基于反射的实时构件模型规约描述研究*)

黄 靖 卢炎生 徐丽萍

(华中科技大学计算机科学与技术学院 武汉 430074)

摘 要 以现有的构件模型为研究基点,应用反射技术,针对实时应用系统的开发,提出一类新的构件模型——反射式实时构件模型,由反射式实时构件语义模型和反射式实时构件语法模型组成。该模型在规约构件应有的功能需求特征的基础上,有效地标识构件的时间约束特征,使得与传统的功能性构件区别开来,能被系统开发者更好地选用。同时,该模型结合反射技术,能根据用户需求的变化,对实时构件进行动态修改,以便更准确地保障实时应用系统的构建与开发,并增强构件设计的活性,达到构件更好实现的效果。反射式实时构件模型既是一个构件理论模型,也是一个工程模型。

关键词 反射,构件模型,实时

The Research of Timing-Constraint Component Model on the Basis of Reflection Technology

HUANG Jing LU Yan-Sheng XU Li-Ping

(School of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan 430074)

Abstract The reflection-based real-time component model mainly researches on timing-constraint component model or horizontal reuse in real-time application software. Starting from traditional component model and using reflection technology, a new component model——reflective real-time component model is brought forward while developing real-time software system. The model is composed of reflective real-time semantic and grammatical model. Besides the necessary functional requirement features of specification components, it can effectively identify the time-constraint feature of components, which makes it different from the traditional functional components and therefore a better choice for the software system developer. Also, integrated with reflection technology, it can make dynamic update according to changed customer requirement to protect the constructing and developing of real-time application system more safely. This enhances the flexibility of component design and results in better products. Reflective real-time component model is a theoretical component model as well as a project model.

Keywords Reflection, Component model, Real-time

1 引言

软件系统的复杂性不断增加,软件危机不断加剧,软构件技术被广泛视为下一代软件开发的新方法。构件技术提供了一种开发环境,允许不同的开发人员在不同的时间段内进行不同的构件活动,以完成同一个构件。这些构件活动包括构件的设计、生产制作、组装、管理以及演化等。其中,构件模型就是这些构件活动的基础,用于指导基于构件的软件开发(Component Based Software Development,CBSD),是软件复用理论研究与实践过程中水平复用的复用单位。它强调构件的标准化,对构件本质属性、构件接口、构件环境以及构件间的交互机制进行定义和抽象描述,提供一个为所有构件生产者和构件复用者接受的一致的表达方式,主要包括构件的语义、语法规约描述。

构件模型研究有很多,代表性的有 Unicon、C2、Wright、Darwin、Rapide、ACME 等,成熟的工业模型有 COM、CCM 和 EJB。实践表明,这些研究和构件模型的主要关注点是软构件对用户功能需求业务逻辑的封装,缺乏在某些特定应用环境中非功能性需求特征的语义和语法规约描述。比如目前在嵌入式实时系统的构建中,就欠缺对功能需求所应承载的时间特征的有效语义、语法规约描述机制。

另外,构件模型作为软件水平复用中构件生产的设计指南,是一种设计时构件。它一边面对的是软件需求分析,一边服务于构件实现,因而需求的变化经常会牵动设计时构件的变动及演化。这就需要构件模型在自身模型中含有对需求变化性的支持,才能更有效地动态完成各项构件活动,实现构件演化。现有的构件模型大多属于"静态"模型,不具有支持需求变化的机制,其衍生的各种构件活动也相应是静态的。

针对这些问题,我们基于反射技术[1,2]设计了一个适应于实时应用系统快速高效设计与开发的专有构件模型—反射式实时构件模型。它能开放设计时构件的内部信息,标识某些构件业务逻辑所要求的时间特征,有别于传统的黑盒功能构件。本文着重介绍反射式实时构件模型语义和语法的规约描述。

2 反射式实时构件模型

反射式实时构件模型,即反射式实时构件模型::=〈实时构件反射语义模型,实时构件反射语法模型〉,设计为一个二维构件模型,从构件语义、构件语法两个维度来提供对构件单元功能属性、实时特征及变化性的支持,解决以往构件模型中实时特征表述和约束能力不足的问题,并支持设计时构件可能有的变动和演化等构件活动。其模型层次如图 1 所示。

由图 1 可见,基于反射的实时构件模型分为 3 层。

①基层

*)本文得到国防预研基金项目(10104010201)资助。**黄 靖** 博士研究生,主要主向为软件工程、分布式计算技术;**卢炎生** 教授,博士生导师,主要研究方向为软件工程、分布式计算技术、特种数据库等;**徐丽萍** 副教授,主要研究方向为软件工程、分布式计算技术等。

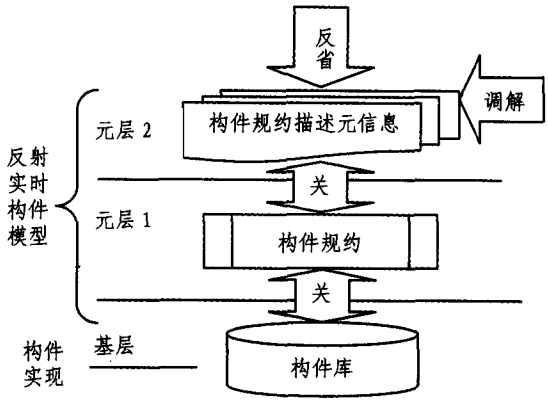

图 1　反射式实时构件模型层次图

模型的基层表示为构件库，它是由平台相关的构件实现及其相应的构件配置文件组成的，其与模型元层 1 的反射关联是通过平台相关的语言映射器来实现的。构件库用于软件开发水平复用时对构件的统一管理与操作，并支持基于刻面的[3]、或基于关键词的已有构件检索方法[4,5]。

