第 41 卷 第 5 期
2005 年 9 月

南京大学学报(自然科学)
JOURNAL OF NANJING UNIVERSITY
(NATURAL SCIENCES)

Vol. 41, No. 5
Sept., 2005

计算机软件新技术研究进展专栏

通用连接器模型及其形式化推导研究*

许 毅**,赵文耘,彭 鑫,张 志
(复旦大学计算机科学与技术系,上海,200433)

摘 要: 基于构件的软件体系结构(SA)由构件与连接器组成,连接器作为构件间的交互实体在 SA 中扮演着重要角色. 现有的连接器模型只能支持特定软件体系结构风格的组装,缺乏统一的连接器模型的支持. 提出一种通用连接器模型用以对不同连接器模型进行描述,并在此基础上对连接器模型进行了扩展,以支持不同体系结构风格的构件组装. 同时,基于 Wright 的软件体系结构描述语言(ADL)和通信顺序进程(CSP)中对于进程的描述方法,给出了通用连接器模型的形式化推导、验证,从而为分析连接器行为,支持连接器自动生成奠定了基础.

关键词: 软件体系结构,连接器,体系结构描述语言,通信顺序进程
中图分类号: TP 311

A General Connector Model and Its Formal Reasoning

Xu Yi, Zhao Wen-Yun, Peng Xin, Zhang Zhi
(Department of Computer Science and Engineering, Fudan University, Shanghai, 200433, China)

Abstract: As software systems become more complex, the overall system structure or software architecture becomes a central design problem. Recently, software architecture has placed significant importance on components and interactions between them. The component-oriented software architecture (SA) is composed of components and connectors which play a very important role in SA. One of architecture research focus is architecture style, and now there are many architecture styles such as client-server system, layered system, blackboard organization and so on which have been used widely. Nonetheless, the existing connector models can only allow the specific SA styles, and lack a general connector model to support them. This paper abstracts the essential of different taxonomies of software connectors, and makes the connectors composed by fine-grained elements, each of which represents a single, well-defined function. And then a general connector model is presented to describe different connector types, which can also extends itself to admit different SA styles' composition. We can also put some non-functional properties such as distribution element, security element and fault-tolerance element into the general connector model. Another research focus is architectural design languages. They help the architectures' design and facilitate the designers' communication. Up to now a variety of architectural design languages such as Wright, UniCon and Rapide have been created to provide software architects with notations for specifying and reasoning about architectural designs. Moreover, an important step towards an engineering discipline of

* 基金项目:国家 863 计划(2004AA112070,2004AA113030),国家自然科学基金(60473061)
收稿日期:2005 - 07 - 08
** 通讯联系人,E-mail:032021162@fudan.edu.cn

software is a formal basis for describing and analyzing the system 's behaviors. Based on the Wright 's research and CSP on formal specification of software architecture, this paper develops the behaviors of the general connector model and its elements, which establishes a foundation to analyze, validate, simulate composite systems and give a strong support to automatically generate.

Key words: software architecture (SA), connector, architecture description language (ADL), communication sequence process (CSP)

软件体系结构(software architecture,简称 SA)是对软件总体结构的描述,即对其构件和构件间交互的高层组织的描述[1]. 它作为一种高层的设计,对系统开发发挥着重要的影响,基于构件的 SA 的设计已成为软件系统设计中的核心问题. 一个好的 SA 设计会成为大型软件系统成功的重要因素,它最重要的一个贡献是将构件之间的交互显式的表示为连接器(connector),并将连接器视为和构件同等重要的一阶实体.

在 SA 的设计过程中,人们针对不同的需求采用了不同的软件构架风格. 体系结构风格定义了一系列系统的结构组织的模式[2],它是对一类具有相似结构的系统体系结构的抽象. 体系结构风格既定义了构件及连接方式的各种属性,又规定了它们的组合规则和限制[1]. 近十年中人们设计了如管道 - 过滤器、事件驱动、基于消息、过程调用式等 SA 风格. 各种不同的 SA 风格的设计均是基于不同类型的连接器[2]. 文献[3]中,连接器被划分成 8 种类型:过程调用类型(Process Call)、事件类型(Event)、流类型(Stream)、分布者类型(Distributor)、数据接入类型(Data Access)、仲裁者类型(Arbitrator)、适配器类型(Adaptor)、联接类型(Linkage). 文献[4]中,非功能属性(如保密性,服务质量等)被认为是连接器描述中的重要组成部分.

SA 的主要研究成果表现为体系结构描述语言(architecture description language,简称 ADL),如 Wright[5]、Rapide[6]、UniCon[7]等. 文献[8]中采用了自省的方式来进行 SA 的描述. Wright 是具有代表性的 ADL 之一,其主要特点是将通信顺序进程[9](communicating sequential processes,简称 CSP)用于 SA 的描述,从而支持对体系结构描述的某些形式化推理(包括相容性检查和死锁检查等). 形式化方法能对系统进行精确的并且无二义性的描述,并能在此基础上进行推理,从而可以对系统应具有的关键性质进行验证,以保证它的正确性.

