反射式集成框架的规约描述方法研究*)

黄 靖[1,2] 卢炎生[1] 徐丽萍[1]

(华中科技大学计算机科学与技术学院 武汉 430074)[1] (华中科技大学管理学院 武汉 430074)[2]

摘 要 反射式集成框架的规约描述方法,主要研究在分布式实时应用领域基于构件的软件开发模式中集成框架的形式化规约描述问题。这种描述方法通过引入反射技术,除了描述集成框架中组成要素的业务逻辑之外,还对各要素的实时性能约束、运行时状态的变化以及可能具有的需求变更等特征进行形式化规约,从而支持软件在需求分析阶段的演化进程,并以指导与实现实时应用软件开发时业务逻辑与系统非功能性特征的关注分离。

关键词 集成框架,反射,实时应用

The Research on the Specification of Reflective Integration Architecture

HUANG Jing[1,2] LU Yan-Sheng[1] XU Li-Ping[1]

(School of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan 430074)[1]

(School of Management, Huazhong University of Science & Technology, Wuhan 430074)[2]

Abstract The research on the specification of reflective integration architecture is mainly involved in the question of the formalization of the integration architecture of the component based software developing pattern in the distributed real-time application domain. This specification introducs reflection technology to formally describe the business logic of various elements in the integration architecture, as well as these elements' characteristics including timing constraints, changes of state in runtime and possible changes of requirements. Therefore the specification can support software evolution in the requirements analyzing phase, and guide to realize separation of concerns on business logic and non-functional features in developing the real-time application software.

Keywords Integration architecture, Reflection, Real-time application

1 引言

基于构件的软件开发(Component-based Software Development, CBSD)逐渐成为当前主流的软件开发模式,特别是在分布式应用中。然而仅仅依赖于一组松散耦合的软构件,并不足以系统化地构造复杂的软件系统,还需要领域相关的构件组织规则,来指导最终集成系统的生成,这就是软件的集成框架(Integration Architecture)。在基于反射技术的软件集成方法[1]中,集成框架的规约是软件需求分析阶段的产物。本文的研究主要针对分布式实时应用领域。引入了新的软件设计开发范型——反射(Reflection)[2]。故在对集成框架的形式化规约中,除了应当描述集成框架各组成要素的业务逻辑之外,还需要表达这些单元的实时性能约束、运行时状态的变化以及可能具有的需求变更等特征,这样才能更为全面地指导分布式实时应用软件的生成。本文将使用Z语言作为一种软件需求分析阶段规格说明书的描述机制,来形式化规约[3]这种基于反射技术的新的软件开发模式。

2 相关工作

文[4]指出Z语言是一种形式化规格说明语言,特别适合描述软件体系结构(Software Architecture,SA)。但以往Z语言主要针对软件中功能性业务逻辑的规约描述,较少关注软件开发过程中的变化性需求、非功能属性以及软件体系结构的动态性等特征。软件工程师们往往在软件系统的实现时,才真正关注这些功能性需求以外的特征,这显然是不对的。其实在软件的需求分析时,就应当很好地关注它们,并相应地在软件需求规格说明中进行正确规约描述,这样有利于软件开发人员从整体上对系统的把握,避免不必要的软件废品。例如,软件工程师能在软件系统成型之前,根据形式化需求规格说明从软件语义上分析和验证软件的可行性,以及当系统内某个部分因为升级或演化时,系统是否仍能保持一致性等特征。在基于构件软件开发的今天,这显得格外重要。

文[5,6]在这个方面弥补了Z语言的不足,它们在Z语言本身的基础上,描述了非功能需求(Non-functional Requirements, NFRs)的表述机制,规约了动态体系结构的描述方法,但未深究软件体系结构的动态性与需求特征的变化性之间的内在原因,以及其与需求特征变化性的表达机制和实现方法的内在关联,同时也未对实时应用领域具体专注。本文认为,软件体系结构的动态性是由于需求特征的变化而造成软件演化的结果,需求特征变化性的表达机制和实现方法是有效规约软件体系结构动态性的保障。如本文研究中引入的关键技术——反射技术[2],从能设计理念和开发方法上统一规划动态性软件体系结构的分析与实现。本文将在文[5,6]研究的基础上,使用Z语言形式化规约基于反射技术的分布

