基于时序逻辑的软件体系结构描述语言 XYZ/ADL*

朱雪阳+, 唐稚松

(中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

A Temporal Logic-Based Software Architecture Description Language XYZ/ADL

ZHU Xue-Yang+, TANG Zhi-Song

(Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: 86-10-62562796, Fax: 86-10-62563894, E-mail: zxy@ios.ac.cn

http://www.ios.ac.cn

Received 2002-08-22; Accepted 2002-12-23

Zhu XY, Tang ZS. A temporal logic-based software architecture description language XYZ/ADL. *Journal of Software*, 2003,14(4):713~720.

Abstract: The architecture description language (ADL) is the foundation of software development based on software architecture. An ADL supporting stepwise refinement can make it more convenient that a good design leads to a good implementation. The architecture description language XYZ/ADL can support the stepwise transition from higher-level architectures to lower-level architectures, because it is based on the temporal logic language (TLL) XYZ/E, which can represent both dynamic semantics and static semantics under a unified logical framework. In this paper, the framework and syntax of XYZ/ADL is presented and its underlying semantics is explained using XYZ/E, and how to describe software architecture and software architecture style using XYZ/ADL is introduced.

Key words: software architecture; architecture description language; temporal logic language XYZ/E; specification

摘 要: 体系结构描述语言(architecture description language,简称 ADL)是基于体系结构的软件开发的基础,便于表示求精的 ADL,使得好的设计能够方便地导出好的实现.时序逻辑语言 XYZ/E 可在统一的逻辑框架下既表示静态语义又表示动态语义,因而基于 XYZ/E 的体系结构描述语言 XYZ/ADL 支持从高层级体系结构到低层级体系结构之间的逐步过渡.系统地阐述了 XYZ/ADL 的概念框架并用 XYZ/E 进行语义解释,介绍了如何用 XYZ/ADL 描述体系结构和体系结构风格.

关键词: 软件体系结构;体系结构描述语言;时序逻辑语言 XYZ/E;规范

中图法分类号: TP311 **文献标识码**: A

在复杂软件系统的开发过程中,体系结构设计受到越来越多的关注.体系结构描述语言(architecture

* Supported by the National Natural Science Foundation of China under Grant Nos.60073020, 60273025 (国家自然科学基金); the National High-Tech Research and Development Program of China under Grant No.2001AA113200 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002cb312200 (国家重点基础研究发展规划(973))

第一作者简介: 朱雪阳(1971－),女,福建莆田人,博士生,讲师,主要研究领域为软件工程,软件体系结构.

description language,简称 ADL)是基于体系结构的软件开发的基础.一个好的 ADL 不仅要有较好的抽象表达能力,还应能够在体系结构设计阶段对有关性质进行分析和推理,最好还能支持从体系结构设计到代码实现的平滑过渡,使得一个好的设计能够方便地导出好的实现;要得到软件工程师的认同和使用,还要有方便的配套开发工具.已有的 ADL,或者只能支持系统高层抽象的体系结构描述与分析(如 WRIGHT[1],ACME[2]);或者是限于特定体系结构风格的描述与分析(如 Aesop[3],C2[4]);有的支持求精,但是规范描述与实现用的是不同的两种语言(如 Rapide[5]),都难以有效地支持基于体系结构的软件开发全过程.

我们提出的体系结构描述语言 XYZ/ADL,是对时序逻辑语言 XYZ/E[6]的扩充.它可以在统一的时序逻辑框架下描述系统静态语义到实现之间不同抽象层次的规范,便于体系结构的逐步求精描述以及相关性质分析;还有一套 XYZ/E 的 CASE 工具集作为底层支撑,便于对软件体系结构进行描述、分析、求精、验证,直至最终系统的编译实现.文献[7]提出了 XYZ/ADL 的整体构思,在文献[6]中介绍了它的直观概念,并描述实例,文献[8,9]也用 XYZ/E 描述了一些常见的组件、连接件及体系结构风格.本文则系统地阐述 XYZ/ADL 的概念框架,将文献[6]中的概念形式化.

本文第 1 节简单介绍作为规范描述语言的 XYZ/E.第 2 节介绍如何用 XYZ/ADL 描述系统的体系结构,并给出语义解释.第 3 节介绍如何用 XYZ/ADL 描述体系结构风格.最后总结全文.