本文提出了一个通用的连接器模型,模型中连接器由更细粒度的行为元素组成,非功能属性也被规约成行为元素. 它能描述现有的各种连接器类型,而且还可以支持新的连接器类型,从而实现不同的 SA 风格,甚至完成不同 SA 风格的互连.

同时基于 Wright 的 ADL 和 CSP 中对于进程的描述,本文还将通用连接器模型的形式化描述进行分解,从而推导出各行为元素的语义. 各行为元素语义能够作为构件组装机制的语义,指导胶合代码(glue code)的生成.

1 相关工作

SA 中连接器的研究一直是一个研究热点,和本文相关研究有:软件体系结构,连接器模型,连接器的行为协议.

在软件体系结构方面,文献[5,10]中,连接器被认为是和构件同等重要的一阶实体. 连接器是软件体系结构设计层次上的一个概念,它作为构件间的交互应清晰的描述交互语义. 对连接器进行系统的分类的工作已经在进行[3,11]. 同时构件间交互的不匹配是构件组装的难点,同时也是构件组装的核心问题,为此非功能属性被广泛地使用. 文献[12]中,提出了使用包装器(wrapper)的方法来实现非功能属

性,文献[4,13]中,将非功能属性作为连接器的一部分,可以自定义的添加.

在连接器模型方面,文献[13]中提出了一个在实现层次上的连接器模型. 文献[11]中提出连接器的元模型,并将其等同于构件,可以对连接器进行继承,组合. 文献[4]中提出了一个细粒度的连接模型,它将非功能属性作为一个附加属性添加在连接器上,该模型与本文模型最大的区别在于前者没有将构件接入与消息转化的行为元素独立出来,而本文模型使用单独的行为元素来完成转化功能,同时还有一个控制中心来对连接器中各行为元素进行控制.

在连接器行为协议方面,文献[14]中采用了有限状态机来描述交互协议. 文献[15]中对连接器的行为协议进行了详细的描述,建立了一套连接器合法性、正确性的判别标准. 文献[16]中对构件组装的行为进行了基于 Wright ADL 的形式化推导. 文献[12]中采用的包装器也采用了 Wright ADL 的形式化描述.

2　通用连接器模型

在基于构件的系统中,连接器作为构件间交互的实体. 不同的 SA 风格中构件的交互方式也不尽相同,因此需要不同类型的连接器. 连接器只是一个抽象的概念,不像构件那样有精确的描述,针对以上的需求,提出了一个连接器通用模型,连接器是由更细粒度的行为元素组成.

一个通用连接器模型应该要解决以下的问题:(1) 代表各种的交互风格;(2) 提供额外非功能性属性的选择,如安全性、事务性、容错性、转化、同步、分布式等;(3) 最大程度支持代码自动生成.

该模型允许构件各种方式接入,其内部使用消息通信. 模型包含一个控制中心(control center)和各种行为元素(element). 其中行为元素可分为两大类:

1. 必需行为元素:Role,Stub.

2. 可选行为元素: De/ Encrytor, Distributor, Dispatcher, Adaptor, Transaction Manager, Interceptor, Logger 等;可选行为元素可自由添加.

控制中心和必需行为元素在组成连接器时是必不可少的,而可选行为元素则可以按照设计者的意图进行插拔. 通过行为元素的不同组合和配置,就可以完成不同构架风格中所需连接器的功能,同时提供一些非功能属性.