②元层 1

元层 1 由构件规约组成，包括构件的语义规约和语法规约，分别描述和记录基层构件库中各个构件实现的语义信息和语法信息，其与模型元层 2 的关联是通过 4.2 节中提取算法(Distill Arithmetic，DA)和生成算法(Generate Arithmetic，GA)来实现的。

③元层 2

元层 2 是对模型元层 1 中构件语义信息、语法信息的进一步归结、抽象的结果，以构件规约描述元信息的方式组织与存储，并提供“反省”与“调解”两类操作接口，供设计人员对模型基层中构件库进行间接按需反射调控，以支持构件库中构件的演化与更新。

3　实时构件的反射语义模型

构件语义以其表达构件对象“做什么”的能力，为软件复用提供语义层的支持，有效地描述构件的有用性和适应性。实时构件的反射语义模型是在以往构件语义理论基础上，形式化描述构件的功能行为语义以及构件方法执行的实时约束特征语义，并抽象实时构件语义信息的元数据，同时能适当地通过反射技术对构件的语义描述(元数据)进行反省(introspection)和调解(intercession)，具体化为查找、匹配、修改等操作，实现构件语义规约层的按需调配，归结于构件的行为反射[6]。

3.1　模型描述

实时构件语义元数据作为实时构件反射语义模型的元信息，由构件的功能行为语义描述和构件的实时特征语义描述抽象组合而成，即{构件功能行为语义描述，构件实时特征语义描述}，是一种元描述。

实时构件反射语义模型是实时构件语义元数据与构件反射语义的复合体，表示为：

实时构件反射语义模型＝实时构件语义元数据×构件反射语义

其中，构件反射语义是指构件元信息对象协议(MOP)的语义描述。另外，语义元层、语义基层分别对应实时构件三层反射模型的元层 2 和元层 1。

3.2　实时构件语义元数据

3.2.1　构件的功能行为语义描述

与普通构件一样，实时构件的执行仍是以方法为基本执行单位。由此我们对实时构件功能行为的语义信息抽象以方法为基本描述单位。依据文[7]中的方法描述，先给出实时构件中不带时间约束特征方法的一般性描述的定义及形式化表示。

定义 1　方法描述。方法描述是对一个构件方法的信息抽象，刻画了该构件方法的特征，包括其名称、参数、返回值、执行前提(pre-condition)、执行结果(post-condition)。方法描述具有以下形式：

method method_name((*Var*: *DomainSort*) *)→*Var*: *RangeSort*

requires pre-condition

ensures post-condition

方法描述 M 在逻辑上分为两部分：

1)方法的签名，记为 *signature*(M)，表示方法的参数及返回值。例如一个整形加法函数 IntAdd 的签名为：*signature*(IntAdd)＝int×int→int。

2)方法的断言，记为 *predicate*(M)，刻画了方法的行为特征，*predicate*(M)$\overset{\text{def}}{=\!=}$ *pre-condition*(M)⇒*post-condition*(M)。

例如，设有方法 member，判断一个实数 a 是否是一维数组 A[1..100]的元素，则 member 的描述为：

method member(*a*:*real*, *A*:*real*[1..100])→*β*:*bool*

requires ¬*empty*(A)

ensures*β*=*a*∈*A*

其中，*signature*(member)＝real×real[1..100]→bool，*predicate*(member)＝ ¬empty(A)⇒β=a∈A。

构件执行是构件中多个相关方法按偏序关系依次执行的活动。故此，构件的功能行为是构件中诸多方法的执行集合体，是软件系统中完成普通任务或实时任务的基础，也是构件复用的语义单位。下面给出不带时间约束特征构件功能行为描述的定义。

定义 2　构件功能行为描述。构件功能行为是构件期望执行状况，其描述由多个方法描述构成，是方法描述的偏序组合描述，定义了构件功能特征以及对外提供的服务。

描述为：⟨method_1，method_2，…，method_n⟩。构件功能行为描述在语义元层就代表了常规构件 *Component*。在这里应用形式化描述语言 CSP[8]，将每个构件方法视作事件(event)，则构件功能行为描述在语义元层就表示为常规构件进程 Component≜μX・(method_1→method_2→…→method_n→X)，其中常规构件字母表为 *αComponent*＝{method_1，method_2，…，method_n}，常规构件进程的一个迹就对应构件的一次执行语义。

3.2.2　构件的实时特征语义描述

实时特征是实时构件中关于构件执行时间约束要求的重要标识，也是区别普通构件的关键所在。实时系统中，每个方法的执行都必须是可预测的，要求每个系统行为均有明确的时间属性，才能保证系统整体的可预测性(predictability)。因而我们将实时特征的语义描述约束到实时系统的基本开发组成单位—构件中，以构件中的方法为构件的实时特征语义描述的基本粒度单位。这样就需要对前面不带时间约束特征的构件方法、构件功能行为进行扩充，得到具有实时约束特征的

语义描述。

定义 3 实时特征描述。实时特征描述是一个有限特征值集合$\{q_1, \cdots, q_n\}$，集合的每一个元素是一个二元组〈name, requirement〉，name 是实时特征要素的名称，如周期、截止期、执行时间等，requirement 是关于 name 的一个断言。

例如：采样周期 SPeriods 等于 10ms，处理时间 PTime 小于 6ms，实时特征描述为{〈SPeriods, Speriods = 10ms〉，〈PTime, PTime<6ms〉}。

