基于框架和角色模型的软件体系结构规约*

冯铁 张家晨 陈伟 金淳兆

(吉林大学计算机科学系 长春 130023)

E-mail: selab@mail.jlu.edu.cn

摘要 软件体系结构的使用是提高软件质量、减少软件开销和促进软件生产率提高的最有效方法之一.该文提出一种基于面向对象框架和角色模型的软件体系结构规约方法,该方法把体系结构基本元素作为首要的规约对象.在上述方法的基础上,设计了一种体系结构描述语言FRADL(framework and role-type based architecture description language).FRADL认为框架是一种构件,包含主动连接机制的角色模型是一种连接器,构件实例与连接器实例的配置构成软件体系结构.

关键词 软件体系结构,框架,角色模型,软件体系结构描述语言,软件重用.

中图法分类号 TP311

随着软件系统规模和复杂性的增加,人们将注意力从软件的算法和数据结构的选择转向软件体系结构的设计和规约上.软件体系结构的研究目的在于,减少软件并发的开销和提高对紧密相关的系统家族成员的共性进行并发的效率[1].简单来说,软件体系结构由系统组成元素——构件、表达构件间的交互作用的连接——连接器、构件和连接器的拓扑结构——配置组成.一般地,一个特定的软件系统可以通过两个步骤来构造:(1)定义若干构件和连接器,(2)实例化构件和连接器,并将它们配置在一起.当然,这样构造的系统又可以作为一个组成元素,用于构造更大的系统.

角色模型是描述对象协作的最有效的方法之一[2,3],角色模型是使用角色类型对一组对象合作的描述[4].Dirk Riehle等人用角色无关、角色隐含、角色禁止和角色等价这4种约束来定义角色模型,从而完成框架的设计和集成.但是,他们并没有从体系结构的连接器角度出发对角色模型进行讨论.本文对角色模型进行了重新定义,将角色模型作为一种具有脚本定义功能的可执行实体而加以抽象,并作为FRADL(framework and role-type based architecture description language)语言的基本元素进行规约.

目前,国际上比较流行的体系结构规约语言有Aesop[5],Unicon[6],ACME[7],Darwin[8],Wright[9],C2[10],Rapide[11],SADL[12]和MetaH[13].上述体系结构规约语言一般都对构件、连接器进行显式或隐式的描述,同时也具有各自的特点,并应用于不同的领域.Aesop特别适用于开发风格相关的软件体系结构,其主要特点是,采用"生成化方法"产生针对特定风格集的环境(fable),但当软件结构具有任意性时就显得功能有限.Unicon通过枚举的方式定义构件类型,因此不支持构件的子类型化.由于SADL仅允许预定义的连接器类型,因此,它所提供的连接器扩充机制是有限的.MetaH,Rapide和Darwin不区分连接器类型和连接器实例.ACME作为一种体系结构互换语言,旨在支持从一种体系结构规约向另一种体系结构规约的转换.Wright语言是以CSP为基础的,当其用作数学模型对体系结构特性进行分析时比较合适,但在实现上比较复杂和困难.C2是一种基于构件和消息传递的、用于GUI领域的层次型体系结构风格,虽然比较容易实现,但其构件只具有半透明性,结构比较单

* 本文研究得到国家自然科学基金(No.69773044)、国家"九五"重点科技攻关项目基金(No.98-780-01-07-07)和吉林大学青年基金(No.1999A506)资助.作者冯铁,1972年生,讲师,主要研究领域为软件体系结构,软件复用技术,面向对象技术.张家晨,1969年生,副教授,主要研究领域为软件复用技术,软件体系结构描述方法,软件自动化方法.陈伟,1969年生,讲师,主要研究领域为软件测试方法.金淳兆,1937年生,教授,博士生导师,主要研究领域为软件工程.

本文通讯联系人:冯铁,长春 130023,吉林大学(北区)计算机科学系

本文2000-01-17收到原稿,2000-04-24收到修改稿

一.本文提出的FRADL语言吸收了上述语言的一些特点,并注意克服它们的一些弱点,特别地,将框架作为构件,使之具有良好的可继承性和可扩展性.FRADL还注意使构件和连接器具有良好的透明性,对该语言的严格性和易实现性进行折衷,并通过连接器的脚本定义装配整个系统,使其具有较好的并发性和可测试性.