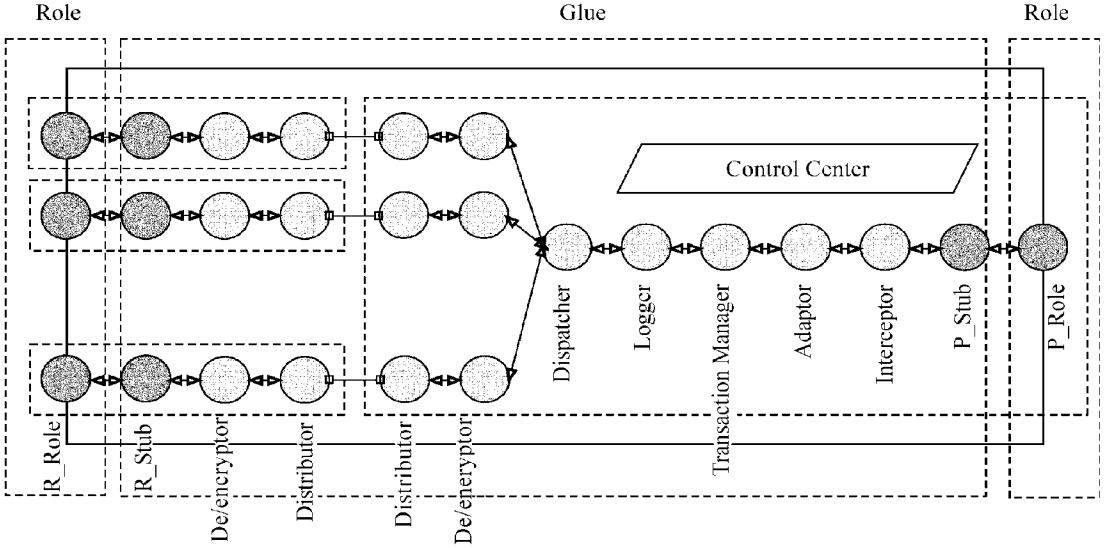


图 1　通用连接器模型

Fig. 1　A general connector model

通用的连接器模型如图 1 所示,其中各行为元素的具体作用分别为:

R_Role、P_Role:分别为服务构件和请求构件的接入点;

R_Stub、P_Stub:它将各式各样的接入方式同消息进行相互转化;

De/encryptor:对消息进行加密、解密,从而实现了安全性这个非功能属性;

Distributor:实现分布式这个非功能属性,图 1 中横向的虚线框可单独分布在一个地址空间;

Dispatcher:负责各接口间的消息分派;

Logger:记录消息日志;可以通过该行为元素动态的分析构件行为.

Transaction Manager:保证事务性这个非功能属性;

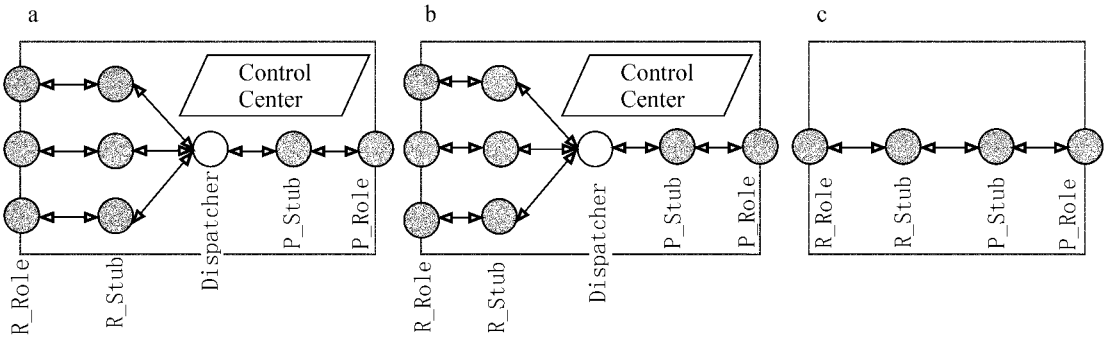
Adaptor:完成服务、请求构件间不匹配的转化;例如:完成参数匹配,类型转化;

Interceptor:决定是否阻塞该消息;或者返回该消息.

Control Center:注册各类交互信息,配置信息. 如注册各类消息的源、汇等信息等. 同时它为其他行为元素提供了统一的访问接口.

图 1 中模型为一个主从结构,P_Role 为主端,R_Role 为从端. 模型中各行为元素间通过通道相连,通道不形成回路. 一个通道包括两部分:一个正向信道,一个逆向信道. 图 1 中可以看出在通用连接器模型中 Glue 和 Role 分别对应于 Wright ADL 的 Glue 和 Role. Glue 内部通道是基于消息方式的,Glue 内部的行为元素应遵循统一的消息标准. Glue 内部的行为元素应该有缓存消息功能. Glue 与 Role 之间的通道是基于具体接入方式,当 Glue 两边接入方式不同时,就实现了不同 SA 风格构件的互连. 连接器的实例化是通过具体构件的 Port 替换连接器 Role 来完成[15].

基于文献[3,11]中对连接器的分类,本文将连接器分为 3 大类:(1)过程调用型;(2)消息型;(3)数据流型. 通过对通用连接器模型中行为元素的选择,可以实现以上 3 大类连接器类型,为易于操作,在各类连接器中不加入任何实现非功能属性的行为元素.



(a)接口连接式连接器模型;(b)C2 连接器模型;(c)管道 - 过滤器连接器模型

图 2 各种类型连接器模型

Fig. 2 Different connector types

(1)过程调用型:接口连接式风格模型如图 2a 所示.

R_Role,P_Role 作为过程、方法调用的行为规约接口;过程、方法和 Glue 内部消息一一对应. Control Center 中记录了 Glue 内部消息分派的配置信息. Dispatcher 是完成服务构件与请求构件之间方法匹配的实际操作者,当 Dispatcher 接受到一个消息,它查询 Control Center 来获取该消息的目的地,从而将消息发往具体的 Stub.

