

# 软件体系结构中接口连接规则研究

何频捷, 李 伟, 李玉龙, 李长云

(湖南工业大学, 湖南 株洲 412008)

**摘 要:** 基于体系结构的软件开发方法以构件作为最小组装粒度, 构件通过其自身定义的接口与外界进行交互。连接件作为一种特殊的构件, 它主要负责管理构件之间信息的交换。构件与连接件之间通过接口进行数据的传递, 因此, 如何从体系结构层次上定义构件及连接件接口的交互规则是构件组装的关键所在。在现有动态体系结构描述框架的基础上, 定义了端口以及通道的连接规则, 并结合实际的案例, 采用动态体系结构描述语言 D-ADL 进行描述, 具体说明了这些规则的应用。从而为系统行为的形式化分析、验证和仿真奠定了基础。

**关键词:** 软件体系结构; 接口; D-ADL; 连接规则

**中图分类号:** TP311.52

**文献标识码:** A

**文章编号:** 1673-629X(2008)08-0121-05

## Research on Interface Connecting Rule in Software Architecture

HE Pin-jie, LI Wei, LI Yu-long, LI Chang-yun

(Hunan University of Technology, Zhuzhou 412008, China)

**Abstract:** Components as the smallest granularity are used to composite the software systems in architecture-based software developing. And they communicate each other by their interfaces. As a special component, connectors also communicate with components by interface. Therefore, how to define the communication rule among components' interfaces on the level of architecture is the key to composite components. On the basis of dynamic architecture description framework, the joint rule among ports and channels was defined in this paper. At the same time, these rules are applied to an example and described with D-ADL. These would build up foundation of formal analyzing, examining and emulating the system's behaviours.

**Key words:** software architecture; interface; D-ADL; connecting rule

## 0 引言

目前, 基于体系结构的软件开发方法越来越被人们所关注。它以构件作为最小的组装粒度, 通过大量复用经过实践的体系结构以及标准化构件来提高软件开发效率以及软件质量, 与其它几种软件开发方法相比有着明显的差别。构件通过接口来定义同外界的信息传递以及所承担的系统责任, 构件接口包括了构件同周围环境的全部交互内容, 也是构件同外界唯一的交互途径<sup>[1]</sup>。

根据被广泛认同的 Garlan & Shaw 模型<sup>[2]</sup>, 即软件体系结构 = {构件, 连接件, 约束}, 其中构件作为一

个独立的计算逻辑实体, 可以是一个程序模块, 也可以是一个独立的系统。连接件可以被看作一种特殊的构件, 可以是过程调用、管道、远程过程调用等, 用于表示构件之间的相互作用。约束一般为对象连接时的规则。构件通过接口来与外界发生交互, 而接口被定义成一组端口的集合。因此, 在基于体系结构的软件开发中, 如何保证物理构件之间数据传递的一致性、正确性以及避免进程的死锁等都是需要考虑的问题, 而定义体系结构中接口的连接规则成为解决问题的关键。

## 1 D-ADL 语言

在文献[3]中提出了一种动态体系结构描述语言 D-ADL, 它将高阶多型(演算作为计算行为和动态行为的统一语义基础, 能显式、独立地表示动态行为。在 D-ADL 中, 构件、连接件和体系结构风格被形式化为高阶  $\pi$  演算中的抽象(abstraction)类型, 构件计算、连接件协调活动、体系结构配置以及演化策略都被模型化为 behaviour, 对应  $\pi$  演算中的进程(process)元素, 构件和连接件的交互点被直接抽象为  $\pi$  演算中的通道

收稿日期: 2007-11-26

基金项目: 国家自然科学基金(60773110); 湖南省教育厅优秀青年项目(06B023); 湖南省学位与研究生研究课题(06B28); 湖南工业大学博士基金

作者简介: 何频捷(1982-), 男, 湖南新邵人, 硕士研究生, 研究方向为软件体系结构、软件框架; 李长云, 博士, 教授, 硕士生导师, 研究方向为软件体系结构、软件自动化。