本文第1节解释框架和角色模型的基本概念,阐明它们与构件和连接器之间的关系.这些关系是进一步研究的基础.第2节简单描述FRADL的语法,并解释其如何达到上述设计目标.第3节给出基于FRADL的一个典型的应用实例.

1 框架、角色模型和体系结构元素

1.1 框架和构件

框架是具有表现解决一系列相关问题家族的抽象设计的类的集合[14].目前,人们对框架核心功能的理解已经基本上得到了共识,但对框架的具体构成形式还没有一个一致的看法,一个普遍让人接受的观点是把框架看作一个类模型.为了使框架更适合作为构件,我们认为框架由类模型、集成角色类型集合、可扩展类集合和集成约束集合组成.

(1) 类模型

类模型是指多个类通过继承、关联、聚合、限定等关系连接所表现的面向对象系统的静态模型.它可以用任何一种面向对象开发方法的类图所表示.

(2) 集成角色类型集合

角色类型是一种由对象扮演的类型.在任意给定时刻,对象都可以扮演几个不同的角色类型.这就使得不同的客户可以对同一个对象拥有不同的视图.同样,不同的对象也可以提供由同一个角色类型所说明的行为.集成角色类型集合由那些作为接口、用于与其他构件进行交互的角色类型所组成.当然,这种扮演行为不是由框架本身决定的,而是通过定义连接器的某一角色类型与框架的某一角色类型之间的扮演关系来确定.这种对应关系体现为FRADL配置定义中的扮演声明.从广义上讲,框架既可以仅由一个类构成,也可以包括上百个类.无论在哪一种情况下,都必须把集成角色类型集合的角色类型映射成某种独立的对象行为.

(3) 可扩展类集合

可扩展类的集合由那些将被继承以满足某种实际需求的类组成.这一特点体现为FRADL构件定义中的扩展说明.

(4) 集成约束集合

除了在定义体系结构的连接器时需要说明约束条件之外,框架必须对与其交互的外部环境和某些连接器、对该构件的可能实现方式以及性能加以约束.这一属性体现为FRADL的构件定义中的约束说明.

作为例子,我们通过考虑图形编辑器框架的一部分来指明构成框架的4个部分.图形编辑器框架能够被扩展成编辑状态图、序列图等面向对象开发中的图形标记的应用系统.它也能够与其他构件,如代码生成器,融合到一起,并通过连接器来完成更为复杂的任务,如图1所示.

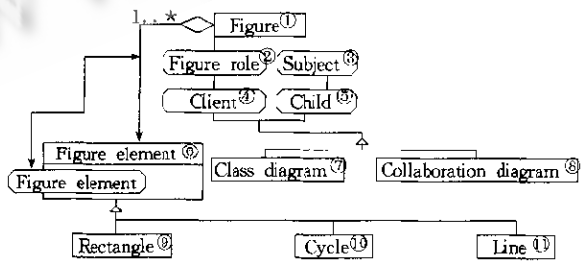


Fig.1 A part of diagram editor framework
图1 图形编辑框架的--部分

在图1中,类Figure定义了4个不同的角色类型:Figure,Subject,Client和Child.这些角色类型定义了

Figure对象的不同行为侧面.角色类型 Figure 描述一般的绘图功能,例如绘画、刷新等.角色类型 Subject 描述当对象状态发生变化时,能够被依赖于 Figure 类实例的其他对象所观察的行为,例如注册、通知等.角色类型 Client 描述 Figure 对象具有申请其他服务的能力,如获得图形设备信息等.角色类型 Child 描述 Figure 对象可以作为某类的子类实例这一特性.类模型是由图 1 中所有的类和继承、关联等类间关系组成.集成角色类型集由角色类型 Figure,Client,Subject,Child 和 FigureElement 组成.当然,还可以通过屏蔽机制将 FigureElement 定义为内部角色,这一点在 FRADL 的初级版本中还没有实现.可扩展类集合由类 Figure 和类 FigureElement 组成.至于集成约束集合,对应于 FRADL 构件约束规约,将在第 3 节中给出它的说明.