有了实时特征描述的定义，接着给出实时构件方法的形式化描述和定义。

定义 4 实时构件方法描述。方法描述是对一个构件方法的信息抽象，刻画了该构件方法的特征，包括名称、参数、返回值、执行前提(pre-condition)、执行结果(post-condition)和方法执行的时间约束特征。有时间约束特征的方法描述具有以下形式：

method method_name((*Var*: *DomainSort*) *) → *Var*: *RangeSort*

requires pre-condition

ensures post-condition

peformance realtime-feature

也就是说，实时构件方法描述是由构件方法描述和实时特征描述组合而成的。可见，实时构件方法语义信息的元数据 M 在逻辑上分为 3 部分：方法的签名、方法的断言、方法的实时约束特征。

接着前面的例子，设有方法 member，判断一个实数 a 是否是一维数组 A[1..100]的元素，判断在 10ms 内完成，则 member 的描述为：

method member(a:*real*, A:*real*[1..100])→β:*bool*

requires ¬*empty*(A)

ensures$\beta = a \in A$

peformance {〈ProcessTime, ProcessTime<10ms〉}

其中，*signature*(member) = real × real[1..100]→bool，*predicate*(member) = ¬empty(A)⇒β=a∈A。

那么，实时构件功能行为描述即为实时构件方法描述的偏序组合描述，其描述形式同构件功能行为描述，主要是将构件功能行为描述中的 $method_n$ 扩展为($method_n$, $performance_n$)，在语义元层也就代表了实时构件 $Component_{realtime}$。值得注意的是，这里并没有对实时构件功能行为给出显式的时间约束特征。考虑到实时构件方法既是实时构件功能行为的功能组成要素，又是实时构件功能行为时间约束特征的基本粒度单位，所以就将其时间特征隐含约束于各相关构件方法的组合中，其值也可以通过实时构件进程 $Component_{realtime}$ 的迹计算得到。应用 TCSP[8]，将每个实时构件方法视作实时事件(real-time)，则实时构件功能行为描述在语义元层就表示为实时构件进程 $Component_{realtime} \overset{\wedge}{=\!=} \mu X \cdot (($method$_1$, performance$_1$)→(method$_2$, performance$_2$)→⋯→(method$_n$, performance$_n$)→X)，其中实时构件字母表为 $\alpha Component_{realtime}$ = {(method$_1$, performance$_1$), (method$_2$, performance$_2$), ⋯, (method$_n$, performance$_n$)}，实时构件进程的一个迹就对应实时构件的一次执行语义。

3.3 构件反射语义

实时构件反射语义模型的元对象协议，实质上是作用于构件反射语义元数据之上的各种操作进程。由于实时构件反射语义模型中元数据主要包括实时构件方法描述以及实时构件功能行为描述，借助形式化描述语言 CSP，我们将此模型中元对象协议(MOP)分为如下两种操作进程，并给出实时构件行为反射语义的概念及其构造方式。

3.3.1 行为反射原子进程

行为反射原子进程是一个 CSP 进程，代表了构件语义反射模型中的一个逻辑操作单元，是实时构件反射行为不需再分的基本组成要素，主要指方法签名反射进程 $P_{signature}$、方法断言反射进程 $P_{predicate}$ 和方法实时特征反射进程 $P_{realtime}$。

例如，对某个待查构件语义描述中一个签名的匹配，记作 $P_{sigmatch}$。设 M_{query} 是待查构件的方法描述，$M_{library}$ 是一个方法实现，$P_{sigmatch} \in P_{signature}$，当且仅当：

$\exists \rho \cdot \rho signature(M_{library}) =$ signature(M_{query})，其中 ρ 是一系列的变量重命名。

3.3.2 行为反射模块进程

行为反射模块进程由若干个行为反射原子进程组成，是构件语义反射模型中的逻辑组合单元，代表了对实时构件方法的一次反射操作，记作 P_{method}。其 *BNF* 表示为：

$\langle P_{method} \rangle ::= [\{P_{signature}\}][\{P_{predicate}\}][\{P_{realtime}\}]$

显然，行为反射原子进程是最简单的行为反射模块进程。

例如，对某个待查构件语义描述中一个签名和断言的匹配，记作 P_{sigpre}，$P_{sigpre} = P_{sigmatch}\ P_{prematch}$。设 M_{query} 是待查构件的方法描述，$M_{library}$ 是一个方法实现，$P_{sigpre} \in P_{method}$，当且仅当：

$(\exists \rho \cdot \rho$signature($M_{library}$) = signature(M_{query})) ∧ (predicate($M_{library}$)⇒predicate(M_{query}))其中 ρ 是一系列的变量重命名。

又如，对某个待查构件语义描述中一个签名的匹配与修改，记作 P_{sigmod}，$P_{sigmod} = P_{sigmatch}\ P_{modify}$。设 M_{query} 是待查构件的方法描述，$M_{library}$ 是一个方法实现，$P_{sigmod} \in P_{method}$，当且仅当：

$(\exists \rho \cdot \rho$signature($M_{library}$) = signature(M_{query})) ∧ ($modify$(signature($M_{library}$)))，其中 ρ 是一系列的变量重命名。

定义 5 行为反射语义。行为反射语义是对实时构件功能行为反射的语义描述，是由一系列行为反射模块进程通过同步并行组装而成，该描述代表了实时构件语义规约反射的逻辑操作模型。

