ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.6, June 2006, pp.1298−1306
DOI: 10.1360/jos171298

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

基于反射的连接器组合重用方法*

熊惠民[1,2,3+], 应 时[1], 虞莉娟[4], 张 韬[1]

[1](软件工程国家重点实验室(武汉大学),湖北 武汉 430072)

[2](武汉大学 计算机学院,湖北 武汉 430072)

[3](华中师范大学 数学与统计学学院,湖北 武汉 430070)

[4](武汉理工大学 自动化学院,湖北 武汉 430070)

A Composite Reuse of Architectural Connectors Using Reflection

XIONG Hui-Min[1,2,3+], YING Shi[1], YU Li-Juan[4], ZHANG Tao[1]

[1](State Key Laboratory of Software Engineering (Wuhan University), Wuhan 430072, China)

[2](Computer School, Wuhan University, Wuhan 430072, China)

[3](Department of Mathematics & Statistics, Huazhong Normal University, Wuhan 430070, China)

[4](School of Automation, Wuhan University of Technology, Wuhan 430070, China)

+ Corresponding author: Phn: +86-27-87392220, E-mail: xiong_hm@tom.com, http://www.whu.edu.cn

Xiong HM, Ying S, Yu LJ, Zhang T. A composite reuse of architectural connectors using reflection. *Journal of Software*, 2006,17(6):1298−1306. http://www.jos.org.cn/1000-9825/17/1298.htm

Abstract: A critical issue for complex software architecture design is the modeling and analysis of connectors. This paper argues for an approach to composite reuse of connectors based on reflection. To support this notion a set of mata-operations of connectors composition steming from the operations between processes in CSP have been identified, and the CSP semantics of composite connector is also provided. As it will be shown that it provides mechanisms for designing connectors in an incremental way, and can be analysed and checked automatically owing to the formal basis.

Key words: software architecture; connector composition; reuse; reflection; Wright; CSP

摘 要: 连接器的建模与分析是复杂软件体系结构设计的一个重要问题.基于反射机制,提出了一种连接器的组合重用方法.为了支持这一想法,借用 CSP 进程间的运算,提出了连接器组合的一组元操作,并给出了组合连接器的 CSP 语义.该方法能为连接器提供一种增量式的开发方式,并且由于其严格的形式化基础,适合进行形式化分析和自动检查.

关键词: 软件体系结构;连接器组合;重用;反射;Wright; CSP

中图法分类号: TP311 文献标识码: A

软件重用在提高软件开发的效率和质量上起着重要的作用,通过重用过去的软件产品和软件开发过程,我

* Supported by the National Nature Science Foundation of China under Grant No.60473066 (国家自然科学基金); the Young Outstanding Talent Foundation of Hubei Province of China under Grant No.2003ABB004 (湖北省青年杰出人才基金)

Received 2006-01-09; Accepted 2006-03-13

们可以更为快速、简单地处理重复出现的问题,从而提高软件的开发效率和质量.近些年来,随着软件系统变得越来越复杂,在软件工程界出现了基于组件的软件工程(CBSE).由于 CBSE 将一个软件系统看作是由组件组成的,从而可以提供一种大粒度的重用机制.虽然组件已被认为是一个软件系统的基本块,但是系统的整体特性只有在组件互连之后才显现出来.于是,连接器作为支持组件交互设计的实体而被提了出来[1],从而组件与连接器形成了软件体系结构.软件体系结构描述语言(ADL)是软件体系结构研究的重要领域,它致力于为描述和分析体系结构提供形式化的方法.然而,目前 ADL 对连接器的构造所提供的支持远远不如对组件的支持[2,3].在本文中,我们希望定义一些连接器的操作,该操作允许我们能够从已有的连接器产生出新的连接器.

我们知道,用连接器来建模的交互机制可能非常简单,如过程调用和数据共享,也有可能非常复杂,如数据库存取的安全协议以及 Internet 的网络协议.而一个复杂协议的实现在软件系统的开发中是非常困难的,而且也容易出现错误.因此,如果我们有某种机制,通过重用旧有的、相对简单的连接器来得到新的、较为复杂的连接器,就可以获得一种增量式的连接器开发方法,从而提高软件开发的质量和效率.另外,这种灵活的组织方式对软件系统的演化也是相当有利的.

