ISSN1000-9825 Journal of Software 软件学报 2000,11(3):410~415

用 XYZ/E 形式化体系结构风格*

焦文品 史忠植

(中国科学院计算技术研究所 北京 100080)

E-mail: wpjiao@yahoo.com

摘要 该文用时序逻辑语言 XYZ/E 刻画了若干种常见的体系结构风格.在刻画风格之前,首先通过研究常见组件及连接方式间的关系分析了风格之间的关系,然后通过对特定组件和连接方式的组合产生了完整的体系结构风格的形式化描述.

关键词 软件体系结构,风格,形式化,XYZ/E.

中图法分类号 TP311

随着软件系统规模的不断扩大,人们理解软件系统的角度千差万别,对系统的理解也就很难达成共识.因此,以形式化的方法来刻画软件系统不仅能明确讲明系统的真实内涵,也有助于设计和使用系统的有关人员更准确地了解系统.

体系结构风格定义了一系列系统的结构组织的模式[1],它是对一类具有相似结构的系统的抽象.体系结构风格既定义了组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[2].研究体系结构风格不仅有助于更准确地把握具有特定风格的软件系统的各种特征,以便设计人员能在系统结构级上尽早达成共识,提高系统的可重用性,而且能帮助设计人员更清楚地了解不同体系结构风格间的异同点,为使用和组合不同风格的组件提供指导.本文将利用 XYZ/E[3]对几种常见体系结构风格进行形式化描述.

XYZ/E 是一种以线性时序逻辑为基础的形式化程序设计语言,它提供了一套形式化规范描述软件系统的手段.在 XYZ/E 中,所有程序单元(语句、程序块)都是合式公式,另外,它还允许在同一程序中包含下面两种命令格式:

$$LB=y\wedge R\Rightarrow \$O(v_1,\ldots,v_k)=(e_1,\ldots,e_k)\wedge \$OLB=z, \tag{1}$$

$$LB=y\wedge R\Rightarrow \Diamond(Q\wedge LB=z), \tag{2}$$

其中 LB 为标号控制变量,y 和 z 为具体的标号,$\$O$ 和 $\Diamond$ 分别表示时序逻辑中的"下一次"和"最终"运算符,而 R 和 Q 分别为前置和后置断言.式(1)表示算法中可执行的一步动作(状态转换),式(2)则为一次过程调用的规范描述.

可见,XYZ/E 既适合在较低层次上描述具体的算法,又能在较高的抽象层次上对程序块进行规范描述.又因为它们都是合式公式,在用 XYZ/E 规范描述各种常见组件、连接方式及体系结构风格的同时,还可以直接检验或证明规范描述的一致性和正确性.

一种体系结构风格是由若干组件以特定的连接方式组合而成的一类系统的抽象,因此,在用 XYZ/E 形式化风格之前,我们将首先利用 E-R 图来分析组件及连接方式之间的关系,并在此基础上进一步分析风格之间的关系.然后根据这些关系从简到繁,逐步对较原始的组件和连接方式进行扩充来描述各种组件、连接方式.最后通过对特定组件和连接方式的组合来产生完整的体系结构风格的形式化描述.

* 本文研究得到国家 863 高科技项目基金(No.863-306-ZT02-01-3)资助.作者焦文品,1969 年生,博士生,主要研究领域为智能软件,软件工程及其理论.史忠植,1941 年生,研究员,博士生导师,主要研究领域为人工智能,智能软件.

本文通讯联系人:焦文品,北京 100080,中国科学院计算技术研究所智能计算机科学室

本文 1998-05-21 收到原稿,1999-04-07 收到修改稿

1 相关工作

体系结构风格为人们解决问题提供了一套简明的体系结构实体及限制的说明.自从体系结构风格提出以来[2],人们就试图以形式化的方法来精确刻画各种体系结构风格.

在文献[4]中,用 Z 表示法(Z notation)刻画了两种风格(pipe-filter 和 event-system).作者首先定义了体系结构的抽象语法,接着详细描述了风格的语义模型,最后又定义了体系结构的具体语法,并通过从抽象语法到语义域的映射函数提供了风格的具体含义.

在文献[5]中,通过定义一组公理和推理规则将风格描述成一种理论.利用抽象模式到具体模式之间的名字映射和风格映射可以证明两种模式之间的相对正确性.但在该理论体系中揉进了 3 种不同的形式化方法.

