# 面向方面的实时系统形式化开发方法*)

陈广明[1] 张立臣[2] 陈生庆[1]

(嘉应学院计算机系 广东梅州514015)[1] (广东工业大学计算机学院 广州510090)[2]

**摘 要** 实时系统复杂性的不断增加以及对可配置性和可重用性要求的不断提高,需要如面向方面和基于组件的软件工程方法的支持,同时实时系统的可信性要求采用形式化方法来开发实时系统。本文试图建立一种面向方面的实时系统形式化开发方法,这种方法对RT-Z进行了面向方面和面向部件的扩展,并通过实时组件模型在需求和设计阶段提供了对基于部件的系统开发方法(CBSD)和面向方面的系统开发方法(AOSD)的支持。本文给出了面向方面的实时Z(AO-RT-Z)的组件模型的框架结构、语法要求、方面的联结和功能接口和非功能接口的定义,重点讨论并证明了面向方面的实时Z(AO-RT-Z)作为规格描述语言的健全性。

**关键词** 面向方面,实时系统,组件,形式化方法,RT-Z,AO-RT-Z

## Aspect-Oriented Formal Development Method for Real-Time Systems

CHEN Guang-Ming[1] ZHANG Li- Cheng[2] CHEN Sheng-Qing[1]

(Computer Department of Jiaying College, Guangdong, Meizhou 514015)[1]

(Computer College, Guangdong University of Technology, Guangzhou510090)[2]

**Abstract** Increasing complexity of real time system, and demands for enabling their configurability and reusability are strong motivations for aspect-oriented and component-based development, and dependability of real time systems requires that formal development methods are taken during real-time development cycle. This paper attempts to establish an aspect-oriented formal development method for real-time systems with the component-based and aspect-oriented extension of RT-Z, its model of real time component can be used for CBSD and ASD. The framework, syntax, weaving aspects, and interface of function and non-function are defined in this article, the healthiness conditions of AO-RT-Z is emphasized and a proof is made.

**Keywords** Aspect-oriented, Real-time system, Component, Formal method, RT-Z, AO-RT-Z

## 1 前言

软件方法学的核心就是将问题模型中"交织"在一起的细节进行有效分解,使被分解的各部分细节独立精化并最终通过有效的手段相结合。因此,确定分解原则和结合方式充分适应软件开发过程的内在特征是其基本要素。面向对象方法将问题模型分解为彼此独立的对象,以消息机制相结合,由于能有效映射大多数事务的客观状态,因此取得了广泛的成功。然而,实际问题中存在着类似实时性、安全策略、异常扑获、日志处理等需求,这些需求的实现需要跨越多个对象和模块,横切(crosscut)整个系统,面向对象方法将它们分解为各对象的内部元素加以处理,破坏了内在的统一性和完整性,无法简单地通过片段的精化推进系统的演化。

基于组件的软件开发(component-based software development,CBSD)将系统分解为相互独立的组件,通过严格的连接规范全面地定义了组件之间的功能接口和非功能接口,这种思想将传统的模块化设计和面向对象的开发手段有效地结合,降低了系统开发的复杂性,增大了软件重用的粒度,同时能有效适应系统的升级需要。

面向方面的编程(Aspect Oriented Programming,AOP)出现于1997年,它吸取了面向对象开发的优点,同时能使开发者关注延伸于整个系统的某个方面特征的实现而无须了解其内在关系,最后通过联结器实现系统不同方面代码的结合,目前支持AOP的语言有AspectC、AspectC++、AspectJ等[1]。随着其应用领域的扩展,从软件生命周期的宏观视角构建对AOP支持的AOSD(Aspect-Oriented Software Department)研究成为热点领域。

组件实现了对系统的功能有效分解,而AOSD有效地解决了系统横切关注点(crosscuting conerns)的处理问题,两者的结合可以极大地改善软件开发过程。其基本的开发过程为:(1)系统分解为组件;(2)通过组件语言实现组件的一般功能性需求;(3)组件中需要横切的需求(crosscuting requirements)通过aspect联结机制实现;(4)非功能性的需求通过aspect的联结机制或其它合适方式加以实现。其目标建立结合组件、面向方面的思想,建立贯穿于整个软件生命周期的整体解决方案,而绝非仅在实现阶段寻求AOP的语言支持[2]。

由于实时系统的复杂性和这种系统中具有多种影响因素相互交织的特性,使得面向方面的开发方法非常适合实时系统的开发。形式化方法具有描述的精确性、精化方法的有效性以及无二义性,同时其推理机制对系统正确性的保证决定了其作为一种重要的方法应用于需求和设计阶段。RT-Z是将形式化方法Z和Timed-CSP相结合而成的支持实时并发特性描述并能应用于需求和设计阶段的的规格描述语言,然而它的结构和语法并不支持组件设计和AOP方法,本文提出的AO-RT-Z(Aspect Oriented RT-Z)扩展了RT-Z,使其支持