为了实现上述目的,本文提出了一种基于反射机制的连接器的组合重用方法,它允许我们通过组合元操作将多个连接器组合起来,得到一个新的连接器.与经典的计算反射不同,我们这里把简单连接器所描述的组件交互协议的运行看作是对象计算,而把简单连接器的组合看作是元计算.由于反射概念是我们解决这类问题的一种通用机制,因此本文的方法应被认为是一种系统的、概念化的解决方案.

本文第 1 节介绍相关工作.第 2 节引出一个实例.第 3 节详细论述我们对实例的描述方案,并在此基础上提出连接器组合的元操作.第 4 节给出连接器组合的形式语义,并据此提出连接器组合的两种相容性检查的分析技术.最后总结全文并展望下一步的工作.

1 相关工作

我们所依赖的最一般的工作是关于 ADL 的研究.目前,大多数 ADL 能显式地建模连接器[3],从而为连接器组合的形式化研究提供了基础.到目前为止,比较典型的 ADL 有 Wright[4−6],Darwin[7,8],Rapide[9]等.

一个软件体系结构可能是静态的,也可能是动态的.由于动态的软件体系结构应该支持系统随使用环境的变化更新交互协议,从而与我们的工作相关.目前,很多 ADL 都支持动态体系结构的建模.Rapide[9]和 Wright[6]用一种类似于面向对象语言的方法来动态生成组件,从而获得动态的软件拓扑结构;Darwin[8]是为了规范分布式系统的体系结构而提出的一种声明式的绑定语言,这种配置语言引进了两种动态机制,即惰性实例化和直接动态实例化.

目前也出现了一些带反射特性的动态 ADL,如 Pilar,ArchWare 等.Cuesta 等人提出了一种通用的反射概念框架 MARMOL,并基于该模型构造了一种反射性体系结构描述语言 Pilar. Pilar 以 CCS 为其形式化基础,并用反射原语进行了扩展.它定义的反射原语包括 Avatar(α),Reify(ρ),Destroy(δ)等[10].ArchWare 项目组则希望将软件体系结构与反射系统的研究进行整合,为基于体系结构的软件开发方法提供一套框架、语言和工具.为了规范动态的体系结构,它提出的描述语言 ArchWare 组合了多个概念:用来规范可执行体系结构的π演算、用作系统自省的 hyper-code 以及一个能增量式地拆散运行系统的分解算子和用于新建和绑定组件的结构反射[11].Cazzola 在大规模体系结构编程(APIL)的研究中将反射概念引入到软件体系结构中,提出了体系结构反射的概念,并与计算反射相对应,将体系结构反射分为两种,即拓扑反射(topological reflection)和策略反射(strategic reflection).APIL 在词法上严格区分元级和基级,但其着眼于软件体系结构与实现之间的因果联系[12].

由于上述动态 ADL 主要专注于软件拓扑结构方面的变化,而较少关注组件交互协议的改变,因此,本文的研究对动态体系结构的建模也是一个补充.

关于高阶连接器(higher-order connectors)的研究与我们的工作直接相关.所谓高阶连接器是指以连接器为参数获得新的连接器的一种操作.高阶连接器的思想来自于 Garlan,他认为采用高阶连接器可以对连接器进行组合以适应新的使用环境,也可以分离出连接器的各种不同的公共特性,从而支持重用并更好地理解系统的体

系结构.Garlan 和 Spitznagel 在 Wright 背景下定义了一组转换操作,它们能对一些通用的交互机制增量式地附加新的能力.这组转换操作包括:数据转换(data transform)、粘结(splice)、加角色(add a role)、通话(sessionize)、聚合(aggregate)等[13].

后来,Antónia Lopes 等人通过一个形参声明(formal parameter declaration)和一个体连接器(body connector)建模了高阶连接器.体连接器定义了操作的特性,而形参声明则描述了能应用该操作的连接器的范围.高阶连接器可以用某个连接器实例化,其结果是将该操作所表示的服务叠加到该连接器上.高阶连接器也可以应用到其他高阶连接器上,其结果仍然是一个高阶连接器.用这种方式,他们先将压缩、容错、安全、监视等各种服务用高阶连接器进行独立的规范,再将它们根据需要进行组合,从而实现了类似于菜单选择那样组合不同的服务.该方法以范畴理论作为连接器的形式语义基础,并且它的实现不依赖于任何特殊的 ADL[14].

与上述方法不同,我们提出的形式化方法采用反射机制直接把体系结构中的连接器当作数据来进行操纵,从而实现连接器的组合,达到连接器重用的目的.