*)本文得到国防预研基金项目(10104010201)资助。**黄 靖** 博士后,主要研究方向为软件工程、分布式计算技术、中间件技术;**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、中间件技术等;**徐丽萍** 博士,副教授,主要研究方向为软件工程、分布式计算技术、中间件技术等。

式实时应用软件开发方法，描述这种新的开发模式的各种需求、状态和特征，特别是通过软件合成或软件集成[7]来构建分布式实时应用软件的复用型框架(Reusable Architecture)的规约方法，用于指导其领域中软件的设计与实现。

3 集成框架模型

集成框架模型(Integration Architecture Model，IAM)是对课题研究中反射式集成框架[1](Reflective Integration Architecture，RIA)的一个概要性认识，涵盖基于反射技术的软件集成(软件合成)方法中集成框架的各个基本组成要素。其目的是能通过集成框架模型，在分布式实时应用软件开发的需求分析阶段，形式化规约软件集成活动中的功能性业务逻辑、实时属性及其可能具有的需求变化等特征，并能根据在软件设计与实现时关联这些需求特征的关键技术——反射技术，模拟集成系统在运行时的调配状态，最终验证软件的正确性和一致性。由此，在软件的需求分析阶段支持软件的演化活动。

对文[1]中基于反射技术的软件集成方法分析可知，集成框架模型 IAM 由构件项 ComponentItem、连接件项 ConnectorItem 和接口项 InterfaceItem 三维元素构成，其中构件项维包含了构件 Component、构件的时间约束特征 Timing、实时构件 Real-timeComponent、反射式构件 ReflectiveComponent、构件的反射式时间约束特征 ReflectiveTiming 和反射式实时构件 ReflectiveReal-timeComponent 等六个单元；连接件项维包含集成中间件 Middleware、集成中间件的实时性能约束特征 Real-timeConstraint、集成中间件中反射式实时性能约束特征 ReflectiveReal-timeConstraint、实时集成中间件 Real-timeMiddleware、反射式中间件 ReflectiveMiddleware 和反射式实时集成中间件 ReflectiveReal-timeMiddleware 等六个单元；接口项维则由功能性接口 FunctionalInterface、接口的时间约束特征 InterfaceTiming 和带时间约束特征的接口 TimingInterface 三个单元组成。因而，集成框架模型 IAM 可描述如下：

IAM::=〈 ComponentItem, ConnectorItem, InterfaceItem 〉；

ComponentItem::=〈 Component, Timing, Real-timeComponent, ReflectiveComponent, ReflectiveTiming, ReflectiveReal-timeComponent 〉；

ConnectorItem::=〈 Middleware, Real-timeConstraint, ReflectiveReal-timeConstraint, Real-timeMiddleware, ReflectiveMiddleware, ReflectiveReal-timeMiddleware 〉；

InterfaceItem::=〈 FunctionalInterface, InterfaceTiming, TimingInterface 〉。

4 反射式集成框架的形式化规约

反射式集成框架 RIA 形式化规约的对象，主要包括集成框架模型 IAM 中的构件、连接件、接口、实时属性、反射操作行为、需求规约变化状态、以及框架操作行为等七类。这里使用 Z 语言中的状态模式(state schema)和操作模式(operation schema)分别规约描述它们。

(1)实时属性

实时属性作为分布式实时应用软件的需求特征之一，在集成框架模型 IAM 中关联的元素有构件项、连接件项和接口项，其在 RIA 中规约的属性项分别有起始时间 start、终止时间 terminate、截止期 deadline、运行周期 period、最差执行时间 WCET 和全局优先级 priority 等。用垂直规约模式(schema)描述如下：

