA 模型的转换.

MDE(model-driven engineering)[12]是软件工程领域新兴的一种软件开发模式.它以模型为首要软件制品,通过元建模(metamodeling)和模型转换来驱动软件的开发,能够较好地解决异构平台间的相互转换问题.随着 MDA(model driven architecture)[13]的提出,MDE 逐渐受到了学术界和工业界的广泛关注,产生了大量的元元模型体系、建模方法和相应的支撑工具.其中具有代表性的是 OMG(object management group)推动的 MDA 框架标准和 ATLAS 研究组提出的 AMMA(ATLAS(Atlantic data systems) model management architecture)[14]开发平台.在 MDE 中,建模过程实际上就是使用建模语言构造模型的过程.相对于具体模型而言,建模语言就是其元模型.反过来看,具体模型就是该元模型的实例模型.构建元模型的过程称为元建模,这需要相应的元建模语言和机制来支持.目前,在 MDE 中有多种元建模体系,即元-元模型体系.常见的元-元建模体系多分为 3~4 层,如 MOF(meta object facility)[15],EMF(eclipse modeling framework)[16],KM3(kernel metameta model)[17]等.

本文采用 MDE 开发模式相应的支撑平台 AMMA 所提供的技术,通过元建模的方法实现 LOTOS 规约到 UML-RT 模型的转换.首先,基于 AMMA 平台的 KM3 元模型体系,通过元建模抽象出 LOTOS 与 UML-RT 的 KM3 元模型,实现 LOTOS 与 UML-RT 的同构化;然后,利用平台的模型转换语言 ATL(ATLAS transformation language)[18],针对 LOTOS 元模型和 UML-RT 元模型构造转换规则.通过将对应的实例模型进行相互转换,实现在 MDE 下 LOTOS 规约到 UML-RT 模型的转换.

本文第 1 节简要介绍 LOTOS 规约和 UML-RT 模型的元建模过程.第 2 节详细描述 LOTOS 规约到 UML-RT 模型的转换.第 3 节以一个实时系统设计为例,讨论该方法的使用效果.第 4 节进行相关工作比较.第 5 节总结全文并给出结论.

1 LOTOS 规约和 UML-RT 模型的元建模

1.1 LOTOS规约

在 LOTOS 中,系统 S 可以通过进程(process)进行定义,进程之间可以通过组合形成更加复杂的进程.一个进程 $P[G]$有一组可以被观察到的门(gate)$G=\{g_1,g_2,\ldots,g_n\}$.动作(action)是 LOTOS 中最基本行为,复杂的行为是通过操作符(operator)将动作进行组合表示的.动作可以分为两大类:一类是通过门可以由环境观察到的;另一类是进程内部的行为,对于环境是不可见的.动作的可见性是可以改变的.这样,可以在对系统建模时,将关注点集中在系统某些方面的特征上.LOTOS 的行为表达见表 1.

Table 1 Syntax of behaviour expression in basic LOTOS

表 1 LOTOS 行为表达式的语法

NAME	SYNTAX
Inaction	**stop**
Action prefix	
-Unobservable	***i***; *B*
-Observable	*g*; *B*
Choice	*B*1[]*B*2
Parallel composition	
-General case	*B*1|[$g_1,\ldots,g_n$]|*B*2
-Pure interleaving	*B*1|||*B*2
-Full synchronization	*B*1||*B*2
Hiding	**hide** $g_1,\ldots,g_n$ **in** *B*
Process instantiation	*p*[$g_1,\ldots,g_n$]
Success termination	**exit**
Sequential composition	*B*1>>*B*2
Disabling	*B*1[>*B*2

典型的 LOTOS 规约与进程的语法定义如下:

specification:

specification typical_spec [***gate list***] (***parameter list***): ***functionality***

type definitions

behaviour

behaviour expression

where

type definitions

process definitions

endspec

process definition:

process typical_proc [***gate list***] (***parameter list***): ***functionality***:=

behaviour expression

where

type definitions

process definitions

endspec

图 2 给出了简化的 LOTOS 元模型.我们没有严格按照 LOTOS 基本概念对应的名称直接定义元模型,而是对 LOTOS 的概念重新进行分析,根据 MDA 中元模型的一些特点进行 LOTOS 元建模.例如,通过声明 Declaration 元模型作为声明性概念的根,可以方便地将这些概念统一起来,为后面的模型转换提供方便[19].