2 一个简单的例子

下面通过一个简单的例子来说明我们的方法.考虑这样一个 C-S 连接器,它表示两个组件 Client 和 Server 之间的连接协议.根据 Allen 的观点[5],一个连接器被定义成一组角色(role)规范和一个粘结(glue)规范.角色描述了每个参与交互的组件预期要发生的行为,或者说,它决定了一个组件如果成为角色实例所应承担的责任.而粘结则描述了对这些角色活动的协调与约束.上述连接器类型用体系结构描述语言 Wright 进行规范,如图 1 所示.

connector C-S-connector=
  role client=(request!x→result?y→Client)∏§
  role server=(invoke?x→return!y→Server)□§
  glue=(client.request?x→server.invoke!x→server.return?y→client.result!y→ **glue**)□§

Fig.1 Wright specification: C-S-connector

图 1 Wright 规范:C-S-connector

现在假使要构造一个比较复杂的 C-S 系统,它只有一个 Server,但有多个 Client.显然,此时 Server 与每个 Client 的通信协议仍然是 C-S-connector,即可以重用已有的连接器.但是,由于该系统要求多个 Client 与同一个 Server 进行交互,Client 之间还必须进行相互协调,因此,又必须对连接器进行相应的修改.

3 我们的方法

我们认为,上述重用过程可以采用连接器的组合来表达.由于 Wright 能够结构清晰地、显式地表达组件间的交互(即连接器),分析系统的组合特性(如无死锁性),是一种理想的体系结构描述方法,因此,我们的工作仍建立在 Wright 基础之上.同时,我们想把软件系统的这种重用行为放在一种通用的框架下去解决,从而为这个问题提供一种系统的、概念化的解决方案.具体的做法是将连接器的组合重用机制用一个反射模型来表达:用 Wright 来描述各个简单的连接器,以此作为反射系统的基级;再引入元级来描述简单连接器的组合.这种方法的形式化理论基础仍然是 CSP[15].

3.1 反射与软件体系结构重用

反射(reflection)是一个系统对自身进行操作和推理的能力[16].如果一个系统能对另一系统的模型进行操作,我们就称这个系统件是另一系统的元系统(meta-system).如果一个系统充当了自身的元系统,那么该系统就是一个反射系统.因此,一个反射系统必须有一个能表示自身的模型,这个模型能够提供对自身状态和行为的自我描述,并且系统的实际状态和行为要始终与自述保持因果联系(causual connection),即对自述的改变能够立即反映到系统实际的状态和行为,而系统的实际状态和行为改变也能够立即在自述中反映出来.反射的这种特性使得它有可能适应改变的环境,从而为系统提供较高的灵活性.

在结构上,一个反射系统可以从两个方面来理解,从而分成两级:基级(base-level)实现系统的对象计算,即描述外部世界的计算行为;而元级(meta-level)则实现系统的元计算,即负责建立和维护系统自述.当然,这个元级还可以看作是基级,再在它上面构造新的元级,即元级的元级,从而形成一个多层的塔式结构.

反射可分为经典的计算反射(computational reflection)与体系结构反射(architecture reflection)[12],前者只对单个组件计算实施反射行为,而后者则要对整个系统的体系结构实施反射计算.也就是说,计算反射与体系结构反射属于两个不同的层次,一个体系结构反射系统各级的体系结构所包含的组件和连接器,还可以有计算反射的结构,因此,它们可以正交.我们这里所采用的反射机制属于体系结构反射的范畴.

反射技术已成功地应用在代码组件的重用方面,例如:JavaBean 就是一种基于反射机制的代码组件重用技术,BeanBox 工具则是利用反射机制,为重用 Bean 组件提供的一种简单、有效的可视化编程手段.反射技术也可以用于软件设计阶段,以支持软件体系结构的重用.我们可以把体系结构的各个组成元素以及体系结构本身作为体系结构层可重用的软件资源,同时将支持软件体系结构重用的各种元操作显式地表达出来.从而,设计人员就可以通过重用已有的体系结构设计结果,得到满足更多需求的软件体系结构,软件体系结构的设计和维护也就可以通过重用体系结构这种大粒度、高抽象的软件资源的方式来完成.在反射机制的支持下,任何一个体系结构重用操作都可以分成 3 个步骤:首先,元级需要从基级获取有关体系结构的元信息;然后根据所执行的元操作,元级修改元信息;最后,根据修改后的元信息,反射生成基级体系结构.