(channel),这使得 D-ADL 具有坚实的语义基础,便于模型检查、执行和求精。

一方面,D-ADL 遵循 Wright 等给出的已被广泛认同的 SA 描述框架,围绕 SA 实体如构件、连接件和配置等进行 SA 建模,如图 1 所示;另一方面,D-ADL 将高阶多型  $\pi$  演算作为行为语义基础,凭借高阶  $\pi$  演算描述动态系统的特征,D-ADL 允许构件、连接件和配置产生变更,并使得对 SA 的自动化分析成为可能。

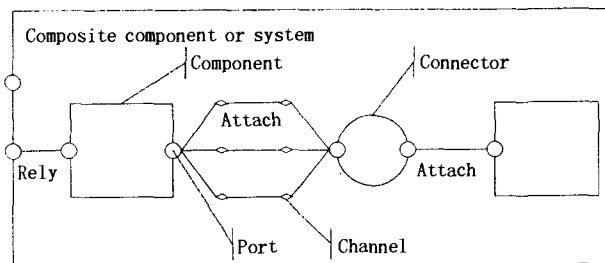


图 1 D-ADL 描述框架

在 D-ADL 中,构件主要完成系统的计算功能,作为一个封装的实体,仅通过其接口与外部环境交互,而构件的接口由一组端口(port)组成。每个端口由一组通道(channel)组成,因此端口可看成通道集合。引入端口让构件的交互功能结构化、局部化,从而方便对其进行分析。通道是最基本的交互点,它的角色就是在 SA 元素间提供通信渠道,构件通过通道发送或接收值。

连接件(connector)是一种特殊的构件,旨在建立构件间的交互以及支配这些交互规则。连接件依据接口和路由行为(routing)描述。显然,连接件的接口也由一组端口构成,连接件的端口同样由一组通道组成。路由行为是计算行为的一个变种,指示构件间的通信轨迹和执行交互规则及约束。

复合构件(复合连接件)由外部端口和内部元素构成。应用系统本身可被视作一个特殊的复合构件。复合构件(复合连接件)的内部元素由一系列成员构件和成员连接件组成,它们通过配置链接在一起。一个构件不能直接连接到另一个构件,构件之间通信和协作只能通过连接件进行。

D-ADL 以高阶多型  $\pi$  演算作为其语义基础,具有天然的对结构和行为演化的支持能力。高阶多型  $\pi$  演算除了能传递通道名外,同时还能将对象名字作为进程进行传递,从而使得 D-ADL 不仅可以描述结构不断变化的并发系统,同时还直接支持系统行为的预定义动态变更。

由于 D-ADL 中的行为规约方法是高度形式化的,能够进行系统行为自动分析,因此为非预设的动态演化提供了间接支持。

## 2 接口的连接规则

本节主要在 D-ADL 描述框架基础上定义了接口的连接规则,并以实例说明了这些规则的应用,同时对连接规则的定义,也进一步完善了 D-ADL 的语法规格。

### 2.1 基本概念定义

在 D-ADL 描述框架中,接口由端口组成,端口则由通道组成,并且通道被单独提取出来进行显式的定义,它在逻辑上对应高阶多型  $\pi$  演算中的通道。为了更好地对构件的功能属性进行形式化描述,将通道的模式分为三种:in、out 和 inout,分别代表输入、输出和输入输出模式。在体系结构中,构件或连接件的类型信息不仅包括构件向外提供的服务接口,还包括构件要求外部提供的服务。构件向外提供的服务通道用 out 表示,构件要求外部提供的服务用 in 表示。通道是构件与连接件之间通信的最小单元。

为了提高动态体系结构模型中接口连接的规模,引入端口的相连。当两个端口进行连接时,通道在模型视图中并不可见,只是隐性地引用了通道的连接规则。

在定义接口之间的连接规则之前,首先给出 D-ADL 一些基本定义。

定义 1 一个端口  $F = \langle E^I, E^O, E^{IO} \rangle$ ,通常由以下几部分组成:

- \* 一个模式类别为 in 的通道集合  $E^I$ 。
- \* 一个模式类别为 out 的通道集合  $E^O$ 。
- \* 一个模式类别为 inout 的通道集合  $E^{IO}$ 。

为方便表示端口的通道集,用上标表示该通道所属的端口名,下标表示该通道的模式类别。例如:端口  $F$  上模式类别为 in 的通道集合用  $E_F^I$  表示。

定义 2 一个通道  $\alpha = \langle P, M \rangle$ ,通常由以下几部分组成:

- \* 一个参数序列  $P$ ,其中所有参数都按一定顺序组合,有对应的参数类型。
- \* 一个通道模式  $M$ , $M$  属于集合  $\{\text{in}, \text{out}, \text{inout}\}$ 。

为方便表示具体通道的参数序列,用下标表示通道名。例如:通道  $\alpha$  的参数序列用  $p_\alpha$  表示。

定义 3 端口的可组合性,composable。给定两个端口  $F$  和  $G$ ,如果  $E_F^O \cap E_G^I = \emptyset$  成立,则称端口  $F$  和  $G$  是可组合的,用  $F \parallel G$  表示。

构件的组合实质就是接口的组合,在软件体系结构中,把接口定义为端口的集合。因此,构件的组合最终表现为端口的组合。每一个端口的输出将成为其它端口的输入,在两个或多个端口进行组合时,首先必须

保证它们不存在相同的输出通道,避免与其它端口相连时造成不一致。

定义4 端口的兼容性,compatibility。考虑端口的兼容性有两种情况:

1) 给定两个端口  $F$  和  $G$ ,如果所有  $F$  的输入都是  $G$  的输出,即  $E_F^I \subseteq E_G^O$ ,反之亦然;

2) 给定两个端口  $F$  和  $G$ ,其中  $F$  的部分输入是  $G$  的输出,但  $F$  的其它输入可由运行环境来提供,那么称  $F$  和  $G$  是相容的,用  $F \sim G$  表示。

定义5 对偶通道。设有两个端口  $F$  和  $G$ ,以及两个通道  $\alpha$  和  $\beta$ ,其中  $\alpha \in E_F^I, \beta \in E_G^O$ ,且  $p_\alpha = p_\beta$ ,那么,称通道  $\alpha$  和  $\beta$  为对偶通道,用  $\alpha \Leftrightarrow \beta$  表示。

其中  $p_\alpha = p_\beta$  表示通道  $\alpha$  和  $\beta$  的参数数量,次序以及类型都完全一致。

定义6 对偶端口。设有两个端口  $F$  和  $G$ ,其所有通道都一一对偶,即  $(E_F^I \Leftrightarrow E_G^O) \wedge (E_F^O \Leftrightarrow E_G^I) \wedge (E_F^IO \Leftrightarrow E_G^OI)$ ,那么,称端口  $F$  和  $G$  为对偶端口,用  $F \Leftrightarrow G$  表示。

其中符号  $\wedge$  表示与关系,即所有条件都需成立。

复合构件(连接件)的端口包括两类:一类是用于演化行为的输入输出,它主要与运行环境进行交互,完成系统的预定义演化;另一类是被映射到成员构件上的端口(通道),这些端口(通道)主要用于计算行为的输入输出。

定义7 端口映射。设有两个端口  $F$  和  $G$ ,其中  $F$  为复合构件的端口, $G$  为其成员构件的端口。端口  $F$  与  $G$  相连称为端口映射,用  $F \rightarrow G$  表示。

定义8 通道映射。设有两个通道  $\alpha$  和  $\beta$ ,其中  $\alpha$  为复合构件端口上的通道, $\beta$  为其成员构件端口上的通道。通道  $\alpha$  与  $\beta$  相连称为通道映射,用  $\alpha \rightarrow \beta$  表示。

定义9 端口连接。设有两个端口  $F$  和  $G$ ,其中  $F$  为构件的端口, $G$  为连接件的端口。端口  $F$  与  $G$  相连称为端口连接,用  $F \leftrightarrow G$  表示。