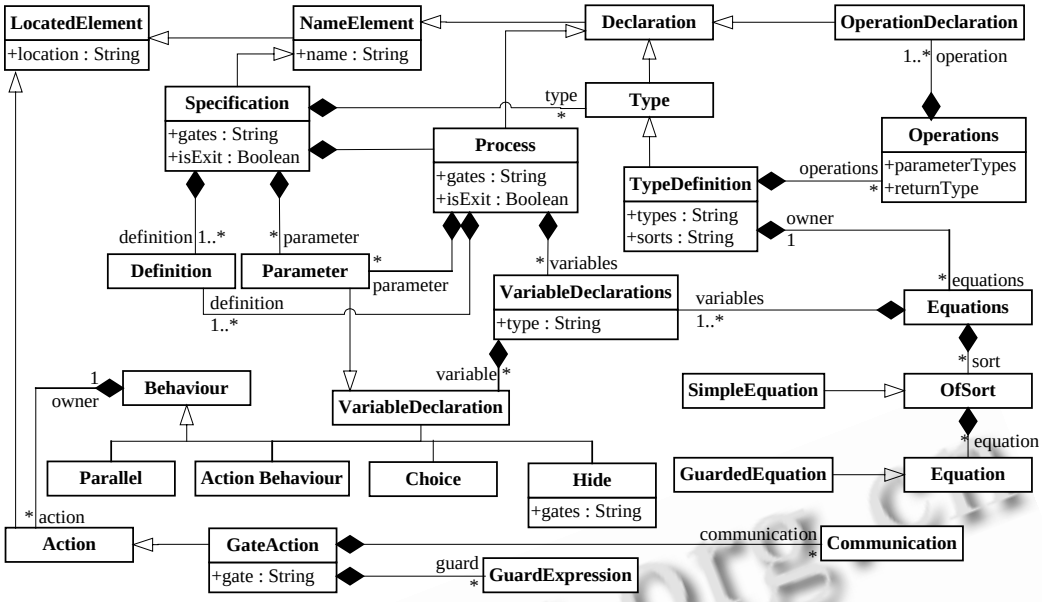


Fig.2 Meta-Model of LOTOS

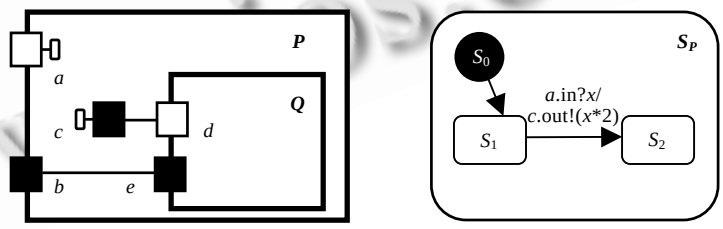
图 2 LOTOS 元模型

1.2 UML-RT模型

相对于标准的 UML 而言,UML-RT 中增加了封装体(capsule)、端口(port)、协议(protocol)、连接子(connector)等 4 类建模元素,使得它能够在 SA 层描述系统的行为.UML-RT 作为一种工业界实时系统分析与建模的工具也一直受到学术界的广泛关注[6,7],特别是它将构件化的软件设计思想融入到其建模结构中,使得它非常适合于描述实时系统的软件体系结构.

UML-RT 在 UML 元模型中加入 4 类新的元素:

封装体.封装体是 UML-RT 中一种非常重要的结构,用来表示复杂实时系统中的主要体系结构元素.封装体具有以下几个特点:① 一个封装体具有 1 个或多个端口,封装体利用这些端口和其他封装体进行通信.端口是封装体与外界进行通信的唯一手段.② 封装体可能包含一个或多个子封装体,它们之间通过连接子连接,封装体的内部结构通常可用一个协作图来描述(如图 3 所示).③ 封装体可以具有一个通过其终止端口接收和发送信号的状态机,状态机用于描述封装体的动态行为.状态机处理来自端口的信号并将处理结果通过端口发送出去.④ 封装体可以动态的使用封装体角色(capsule role),封装体角色位于封装体内部,并且不能独立于封装体而存在.缺省情况下,封装体角色随着包含它的封装体自动产生,当封装体撤销时撤销.



(a) (b)

Fig.3 Structure and state machine diagrams of a capsule

图 3 封装体结构图与状态机图

端口.端口是封装体实例的边界对象,它们是封装体的一部分.即端口在其相应的封装体被创建时创建,在其相应的封装体被撤销时撤销.从封装体的外部看,所有端口都是相应对象的接口,只能通过端口标识和它在相应协议中所充当的角色加以区分.从封装体内部来看,端口可以分为中继端口(relay port)和终止端口(end port)两类.它们的区别在于:中继端口与它所属封装体的子封装体相连,终止端口与它所属的状态机相连(如图 3(a)中,*b*,*d*,*e* 端口是中继端口,*a*,*c* 端口是终止端口).

协议.协议用于定义封装体相连端口之间有效的信息流,用协议角色(protocol role)定义端口的类型,即指明特定端口要实现相应协议角色所定义的行为.

连接子.连接子用于连接端口,它们是基于信号的通信通道的抽象视图.对于两个协议角色而言,如果其中一个协议角色的每一个输入信号都是另一个协议角色的输出信号、每一个输出信号都是另一协议角色的输入信号,那么称这两个协议角色是互补的.

UML-RT 提供状态机来建模封装体的内部行为.状态机是一个通过迁移连接的有向图,除了初始迁移自动执行以外,状态机中其他迁移需要被状态机端口的消息触发后才能执行.封装体状态机中没有结束状态,因为封装体是一个不会终止的活动类.尽管封装体结构图中可能有多个状态机图标,但是每个封装体最多具有一个状态机.封装体的多个终止端口可以和同一个状态机图标相连,因此在封装体结构图中通常会出现重复的状态机图标,它们表示同一个状态机.

在进行UML-RT元建模之前,必须先建立UML-RT构造类与UML 2元类之间的映射,因为在UML的profile定义中,构造型必须通过扩展(extension)关系应用到某些元类型(metaclass)上,表示该构造型在实例化时将会作为附加特性绑定到某个指定元模型的实例上.图 4 给出了 UML-RT 的构造型.