*)国家自然科学基金(No.60474072、No.60174050)、广东省自然科学基金(No.04009465、No.010059)、广东省高校自然科学研究项目(No.Z03024)基金资助。陈广明 讲师,硕士,主要研究方向:软件工程技术、形式化方法;陈生庆 讲师,硕士,主要研究方向:软件工程技术、实时系统;张立臣 教授,主要研究方向:分布式、实时系统。

AOSD 和 CBSD，同时论证了这种扩展的有效性和合适性。文章的第2节介绍基于 Timed-CSP 和 Z 而集成的形式化方法 RT-Z；第3节定义了 AO-RT-Z(Aspect Oriented RT-Z)的框架结构；第4节给出了 AO-RT-Z 的语义模型，证明了该框架作为形式语言的健全性；最后对全文进行总结。

## 2 形式化方法 RT-Z

形式化方法以数学为基础，能够清晰、精确、抽象、简明地规范和验证软件系统及其性质，极大地提高软件的安全性和可靠性。由于单一形式化方法的描述能力有限，难以将复杂系统的各层面的需求建立在一个统一的框架中。而将不同形式化方法相结合并重新定义其语法结构和基本语义，形成集成的形式化方法，能在很大程度上弥补这方面的不足。

为实时系统开发而设计的形式化规格说明方法 RT-Z 结合了 Z 的状态描述能力和 Timed-CSP 的进程描述能力，其基本原理是以 Timed-CSP 进程项表达式表示系统的实时、并发关系，用 Z 的操作模式表示进程所形成的状态转化，而 Z 部分的状态及其约束条件又成为系统进程转换的依据[3~6]。RT-Z 以此为基础来定义的语法、结构和语义模型，实际上用 Z 定义 Timed-CSP 的语义模型，即可在保持 Z 的几乎所有语法要素的情况下同 Timed CSP 在语义解释上相一致。由于 Timed CSP 有不同的语义模型，因此这种定义是有差别的，这也使最终形成的规格说明语言规范有所不同。

RT-Z 并非单纯的规格说明语言，它同时是一种具有严格数学精确性的建模语言，同专用实时语言 PEARL(Process and Experiment Automation Realtime Language)相结合的统一开发方法可以完成从系统的需求获取直到代码的最终实现[8,9]。基于一阶谓词逻辑和集合论的 Z 模式和基于进程代数的 CSP 从语法结构、语义机制和表达方式都很适合面向方面的软件开发。鉴于实时和嵌入式系统开发关注点的多样性和复杂性，RT-Z 对 AOSD 的支持扩展具有良好的应用价值。

## 3 AO-RT-Z(Aspect Oriented RT-Z)的框架结构

### 3.1 基本原理

CBSD 采用黑盒方式，强调组件的封装和隐藏性，而 AOSD 更强调通过联结(weaving)过程来改变组件内部的代码，优化或定制系统的横切关注点(crosscut concert)，采用的是白盒方式，为了使二者能有机地结合，本文提出一种“灰盒组件”(gray box component)的概念，它能够在尽量保持黑盒组件主要特征的情况下通过定义的方面接口(aspects interface)通过联结过程改变其内部代码。整个组件的原理如图1所示。

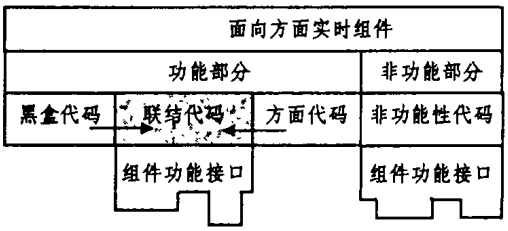


图1 面向方面的实时组件基本原理

同传统组件实现原理不同，“灰盒组件”规定了方面代码的表达方式及其同黑盒代码的联结方式。由于 RT-Z 并不支持此种要求，因此需要对其结构、语法和基本语义进行必要的扩充。

### 3.2 AO-RT-Z 组件的构成

AO-RT-Z 组件框架改变了 RT-Z 将 CSP 部分和 Z 部分分别定义的结构特点，更直观地体现了组件的封装性，同时按照组件的需要增加了接口和面向方面扩展的语法规范。