1 时序逻辑语言 XYZ/E 简介

XYZ/E 的基本语言成分是条件元(conditional elements),有两种形式:

$$LB=y\wedge R\Rightarrow \$Ov=e\wedge \$OLB=z, \tag{1}$$

$$LB=y\wedge R\Rightarrow @(Q\wedge \$OLB=z). \tag{2}$$

形式(1)所示条件元直接定义了程序相邻状态之间的转换关系;形式(2)所示条件元表示程序抽象规范,其中符号@可以是下一时刻算子\$O 或最终时刻算子◇;两种形式的条件元都是时序逻辑公式,其语义即为此时序逻辑式的语义模型.

单元(unit)是一个条件元序列,具有如下形式:

$$\Box[A_1;\ldots;A_n]$$

$$\text{WHERE } B_1\wedge\ldots\wedge B_m$$

其中 $A_1,A_2,\ldots,A_n$ 是条件元;$B_1\wedge\ldots\wedge B_m$ 是一个时序逻辑公式,是对单元的约束或是某些特别谓词的定义;符号“;”和保留字 WHERE 等同于逻辑联结词合取.

单元也是一个时序逻辑公式,其语义对应于此时序逻辑式的语义模型.如果一个单元中的所有 $A_i(i=1,\ldots,n)$ 都是式(1)的形式,并且不包含 WHERE 部分,那么它就构成一个可执行的程序段;如果都是式(2)的形式,那么它就是一个程序的抽象规范;一个单元也可以既有式(1)形式的条件元又有式(2)形式的条件元,从而能够表达不同程度的抽象性.

XYZ/E 提供输入命令:

$$LB=y\wedge R\Rightarrow \text{ChNm}?x\wedge \$OLB=z$$

和输出命令:

$$LB=y\wedge R\Rightarrow \text{ChNm}!e\wedge \$OLB=z,$$

其中,“ChNm”是通道名,x 是变量,e 是表达式.“ChNm?x”表示通过通道 ChNm 将数据送入变量 x,ChNm 与 x 的类型必须匹配;“ChNm!e”表示将 e 的值经过通道 ChNm 送出,ChNm 与 e 的类型必须匹配.

互相交互的单元可以用“||”进行组合,它们之间的交互是通过显式的数据传递来实现的.参与交互的单元并不直接指出对方的单元名,而是指定一个通道来进行数据传递.两个单元之间能够进行组合的前提是:一个单元有一个通过某个通道的数据输出(ChNm!e),另一个单元要求一个来自相同通道的输入(ChNm?x).

2 体系结构描述

不同的 ADL 对体系结构概念有不同的解释,我们支持 Medvidovic 的观点[10]:一个 ADL 应能显式地描述组件、连接件以及体系结构配置,并且能够提供相应的工具支持.

组件是一个具有一定功能的逻辑单元或者存储对象.它可能小到只是一个过程,也可能大到是整个系统.所以,我们将组件分为简单组件(simple component)和复合组件(composite component)两种.简单组件是不可细分的实体,一般对应于实现代码的一个过程、进程或数据存储;复合组件包含若干个子组件,子组件可以是简单组件,也可以是复合组件.连接件定义了组件之间的交互方式和规则.连接件在实现代码中不一定有对应的代码块,例如过程调用、共享变量等;但有时连接件也可能是复杂的协议,如客户/服务器协议、远程过程调用等.XYZ/ADL 明确区分组件、连接件的类型和实例.以下未指明为实例的均指类型.

体系结构配置通过将组件、连接件组装成一个更大的复合组件来描述.一个系统(或子系统)就是一个复合组件,它的体系结构配置就是系统的总体布局.

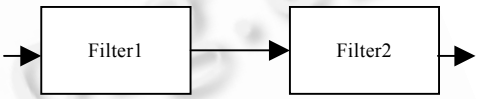

Fig.1 CompositeFilter

图 1 CompositeFilter

图 1 是一个简单系统 CompositeFilter 示意图,系统由两个过滤器 Filter1,Filter2 通过管道 Pipe 连接而成.下面,我们将结合这个例子来介绍 XYZ/ADL.

2.1 简单组件

组件分为接口(interface)和计算(computation)两个部分.在使用一个组件进行体系结构配置时,通过接口了解该组件"应如何使用"、"能做什么",因此接口包括一组说明组件与外部环境交互行为的端口(port)描述和说明组件功能的功能规范(function specification).计算规范(computation specification)描述组件的具体行为.