RT_Attribute

start : R
terminate : R
deadline : R
period : R
WCET : R
priority : R

---

start < terminate ≤ deadline
terminate < WCET

其对应的水平规约模式(schema)为：RT_Attribute ≙ [start, terminate, deadline, period, WCET, priority : R | (start<terminate≤deadline) ∧ (terminate<WCET)]。

(2)构件

构件对象封装了软件功能性业务逻辑，提供与外界协作的端口，包含着其与实时属性的关联，以及按需调配的操作语义。用垂直规约模式(schema)描述如下：

RIA_Component

component_id : IDENTIFIER
component_context : ℙCSP
component_semantic : ℙCSP
port_to_RT : Components ⇸ RT_Attributes
port_to_interface : Components ⤖ Interfaces
port_to_reflection : Components ⇸ Reflections

∃ c : Components | c ∈ **dom**(port_to_RT) ·
(port_to_RT(c)(start) < port_to_RT(c)(terminate) ≤ port_to_RT(c)(deadline) =
∧ (port_to_RT(c)(terminate) < port_to_RT(c)(WCET))
∀ c : Components | c ∈ **dom**(port_to_interface) ·
port_to_interface(c)(dir) ∈ {in, out} ∧ port_to_interface(c)(mode) ∈ {provide, use}
∃ c : Components | c ∈ **dom**(port_to_reflection) ·
port_to_reflection(c)(mode) ∈ {introspection, intercession}

其对应的水平规约模式(schema)为：

PIA_Component≙[component_id:IDENTIFIER;
component_context,component_semantic:ℙCSP;
port_to_RT:Components⇸RT_Attributes;
port_to_interfsace:Components⇸Interfaces;
port_to_reflection:Component s⇸Reflections|
(∃c:Components|c∈**dom**(port_to_RT)·
(port_to_RT(c)(start)<port_to_RT(c)(terminate)
≤port_to_RT(c)(deadline))
∧(port_to_RT(c)(terminate)<port_to_RT(c)(WCET)))
∧(∀c:Components|c∈**dom**(port_to_interface)·
port_to_interface(c)(dir)∈{in,out}
∧port_to_interface(c)(mode)∈{provide,use})
∧(∃c:Components|c∈**dom**(port_to_refledction)·
port_to_reflection(c)(mode)∈{introspection, intercession})
]。

(3)连接件

连接件对象中包含着自身的行为语义，以及其与外界协作的端口、实时属性和反射操作的关联。为节省篇幅，接下来的规约对象仅以水平规约模式描述。连接件的水平规约模式(schema)描述如下：

RIA_Connector$\overset{\wedge}{=}$[connector_name:INDENTIFIER;
connector_semantic:ℙTCSP;
connector_to_RT:Connectors⇸RT_Attributes;
connector_to_interface:Connectors⇸Interfaces;
connector_to_reflection:Connectors⇸Reflections|
(∃m:Connectors|m∈**dom**(connector_to_RT)·
(connector_to_RT(m)(start)<connector_to_RT(m)(terminate))
∧(connector_to_RT(m)(terminate)≤connector_to_RT(m)(deadline))
∧(connector_to_RT(m)(terminate)<connector_to_RT(m)(WCET))
∧(∀m:Connectors|m∈**dom**(connector_to_interface)·
connector_to_interface(m)(role)∈{sink, source}
∧connector_to_interface(m)(mode)∈({publishs, consumes})
∧(∃m:Connectors|m∈**dom**(connector_to_reflection)·
connector_to_reflection(m)(mode)∈({introspection, intercession})
]。