设 P、Q 是行为反射模块进程，均归属于反射模型中的反省或调解操作，则实时构件的行为反射语义构造方式表示如下：

①顺序构造，记作 P;Q，表示先执行反射模块进程 P，P 成功完成后再执行反射模块进程 Q。②外部条件构造，记作 P□Q，表示反射操作由外部环境激励，并决定执行反射模块进程 P 还是反射模块进程 Q。这种情况多表现为构件设计者根据各种需要对构件进行检索或设计调整等。③内部条件构造，记作 P◇Q，表示反射操作由构件自身内部激励，并决定执行反射模块进程 P 还是反射模块进程 Q。该方式大多表现在构件演化中，反射需求的激励由构件自身产生。④中断构造，记作 P▽Q，表示反射模块进程 P 执行中被反射模块进程 Q 中断，等 Q 执行完毕再继续执行 P，主要用于突发操作的构造。⑤基本进程，STOP 或 SKIP，前者表示不执行任何操作的进程(处于反射休止状态)，后者表示反射操作成功后的休止进程，即 SKIP≡√→STOP。⑥前缀，记作 α→P，表示执行过事件 α 后，变为和反射模块进程 P 等价。

于是，实时构件行为反射进程 $\mathscr{R} ::=$ STOP | SKIP | α→P | P;Q | P□Q | P◇Q | P▽Q。其中，实时构件行为反射

语义能递归定义。

4 实时构件的反射语法模型

构件语法通过某种规约语言，从语法结构上描述构件对象的功能需求，为构件复用提供了语法层的支持，有效地保证了构件规范化设计和构件框架代码的自动生成。其研究有很多，如 COM、CORBA 规约等，然而大多不具有对构件实时特征需求的描述，并且一旦用某种规约语言描述生成，就很难与其它规约语言描述的构件互操作，且难以修改，缺乏一定的构件设计灵活性和动态性。实时构件的反射语法模型是在传统功能需求构件的语法规约基础上，融入构件实时约束特征的语法描述，提取、组合成实时构件语法规约的元数据，并引入反射机制，来实现实时构件语法规约的按需调配，即实现实时构件设计中的结构反射[6]。

4.1 模型描述

实时构件语法规约元数据，作为实时构件反射语法模型的元信息，是对实时构件语法结构的描述抽象，是一种元描述，由构件功能需求语法描述和构件实时约束特征语法描述的抽象组合而成，即{构件功能需求语法描述，构件实时约束特征语法描述}。

实时构件反射语法模型是实时构件语法元数据和构件反射语法的复合体，表示为：

实时构件反射语法模型 = 实时构件语法元数据 × 构件反射语法

其中，构件反射语法是指构件元信息对象协议(MOP)的语法表示。语法元层、语法基层分别对应实时构件 3 层反射模型的元层 2 和元层 1。

4.2 实时构件语法元数据

4.2.1 构件功能需求语法描述

功能属性是构件作为一个独立的软件单元所具有的基本特征，包括向外提供的服务和对外所需的要求。其相应的语法规约研究不少，OMG/CORBA3.0 规范就是一个业界公认的开放性标准，然而它缺乏对构件规约描述语言的进一步本体认识，不能从更高、更抽象的语法规约概念层来分析、反射具体的设计时构件规约描述语言，因而不能有效地把握与实现构件设计时的动态性。

本文的研究则是以其规范中的 CCM(CORBA Component Model)作为构件功能需求语法反射模型中的语法基层，进行相应的抽象，得到实际可操作的元数据，组合成构件功能需求的元描述。其目的是提取设计时构件的语法规约元信息，实施反射计算，实现构件的结构反射。模型原理图如图 2 所示。

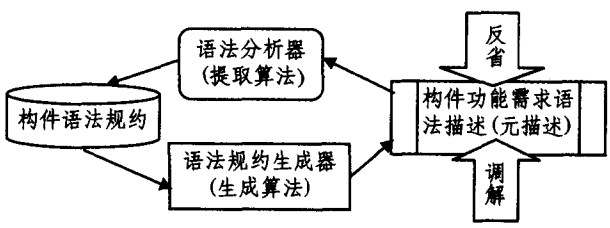

图 2 构件功能需求语法规约反射模型结构示意图

模型中涉及的关键技术有语法元层中元数据的表示形式、构件功能需求语法规约的元信息提取算法以及构件功能需求语法规约的生成算法。以下分别介绍。

1)元数据的表示形式

语法元层中的元数据(结构元数据)用于表示和记录构件语法规约的语法结构信息，以元信息对组(表)的形式给出，是构件实现中反射编译器的基础。

文[10]中给出，CORBA 构件模型(CORBA Component Model，CCM)包含 4 个相互协作的对象模型，即抽象构件模型(Abstract Component Model)、容器模型(Container Model)、包装模型(Packaging Model)和部署模型(Deployment Model)，它们构成了企业计算中完整的服务器端体系结构。反射式实时构件的语法模型研究，是以抽象构件模型的 IDL 文本作为元数据提取的宿主。其中，CCM 构件语法规约中主要包括以下 5 个部分的描述：Home 构件宿主、Facet 刻面、Receptacles 插座、Event Sources/Sinks 事件源/事件槽、Attributes 属性。抽象构件模型图如图 3 所示。

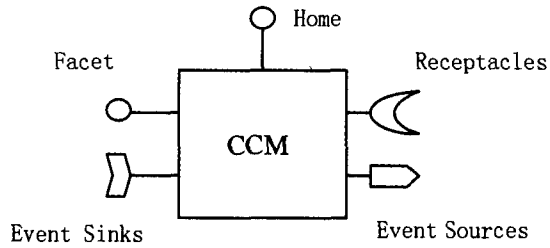

图 3 抽象构件模型图

经分析，CCM 抽象构件模型 IDL 文本中 5 个构件语法单位和一个 component 语法单元所对应的主要结构元数据—元信息对组(表)可分别表示为 表 1、表 2、表 3、表 4a、表 4b、表 4c、表 5 和表 6。