每个端口是组件对外部环境的一个请求,或是能够为环境提供的一个服务,用通道表示.它包括通道的数据类型声明和通道的行为描述两部分,语法如下:

%PORT PortName==DataType Declaration; [PortBehavior].

其中,DataType Declaration 声明该端口所能接受的数据类型;PortBehavior 刻画该端口的行为,同时也是组件的部分行为(外部可见行为),规定了外部环境应如何通过该端口与组件交互,由一个 XYZ/E 单元表示.

功能规范说明该组件"做什么",是一个时序逻辑公式,语法如下:

%FUNCTION==[Function Specification].

计算规范将各个端口连接为一个整体,并且对功能规范进行求精,是完整的组件行为描述.这部分由一个 XYZ/E 单元表示,可以是不同抽象层次的行为.语法如下:

%COMPUTATION==[Computation Specification].

如果组件 Filter1 有一个数据输入端口 DataIn 输入整数,一个数据输出端口 DataOut 输出结果,该组件用来计算输入数的阶乘,则其抽象描述如下:

%COMPONENT **Filter1**==[
%PORT **DataIn**==INT;
□[*LB*=Start⇒DataIn?*x*∧$O*LB*=*L*1;
LB=*L*1∧~(*x*=EOF)⇒DataIn?*x*∧$O*LB*=*L*1;
LB=*L*1∧(*x*=EOF)⇒$O*LB*=EXIT]
%PORT**DataOut**==INT;
□[*LB*=Start⇒$O*LB*=*L*1;
LB=*L*1∧~(*y*=EOF)⇒DataOut!*y*∧$O*LB*=*L*1;
LB=*L*1∧(*y*=EOF)⇒DataOut!EOF∧$O*LB*=EXIT]
%FUNCTION==[*x*>0→◇(*y*=*x*!)]
%COMPUTATION==□[*LB*=Start⇒$O*LB*=*L*0;

LB=*L*0⇒DataIn?*x*∧$O*LB*=*L*1;
LB=*L*1∧~(*x*=EOF)⇒$O*LB*=*L*2;
LB=*L*1∧(*x*=EOF)⇒$O*LB*=End;
LB=*L*2∧~(*x*<0)⇒◇(*y*=*x*!∧*LB*=*L*3);
LB=*L*2∧(*x*<0)⇒$O*LB*=*L*0;
LB=*L*3⇒DataOut!*y*∧$O*LB*=*L*0;
LB=End⇒DataOut!EOF∧$O*LB*=EXIT
]WHERE 1=0!∧*g*=(*x*-1)!∧*y*=*g***x*→*y*=*x*!;
]

从组件 Filter1 的描述中我们可以看到,每个端口规范展示了部分组件信息.DataIn 端口说明了何种数据能够通过该端口读入以及如何读入;DataOut 端口则说明了何种数据是如何输出的.功能规范是组件最高层抽象的规范描述,计算规范可以看作是它的求精(refinement),而且可以是不同抽象层次的.例如,组件 Filter1 的计算规范可被求精到最底层(实现代码,全部由式(1)的条件元组成):

%COMPUTATION==□[*LB*=Start⇒$O*LB*=*L*0;
LB=*L*0⇒DataIn?*x*∧$O*LB*=*L*1;
LB=*L*1∧~(*x*=EOF)⇒$O*LB*=*L*2;
LB=*L*1∧(*x*=EOF)⇒$O*LB*=End;
LB=*L*2∧~(*x*<0)⇒$O*i*=1∧$O*y*=1∧$O*LB*=*L*3;
LB=*L*2∧(*x*<0)⇒$O*LB*=L0;
LB=*L*3∧(*i*<*x*)⇒$O*y*=*y***i*∧$O*i*=*i*+1∧$O*LB*=*L*3;
LB=*L*3∧~(*i*<*x*)⇒DataOut!*y*∧$O*LB*=*L*0;
LB=End⇒DataOut!EOF∧$O*LB*=EXIT]

简单组件的求精体现在计算规范的求精上,如果它的行为比较复杂,可以通过逐步求精到最底层的实现代码,中间步骤描述既包含形式(1)的条件元又包含形式(2)的条件元.

组件及其内部的各部分之间关系的语义解释如下:

定义 1. 一个组件的 XYZ/ADL 描述 DC 分为接口描述和计算规范两个部分.接口描述包含一组端口和一个功能规范,计算规范是组件的完整行为描述.若 DC 的端口行为描述为 $PBehavior_1,\ldots,PBehavior_k$,功能规范为 *f*,计算规范为 ComSpec,则它们之间有如下关系:$Pbehavior_i(i=1..k)$是 ComSpec 在各个端口上的投影,并且 ComSpec⊆*f*,即 ComSpec 是 *f* 的求精.