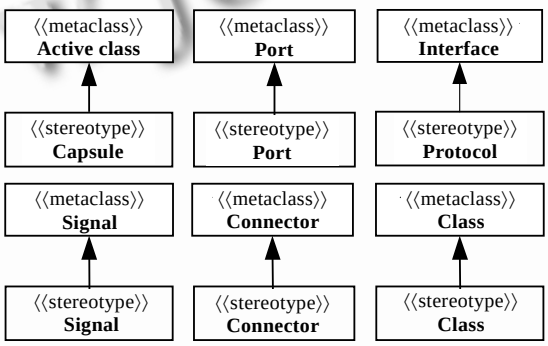


Fig.4 Stereotypes in UML-RT

图 4 UML-RT 的构造型

UML-RT Capsule-UML 2 Active Class.在 UML-RT 中,封装体是实时系统的基本建模元素.和 UML 1.x 中的类一样,封装体具有属性和操作,并且也可以参与依赖、继承和关联的关系.与类不同,封装体能够表示系统的控制流.封装体与类的不同之处包括:① 可以拥有公共端口;② 能够通过嵌套来说明自身的内部组织和行为;③ 消息传递代替方法调用;④ 状态机代替操作用于定义行为.与 UML 1.x 不同,UML 2 类更有表达力,它可以拥有端口作为活动点来发送与接收信号,允许嵌套并且能够通过状态机描述自身的内部行为.UML 2 活动类是针对每个类实例带有不同控制流的类,因此我们自然想到将 UML-RT 的封装体映射为 UML 2 的活动类.由于 UML 2 缺省情况下端口是公共的,为了精确地将封装体语义映射到 UML 2 上,UML 2 活动类仅需声明:类的属性和操作是私有的,这使得通信仅仅指的是公共端口和信号.

UML-RT Port-UML 2 Port.UML-RT 端口是封装体之间通信的基本分类器.端口是发送消息到封装体实例以及从封装体实例接收消息的对象.一个端口通常对应一个协议角色.协议角色定义了端口类型.为了让两个端口能够通过连接子相连,端口必须是兼容的,一个协议角色 Out 集的每个信号必须在其他协议角色的 In 集里面.UML 2 引入了 UML 1.x 中没有的端口分类器,它的语义和 UML-RT 中的端口非常相似,即端口表示类或者构

件与环境之间的交互点.通过连接子连接的两个端口主要不同在于交互的定义上.为了说明端口能够发送或者接收的消息,在 UML 2 中端口关联于接口,而 UML-RT 中端口关联于协议角色.

UML-RT Protocol-UML 2 Interface.UML-RT 协议是合同,它定义了协议中参与者之间能够被交换消息的类型.因此,协议由不同的被称为协议角色的参与者组成,这些参与者被称为协议角色,协议角色通过发送和接收的消息定义.协议主要被用于区分端口的类型,端口扮演通信关系中参与者的角色.我们把 UML 2 的接口映射到 UML-RT 协议上,一个接口定义一个合同,它表示一组前后一致公共特征集合和义务的声明,并且它不能被实例化.

UML-RT Signal-UML 2 Signal.信号分类器没有明显的变化,它在 UML-RT 与 UML 2 中具有相同的语义.当某个对象执行的发送行为产生一个信号消息并被另外一个对象接收时,一个信号事件就产生了.信号事件可能源于行为或者某个操作的执行,表示一个指定异步消息的接收.信号事件的接收通常触发目标对象状态机的状态变迁.

UML-RT Connector-UML 2 Connector.UML-RT 中连接子是在协作的封装体之间为了通信允许互联的元素,互联建立在兼容的封装体端口上.在 UML 2 中,它说明了在两个或者更多连接元素实例之间允许通信的连接.因此,连接子除了能够被使用的抽象级别以外,语义类似.在 UML-RT 中,连接子指的是实现级别,而 UML 2 指的是实现级别或者更多的抽象级别.

UML-RT Class–UML 2 Class.UML-RT 类描述了设计时一个或者更多具有相同结构、属性、操作和行为的不同对象,在运行时表现为实现一个共同目标而协作的类的多个实例.UML 2 中类进行了几个重要的改变,它保留了 UML 1.x 的属性并且丰富几个其他属性.事实上,类表示拥有共同属性、限制和语义的对象集合.此外,增加了新的属性,如类的嵌套以及拥有端口.

基于 UML-RT 的构造型可以进一步给出简化的 UML-RT 元模型(如图 5 所示).

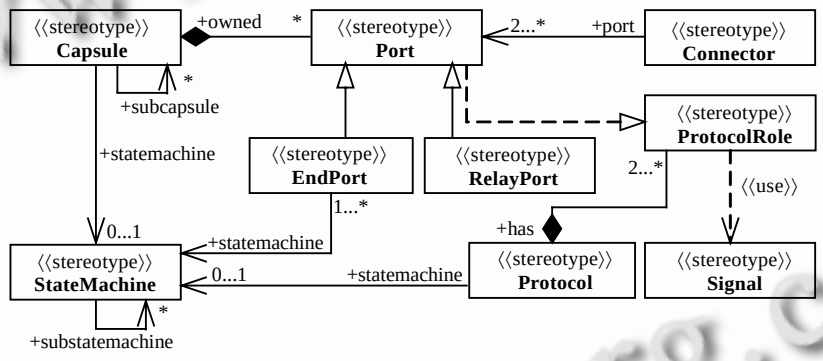


Fig.5 Meta-Model of UML-RT

图 5 UML-RT 元模型

2 LOTOS 规约到 UML-RT 模型的转换

本节主要讨论 LOTOS 规约到 UML-RT 模型的转换,我们建立一组 LOTOS 到 UML-RT 语义映射规则,利用语义映射规则可将 LOTOS 规约转换成 UML-RT 模型.

2.1 LOTOS规约到UML-RT模型的转换策略

(1) LOTOS 的类型转换为 UML-RT 的类型

LOTOS 基本类型主要有 InterType,RealType,BooleanType 和 StringType.我们可以将 InterType,RealType,BooleanType 和 StringType 直接映射到 UML-RT 的 Signal 的 Interger,Real,Boolean 和 String.