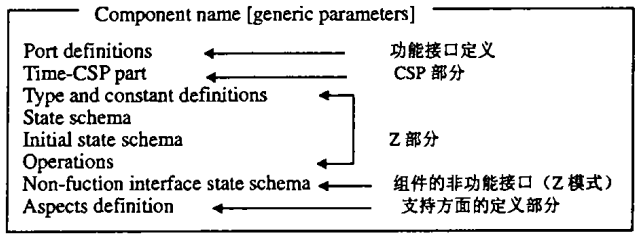


图2 AO-RT-Z 的实时组件框架

①Port definitions 定义了组件间的功能接口，接口的表示为 Port $p:[p1:ty1;\ p2:ty2;\cdots\cdots pn:tyn]$，它代表了组件通信的事件集合，$P$ 是通道名，与其同名的 Z 操作模式定义了对该具体的操作，$[p1,\ p2,\cdots,pn]$是参数名，可以为空，$ty1,ty2,\cdots,tyn$ 是参数的类型，不同组件接口通过这些事件进行同步，从而实现了组件的连接。

②Time-CSP 部分是一组 CSP-Name＝CSP-Process 结构的表达式集合，此处的 CSP-Process 使用经过 Z 类型扩展的 CSP 模型检查器 FDR(Failure divergence relation)的语法结构[7]，为统一起见，使用关键字 main 作为主进程的标识。

③Type and constant definitions 定义了组件中的类型和常量；State schema 定义了组件中的状态模式；Initial state schema 初始化时的状态模式；Operations 定义了操作模式，给出了对同名事件的响应操作。

④Non-fuction interface state schema 用 Z 的状态模式定义了组件之间的非功能接口部分，它包括 provides 和 allows 两个部分，allows 定义了装配该组件所需要的环境状态及对其他组件的状态要求，provides 表明该组件的非功能状态。

⑤Aspects definition 部分定义了组件对方面的支持，它是三元组$\langle pointcut, advice, mode\rangle$的集合，其中 *pointcut* 是 Time-CSP 部分所含有的进程项，而 *advice* 同名于某个操作模式的 Time-CSP 进程项，$mode::=\langle posimode, timedmode\rangle$，$posimode\in\{before, after, repeat\}$，$timedmode\in\{\langle time, event\rangle \mid time:R^{+}\wedge event:\sum^{+}\}$，*time* 表示时间项，$\sum^{+}$表示事件集合，其中 NULL 表示空事件，因此二元组$\langle time, event\rangle$表示事件及其上的时间约束。

### 3.3 AO-RT-Z 组件的接口

非功能接口采用标准 Z 模式，经转换可用 CSP 的 Failure-Devegence 语义模型表示为二元组$(\{F(C.\text{provide}), F(C.\text{allow})\}, \{D(C.\text{provide}), D(C.\text{allow})\})$

**定义1** 组件 $C1$ 与 $C2$ 是非功能匹配的当且仅当 $F(C.\text{provide})\subseteq F(C.\text{allow})\wedge F(D.\text{provide})\subseteq(D.\text{allow})$

从定义1可知所有能够连接的组件必是非功能匹配的。

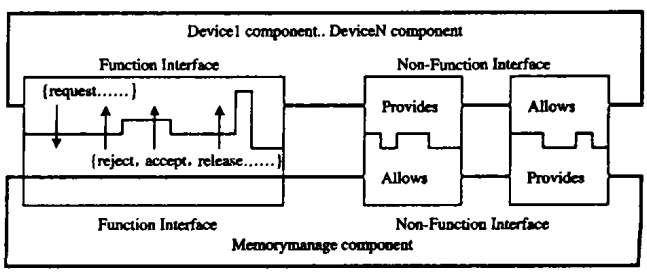


图3 内存管理组件

下面以内存管理系统为例说明组件功能接口的连接，内存管理组件(Memory Manage Component，简称 MMC)有功能性接口和非功能接口。功能接口通过通道 request，reject，accept，release 一组设备组件 Device1 component，…，DeviceN component(简称 DC1…DCN)相连接，由于设备可以有多个，它们分别同 MC 相连接，精确的连接语义可表示为 Timed-CSP 表达式：

$$\begin{aligned}MemorySystem = (\cdots((\ & Memorymanage[|\{|request, reject, accept, release\ |\}|]\ Device1)\\ & [|\{|request, reject, accept, release|\}|]\ Device2)\\ & \cdots)\\ & [|\{|request, reject, accept, release\ |\}|]DeviceN)\\ & \backslash[|IntMemoryProcessor \cup IntDevice1Processor \cup\\ & IntDevice2Processor \cup \cdots \cup IntDeviceNProcessor\\ & |]\end{aligned}$$

其中 *IntMemoryprocessor*，*IntDeviceI* 等表示组件内部进程，它们在系统中是不可见的。

### 3.4 AO-RT-Z 组件的方面联结

Time-CSP 项的 BNF 定义为[11]：

$$P ::= STOP \mid SKIP \mid WAIT\ t \mid a \rightarrow P \mid P;P \mid P \overset{t}{\triangleright} P \mid P \Box P \mid P \sqcap P \mid \sqcap_{i \in I} P_i \mid a:A \rightarrow Pa \mid$$

$$PA \| AP \mid P \| \mid \mid P \backslash A \mid f(P) \mid f^{-1}(P) \mid X \mid \mu X \cdot P$$