定义10 通道连接。设有两个通道  $\alpha$  和  $\beta$ ,其中  $\alpha$  为构件端口上的通道, $\beta$  为连接件端口上的通道。通道  $\alpha$  和  $\beta$  相连称为通道连接,用  $\alpha \leftrightarrow \beta$  表示。

## 2.2 连接规则

在基于体系结构的软件开发过程中,开发什么样的构件,以及如何进行构件组装等一系列工作都必须在端口和通道的连接规则下进行。同时,D-ADL 是一种动态体系结构描述语言,它除了需要描述构件的静态配置与组装外,还包括构件的动态绑定与演化。为了确保系统能沿着预设的功能进行演化,还需要定义端口和通道的连接规则。结合上一节的定义,本节将给出端口和通道的连接规则。由于本节侧重于规则

的讨论,因此实例的描述中所涉及到的 D-ADL 语法规则可参见文献[3]。

规则1 有构件  $C1$ 、 $C2$  以及连接件  $C0$ ,其中端口  $\gamma$  属于  $C1$ ,由  $\alpha$  和  $\mu$  通道组成,端口  $\Psi$  属于  $C0$ ,由  $\beta$  和  $\sigma$  通道组成,那么,宏观上,构件  $C1$  和  $C2$  必须通过连接件  $C0$  相连;微观上,相连的两组通道  $\{\alpha, \beta\}$  和  $\{\mu, \sigma\}$  必须为对偶通道,即  $\{\alpha, \beta\} \Leftrightarrow \{\mu, \sigma\}$ 。

规则2 有构件  $C1$ 、 $C2$  以及连接件  $C0$ ,其中端口  $\gamma$  属于  $C1$ ,端口  $\Psi$  属于  $C0$ ,那么,宏观上,构件  $C1$  和  $C2$  必须通过连接件  $C0$  相连;微观上,相连的两个端口  $\gamma$  和  $\Psi$  必须为对偶端口,即  $\gamma \Leftrightarrow \Psi$ 。

在构件  $C1$  和连接件  $C0$  进行实际交互时,考虑到可能有连接时端口兼容(即端口  $\gamma$  的所有通道在  $\Psi$  上都能找到对应的对偶通道,反之亦然)的情况,因此可以放宽接口的连接条件以保证组合后构件服务的完整性。故对规则2的扩展如下:

规则2\* 有构件  $C1$ 、 $C2$  以及连接件  $C0$ ,其中端口  $\gamma$  属于  $C1$ ,端口  $\Psi$  属于  $C0$ ,那么,宏观上,构件  $C1$  和  $C2$  必须通过连接件  $C0$  相连;微观上,相连的两个端口  $\gamma$  和  $\Psi$  必须是兼容的,即  $\gamma \sim \Psi$ 。

通过规则1和规则2、2\*,可以描述所有构件之间的连接,即两个构件必须通过连接件来完成,构件是一个计算单元或数据存储,它所有的交互都需要通过接口来完成,在 D-ADL 中,通道是构件与连接件交互的最小单位,通道的模式决定了其所提供的服务功能以及所需外部提供的功能。我们统一将信息的交互提交给连接件来处理,赋以它们明确的责任与分工,从而有利于对其进行形式化的描述和推导。

下面以订单支付系统为例说明端口连接规则的应用,简化后的系统仅包括订单的支付流程。其体系结构如图2所示。

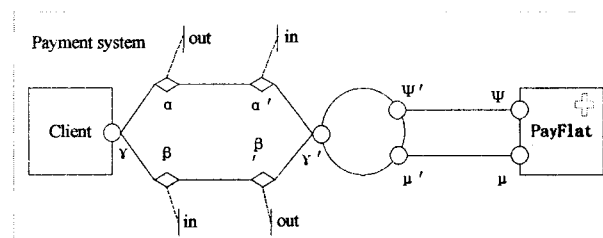


图2 订单支付系统体系结构

整个支付过程的数据处理流程为:客户(Client)首先经过通道  $\alpha$  发送订单号和支付方式,接着由通道  $\alpha'$  接收数据,并由连接件负责数据处理,然后通过端口  $\Psi'$  将数据传送给端口  $\Psi$ ,最后交由支付平台(PayFlat)进行处理。处理完毕后通过端口  $\mu$  发出确认信息,经过  $\mu'$  端口传送给连接件进行处理,最后由连接件经过通道  $\beta'$  发送出去,并由通道  $\beta$  接收确认信息,并传送给