(2) LOTOS 的 Specification 转换为 UML-RT 的 Capsule 与 StateMachine

LOTOS 的 Specification 是 LOTOS 规约中最主要的实体单元,因此我们将 Specifications 作为转换的顶层部分,并把它作为转换开始的入口.UML-RT 的 Capsule 用来表示复杂实时系统中的主要体系结构元素,Capsule 是整个系统的顶层模型,所以,LOTOS 的 Specification 可以转换为 UML-RT 的 Capsule.

LOTOS 的 Specification 定义了 behaviour expression,behaviour expression 用于描述 Specifications 的行为.UML-RT 的 Capsule 拥有一个通过其 end ports 接收和发送信号的 StateMachine,StateMachine 用于描述 Capsule 的动态行为.因此,LOTOS 的 Specification 的 behaviour expression 转换为 UML-RT 的 StateMachine.

(3) LOTOS 的 Process 转换为 UML-RT 的 SubCapsule 与 SubStateMachine

LOTOS 的 Specification 可以包含 1 个或多个 Process.Specification 与 Process 的区别在于:① Specification 的 behaviour expression 在关键词 behaviour 的前面,而 Process 通过“:=”定义 behaviour expression;② type definifitions 出现在 Specification 的 behaviour expression 的前面,而 Process 只允许出现在 behaviour expression 的后面.Specification 的 type definition 是全局定义,因此它能够在 Specification 的 parameter list 中查到.通过比较可知,Process 与 Specification 除了语法表达不同之外,它们的元模型是基本相同的.因此,Process 可以和 Specification 进行相同的转化.UML-RT 的 Capsule 可以包含多个 SubCapsule,因此,LOTOS 的 Process 可以转换为 UML-RT 的 SubCapsule.

LOTOS 的 Process 和 Specification 一样也定义了 behaviour expression,因此,LOTOS 的 Process 的 behaviour expression 转换为 UML-RT 的 SubStateMachine.

(4) LOTOS 的 ActionPrefix 转换为 UML-RT 的 Transition

LOTOS 的 ActionPrefix 产生一个新的 behaviour expression.ActionPrefix 可以映射为 StateMachine 中的 Transition,因此,LOTOS 的 ActionPrefix 转换为 UML-RT 的 StateMachine 的 Transition.

(5) LOTOS 的 Gate 转换为 UML-RT 的 Port

LOTOS 的 Gate 是 Process 与外界进行交互的接口,这与 UML-RT 的 Port 具有同样的作用.因此,LOTOS 的 Gate 转换为 UML-RT 的 Port.

(6) LOTOS 的 Communication 转换为 UML-RT 的 Protocol 和 Connector

LOTOS 的 Communication 用于描述进程之间的多种通信行为,既描述了行为规约(communication protocol),又给出了通信进程之间的结构信息(communication channel).UML-RT 的 Protocol 描述了与 Capsule 相连的端口之间有效信息流(signal),但它只指出行为上的约束而没有任何结构元素.而连接子则是实实在在的物理对象,其功能仅仅是将信号从一个端口传送到另一个端口.因此,LOTOS 的 Communication 转换为 UML-RT 的 Protocol 和 Connector.

2.2 ATL转换规则

ATL 模型转换规则是针对源模型和目标模型的元模型进行定义的,通过 ATL 虚拟机的执行将源模型的实例模型转换为目标模型的实例模型.因此,构建 LOTOS 到 UML-RT 的转换就是针对 LOTOS 和 UML-RT 的 KM3 元模型定义具体的 ATL 规则.图 6 给出了在 ATL 下 LOTOS 到 UML-RT 转换规则定义和应用的视图.

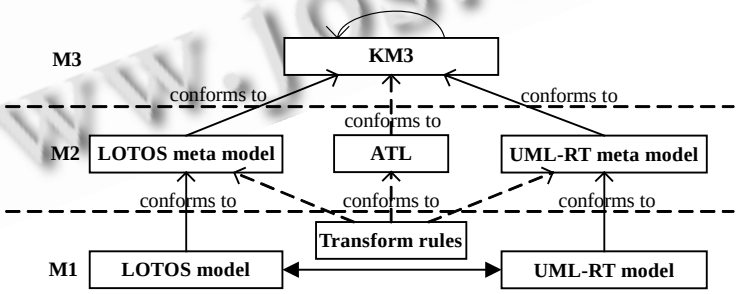


Fig.6 ATL model transformation

图 6 ATL 模型转换视图

根据第 2.1 节的转换策略,我们定义了 LOTOS 到 UML-RT 转换的 ATL 规则,见表 2.

Table 2 Rules in LOTOS2UML-RT.atl

表 2 LOTOS2UML-RT.atl 中的规则

Name	Description
IntegerType2Integer	Match LOTOS.IntegerType to UML-RT.Integer
RealType2Real	Match LOTOS.RealType to UML-RT.Real
BooleanType2Boolean	Match LOTOS.BooleanType to UML-RT.Boolean
StringType2String	Match LOTOS.StringType to UML-RT.String
Specification2Capsule	Match LOTOS.Specification to UML-RT.Capsule
Specification2StateMachine	Match LOTOS.Specification to UML-RT.StateMachine
Process2SubCapsule	Match LOTOS.Process to UML-RT.SubCapsule
Process2SubStateMachine	Match LOTOS.Process to UML-RT.SubStateMachine
ActionPrefix2Transition	Match LOTOS.ActionPrefix to UML-RT.Transition
Gate2Port	Match LOTOS.Gate to UML-RT.Port
Communication2Protocol	Match LOTOS.Communication to UML-RT.Protocol
Communication2Connector	Match LOTOS.Communication to UML-RT.Connector