**定义2** 对于三元组$\langle X, Y, \langle p, \langle t, u\rangle\rangle\rangle$，$X$ 和 $Y$ 为 Time-CSP 的两个进程项符号，$\langle t, u\rangle$为具有时间约束性的事件，两个进程项符号的胶合表示为 $Z = glue(X, Y)$，如果 $u =$ NULL，$glue(X,Y) = (X \overset{t}{\rightsquigarrow} Y)$；如果 $u = NULL$，$glue(X, Y) = (X \xrightarrow{\langle t,u\rangle} Y)$；方面的联结(weaving aspects)过程为：(1)当 $p =$after $X = glue(X,Y)$；(2)当 $p =$before $X = glue(Y, X)$；(3)当 $p =$repeat $X = Y$。

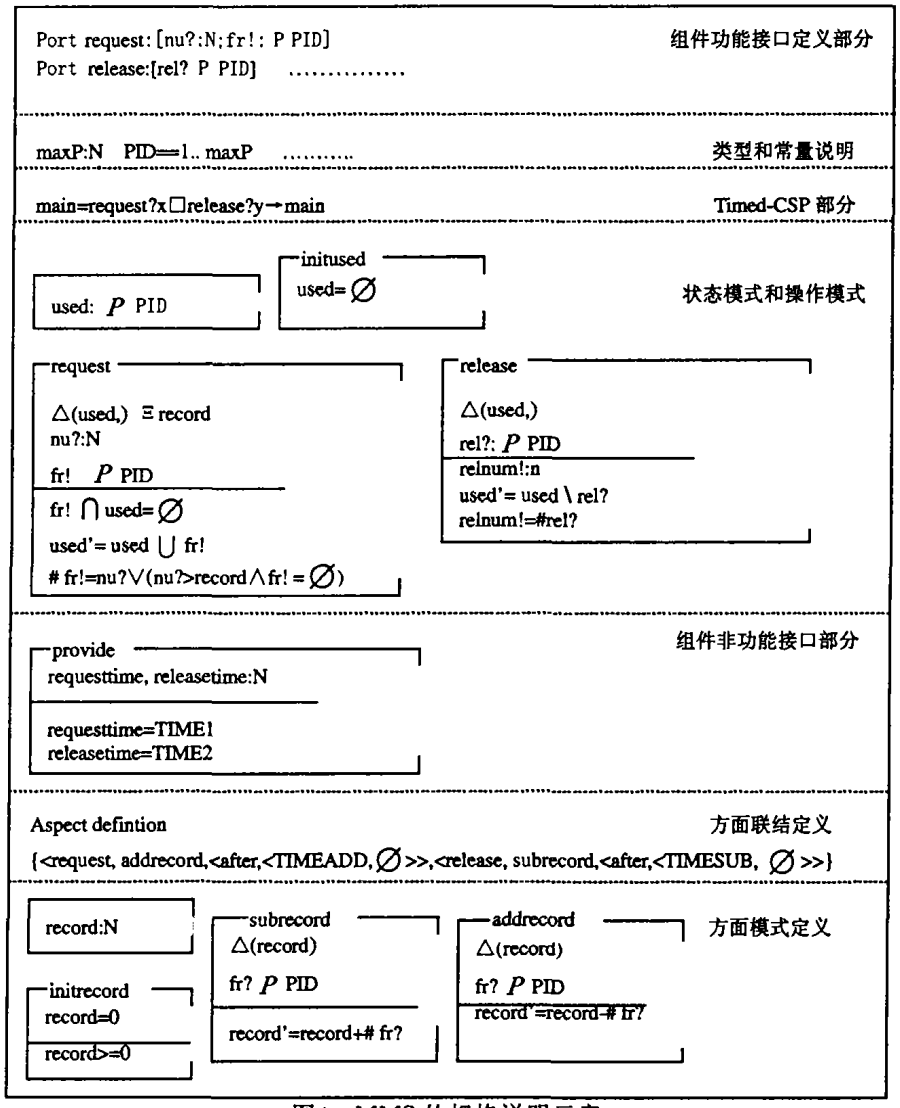


图4 MMS 的规格说明示意

联结过程将方面定义嵌入了规格说明，从而实现了对 AOP 的支持。图4给出内存管理组件的框架，下面说明方面的联结过程。

内存管理组件的主要功能是接受外部设备的申请，分配和释放内存，本例忽略了许多细节以突出主要问题，当接受到其它组件的请求产生 request 事件，并根据参数分配内存，产生的 release 事件用于释放内存，Timed-CSP 进程描述了系统的并发进程。