客户。

该实例中的通道  $\alpha$  如果需要和  $\alpha'$  相连,则它们必须是对偶通道。在 D-ADL 中,通道  $\{\alpha, \alpha'\}$  和通道  $\{\beta, \beta'\}$  的描述如下:

PortType Tcaccess is free( $\alpha$ :ChannelType T $\alpha$  is out(orderform:string, orderform:int),  $\beta$ :ChannelType T $\beta$  is in(confirminfo:string))

PortType Tcon is free( $\alpha$ :ChannelType T $\alpha$  is in(orderform:string, orderform:int),  $\beta$ :ChannelType T $\beta$  is out(confirminfo:string))

其中 Tcaccess 是客户端端口定义,通道  $\alpha$  需要发送订单号和支付方式,通道  $\beta$  需要接收确认信息。Tcon 连接件的端口类型,通道  $\alpha'$  需要接收订单号和支付方式,  $\beta'$  通道需要发送确认信息。根据规则 2, 只有对偶的两个端口才能直接相连。因此在该系统中,引入对偶端口  $\{\Psi, \Psi'\}$  和  $\{\mu, \mu'\}$ , 其 D-ADL 描述如下:

Port  $\Psi'$  is out (A)

Port  $\mu'$  is in (B)

Port  $\Psi$  is in (A')

Port  $\mu$  is out (B')

其中 A, B, A', B' 是通道集合, A 和 A' 为一对对偶通道集, B 和 B' 是一对对偶通道集。

规则 3 有构件 C1、C2, 其中端口  $\gamma$  属于 C1, 端口  $\zeta$  属于 C2, 那么, 构件 C1 与 C2 组合成一复合构件时, 其两个构件的端口  $\gamma$  和  $\zeta$  必须是可组合的, 即  $\gamma \parallel \zeta$ 。

在 D-ADL 中, 规定两个或多个原子构件可以进行组合, 组合的方式有三种:

1) 通过与连接件相连进行组合;

2) 不通过连接件, 原子构件之间直接进行组合;

3) 混合式: 多个构件进行组合, 其组合方式包括 1) 和 2) 两种。其中组合方式 1) 是通过连接件进行组合, 因此它必须满足规则 1 和 2; 组合方式 2) 没有涉及到端口的直接相连, 但必须满足端口的可组合性, 从而保证构件所提供服务的完整性和一致性。组合方式 3) 是方式 1) 和 2) 的混合模式, 它除了需要满足规则 1 和 2 外, 还需要满足端口的可组合性条件。组合后的构件称为复合构件。

规则 4 对于每一个端口映射, 外部端口只能映射到一个成员构件端口上; 对于通道映射, 外部通道只能映射到一个成员构件端口的通道上。

规则 5 对于每一个端口相连, 一个端口只允许最多连接一个对偶端口; 对于通道相连, 一个通道只允

许最多连接一个对偶通道。

规则 4 和规则 5 的定义表面上削弱了语言的表达能力, 实际上并不如此, 为了说明 D-ADL 的复合构件组合与其它复合构件组合的不同之处, 首先给出了没有使用规则 2 和规则 4 的复合构件的组合方式, 如图 3 所示, 在该订单支付系统中的 PayFlat 构件为复合构件, 它主要用于完成在线支付功能。该复合构件包括了三个成员构件: 信用卡(图中 A1), 支付宝(图中 A2) 以及数据处理构件(图中 A3)。其中构件 A1 和 A2 是属于选择关系, 客户可以选择其中任何一种支付方式进行支付, 而构件 A3 主要是完成转帐和交易完成的处理工作。