除此之外,众多体系结构描述语言都在一定程度上提供了某种形式化体系结构风格的特定手段[1].但这些方法普遍存在下列不足之处:(1) 在研究体系结构风格时往往不是从同一角度出发,如有的从功能的角度,有的从数据分布的角度,而有的纯粹是从其组件布局的角度.这样不仅模糊了风格之间的界限,相互之间存在交叉,而且加大了以统一的形式化基础刻画各种风格的难度.(2) 在刻画各种风格时,一般只规定了各种元素应受到的限制,而对于通过各种连接方式将组件连接起来之后所形成的系统到底是什么没有清楚地描述出来.

2 风格间的关系

在刻画风格时,我们除了希望从简到繁逐步扩充之外,也希望能在此过程中充分重用已定义的信息.因此,在用 XYZ/E 形式化风格之前,我们首先来讨论一下风格之间的关系.而风格间的关系是由组件及连接方式之间的关系来体现的.

2.1 常见组件间的关系

在文献[1]中,列出了很多常见的组件,如 Computation(计算功能)、Memory(数据存储)、Manager(管理器)、Controller(控制器)、Link(链接)、Data(数据结构)等.把上述组件看成实体,用实体联系图(E-R 图)可以将它们之间的关系表示为图 1.图中各组件间除了具有连线所示的关系之外,下层组件更为原始,而上层组件相对比较高级.

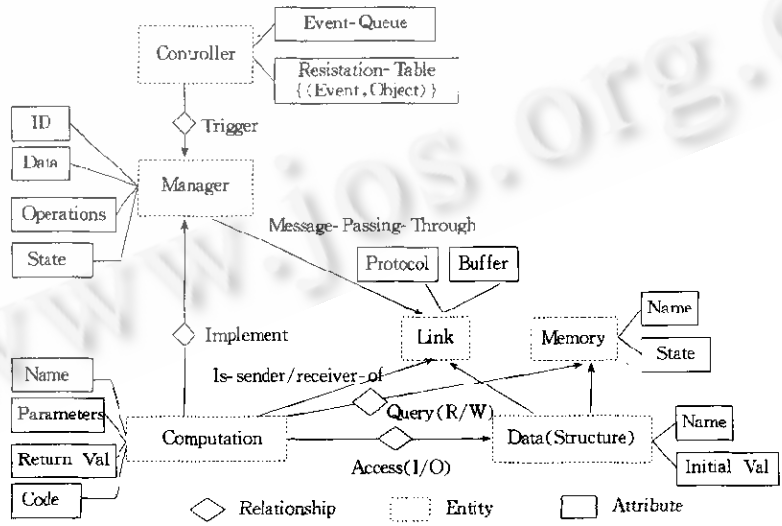

Fig. 1 The entity-relationship diagram of common components
图1 常见组件间的实体关系图

2.2 常见连接方式间的关系

在文献[1]中,列出了很多常见的连接方式,如 Procedure-Call(过程调用)、Data-Flow(数据流)、Implicit-

Triggering(隐式触发)、Message-Passing(消息传递)、Shared-Data(共享数据)等.把上述连接方式看作实体,用E-R图可以将它们之间的关系表示为图2.

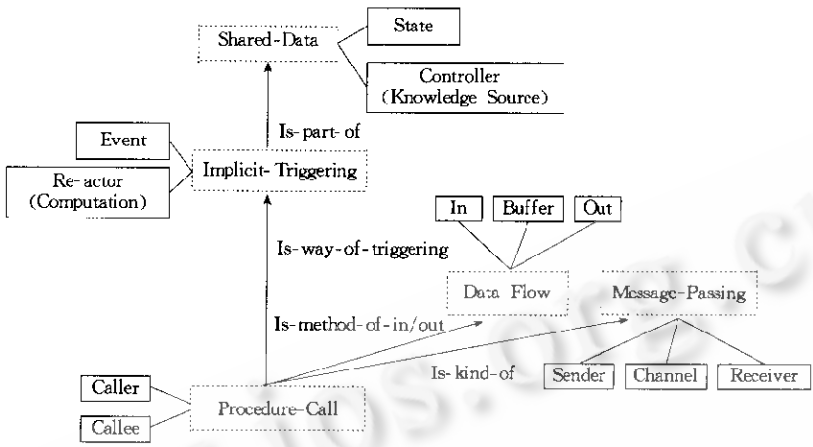

Fig.2　The entity-relationship diagram of common connectors
图2　常见连接间的实体联系图

2.3　常见风格的构成(成分)

上述常见组件及连接方式的不同组合,可以构成很多具有不同风格的体系结构.如:

- Concurrent Processes 由 Computation 通过 Message-Passing 方式组合而成;
- Pipe-Filter 由 Computation 通过 Procedure-Call 方式组合而成;
- Client/Server 由 Computation,Manager 通过 Procedure-Call 方式组合而成;
- Event-Based 由 Controller 和响应事件的对象(Manager)通过 Implicit-Triggering 方式组合而成;
- Object-Orientation 由 Manager 通过 Message-Passing 方式组合而成.