(2)消息型:对于消息型而言,不同的 SA 风格所需的消息连接器模型有所不同. C2 风

格连接器模型如图 2b 所示. R_Role,P_Role 作为构件发送、接收同步消息或者异步消息的行为规约. Control Center 中记录了 Glue 内部消息分派的配置信息. Dispatcher 是服务构件与请求构件之间消息分派的实际操作者,当 Dispatcher 接受到一个消息,它查询 Control Center 来获取该消息的目的地集,从而将消息发往的指定的 Stub.

(3)数据流型:管道 - 过滤器风格模型如图 2c 所示. R_Role,P_Role 作为两个数据区,以及在这两个数据区上的行为规约.

以上 3 种类型的连接器均可以自由的添加非功能属性行为元素,如添加 Adaptor 行为元素完成消息数据格式的转化. 如果在连接器模型的两端使用不同的接入方式,则可实现不同 SA 风格构件的交互.

3　形式化推导

本文基于 Wright ADL 和 CSP 中关于进程的描述,结合第 2 节提出的连接器模型和各种类型的连接器的形式化描述,对连接器模型进行形式化推导. 首先介绍 CSP 中形式化规约的相关知识,并给出基本的推导法则;然后对通用连接器中各行为元素行为进行推导.

3.1　**进程标记及定理**　本文采用 CSP[9]中的一个子集来描述连接器模型中的 role, port 及通用连接器模型中的行为元素,使用的基本记号如下:

进程(process):使用一系列事件序列来表示一个进程实体,例如上文中提到的各个 role、port 均是一个进程. STOP 是一个特殊进程,表示什么事都不做的进程.

字母表(alphabet):进程所能执行的事件的总和;特殊事件:事件"√"代表进程成功结束. 进程 P 的字母表表示为 αP.

前缀(prefix):设有一事件为 x,有一进程为 P,则另一进程先执行事件 x,然后按照进程 P 的说明进行动作;用 $(x \rightarrow P)$ 表示.

非确定性选择(non-deterministic choice):进程 A 或者按照 P 动作,或者按照 Q 动作,而这种选择是进程自发做出的. 用 $P \sqcap Q$ 表示.

确定性选择(deterministic choice):进程 A 或者按照 P 动作,或者按照 Q 动作,而这种选择是环境做出的. 用 $P \Box Q$ 表示.

通信(communication):使用二元对 $c.v$ 来描述的一个事件,c 是发生通信的通道(channel)的名字,v 是通道传递的消息(message). 使用"?"表示接受消息,"!"表示发送消息.

屏蔽(concealment):设 C 是进程 P 需要屏蔽掉的事件的集合,用 $P \setminus C$ 表示,它代表了一个似 P 动作的进程,只是在 C 中的任何事件的发生都被屏蔽掉了.

并发(parallel):进程 P 和 Q 的并发表示为 $P||Q$,它代表 P, Q 字母表中共有的事件需要同步执行,非共有事件不需要同步. 考虑连接器的形式化描述,可以将连接器定义为其中所有 role 和 glue 的并发.

进程标记(process labeling):标记为 l 的进程 P 表示为 $l:P$. 则进程中所有的事件也标记上与其所属进程相同的名字.

在进程表达式中 $\rightarrow$ 的优先级高于 $\Box, \sqcap, ||$ 操作符. 所以 $e \rightarrow f \rightarrow P || g \rightarrow Q = (e \rightarrow f \rightarrow P) || (g \rightarrow Q)$. 对于其他操作符,未定义显示的优先级,所以在使用时用括号来确定优先级.

有了 CSP 的基本记号,就可以对客观事物描述其动态行为,从而为推理、验证事物行为提供了强有力的工具. 使用 CSP 的基本记号和 Wright ADL 可以对各种类型的连接器进行形式化描述. 一个过程调用式连接器的形式化描述如下:

Connector

Role $R\_Role1 = invoke \rightarrow return \rightarrow R\_Role1 \Box \surd$;

//取 R_Role 的个数为 2

Role $R\_Role2 = invoke \rightarrow return \rightarrow R\_Role2 \Box \surd$;

Role $P\_Role = call1 \rightarrow result1 \rightarrow P\_Role \sqcap call2 \rightarrow result2 \rightarrow P\_Role \sqcap \surd$;

Glue glue = $P\_Role.call1 \rightarrow R\_Role1.invoke \rightarrow R\_Role1.return \rightarrow P\_Role.result1 \rightarrow glue \square$

$P\_Role.call2 \rightarrow R\_Role2.invoke \rightarrow R\_Role2.return \rightarrow P\_Role.result2 \rightarrow glue \square \surd$