框架具有如下特点:(1) 框架可以是软件系统中的主动实体,拥有自己的控制流,控制和调用系统中的其他部分.它也可以是被动实体,被系统中的其他部分所调用,即更像是类库.所以,框架不仅能够表示它提供给其他框架和外界环境的计算和处理,而且能够表示它对其他框架构件和外界环境所请求的计算和处理.(2) 为了表示拓扑结构特性以及对软件体系结构进行推理和分析,框架能够说明一部分系统约束.(3) 框架通常被设计成可以满足多种不同使用目的的实体,它必须能够在不可预见的上下文中进行扩展和集成,尤其是它能同时在多个方向上进行扩展.因此,框架应该体现具有不同抽象层次和扩展方向的灵活性.

构件是具有扩展性和集成性的计算和状态存储的场所.构件由接口和计算两个基本部分组成.

• 接口.一个构件的接口是它和外界的一组交互作用点.接口描述构件需求的服务和提供给外界的服务,特别是构件能够接受的消息和发出的通知.构件通过接口与连接器的角色相匹配,这样,若干构件就在彼此不知道的情况下由连接器组装成一个有机的整体,完成系统家族的共同行为.

• 计算.构件的计算部分描述构件的实际处理,实现接口所描述的实际加工.

除了上述两个基本部分以外,构件还应描述其实现约束,如:开发平台和交互约束.另外,构件与外界的交互行为也要进行说明,如哪一接口在收到什么消息时触发哪些计算,发出什么通知.最后,构件的各个接口分别体现了构件的哪个行为侧面,也应在构件的规约中加以说明.框架和构件的成分对应关系见表 1.

Table 1 Mapping between framework and component

表 1 框架和构件的成分对应关系

Framework①	Component②
Role type set+requirement and notification⑤	Interface defination③
Extensible class set⑤	Component extension⑧
Class model. Interaction⑦	Behavior description⑧
Integration constrain set⑨	Constraint specification⑩
Mapping of role type with methods⑪	Mapping⑫
Class model. Method⑬	Computation⑭

①框架,②构件,③角色类型集+请求和通知,④接口定义,⑤可扩展类,⑥构件扩展,⑦类模型.对象交互,⑧行为描述,⑨集成约束集合,⑩约束说明,⑪角色类型和类方法的对应,⑫映射,⑬类模型.类方法,⑭计算.

在我们的方法中,把框架作为构件加以使用.例如,在规约语言中对构件的接口、行为等描述是通过角色类型集、类模型中的对象方法等基本概念加以说明的.

1.2 角色模型和连接器

角色类型和它们之间的约束及其关系就构成了一个角色模型.也就是说,角色模型是由角色类型及其约束和关系所创建的一组运行时对象合作的约束规约.然而,至今还没有一种将其用作软件体系结构连接器的方法.从软件体系结构角度出发,本文将角色模型界定为角色等价约束和角色协同关系构成的对象交互.

(1) 角色等价(role-equivalent).假设对象 a 扮演由角色类型 A 所定义的角色,对象 b 扮演由角色类型 B 所定义的角色.如果 a 所完成的行为 b 也需要完成,反之亦然,则 role equivalent(A,B)成立.它被表示为<⊣⊢>.

角色等价关系主要用于连接器的组装方面.即由一些重用库中的简单连接器定制较为复杂的连接器.

(2) 协同关系(cooperation).假设对象 a 扮演由角色类型 A 所定义的角色,对象 b 扮演由角色类型 B 所定义的角色,如果对象 a 与对象 b 之间有相互要求的服务序列,则有 Cooperation(A,B)成立.协作关系表示为◀⊢▶.

至于具体的协同行为则由脚本序列加以定义.