尽管计算规范已是完整的组件行为描述,对组件进行接口描述仍是必要的.有了接口描述,在结构配置时,只须考虑连接的接口之间的一致性问题,可以简化结构分析;另一方面,可以提高组件描述的灵活性和抽象程度.各组件在逻辑上是互相独立的,它的正确性不受其他组件的影响,在组件规范中只要说明该组件的自身行为,而它可能进行的交互则由连接件规范描述.这样就可以提高组件的独立性,有利于重用和维护.

2.2 连接件

连接件用于描述某类交互的共有特性,它的实例可用于连接满足要求的不同组件.每个连接件分为接口和交互协议(interact protocol)两部分.接口由一组角色(role)定义,表明参加该类交互的组件所应有的外部行为(即端口类型).交互协议描述如何将角色连接在一起产生交互,从而将参与交互的组件计算规范组合起来.

角色用通道表示,包括通道的数据类型声明和通道的行为描述两部分,语法如下:

%ROLE RoleName==DataType Declaration; [RoleBehavior].

其中,DataType Declaration 声明该角色所能接受的数据类型,或者说是作为这一角色参与交互的端口所应接受的数据类型;RoleBehavior 是对该角色行为的限制,由一个 XYZ/E 单元表示.

交互协议由一个 XYZ/E 单元描述,语法如下:

%GLUE==[Interact Protocol].

在上例中,连接件 Pipe 用来将 Filter1 的输出传送给 Filter2.它有两个角色 Source 和 Sink,分别对应于 Filter1 的 DataOut 和 Filter2 的 DateIn.其 XYZ/ADL 描述如下:

%CONNECTOR **Pipe**==[

%ROLE **Source**==INT;

□[LB=Start⇒OLB$=$L$1;

LB=$L$1∧~(y=EOF)⇒Source!y∧OLB$=$L$1;

LB=$L$1∧(y=EOF)⇒Source!EOF∧OLB$=EXIT]

%ROLE **Sink**==INT;

□[LB=Start⇒Sink?x∧OLB$=$L$1;

LB=$L$1∧~(x=EOF)⇒Sink?x∧OLB$=$L$1;

LB=$L$1∧(x=EOF)⇒OLB$=EXIT]

%GLUE==□[LB=Start⇒OLB$=$L$0;

LB=L0⇒Source?x∧OLB$=$L$1;

LB=$L$1∧~(x=EOF)⇒Sink!x∧OLB$=$L$0;

LB=$L$1∧(x=EOF)⇒OLB$=End;

LB=End⇒Sink!EOF∧OLB$=EXIT]

]

任何具有 Sink 所描述的外部行为的组件都可以从 Pipe 读取数据,而任何具有 Source 所描述的外部行为的组件都可以向 Pipe 写出数据.角色表示的是参与交互的组件的行为,交互协议则表示组件间的组合.因此,交互协议中对通道的使用与角色中的互补:如果有一个角色通过某个通道输入数据,交互协议中就应有数据从同一个通道输出;如果有一个角色输出一个数据到某个通道,交互协议中就应有从该通道的数据输入.可见,交互协议可与每个角色直接交互,而每个角色只能与交互协议直接交互.

下面,我们用 XYZ/E 给出连接件规范的语义解释.

定义 2. 连接件 CON 由一组角色 $R_1,\ldots,R_n$ 以及交互协议 GLUE 定义,其中每个角色 R_i 的行为描述为 $\mathrm{RBehavior}_i(i=1\ldots n)$,那么 CON 的规范是一个 XYZ/E 单元:

$$\mathrm{GLUE}\|(\mathrm{RBehavior}_1\|\ldots\|\mathrm{RBehavior}_n),$$

其中,在 GLUE 中出现的通道集合为 $\{R_1,\ldots,R_n\}$.

2.3 复合组件

复合组件是由一些组件根据一定的要求连接而成的,这些子组件还有可能是复合组件.所以,如果将一个体系结构设计放在三维空间上考虑,复合组件则在横向上体现了组件之间的体系结构配置关系,在纵向上体现了组件之间的层次关系.