在组件功能分解过程中将可用内存的记录作为一个方面从主功能中剥离，起到了简化作用。该方面定义的操作模式 addrecord 和 subrecord 在分配和释放内存后记录可用内存，根据定义2，在联结(weaving aspect)后，主进程转换为 main = ( request? $x$ $\overset{TIMEADD}{\rightsquigarrow}$ addrecord ) □ ( release? $y$ $\overset{TIMESUB}{\rightsquigarrow}$ main)，因此组件能够实现对内存空间的判定，并据此进行分配。

## 4 AO-RT-Z 的语义模型

根据 AO-RT-Z 的框架结构和有关联结的定义，其基本

构成元素仍为 Timed-CSP 的进程项和 Z 模式，将 Z 部分定义为 Timed-CSP 的语义模型是实现两种形式化方法有效结合的前提。同时，只有满足某种约束条件的数学形式才能合理地表现实际系统。本节将形式化地定义 AO-R-Z 的元素，赋予其 FD(failures divergence)语义，并证明其作为规格说明的合适性(well defined)，AO-RT-Z 不同于标准的 RT-Z，由于 FD 模型没有实时化，因此需要将事件解释为具有时间约束性，从本质看这种解释将不影响本问题的讨论。

### 4.1 Timed-CSP 的 FD 语义模型及健全条件

**定义3** $A$ 是一个可能的事件符号，CSP 语义规格说明是一个二元组$\langle F,D\rangle$，失效(failure)集合 $F:\text{seq}A\leftrightarrow\mathbb{P}A$ 是一个由 $A$ 的序列和 $A$ 的子集构成的二元关系，$\langle tr,X\rangle\in F$，$tr$ 是一个具有时间约束性的事件序列，当系统处理 $tr$ 事件序列后，事件 $e\in X$ 的通信将被拒绝。发散(divergence)集合 $D:\mathbb{P}\text{seq}A$ 是系统可以发散的事件序列的集合，$D\subseteq_{\text{dom}}F$，

**定义4** 对于任何 $s,t\in seqA$ 并且任何 $X,Y\subseteq A$，下面的条件称为健全条件：

$$F\neq\varnothing\wedge\langle s\frown t,\varnothing\rangle\in F\Rightarrow\langle s,\varnothing\rangle\in F \quad (a)$$

$$\langle t,X\rangle\in F\wedge Y\subseteq X\Rightarrow\langle t,Y\rangle\in F \quad (b)$$

$$\langle t,X\rangle\in F\wedge\forall\, a:Y\cdot\langle t\frown\langle a\rangle,\varnothing\rangle\notin F\Rightarrow\langle t,x\cup Y\rangle\in F \quad (c)$$

$$(s\in D\Rightarrow S\frown t\in D)\wedge(s\in D\Rightarrow\langle s\frown t,x\rangle\in F) \quad (d)$$

符号$\frown$表示序列的拼接，满足这些条件的$\langle F,D\rangle$被称为健全的，由$\langle F,D\rangle$为基本元素的规格说明是合适的。

(a)要求一个事件序列的前缀在关系 $F$ 所体现的运算中封闭于该序列，直观的解释就是如果系统处理事件序列 $s$ 后存在被拒绝通信的事件，那么该序列的任何前缀也应如此；(b)要求被拒绝事件的子集封闭于该拒绝事件，即如果对事件序列 $t$，$X$ 是拒绝事件的集合，那么 $X$ 的子集 $Y$ 也是事件序列 $t$ 的拒绝事件集合；(c)$\langle t\frown\langle a\rangle,\varnothing\rangle\notin F$ 表示事件序列 $t\frown\langle a\rangle$ 不拒绝任何事件的通信，这是系统所不应容许的，因此在处理事件序列 $t$ 后将拒绝事件$\langle a\rangle$；(d)以一个发散(divergence)的事件序列为前缀的事件序列也是发散的，系统发散时，任何事件的通信都将被拒绝。显然，健全条件规范了合适的规格说明，使系统的合式公式能得到合理的解释。

### 4.2 AO-RT-Z 的 FD 语义模型的定义

为了实现整个规格说明的语义的一致性，需要定义 AO-RT-Z 框架中的其它部分的 FD 语义。$Id_c$ 表示所有通道的标识符；$Id_v$ 表示变量名；$Id_p$ 表示参数名；$Vaule$ 表示具体的值。这样状态系统状态可以表示为一个有限部分函数 $State==Id_c\nrightarrow Vaule$，而参数 $Parameter==Id_p\nrightarrow Vaule$，系统的事件可以表示为 $Event==Id_c\times Parameter$，显然经过联结过程，未包含 Timed-CSP 部分的 AO-RT-Z 组件可抽象为如下的基本元素：