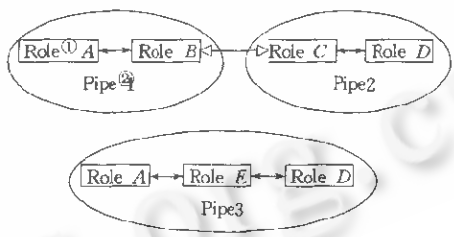
连接器的规约由角色和处理这两个基本要素构成.连接器的角色定义如前述角色类型的定义.例如,一个管道连接器有两个角色:数据源和数据池.一个事件广播连接器的角色可以有一个播音员和一个或多个倾听者.连接器的处理描述由连接器所连接的参与者如何在一起工作而产生一种交互.而且,处理也可以拥有自己的控制线程和数据存储.这使得连接器成为一种可执行的实体.例如,除了角色类型"源"和"池"之外,管道连接器也应该包括一个控制数据存储的实体.这一实体即为管道连接器的处理.

很明显,对象间的角色等价约束和角色协作关系不能完全描述构件之间复杂的相互作用.例如,假设我们构造一个特别的管道-过滤器体系结构,其中要求数据源和数据池之间至多有100个数据在缓冲区中存储,而数据池和存储管理器之间的通信必须用TCP/IP协议实现,这时,角色模型就显示出了自己的弱点.为了加强连接器的对体系结构的约束,我们通过在FRADL中强化连接器的约束条件来达到这一目的.

本文对角色模型所表示的一种约束和一种关系加以改进并作为连接器加以使用.在规约语言中对连接器的角色、处理等描述是通过角色类型、对象交互描述等加以说明的.需要指出的是,除了角色类型所提供的角色协同关系和角色等价关系之外,我们在连接器的规约中引入了描述处理的三元式序列,每一个三元式均指明消息发出者、消息名称和消息接收者.这些由三元式表示的、可并发进行的处理序列在某些事件发生时,也即在某些触发条件成立时被触发.触发条件在三元式的前方限定.在下面的章节中,我们对连接器处理机制还有更详细的说明.这一机制使得连接器成为一种可以引发和协调系统行为的主动实体.

1.3 体系结构配置

连接器角色说明对象行为的一个侧面,而这些对象行为侧面可以和框架的集成角色类型集中的某一角色类型所匹配;从而可以由框架的某个类实例来扮演.这一过程就构成了体系结构的配置.配置用以定义系统总体结构.在上文所提出的机制中,角色模型仅要求由存在某个框架的某个对象行为来扮演它所定义的角色,而不必指明具体由哪个框架的哪个对象行为扮演.这也符合构件无关性的需求,即构件只知道自己的需要和自己所能提供的服务,但并不知道自己与哪个构件合作或与哪个连接器相连.换句话说,构件和连接器都作为类型使用,它们都需要进行实例化,其实例需要根据具体情况经由配置规约匹配在一起.



①角色,②管道.

Fig. 2 Connection between two connectors by role equivalence
图2 通过角色等价建立两个连接器的连接

连接器之间的可连接构造是FRADL的另一个特点.连接器的连接是指由两个或两个以上的连接器组装后形成一个新的连接器.这一功能是利用角色类型间的等价约束在配置部分给以实现的.如图2所示,Pipe1和Pipe2经等价约束形成Pipe3.

2 基于框架和角色类型的体系结构描述语言——FRADL

FRADL融合了上述概念和原理,其规约中包含并混合使用了诸如角色类型、接口等标记.本文只提供语言的主体部分.例如,任何以"-list"结尾的非终极符如"xxx-list"都以如下方式定义:

xxx-list::=null|xxx";"xxx-list.