ATL 规则具体内容如下:

```
1.  module LOTOS2UML_RT; --Module Template
2.  create OUT: UML_RT from IN: LOTOS;
3.  rule Specification2UML_RT {
4.          from s: LOTOS!Specification
5.          to t: UML_RT!Capsule (
6.                  name←t.name+'_Capsule',
7.                  owned←Set{p}
8.          ),
9.          p: distinct UML_RT!Port foreach (e in s.gates.split(','))(
10.         self←thisModule.getPort(e)
11.         )
12.         do {
13.         t.subcapsule←s.processes;
14.         }
15.         }
16.  entrypoint rule getPort(n:String){
17.         to t: UML_RT!Port
18.         do {
19.                 t.name←n+'_Port';
20.                 if (thisModule.isEndPort(n))
21.                         t.statemachine←thisModule.getStateMachine(n);
22.         }
23.  }
24.  helper def: isEndPort(n:String):Boolean=
25.         LOTOS!GateAction.allInstances()→collect(e|e.name)→include(n);
26.  entrypoint rule getStateMachine(n:String){
27.         to t: UML_RT!StateMachine(
28.                 name←n+'_StateMachine'
29.         )
```

30. }
31. …

3 实 例

3.1 问题描述

下面通过一个生产室(production cell,简称 PC)的例子来说明如何基于 LOTOS 规约生成 UML-RT 模型.PC 包括为了锻压金属板坯而必须协作的几个部件,部件以尽可能快的速度同时处理金属板坯,它们必须避免将金属板坯掉地或者彼此发生冲突.PC 处理金属板坯,金属板坯通过输送带(feed belt)被输送到压机(press)上.机械臂(Arm1)从输送带上将板坯取下并将它放到压机上,然后机械臂从压机上移开,压机处理金属板坯并再次打开.最后,另外一个机械臂(Arm2)将锻压好的金属盘子从压机上取出并将它放到成品带(deposit belt)上.图 7 为 PC 的俯视图,图 8 为机器人和压机的侧视图.具体解释如下:

- 为了增大压机的使用效率,机器人(robot)装了两个机械臂,因此才有可能当压机正在锻压一只机械臂送来的板坯时,另一只机械臂能够拿起一个新的板坯.
- 机械臂处于不同的水平面,并且它们不能垂直移动.这就解释了为什么一个升降轮盘(elevating rotary table)要放置在输送带和机械臂之间.
- 两只处于不同水平的机械臂的另外一个结果是导致了压机的 3 种状态:① 打开并且准备通过低机械臂(Arm2)取出锻压好的盘子;② 打开并且通过高机械臂(Arm1)放上去一个金属板坯;③ 锻压时关闭.

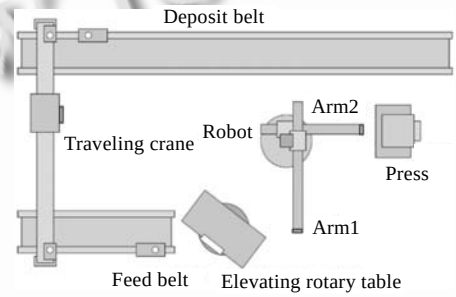


Fig.7 Top view of the production cell
图 7 PC 的俯视图

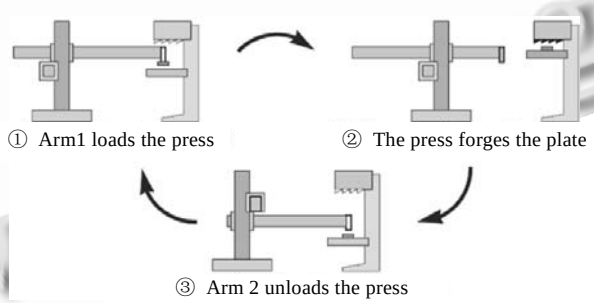


Fig.8 Side view of robot and press
图 8 机器人和压机的侧视图

3.2 LOTOS规约

因为 PC 系统整个模型较大,所以下面我们仅对机器人进行分析与建模.机器人包括两只垂直的机械臂,由

于技术原因,机械臂被安装在两个不同的水平上.每只机械臂能够水平伸缩,两臂连带旋转,机械臂必须能够水平移动,因为升降轮盘、压机和输送带放置在距机器人旋转中心不同距离的地方.每只机械臂的末端装上了电磁铁,这样,机械臂才能拿起金属盘子.机械臂的任务包括从升降轮盘上取金属板坯到压机上,并将锻造好的金属盘子从压机上取下放到成品带上.机器人的一个工作周期主要分为如下 4 步:

- 机器人顺时针旋转直到 Arm1 面向桌子,然后伸出 Arm1 从桌子上拿起一个金属板坯.
- 机器人逆时针旋转直到 Arm2 指向压机,然后伸出 Arm2 从压机上拿起一个已锻压好的盘子.
- 机器人逆时针旋转直到 Arm2 指向 Deposit 带,然后伸出 Arm2 并将锻压好的盘子放置在 Deposit 带上.
- 机器人逆时针旋转直到 Arm1 指向压机,然后伸出 Arm1 并将金属板坯放在压机上.