ComponentStruct

$states,initial: P\ State$

$channels: P\ Id_c$

$trans: Event\rightarrow(State\nrightarrow P\ State)$

$simple_p: Id_c\nrightarrow Id_p$

$in_p,out_p: Id_c\nrightarrow P\ Parameter$

$enable: Id_c\times State\nrightarrow P\ Parameter$

图5 组件规格说明的基本元素

这些元素包括通道、状态和初始状态的集合 $channels$，$state$，$initial$；转换关系 $trans$；每个通道的输入、输出和一般参数集合 $in_p$，$out_p$，$simple_p$，以及通道在每个状态的可用参数集合 $enable$。集合 $trans(e)(st)$包含了所有在状态 $st$ 下经过事件 $e$ 所可以达到的状态集合。

为了应用的一般性，下面的定义基于 ComponentStruct。

$$\begin{aligned}&reachable:ComponentClass\rightarrow(seq\ Event\rightarrow P\ State)\\&\forall\, C:ComponentStruct\cdot D(C)=\\&\quad reachable(C)(\langle\rangle)=C.initial\wedge\\&\quad\forall\, tr:seq\ Event\cdot reachable(C)(tr\frown\langle e\rangle)=\\&\qquad\{st:State\mid(\exists\, st_1:reacher(C)(tr)\cdot st\in C.trans(e)(st_1))\}\end{aligned}$$

$reachable$ 递归地定义了组件 $C$ 在经过事件序列后可能达到的状态集合。

$$\begin{aligned}&D:ComponentStruct\rightarrow P\ seq\ Event\\&\forall\, C:ComponentStruct\cdot D(C)=\\&\quad\{s,t:seq\ Event;c:Id_c;v:Parameter\mid\\&\qquad\exists\, st:reacher(C)(s)\cdot((C.simple_p(c)\lhd_v)\in C.enable(c,st)\wedge\\&\qquad\quad\forall\, o:C.out_p(c)\cdot C.trans(c,v\oplus o)(st)=\varnothing)\cdot(s\frown\langle c,v\rangle\frown t)\}\end{aligned}$$

函数 $D$ 定义了组件可以发散的事件序列的集合，其中 $enable(c,st)$表示了在特定的通道和系统状态下的可能的参数，$simple_p(c)$表示在通道 $c$ 中的一般参数，$out_p(c)$表示输出参数，从定义不难看出在事件序列前缀 $s$ 下存在可达状态，而对事件$\langle c,v\rangle$则不存在可达状态。显然任何包含以 $s\frown\langle c,v\rangle$为前缀的事件序列也不应该存在可达状态，而这样的事件序列也应该拒绝任何通信。这一点在函数 $F$ 的定义中得以体现。

$$\begin{aligned}&refusal:ComponentStruct\rightarrow(State\mid\rightarrow PPEvent)\\&\forall\, C:ComponentStruct\cdot(dom\ refusal(C)=C.states\wedge\\&\forall\, st:C.states\cdot\forall\, x:refusal(C)(st)\cdot C.enable(c,st)\\&\forall\, c:C.channels\cdot\forall\, sp:\cdot C.enable(c,st)\cdot\forall\, ip:C.in_p(c)\cdot\\&\exists\, op:C\cdot out_p(c)\cdot((c,i_p\cup o_p\cup s_p))\notin X\wedge\\&\quad(C.trans(c,i_p\cup o_p\cup s_p)(st)\neq\varnothing\vee\\&\quad\forall\, o_p:C.out_p(c)\cdot C.trans\ ((c,i_p\cup o_p\cup s_p)(st)=\varnothing)))\end{aligned}$$

$refusal$ 函数表示了组件中的状态和在该状态下所拒绝事件幂的关系，选择这种结构有利于其后对 $F$ 函数的定义。

$$\begin{aligned}&F:ComponentStruct\rightarrow P(sep\ Event\times P\ Event)\\&\forall\, C:ComponentStruct\cdot F(C)=\\&\quad\{\langle tr,X\rangle/\exists\, st:reacher(C)\cdot X\in refusal(C)(st)\}\\&\quad\cup\{\langle tr,X\rangle\mid D(C)\times P\ Event\}\end{aligned}$$

函数 $F$ 定义了组件失效(failures)集合。

由以上组件元素的定义可得如下定理：

**定理1** AO-RT-Z 的动态语义$\langle F(C),D(C)\rangle$满足健全条件。

证明：由 $refusal$ 的定义容易得出以下结论：对任意组件 $C$ 和状态 $st$ 有：

$$\forall\, X:refusal(C)(st)\cdot\forall\, Y\subseteq X\cdot refusal(C)(st) \quad (e)$$