FRADL被分为如下4个部分:

(1) ComDN(component define notation).ComDN用于描述构件及其接口、计算、行为、映射和约束.在构件与连接器进行的连接中,无论数据的流向如何,都由连接器向构件发出请求,并在构件完成任务后通知连接器.所以,在接口描述中由in和out指明数据流方向,由with短语指明与该角色类型相关的请求和通知(它们是成对出现的).计算部分列出构件自身的方法.行为是由交互列组成的.如,收到某请求消息则执行某计算方法,之后发出某通知消息.映射指明角色类型和一组具有该角色类型所指明的行为侧面相关的方法集合.约束表明对

该构件实现的选择、平台的推荐以及性能的限定.ComDN 也支持构件的改造,这一点与前面提及的框架可扩展类集合相对应.

```
componen::=component-new|component-extension
component-new::=
  Component com-name is
      [class-model class-model-description]
      interfaces interface-list
      computations comput-list
      behaviors behavior-list
      [maps {"("role-type-name,compu-list ");"}]
      [constraints com-constraint-list]
  End Component com-name
component-extension::=
  Component com-name is extension of com-name
      with interfaces-Adding interface-extension-list
      computations-Adding compu-extension-list
      behaviors-Adding behavior-extension-list
      [Maps-Adding {"("role-type-name,compu-list");"}]
      [constraints-Adding constraint-extension-list]
  End component com-name
Class-model-description::=
      member-class(class-name-list)
      role-allocation role-allocation-declaration
      [Structure
          [inheritance={class-name-list from class-name}]
          [aggregation
              {class-name-list in class-name}]
          [association
              {"("rname","class-name","rs","class-name")"}]]
      [extensible-class class-list]
rs::=number arrow number
arrow::="→"|"←→"
interface::=port (in|out) role-type-name with (require-list "&" notification-list)
require::=message message-name [(parameter-list)]
notification::=message message-name[(parameter-list)]
compu::=method method-name[(parameter-list)]
behavior::=meta-behavior "→" behavior|behavior "→" eta-behavior
          |internal-behavior-list"→"behavior|internal-behavior-list
meta-behavior::=receive require |receive notification|send notification| send require
internal-behavior::=compu-list
com-constraint::=implementation language-list on platform-list ";" performance description
```

(2) ConDN(connection define notation).ConDN 用于描述连接器及其角色、处理和约束.角色指明将由构件的角色类型扮演的部分.处理指明对象交互的一组场景和初始状态,其中允许用户定义内部的对象,实现角色间的交互行为,如:某角色 A 向角色 B 发出 C 消息,在结果 D 条件下引发角色 E 向角色 F 发出 G 消息表示为 $(A,C,B)\rightarrow D(E,G,F)$.这一部分也可以用 CSP 表示,这里不采用 CSP 是因为本方法是以对象交互为基础的,

所以用(对象,消息,对象)三元组来表示更合适.它是进行如死锁分析等特性分析的重要依据.约束表明对该连接器的数据流量、通信协议和性能等特性的限定.

connector::=connector-new|connector-inherit

connector-new::=

Connector con-name **is**

roles role-type-name-list

connections-Adding connection-list

process center-controllor

[**constraints** con-constraint-list]

End Connector con-name

connector-inherit::=

Connector con-name **is inheritance of** con-name

with roles-Adding role-type-name-list

connections-Adding connection-list

process-overloading center-controller

[**constraints-Adding** con-constraint-list]

End Connector con-name

connection::=basic-constraint "("role-type-name ","role-type-name")"

basic-constraint::=**role-cooperation**

con-constraint::=[[communication-protocol]|[dataflow-rate]|performance description]

center-controller::=[{"/"internal object"/"}]"{"object-interaction"}" scenario-list

scenario::=object-interaction→[condition]object-interaction

object-interaction::="("role "," message "," role ")"

(3) ACN(architecture configuration notation).ACN 用于描述体系结构配置.配置表现了构件实例和连接器实例的连接关系.在角色-扮演-声明中,构件的角色类型与连接器的角色类型联系在一起.

configuration::=

Configuration config-name **is**

[**Struct**

con-name "=" subconnectorlist"+"equalitylist

]

component-declaration

[connector-declaration]

role-play-declaration

End Configuration

subconnector::=con-name

equality::=**role-equal**"("role type-name ","role-type-name")"

component-declaration::={concrete-com-name **is instance of type** com-name}

connector-name::={concrete-con-name **is instance of type** con-name}

role-play-declaration::={concrete-con-name "." role-type-name **is played by**

concrete-com-name "." role-type-name}

(4) SCN(system construction notation).SCN 描述对已存在的基于软件体系结构的应用系统的修改.若从原系统中移出某构件或连接器,则与该构件相连的连接器以及与该连接器相连的构件也相应移出.若添加某构件,则需指明它作为哪一连接器的哪一端.添加某连接器时亦有相似过程.