• Home 单元结构元数据见表 1。

表 1 home 元信息对组(表)

语法项	home	〈home_name〉	supports	〈interface_name〉	manages	〈component_type〉	〈explicit_operations〉
对应值	home	自定义	supports/none	自定义	manages	自定义	定义数组

• Facet 单元结构元数据见表 2。

表 2 Facet 元信息对组(表)

语法项	provides	〈interface_type〉	〈name〉
对应值	provides	自定义	自定义

• Receptacles 单元结构元数据见表 3。

表 3 Receptacles 元信息对组(表)

语法项	uses	〈interface_type〉	〈receptacle_name〉
对应值	uses	自定义	自定义

• Event Sources/Sinks 单元结构元数据见表 4(a、b、c)。

表 4a Event Sources publishes 元信息对组(表)

语法项	publishes	〈event_type〉	〈event_name〉
对应值	publishes	自定义	自定义

表 4b Event Sources emits 元信息对组(表)

语法项	emits	〈event_type〉	〈event_name〉
对应值	emits	自定义	自定义

表 4c Event Sinks 元信息对组(表)

语法项	consumes	⟨event_type⟩	⟨event_name⟩
对应值	consumes	自定义	自定义

表 5 Attributes 元信息对组(表)

语法项	attribute	⟨attribute_type⟩	⟨attribute_name⟩
对应值	attribute	自定义	自定义

• Attributes 单元结构元数据见表 5。

• Component 单元结构元数据见表 6。

表 6 Component 元信息对组(表)

语法项	component	⟨component_name⟩	⟨attribute_metadata_list⟩	⟨facet_metadata_list⟩	⟨receptacles_metadata_list⟩	⟨event sources_metadata_list⟩	⟨event sinks_metadata_list⟩
对应值	component	自定义	自定义数组/none	自定义数组/none	自定义数组/none	自定义数组/none	自定义数组/none

结构元数据及各表信息解释如下：

• 结构元数据实质上是一种"型-值"对。

元数据"型"是指以上各表中语法项列集的依次顺序组合，如 Facet 单元结构元数据型为：(provides, ⟨interface_type⟩, ⟨name⟩)；元数据"值"是指各表中对应值列集的依次顺序组合，如 Facet 单元结构元数据值为：(provides, 自定义, 自定义)。表中的"自定义"是指在 IDL 文本中由构件设计者针对语法项，按照命名规则所书写的对应值；"none"表示该项对应值可无；"自定义数组"表示该列上语法项对应的字符串数组，如 Attributes 字符串数组。

• 表中各项列集(型或值)依次顺序组合成一个字符串数组。

以上每个语法单位都对应一个字符串数组，各个数组的长度不一，由各自结构元数据"型"的组成决定。这种记录和存贮构件语法结构信息的字符串数组有利于各种结构反射操作。

• Component 语法单元结构元数据实质上是前面 5 种语法单元一种综合。

这个"综合"的值表示为一个 Component 字符串数组，数组的有些项可能又是由语法项对应的字符串数组构成。

• Attributes 结构元数据"型"中的⟨attribute_type⟩语法项被扩充。

Attributes 语法单元结构元数据"型"中的⟨attribute_type⟩语法项，固定地增加 exec_time, exec_peri, deadline, pri 4 种类型，来分别表示该构件的执行时间、执行周期、截至期、优先级等类型。另外，这 4 个类型被组合成一个组合类型 performance。这些类型用于标识构件的实时特征，其值同样也可实施射操作，适应实时构件的按需访问或修改。

2)提取算法(Distill Arithmetic，简称 DA 算法)

构件功能需求语法规约的元信息提取算法，是对构件自身的功能需求语法规约描述进行分析，抽取该构件语法结构元信息的过程，是一种元信息的抽取算法。其输入为构件语法规约的 IDL 文本，输出则是物理上以字符串数组形式表示和存储的、各主要语法单元所对应的结构元数据—元信息对组(表)。

下面用 CSP 来描述 DA 算法，先给出几个定义。

定义 6 前端词法分析进程 Lex、前端语法分析进程 Parser、前序遍历进程 Scan、元数据生成进程 Get_Meta_Data。其中进程 Scan 用于前序遍历前端进程所得到的语法树中的每个语法节点，进程 Get_Meta_Data 用于生成构件语法结构元数据。

定义 7 前端词法分析进程字母表 αLex、前端语法分析进程字母表 αParser、前序遍历进程字母表 αScan、元数据生成进程字母表 αGet_Meta_Data。

定义 8 DA 算法分为前端和后端两部分，分别对应 Front 进程和 End 进程。

定义 9 前端 Front 进程具有两个通道：DA_{in} 和 $Front_{out}$，通道字母表分别为 αDA_{in} 和 $\alpha Front_{out}$，其中 $\alpha c(DA_{in})=\{(IDL+CIDL)\mid DA_{in}.(IDL+CIDL)\in\alpha DA_{in}\}$；$\alpha c(Front_{out})=\{Out\mid Front_{out}.Out\in\alpha Front_{out}\}$。这里 Out 是一个重要的数据结构，用于记录和存储前端进程处理的结果——语法树。