类似由 $reachable$ 的定义有

$$\forall\, s,t:seq\ Event\cdot reachable(C)(s\frown t)\neq\varnothing\Rightarrow reachable(C)(s)\neq\varnothing \quad (f)$$

假设 $C.initial$ 非空，对所有的初始状态 $st$ 有 $reachable(C)(\langle\rangle)\neq\varnothing$，这样$(\langle\rangle,\varnothing)\in F(C)$，再考虑(f)可得结论(a)；由 $F(C)$的定义及(e)可得结论(b)；由 $D(C)$和 $F(C)$定义可得结论(d)。

下面证明(c)假设$\langle t,X\rangle\in F(C)$并且$\wedge\forall\, a:Y\cdot\langle t\frown\langle a\rangle,\varnothing\rangle\notin F(C)$，根据(d)有$\forall\, a:Y\cdot\langle t\frown\langle a\rangle\rangle\notin D(C)$ *，令 $st1\in reachable(C)(t)$，这样有 $X$ 满足 $X\in refusal(C)(st1)$，只需证明 $X\cup Y\in refusal(C)(st1)$即可，让 $c\in C.channels$，$sp\in C.enable(c,st1)$，$ip\in C.inp(c)$。设存在 $op\in C.outp(c)$，这样 $refusal$ 的定义条件得到满足。

情况一：当 $C.trans(c,i_p\cup o_p\cup s_p)(st1)\neq\varnothing$，则$\langle t\frown\langle c,i_p\cup o_p\cup s_p\rangle,\varnothing\rangle\in F(C)$，因此$\langle c,i_p\cup o_p\cup s_p\rangle\notin Y$。

情况二：$\forall\, op:C.out_p(c)\cdot C.trans\ (c,i_p\cup o_p\cup s_p)(st)=\varnothing$，由 $D$ 的定义知$(t\frown\langle c,i_p\cup o_p\cup s_p\rangle)\in D(C)$，同 * 矛盾，因

(下转第217页)

对于这种问题的解决办法之一是:编码每一种状态和与该状态下的给定的合法输入符号对应的下一个状态。这实际是手工编码构造一个特定的自动机。对于小的规则,或者系统中只存在一种规则,这是一种可行的方法。它比较直观,简明。但是系统中规则显然不只一种,并且有些规则是复杂的。那么,自动构建自动机就显得相当必要了。

根据元数据建模的步骤,本问题的可变部分就在于自动机本身。因此需要对自动机进行建模。一个自动机包含一个状态集合,一个输入符号集合,一个转移集合和一个初始状态。图3给出了一个简化的自动机的元数据片断。

```
〈AutoMata〉
 〈StateDefine〉
  〈State name="INIT" description="start auto machine"/〉
  〈State name="DEC" description="variable declaration"/〉
  〈State name="REF" description="reference"/〉
  〈State name="ASS" description="assignment"/〉
  〈State name="EXC" description=" something error"/〉
 〈/StateDefine〉
 〈TransitionDiagram startState="INIT"〉
  〈StateItem state="INIT"〉
  〈Transition input="a" nextState="ASS"/〉
  〈Transition input="r" nextState="REF"/〉
  〈Transition input="u" nextState="DEC"/〉
 〈/StateItem〉
 〈StateItem state="DEC"〉
  〈Transition input="a" nextState="ASS"/〉
  〈Transition input="r" nextState="EXC"/〉
  〈Transition input="u" nextState="EXC"/〉
  〈Transition input="X" nextState="INIT"/〉
 〈/StateItem〉
   〈!--……--〉
 〈/TransitionDiagram〉
〈/AutoMata〉
```

图3 一个简化的自动机的元数据片断

利用元数据,我们实现了自动机构建的自动化。定义新的规则,只需要一份描述该规则的元数据,利用自动机引擎,就可以动态建立一个新的自动机来识别新的规则。这样就实现了灵活的扩展。

**总结和展望** 本文通过对构建软件产品簇的过程进行分析,提出框架和构件是建立可复用的灵活系统的基本部件,并且提出了基于 XML 描述的元数据作为粘合剂连接框架和构件的方式。实践证明,这种方式构造的系统具有足够的可扩展性和实现的简单性。在北航软件工程研究所开发的基于框架和插件技术的可复用软件测试平台中,元数据建模和以元数据连接不同部件的技术是核心和基础技术之一。我们相信,在该平台中,元数据技术将得到很好的应用和发展。

## 参 考 文 献


1 Czarnecki K, Ulrich W. Eisenecker Generative Programming : Methods, Tools, and Applications Addison Wesley/Pearson, 2000