system::=system-new|system-adaptation

system-new::=**System** system-name **is**

built on config-name
End System

system-adaptation::=
System system-name **is adaptation of** system-name
with Removing architecture-element-list
Adding architecture-element-list";"
role-play-declaration
End System

3 例 子

假设将一个字符串逆序排列后置入一个栈中是某个子系统的需求.根据软件体系结构的观点,可以构造字符串处理构件和栈构件.字符串处理构件完成一般的诸如字符串逆序、将字符串小写改为大写、将组成字符串的字母排出所有顺序的组合等功能.栈构件完成压栈、出栈、取栈顶元素、判定栈是否为空等功能.另外,还需要构造一个连接器,并对它们进行配置.

首先描述管道连接器.连接器定义了一种不知道与哪两个具体构件相连的类型.在管道连接器中,通过两个角色 source 和 sink 描述以后可能与之相连的两个构件具有的交互行为,一个内部对象 buffer 描述管道在数据传递过程中对数据的缓冲管理.

Connnector pipe **is**
Roles source;sink
Connections role-cooperation(source,sink);
process /buffer/
((buffer,check,buffer))
(buffer,check,buffer)→**empty**(buffer,send,source)
(buffer,check,buffer)→**full**(buffer,receive,sink)
(buffer,check,buffer)→**well**(buffer,store,sink)
(source,sent,buifer)→(buffer,check,buffer)
(sink,received,buffer)→(buffer,check,buffer)
constraints NUMBER(buffer)<=100
PROTOCOL(source,sink)="TCP/IP"
End Connector pipe

然后,描述分别作为数据源和数据池的两个构件.

Component string processor **is**
interfaces port out output **with message** Reverse_ask();**message** Capitalize_ask()
& **message** Reversed(string);**message** Capitalized(string)
computations method reverse(string);**method** capitalize(string)
behaviors receive message Reverse_ask()→**method** reverse(string) →**send message** Reversed(string)
receive message Capitalize_ask()→**method** capitalize(string) →**send message** Capitalized(string)
maps(output.**method** reverse(string),**method** capitalize(string))
constraints implementation "C++"; "Delphi" **on** "UNIX"; "WinNT"
End Component string-processor
Component stack **is**
interfaces port in operation **with message** push-ask(string);**message** pop-ask();**message** Get-top-ask()

& **message** pushed();**message** popped(string);
message top-string(string);**message** empty()

computations **method** push(string);**method** pop(string);**method** top(string);**method** is-empty(yes-or-no)

behaviors **receive message** push-ask(string)→**method** push(string)→**send message** pushed();
receive message pop-ask()→**method** pop(string)→**send message** popped(string);
receive message Get-top-ask()→method top(string);**method** is-empty(yes-or-no)→**send message** popped(string);

maps (operation,**method** push(string);**method** pop(string);**method** top(string);**method** is-empty(yes-or-no))

constraints implementation "C++";"Delphi" **on** "UNIX";"WinNT";response time<=3 second

End Component stack

配置定义如下:

Configuration pipeline **is**

Com1 **is instance of type** string-processor

Com2 **is instance of type** stack

Con **is instance of type** pipe

Con.source **is played by** Com1.output

Con.sink **is played by** Com2.operation

End Configuration

本文所提出的体系结构规约方法为软件系统的某些关键属性的确定建立了基础.例如,可以通过对管道连接器的process进行分析,判定在上例中是否有死锁的可能.因为连接器的process部分列出了可分布实现的对象间的消息序列,构件的behavior部分列出了它的行为序列,所以可画出表达并发交互行为PNG(Petri net graph)和相应的可达树.若可达树出现无法返回的叶节点,则有发生死锁的可能.