限于篇幅,对 C2 连接器模型、管道 - 过滤器连接器模型从略. 同时,连接器还可以连接不同的接入方式,如连接器一端是过程调用式,另一端时消息式,其描述如下:

ConnectorRole $R\_Role = left'?msg \rightarrow left!msg \rightarrow R\_Role \square \surd$; // $left$ 为正向通道, $left$ 为逆向通道

Role $P\_Role = call \rightarrow result \rightarrow P\_Role \prod \surd$;

Glue $glue = P\_Role.call \rightarrow R\_Role.left'?msg \rightarrow R\_Role.left!msg \rightarrow P\_Role.result \rightarrow glue \square \surd$;

同时本文还使用文献[9]中通信这一章的基本定理:

定 理 1 设 P 和 Q 为进程, c, d 为不同通道, P 用它输出, Q 用它输入. 则

$$(c!v \rightarrow P) \parallel (c?x \rightarrow Q(x)) = c!v \rightarrow (P \parallel Q(v)) \quad (1)$$

$$(c!v \rightarrow P) \parallel (d?x \rightarrow Q(x)) = c!v \rightarrow P \quad (2)$$

定 理 2 设进程 $R = P \rightarrow c!v \rightarrow c?v \rightarrow Q(v) \rightarrow R$; $S = P \rightarrow c!v \rightarrow S$; $T = c?x \rightarrow Q(x) \rightarrow T$; 且 $\alpha P \cap \alpha Q = \varnothing$; 则

$$R = S \parallel T \quad (3)$$

证明:可由定理 1 中(1)式得到.

3.2 **推导行为元素语义** 上面对各种类型的连接器都使用了 Wright ADL 进行描述, Role 代表了连接器期望接入构件 Port 的行为, Glue 则代表了所有 Role 间的行为规约. 从图 1 可以看出,通用连接器模型中的 Role 只包含一种行为元素(role 行为元素),而 Glue 则包含了 stub 行为元素和其他非必需行为元素. 因此,只须对 Glue 推导其中包含的行为元素的语义.

根据上面各种连接器类型的形式化描述,可以将 Glue 的语义按如下的方式描述出来:

Glue $glue = M1 \square M2 \square \cdots \square Mn$

(记作 $\prod_{i=1}^{n}{}^{\square} Mi$)

其中进程 $M1, M2 \cdots Mn$ 为 glue 可能的行为序列,表明了 glue 所有可能的行为的协作方式. 这里可以认为 $M1, M2, \cdots, Mn$ 间的行为元素是不相交的,即 $\alpha Mi \cap \alpha Mj = \varnothing (1 \leq i, j \leq n$ 且 $i \neq j)$;可以将 Mi 的行为序列定义为 $Mi = Si(\{\rightarrow\}, \sum i)$.

其中 $\rightarrow$ 为前缀操作符; $\sum i = \alpha(R1:role) \cup \alpha(R2:role) \cup \cdots \cup \alpha(Rn:role) \cup \surd\}$;函数 $Si(OP, A)$ 表示将 A 中的事件用 OP 中的操作符进行序列化.

通用连接器中的 glue 形式化描述出来后,需要按以下步骤来进行扩充、分解. 从而得到相应行为元素的语义行为.

1. 扩 充:对 glue 中 Mi 的行为进行扩充,任意选取 Mi 事件序列中两个连续的事件 $R1.a$、$R2.b$,分 3 种情况进行扩充.

(1) 如行为元素 $R1 = R2$ 则无需扩充.

(2) 如果行为元素 $R1 \neq R2$,且 $R1$ 与 $R2$ 间不存在其他行为元素,则在 $R1.a$、$R2.b$ 之间添加信道事件 $C_{R1R2}!m$, $C_{R1R2}?m$. 从而扩充成 $R1.a \rightarrow C_{R1R2}!m \rightarrow C_{R1R2}?m \rightarrow R2.b$ (C_{R1R2}是连接从 $R1$ 到 $R2$ 的信道).

(3) 如果行为元素 $R1 \neq R2$,且 $R1$ 与 $R2$ 间存在其他行为元素,则在 $R1.a$、$R2.b$ 之间添加通道事件 $C1_{R1R2}!m$, $C1_{R1R2}?m$, $C2_{R1R2}!m$, $C2_{R1R2}?m$ 以及待分解的进程 S. 从而扩充成 $R1.a \rightarrow C1_{R1R2}!m \rightarrow C1_{R1R2}?m \rightarrow S \rightarrow C2_{R1R2}!m \rightarrow C2_{R1R2}?m \rightarrow R2.b$ ($R1$ 经信道 $C1$, $C2$ 到 $R2$ 相连, S 是 $C1$ 与 $C2$ 之间行为元素的进程).

通过进行以上 3 个步骤, Mi 可扩充为: $Mi = S'i(\{\rightarrow\}, \sum i \cup \sum i')$; $\sum i'$为所有添加信道上的事件和添加进程上的事件的集合.