假设根据自然语言需求获得的机器人 LOTOS 规约如下:

```
process Robot[Extend_A1,Ext_A1,Retract_A1,Ret_A1,Stop_A1,Extend_A2,Ext_A2,Retract_A2,Ret_A2,
        Stop_A2,Load_A2,PrsRdunload]:=
RC[PrsRdunload,Extend_A2,Ext_A2,Stop_A2,Load_A2,Retract_A2]
||
Arm1[Extend_A1,Ext_A1,Retract_A1,Ret_A1,Stop_A1]
||
Arm2[Extend_A2,Ext_A2,Retract_A2,Ret_A2,Stop_A2]
where
    process
        Arm1[Extend_A1,Ext_A1,Retract_A1,Ret_A1,Stop_A1]:=
        Extend_A1;Ext_A1;Stop_A1;Arm1[Extend_A1,Ext_A1,Retract_A1,Ret_A1,Stop_A1]
        []
        Retract_A1;Ret_A1;Stop_A1;Arm1[Extend_A1,Ext_A1,Retract_A1,Ret_A1,Stop_A1]
    endproc
    process
        Arm2[Extend_A2,Ext_A2,Retract_A2,Ret_A2,Stop_A2]:=
        Extend_A2;Ext_A2;Stop_A2;Arm2[Extend_A2,Ext_A2,Retract_A2,Ret_A2,Stop_A2]
        []
        Retract_A2;Ret_A2;Stop_A2;Arm2[Extend_A2,Ext_A2,Retract_A2,Ret_A2,Stop_A2]
    endproc
    process
        RC[PrsRdunload,Extend_A2,Ext_A2,Stop_A2,Load_A2,Retract_A2]:=
        PrsRdunload;Extend_A2;Ext_A2;Stop_A2;Load_A2;Retract_A2;
        RC[PrsRdunload,Extend_A2,Ext_A2,Stop_A2,Load_A2,Retract_A2]
    endproc
endproc
```

3.3 LOTOS规约到UML-RT模型的转换

通过应用 LOTOS 规约到 UML-RT 模型的转换方法,将 Robot 的 LOTOS 规约转换为 UML-RT 模型,这个过程相当于对 LOTOS 到 UML-RT 的 ATL 规则的一次实例化.首先,基于 AMMA 平台中的 TCS 模块实现 LOTOS 文本域到模型域的转换,通过构造 Injector 直接完成 Robot 的 LOTOS 规约到 Robot.ecore 的形式转换.然后,在 ADT 中提供相应的配置信息,以便 ATL 虚拟机得以执行.图 9 给出了该配置信息的定义界面,其内容将保存在以“.launch”结尾的 ATL 配置文件中.

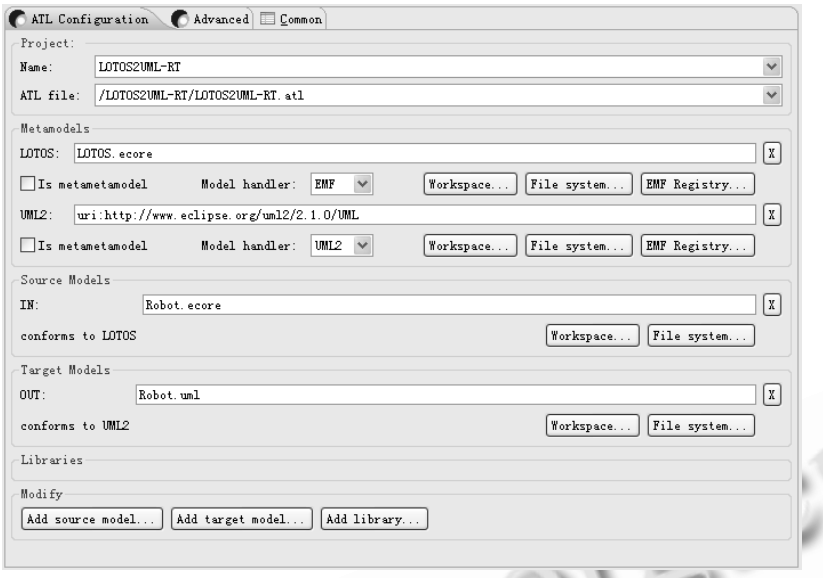


Fig.9 ATL configuration

图 9 ATL 配置视图

基于 AMMA 平台转换生成的 UML-RT 模型表示如图 10~图 14 所示.