4 结束语

本文讨论了框架和角色模型在软件体系结构中的应用,并提出了以它们作为基本构成元素的体系结构规约方法.本文提出的体系结构规约语言FRADL是以框架和角色模型为基础,吸收诸如Wright,C2,Rapid语言的语法特点的一种软件体系结构描述语言.但是FRADL与这些体系结构描述语言有较大的差别.FRADL有如下几个特点:(1)支持面向对象技术的所有特性,包括类、消息发送、继承等;(2)以框架为构件,从而支持大粒度的软件重用;(3)以角色模型为基础,构造具有控制的连接器,精确表达构件的交互协议;(4)构件的行为描述和连接器的处理描述特别适合构造分布并发对象,并支持有效的系统特性分析,如避免死锁;(5)支持不同抽象级别的结构描述,可用于系统分析和设计阶段.

FRADL的重要目标之一是研究软件重用技术和面向对象的程序自动化技术,基于FRADL,我们设计了构件和连接器的重用支撑工具,如构件库的构造和管理工具;还设计了从配置定义向指定的程序设计语言的转换工具,但其中需要部分人工干预.本文的研究是一项探索性的工作,还需要进一步提高FRADL的描述能力,继续研究基于框架和角色模型的体系结构构造技术.

参考文献

1 Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, Inc., Simon & Schuster Bejing Office, Tsinghua University Press, 1996

2 Andersen E P. Conceptual modeling of objects [Ph.D. Thesis]. Oslo, Norway: University of Oslo, 1997

3 Helm R, Holland I M, Gangopadhyay D. Contracts: specifying behavioral compositions in object-oriented systems.

Sigplan Notices, 1990,25(10):169~180
4 Riehle D, Gross T. Role model based framework design and integration. Sigplan Notices, 1998,33(10):117~133
5 Garlan D, Allen R, Ockerbloom J. Exploiting style in architectural design environments. In: Kaiser G E ed. Proceedings of the ACM SIGSOFT'94: the 6th Symposium on the Foundations of Software Engineering. New York: ACM Press, 1994. 175~188
6 Shaw M, DeLine R, Klen D V *et al*. Abstractions for software architecture and tools to support them. IEEE Transactions on Software Engineering, 1995,21(4):314~355
7 Garlan D, Monroe R, Wile D. ACME: an architectural interconnection language. Technical Report, CMU-CS-95-219, Carnegie Mellon University, 1995
8 Magee J, Kramer J. Dynamic structure in software architectures. In: Kaise G E ed. Proceedings of the ACM SIGSOFT'96: the 4th Symposium on the Foundations of Software Engineering. New York: ACM Press, 1996. 3~14
9 Allen R, Garlen D. The Wright Architectural Specification Language. http://www.cs.cmu.edu/afs/cs/project/www/paper-abstracts/wright-tr.html
10 Talor R N, Medvidovic N, Anderson K M *et al*. A component- and message-based architectural style for GUI software. IEEE Transactions on Software Engineering, 1996,22(6):390~406
11 Luckham D C, Vera J. An event-based architecture definition language. IEEE Transactions on Software Engineering, 1995,21(9):717~734
12 Moriconi M, Qian X, Riemenschneider R A. Correct architecture refinement. IEEE Transactions on Software Engineering, 1995,21(4):356~372
13 Vestal S. MetaH programmer's manual. Version 1.09. Technical Report, Honeywell Technology Center, 1996
14 Leveraging Object-Oriented Frameworks. White Paper, Taligent, Inc., 1996

Software Architecture Specification Based on a Framework and Role Type

FENG Tie ZHANG Jia-chen CHEN Wei JIN Chun-zhao

(*Department of Computer Science Jilin University Changchun* 130023)

Abstract Development based on software architecture is one of the most promising methods for improving software quality, reducing software cost and raising software productivity. Viewing architecture elements as first-class entities, the authors present a specification method based on framework and role type to alleviate the problems that conventional methods may bring, and define a specification language FRADL (framework and role-type based architecture description language) in this paper. In FRADL, framework is considered as a component and role type with active connecting faculty is defined as a connector. The instances of components and connectors are configured to compose the architecture for a family of software systems.

Key words Software architecture, framework, role type, software architectural description language, software reuse.