(4)接口

接口是构件和连接件与外界协作的基础设施，其水平规约模式(schema)描述如下：

RIA_Interface$\overset{\wedge}{=}$[nterface_id:INDENTIFIER;
interface_semantic:ℙMethods;
interface_to_connector:Interfaces⇸Connectors;
interface_to_port:Interfaces⇸Components;
interface_to_RT:Interfaces⇸RT_Attributes|
(∃i:Interfaces|i∈**dom**(interface_to_RT)·
(interface_to_RT(i)(start)<interface_to_RT(i)(terminate))
∧(interface_to_RT(i)(terminate)≤interface_to_RT(i)(deadline))
∧(interface_to_RT(i)(terminate)<interface_to_RT(i)(WCET))
∧(∀i:Interfaces|i∈**dom**(interface_to_port)·
interface_to_port(i)(dir)∈{in, out}
∧interface_to_port(i)(mode)∈{provide, use})
∧(∀i:Interfaces|i∈**dom**(interface_to_connector)·
interface_to_connector(i)(role)∈{sink, source}
∧interface_to_connector(i)(mode)∈{publishs, consumes})
]。

(5)反射操作

反射操作是实现系统动态性和按需调配系统各组成部分的一种方法与措施，其水平规约模式(schema)描述如下：

RIA_Reflection$\overset{\wedge}{=}$[mode:Reflection_modes;
object:RIA_Objects;
reflection_to_port:Reflections→Components;
reflection_to_connector:Reflections→Connectors;
reflection_to_RT:Reflections→RT_Attributes;
Check;
Adjust|
(mode∈{*introspection*, *intercession*})
∧(object∈{Components, Connectors, RT_Attributes})
]。

其中Check操作模式(operation schema)的水平规约模式(schema)描述如下：

Check$\overset{\wedge}{=}$[mode:Reflection_modes;
object?:RIA_Objects;
r!:Response|
(mode=introspection)
∧(**let** object=RT_Attribute)·(r=success)
∧(**let** object=RIA_Component)·(r=success)
∧(**let** object=RIA_Connector)·(r=success)
]。

Adjust操作模式(operation schema)的水平规约模式(schema)描述如下：

Adjust$\overset{\wedge}{=}$[mode:Reflection_modes;
object?:RIA_Objects;
r!:Response|
(mode=intercession)
∧(**let** object=RT_Attribute)·(r=success)
∧(**let** object=RIA_Component)·(r=success)
∧(**let** object=RIA_Connector)·(r=success)
]。

(6)需求规约变化状态

需求规约变化状态RIA_State描述了当需求发生变动时，由反射操作行为引发规约对象变化的状态更替，其由ΞS和ΔS两种模式(schema)构成，即RIA_State$\overset{\wedge}{=}$[ΞS;ΔS]。其中：①ΞS$\overset{\wedge}{=}$S∧S′,S=S′；②ΔS$\overset{\wedge}{=}$S∧S′,S≠S′。这里的S可为上述实时属性、构件或连接件中任一模式，S′为其需求发生变化后的模式。

(7)框架操作行为

框架操作行为RIA_Operation规约了对反射式集成框架RIA中组成元素的各种行为语义，例如对构件规约、实时属性规约的初始化等操作。

由此，反射式集成框架RIA可整体规约如下：

RIA $\overset{\wedge}{=}$ RT_Attribute⨾RIA_Component⨾RIA_Connector⨾RIA_Interface⨾RIA_Reflection⨾RIA_State⨾RIA_Operation。

5 实例与验证

5.1 实例描述

课题中有一分布式实时信号采集系统，该系统的结构采用客户/服务器模式，即信号采集端负责采集信号，并实时地向信号处理端发送采集信号；信号处理端负责处理与显示采集信号，并适时地向信号采集端发送指令。该系统除了本身具有分布式和实时性的特点外，还要求具有一定的开放性，即系统在运行时能根据需要，动态地添加或删除其中的某些信号采集端，且仍能保持系统的正确性和一致性。

5.2 需求规约与验证过程

系统的设计与开发，是选用基于构件的软件开发模式，因而在其需求分析阶段，相应地对系统实施构件化规约，即将系统分为信号采集、信号处理和信号传输三类构件单元来进行需求规约。这里应用第4节中描述的反射式集成框架RIA进行系统的框架设计与规约，并以此支持系统的动态性与开放性。下面给出实例系统的规约及验证过程。