定义 10 后端 End 进程具有两个通道：End_{in} 和 DA_{out}，通道字母表分别为 αEnd_{in} 和 αDA_{out}。其中 $\alpha c(End_{in})=\{Out\mid End_{in}.Out\in\alpha End_{in}\}$；$\alpha c(DA_{out})=\{meta\_data\mid DA_{out}.meta\_data\in\alpha DA_{out}\}$。这里 meta_data 是一个重要的数据结构，用于记录和存储构件语法结构元数据。

定义 11 DA 算法进程的字母表 $\alpha=\alpha Lex\cup\alpha Parser\cup\alpha Scan\cup\alpha Get\_Meta\_Data\cup\alpha DA_{in}\cup\alpha Front_{out}\cup\alpha End_{in}\cup\alpha DA_{out}$。

DA 算法描述如下。

输入：反射式实时构件语法模型 IDL 文本和 CIDL 文本；

输出：字符串数组形式的构件语法结构元数据 meta_data；

前端进程 $Front\overset{\wedge}{=\!=}\mu X\cdot(DA_{in}?(IDL+CIDL)\rightarrow Lex\rightarrow Parser\rightarrow Frontout!\ Out\rightarrow X)$；

后端进程 $End\overset{\wedge}{=\!=}\mu X\cdot(End_{in}?\ Out\rightarrow Scan\rightarrow Get\_Meta\_Data\rightarrow DA_{out}!\ meta\_data\rightarrow X)$；

算法 DA=Front ; End。

算法性质：如果通道进程 $Front_{out}!\ Out\rightarrow STOP$，即 Front→STOP，那么 DA→STOP，DA 算法失败。否则$\{\surd\}\in\alpha Front$，有 $Out\neq\Phi$，DA→$\surd$。

3)生成算法(Generate Arithmetic，简称 GA 算法)

构件功能需求语法规约的生成算法，是首先对提取算法得到的构件语法结构元信息，与经过反射操作之后的构件语法结构元信息进行差异、变动比较，然后依据比较信息，反向生成新的构件功能需求语法规约描述的过程，是提取算法的逆过程。其输入为经反射操作后得到的构件语法结构元信息，输出则是新的构件语法规约的 IDL 文本。

下面用 CSP 来描述 GA 算法，先给出几个定义。

定义 12 语法树扫描进程 Scan_Ast、语法树设置进程 Set_Ast、构件语法模型文本生成进程 Generate。其中进程 Scan_Ast 用于扫描 DA 算法中得到的初始语法树，进程 Set_Ast 用于重新设置该语法树中每个语法结点的进程，进程 Generate 用于根据重新设置后的语法树来逆向生成构件语法模型文本。

定义 13 语法树扫描进程字母表 αScan_Ast、语法树设

置进程字母表 αSet_Ast、构件语法模型文本生成进程字母表 αGenerate。

定义 14 GA 算法具有两个通道：GA_{in} 和 GA_{out}，通道字母表分别为 αGA_{in} 和 αGA_{out}，其中 $\alpha c(GA_{in})=\{\ meta\_data' \mid GA_{in}.\ meta\_data' \in \alpha GA_{in}\}$。这里的 meta_data′是在反射操作中经过差异、变动比较后的构件语法结构元数据；$\alpha c(GA_{out})=\{\ (IDL+CIDL) \mid GA_{out}.(IDL+CIDL) \in \alpha GA_{out}\}$。

定义 15 GA 算法进程的字母表 $\alpha=\alpha Scan\_Ast \cup \alpha Set\_Ast \cup \alpha Generate \cup \alpha GA_{in} \cup \alpha GA_{out}$。

GA 算法描述如下。

输入：在反射操作中经过差异、变动比较后的构件语法结构元数据 meta_data′；

输出：反射后的实时构件语法模型 IDL 文本和 CIDL 文本；

$$GA \overset{\wedge}{=\!=} \mu X \cdot (GA_{in}?\ meta\_data' \rightarrow Scan\_Ast \rightarrow Set\_Ast \rightarrow Generate \rightarrow GA_{out}!\ (IDL+CIDL) \rightarrow X)。$$

值得注意的是，DA 算法中参数 out 是一个全局变量，能被 DA 算法与 GA 算法读取。另外，将普通功能性构件语法结构元数据 meta_data、meta_data′统一表示为 M_{fun}；将实时性构件语法结构元数据统一表示为 M_{rt}。

4.2.2 *构件实时特征语法描述*

在 CCM 的规范中，是没有关于构件实时特征语法描述的，它仅对构件的功能属性提供了语法支持。为描述和刻画包含时间约束特征的构件—实时构件，我们在 CCM 规范的基础上进行相应扩充，提供构件实时特征描述的语法机制。由于构件是实时应用系统中一个执行整体，故以构件自身为实时特征的约束单位。其中构件实时特征的描述内容定义为构件执行时间、构件执行周期、构件截止期和构件优先级（根据需要，还可以进一步扩充），并分别对应于扩充之后的 CCM 规范中 Attributes 语法单位的 exec_time、exec_peri、deadline 和 pri4 种固定属性类型，即构件的时间约束特征元数据形式化表示为：performance::=〈exec_time, exec_peri, deadline, pri〉。另外，构件实时特征语法描述的提取算法、生成算法类似于上述，这里不再重复。从实时构件语法规约中能抽离出一个个的元信息对组 Mrt::=〈Mfun, performance〉，其标识的设置也是为了支持实时构件的运行时反射机制。例如，构件运行时对其按需装载或卸载，也便于选用满足要求的实时构件用于实时应用系统的开发。同时，也能在构件设计时，从实时构件语法规约层对构件实时特征标识进行某些反射操作，如查找、修改元信息对组 Mrt。以上这些都为构件的反射提供了设施。