复合组件的接口描述与简单组件相同,而它的行为是通过几个子组件的连接来表示的.所以,在描述中我们必须说明一个复合组件包含哪些组件和连接件实例,它们是如何连接在一起的.

复合组件的组合声明指示它包含哪些组件和连接件实例,语法如下:

%COMPOSITION==[

ComponentInstanceName:ComponentName;

…

ConnectorInstanceName:ConnectorName;

…]

复合组件的连接定义表明它的内部体系结构配置,语法如下:

%ATTACHMENTS==[

ComponentInstanceName.PortName # ConnectorInstanceName.RoleName;

…

ComponentInstanceName.PortName ## PortName;
…]

其中“#”定义了连接(attach)操作,表示左边的组件端口与右边的连接件角色相连接,从中可以看出哪个组件参与了哪一种交互.有些子组件的端口没有和某个角色相连接,而是被指定为复合组件的端口,称为绑定(bind)操作,由“##”定义,右式是该复合组件的端口,因此可将组件名省略.

复合组件 CompositeFilter 由一个 Filter1 实例和一个 Filter2 实例通过一个 Pipe 实例连接而成,分别将 Filter1 的 DataIn 和 Filter2 的 DataOut 作为它的 In 和 Out 端口.其 XYZ/ADL 描述如下:

```
%COMPONENT CompositeFilter==[
    %PORT In==INT; [PortBehavior]
    %PORT Out==INT; [PortBehavior]
    %FUNCTION==[Function Specification]
    %COMPOSITION==[f1:Filter1;
                   f2:Filter2;
                   p:Pipe]
    %ATTACHMENTS==[f1.DataOut # p.Source;
                   f2.DataIn # p.Sink;
                   f1.DataIn ## In;
                   f2.DataOut ## Out]
]
```

连接件描述的是抽象的交互行为,用角色代表参与交互的各个组件,一旦连接件与具体的组件连接成复合组件,它的各个角色就被各参与交互的组件端口所代替.非形式化地可将“ComIn.Port1 # ConIn.Role2”解释为组件 ComIn 在通过 Port1 参与由连接件 ConIn 表示的交互时扮演了的角色 Role2,“ComIn.Port1 ## Port2”解释为组件 ComIn 的 Port1 被指定为该复合组件的端口 Port2.形式化地解释连接操作语义如下:

定义 3. 连接件 CON 由一组角色 $R_1,\ldots,R_n$ 以及交互协议 GLUE 定义.$P_1,\ldots,P_n$ 是一组端口,每个 P_i 的行为描述为 $\mathrm{PBehavior}_i(i=1\ldots n)$,那么,当执行 $P_i \# R_i(i=1\ldots n)$操作以后,GLUE 中出现的 R_i 全部替换为 P_i,并且 CON 的规范是一个 XYZ/E 单元:

$$\mathrm{GLUE}\|(\mathrm{PBehavior}_1\|\ldots\|\mathrm{PBehavior}_n),$$

其中,在 GLUE 中出现的通道集合为$\{P_1,\ldots,P_n\}$.

CompositeFilter 的行为描述是通过它的子组件的连接来表示的,对应的 XYZ/E 描述如下:

```
%COMPUTATION==□[LB=Start⇒$Of1=Filter1{1}∧$Of2=Filter2{1}∧$Op=Pipe{1}∧$OLB=L0;
                LB=L0⇒$OIn=f1.DataIn∧$OOut=f2.DataOut∧$OLB=L1;
                LB=L1⇒||[f1.COMPUTATION;p.GLUE;f2.COMPUTATION];
                LB=End⇒$OLB=EXIT]
```

复合组件的语义解释如下:

定义 4. 若复合组件 CCOM 定义了 *pNum* 个端口 $\mathrm{CPort}_p(p=1..pNum)$;声明了 n 个组件实例 $\mathrm{ComIn}_1{:}\mathrm{ComType}_1,\ldots,\mathrm{ComIn}_n{:}\mathrm{ComType}_n$,表示每个 $\mathrm{ComType}_i$ 计算规范的单元为 $\mathrm{ComSpec}_i$,;声明了 m 个连接件实例 $\mathrm{ConIn}_1{:}\mathrm{ConType}_1,\ldots,\mathrm{ConIn}_m{:}\mathrm{ConType}_m$,每个 $\mathrm{ConType}_j$ 有 r_j 个角色,交互协议为单元 $\mathrm{ConGLUE}_j$;定义了 $\mathrm{SUM}_{j=1..m}r_j$ 个连接 $\mathrm{Port}_k \# \mathrm{Role}_k(k=1..\mathrm{SUM}_{j=1..m}r_j)$和 *pNum* 个绑定 $\mathrm{Port}_p \#\# \mathrm{CPort}_p(p=1..pNum)$,则所有 $\mathrm{ConGLUE}_j$ 中出现的通道 Role_k 都被替换为相应的 Port_k,所有在绑定定义中出现的 Port_p 都被替换为相应的 CPort_p,并且 CCOM 的行为规范是一个 XYZ/E 单元:

$$(\|i{:}1..n(\mathrm{ComInSpec}_i{:}\mathrm{ComSpec}_i))\|(\|j{:}1..m(\mathrm{ConInGLUE}_j{:}\mathrm{ConGLUE}_j)).$$

我们将一个完整的系统看作一个复合组件,对系统的体系结构描述除了将保留字%COMPONENT 换为%SYSTEM 以外,其他完全相同.

XYZ/ADL 的体系结构元素之间的关系如图 2 所示.其中,组件的端口与连接件的角色必须一致才能连接,

复合组件的端口与子组件的端口必须一致才能绑定.关于这些一致性的分析,我们将另文加以详述.

3 体系结构风格

通过前面对 XYZ/ADL 的介绍,我们已经能用它来描述一个具体的系统了.但是在很多情况下,软件设计人员更关心的可能是某类具有相似性质的系统族,而不仅仅是孤立地对单个系统进行体系结构设计.体系结构风格(architectural style)就是用来刻画具有相似结构和语义性质的一类系统族的.它定义一组组件、连接件的类型以及对组件、连接件如何连接的约束[11].

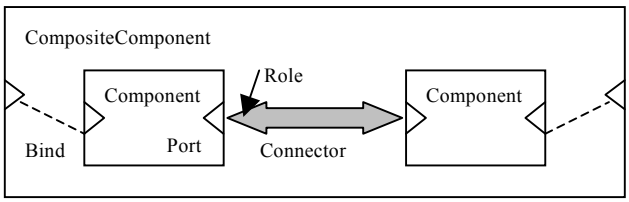

Fig.2 Relation among architectural elements
图 2 体系结构元素之间的关系

较之具体的体系结构而言,其风格更多一些灵活性,而约束和完整性则少一些.如图 1 所示的例子是一个具有管道-过滤器风格的体系结构,它的组件个数、组件端口个数、连接件个数以及组件的行为规范都是确定的.而一个管道-过滤器风格只要求数据是通过管道连接件传送,并不限定过滤器、过滤器端口及管道的个数,而且对过滤器的行为规范并无要求,只需其端口与管道的角色要求一致即可.可见,在体系结构风格中,我们可能只描述组件、连接件的部分性质,只对它们作部分约束.

为了便于描述这类情况,XYZ/ADL 引入了端口类型(port type)这一概念.端口类型可用作组件端口或连接件角色,它的表示形式与端口和角色的相同.例如,在管道-过滤器风格中引入 DataInput 和 DataOutput 端口类型,定义如下:

%PORTTYPE **DataInput**==INT; [同端口 **DataIn**]

%PORTTYPE **DataOutput**==INT; [同端口 **DataOut**]

在风格描述中,还要给出它们连接的约束条件.风格的约束条件是一个断言,由一个逻辑公式表示,任何声明为该风格的体系结构都必须满足这个断言.在约束定义中要用到的集合有:

Components:一个体系结构中所有组件的集合.

Connectors:一个体系结构中所有连接件的集合.

Ports(Com):组件 Com 所有端口的集合.

Roles(Con):连接件 Con 所有角色的集合.

Attachments:一个体系结构中所有连接的集合.一个连接是一个二元组(Com.port,Con.role).

用到的操作有:

Name(e):e 的名字.e 可能是一个组件、连接件、端口或角色.

Type(e):e 的类型.e 可能是一个组件、连接件、端口或角色.

Computation(Com):组件 Com 的计算规范.

Function(Com):组件 Com 的功能规范.

Glue(Con):连接件 Con 的交互协议.

连接方式对于确定不同的体系结构风格起着重要的作用,所以在风格描述中必须定义连接件类型,或者引用已定义过的连接件.对于组件类型,可根据情况定义类型或者只是对其进行部分约束.