(1) 模式的具体化

RIA使用Z语言规约了集成框架模型IAM中七类对象的模式(schema)，在实例系统中将针对系统的结构与组成要素，分别具体化这些规约描述。这里以实时属性、第$i(i\in\mathbb{N})$个信号采集端构件、和其与信号处理构件间的连接件为例描述规约模式的具体化过程。

specifying_RT_Attribute$\overset{\wedge}{=}$RT_Attribute[start?:= S,

terminate? := F, deadline? := D, period? := T, WCET? := wect, priority? := P];

specifying_RIA_Component $\hat{=}$ RIA_Component[component_id? := C_i, component_semantic? := P_{in}^{Ci}; P_{com}^{Ci}; P_{out}^{Ci}, port_to_RT (component_id?) := RT_Attribute [start?, terminate?, deadline?, period?, WCET?, priority?], port_to_interface (component_id?) := RIA_Interface$^{[interface_id?]}$, port_to_reflection (component_id?) := RIA_Reflection[mode?]];

specifying_RIA_Connector $\hat{=}$ RIA_Connector [connector_name? := C_i2S, connector_semantic? := P_{in}^{Ci2S}; P_{com}^{Ci2S}; P_{out}^{Ci2S}, connector_to_RT (connector_name?) := RT_Attribute [start?, terminate?, deadline?, period?, WCET?, priority?], connector_to_interface (connector_name?) := RIA_Interface [interface_id?], connector_to_reflection(connector_name?) := RIA_Reflection[mode?]]。

(2) 环境的定义

使用 RIA 中的 RIA_Define_Context 操作模式(operation schema)定义实例系统的运行时环境。

RIA_Define_Context$\hat{=}$[component? : Components; interface? : Interfaces; connector? : Connectors; rt_attribute? : RT_Attributes; machine? : FMachine; language? : F Language; context! : þTCSP | context!]。

(3)规约对象的初始化与激活

使用 RIA 中的 RIA_Instance 和 RIA_Active 初始化并激活实例系统中的各个规约对象。

RIA_Instance$\hat{=}$[object? : RIA_Objects; machine? : FMachine; language? : FLanguage; r! : Response | (object ∈{Components, Interfaces, Connectors}) ∧ (objec∉ InstObjects) ∧ (InstObjects′= InstObjects ∪ {object?}) ∧ (object? (context) = (machine?, language?)) ∧ (r! = {success, failure})];

RIA_Active$\hat{=}$[object? : RIA_Objects; r! : Response | (object∈{Components, Interfaces, Connectors}) ∧ (object∉ ActiveObjects) ∧ (ActiveObjects′= ActiveObjects ∪ {object?}) ∧ (r! = {success, failure})]。

(4) 验证

设实例系统由 $i-1$ 个信号采集构件和一个信号处理构件组成,在需求规约中分别命名为 C_1、C_2……C_{i-1} 和 S,现有一个信号采集构件 C_i 要求在系统运行时动态加入,那么就需要在语义上分析与验证已有系统与新加入构件间的适配性及兼容性,以保证合成或集成后的系统仍能维持正确性与一致性,否则就需要重新反射调配已有系统或新加构件的组成要素,以致不使系统崩溃。另设已有系统表示为 A_{i-1},C_i 与其关联的连接件为 C_i2S,集成后系统为 A_i。这里将按照如下规约的过程,并借用形式化工具 *Zans*[8] 来验证实例系统的动态性。

active C_i;
assign RT_Attribute to C_i;
active C_i2S;
assign RT_Attribute to C_i2S;
link C_i to C_i2S;
link C_i2S to A_{i-1};