3.2 连接器组合的描述

考虑上述简单的连接器组合例子,其描述如图 2 所示.

```
connector C2-S-connector
    composite:   cs1: C-S-connector
                 cs2: C-S-connector
    role client1=Substitute(cs1.client)
    role client2=Substitute(cs2.client)
    role server=AlternativeMerge(cs1.server,cs2.server)
    glue=Choice(cs1.glue,cs2.glue)
```

Fig.2 A simple example: C2-S-connector specification

图 2 一个简单的例子:C2-S-connector 连接器规范

该形式规范首先标明 C2-S-connector 是一个组合连接器,为此,我们引进了一个保留字 composite.图 2 中的 composite 描述说明,该组合连接器是通过两个具有同一类型——C-S-connector 的连接器组合得到的.

为了规范一个连接器类型,Wright 为它的每一个角色以及角色间的粘结都提供一个进程描述.同样地,为了规范一个组合连接器,我们也提供对这样一些结构的描述.

该形式规范接着描述了组合连接器包含的 3 个角色.

角色 client1 描述了该交互协议中一个 client 的行为,角色 client2 则描述了另一个 client 的行为.函数 Substitute 表明这两个角色实际上分别是其子连接器 cs1 和 cs2 的 client 角色.或者反过来说,cs1 和 cs2 的 client 角色分别由 client1 和 client2 来充当.从而,client1 具有与 cs1.client 相同的行为规范,client2 具有与 cs2.client 相同的行为规范.如果我们用 Wright 来表达,那么这段描述等同于

$$client1=request!x\rightarrow result?y\rightarrow client1 \sqcap \S$$

$$client2=request!x\rightarrow result?y\rightarrow client2 \sqcap \S$$

第 3 个角色是 server,它引进了一个新的函数 AlternativeMerge.从直觉上讲,在该实例中,两个子连接器中的 server 角色将来要被同一个组件端口来实例化,即要对外界表现为一个角色.因此,组合连接器 server 角色所代表的服务既要能被 cs1.client 所调用,又要能被 cs2.client 所调用.即它既要能充当 cs1.server 的角色,又要能充当 cs2.server 的角色.AlternativeMerge 函数实现了用一个角色充当多个角色的目的,它每次选择“扮演”多个角色中的一个角色.由于 cs1.server 与 cs2.server 的行为规范相同,因此,如果用 Wright 来表达,那么这段描述等同于

server=invoke?x→return!y→server□§

该形式规范最后描述了对上述 3 个角色的约束与协调——粘结 glue.它又引进了一个函数 Choice,其含义是:选择 cs1.glue 与 cs2.glue 中的某一个来充当连接器 C2-S-connector 的粘结,并且这种选择是由其所在的外部环境所决定.这显然与我们的直觉相符,连接器到底选择哪一种协调方式,就看 server 是为哪一个 client 服务,并且这种选择权在 client,即哪一个 client 提出请求就为哪一个服务.如果用 Wright 来表达的话,那么这段描述等同于

glue=(client1.request?x→server.invoke!x→server.return?y→client1.result!y→**glue**)□(client2.request?x→server.invoke!x→server.return?y→client2.result!y→**glue**)□§

在图 2 中,为了描述组合连接器 C2-S-connector,我们引进了 3 个函数 Substitute,AlternativeMerge,Choice 来详细地说明 composite 的含义.这 3 个函数是对角色或者粘结来进行操作的,因此属于元计算的范畴,我们称其为元操作,这也是称这种组合方法是基于反射机制的原因.

3.3 连接器组合的元操作

如前所述,引进连接器组合元操作的目的是为了将多个角色或者粘结组合地连接起来.由于我们把角色和粘结都看作进程,因此,连接器组合的元操作实质上可以看作是进程间的运算.CSP 关于进程间的运算有一个丰富的构造集,我们采用其中的一个子集来定义连接器组合的元操作.

连接器组合的元操作包括两类:一类是关于角色的元操作,另一类是关于粘结的元操作.关于角色的元操作主要有:

Substitute:角色的替代.利用它,可以实现用一个角色来充当另一个已经定义的角色;

ConcurrencyMerge:角色的并行合一.利用它,可以实现用一个角色来同时充当多个已经定义的角色,并且它“扮演”的多个角色之间应并行协调;

AlternativeMerge:角色的选择合一.利用它,可以实现用一个角色来完成多个已经定义的角色的功能,并且在每一次完整的交互中该角色只能充当其中的某一个角色.