可以看出,各种体系结构风格间的关系并不是简单的层次关系,在刻画风格时,所重用的一般不是其他风格的整体形式规范,而是那些比构成该风格的组件或连接方式更原始的组件或连接方式的规范描述.

3　用 XYZ/E 形式化体系结构风格

3.1　用 XYZ/E 描述常见组件

根据前文所示的组件关系图,我们在描述组件时,依照从低级到高级的次序进行.这样,在描述较高级的组件时就可以直接重用低级组件的描述信息.

- Data(structure)

$$\%\text{Type}\ [\text{Struct\_Name}==\text{Record}(\text{Field\_List})],$$

其中 Struct_Name 为数据结构的名称,其结构定义方式与 Pascal 语言中的 Record 相似.

- Computation

在 XYZ/E 中,Computation 就是过程/进程或函数,其定义为

$$\%\text{Proc Proc\_Name}(\text{Parameters})==[\text{Statements}]\ \text{where}\ (\text{Constraint}),$$

其中 Proc_Name 为过程/进程名(对于函数,还需要定义返回值的类型),Statements 为实现该过程的语句序列,其形式如式(1)或式(2).

- Manager

在 XYZ/E 中,与 Manager 相对应的对象被定义成一个抽象数据类型,其定义为

$$\%\text{OOPROG Obj\_Name}==[\text{DataStructure};\text{Computations}]\ \text{where}\ (\text{Constraint}),$$

其中 Obj_Name 为对象名.

• Controller

虽然在 XYZ/E 中没有直接定义 Controller 的对应成分，但 XYZ/E 中的进程能够通过通道(channel)获取环境的状态信息，并能根据所获得的状态触发相应的进程. 即 Controller 相当于 XYZ/E 中的选择语句：

$$LB=y\wedge R\Rightarrow!!\ [C_1|>D_1,\ldots,C_m|>D_m],$$

其含义为：在前置条件 R 满足的情况下，当外界环境中出现事件 C_i 时，调用过程 D_i.

XYZ/E 的语句或程序块的语义与普通程序设计语言的语义差别不大，这里不再详细描述上述组件的形式语义.

3.2 用 XYZ/E 描述常见连接方式

在 XYZ/E 中，过程调用和进程通信是最基本的两种连接方式，其他各种连接方式都是在此基础上扩展而来的.

• Procedure-Call

在 XYZ/E 中过程调用的基本形式为(设被调用的过程名为 Callee)

$$LB=y\Rightarrow\text{Callee}\wedge\ \$OLB=z,$$

其语义为

$$\frac{\{P\}\text{Callee}\{Q\}}{LB=y\wedge P\Rightarrow\Diamond(Q\wedge\ \$OLB=z)},$$

其中 P,Q 分别为过程 Callee 的前置和后置断言.

• Message-Passing

此连接方式由通道(channel)、消息发送者及接收者(sender/receiver)组成，设连接发送者和接收者的通道名为 c，则 Sender/Receiver 用 XYZ/E 可定义如下：

$$\text{Sender}==[LB=y\Rightarrow((c!\ w\wedge\ \$OLB=z)^{-}],$$

$$\text{Receiver}==[LB=y\Rightarrow(c?\ u\wedge\ \$OLB=z)^{-}],$$

那么 Message-Passing 可以定义成 Sender 和 Receiver 这两个并发进程的组合：

$$\text{Message-Passing}==\|\ [\text{Sender};\text{Receiver}],$$

其形式语义为 $P\rightarrow Q(u/w)$，$Q(u/w)$ 表示将 Q 中所有 u 的自由出现都用 w 代替，

$$\frac{\{P\}\|\ [\text{Sender};\text{Receiver}]\{Q\}}{LB=y\wedge P\Rightarrow\Diamond(Q\wedge\ \$OLB=z)}.$$