其中,"**link** C_i to C_i2S"和"**link** C_i2S to A_{i-1}"根据实时构件合成的语义分析方法,进行适配性验证和可连接性判断。最后执行 Zans 中的命令 execute 可得,当且仅当 ∃c : Components, m : Connectors, s : Components | (c≠s) ∧ (c∈ InstComponents ∧ c ∈ ActiveComponents) ∧ (s ∈ InstComponents ∧ s ∈ ActiveComponents) ∧ (m ∈ InstConnectors ∧ s∈ ActiveConnectors) • ((port_to_interface(c)=connector_to_interface(m)) ∧ (((port_to_interface(c)(dir)=in) ∧ (connector_to_interface(m)(role)=sink) ∧ (port_to_interface(c)(mode)=use) ∧ (connector_to_interface(m)(mode)=consumes)) ∨ ((port_to_interface(c)(dir)=out) ∧ (connector_to_interface(m)(role)=source) ∧ (port_to_interface(c)(mode)=provide) ∧ (connector_to_interface(m)(mode)=publishs)))) ∧ ((port_to_interface(s)=connector_to_interface(m)) ∧ (((port_to_interface(s)(dir)=in) ∧ (connector_to_interface(m)(role)=sink) ∧ (port_to_interface(s)(mode)=use) ∧ (connector_to_interface(m)(mode)=consumes)) ∨ ((port_to_interface(s)(dir)=out) ∧ (connector_to_interface(m)(role)=source) ∧ (port_to_interface(s)(mode)=provide) ∧ (connector_to_interface(m)(mode)=publishs)))))时,有 $A_i \rightarrow \surd$,即 $A_i \hat{=} A_{i-1} {}^{0}_{9} C_i {}^{0}_{9} C_i$2S;否则 $A_i \rightarrow$ SKIP,即信号采集构件添加失败。

小结 集成框架是基于反射技术的软件合成与软件集成中,软件设计和开发的指导原则。为增强实现系统的完整性、灵活性和动态性,有必要在系统开发之初,即需求分析阶段,就尽早地重视系统中功能性业务逻辑与非功能属性之间的关注分离[9]。为此,本文在相关研究的基础上,针对分布式实时应用软件的开发,提出了一类反射式集成框架(Reflective Integration Architecture),其规约描述机制既能准确地描述系统需求,又能呈现出系统实现后所具有的非功能性特征,同时可以借助形式化工具,从语义上对需求规约进行系统动态性的验证,进而在需求分析阶段支持软件的演化,最终可以指导软件开发时业务逻辑与系统非功能性特征的真正关注分离。

在软件的设计与开发活动中,反射式集成框架同样也是一类可复用单元。

参 考 文 献

1 黄靖,卢炎生,徐丽萍. 基于反射技术的有性能约束特征的软件集成方法研究. 小型微型计算机系统, 2005, 26(7): 1264～1269
2 Smith B C. Reflection and semantics in Lisp. In: Proceeding of the 1984 ACM Principles of Programming Language Conference, ACM, 1984. 23～25
3 Wing J M. A Specifier's Introduction to Formal Methods. IEEE Computer, 1990, 23(9): 10～22
4 Spivey J M. Understanding Z: A Specification Language and Its Formal Semantics. Cambridge: Cambridge University Press, 1998
5 Justo G R R, Cunha P R F. Formal Specification of Evolving Distributed Software Architectures. In: Proceedings of the 9th International Workshop on Database and Expert Systems Applications, 1998. 548～553
6 Rosa N S, Justo G R R, Cunha P R F. Incorporating Non-functional Requirements into Software Architectures. In: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing. Lecture Notes in Computer Science, 2000. 1009～1018
7 黄靖, 赵海光. 软件复用、软件合成与软件集成. 计算机应用研究, 2004, 9: 118～120
8 Jia Xiaoping. A Tutorial of ZANS-a Z animation system: [Technical report]. School of Computer Science, Telecommunication and Information Systems, DePaul University, Chicago, Illinois. USA, 1995
9 Malenfant J, Jacques M, Demers F N. A tutorial on behavioral reflection and its implementation. In: Proceedings of the Reflection '96 Conference. San Francisco, 1996. 1～20