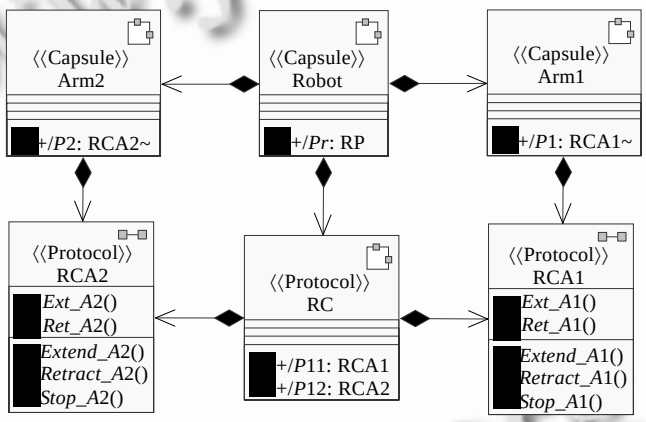


Fig.10 Robot class diagram

图 10 机器人类图

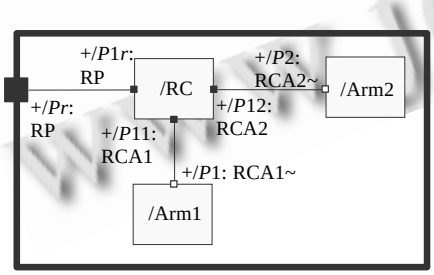


Fig.11 Robot collaboration diagram

图 11 机器人协作图

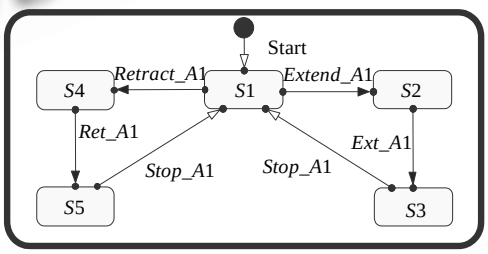


Fig.12 Arm1 statechart diagram

图 12 Arm1 状态图

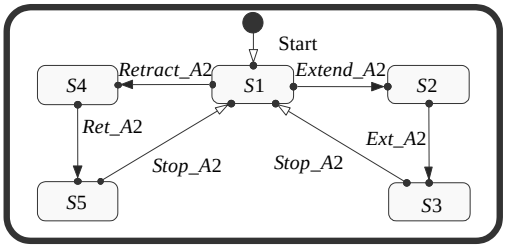


Fig.13 Arm2 statechart diagram

图 13 Arm2 状态图

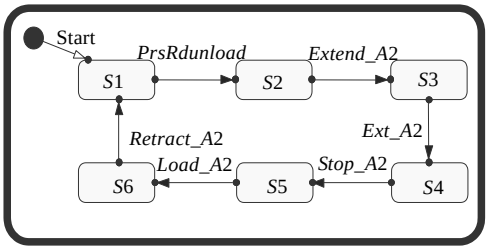


Fig.14 RC statechart diagram

图 14 RC 状态图

最后生成的 UML-RT 模型可以导入到 IBM 公司提供的 Rational Rose RealTime[20]软件中直接使用,利用 Rational Rose RealTime,建模人员能够基于 UML-RT 创建实时软件的模型,生成实现代码、编译、执行和调试应用.

4 相关工作

需求规约和 SA 构造是软件生命周期的两个关键活动:需求规约关注如何描述问题空间;而 SA 则主要关注如何描述解空间.所以,需求规约和 SA 领域中的绝大多数研究工作都相对独立.在需求阶段研究 SA,主要有如下两种工作:一是用 SA 的概念和描述手段在较高抽象层次描述问题空间的软件需求;二是探讨如何从软件需求规约自动或半自动地变换到 SA 模型.

从软件需求模型向 SA 模型的转换主要关注两个问题:① 如何根据需求模型构建 SA 模型;② 如何保证模型转换的可追踪性[21].针对这两个问题的解决方案,随着所采用的需求模型的不同而各异.文献[22]提出的 AGADUC(automated reverse engineering of UC diagram)过程能够生成描述用例文本内部工作流变化的类活动图(activity-like diagram).该过程不需要大量用例描述就能够给出用例之间相互依赖的信息,从而让使用者通过生成的类活动图掌握工作流变化.在采用特征模型描述系统需求的方法中,文献[23]采用责任作为从需求模型向 SA 模型转换的桥梁,通过“特征-责任-构件”的映射关系来维护可追踪性;文献[24]提出一种通用的方法 CBSP,该方法通过使用体系结构层次概念(构件、连接子等)来逐步精化需求,从而得到 SA 的设计模型.除了上述方法之外,还有大量的研究者参与进来,如文献[25]使用实时接口自动机网络来描述实时软件系统的构件式设计模型,使用带布尔不等式时间约束的 UML 顺序图表示基于场景的需求规约,对系统设计阶段实时软件构件的动态行为进行形式化分析与检验;文献[26]从主动服务、需求驱动、自主聚合的角度,提出了需求驱动的主动网构实体聚合模型.在该模型中,主动网构实体形成服务 Agent,这些服务 Agent 主动发现需求并向需求聚集.针对同一需求聚集形成的服务 Agent 群,通过机制设计、协商、协作最终形成能够满足需求的多 Agent 系统.

与以上研究工作相比,本文在从软件需求规约自动变换到 SA 模型方面另辟新径:从模型转换的角度来看,基于 LOTOS 规约生成的 SA 模型保留了形式化语义,所以通过这种方法建立的 SA 模型比通过自然语言规约建立的 SA 模型更为精确,不存在二义性;从可追踪性的角度来看,LOTOS 规约到 SA 模型的转换都有元模型之间映射与之对应,因此需求和 SA 设计之间保持了良好的可追踪性.此外,本文的工作也为可信软件设计提供了新的思路,给出了一套实践中切实可行的解决方案.

5 结束语

本文的工作包括:建立了自然语言规约到 SA 模型的转换框架;基于 AMMA 平台的 KM3 元模型体系,通过元建模抽象出 LOTOS 与 UML-RT 的 KM3 元模型,实现 LOTOS 与 UML-RT 的同构化;针对 LOTOS 元模型和 UML-RT 元模型构造转换规则,通过将对应的实例模型进行相互转换,实现在 MDE 下形式化规约到 SA 模型的转换;最后以一个实时系统设计为例,讨论该方法的使用效果.

本文通过一个较为复杂的实例研究了形式化规约到实时软件体系结构模型的转换问题,因此下一步需要进行更多复杂实例的研究工作,以便能够总结出完善的转换途径.此外,本文没有讨论实时系统包含时间、资源等非功能性质的需求规约到 SA 模型的转换问题,因此,探讨包含时间、资源等非功能性质的形式化规约到 SA 模型的转换方法也是我们下一步的研究工作.

References:

[1] Hoare CAR. Communicating Sequential Processes. London: Prentice-Hall, Inc., 1985. 45−72.

[2] Milner R. A Calculus of Communicating Systems. Berlin: Springer-Verlag, 1982.

[3] ISO. ISO 8807. LOTOS Standard, 1989. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=16258

[4] Selic B, Rumbaugh J. Using UML for modeling complex real-time systems. 1998. http://www.ibm.com/developerworks/rational/library/content/03July/1000/1155/1155_umlmodeling.pdf

[5] Selic B, Gullekson G, Ward P. Real-Time Object-Oriented Modeling. New York: John Wiley & Sons, Inc., 1994.