扩充后的 Mi 本身的语义并没有改变,扩充只是将行为元素的交互变成基于消息机制的基础上,同时将 Glue 内部的行为元素表达出来. 扩充后 Mi 表示的语义信息更加丰富了.

同时扩充后的 Mi 一样保证 $\alpha Mi \cap \alpha Mj = \emptyset$; $(1 \leq i, j \leq n$ 且 $i \neq j)$.

2. 分　解: 对扩充后的 Mi 进行分解,按照以下步骤进行.

(1) 选定 Mi 中一个特定的行为元素 R;

(2) 将 Mi 中属于 R 的事件从 Mi 中选择出来加入集合 E;同时这些事件从 Mi 中删除;

(3) 将 Mi 中与 R 相关的通道上相关的事件(相关事件包括相对 R 的正向信道的发送消息事件和逆向信道的接收消息事件)加入集合 E;同时这些事件从 Mi 中删除;

(4) 将 E 中的事件按照原来在 Mi 中的顺序使用后缀操作符"→"重新组成新进程 Mij;

(5) 清空集合 E,如果 Mi 不存在确定的行为元素,结束;否则跳转到步骤(1).

经过以上步骤, Mi 可分解成多个行为进程: $Mi = S'i(\{\rightarrow\}, \sum \cup \sum')$,分解出行为进程, $Mi1 \cdots Mim$ (m 为 Mi 包含的行为元素的个数); $Mij = S'ij(\{\rightarrow\}, \sum ij)$ $(1 \leq j \leq m)$ ($\sum ij$ 即为步骤(2),(3)选择出来的事件集合 E).

可以得到一个推论:

推　论 1　$Mi = (\prod_{j=1}^{m} \| Mij) \| MSi$; $(1 \leq i \leq n$; $MSi = Mi \setminus (\sum i1 \cup \sum i2 \cup \cdots \cup \sum im))$

证明: 根据 Mi 分解的步骤,可得 $\sum ij \cap \sum ik = \emptyset$ $(1 \leq j, k \leq m$ 且 $j \neq k)$,即 $\alpha Mij \cap \alpha Mik = \emptyset$.

根据 4.1 节中定理 2

$$Mi = Mi1 \| (Mi \setminus \sum i1) =$$
$$Mi1 \| (Mi2 \| (Mi \setminus (\sum i1 \cup \sum i2))) =$$
$$\cdots$$
$$Mi1 \| Mi2 \cdots \| Mim \| (Mi \setminus (\sum i1 \cup \sum i2 \cup \cdots \cup \sum im)) =$$
$$(\prod_{j=1}^{m} \| Mij) \| MSi$$

根据 $Mi \setminus (\sum i1 \cup \sum i2 \cup \cdots \cup \sum im)$ 即为 MSi.

3. 提　取: 完成对 glue 的分解后,须对 Glue 中各行为元素的行为进行提取. 定义集合 Fi 是 Mi 的行为元素集合,集合 F 是 glue 的行为元素集合. 可得 $F = F1 \cup F2 \cdots \cup Fn$.

设 $F = \{f1, f2 \cdots fk\}$,行为元素的顺序是按照从通用连接器模型主端开始进行广度遍历所得的顺序,并设

$$Mifj = \begin{cases} Mik & \text{当 } Mik \text{ 中的行为元素为 } fj \\ \text{STOP} & \text{其　他} \end{cases}$$

从而可以改写 Mi 分解表达式的形式:

$$Mi = (\prod_{j=1}^{m} \| Mij) \| MSi =$$
$$(\prod_{j=1}^{k} \| Mifj) \| MSi$$

(根据 $P \| STOP = P$[8])

为提取 glue 中行为元素的行为,首先证明两个引理:

引　理 1　设 $Mjfp \cap Mjfq \neq \emptyset$; 则 $Mjfp \| (\prod_{i=1}^{n} \square Mifq) = Mjfp \| Mjfq$.

证明:因为 $\alpha Mi \cap \alpha Mj = \emptyset$ $(1 \leq i, j \leq n$ 且 $i \neq j)$

所以 $\alpha Mifp \cap \alpha Mjfq = \emptyset$ $(1 \leq i, j \leq n$ 且 $i \neq j$; $1 \leq p, q \leq k)$

又因为 $\alpha(\prod_{i=1}^{n} \square Mifq) = \bigcup_{i=1}^{n} \alpha Mifq$; 所以 $\alpha(\prod_{i=1}^{n} \square Mjfp) \cap \alpha Mifq \neq \emptyset$;所以 $Mjfp \| (\prod_{i=1}^{n} \square Mifq) = Mjfp \| Mjfq$.