关于粘结的元操作主要有:

Parallel:该操作将两个或者多个粘结进程并行地组合起来.如果用这种组合得到的粘结进程去规范某个角色行为,那么该角色无论何时要想参与某一个事件,都必须得到各个子粘结进程的共同允许.

Decision:该操作将两个或者多个粘结进程不确定性选择地组合起来.这里的不确定性选择指的是:组合得到的粘结进程究竟选择哪一个子粘结进程去规范角色的某一次完整的交互行为,由其自身来决定.

Choice:该操作将两个或者多个粘结进程选择地组合起来.这种选择可能是上述的不确定性选择,也可能是确定的选择,即选择权在其所在环境的选择.如果它所规范的角色在某次完整的交互中想要参与的初始事件仅被某个子粘结进程所允许,那么组合粘结进程就选择该子粘结进程去承担该次交互的协调任务;否则,如果角色想要参与的初始事件为多个子粘结进程所允许,那么它就会任意选择其中的某个子粘结进程去承担此次交互的协调任务.

Interleave:该操作将两个或者多个粘结进程交错地组合起来.如果用这种组合得到的粘结进程去协调和约束某个角色的行为,那么该角色无论何时要想参与某一个事件,只需得到某个子粘结进程的允许即可.当然,如果此时有多个子粘结进程都允许该事件发生,那么组合粘结进程就会任意选择其中的某个子粘结进程去承担允许该事件发生的责任.

Follow:该操作将两个或者多个粘结进程顺序地组合起来.用这种组合得到的粘结进程依次用其子粘结进程去协调和约束其所规范的角色的行为,当然,后续的子粘结进程要想承担这种责任,必须满足前行的子粘结进程能够成功终止.

Interrupt:该操作将两个或者多个粘结进程顺序中断地组合起来.用这种组合得到的粘结进程可以随着后续子粘结进程初始事件的发生,用后续的子粘结进程去中断和接替前行的子粘结进程,并获得协调和约束角色的责任.

Lightning:该操作可以看作是 Interrupt 的一种特殊情形,它将两个粘结进程顺序中断地组合起来.但与

Interrupt 不同的是,前行子粘结进程被中断并不取决于后续子粘结进程初始事件的发生,而是某个被定义的中断事件.为了表示这个特殊事件,我们把它作为第 3 个参数引入到 Lightning 函数中.

4 语义与分析

类似于 Wright,我们将连接器组合的描述转化为纯粹 CSP 的语言,以此来表达它的形式语义.对于一个给定的连接器组合描述,其 CSP 语义可以通过以下 3 个步骤得到:

(1) 获得组合连接器角色的 CSP 语义.

一个组合连接器的角色是对简单连接器的角色进行替代或合一得到的,因此,它的 CSP 语义可以通过将这两类角色的元操作转换为 CSP 进程间的运算来给出.

定义 **1**. 假使某组合连接器包含角色 R,

① 若 $R=Substitute(c.r)$,则 R 的形式语义为进程 r;

② 若 $R=ConcurrencyMerge(c_1.r_1,c_2.r_2,\ldots,c_n.r_n)$,则 R 的形式语义为进程 $r_1 \| r_2 \| \ldots \| r_n$;

③ 若 $R=AlternativeMerge(c_1.r_1,c_2.r_2,\ldots,c_n.r_n)$,则 R 的形式语义为进程 $r_1 \square r_2 \square \ldots \square r_n$.

为了描述组合连接器粘结的 CSP 语义,我们需要引进如下定义:

定义 **2**. 假使某连接器组合描述包含角色 R,

① 若 $R=Substitute(c.r)$,则称该组合描述包含有重标识 $\{R/c.r\}$;

② 若 $R=ConcurrencyMerge(c_1.r_1,c_2.r_2,\ldots,c_n.r_n)$,则称该组合描述包含有重标识 $\{R/c_1.r_1\},\{R/c_2.r_2\},\ldots,\{R/c_n.r_n\}$;

③ 若 $R=AlternativeMerge(c_1.r_1,c_2.r_2,\ldots,c_n.r_n)$,则称该组合描述包含有重标识 $\{R/c_1.r_1\},\{R/c_2.r_2\},\ldots,\{R/c_n.r_n\}$.

(2) 获得组合连接器粘结的 CSP 语义.