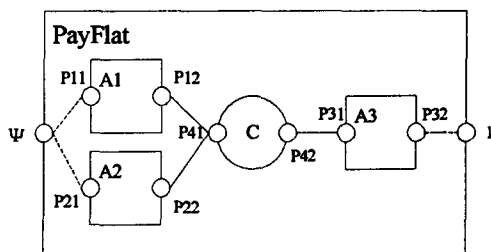


图 3 支付平台复合构件的常规描述

在图 3 中, 成员构件 A1 和 A2 的端口都连接到复合构件的端口  $\Psi$  上, 这样就使得系统存在语义歧义。因为从端口  $\Psi$  传递过来的数据, 不知道是并行的发送到 P11 和 P21, 还是选择发送到 P11 和 P12 上。而在该支付平台中, 由于用户可以选择任意一种支付方式进行支付, 所以端口  $\Psi$  的数据是需要选择发送到端口 P11 和 P21 中, 对于选择条件的定义, 则需要 ADL 在复合构件语法规格里专门设计这种映射逻辑和连接逻辑的额外机制, 这一方面增加了语言的复杂性, 另一方面也造成了语言理解的模糊性, 而连接件却恰好承担这一职责, 它负责构件的路由选择以及数据的交互。

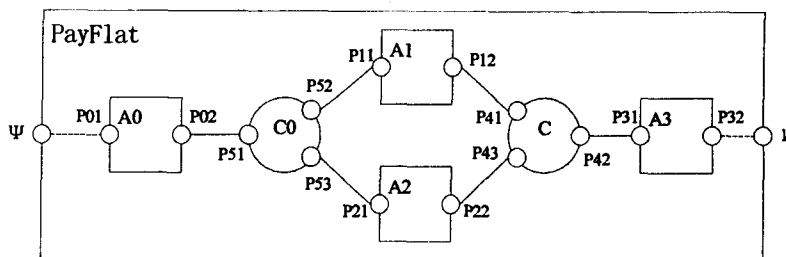


图 4 支付平台复合构件的 D-ADL 描述

因此, 在 D-ADL 中采用图 4 的体系结构对订单支付系统中的复合构件 PayFlat 进行描述。首先根据将连接件 C 的接口分成两组, 端口 P41 和 P43, 其中 P43 的规格和端口 P41 相同, 同时增加了一个连接件 C0 和构件 A0, 这样即满足了规则 2 和规则 3。原来 P11 和 P21 端口与外部端口  $\Psi$  的连接直接改成两个端口通过连接件 C0 连接, 其原先的交互协议通过连接

件的路由进行管理。这样既方便对构件和连接件进行统一的形式化描述,同时也有利于复合接口交互协议的推导(端口  $\Psi$  的规格、交互协议与构件 A0 端口 P01 的完全一致)。

规则 6 设端口  $\alpha$  属于复合构件 A0,端口  $\alpha'$  属于其成员构件 A1,则端口  $\alpha$  映射到端口  $\alpha'$  的必要条件是端口  $\alpha$  的所有通道集都需要映射到端口  $\alpha'$  的通道集上,即  $(E_a^I \rightarrow E_a'^I) \wedge (E_a^O \rightarrow E_a'^O) \wedge (E_a^{IO} \rightarrow E_a'^{IO})$ 。

在 D-ADL 语言中,为了提高连接规模,引入了端口的映射关系,其实质是通道之间的映射。

规则 7 设端口  $\alpha$  属于复合构件 A0, $\alpha'$  属于连接件(复合连接件)C0,则端口  $\alpha$  和端口  $\alpha'$  相连的必要条件是端口  $\alpha$  和  $\alpha'$  兼容,即  $\alpha \sim \alpha'$ 。