• Data-Flow

在连接方式的 E-R 图(如图 2 所示)中已经指出，数据流由存放数据的缓冲区(buffer)以及向缓冲区发送或读取数据的操作 in/out 组成. 在 XYZ/E 中，in/out 可以分别定义为(假设 Data_Flow 以先进先出的队列方式实现)：

$$\text{in}==[LB=y\wedge(\text{BufLen}+\text{InData}<=\text{MaxLen})\Rightarrow\Diamond(\ \$\text{Obuf}=\text{Buf}+\text{InData}\wedge\ \$OLB=z)],$$

$$\text{out}==[LB=u\wedge(\text{BufLen-OutData}>=0)\Rightarrow\Diamond(\text{Buf}=\text{OutData}+\ \$\text{Obuf}\wedge\ \$OLB=v)],$$

那么 Data-Flow 可以定义成 in 和 out 这两个并发进程的组合：

$$\text{Data\_Flow}==\|\ [\text{in};\text{out}].$$

• Implicit-Triggering

XYZ/E 是一种逻辑语言，其中的各种成分都可以表示成逻辑公式，因此，事件(event)在 XYZ/E 中实质上也是一个合式公式. 设当出现事件 E 时，所触发的过程或进程为 Re-actor，则这种隐式触发可以用 XYZ/E 定义为

$$LB-y\wedge E\rightarrow\text{Re-actor}\wedge\ \$OLB=z,$$

其形式语义为

$$E\rightarrow P,$$

$$\frac{\{P\}\text{Re-actor}\{Q\}}{LB=y\wedge E\Rightarrow\Diamond(Q\wedge\ \$OLB=z)}.$$

• Shared-Data

Shared-Data 与 Implicit-Triggering 的主要区别在于,后者由事件来驱动对象,而前者由共享数据的状态来驱动对象,这两者在 XYZ/E 中没有很明显的区别.

3.3 形式化体系结构风格

通过组合特定的组件及连接方式就可以得到体系结构风格的形式描述.下面我们仅就几种常见风格的规范描述和形式语义进行刻画.

• Concurrent Processes 风格

设系统 P 由 n 个并发进程$(P_1,P_2,\ldots,P_n)$组合而成,其中 $P_i==\{R_i\}P_i\{Q_i\}$,则系统定义为

$$P==\|[P_1,P_2,\ldots,P_n],$$

其形式语义为

$$\frac{LB=y\Rightarrow\|[P_1,P_2,\ldots,P_n]\wedge\$OLB=z}{LB=y\wedge\bigcap_{i=1}^{n}R_i\Rightarrow\Diamond(\bigcap_{i=1}^{n}Q_i\wedge\$OLB=z)}.$$

• Event-Based 风格

设系统 S 中存在可识别的事件序列 $E_1,E_2,\ldots,E_n$,这些事件所驱动的对象有 $Obj_1,Obj_2,\ldots,Obj_m$. 其中事件驱动器(controller)可以用 XYZ/E 表示为

$$\text{Controller}==[LB=y\wedge E_1\Rightarrow c!\ id_1\wedge\$OLB=z|\ldots|E_m\Rightarrow c!\ id_m\wedge\$OLB=z;].$$

所驱动的对象中响应对应事件的进程可以定义为

$$Obj_i==[LB=w\Rightarrow c?\ id\wedge id=id_i\wedge\$OLB=w_i;LB=w_i\Rightarrow(\text{Re-actor}_i\wedge\$OLB=u],$$

则系统可定义为

$$S==\|[\text{Controller},Obj_1,Obj_2,\ldots,Obj_n].$$

• Pipe-Filter 风格

为简单起见,设此风格由两个 Filter(F_1,F_2)通过 Data-Flow(数据流)Pipe(P_1)连接而成. 其中,Filter 为具有 Main/Subroutine 风格的组件,它由 Input,Data-Process,Output 这 3 个子过程组成. 整个 Pipe-Filter 可以用图 3 表示.

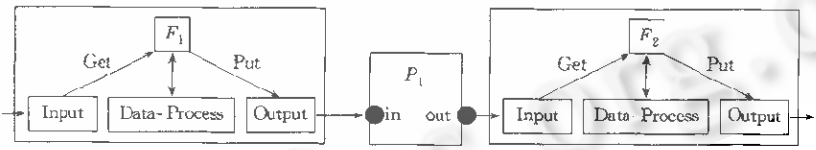

Fig. 3　A pipe-filter style
图3　Pipe-Filter风格示意图

其中　$F_1==[LB=y\wedge\Rightarrow\text{Input}_1\wedge\$OLB=y_1;LB=y_1\Rightarrow\text{Data\_Process}_1\wedge\$OLB=y_2;$