管道-过滤器风格可描述如下:

%STYLE **Pipe-Filter**==[

%PORTTYPE **DataInput**==[同上]

%PORTTYPE **DataOutput**==[同上]

%CONNECTOR **Pipe**[同上]]

WHERE[∀con:Connectors(Type(con)=Pipe)

∧∀com:Components·∀p:Ports(com)(Type(p)=DataInput∨Type(p)=DataOutput)]

子风格是对原有风格的求精,它的连接件类型不变.因此,定义某风格的子风格只要写出进一步的约束条件即可.例如,定义一个星型管道-过滤器风格作为管道-过滤器风格的子风格,描述如下:

%STYLE StarPipeFilter: Pipe-Filter

WHERE [∃center:Components·∀con:Connectors·∃r:Roles(con);p:Ports(center)

((center.p),(con.r))∈Attachments]

∧[∀com:Components·∃con:Connectors;r:Roles(con);p:Ports(con)((com.p),(con.r))∈Attachments)]

反之,对于已给出的两个风格,要判定它们是否有求精关系,我们给出如下的形式定义:

定义 5. A,B 是两个体系结构风格,若 A.Connectors=B.Connectors,并且 A.Constrains→B.Constrains,则 A 是 B 的一个子风格.

4 结 语

本文系统地阐述了 XYZ/ADL 的概念框架,与之对应地定义了一套体系结构层次的语法,并用 XYZ/E 解释其语义.限于篇幅,本文所用例子较小,在文献[6]中,有利用 XYZ/E 描述操作系统内核体系结构、描述移动通信监控系统体系结构的例子.我们将集成 XYZ 系统原有的 CASE 工具设计实现 XYZ/ADL 工具,使其能够支持体系结构设计的求精、分析、验证及执行代码的生成.

致谢 中国科学院软件研究所的李广元副博士对本文的完成提出了很多有益的建议,在此表示感谢.

References:

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

[2] Garlan D, Monroe R, Wile D. Acme: an architecture description interchange language. In: Johnson JH, ed. Proceedings of the CASCON'97: The 7th Annual IBM Centre for Advanced Studies Conference. 1997. 169~183. http://www-2.cs.cmu.edu/afs/cs/project/compose/ftp/pdf/acme-cascon97.pdf.

[3] Garlan D, Allen R, Ockerbloom J. Exploiting style in architectural design environments. In: Wile D, ed. Proceedings of the SIGSOFT'94: The 2nd Symposium on the Foundations of Software Engineering. ACM Press, 1994. 175~188. http://www-2.cs.cmu.edu/afs/cs/project/able/ftp/aesop-fse2/aesop-fse2.pdf.

[4] Medvidovic N, Oreizy P, Robbins JE, Taylor RN. Using object-oriented typing to support architectural design in the C2 style. In: Garlan D, ed. Proceedings of the ACM SIGSOFT'96: The 4th Symposium on the Foundations of Software Engineering. ACM Press, 1996. 24~32. http://www.isr.uci.edu/architecture/papers/ADL-FSE96.pdf.

[5] Luckham DC, Vera J. An event-based architecture definition language. IEEE Transactions on Software Engineering, 1995,21(9):717~734.

[6] Tang ZS, *et al*. Temporal Logic Programming and Software Engineering. Beijing: Science Press, 2002 (in Chinese).

[7] Tang ZS. Three stages of XYZ system research project. In: Zhang SM, Nakakoji K, eds. Proceedings of the ISFST'98: International Symposium on Future Software Technology. 1998. 7~12. http://www.iijnet.or.jp/sea.

[8] Zhou YX, Ai B. A study of architecture modeling. Journal of Software, 1998,9(11):866~872 (in Chinese with English abstract).

[9] Jiao WP, Shi ZZ. Formalizing architectural style with XYZ/E. Journal of Software, 2000,11(3):410~415 (in Chinese with English abstract).

[10] Medvidovic N, Taylor RN. A classification and comparison framework for software architecture description language. IEEE Transactions on Software Engineering, 2000,26(1):70~93.

[11] Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. New Jersey: Prentice-Hall International, Inc., 1996.

附中文参考文献:

[6] 唐稚松,等.时序逻辑程序设计与软件工程.北京:科学出版社,2002.

[8] 周莹新,艾波.软件体系结构建模研究.软件学报,1998,9(11):866~872.

[9] 焦文品,史忠植.用 XYZ/E 形式化体系结构风格.软件学报,2000,11(3):410~415.