在 D-ADL 中,复合构件的行为分为两种<sup>[4]</sup>:一种是表示其处理业务逻辑的计算功能;另一种是用于预定义演化行为,需要显示地表示。根据复合构件的每一个交互活动都需要通过接口来完成,因此,可以将复合构件的端口分为两类:一类端口主要映射到成员构件端口上,属 port 类;另一类端口主要用于演化行为的输入输出,与外界环境进行交互,属 Eport 类。

仍以上述订单支付系统为例进行说明。如图 4 所示,其中复合构件 PayFlat 的端口包括  $\Psi$  和  $\mu$ ,其中  $\Psi$  映射到端口 P01 上,主要用于传送订单号和支付方式; $\mu$  映射到端口 P32 上,主要用于传送确认信息。它们的 D-ADL 语法描述如下:

```
Port  $\Psi$  is in (A)
Port  $\mu$  is out (B)
Port P01 is in (A')
Port P32 is out (B')
```

其中 A,B,A',B' 是通道集合,A 中的通道与 A' 通道一一对应,B 中的通道与 B' 通道一一对应。

在复合构件中端口的映射描述如下:

```
Configuration is {
Payflat  $\Psi$  rely P01
Payflat  $\mu$  rely P01
.....
}
```

规则 6 和规则 7 进一步定义了端口与通道的参数

类型规则。当多个构件进行组装的时候,除了要满足规则 1、2、3、4、5 外,还需要保证端口之间传送的数据类型是匹配的,这样才能保证传递数据的正确性。在 D-ADL 中的元素类型有多种,其中包括常规类型和体系结构类型<sup>[5]</sup>,在此不做过多的阐述,因此在端口的连接中主要还需要考虑到传输参数的兼容,包括所传递参数的数量、类型。

### 3 结束语

组装构件除了要考虑构件功能的组合外,还需要考虑接口的组装。在 D-ADL 框架中定义了构件组装的宏观理论及交互协议的推导,为了进一步完善 D-ADL,文中定义了接口的连接规则,其中定义 1~10 主要为端口连接规则提供必要的支持;规则 1、2、3、4、5 主要从结构上对端口的连接进行约束;规则 6、7 主要从端口本身的类型、参数类型以及参数个数进行一致性约束。整个定义有助于系统的设计与推演,同时也有利于避免系统行为的偏离以及系统死锁。

目前已开发了一套原型系统(SASM),在该原型系统中已经实现了构件与连接件的连接,但还未能完整地将端口的连接规则、构件的行为规约以及接口的交互规约应用到系统中。因此,下一步的研究工作包括:继续完善软件体系结构描述框架,利用体系结构端口连接规则进行构件动态组合失配检测,将端口连接规则应用到原型系统,直至完善整个系统。

### 参考文献:

- [1] 张世琨,张文娟,杨美清,等.基于软件体系结构的可复用构件制作和组装[J].软件学报,2001,12(9):1351-1359.
- [2] 冯冲,江贺,冯静芳.软件体系结构理论与实践[M].北京:人民邮电出版社,1996:7-8.
- [3] 李长云,李赣生,何频捷.一种形式化的动态体系结构描述语言[J].软件学报,2006,17(6):1349-1359.
- [4] Oreizy P, Gorlick M, Taylor R, et al. An architecture-based approach to self-adaptive software[J]. IEEE Intelligent Systems, 1999, 14(3):54-62.
- [5] 李长云.基于体系结构的软件动态演化研究[D].杭州:浙江大学,2005:26-27.

(上接第 120 页)

- [5] Witten I H, Frank E. 数据挖掘实用机器学习技术[M]. 第 2 版. 北京:机械工业出版社,2006:76-80.
- [6] 王锐,李晶,熊海蕴,等.基于关联规则的 Apriori 算法的可视化实现方法[J]. 计算机工程与设计,2007,28(4):757-759.
- [7] 芦洁,刘志镜.挖掘关联规则中对 Apriori 算法的一个改进[J]. 微电子学与计算机,2006,23(2):10-12.
- [8] 骆嘉伟,王艳,杨涛,等.一种结合完全连接的改进 Apriori 算法[J]. 计算机应用,2006,26(5):1174-1177.