$LB=y_2\Rightarrow\text{Output}_1\wedge\$OLB=z],$

$F_2==[LB=w\Rightarrow\text{Input}_2\wedge\$OLB=w_1;LB=w_1\Rightarrow\text{Data\_Process}_2\wedge\$OLB=w_2;$

$LB=w_2\Rightarrow\text{Output}_2\wedge\$OLB=u],$

$P_1==\|[\text{In};\text{Out}].$

而　$F_1.\text{Output}_1==[LB=k\Rightarrow P_1.\text{In}\wedge\$OLB=\text{RETURN}],$

$F_2.\text{Input}_2==[LB=m\Rightarrow P_1.\text{Out}\wedge\$OLB=\text{RETURN}],$

于是,整个 Pipe-Filter 可以定义为

$$Pipe\text{-}Filter==\|[F_1;F_2].$$

• Client/Server 风格

设 Client/Server 由组件 Client,Server 组成,其中,

$$\text{Client}==[LB=y\wedge\Rightarrow\text{Request}\wedge\$OLB=y_1;LB=y_1\Rightarrow\text{Computation}\wedge\$OLB=z],$$

$$\text{Server} == [LB = w \Rightarrow \text{Provide} \wedge \$OLB = u].$$

若 Client，Server 间的连接方式为 Message-Passing(MP)，则可定义

$$\begin{aligned}\text{Client.Request} == [&LB = r \Rightarrow \text{MP.Sender(Request)} \wedge \$OLB = r_1;\\ &LB = r_1 \rightarrow \text{MP.Receiver(Result)} \wedge \$OLB - \text{RETURN}],\\ \text{Server.Provide} == [&LB = p \Rightarrow \text{MP.Receiver(Request)} \wedge \$OLB = p_1;\\ &LB = p_1 \rightarrow \text{Computation} \wedge \$OLB = p_2;\\ &LB = p_2 \Rightarrow \text{MP.Sender(Result)} \wedge \$OLB = \text{RETURN}].\end{aligned}$$

那么 Client/Server 可定义为

$$CS == \| [\text{Client}; \text{Server}].$$

4 结束语

与其他文献[1,4,5]中所采用的形式化策略不同，我们在规范描述体系结构风格时，首先单独刻画了几种常见的组件及连接方式．这样，只需组合特定的组件和连接方式就能得到新的风格，即在定义风格时可以直接重用已定义好的规范说明．为了重用已描述的各种规范说明，我们用实体联系图(E-R 图)描述了几种常见组件及连接方式间的关系，在此基础上分析了体系结构风格之间的关系．我们发现，由组件及连接方式构成的体系结构风格之间并不是简单的平行关系，也不是层次关系，而是呈现复杂的网状关系．所以，要重用已有的规范信息，只能从重用组件及连接方式着手．

根据风格间的关系，我们采取了逐步扩展简单风格以形成复杂风格的策略，用时序逻辑语言 XYZ/E 刻画了几种常见的体系结构风格．

我们在形式化体系结构风格时，并不局限于仅给出组件、连接方式及风格的静态属性，而是要说明由特定组件和连接方式构成的风格到底是什么，这一点正是其他形式化方法所忽视的，或者是做得不够的．

参考文献

1 Shaw M, Garlan D. Software architecture: perspectives on an emerging discipline. New Jersey: Prentice Hall, Inc., 1996

2 Garlan D, Shaw M. An introduction to software architecture. In: Ambriola V, Tortora G eds, Advances in Software Engineering and Knowledge Engineering. Singapore: World Scientific Publishing Company, 1993. 1～39

3 Tang Zhi-song. A temporal logic language for system implementation. In: Nigel H R eds. Systems Implementation 2000. Berlin: Chapman & Hall, 1998

4 Abowd G, Allen R, Garlan D. Using style to understand descriptions of software architecture. Software Engineering Notes, 1993, 118(3): 9～20

5 Moriconi M, Qian X. Correctness and composition of software architectures. ACM SIGSOFT Software Engineering Notes, 1994, 19(5): 164～174

Formalizing Architectural Styles with XYZ/E

JIAO Wen-pin SHI Zhong-zhi

(Institute of Computing Technology The Chinese Academy of Sciences Beijing 100080)

Abstract In this paper, a temporal logic language XYZ/E is used to formalize several common architectural styles. Before the styles described, the relationship among them is analyzed based on the relationships among components and connectors, and then the complete formal descriptions of the architectural styles are derived from the compositions of specific components and connectors.

Key words Software architecture, style, formalization, XYZ/E.