一个组合连接器的粘结的 CSP 语义也可以通过将粘结的元操作转换为 CSP 进程间的运算来给出.不过,由于连接器组合后角色都经过了替代或合一,因此,各个子粘结进程在组合时也要进行相应的重新标识.

定义 **3**. 假使某个组合连接器含有子连接器 c,且其组合描述中包含关于 c 的重标识有 $\{R_1/c.r_1\},\{R_2/c.r_2\},\ldots,\{R_n/c.r_n\}$,则

$$ReL(c.glue)=c.glue/\{R_1/c.r_1\},\{R_2/c.r_2\},\ldots,\{R_n/c.r_n\}.$$

这里,$c.glue/\{R_1/c.r_1\},\{R_2/c.r_2\},\ldots,\{R_n/c.r_n\}$ 表示将 $c.glue$ 的进程表达式中所有标记 $c.r_1,c.r_2,\ldots,c.r_n$ 分别替换为新的标记 $R_1,R_2,\ldots,R_n$.

定义 **4**. 假使某连接器组合描述包含粘结 $glue$,

① 若 $glue=Parallel(c_1.glue,c_2.glue,\ldots,c_n.glue)$,则粘结 $glue$ 的形式语义为进程

$$ReL(c_1.glue) \| ReL(c_2.glue) \| \ldots \| ReL(c_n.glue);$$

② 若 $glue=Decision(c_1.glue,c_2.glue,\ldots,c_n.glue)$,则粘结 $glue$ 的形式语义为进程

$$ReL(c_1.glue) \sqcap ReL(c_2.glue) \sqcap \ldots \sqcap ReL(c_n.glue);$$

③ 若 $glue=Choice(c_1.glue,c_2.glue,\ldots,c_n.glue)$,则粘结 $glue$ 的形式语义为进程

$$ReL(c_1.glue) \square ReL(c_2.glue) \square \ldots \square ReL(c_n.glue);$$

④ 若 $glue=Interleave(c_1.glue,c_2.glue,\ldots,c_n.glue)$,则粘结 $glue$ 的形式语义为进程

$$ReL(c_1.glue) \,|||\, Rel(c_2.glue) \,|||\ldots|||\, ReL(c_n.glue);$$

⑤ 若 $glue=Follow(c_1.glue,c_2.glue,\ldots,c_n.glue)$,则粘结 $glue$ 的形式语义为进程

$$ReL(c_1.glue);ReL(c_2.glue);\ldots;ReL(c_n.glue);$$

⑥ 若 $glue=Interrupt(c_1.glue,c_2.glue,\ldots,c_n.glue)$,则粘结 $glue$ 的形式语义为进程

$$ReL(c_1.glue) \Delta ReL(c_2.glue) \Delta \ldots \Delta ReL(c_n.glue);$$

⑦ 若 $glue=Lightning(c_1.glue,c_2.glue,event)$,则粘结 $glue$ 的形式语义为进程

$$ReL(c_1.glue) \,\hat{\varsigma}\, ReL(c_2.glue) \,\hat{\varsigma}\, \ldots \,\hat{\varsigma}\, ReL(c_n.glue),\text{且}\varsigma=event;$$

(3) 获得组合连接器类型的 CSP 语义.

非形式地,Wright 连接器把各个角色看作是相互独立的进程,而由粘结来协调和限制它们的行为.由此,我们可以得到一个组合连接器类型的 CSP 语义.

定义 5. 假使某组合连接器包含角色 $r_1,r_2,\ldots,r_n$ 和粘结 *glue*,且它们的 CSP 语义分别为 $S(r_1),S(r_2),\ldots,S(r_n)$和 $S(glue)$,那么,该组合连接器类型的形式语义为以下进程

$$S(glue) \parallel (r_1:S(r_1) \parallel r_2:S(r_2) \parallel \ldots \parallel r_n:S(r_n)).$$

语义是分析的基础.借助已给出的连接器组合的形式语义,我们容易将 Wright 的分析技术扩展到新的背景.因为连接器组合规范的是连接器类型,因此我们可以独立于组件的分析来对组合连接器进行分析.这种方法可以提供的分析技术包括以下两种相容性的检查:

检查 1:连接器的每个角色都是无死锁的.