注意，实时构件语法规约中实时特征绑定的单元是构件自身，而非构件中的功能接口单元，因而该构件实现中不同的功能接口调用执行顺序，同样会对应不同的构件实时特征值，因而有着不同的设计方案。这是不同于实时构件语义规约的地方。

4.3 构件反射语法

现有的构件语法通常是用某种规约语言（如 IDL、CDL），从语法结构的角度来定义设计时构件的功能目标的。它着重描述构件的功能接口，并不规定构件内部的具体实现行为，是一种静态的语法规约描述机制。一旦构件设计时规约描述确定，直至构件生成不能更改，因而对构件的适应性支持不够。同时，该构件的语法结构中，还缺乏对构件功能运行时实时特征作出显式的语法标识，不利于实时构件的区分、获取与使用。

基于反射技术，我们从以下两个方面开放构件语法元数据（4.2 节已给出元数据的表示），并提供其反省和调解操作的元接口（meta_interface），实现实时构件的结构反射，增强构件模型的灵活性、适应性和可操作性，确保基于实时构件的实时应用系统设计与开发。

1）构件语法结构自身的反射

封装并实现同一个功能需求的构件，可能会对应着不同的构件语法结构。这些设计时构件的语法结构，在经过不同的平台相关的构件编译器向某种高级语言（如 Java，C++）语法映射后，会在不同的目标机上生成有不同运行时间等实时特征的构件代码框架，最终影响着具有不同实时需求的构件使用者，对实现同一个功能需求、选取有着不同构件语法结构的构件。即根据用户的功能需求和实时要求，查询构件的语法结构，进行可能有的适应性修改（不满足实时要求的情况下），获取最终满足要求的实时构件。

另外需要指出的是，不同的构件语法结构在不同的目标机上对应的有着不同时间特征的代码框架是个经验值，是由构件的设计和开发者来收集和积累的。

2）构件语法规约中对实时特征描述的反射

基于反射的实时构件模型中，提供了标识构件功能运行时时间特征的语法机制，这些标识支持反省和调解操作，能用于查询和修改，适应实时应用系统设计和开发中对实时构件选用的变化性。如前所述，封装描述同一功能需求的不同构件语法结构，会对应着不同的实时特征。这些元数据可用于在实时构件设计时，随构件语法结构的变化，进行检索和适应性修改。修改得到合乎要求的实时特征组合，形成新的设计时构件。

对于上述讨论的实时构件模型中结构反射的两种情形，图 4 给出了其反射操作示意图。

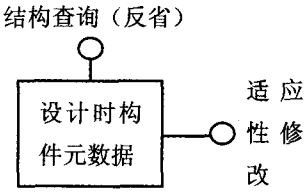

图 4 实时构件模型结构反射操作示意图

最后，用 IDL 描述实时构件模型 4 类主要的结构反射操作，其它的相关操作均可由这些操作派生得到。

```
module ReflectiveOPerate{
  ……
  interface Boolean visit_meta_data ( in meta_data_type meta_data)
        raises(); //访问构件功能特征语法结构元数据
  interface Boolean update_meta_data ( inout meta_databs—type meta_data)
        raises(); //修改构件功能特征语法结构元数据
  interface Boolean visit_meta_databs—performance ( in meta_data_performance_type metabs—data_performance)
        raises(); //访问构件实时特征语法结构元数据
  interface Boolean update_meta_data_performance(inout meta_data_performance_typemeta_data_performance)
        raises(); //修改构件实时特征语法结构元数据
```

……

}

5 运行时实时构件的反射流程

反射实时构件在运行时，可能由于构件运行时环境的变化，或用户某些特定的要求，需要更换不同的构件实现，来组装新的实时应用软件系统。例如，实时应用系统运行时，可能要求替换效率更高的实时调度构件，或选取具有完成时间约束特征不同的普通功能构件，这时需要依照反射实时构件模型，对构件库中的实时构件进行相应的按需调控，获取新的满足变化性要求的实时应用系统。其过程依次顺序对应反射实时构件模型中两种不同的反射流——UP 流和 DOWN 流。

5.1 UP 流

UP 流即从反射实时构件模型的基层，经元层 1 上升到元层 2，完成一个上升流过程。这个过程首先从实时应用系统中剥离出需要替换的实时构件，在构件库中查找其对应的配置属性文件，获得其相关的构件规约，进行语义的抽象和语法的提取，最后得到该实时构件的规约描述元信息，并进行分析。设 UP 流对应进程 $Flow_{up}$，其字母表为 $\alpha Flow_{up}$。

5.2 DOWN 流

DOWN 流即从反射实时构件模型的元层 2，经元层 1 下降到基层，完成一个下降流过程。这个过程首先对 UP 流得到的最终元信息分析结果，进行按需的“反省”或“调解”操作，获取新的实时构件规约描述元信息，并根据生成算法形成新的构件规约，然后借用用户平台相关的编译器生成新的满足要求的构件实现，最后装配到原有的实时应用系统中去。同时组织该构件实现和构件配置文件入库，以便今后对类似构件的使用。设 DOWN 流对应进程 $Flow_{down}$，其字母表为 $\alpha Flow_{down}$。

这两种方向截然不同的反射流在基于反射实时构件的实时应用系统中，正好形成了一个完整的反射流，实现了实时构件的按需变动和演化，增强了实时应用软件系统的适应性和健壮性。那么，运行时实时构件的反射流程可表示为 $ReflectiveFlow \overset{\Lambda}{=\!=} \mu X \cdot (Flow_{up} \rightarrow Flow_{down} \rightarrow X)$。