[6] de Melo Bezerra J, Hirata CM. A semantics for UML-RT using π-calculus. In: Proc. of the 18th IEEE/IFIP Int'l Workshop on Rapid System Prototyping (RSP 2007). Porto Alegre: IEEE Computer Society, 2007. 75−81.

[7] Leue S, Mayr R. A scalable incomplete test for the boundedness of UML RT models. In: Jensen K, Podelski A, eds. Proc. of the 10th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004). Berlin: Springer-Verlag, 2004. 327−341.

[8] Ilic D. Deriving formal specifications from informal requirements. In: Proc. of 31st Annual Int'l Computer Software and Applications Conf. (COMPSAC 2007). Beijing: IEEE Computer Society, 2007. 145−152.

[9] Jackson M, Zave P. Deriving specifications from requirements: an example. In: Proc. of the 17th Int'l Conf. on Software Engineering (ICSE'95). Seattle: IEEE Computer Society, 1995. 15−24.

[10] Gunter C, Gunter E, Jackson M, Zave P. A reference model for requirements and specifications. IEEE Software, 2000, 17(3): 37−43.

[11] Turner KJ. Incremental requirements specification with LOTOS. Requirements Engineering, 1997,2(3):132−151.

[12] France R, Rumpe B. Model-Driven development of complex software: A research roadmap. In: Knight J, ed. Proc. of the 29th Int'l Conf. on Software Engineering (ICSE). Minneapolis: IEEE Computer Society, 2007. 37−54.

[13] Miller J, Mukerji J. MDA guide version 1.0.1. OMG, 2003. http://www.omg.org/docs/omg/03-06-01.pdf

[14] Bézivin J, Jouault F, Rosenthal P, Valduriez P. Modeling in the large and modeling in the small. In: Aßmann U, Aksit M, Rensink A, eds. Proc. of the Model Drive Architecture: European MDA Workshops: Found and Applic (MDAFA 2003 and MDAFA 2004). Berlin: Springer-Verlag, 2005. 33−46.

[15] OMG. Meta object facility core specification v2.0. 2006. http://www.omg.org/cgi-bin/apps/doc?formal/06-01-01.pdf

[16] Budinsky F, Steinberg D, Merks S, Ellersick R, Crose TJ. Eclipse Modeling Framework: A Developer's Guide. Boston: Addison-Wesley Professional, 2003.

[17] Jouault F, Bezivin J. KM3: A DSL for metamodel specification. In: Gorrieri R, Wehrheim H, eds. Proc. of the 8th IFIP Int'l Conf. on Formal Methods for Open Object-Based Distributed Systems. Berlin: Springer-Verlag, 2006. 171−185.

[18] ATLAS Team. ATLAS transformation language (ATL) home page. 2005. http://www.eclipse.org/gmt/atl/

[19] Zhang T, Jouault F, Bezivin J, Zhao JH. A MDE based approach for bridging formal models. In: Proc. of the 2nd IEEE & IFIP Theoretical Aspects of Software Engineering (TASE 2008). Nanjing: IEEE Computer Society, 2008. 113−116.

[20] IBM Rational Rose Technical Developer. Rational Rose RealTime. Version 6.5. 2000. http://www-01.ibm.com/software/awdtools/developer/technical/

[21] Mei H, Shen JR. Progress of research on software architecture. Journal of Software, 2006,17(6):1257−1275 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

[22] El-Attar M, Miller J. AGADUC: Towards a more precise presentation of functional requirement in use case models. In: Proc. of the 4th Int'l Conf. on Software Engineering Research, Management and Applications (SERA 2006). Seattle: IEEE Computer Society, 2006. 346−353.

[23] Zhang W, Mei H, Zhao HY, Yang J. Transformation from CIM to PIM: A feature-oriented component-based approach. In: Proc. of the 8th Int'l Conf. on Model Driven Engineering Languages and Systems (MoDELS 2005). Berlin: Springer-Verlag, 2005. 248−263.

[24] Medvidovic N, Dashofy EM, Taylor RN. On the role of middleware in architecture-based software development. Int'l Journal of Software Engineering and Knowledge Engineering, 2003,13(4):367−393.

[25] Hu J, Yu XF, Zhang Y, Li XD, Zheng GL. Scenario-Based consistency verification of component-based real-time system designs. Journal of Software, 2006,17(1):48−58 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/48.htm [doi: 10.1360/jos170048]

[26] Zheng LW, Jin Z. Requirement driven automated aggregation of active Internetware entities. Journal of Software, 2008,19(5): 1083−1098 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1083.htm [doi: 10.3724/SP.J.1001.2008.01083]

附中文参考文献:

[21] 梅宏,申峻嵘.软件体系结构研究进展.软件学报,2006,17(6):1257−1275. http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

[25] 胡军,于笑丰,张岩,李宣东,郑国梁.基于场景构件式实时软件设计的一致性检验.软件学报,2006,17(1):48−58. http://www.jos.org.cn/1000-9825/17/48.htm [doi: 10.1360/jos170048]

[26] 郑丽伟,金芝.需求驱动的主动网构实体聚合.软件学报,2008,19(5):1083−1098. http://www.jos.org.cn/1000-9825/19/1083.htm [doi: 10.3724/SP.J.1001.2008.01083]

祝义(1976－),男,江西九江人,博士生,讲师,主要研究领域为软件工程,形式化方法.

黄志球(1965－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,形式化方法,数据仓库.

曹子宁(1972－),男,博士,教授,主要研究领域为形式化方法.

周航(1978－),男,博士,讲师,主要研究领域为软件工程,软件测试与形式化验证,Petri 网应用.

刘亚萍(1985－),女,硕士,主要研究领域为模型转换,形式化方法.