这是对连接器角色内部相容性的检测.由于组合连接器的每个角色是在重用已有连接器的角色基础上得到的,因此,这种检查可以分为两种情形:若组合连接器的某个角色是通过替换或者选择合一得到的,那么对子连接器相应角色的检查结果仍然适用于组合连接器的这一角色;若组合连接器的某个角色是通过并行合一得到的,那么就必须重新检查.因为对于一个并行合一的角色进程,可能会出现这样的问题:在某个时候,虽然它的子角色都各自能参与某些事件,但它却不能参与任何一个事件.

检查 2:连接器是无死锁的.

这种相容性的检查是对连接器整体的检查.因此,检查 1 如果通不过,也会反映到检查 2 中.角色规范了充当其实例的组件预期要发生的行为,而粘结规范的是对这些行为的协调与约束.角色规范与粘结规范是否会出现矛盾,就需要用检查 2 来考察.

上述两种检查可以直接重用 Wright 的方法,并且借助于 FDR[17]模型检查工具的帮助,很容易自动实现.FDR 不仅能给出检查的结果,而且对于错误的情形还能向用户提供导致死锁发生的迹的信息.这些信息能够帮助设计者调试和修改其设计的连接器组合规范.

下面以组合连接器 C2-S-connector 为例,说明如何用 FDR 软件来作第 1 种相容性检查.第 2 种相容性检查的方法与此类似.由于 FDR 只能判断 CSP 进程间是否存在精化关系,为此,我们必须将 CSP 进程的无死锁性转化为某种精化关系.事实上,我们有以下结论[4]:

定理. 一个字符表为 A 的进程 P 无死锁当且仅当它为进程 DF_A 的精化,这里

$$DF_A=(\prod e:A \cdot e\rightarrow DF_A)\prod \S.$$

即 DF_A 表示字符表为 A 的最不确定的无死锁进程.

为了使用该检查工具,我们还必须对组合连接器的角色描述用 FDR 符号来进行重新编码.根据定义 1 给出的角色语义,组合连接器 C2-S-connector 包含了以下角色进程:

client1=request!x→result?y→client1∏§

client2=request!x→result?y→client2∏§

server=(invoke?x→return!y→server□§)□(invoke?x→return!y→server□§)

它们在 FDR 中可编码为:

Client1=(request–>result–>Client1)|~|TICK

Client2=(request–>result–>Client2)|~|TICK

Server=((invoke–>return–>Server)[ ]TICK)[ ]((invoke–>return–>Server)[ ]TICK)

另外,如前所述,我们还需要定义以下两个最不确定的无死锁进程:

DF1=(request–>DF1)|~|(result–>DF1)|~|TICK

DF2=(invoke–>DF2)|~|(return–>DF2)|~|TICK

最后执行命令:

Check “DF1” “Client1”;

Check “DF1” “Client2”;

Check “DF2” “Server”;

这样就完成了对组合连接器 3 个角色的无死锁检查.

5 结论以及将来的工作

连接器设计的一个重要问题是重用已有的连接器来构造新的连接器.本文在 Wright 连接器的形式化基础上为连接器的组合重用提供了一个形式框架.该框架基于反射机制提出了一组连接器组合的元操作.通过这组元操作,可以增量式地开发连接器,从而允许软件设计者以一种系统的方式来定义更复杂的交互协议.由于其严格的形式化基础,该方法可进行连接器组合的相容性检查.

遵循 Wright 的方法,为了获得一个组合连接器,我们在组合描述中为每一个构成成分(即角色和粘结)提供一个进程运算的描述,这进一步论证了 Wright 方法表达体系结构的有效性.从而一般地,我们认为用进程代数作为 ADL 的形式化基础是一种合适的选择.

本文选取了 CSP 的一部分运算符来构成连接器组合的一组元操作.下一步,我们需要研究的是这组元操作的表达能力是否足够丰富、是否还需要对它们进行扩充.由于篇幅所限,本文仅选择了一个简单的例子对这种方法进行说明.因此,为了回答上述问题,我们在今后的研究中还需要做更多的实例分析.

另外,为了简单起见,我们未允许角色和粘结执行复合的元操作.很显然,这种扩充是容易完成的.由于元操作的复合可以采用多次组合连接器的方法来等价地实现,从而这样做并不能增强表达能力,但它可以更方便地表达我们所需要的组合方式.

另一个值得研究的重要问题是如何将这种方法应用到动态环境里.如前所述,当前动态 ADL 主要关注软件体系结构拓扑方面的动态变化,这里提出的连接器组合提供了一种对组件交互协议进行变更的方法.如果能将这两方面结合起来,可增强动态 ADL 的表达能力.