引　理 2　$(\prod_{i=1}^{n} \square (\prod_{j=1}^{k} \| Mifj)) = (\prod_{j=1}^{k} \| (\prod_{i=1}^{n} \square Mifj))$

证明:

$$(\prod_{j=1}^{k} \| (\prod_{i=1}^{n} \square Mifj)) =$$
$$(\prod_{t=1}^{n} \square Mtf1) \| (\prod_{j=2}^{k} \| (\prod_{i=1}^{n} \square Mifj)) =$$
$$\prod_{t=1}^{n} \square (Mtf1 \| (\prod_{j=2}^{k} \| (\prod_{i=1}^{n} \square Mifj))) =$$
$$\prod_{t=1}^{n} \square (Mtf1 \| (\prod_{i=1}^{n} \square Mif2) \| (\prod_{j=3}^{k} \| (\prod_{i=1}^{n} \square Mifj))) =$$
$$\prod_{t=1}^{n} \square ((Mtf1 \| Mtf2) \| (\prod_{j=3}^{k} \| (\prod_{i=1}^{n} \square Mifj))) =$$

(根据引理 1)

$$\cdots = (\prod_{i=1}^{n} \square (\prod_{j=1}^{k} \| Mifj))$$

推　论 2　$glue =$

$$(\prod_{j=1}^{k} \| (\prod_{i=1}^{n} \square Mifj)) \| (\prod_{i=1}^{n} \square MSj)$$

证明: $glue = \prod_{i=1}^{n}{}^{\square} Mi = \prod_{i=1}^{n}{}^{\square}((\prod_{j=1}^{k}{}^{\|} Mifj) \| MSi) = \prod_{i=1}^{n}{}^{\square}(\prod_{j=1}^{k}{}^{\|} Mifj \| \prod_{k=1}^{n}{}^{\|} MSk =$

(根据定理 1 中(2)式)

$(\prod_{i=1}^{n}{}^{\square}(\prod_{j=1}^{k}{}^{\|} Mifj)) \| (\prod_{k=1}^{n}{}^{\|} MSk) =$

(提取公共项)

$(\prod_{j=1}^{k}{}^{\|}(\prod_{i=1}^{n}{}^{\square} Mifj)) \| (\prod_{k=1}^{n}{}^{\|} MSk)$

(根据引理 2)

得证.

由推论 2 可以得到行为元素的行为: $Fi = \prod_{j=1}^{n}{}^{\square} Mjfi$ (Fi 为行为元素 fi 的形为),而 $\prod_{k=1}^{n}{}^{\|} MSk$ 就是待分解的行为元素的行为规约,当 $\prod_{k=1}^{n}{}^{\|} MSk \neq \emptyset$ 时,可以对 $\prod_{k=1}^{n}{}^{\|} MSk$ 重新扩充、分解、提取出相应的行为元素的功能行为. 同理可得其他行为元素的功能行为.

3.3 **例 子** 对图 2 接口连接式连接器分解出行为元素的功能,由 3.1 节得到的形式化描述. 得出

$M1 = P\_Role.call1 \rightarrow R\_Role1.invoke \rightarrow R\_Role1.return \rightarrow P\_Role.result1 \rightarrow glue$

$M2 = P\_Role.call2 \rightarrow R\_Role2.invoke \rightarrow R\_Role2.return \rightarrow P\_Role.result2 \rightarrow glue$

$M3 = \surd$

1. **扩 充**

$M1 = P\_Role.call1 \rightarrow C_{PS}! \ m1 \rightarrow C_{PS}? \ m1 \rightarrow S1 \rightarrow C_{SR1}! \ m1 \rightarrow C_{SR1}? \ m1 \rightarrow R\_Role1.invoke \rightarrow R\_Role1.return \rightarrow C_{R1S}! \ m1 \rightarrow C_{R1S}? \ m1 \rightarrow S2 \rightarrow C_{SP}! \ m1 \rightarrow C_{SP}? \ m1 \rightarrow P\_Role.result1 \rightarrow glue$

同理对 $M2$ 进行, $M3$ 无行为元素,所以无需扩充.

2. **分 解** 设集合 P 是行为元素 P_Role 中的事件及其相关事件的集合, R 是行为元素 R_Role 中的事件及其相关事件的集合. 则

$P = \{P\_Role.call1, C_{PS}! \ m1, C_{SP}? \ m1, P\_Role.result1, \surd\}$

$R = \{C_{SR1}? \ m1, R\_Role1.invoke, R\_Role1.return, C_{R1S}! \ m1, \surd\}$

$M11 = P\_Role.call1 \rightarrow C_{PS}! \ m1 \rightarrow C_{SP}? \ m1 \rightarrow P\_Role.result1 \rightarrow (glue \setminus (\alpha glue - P))$

$M12 = C_{SR1}? \ m1 \rightarrow R\_Role1.invoke \rightarrow R\_Role1.return \rightarrow C_{R1S}! \ m1 \rightarrow (glue \setminus (\alpha glue - R))$

$MS1 = C_{PS}? \ m1 \rightarrow S1 \rightarrow C_{SR1}! \ m1 \rightarrow C_{R1S}? \ m1 \rightarrow S2 \rightarrow C_{SP}! \ m1 \rightarrow (glue \setminus (P \cup R))$

$M1 = M11 \| M12 \| MS1$

同理对 $M2, M3$ 进行分解.