结束语 本文以现有的构件模型为研究基点，应用反射技术，针对实时应用系统的开发，提出了一类新的构件模型——反射式实时构件模型。它由反射式实时构件语义模型和反射式实时构件语法模型组成。该模型在规约了构件应有的功能需求特征的基础上，有效地标识了构件的时间约束特征，使得它与传统的功能性构件区开来，能被系统开发者更好地选用。同时，该模型结合反射技术，还能根据用户需求的变化，对实时构件进行动态的设计时修改，以便更准确地保障实时应用系统的构建与开发。这样，增强了构件设计的灵活性，达到了构件更好实现的效果。

反射式实时构件模型既是一个构件理论模型，也是一个工程模型。本文着重给出了它的规约描述机理。在接下来的工作中，我们将进一步完善该实时构件模型的反射基础设施，尤其是语法模型的构件实现所需的反射实时构件运行时环境。

参 考 文 献

1 Smith B C. Reflection and semantics in a procedural language: [LCS Technical ReportTR-272]. Cambridge, MA: M I T, 1982

2 Smith B C. Reflection and semantics in Lisp. In: Proceeding of the 1984 ACM Principles of Programming Language Conference, ACM, December 1984. 23～25

3 王渊峰，张涌，任洪敏. 基于刻面描述的构件检索. 软件学报，2002，13(8)：1546～1552

4 Damiani E, Fugini M G, Bellettini C. A hierarchy-aware approach to faceted classification of objected-oriented components. ACM Transactions on Software Engineering and Methodology, 1999, 8(3): 215～262

5 Henninger S. Supporting the process of satisfying information needs with reusable software libraries: an empirical study. In: Samadzadeh M H, Mansour K Z, eds. Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability. Seattle, WA: ACM Press, 1995. 267～270

6 Cazzola W. Evaluation of Object-Oriented Reflective Model. In: Proceeding of ECOOP Workshop on Reflective Object-Oriented programming and Systems(EWROOPS'98), Brussels, Belgium, July 1998

7 Zaremski A M, Wing J M. Specification Matching of Software Components. ACM Trans Softw Eng Method, 1997, 6(4): 333～369

8 Hoare C A R. Communicating Sequential Processing. Prentice Hall, 1985

9 Schneider S. An Operational Semantics for Timed CSP. Information and Computation Programming Research Group, Oxford University Computing Laboratory

10 Object Management Group. CORBA Components, version 3.0. OMG, Inc. June 2002

(上接第 227 页)

$Exp(\Sigma_{31}) = (P_1P_2(P_3)^*(P_4+P_5))^\alpha \parallel (P'_1P_2(P_3)^*(P_4+P_5))^\alpha$，其中 $P'_1, P_1 (i=2,3,4,5)$ 分别如图 5 所示。

容易求得 Σ_{32} 三个基本进程段就是图 2 中的 P_3，P_4 和 P_5 并且可以求得 $Exp(\Sigma_{32}) = P_3^*(P_4+P_5)$。故：

$$\begin{aligned} Exp(\Sigma_3) &= Exp(\Sigma_{31}) \parallel Exp(\Sigma_{32}) \\ &= (P_1P_2(P_3)^*(P_4P_5))^\alpha \parallel (P'_1P_2(P_3)^*(P_4P_5))^\alpha \parallel P_3^*(P_4+P_5) \end{aligned}$$

结束语 由于 S-网具有很好的结构特征，其进程行为易于分析和描述。本文分析了结构简单的 S-网的进程行为，给出了各类 S-网的进程表达式的求取方法。我们分析 S-网的进程行为是为结构复杂 Petri 网的进程行为描述服务的。通过适当的分解方法可以将一个结构复杂的 Petri 网分解成结构简单的子网(如，S-网)，分析分解过程中满足的进程语义，通过分解后的子网来分析原系统的进程行为。或者将一些结构简单的子网合成一个结构复杂的 Petri 网，分析合成过程中的进程语义，通过结构简单的子网分析合成后的复杂网系统的进程行为。通过分解(合成)分析结构复杂 Petri 网的进程行为，我们已经得到了相关的方法和结果，将会在其他的文章中给出。

参 考 文 献

1 袁崇义. Petri 网原理. 北京：电子工业出版社，1998

2 Peterson J. Petri net theory and the modeling of systems. 吴哲辉译. 徐州：中国矿业大学出版社，1989

3 Murata T. Petri Nets: Properties, Analysis and Applications. Proceedings of The IEEE, 1989, 77(4)

4 Gltz U, Reisig W. Processes of place/transition net, LNCS, New York: Springer-Verlag, 1983, 154: 264～277

5 Lu Ruqian. P/R nets and P/R processes. Science in China (Series E), 1992, 35(1): 21～31

6 曾庆田，吴哲辉. Petri 网的进程网系统. 计算机学报，2002，25(12)：1308～1315

7 Wu Zhehui. Process expression of bounded Petri net, Science in China (Series E), 1996, 39(1): 37～49

8 吴哲辉，王培良，赵茂先. 无界公平 Petri 网的进程表达式. 计算机学报，2000，23(4)：337～344

9 曾庆田，吴哲辉. 无界 Petri 网的进程表达式，计算机学报，2003，26(12)：1629～1636

10 曾庆田，吴哲辉. 类 S－图的语言性质分析. 计算机科学，2002，29(5)：120～123

11 段华，曾庆田. S-网的活性分析. 小型微型计算机系统，2004，25(11)：1975～1978