2 Mili H, et al. Reuse-Based Software Engineering Techniques, Organization, and Controls by John Wiley & Sons, 2002

3 Parsons D, Rashid A, Speck A. A "framework" for object oriented frameworks design; Telea, A. ; Technology of Object-Oriented Languages and Systems, 1999. In: Proc. of, 1999. 141～151

4 Gamma E, et al. Design Patterns: Elements of Reusable Object-Oriented software Addison Wesley/Pearson, 1994

5 Bretherton F P, Singley P T. Metadata: a user's view Scientific and Statistical Database Management, 1994. In: Proc. Seventh Intl. Working Conf. on, Sept. 1994. 28～30

6 Atarashi R S, Kishigami J, Sugimoto S. Metadata and new challenges Applications and the Internet Workshops, 2003. In: Proc. 2003 Symposium on, Jan. 2003. 395～398

7 Pittas N, Jones A C, Gray W A. Evolution support in large-scale interoperable systems: a metadata driven approach Database Conference, 2001. ADC 2001. In: Proc. 12th Australasian, 2001. 161～168

8 de Carvalho Moura A M, Estevao da Silva L A. Machado Campos M L. A metadata approach for designing configurable interfaces in digital libraries Database and Expert Systems Applications, 2001. In: Proc. 12th Intl. Workshop on, Sept. 2001. 942～947

9 http://ant.apache.org


(上接第192页)

而有$\langle c, i_p \cup o_p \cup s_p \rangle \notin Y$。

综合情况一、情况二,(c)得证。

**总结及今后的工作** AO-RT-Z 扩展形式化规格说明 RT-Z 于面向方面的实时组件设计,提供了对 AOP 的有效支持,利用形式化方法所固有的推理证明机制和辅助工具,能够保证实时和嵌入式系统的重要组件规格和设计的无二义性,方面联结的支持保证了系统模块分析的独立性、简洁性和正确性,为 AOP 提供了有力的支持。

专门的实时语言通常都能具有对实时和嵌入式系统各方面的独立编程支持,因此将 AO-RT-Z 同这些语言相结合构建一个完整的开发框架具有良好的应用价值。

UML、AOP 技术及形式化的结合,充分利用 AOP 的思想和 UML 的强大的建模能力和形式化方法的严谨性及语义,是今后研究的一个方向。

把时间分离出来构造为一个时间方面,建立一个比较完整的时间模型来建模系统时间,促使时间方面与系统其它方面分开,能够进行单独的时间方面设计,同时又能够根据需要把设计好的时间方面织入到系统中,从而简化了实时系统建模的复杂性,是今后的另一个研究方向。

## 参 考 文 献


1 The AspectJ Programming Guide. Xerox Corporation ,September 2002. Available at: http://aspectj. org/doc/dist/progguide/index. html

2 任洪敏,钱乐秋. 构件组装及其形式化推导研究. 软件学报,2003, 14(6):1669～1677

3 Suhl C. RT-Z: An integration of Z and timed CSP[R]. In: [AGT99], 1999. 51～65

4 Suhl C. Applying RT-Z to develop safety-critical systems[C]. In: Proc. of the Third Intl. Conf. on Fundamental Approaches to Software Engineering (FASE 2000), number 1783 in Lecture Notes in Computer Science, Springer-Verlag, 2000. 51～65

5 ZUM'98. The Z Formal Specification Language[C]. number 1493 in Lecture Notes in Computer Science, Springer-Verlag, 1998. 5～23

6 Andrews J H. Process-algebraic foundations of aspect-oriented programming. In: Proc. of the Third Intl. Conf. on Metalevel Architectures and Separtion of Crosscutting Concerns, volume 2192 of Lecture Notes in Computer Science, Benlin: Springer-Verlag, 2001. 187～209

7 RAISE Language Group. The RAISE Specification Language[C]. BCS Practitioner Series. Prentice-Hall, 1992

8 陈广明,陈生庆,张立臣. Z 实时扩展及基于多视点的应用模式. 计算机应用,2005,25(2)

9 ftp://ftp. irt. uni-hannover. de/pub/pearl/report. pdf(in English)

10 Fischer C. How to combine Z with a process algebra. [R]. In J. P. Bowen, A. Fett, M. G. Hinchey, editors ,2000

11 Stankavic J, Zhu R, Poornalingam R, et al. "VEST": an aspect-based composition tool for real-time systems. In: Proc. of the 9th Real-Time Applications Symposium 2003, Toronto, Canada: IEEE Computer Society Press, May 2003. 110～123

12 stoyenko A, Marlowe T. Polynomial-Time Transformations and Schedulablity Analysis of Parallel Real-Time Programs with restricted Resource Contention Real-Time Systems. Fall 1992. 307～329

13 Schneider S. An Operational Semantics for Timed CSP[C]. number 1453 in Lecture Notes in Computer Science, Springer-Verlag, 1997. 45～61