3. **提 取**

$P\_Stub = M11 \square M21 \square M31$

$= P\_Role.call1 \rightarrow C_{PS}! \ m1 \rightarrow C_{SP}? \ m1 \rightarrow P\_Role.result1 \rightarrow (glue \setminus (\alpha glue - P)) \square$

$P\_Role.call2 \rightarrow C_{PS}! \ m2 \rightarrow C_{SP}? \ m2 \rightarrow P\_Role.result2 \rightarrow (glue \setminus (\alpha glue - P)) \square \surd$

$= P\_Role.call1 \rightarrow C_{PS}! \ m1 \rightarrow C_{SP}? \ m1 \rightarrow P\_Role.result1 \rightarrow P\_Stub \square$

$P\_Role.call2 \rightarrow C_{PS}! \ m2 \rightarrow C_{SP}? \ m2 \rightarrow P\_Role.result2 \rightarrow P\_Stub \square \surd$

由于递归,所以 $glue \setminus (\alpha glue - P) = P\_Stub$.

同理可对其他的行为元素进行提取,并可对待分解的行为再进行以上步骤.

4 结论及下一步工作

软件工程的一个重要挑战是对软件架构协议进行形式化描述,本文着眼于连接器,提出了通用连接器模型,对不同连接器类型进行描述;而且可以自定义连接器并加入非功能属性,从而实现不同 SA 风格构件的组装. 在 Wright ADL 和 CSP 的基础上完成了对通用连接器模型的形式化描述;给出了对通用连接器模型中

的行为元素形式化规约过程;可从行为元素形式化规约证连接器的整体行为的合法性.

进一步的研究工作包括通用连接器模型的基础上进行相关 CASE 工具的开发,连接器代码的自动生成,并进行更多的实例建模和分析.

References

[1] Garlan D, Shaw M. An introduction to software architecture. Ambriola V, Tortoreads G. Advances in Software Engineering and Knowledge Engineering, Vol I. New Jersey: World Scientific Publishing Company, 1993: 1～39.

[2] Shaw M, Garlan D. Software architecture: Perspectives on an emerging discipline. New Jersey: Prentice Hall, Inc, 1996.

[3] Nikunj R M, Nenad M. Understanding software connector compatibilities using a connector taxonomy. Proceedings of First Workshop on Software Design and Architecture, India Bangalore, 2002.

[4] Tomas B, Frantisek P. Scalable-element based connectors. Proceedings of software engineering research and applications. USA: San Francisco, 2003.

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

[6] David C L, James V. An event-based architecture definition language. IEEE Transactions on Software Engineering, 1995, 21(9): 717～734.

[7] Shaw M, DeLine R, Klein D, *et al*. Abstractions for software architecture and tools to support them. IEEE Transactions on Software Engineering, 1995, 21(4): 314～335.

[8] Ma X X, Zhang X L, Lv J. Description and implementation of dynamic software architectures: A reflective approach. Journal of Nanjing University (Natural Sciences), 2004, 40(2):146～155. (**马晓星,张小蕾,吕　建. 自省的动态软件体系结构描述与实现. 南京大学学报(自然科学)**,2004, 40(2):146～155).

[9] Hoare C A R. Communicating sequential processes. New Jersey: Prentice Hall, 1985.

[10] Shaw M. Procedure calls are the assembly language of system interconnection: Connectors deserve first-class status. Proceedings of the Workshop on Studies of Software Design, 1993.

[11] Adel S, Mourad O, Tahar K. Software Connectors reuse in component-based systems. IEEE International Conference, 2003: 543～550.

[12] Bridget S, David G. A compositional formalization of connector wrappers. 25th International Conference on Software Engineering, 2003: 374～384.

[13] Bridget S, David G. A compositional approach for constructing connectors. IEEE/ IFIP(International Federation for Information Processing Conference), 2001: 148～157.

[14] Singh G, Mao Z. Structured design of communication protocols. IEEE International Conference on Distributed Computing Systems, 1996: 360～367.

[15] Robert A, David G. Formalizing architectural connection. 16th International Conference on Software Engineering, 1994: 71～80.

[16] Ren H M, Qian L Q. Research on component composition and its formal reasoning. Journal of Software, 2003, 14(6):1 066～1 074. (**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):1 066～1 074).