References:

[1] Shaw M. Architectural issues in software reuse: It’s not just the functionality, it’s the packaging. In: Samadzadeh M, Zand M, eds. Proc. of the 1995 Symp. on Software Reusability. New York: ACM Press, 1995. 3−6.

[2] Sun CA, Jin MZ, Liu C. Overviews on software architecture research. Journal of Software, 2002,13(7):1228−1237 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/1228.htm

[3] Medvidovic N, Taylor RN. A classification and comparison framework for software architecture description languages. IEEE Trans. on Software Engineering, 2000,26(1):70−93.

[4] Allen RJ. A formal approach to software architecture [Ph.D. Thesis]. Pittsburgh: Carnegie Mellon University, 1997.

[5] Allen R, Garlan D. A formal basis for architectural connection. ACM Trans. on Software Engineering and Methodology, 1997,6(3): 213−249.

[6] Allen R, Douence R, Garlan D. Specifying and analyzing dynamic software architectures. In: Astesiano E, ed. Proc. of the 1st Int’l Conf. on Fundamental Approaches to Software Engineering. Berlin: Springer-Verlag, 1998. 21−37.

[7] Magee J, Dulay N, Eisenbach S, Kramer J. Specifying distributed software architectures. In: Schafer W, Botella P, eds. Proc. of the 5th European Software Engineering Conf. Berlin: Springer-Verlag, 1995. 137−153.

[8] Magee J, Kramer J. Dynamic structure in software architectures. In: Kaiser GE, ed. Proc. of the 4th ACM SIGSOFT Symp. on Foundations of Software Engineering. New York: ACM Press, 1996. 3−14.

[9] Luckham DC, Augustin LM, Kenney JJ, Vera J, Bryan D, Mann W. Specification and analysis of system architecture using rapide. IEEE Trans. on Software Engineering, 1995,21(4):336−355.

[10] Cuesta CE, de la Fuente P, Barrio-Solorzano M. Dynamic coordination architecture through the use of reflection. In: Lamont GB, ed. Proc. of the 2001 ACM Symp. on Applied Computing. New York: ACM Press, 2001. 134−140.

[11] Morrison R, Kirby G, Balasubramaniam D, Mickan K, Oquendo F, Cimpan S, Warboys B, Snowdon B, Greenwood RM. Support for evolving software architectures in the ArchWare ADL. In: Proc. of the 4th Working IEEEE/IFIP Conf. on Software Architecture. Washington: IEEE Computer Society, 2004. 69−78.

[12] Cazzola W, Savigni A, Sosio A, Tisato F. Architectural reflection: Concepts, design, and evaluation. Technical Report, RI-DSI 234-99, Milano: University of Milano Bicocca, 1999.

[13] Spitznagel B, Garlan D. A compositional approach for constructing connectors. In: Kazman R, Kruchten P, Verhoef C, van Vliet H, eds. Proc. of the 2nd Working IEEEE/IFIP Conf. on Software Architecture. Washington: IEEE Computer Society, 2001. 148−157.

[14] Lopes A, Wermelinger M, Fiadeiro JL. Higher-Order architectural connectors. ACM Trans. on Software Engineering and Methodology, 2003,12(1):64−104.

[15] Hoare CAR. Communicating Sequential Processes. Englewood Cliffs. New Jersey: Prentice Hall, 1985.

[16] Maes P. Concepts and experiments in computational reflection. In: Meyrowitz N, ed. Proc. of Conf. on Object-oriented Programming Systems, Languages and Applications. New York: ACM Press, 1987. 147−155.

[17] Formal Systems (Europe) Ltd. Failures divergence refinement: User manual and tutorial. Version 1.3, 1993. http://www.fsel.com

附中文参考文献:

[2] 孙昌爱,金茂忠,刘超.软件体系结构研究综述.软件学报,2002,13(7):1228−1237. http://www.jos.org.cn/1000-9825/13/1228.htm

熊惠民(1971－),男,湖北汉川人,博士生,讲师,主要研究领域为软件体系结构,软件形式化方法,计算模型.

虞莉娟(1975－),女,讲师,主要研究领域为电路理论,自动控制理论及应用.

应时(1965－)男,博士,教授,博士生导师,主要研究领域为软件体系结构,软件的可重用性与互操作性,面向 Aspect 的软件开发.

张韬(1979－),女,博士生,主要研究领域为软件体系结构,面向服务软件工程,软件重用.